Evrostos: The rLTL Verifier*

Tzanis Anevlavis
University of California, Los Angeles
janis10@ucla.edu

Daniel Neider
Max Planck Institute for Software Systems
neider@mpi-sws.org

Matthew Phillipe
Universitè catholique de Louvain
matthew.philippe@uclouvain.be

Paulo Tabuada
University of California, Los Angeles
tabuada@ucla.edu

ABSTRACT

Robust Linear Temporal Logic (rLTL) was crafted to incorporate the notion of robustness into Linear-time Temporal Logic (LTL) specifications. Technically, robustness was formalized in the logic rLTL via 5 different truth values and it led to an increase in the time complexity of the associated model checking problem. In general, model checking an rLTL formula relies on constructing a generalized Büchi automaton of size $5^{|\varphi|}$ where $|\varphi|$ denotes the length of an rLTL formula φ. It was recently shown that the size of this automaton can be reduced to $3^{|\varphi|}$ (and even smaller) when the formulas to be model checked come from a fragment of rLTL. In this paper, we introduce *Evrostos*, the first tool for model checking formulas in this fragment. We also present several empirical studies, based on models and LTL formulas reported in the literature, confirming that rLTL model checking for the aforementioned fragment incurs in a time overhead that makes the verification of rLTL practical.

CCS CONCEPTS

• **Theory of computation → Logic and verification**; **Verification by model checking**; **Modal and temporal logics**;

KEYWORDS

Robustness; Formal methods; Temporal logic; Model Checking.

ACM Reference Format:
Tzanis Anevlavis, Daniel Neider, Matthew Phillipe, and Paulo Tabuada. 2019. Evrostos: The rLTL Verifier. In *22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '19), April 16–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3302504.3311812

1 INTRODUCTION

One of the most important problems in formal methods is model checking. Intuitively, it asks whether the model of a given system exhibits a specified behavior. The success of model checking lies in using finite structures to describe infinite behaviors, and it is known to be decidable for many temporal logics [5, 19]. The model of a system is given by a finite state structure, and the properties to be verified are formally given as a temporal logic specification.

Robustness (Greek: ευρωστία; evrostía) is a property widely recognized in the control community as essential in any feedback control loop. However, as we move from control problems that can be described by differential equations to control problems where software plays a more dominant role, such as in Cyber-Physical Systems (CPS), identifying the "correct" notion of robustness has remained a challenge [3, 6, 18, 21, 23]. An approach is given in [12], [13, Chapter 7], by using models that combine continuous and discrete behavior. Another direction is proposed in [7, 9, 10, 15] by introducing a logic enabling reasoning over real-valued signals. In that logic, robustness is derived from the real-valued nature of the signals, whereas in this paper, we reason over Boolean signals and robustness is derived from the temporal evolution of these signals.

Specifications for cyber components are typically written as:

$$\varphi \Rightarrow \psi,$$

where φ is an assumption about the environment and ψ is a system guarantee. The semantics of classical propositional logic renders this formula syntactically equivalent to:

$$\neg\varphi \vee \psi.$$

If the assumption φ is violated, nothing can be said about the guarantee ψ. Not even if "φ is close to being satisfied".

An alternative is proposed in [22], where the idea that "small" violations of the assumption lead to "small" violations of the guarantee inspired a new logic termed *robust Linear-time Temporal Logic* (rLTL). To formalize this idea, rLTL adopts a 5-valued semantics: the truth value of an rLTL formula is interpreted as corresponding to *true* or to different *shades of false*. Each one of the truth values is written as a sequence of four bits adhering to the following order, $0000 \prec 0001 \prec 0011 \prec 0111 \prec 1111$, with truth value 1111 corresponding to true, and the rest to different shades of false. Robustness now enters the picture via the rLTL semantics for implication. To illustrate the underlying idea, consider the rLTL formula[1] $\boxdot p \Rrightarrow \boxdot q$, which has truth value 1111 (corresponding to true) when $\Box p$ implies $\Box q$, *and* weakening the assumption $\Box p$ to $\Diamond\Box p$ implies the guarantee $\Diamond\Box q$, *and* weakening $\Diamond\Box p$ to $\Box\Diamond p$ implies $\Box\Diamond q$, *and* weakening $\Box\Diamond p$ to $\Diamond p$ implies $\Diamond q$. In contrast, the corresponding LTL formula $\Box p \Rightarrow \Box q$, would be

*This work was partially supported by the NSF grant 1645824 and by the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196.

HSCC '19, April 16–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6282-5/19/04…$15.00
https://doi.org/10.1145/3302504.3311812

[1] An rLTL formula is obtained from an LTL formula by superimposing dots or dashes to the LTL operators.

vacuously true if the assumption $\Box\, p$ is weakened, giving no insight to what happens to the guarantee $\Box\, q$.

Increasing the number of truth values increases the complexity of verifying rLTL specifications. More precisely, verifying an LTL formula φ requires the construction of a Generalized Büchi Automaton (GBA) with $O(2^{|\varphi|})$ states, where $|\varphi|$ is the number of subformulas in φ. However, an extension of this technique to verifying rLTL formulas, see [22, Theorem 4.9], relies on constructing a GBA with $O(5^{|\varphi|})$ states. As the complexity of model checking is proportional to the size of this automaton, this results in a large overhead in time complexity.

Recently, refined upper bounds for the rLTL verification problem were proved in [1]. The key idea is to think of the truth value of rLTL formulas as a sequence of 4 bits, and each bit as corresponding to the truth value of an LTL formula. As shown in [22], the difficulty of dealing with the 5-valued semantics of rLTL formulas lies in the fact that the four bits of a truth value are coupled in general, leading to exponentially longer formulas. Fortunately, it is possible to restrict rLTL to a fragment consisting of formulas that occur frequently in real-world applications and for which the complexity is reduced. In particular, it is shown in [1] that for the rLTL fragment of formulas of the form $\psi_1 \Rrightarrow \psi_2$, where ψ_1 and ψ_2 are rLTL formulas not containing robust implications or robust releases, the rLTL verification problem is solved by performing at most 4 LTL model checking steps, each using a GBA of size at most

$$O\left(2^{|\varphi|-k(\varphi)}3^{k(\varphi)}\right),$$

where $|\varphi|$ is the number of subformulas in φ, and $k(\varphi)$ is the number of *robust always* ($\boxdot$) operators in φ. In the worst case, $k(\varphi) = |\varphi|$ and the time complexity is proportional to $3^{|\varphi|}$, a considerable improvement over the bound $5^{|\varphi|}$ proved in [22] and closer to the tight LTL bound $2^{|\varphi|}$.

More than 60% of the rLTL versions of the LTL formulas available in the Büchi store, an open repository of Büchi automata [24], were in this fragment. Moreover, this rLTL fragment includes the GR(1) fragment for which efficient verification and synthesis algorithms are known. Hence, it is a practically significant and powerful fragment. Therefore, the ability to solve the rLTL model checking problem on the aforementioned fragment will significantly contribute to improve the robustness of system designs.

Towards this end, we present the first tool to perform rLTL model checking, *Evrostos: The rLTL Verifier*[2]. Evrostos is open-source and publicly available. It consists of two different components: (i) an rLTL-to-LTL translator for rLTL formulas in the aforementioned fragment, and (ii) the popular symbolic model checker NuSMV [4].

Using Evrostos we conducted several empirical studies with system models and LTL formulas reported in the literature. The results of these studies support the theoretical findings in [1] that the time overhead of rLTL for the fragment discussed is small and close to the one of LTL in most of the cases.

The structure of the paper is as follows. In Section 2 we present a motivating example of a real-life aircraft control model exposing the weakness of the logical implication in LTL. Section 3, briefly presents the syntax and semantics of rLTL, as well as the rLTL model checking problem. We also present the rLTL fragment of interest and provide theoretical results for the algorithm solving the verification problem. Section 4 introduces in detail Evrostos: The rLTL Verifier, which implements the algorithm of [1]. Finally, Section 5 presents case studies showing how rLTL verification for that fragment compares to LTL verification in terms of execution time. The empirical results also show how rLTL verification provides much more detailed information than LTL verification.

2 COMPARING rLTL WITH LTL ON AN EXAMPLE

The aim of this paper is not only to provide the first tool for rLTL verification, but also to motivate the widespread use of rLTL to specify and verify robustness properties. With this in mind, we present a case study exposing one of the weaknesses of logical implication in LTL; if the assumption is false, the implication is vacuously true and nothing can be deduced about the guarantee. In rLTL this is not the case; when the assumption is not true, depending on the truth value of the implication, we can conclude important information about the guarantee.

We use the model of an Automated Air Traffic Control System as developed in [25], where a model for the Automated Airspace Concept (AAC)[3] [8] is described. AAC ensures the safe separation of commercial aircraft within a given airspace sector so that safe distance between the aircraft is always guaranteed to prevent potential aircraft collisions. AAC detects potential conflicts between aircrafts and resolves them by generating resolution maneuvers and sending them to human pilots. Different controllers are responsible for handling short-term and near-term conflicts, as well as collision avoidance. Initially, the AutoResolver, addresses short-term conflicts (20 minutes to 3 minutes in the future). Then, the Tactical Separation Assured Flight Environment (TSAFE), addresses near-term conflicts (less than 3 minutes in the future). Finally, the Traffic Alert and Collision Avoidance System (TCAS), addresses possible collisions less than 30 seconds away. TSAFE overrides the AutoResolver and TCAS overrides both. After the conflict is resolved by any of the controllers, they relinquish control of the aircraft.

The model built in [25] consists of three aircrafts. When an alert involving two aircrafts is issued, one of the controllers will resolve it as in the scheme above. These alerts are represented by the variable "alert$_{ij}$" for aircrafts i and j, taking values in the set $\{non, AT, BT\}$, where *non* means "no potential collision", *AT* means "potential collision, time is Above Threshold for TSAFE", and *BT* means "potential collision, time is Below Threshold for TSAFE".

We consider a simplified version of this setup that could take place in a crowded airport airspace, or in a drone fleet coordination scenario where alerts are issued infinitely often given the high density of aircrafts. In this version we have only two controllers, the AutoResolver and the TSAFE controller, and 3 aircrafts. We would like to verify the following specification: *"If after some point in time there are no collision alerts for aircraft 1, then the TSAFE controller will relinquish control of the aircraft after resolving a collision".*

[2] Evrostos is a tool that Efficiently Verifies RObuSTness Of Specifications. It is available online at https://github.com/janis10/evrostos.

[3] AAC is a high-level generic framework proposed as a candidate for the Next Generation Air Traffic Control System, which was under development at NASA. Thus, we are considering a real-time, practical model to show the significance of our findings.

This is written in LTL as

$$(\Diamond\Box\ (\text{alert}_{12} = \text{non} \wedge \text{alert}_{13} = \text{non})) \Rightarrow (\Box\ (\Diamond\ \neg\text{tsafeControl}_1)), \quad (1)$$

where the boolean variable "tsafeControl_1" is true when TSAFE is controlling aircraft 1, and false otherwise.

The specification in rLTL is written as

$$(\diamondsuit\boxdot\ (\text{alert}_{12} = \text{non} \wedge \text{alert}_{13} = \text{non})) \Rrightarrow (\boxdot\ (\diamondsuit\ \neg\text{tsafeControl}_1)). \quad (2)$$

In practice, a more detailed specification would be used, but formula (2) suffices for our purposes.

Consider the following scenario: a bug in TSAFE of aircraft 1 makes it always be in control of the aircraft once it's triggered, resulting in the consequent of (1) to be *false*. Verifying LTL specification (1) on NuSMV for the simplified setup returns:

```
Specification F G(alert12 = non & alert13 = non)
                    -> G (F ! tsafeControl1) is TRUE.
```

Even if the controller is incorrect, the specification is true as its assumption is never satisfied (alerts are issued infinitely often). This misleads the designer in believing the controller behaves as expected.

Verifying rLTL specification (2) on Evrostos returns:

```
Model Checking of the original rLTL formula returns
                                    truth value 0001.
```

This truth value (see Section 3 for a definition of the rLTL semantics) tells us that even if there are no alerts infinitely often, the controller of aircraft 1 will not relinquish control infinitely often but finitely often. In other words, there will be infinitely many time instances when there is no alert and yet control is not relinquished. Already this illustrative example shows that rLTL model checking provides much more insight than classical LTL model checking.

3 rLTL MODEL CHECKING

As discussed in the introduction, the main goal of rLTL is to embed a notion of robustness into LTL. With this in mind, the syntax of rLTL closely resembles that of LTL using *robust* versions of LTL operators.

Definition 1 (rLTL syntax). *Let $\mathcal{P}$ be a nonempty, finite set of atomic propositions. The set of all rLTL formulas on $\mathcal{P}$, written* rLTL($\mathcal{P}$)*, is the smallest set satisfying*

- $\mathcal{P} \subset$ rLTL($\mathcal{P}$) *and*
- *if φ and ψ are elements of* rLTL($\mathcal{P}$)*, then $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \Rrightarrow \psi$, $\odot\ \varphi$, $\boxdot\ \varphi$, $\diamondsuit\ \varphi$, $\varphi\ \mathcal{R}\ \psi$ and $\varphi\ \mathcal{U}\ \psi$ are elements of rLTL($\mathcal{P}$) as well.*

The length of a formula $\varphi \in$ rLTL($\mathcal{P}$) is denoted by $|\varphi|$ and is the number of subformulas it contains.

Given a word $\sigma \in \left(2^{\mathcal{P}}\right)^{\omega}$ and a formula $\varphi \in$ rLTL($\mathcal{P}$), the semantics of rLTL provides the degree to which σ satisfies the LTL counterpart[4] of φ. This is captured by using a 5-valued semantics; one truth value corresponds to **true** and the others to *different shades of* **false**.

[4]The LTL counterpart of any rLTL formula is obtained by removing all the dots or dashes superimposed on the operators.

Table 1: The Full Semantics of rLTL and the ltl **operator.**

Operator	Symbol	Semantics, for $p \in \mathcal{P}, \varphi, \psi \in$ rLTL($\mathcal{P}$).
Atomic Proposition		$\forall 1 \le i \le 4 : \text{ltl}(i, p) = p.$
Negation	$\neg$	$\forall 1 \le i \le 4 : \text{ltl}(i, \neg\varphi) = \neg\text{ltl}(1, \varphi).$
Disjunction	$\vee$	$\forall 1 \le i \le 4 : \text{ltl}(i, \varphi \vee \psi) = \text{ltl}(i, \varphi) \vee \text{ltl}(i, \psi).$
Conjunction	$\wedge$	$\forall 1 \le i \le 4 : \text{ltl}(i, \varphi \wedge \psi) = \text{ltl}(i, \varphi) \wedge \text{ltl}(i, \psi).$
Robust Implication	$\Rrightarrow$	$\forall 1 \le i \le 3 :\ \text{ltl}(i, \varphi \Rrightarrow \psi) = (\text{ltl}(i, \varphi) \Rightarrow \text{ltl}(i, \psi)) \wedge \text{ltl}(i+1, \varphi \Rrightarrow \psi),$ $\text{ltl}(4, \varphi \Rrightarrow \psi) = (\text{ltl}(4, \varphi) \Rightarrow \text{ltl}(4, \psi)).$
Next	$\odot$	$\forall 1 \le i \le 4 : \text{ltl}(i, \odot\ \varphi) = \bigcirc\ \text{ltl}(i, \varphi).$
Robust Always	$\boxdot$	$\text{ltl}(1, \boxdot\ \varphi) = \Box\ \text{ltl}(1, \varphi),$ $\text{ltl}(2, \boxdot\ \varphi) = \diamondsuit\Box\ \text{ltl}(2, \varphi),$ $\text{ltl}(3, \boxdot\ \varphi) = \Box\diamondsuit\ \text{ltl}(3, \varphi),$ $\text{ltl}(4, \boxdot\ \varphi) = \diamondsuit\ \text{ltl}(4, \varphi).$
Robust Eventually	$\diamondsuit$	$\forall 1 \le i \le 4 : \text{ltl}(i, \diamondsuit\ \varphi) = \diamondsuit\ \text{ltl}(i, \varphi).$
Robust Until	$\mathcal{U}$	$\forall 1 \le i \le 4 : \text{ltl}(i, \varphi\ \mathcal{U}\ \psi) = \text{ltl}(i, \varphi)\ \mathcal{U}\ \text{ltl}(i, \psi).$
Robust Release	$\mathcal{R}$	$\text{ltl}(1, \varphi\ \mathcal{R}\ \psi) = \text{ltl}(1, \varphi)\ \mathcal{R}\ \text{ltl}(1, \psi),$ $\text{ltl}(2, \varphi\ \mathcal{R}\ \psi) = \diamondsuit\Box\ \text{ltl}(2, \psi) \vee \diamondsuit\ \text{ltl}(2, \varphi),$ $\text{ltl}(3, \varphi\ \mathcal{R}\ \psi) = \Box\diamondsuit\ \text{ltl}(3, \psi) \vee \diamondsuit\ \text{ltl}(3, \varphi),$ $\text{ltl}(4, \varphi\ \mathcal{R}\ \psi) = \diamondsuit\ \text{ltl}(4, \psi) \vee \diamondsuit\ \text{ltl}(4, \varphi).$

Formally, the truth value of an rLTL formula is a 4-tuple belonging to the set $\mathbb{B}_5$, where $\mathbb{B}_5[n] \in \mathbb{B}_5$, for $0 \le n \le 4$, is the truth value with n bits set to 1:

$$\begin{aligned}\mathbb{B}_5 &= \{0000, 0001, 0011, 0111, 1111\} \\ &= \{\mathbb{B}_5[0], \mathbb{B}_5[1], \mathbb{B}_5[2], \mathbb{B}_5[3], \mathbb{B}_5[4]\}.\end{aligned}$$

The truth values are ordered as follows:

$$0000 \prec 0001 \prec 0011 \prec 0111 \prec 1111. \quad (3)$$

Hence, a truth value in $\mathbb{B}_5$ can be viewed as a sequence of 4 bits, with 1111 corresponding to **true** and the remaining ones corresponding to different shades of **false**. For example consider the rLTL formula $\boxdot\ p$. Truth value $\mathbb{B}_5[4] = 1111$ corresponds to the LTL formula $\Box\ p$ being satisfied, $\mathbb{B}_5[3] = 0111$ corresponds to $\diamondsuit\Box\ p$ being satisfied, $\mathbb{B}_5[2] = 0011$ corresponds to $\Box\diamondsuit\ p$ being satisfied, $\mathbb{B}_5[1] = 0001$ corresponds to $\diamondsuit\ p$ being satisfied, and $\mathbb{B}_5[0] = 0000$ corresponds to $\Box\ \neg p$.

In order to introduce the semantics of rLTL, we construct four LTL formulas for each rLTL formula φ and interpret the corresponding 4 tuple of truth values as the truth value of φ. The translation of one rLTL formula to four LTL formulas is done via the ltl operator defined below.

Definition 2 (rLTL semantics). *For a set of atomic propositions $\mathcal{P}$, we define the operator*

$$\text{ltl} : \{1, \ldots, 4\} \times \text{rLTL}(\mathcal{P}) \to \text{LTL}(\mathcal{P}) \quad (4)$$

as in Table 1. The rLTL semantics is defined as a function

$$V : \left(2^{\mathcal{P}}\right)^{\omega} \times \text{rLTL}(\mathcal{P}) \to \mathbb{B}_5,$$

where for any $\sigma \in \left(2^{\mathcal{P}}\right)^{\omega}$, $\varphi \in$ rLTL($\mathcal{P}$) and $1 \le i \le 4$, the ith bit $V_i(\sigma, \varphi)$ of the valuation $V(\sigma, \varphi)$ is given by:

$$V_i(\sigma, \varphi) = W\left(\sigma, \text{ltl}(i, \varphi)\right),$$

where $W(\sigma, \psi)$ is the truth value of the LTL formula ψ evaluated on the infinite word σ.

The difficulty of dealing with the 5-valued semantics of rLTL formulas lies in the fact that the four bits of a truth value are coupled by robust implications and negations. Intuitively, negation changes true to false and *all shades of false* to true. In particular, the value of the each bit depends on the value of the leftmost bit. In a similar manner, each bit of the robust implication, needs the value of the next bit to the right.

To address this the authors in [1] identified a fragment of rLTL for which bitwise computations are possible. We present this in the next section.

3.1 The rLTL Model Checking Problem

The model checking problem for LTL asks whether or not a model (set of words) satisfies an LTL specification. In rLTL, the model checking problem is intuitively understood as the question "to what degree does a model satisfy a specification"?

Problem 1 (The Model Checking Problem for rLTL). *Given a set of atomic propositions $\mathcal{P}$, a set set of words $\mathcal{L} \subseteq \left(2^{\mathcal{P}}\right)^{\omega}$ recognized by a Generalized Büchi Automaton (GBA) $\mathcal{A}$, and $\varphi \in \text{rLTL}(\mathcal{P})$, compute*

$$b(\mathcal{L}, \varphi) = \min_{\sigma \in \mathcal{L}} V(\sigma, \varphi). \tag{5}$$

Note that $V(\sigma, \varphi) \in \mathbb{B}_5$, and the minimum follows the ordering defined in Equation (3).

In [22, Theorem 4.9], the authors provide a technique for rLTL model checking that follows the standard steps of LTL model checking. Given an rLTL formula φ, a GBA with $N_{\varphi}^{\text{rLTL}} = O(5^{|\varphi|})$ states, and $F_{\varphi}^{\text{rLTL}} = O(|\varphi|)$ accepting conditions is constructed, and intersected with the GBA recognizing the language $\mathcal{L}$. Following the standard approach for automaton-based LTL model checking the time complexity of rLTL model checking is proportional to

$$O(5^{|\varphi|}). \tag{6}$$

In [1, Theorem 3.2], the authors introduced the rLTL fragment

$$\{\psi, \psi_1 \Rrightarrow \psi_2 \mid \psi, \psi_1, \psi_2 \in \widetilde{\text{rLTL}}(\mathcal{P})\},$$

where $\widetilde{\text{rLTL}}(\mathcal{P}) \subset \text{rLTL}(\mathcal{P})$ is the set of all rLTL formulas that do not contain the operators $\Rrightarrow$ or $\mathcal{R}$. For this fragment they refined complexity bounds for the corresponding rLTL model checking problem and proved that it can be solved by performing at most four LTL model checking steps as shown in Algorithm **??**. We state their main result here in the following theorem.

Theorem 3.1. *Consider a set of atomic propositions $\mathcal{P}$, a set $\mathcal{L} \subseteq \left(2^{\mathcal{P}}\right)^{\omega}$ recognized by a GBA $\mathcal{A}$ with N states. Let φ be any formula in the rLTL fragment*

$$\{\psi, \psi_1 \Rrightarrow \psi_2 \mid \psi, \psi_1, \psi_2 \in \widetilde{\text{rLTL}}(\mathcal{P})\}. \tag{7}$$

*Algorithm **??** computes $b(\mathcal{L}, \varphi) = \mathbb{B}_5[\ell]$, $0 \leq \ell \leq 4$ by performing* $\min(\ell + 1, 4)$ *LTL model-checking steps, each using an automaton of size at most*

$$O\left(2^{|\varphi| - k(\varphi)} 3^{k(\varphi)}\right), \tag{8}$$

where $k(\varphi)$ is the number of robust always ($\boxdot$) *operators in φ.*

Our tool implements Algorithm **??**. We note that from now on, we will only be dealing with rLTL formulas that belong to the fragment described by (7).

Data: A language $\mathcal{L}$ generated by a GBA $\mathcal{A}$, a formula $\varphi \in \text{rLTL}(\mathcal{P})$.
Result: Computes $b(\mathcal{L}, \varphi)$ (see (5)).
for $j = 0, \ldots, 3$ **do**
  $w := \inf_{\sigma \in \mathcal{L}} W(\sigma, \text{ltl}(4 - j, \varphi))$.
  if $w = 0$ **then**
    return $\mathbb{B}_5[j]$
  end
end
return $\mathbb{B}_5[4]$

Algorithm 1: rLTL model checking algorithm.

4 VERIFYING ROBUST LTL FORMULAS: HERE COMES EVROSTOS!

Evrostos is an open-source tool, consisting of two different components. The first component is an rLTL-to-LTL translator for rLTL formulas in the fragment (7). An rLTL formula is translated into four LTL formulas, one for each bit of the rLTL formula. This bit-wise translation is done recursively as dictated by Table 1. The second component implements the bit-wise model-checking algorithm developed in [1], using NuMSV [4] to model-check each of the four LTL formulas. The tool starts at the rightmost bit, and terminates if for any bit the result of the LTL model checking is *false* or if all four bits are checked. In Evrostos we use the notation for the rLTL and the LTL operators as in Table 2.

4.1 Translating rLTL to LTL

The translation of an rLTL formula to four LTL formulas essentially implements the function ltl from Definition 2. This operator takes a pair of an rLTL formula and a bit i, $1 \leq i \leq 4$, as input, and returns the corresponding LTL formula for that bit. For example, the rLTL formula

$$\mathsf{p} \Rrightarrow \boxdot\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p})),$$

is transformed into the following 4 formulas:

$$\begin{aligned}
\texttt{bit1} : &(\mathsf{p} \Rightarrow \Box\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p}))) \wedge (\mathsf{p} \Rightarrow \Diamond \Box\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p}))) \\
&\wedge (\mathsf{p} \Rightarrow \Box \Diamond\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p}))) \wedge (\mathsf{p} \Rightarrow \Diamond\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p}))), \\
\texttt{bit2} : &(\mathsf{p} \Rightarrow \Diamond \Box\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p}))) \wedge (\mathsf{p} \Rightarrow \Box \Diamond\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p}))) \\
&\wedge (\mathsf{p} \Rightarrow \Diamond\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p}))), \\
\texttt{bit3} : &(\mathsf{p} \Rightarrow \Box \Diamond\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p}))) \wedge (\mathsf{p} \Rightarrow \Diamond\, (\mathsf{r}\, \mathcal{U}\, (\neg \mathsf{p}))),
\end{aligned}$$

Table 2: Syntax of rLTL and LTL in Evrostos.

rLTL			LTL		
Operator	Symbol	Evrostos Symbol	Operator	Symbol	Evrostos Symbol
Negation	$\neg$	!	Negation	$\neg$	!
Disjunction	$\vee$	|	Disjunction	$\vee$	|
Conjunction	$\wedge$	&	Conjunction	$\wedge$	&
Robust Implication	$\Rrightarrow$	=>	Implication	$\Rightarrow$	– >
Next	$\odot$	*rX*	Next	$\bigcirc$	*X*
Robust Always	$\boxdot$	*rG*	Always	$\Box$	*G*
Robust Eventually	$\Diamond$	*rF*	Eventually	$\Diamond$	*F*
Robust Until	$\mathcal{U}$	*rU*	Until	$\mathcal{U}$	*U*
Robust Release	$\mathcal{R}$	*rR*	Release	$\mathcal{R}$	*R*

bit4 : (p $\Rightarrow\Diamond$ (r $\mathcal{U}$ ($\neg$p))),

as the rLTL semantics of Table 1 dictate.

4.2 Model checking with NuSMV

The second component of our tool is built on NuSMV. We opted for NuSMV as our symbolic model verifier because it is open-source, well-documented, and widely used in industry [11, 14, 17, 20, 25]. We skip the operating details of NuSMV as our tool handles the interactions between different components internally, but a user manual can be found online [4].

For a given model, we use NuSMV to check if the LTL formula corresponding to each bit of the rLTL formula is true or false. In implementing Algorithm **??**, our tool stops the process when the truth value returned by NuSMV is false or if all bits are checked. Consider again the formula example above. NuSMV checks the bit4 formula. Assume this specification is true for a given model, then the bit3 formula is checked, and if for example it is false the process stops. Then the rLTL verification ends with the truth value 0001. The benefit of this approach against a naive implementation of an rLTL-to-LTL translation and model checking is that it stops early (if possible), thus avoiding unnecessary computations.

4.3 Interface and standard I/O

Evrostos is a user-friendly and easy to run tool. It runs from the terminal just by typing the following command: "./evrostos -options". A .txt file with the rLTL formulas can be an input to the tool, or the user can type the rLTL formula directly on the terminal. If multiple specifications should be checked for the same model the .txt file option is used with one rLTL formula per line.

Upon completion of the rLTL model checking, Evrostos compiles a report with the following contents:

- Name of model file used;
- Original rLTL formulas;
- Translated LTL formulas;
- Result of the rLTL model checking;
- Execution time of the rLTL model checking.

Now that we have introduced the main components of Evrostos, we are ready to present our experimental evaluation and comparison to classical LTL model checking.

5 EXPERIMENTAL RESULTS

In this final section, we compare the rLTL fragment (7) with the corresponding LTL fragment in terms of time complexity as indicated by the respective execution times for model checking, and show that verifying rLTL formulas provides much more information than verifying the corresponding LTL formulas.

5.1 rLTL time overhead and *False* against *Shades of False*

We now compare the execution times of rLTL model checking using Evrostos and classical LTL model checking using NuSMV. We aim to show that rLTL is not much more expensive in terms of execution time. The system models used are the automated air traffic control system described in Section 2, an abstract version of the same model [25], and a telephone system as described in [2, 16].

Table 3: Telephone System: Execution times of rLTL & LTL.

Formula	rLTL Truth Value	rLTL Time (s)	LTL Truth Value	LTL Time (s)	Time Complexity Blowup
$\boxdot(\neg(tt1 \wedge d12) \vee td2)$	0001	319.10	FALSE	265.29	1.04
$\boxdot(\neg(msg2) \vee ((d21 \wedge tcs12) \vee (d24 \wedge tcs42)))$	0011	26.71	FALSE	11.54	1.11
$\boxdot(\neg(ph1forw0) \vee \boxdot(\neg(ring1 \vee td1)))$	0001	352.07	FALSE	278.05	1.04
$\boxdot(\neg(tcs12 \wedge ring1) \vee ringt3)$	0001	157.95	FALSE	117.01	1.06
$\boxdot(\neg(msg3) \vee ((d31 \wedge tcs13) \vee (d34 \wedge tcs43)))$	0011	139.25	FALSE	55.91	1.12
$\boxdot(\neg(try1) \vee \odot(ringt1 \vee busyt1))$	0011	541.17	FALSE	220.24	1.16
$\boxdot(\neg(tt1 \wedge d13) \vee td3)$	1111	10.43	TRUE	3.01	1.26
$\boxdot(\neg(tcs13) \vee \boxdot(\neg(d31 \wedge (ringt3 \vee tt3))))$	0001	94.46	FALSE	91.92	1.00
$\boxdot(\neg(tcs42) \vee \boxdot(\neg(d24 \wedge (ringt2 \vee tt2))))$	0001	11.10	FALSE	7.53	1.05

For reference, all the simulations were conducted with a MacBook Pro (Retina, Mid 2012), with 4 cores @ 2.3 GHz Intel Core i7 Processor, and 8 GB 1600 MHz DDR3.

Tables 3 and 4 show the specifications[5] and the time taken to model check them. At a first glance, rLTL verification takes more time, something expected since the time for rLTL verification is proportional to $3^{|\varphi|}$ for an rLTL formula φ. To meaningfully compare the execution times we compute the *time complexity blowup*, ζ. Let t_{rLTL} be the time required to solve the rLTL verification problem and t_{LTL} be the corresponding time for the LTL verification problem. If $t_{LTL} = 2^{|\varphi|}$ for a formula φ, and knowing that t_{rLTL} is larger we write $t_{rLTL} = 2^{\zeta|\varphi|}$ and ask what is the exponent ζ that describes the overhead. From the two expressions above we obtain:

$$\zeta = 1 + \frac{log\left(\frac{t_{rLTL}}{t_{LTL}}\right)}{|\varphi|} \tag{9}$$

Since the time complexity of rLTL is proportional to $3^{|\varphi|}$, we have an upper bound $\bar{\zeta} = log_2(3) = 1.58$. Looking now at Tables 3 and 4, we observe that Time Complexity Blowup, ζ, is well below this upper bound and much closer to 1 in all the cases, meaning that the time complexity of rLTL for the fragment we are considering is close to that of LTL.

Finally, in Tables 3 and 4 (abstract model), the difference between a formula not being true in LTL and in rLTL is illustrated. Almost all these specifications share a common trait: a property should hold at every time step of the execution, i.e., specifications of the form $\boxdot\,\varphi$. In LTL, if it is violated even once, the truth value is false. In rLTL on the other hand, as we can see it is rarely the case that it is never satisfied. For example in Table 4 the specifications in lines 3 and 4 (abstract model) have truth values 0011 meaning that they are both satisfied and violated infinitely often over the executions of the system model. This can be interpreted intuitively as "how frequently" a formula is satisfied over the set of all infinite words of a system model. In particular, if LTL reports false, then there are in fact various rLTL truth values, giving information about the different reasons preventing different formulas from having true as their truth value. For example, the truth value 0001 in the first line in Table 3 states that a property is false but satisfied finitely many times, and the truth value 0011 in the second line that it is also false but satisfied infinitely many times. In other words, rLTL provides more fine-grained information which is one of its advantages.

[5] In Tables 3 and 4 we present the rLTL versions of the specifications tested. The LTL versions can be obtained simply by removing the dots and dashes from the operators.

Table 4: Automated Air Traffic Control System: Execution times of rLTL & LTL.

	AAC Original Model					AAC Abstract Model				
Formula	rLTL Truth Value	rLTL Time (s)	LTL Truth Value	LTL Time (s)	Time Complexity Blowup	rLTL Truth Value	rLTL Time (s)	LTL Truth Value	LTL Time (s)	Time Complexity Blowup
$\boxdot (\neg(ctr1 \wedge ta12non \wedge ta13non \wedge ta23non \wedge(\odot \neg ta12at) \wedge (\odot \neg ta13at) \wedge (\odot \neg ta23at)) \vee (\odot (\odot ctr1)))$	1111	12.82	TRUE	3.43	1.08	1111	1.03	TRUE	0.24	1.09
$\boxdot (\neg(ta12bt \wedge \neg ta13at \wedge \neg ta23at \wedge ctr0 \wedge tsscmd \wedge tswin) \vee (\ominus (tscmd1 \vee tscmd2)))$	1111	1.81	TRUE	0.35	1.12	1111	0.51	TRUE	0.08	1.13
$\boxdot (\neg(ta12at \wedge \neg ta13at \wedge \neg ta23at \wedge ctr0 \wedge \neg ac1tscmd \wedge \neg ac2tscmd) \vee (\diamondsuit (tscmd1 \vee tscmd2)))$	1111	726.79	TRUE	291.00	1.06	0011	1.01	FALSE	0.31	1.08
$\boxdot (\neg(ac1tscmddone \wedge \neg ac1tscmd \wedge tsctr1) \vee \odot (\neg tsctr1))$	1111	13.11	TRUE	3.78	1.15	0011	0.51	FALSE	0.08	1.22
$\boxdot (\neg(ac1tccmd \wedge \neg ac1tscmddone) \vee ((\neg ac1tscmddone)\ \mathcal{U}\ ac1tccmddone))$	1111	1.55	TRUE	0.29	1.22	0000	0.26	FALSE	0.04	1.25
$\boxdot (\neg(ac1tscmd \wedge \neg ac1ctrcmddone) \vee ((\neg ac1ctrcmddone)\ \mathcal{U}\ ac1tscmddone))$	1111	144.52	TRUE	35.03	1.19	0001	0.33	FALSE	0.08	1.19
$\boxdot (\neg(\neg ta12non) \vee (\diamondsuit ta12non))$	1111	140.85	TRUE	25.87	1.35	0001	0.30	FALSE	0.08	1.27
$(\boxdot\diamondsuit (ta12non \wedge ta13non)) \Rrightarrow (\boxdot (\diamondsuit (\neg tsctr1)))$	1111	94.68	TRUE	14.34	1.21	1111	0.59	TRUE	0.10	1.20
$\boxdot (\neg(\neg ctr1 \wedge \neg ac1ctrcmddone) \vee (((\neg ac1ctrcmddone)\ \mathcal{U}\ ctr1) \vee (\diamondsuit ac1tscmddone)))$	0001	157.57	FALSE	80.31	1.06	0001	0.36	FALSE	0.09	1.13
$\boxdot (\neg(tsctr1 \wedge \neg ac1ctrcmddone) \vee (((\neg ac1ctrcmddone)\ \mathcal{U}\ ctr1) \vee (\diamondsuit ac1tscmddone)))$	0001	133.03	FALSE	81.52	1.05	0001	0.36	FALSE	0.09	1.14
$\boxdot (\neg(ac1tscmd \wedge \neg ac2tscmd \wedge \neg ac2tscmddone) \vee ((\neg ac1tscmddone \wedge \neg ac2tscmddone)\ \mathcal{U}\ ac1tscmddone))$	0011	154.24	FALSE	91.93	1.04	0001	0.70	FALSE	0.16	1.13
$\boxdot (\neg(tscmd1 \wedge ctr1) \vee (\odot (\neg ctr1)))$	1111	4.83	TRUE	1.53	1.18	1111	0.52	TRUE	0.09	1.28

6 CONCLUSIONS

In this paper we presented *Evrostos: The rLTL Verifier*, an open-source, publicly available tool, and the first to perform rLTL model checking. It operates on an rLTL fragment for which the time complexity of the verification problem approaches that of LTL. We provided a pedagogical example of a controller specification being vacuously true in LTL, but not in rLTL, proving that this particular controller was not robust. Moreover, we showed that the execution time for rLTL verification is within acceptable ranges compared to the corresponding LTL verification execution time, and also the truth values of rLTL make it more informative compared to LTL. We believe this new tool will motivate the widespread use of rLTL to specify and verify robustness properties.

REFERENCES

[1] T. Anevlavis, M. Philippe, D. Neider, and P. Tabuada. 2018. Verifying rLTL formulas: now faster than ever before!. In *2018 IEEE Conference on Decision and Control (CDC)*. 1556–1561. https://doi.org/10.1109/CDC.2018.8619014

[2] Shoham Ben-David, Baruch Sterin, Joanne M. Atlee, and Sandy Beidu. 2015. Symbolic Model Checking of Product-line Requirements Using SAT-based Methods. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 189–199.

[3] Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, Georg Hofferek, Barbara Jobstmann, Bettina Könighofer, and Robert Könighofer. 2014. Synthesizing Robust Systems. *Acta Inf.* 51, 3-4 (June 2014), 193–220.

[4] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. 2002. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV '02)*. Springer-Verlag, London, UK, UK, 359–364.

[5] Edmund M Clarke, Orna Grumberg, and Doron Peled. 1999. *Model checking*. MIT press.

[6] Eric Dallal, Daniel Neider, and Paulo Tabuada. 2016. Synthesis of safety controllers robust to unmodeled intermittent disturbances. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE, 7425–7430.

[7] Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 92–106.

[8] Heinz Erzberger and K Heere. 2010. Algorithm and operational concept for resolving short-range conflicts. In *Proceedings of The Institution of Mechanical Engineers Part G-journal of Aerospace Engineering - PROC INST MECH ENG G-J A E*, Vol. 224. 225–243.

[9] Georgios E Fainekos and George J Pappas. 2006. Robustness of temporal logic specifications. In *Formal Approaches to Software Testing and Runtime Verification*. Springer, 178–192.

[10] Georgios E Fainekos and George J Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410, 42 (2009), 4262–4291.

[11] Xiang Gan, Jori Dubrovin, and Keijo Heljanko. 2014. A symbolic model checking approach to verifying satellite onboard software. *Science of Computer Programming* 82 (2014), 44 – 55. Special Issue on Automated Verification of Critical Systems (AVoCS'11).

[12] Rafal Goebel, Joao Hespanha, Andrew R Teel, Chaohong Cai, and Ricardo Sanfelice. 2004. Hybrid systems: Generalized solutions and robust stability. *IFAC Proceedings Volumes* 37, 13 (2004), 1–12.

[13] Rafal Goebel, Ricardo G Sanfelice, and Andrew R Teel. 2012. *Hybrid Dynamical Systems: modeling, stability, and robustness*. Princeton University Press.

[14] J. Lahtinen, J. Valkonen, K. Björkman, J. Frits, I. Niemelä, and K. Heljanko. 2012. Model checking of safety-critical software in the nuclear engineering domain. *Reliability Engineering & System Safety* 105 (2012), 104 – 113. ESREL 2010.

[15] Rupak Majumdar, Elaine Render, and Paulo Tabuada. 2013. A theory of robust omega-regular software synthesis. *ACM Transactions on Embedded Computing Systems (TECS)* 13, 3 (2013), 48.

[16] Malte Plath and Mark Ryan. 2001. Feature integration using a feature construct. *Science of Computer Programming* 41, 1 (2001), 53 – 84.

[17] Kristin Y. Rozier. 2011. Linear Temporal Logic Symbolic Model Checking. *Computer Science Review* 5, 2 (2011), 163 – 203.

[18] Matthias Rungger and Paulo Tabuada. 2016. A notion of robustness for cyber-physical systems. *IEEE Trans. Automat. Control* 61, 8 (2016), 2108–2123.

[19] Philippe Schnoebelen. 2002. The Complexity of Temporal Logic Model Checking. *Advances in modal logic* 4, 393-436 (2002), 35.

[20] Alireza Souri, Nima Jafari Navimipour, and Amir Masoud Rahmani. 2018. Formal verification approaches and standards in the cloud computing: A comprehensive and systematic review. *Computer Standards & Interfaces* 58 (2018), 1 – 22.

[21] Paulo Tabuada, Sina Yamac Caliskan, Matthias Rungger, and Rupak Majumdar. 2014. Towards robustness for cyber-physical systems. *IEEE Trans. Automat. Control* 59, 12 (2014), 3151–3163.

[22] Paulo Tabuada and Daniel Neider. 2015. Robust Linear Temporal Logic. *arXiv preprint arXiv:1510.08970* (2015).

[23] Danielle C Tarraf, Alexandre Megretski, and Munther A Dahleh. 2008. A framework for robust stability of systems over finite alphabets. *IEEE Trans. Automat. Control* 53, 5 (2008), 1133–1146.

[24] Yih-Kuen Tsay, Ming-Hsien Tsai, Jinn-Shu Chang, and Yi-Wen Chang. 2011. Büchi store: an open repository of büchi automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 262–266.

[25] Yang Zhao and Kristin Yvonne Rozier. 2014. Formal Specification and Verification of a Coordination Protocol for an Automated Air Traffic Control System. *Sci. Comput. Program.* 96, P3 (Dec. 2014), 337–353.