

Robust, Expressive, and Quantitative Linear Temporal Logics: Pick any Two for Free

Daniel Neider

Max Planck Institute for Software Systems
67663 Kaiserslautern, Germany
neider@mpi-sws.org

Alexander Weinert

German Aerospace Center (DLR)
Simulation and Software Technology
51147 Cologne, Germany
alexander.weinert@dlr.de

Martin Zimmermann

University of Liverpool
Liverpool L69 3BX, United Kingdom
martin.zimmermann@liverpool.ac.uk

Linear Temporal Logic (LTL) is the standard specification language for reactive systems and is successfully applied in industrial settings. However, many shortcomings of LTL have been identified in the literature, among them the limited expressiveness, the lack of quantitative features, and the inability to express robustness. There is work on overcoming these shortcomings, but each of these is typically addressed in isolation. This is insufficient for applications where all shortcomings manifest themselves simultaneously.

Here, we tackle this issue by introducing logics that address more than one shortcoming. To this end, we combine the logics Linear Dynamic Logic, Prompt-LTL, and robust LTL, each addressing one aspect, to new logics. For all combinations of two aspects, the resulting logic has the same desirable algorithmic properties as plain LTL. In particular, the highly efficient algorithmic backends that have been developed for LTL are also applicable to these new logics. Finally, we discuss how to address all three aspects simultaneously.

1 Introduction

Linear Temporal Logic (LTL) [21] is amongst the most prominent and most important specification languages for reactive systems, e.g., non-terminating controllers interacting with an antagonistic environment. Verification of such systems against LTL specifications is routinely applied in industrial settings nowadays [10, 13]. Underlying this success story is the exponential compilation property [25]: every LTL formula can be effectively translated into an equivalent Büchi automaton of exponential size (and it turns out that this upper bound is tight). In fact, almost all verification algorithms for LTL are based on this property, which is in particular true for the popular polynomial space model checking algorithm and the doubly-exponential time synthesis algorithms. Other desirable properties of LTL include its compact and variable-free syntax and its intuitive semantics.

Despite the success of LTL, a plethora of extensions of LTL have been studied, all addressing individual and specific shortcomings of LTL, e.g., its limited expressiveness, its lack of quantitative features, and its inability to express robustness. Commonly, extensions of LTL as described above are only studied in isolation—the logics are either more expressive, or quantitative, or robust. One notable exception is Parametric LDL (PLDL) [12], which adds quantitative operators and increased expressiveness while maintaining the exponential compilation property and intuitive syntax and semantics. In practical settings, however, it does not suffice to address one shortcoming of LTL while ignoring the others. Instead,

one needs a logic that combines multiple extensions while still maintaining the desirable properties of LTL. The overall goal of this paper is, hence, to bridge this gap, thereby enabling expressive, quantitative, and robust verification and synthesis.

It is a well-known fact that LTL is strictly weaker than Büchi automata, i.e., it does not harness the full expressive power of the algorithmic backends. Thus, increasing the expressiveness of LTL has generated much attention [17, 24, 25, 26] as it can be easily exploited: as long as the new logic also has the exponential compilation property, the same optimized backends as for LTL can be used. A prominent and recent example of such an extension that yields the full expressive power of Büchi automata is Linear Dynamic Logic (LDL) [24], which adds to LTL temporal operators guarded by regular expressions. As an example, consider the specification “ p holds at every even time point, but may or may not hold at odd time points”. It is well-known that this property is not expressible in LTL, as LTL, intuitively, is unable to count modulo a fixed number. However, the specification is easily expressible in LDL as $[r]p$, where r is the regular expression $(\tau\tau \cdot \tau\tau)^*$. The formula requires p to be satisfied at every position j such that the prefix up to position j matches the regular expression r (which is equivalent to j being even), i.e., $\tau\tau$ is an atomic regular expression that matches every letter. In this work, we consider LDL instead of the alternatives cited above for its conceptual simplicity: LDL has a simple and variable-free syntax based on regular expressions as well as intuitive semantics (assuming some familiarity with regular expressions).

Another serious shortcoming of LTL (and LDL) is its inability to adequately express timing bounds. For example, consider the specification “every request q is eventually answered by a response p ”, which is expressed in LTL as $\Box(q \rightarrow \Diamond p)$. It is satisfied, even if the waiting time between requests q and responses p diverges to infinity, although such a behavior is typically undesired. Again, a long line of research has addressed this second shortcoming of LTL [1, 12, 15, 16, 28]. The most basic one is Prompt-LTL [16], which adds the prompt-eventually operator \Diamond_p to LTL. The semantics is now defined with an additional parameter k , which bounds the scope of \Diamond_p : $\Box(q \rightarrow \Diamond_p p)$ requires every request q to be answered within k steps, when evaluated with respect to k . The resulting logic is a quantitative one: either one quantifies the parameter k existentially and obtains a boundedness problem, e.g., “is there a bound k such that every request can be answered within k steps”, or one even aims to determine the optimal bound k . Again, Prompt-LTL retains the desirable properties of LTL, i.e., the exponential compilation property as well as intuitive syntax and semantics. Furthermore, Prompt-LTL captures the technical core of the alternatives cited above, e.g., decision problems for the more general logic PLTL [1] can be reduced to those for Prompt-LTL. For these reasons, we study Prompt-LTL in this work.

Finally, a third line of extensions of LTL is concerned with the concept of robustness, which is much harder to formalize. This is reflected by a multitude of incomparable notions of robustness in verification [5, 6, 8, 9, 11, 18, 20, 22, 23]. Here, we are interested in robust LTL (rLTL) [23], which equips LTL with a five-valued semantics that captures different degrees of violations of universal specifications. As an example, consider the specification “if property φ always holds true, then property ψ also always holds true”, which is expressed in LTL as $\Box\varphi \rightarrow \Box\psi$ and is typical for systems that have to interact with an antagonistic environment. In classical semantics, the whole formula is satisfied as soon as the assumption φ is violated once, even if the guarantee ψ is violated as well. By contrast, the semantics of robust LTL ensures that the degree of the violation of $\Box\psi$ is always proportional to the degree of the violation of $\Box\varphi$. To this end, the degree of a violation of a property $\Box\varphi$ is expressed by five different truth values: either φ always holds, or φ is violated only finitely often, violated infinitely often, violated almost always, or violated always. Again, robust LTL has the exponential compilation property and an intuitive syntax (though its semantics is more intricate). In this work, we consider robust LTL, as it is the first logic that intrinsically captures the notion of robustness in LTL. In particular, formulas of robust LTL are evaluated over traces with Boolean truth values for atomic propositions and do not require

for the non-strict variant of \prec and define $\min \emptyset = 1111$ and $\max \emptyset = 0000$ when the operators range over subsets of \mathbb{B}_4 .

Throughout this work, we fix a finite non-empty set P of atomic propositions. For a set $A \subseteq P$ and a propositional formula ϕ over P , we write $A \models \phi$ if the variable valuation mapping elements in A to 1 and elements in $P \setminus A$ to 0 satisfies ϕ . A trace (over P) is an infinite sequence $w \in (2^P)^\omega$. Given a trace $w = w(0)w(1)w(2)\cdots$ and a position $j \in \mathbb{N}$, we define $w[0, j] = w(0)\cdots w(j-1)$ and $w[j, \infty) = w(j)w(j+1)w(j+2)\cdots$, i.e., $w[0, j]$ is the prefix of length j of w and $w[j, \infty)$ the remaining suffix. In particular, $w[0, 0]$ is empty and $w[0, \infty)$ is w .

Our work is based on three logics, Robust Linear Temporal Logic ($\text{rLTL}(\Box, \Diamond)$) [23], Linear Dynamic Logic (LDL) [24], and Prompt Linear Temporal Logic (Prompt-LTL) [16], which we briefly review in the following three subsections. More formal definitions can be found in the original publications introducing these logics and in the full version of this work [19].

We define the semantics of all these logics by evaluation functions V mapping a trace, a formula, and a bound (in the case of a quantitative logic) to a truth value. This is prudent for robust semantics, hence we also use this approach for the other logics, which are typically defined via satisfaction relations. In particular, V^R , V^D , and V^P denote the evaluation functions of $\text{rLTL}(\Box, \Diamond)$, LDL, and Prompt-LTL, respectively. Nevertheless, our definitions here are equivalent to the original definitions.

2.1 Robust Linear Temporal Logic

The main impetus behind the introduction of robust LTL was the need to capture the concept of robustness in temporal logics. As a first motivating example consider the LTL formula $\Box p$, stating that p holds at every position. Consequently, the formula is violated if there is a single position where p does not hold. However, this is a very ‘‘mild’’ violation of the property and there are much more ‘‘severe’’ violations. As exhibited by Tabuada and Neider, there are four canonical *degrees* of violation of $\Box p$: (i) p is violated at finitely many positions, (ii) p is violated at infinitely many positions, (iii) p is violated at all but finitely many positions, and (iv) p is violated at all positions. These first three degrees are captured by the LTL formulas $\Diamond \Box p$, $\Box \Diamond p$, and $\Diamond p$, which are all weakenings of $\Box p$. All five possibilities, satisfaction and four degrees of violation, are captured in robust LTL by the truth values

$$1111 \succ 0111 \succ 0011 \succ 0001 \succ 0000$$

introduced above. By design, the formula $\Box p$ of robust LTL¹ has

- truth value 1111 on all traces where p holds at all positions,
- truth value 0111 on all traces where p holds at all but finitely many positions,
- truth value 0011 on all traces where p holds at infinitely many positions and does not hold at infinitely many positions,
- truth value 0001 on all traces where p only holds at finitely many positions, and
- truth value 0000 on all traces where p holds at no position.

As a further example, consider the formula $\Box p \rightarrow \Box q$. For this formula, the robust semantics captures the intuition described in the introduction: the implication is satisfied (i.e., has truth value 1111), if

¹Following the precedent for robust LTL, we use dots to distinguish operators of robust logics from those of classical logics throughout the paper.

the degree of violation of the property “always q ” is at most the degree of violation of the property “always p ”. Thus, if p is violated finitely often, then q may also be violated finitely often (but not infinitely often) while still satisfying the implication.

Conjunction and disjunction are defined as usual using minimization and maximization relying on the order indicated above while negation is based on the intuition that 1111 represents satisfaction and all other truth values represent degrees of violation. Hence, a negation $\neg\varphi$ is satisfied (i.e., has truth value 1111), if φ has truth value less than 1111, and it is violated (i.e., has truth value 0000) if φ has truth value 1111. Finally, the semantics of the eventually operator is defined as usual, i.e., the truth value of $\diamond\varphi$ on w is the maximal truth value that is assumed by φ on some suffix of w .

This intuition is formalized in the evaluation function V^R , which maps a trace $w \in (2^P)^\omega$ and an rLTL(\square, \diamond) formula φ to a truth value $V^R(w, \varphi)$ in \mathbb{B}_4 [23]. Note that, for the sake of simplicity, we restrict ourselves to rLTL(\square, \diamond), i.e., robust LTL with the always and eventually operators, but without next, until, and release. We comment on the effect of this restriction when defining the combinations of logics.

2.2 Linear Dynamic Logic

The logic LDL has only two temporal operators, $\langle r \rangle$ and $[r]$, which can be understood as guarded variants of the classical eventually and always operators from LTL, respectively. Both are guarded by regular expressions r over the atomic propositions that may contain tests, which are again LDL formulas. These two operators together with Boolean connectives capture the full expressive power of the ω -regular expressions, i.e., LDL exceeds the expressiveness of LTL.

Formally, a formula $\langle r \rangle \varphi$ is satisfied by a trace w , if there is some j such that the prefix $w[0, j)$ matches the regular expression r and the corresponding suffix $w[j, \infty)$ satisfies φ . Dually, a formula $[r] \varphi$ is satisfied by a trace w if for every j with $w[0, j)$ matching r , $w[j, \infty)$ satisfies φ . Thus, while the classical eventually and always operator range over all positions, the operators of LDL range only over those positions whose induced prefix matches the guard of the operator.

The semantics of LDL is captured by the evaluation function V^D mapping a trace $w \in (2^P)^\omega$ and an LDL formula φ to a truth value $V^D(w, \varphi)$ in \mathbb{B} [7, 24].

2.3 Prompt Linear Temporal Logic

To express timing constraints, the logic Prompt-LTL adds the prompt-eventually operator \diamond_p to LTL. For technical reasons [1], this requires to disallow negation and implication. Intuitively, the new operator requires its argument to be satisfied within a bounded number of steps.

Thus, the semantics is given by an evaluation function V^P that maps a trace $w \in (2^P)^\omega$, a Prompt-LTL formula φ , and a bound $k \in \mathbb{N}$ to a truth value $V^P(w, k, \varphi)$ in \mathbb{B} [16]. This function is defined as usual for all Boolean and standard temporal operators (ignoring the bound k), while a formula $\diamond_p \varphi$ is satisfied with respect to the bound k if φ holds within the next k steps, i.e., the prompt-eventually behaves like the classical eventually with a bounded scope.

3 Robust and Prompt Linear Temporal Logic

We begin our treatment of combinations of the three basic logics by introducing robust semantics for Prompt-LTL, obtaining the logic rPrompt-LTL. To this end, we add the prompt-eventually operator to

rLTL(\square, \diamond) while disallowing implications and restricting negation to retain decidability (cf. [1]). The formulas of rPrompt-LTL are given by

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \diamond \varphi \mid \square \varphi \mid \diamond_p \varphi,$$

where p ranges over the set P of atomic propositions. The size $|\varphi|$ of a formula φ is the number of its distinct subformulas.

The semantics of rPrompt-LTL is given by an evaluation function V^{RP} mapping a trace w , a bound k for the prompt-eventualities, and a formula φ to a truth value in \mathbb{B}_4 . To simplify our notation, we write $V_i^{\text{RP}}(w, k, \varphi)$ for $i \in \{1, 2, 3, 4\}$ to denote the i -th bit of $V^{\text{RP}}(w, k, \varphi)$, i.e.,

$$V^{\text{RP}}(w, k, \varphi) = V_1^{\text{RP}}(w, k, \varphi)V_2^{\text{RP}}(w, k, \varphi)V_3^{\text{RP}}(w, k, \varphi)V_4^{\text{RP}}(w, k, \varphi).$$

The semantics of Boolean connectives as well as of the eventually and always operators is defined as for robust LTL. The motivation behind these definitions is carefully and convincingly discussed by Tabuada and Neider [23]. The semantics of the prompt-eventually operator bounds its scope to the next k positions as in classical Prompt-LTL [16].

- $V^{\text{RD}}(w, k, p) = \begin{cases} 1111 & \text{if } p \in w(0), \\ 0000 & \text{if } p \notin w(0), \end{cases}$ • $V^{\text{RD}}(w, k, \neg p) = \begin{cases} 1111 & \text{if } p \notin w(0), \\ 0000 & \text{if } p \in w(0), \end{cases}$
- $V^{\text{RD}}(w, k, \varphi_0 \wedge \varphi_1) = \min\{V^{\text{RD}}(w, k, \varphi_0), V^{\text{RD}}(w, k, \varphi_1)\},$
- $V^{\text{RD}}(w, k, \varphi_0 \vee \varphi_1) = \max\{V^{\text{RD}}(w, k, \varphi_0), V^{\text{RD}}(w, k, \varphi_1)\},$
- $V^{\text{RP}}(w, k, \diamond \varphi) = b_1 b_2 b_3 b_4$ where $b_i = \max_{j \in \mathbb{N}} V_i^{\text{RP}}(w[j, \infty), k, \varphi)$,² and
- $V^{\text{RP}}(w, k, \square \varphi) = b_1 b_2 b_3 b_4$ where
 - $b_1 = \min_{j \in \mathbb{N}} V_1^{\text{RP}}(w[j, \infty), k, \varphi)$, i.e., $b_1 = 1$ iff φ holds always,
 - $b_2 = \max_{j' \in \mathbb{N}} \min_{j' \leq j} V_2^{\text{RP}}(w[j, \infty), k, \varphi)$, i.e., $b_2 = 1$ iff φ holds almost always,
 - $b_3 = \min_{j' \in \mathbb{N}} \max_{j' \leq j} V_3^{\text{RP}}(w[j, \infty), k, \varphi)$, i.e., $b_3 = 1$ iff φ holds infinitely often, and
 - $b_4 = \max_{j \in \mathbb{N}} V_4^{\text{RP}}(w[j, \infty), k, \varphi)$ i.e., $b_4 = 1$ iff φ holds at least once.
- $V^{\text{RP}}(w, k, \diamond_p \varphi) = b_1 b_2 b_3 b_4$ where $b_i = \max_{0 \leq j \leq k} V_i^{\text{RP}}(w[j, \infty), k, \varphi)$.

It is easy to verify that $V^{\text{RP}}(w, k, \varphi)$ is well-defined, i.e., $V^{\text{RP}}(w, k, \varphi) \in \mathbb{B}_4$ for all w, k , and φ .

Example 1. Consider the formula $\varphi = \square \diamond_p s$, where we interpret occurrences of the atomic proposition s as synchronizations. Then, the different degrees of satisfaction of the formula express the following possibilities, when evaluating it with respect to $k \in \mathbb{N}$: (i) the distance between synchronizations is bounded by k , (ii) from some point onwards, the distance between synchronizations is bounded by k , (iii) there are infinitely many synchronizations, and (iv) there is at least one synchronization. Note that the last two possibilities are independent of k , which is explained by simple logical equivalences, e.g., the third possibility reads actually as follows: there are infinitely many positions such that a synchronization occurs within distance k . However, it is easy to see that is equivalent to the property stated above.

In the next two sections, we solve the model checking problem and the synthesis problem for rPrompt-LTL. To this end, we translate every rPrompt-LTL formula into a sequence of five Prompt-LTL formulas that capture the five degrees of satisfaction and violation by making the semantics of the robust always operator explicit. This is a straightforward generalization of the, in the terms of the introduction, reduction-based approach to robust LTL [23].

²This definition is equivalent to $V^{\text{RP}}(w, k, \diamond \varphi) = \max_{j \in \mathbb{N}} V^{\text{RP}}(w[j, \infty), k, \varphi)$ due to monotonicity of the truth values, which is closer to the classical semantics of the eventually operator. A similar equivalence holds for $\diamond_p \varphi$.

Lemma 1. *For every rPrompt-LTL formula φ and every $\beta \in \mathbb{B}_4$, there is a Prompt-LTL formula φ_β of size $\mathcal{O}(|\varphi|)$ such that $V^{\text{RP}}(w, k, \varphi) \succeq \beta$ if and only if $V^{\text{P}}(w, k, \varphi_\beta) = 1$.*

Note that the logic $\text{rLTL}(\Box, \Diamond)$ is not a fragment of rPrompt-LTL as we have to disallow negation and implication to retain decidability [1]. Conversely, Prompt-LTL is also not a fragment of rPrompt-LTL as we omitted the next, until, and release operator. However, we present a reduction-based approach from rPrompt-LTL to Prompt-LTL. Thus, one could easily add the additional temporal operators to rPrompt-LTL while maintaining the result of Lemma 1. We prefer not to do so for the sake of accessibility and brevity.

3.1 Model Checking

Let us now consider the rPrompt-LTL model checking problem, which asks whether all executions of a given finite transition system satisfy a given specification expressed as an rPrompt-LTL formula with truth value at least $\beta \in \mathbb{B}_4$. More formally, we assume the system under consideration to be modeled as a (labeled and initialized) transition system $\mathcal{S} = (S, s_I, E, \lambda)$ over P consisting of a finite set S of states containing the initial state s_I , a directed edge relation $E \subseteq S \times S$, and a state labeling $\lambda: S \rightarrow 2^P$ that maps each state to the set of atomic propositions that hold true in this state. A path through \mathcal{S} is a sequence $\rho = s_0 s_1 s_2 \dots$ satisfying $s_0 = s_I$ and $(s_j, s_{j+1}) \in E$ for every $j \in \mathbb{N}$, and $\Pi_{\mathcal{S}}$ denotes the set of all paths through \mathcal{S} . Finally, the trace of a path $\rho = s_0 s_1 s_2 \dots \in \Pi_{\mathcal{S}}$ is the sequence $\lambda(\rho) = \lambda(s_0)\lambda(s_1)\lambda(s_2)\dots$ of labels induced by ρ .

Problem 1. *Let φ be an rPrompt-LTL formula, \mathcal{S} a transition system, and $\beta \in \mathbb{B}_4$. Is there a $k \in \mathbb{N}$ such that $V^{\text{RP}}(\lambda(\rho), k, \varphi) \succeq \beta$ holds true for all paths $\rho \in \Pi_{\mathcal{S}}$?*

Our solution relies on Lemma 1 and on Prompt-LTL model checking being in PSPACE [16].

Theorem 1. *rPrompt-LTL model checking is in PSPACE.*

We do not claim PSPACE-hardness because model checking the fragment of LTL with disjunction, conjunction, always, and eventually operators only (and classical semantics) is NP-complete [3]. Since this fragment can be embedded into rPrompt-LTL (via a translation of this LTL fragment into rPrompt-LTL using techniques similar to those presented by Tabuada and Neider [23] for translating LTL into $\text{rLTL}(\Box, \Diamond)$), we obtain at least NP-hardness for Problem 1. As we have no next, until, and release operators (by our own volition), we cannot easily claim PSPACE-hardness. In contrast, the solution of the Prompt-LTL model checking problem consists of a reduction to LTL model checking that introduces until operators (see [16]). Hence, we leave the fragment mentioned above, for which NP membership is known. However, adding next, until, and release to rPrompt-LTL yields a PSPACE-hard model checking problem.

3.2 Synthesis

Next, we consider the problem of synthesizing reactive controllers from rPrompt-LTL specifications. In this context, we rely on the classical reduction from reactive synthesis to infinite-duration two-player games over finite graphs. In particular, we show how to construct a finite-state winning strategy for games with rPrompt-LTL winning conditions, which immediately correspond to implementations of reactive controllers. Throughout this section, we assume familiarity with games over finite graphs (see, e.g., [14, Chapter 2]).

We consider rPrompt-LTL games over P , which are triples $\mathcal{G} = (G, \varphi, \beta)$ consisting of a labeled game graph G , an rPrompt-LTL formula φ , and a truth value $\beta \in \mathbb{B}_4$. A labeled game graph $G = (V_0, V_1, E, \lambda)$

consists of a directed graph $(V_0 \cup V_1, E)$, two finite, disjoint sets of vertices V_0 and V_1 , and a function $\lambda: V_0 \cup V_1 \rightarrow 2^P$ mapping each vertex v to the set $\lambda(v)$ of atomic propositions that hold true in v . We denote the set of all vertices by $V = V_0 \cup V_1$ and assume that game graphs do not have terminal vertices, i.e., $\{v\} \times V \cap E \neq \emptyset$ for each $v \in V$.

As in the classical setting, rPrompt-LTL games are played by two players, Player 0 and Player 1, who move a token along the edges of the game graph ad infinitum (if the token is currently placed on a vertex $v \in V_i, i \in \{0, 1\}$, then Player i decides the next move). The resulting infinite sequence $\rho = v_0 v_1 v_2 \dots \in V^\omega$ of vertices is called a play and induces a trace $\lambda(\rho) = \lambda(v_0)\lambda(v_1)\lambda(v_2)\dots \in (2^P)^\omega$.

A strategy of Player 0 is a mapping $f: V^*V_0 \rightarrow V$ that prescribes where to move the token depending on the finite play prefix constructed so far. A play $v_0 v_1 v_2 \dots$ is played according to f if $v_{j+1} = f(v_0 \dots v_j)$ for every j with $v_j \in V_0$. A strategy f of Player 0 is winning from a vertex $v \in V$ if there is a $k \in \mathbb{N}$ such that all plays ρ that start in v and that are played according to f satisfy $V^{\text{RP}}(\lambda(\rho), k, \varphi) \succeq \beta$, i.e., the evaluation of φ with respect to k on $\lambda(\rho)$ determines the winner of the play ρ . Further, a (winning) strategy is a finite-state strategy if there exists a finite-state machine computing it in the usual sense (see [14, Chapter 2] for details).

We are interested in solving rPrompt-LTL games, i.e., in solving the following problem.

Problem 2. *Let \mathcal{G} be an rPrompt-LTL game and v a vertex. Determine whether Player 0 has a winning strategy for \mathcal{G} from v and compute a finite-state winning strategy if so.*

Again, our solution to this problem relies on Lemma 1 and the fact that solving Prompt-LTL games is in 2EXPTIME [16, 27].

Theorem 2. *Solving rPrompt-LTL games is 2EXPTIME-complete.*

Here we have a matching lower bound, as solving games with LTL conditions without next, until, and release is already 2EXPTIME-hard [2].

4 Robust Linear Dynamic Logic

Next, we “robustify” LDL by generalizing the ideas underlying robust LTL to LDL, obtaining the logic rLDL. Again, following the precedent of robust LTL, we equip robust operators with dots to distinguish them from non-robust ones. The formulas of rLDL are given by the grammar

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \langle \cdot r \cdot \rangle \varphi \mid [\cdot r \cdot] \varphi \quad r ::= \phi \mid \varphi? \mid r+r \mid r;r \mid r^*,$$

where p ranges over the atomic propositions in P and ϕ over propositional formulas over P . We refer to formulas of the form $\langle \cdot r \cdot \rangle \varphi$ and $[\cdot r \cdot] \varphi$ as diamond formulas and box formulas, respectively. In both cases, r is the guard of the operator. An atom $\varphi?$ of a regular expression is a test. We use the abbreviations $\text{tt} = p \vee \neg p$ and $\text{ff} = p \wedge \neg p$ for some $p \in P$ and note that both are formulas and guards. We denote the set of subformulas of φ by $\text{cl}(\varphi)$. Guards are not subformulas, but the formulas appearing in the tests are, e.g., we have $\text{cl}(\langle \cdot p?; q \cdot \rangle p') = \{p, p', \langle \cdot p?; q \cdot \rangle p'\}$. The size $|\varphi|$ of φ is the sum of $|\text{cl}(\varphi)|$ and the sum of the lengths of the guards appearing in φ (counted with multiplicity and measured in the number of operators).

Before we introduce the semantics of rLDL we first recall the semantics of the robust always operator $\Box\varphi$ in robust LTL. To this end, call a position j of a trace φ -satisfying if the suffix starting at position j satisfies φ . Now, the robust semantics are based on the following five cases, where the latter four distinguish various degrees of violating the formula $\Box\varphi$: either all positions are φ -satisfying

(\square), almost all positions are φ -satisfying ($\diamond\square$), infinitely many positions are φ -satisfying ($\square\diamond$), some position is φ -satisfying (\diamond), or no position is φ -satisfying.

A similar approach for a formula $[\cdot r] \varphi$ would be to consider the following possibilities, where a position j of a trace w is an r -match if the prefix of w up to and including position $j - 1$ is in the language of r : all r -matches are φ -satisfying, almost all r -matches are φ -satisfying, infinitely many r -matches are φ -satisfying, some r -match is φ -satisfying, or no r -match is φ -satisfying. On a trace w with infinitely many r -matches, this is the natural generalization of the robust semantics. A trace, however, may only contain finitely many r -matches, or none at all. In the former case, there are not infinitely many φ -satisfying r -matches, but all r -matches could satisfy φ . Thus, the monotonicity of the cases is violated. We overcome this by interpreting “almost all” as “all” and “infinitely many” as “some” if there are only finitely many r -matches.³

Also, the guard r may contain tests, which have to be evaluated to determine whether a position is an r -match. For this, we have to use the appropriate semantics for the robust box operator. For example, if we interpret $[\cdot r] \varphi$ to mean “almost all r -matches satisfy φ ”, then the robust box operators in tests of r are evaluated with this interpretation as well. This may, however, violate monotonicity (see Example 3), which we therefore hardcode in the semantics.

We now formalize the informal description above and subsequently show that this formalization satisfies all desired properties. To this end, we again define an evaluation function V^{RD} mapping a trace w and a formula φ to a truth value. Also, we again denote the projection of $V^{\text{RD}}(w, \varphi)$ to its i -th bit by $V_i^{\text{RD}}(w, \varphi)$. For atomic propositions and Boolean connectives, the definition is the same as for rPrompt-LTL introduced above (ignoring the bound k) and for negation and implication, the definition is the same as for robust LTL (cf. [23]):

- $V^{\text{RD}}(w, \neg\varphi) = \begin{cases} 0000 & \text{if } V^{\text{RD}}(w, \varphi) = 1111, \\ 1111 & \text{if } V^{\text{RD}}(w, \varphi) \neq 1111, \end{cases}$ and
- $V^{\text{RD}}(w, \varphi_0 \rightarrow \varphi_1) = \begin{cases} 1111 & \text{if } V^{\text{RD}}(w, \varphi_0) \preceq V^{\text{RD}}(w, \varphi_1), \\ V^{\text{RD}}(w, \varphi_1) & \text{if } V^{\text{RD}}(w, \varphi_0) \succ V^{\text{RD}}(w, \varphi_1). \end{cases}$

To define the semantics of the diamond and the box operator, we need to first define the semantics of the guards: The match set $\mathcal{R}_i^{\text{RD}}(w, r) \subseteq \mathbb{N}$ for $i \in \{1, 2, 3, 4\}$ contains all positions j of w such that $w[0, j]$ matches r and is defined inductively as follows:

- $\mathcal{R}_i^{\text{RD}}(w, \phi) = \{1\}$ if $w(0) \models \phi$ and $\mathcal{R}_i^{\text{RD}}(\phi, w) = \emptyset$ otherwise, for propositional ϕ .
- $\mathcal{R}_i^{\text{RD}}(w, \varphi?) = \{0\}$ if $V_i^{\text{RD}}(w, \varphi) = 1$ and $\mathcal{R}_i^{\text{RD}}(w, \varphi?) = \emptyset$ otherwise.
- $\mathcal{R}_i^{\text{RD}}(w, r_0 + r_1) = \mathcal{R}_i^{\text{RD}}(w, r_0) \cup \mathcal{R}_i^{\text{RD}}(w, r_1)$.
- $\mathcal{R}_i^{\text{RD}}(w, r_0; r_1) = \{j_0 + j_1 \mid j_0, j_1 \geq 0 \text{ and } j_0 \in \mathcal{R}_i^{\text{RD}}(w, r_0) \text{ and } j_1 \in \mathcal{R}_i^{\text{RD}}(w[j_0, \infty), r_1)\}$, i.e., for j to be in $\mathcal{R}_i^{\text{RD}}(w, r_0; r_1)$, it has to be the sum of natural numbers j_0 and j_1 such that w has a prefix of length j_0 that matches r_0 and $w[j_0, \infty)$ has a prefix of length j_1 that matches r_1 .
- $\mathcal{R}_i^{\text{RD}}(w, r^*) = \{0\} \cup \{j_1 + \dots + j_\ell \mid 0 \leq j_\ell \in \mathcal{R}_i^{\text{RD}}(w[j_1 + \dots + j_{\ell-1}, \infty), r) \text{ for all } \ell' \in \{1, \dots, \ell\}\}$, where we use $j_1 + \dots + j_0 = 0$. Thus, for j to be in $\mathcal{R}_i^{\text{RD}}(w, r^*)$, it has to be expressible as $j = j_1 + \dots + j_\ell$ with non-negative j_ℓ such that the prefix of w of length j_1 matches r , the prefix of

³There is an alternative definition inspired by the semantics of LTL on finite traces: Here, both $\diamond\square\varphi$ and $\square\diamond\varphi$ are equivalent to “ φ holds at the last position”. This suggests interpreting “almost all r -matches are φ -satisfying” and “infinitely many r -matches are φ -satisfying” as “the last r -match is φ -satisfying” in case there are only finitely many r -matches. Arguably, this definition is less intuitive than the one we propose to pursue.

length j_2 of $w[j_1, \infty)$ matches r , and in general, the prefix of length $j_{\ell'}$ of $w[j_1 + \dots + j_{\ell'-1}, \infty)$ matches r , for every $\ell' \in \{1, \dots, \ell\}$.

Due to tests, membership of j in $\mathcal{R}_i^{\text{RD}}(w, r)$ does, in general, not only depend on the prefix $w[0, j)$, but on the complete trace w . Also, the semantics of the propositional atom ϕ differs from the semantics of the test $\phi?$: the former consumes an input letter, while the latter one does not. Thus, rLDL (as LDL) features both kinds of atoms. We define the intuition given above via

- $V^{\text{RD}}(w, \langle \cdot r \cdot \rangle \phi) = b_1 b_2 b_3 b_4$ where $b_i = \max_{j \in \mathcal{R}_i^{\text{RD}}(w, r)} V_i^{\text{RD}}(w[j, \infty), \phi)$ and
- $V^{\text{RD}}(w, [\cdot r \cdot] \phi) = b_1 b_2 b_3 b_4$ with $b_i = \max\{b'_1, \dots, b'_i\}$ for every $i \in \{1, 2, 3, 4\}$, where
 - $b'_1 = \min_{j \in \mathcal{R}_1^{\text{RD}}(w, r)} V_1^{\text{RD}}(w[j, \infty), \phi)$,
 - $b'_2 = \begin{cases} \max_{j' \in \mathbb{N}} \min_{j \in \mathcal{R}_2^{\text{RD}}(w, r) \cap \{j', j'+1, j'+2, \dots\}} V_2^{\text{RD}}(w[j, \infty), \phi) & \text{if } |\mathcal{R}_2^{\text{RD}}(w, r)| = \infty, \\ \min_{j \in \mathcal{R}_2^{\text{RD}}(w, r)} V_2^{\text{RD}}(w[j, \infty), \phi) & \text{if } 0 < |\mathcal{R}_2^{\text{RD}}(w, r)| < \infty, \\ 1 & \text{if } |\mathcal{R}_2^{\text{RD}}(w, r)| = 0, \end{cases}$
 - $b'_3 = \begin{cases} \min_{j' \in \mathbb{N}} \max_{j \in \mathcal{R}_3^{\text{RD}}(w, r) \cap \{j', j'+1, j'+2, \dots\}} V_3^{\text{RD}}(w[j, \infty), \phi) & \text{if } |\mathcal{R}_3^{\text{RD}}(w, r)| = \infty, \\ \max_{j \in \mathcal{R}_3^{\text{RD}}(w, r)} V_3^{\text{RD}}(w[j, \infty), \phi) & \text{if } 0 < |\mathcal{R}_3^{\text{RD}}(w, r)| < \infty, \\ 1 & \text{if } |\mathcal{R}_3^{\text{RD}}(w, r)| = 0, \end{cases}$
 - $b'_4 = \begin{cases} \max_{j \in \mathcal{R}_4^{\text{RD}}(w, r)} V_4^{\text{RD}}(w[j, \infty), \phi) & \text{if } |\mathcal{R}_4^{\text{RD}}(w, r)| > 0, \\ 1 & \text{if } |\mathcal{R}_4^{\text{RD}}(w, r)| = 0. \end{cases}$

To give an intuitive description of the semantics, let us first generalize the notion of r -matches and ϕ -satisfiability. We say that a position j of w is an r -match of degree β if $j \in \mathcal{R}_i^{\text{RD}}(w, r)$ for the unique i with $\beta = 0^{i-1} 1^{5-i}$, which requires all tests in r to be evaluated w.r.t. V_i^{RD} (i.e., to some truth value at least β). Similarly, we say that a position j of w is ϕ -satisfying of degree β if $V^{\text{RD}}(w[j, \infty), \phi) \succeq \beta$, or if, equivalently, $V_i^{\text{RD}}(w[j, \infty), \phi) = 1$ for the unique i with $\beta = 0^{i-1} 1^{5-i}$.

Now, consider the b'_i defining the semantics of the robust box operator: We have $b'_1 = 1$ if all r -matches of degree 1111 are ϕ -satisfying of degree 1111. This is in particular satisfied if there is no such match. Further, if there are infinitely (finitely) many r -matches of degree 0111, then $b'_2 = 1$ if almost all (if all) those matches are ϕ -satisfying of degree 0111. Dually, if there are infinitely (finitely) many r -matches of degree 0011, then $b'_3 = 1$ if infinitely many (at least one) of those matches are (is) ϕ -satisfying of degree 0011. Finally, if there is at least one r -match of degree 0001, then $b'_4 = 1$ if at least one of those matches is ϕ -satisfying of degree 0001. The cases where there is no r -match are irrelevant due to monotonicity, so we hardcode them to 1.

Example 2. Consider the formula $[\cdot r \cdot] q \rightarrow [\cdot \text{tt}; r \cdot] p$ with $r = (\text{tt}; \text{tt})^*$, which expresses that the degree of violation of q at even positions should at most be the degree of violation of p at odd positions. Such a property cannot be expressed in rLTL(\square, \diamond), as even $[\cdot r \cdot] q$ is known to be inexpressible in LTL [4].

First, we state that the semantics is well-defined. This is not obvious due to the case distinctions and the use of the matching sets $\mathcal{R}_i^{\text{RD}}$ for different i .

Lemma 2. We have $V^{\text{RD}}(w, \phi) \in \mathbb{B}_4$ for every trace w and every formula ϕ .

To conclude the definition of the semantics, we give an example witnessing that the maximization over the b'_i in the semantics of the box operator is indeed necessary to obtain monotonicity.

Example 3. Let $\phi = [\cdot r \cdot] \text{ff}$ with $r = ([\cdot \text{tt}^* \cdot] p)?$. Moreover, consider the trace $w = \emptyset\{p\}^\omega$. Then, we have $V^{\text{RD}}(w, [\cdot \text{tt}^* \cdot] p) = 0111$ and consequently $\mathcal{R}_1^{\text{RD}}(w, r) = \emptyset$ and $\mathcal{R}_2^{\text{RD}}(w, r) = \{0\}$. Therefore,

$\min_{j \in \mathcal{R}_1^{\text{RD}}(w,r)} V_1^{\text{RD}}(w[j, \infty), \mathbf{ff}) = \min \emptyset = 1$, but $\min_{j \in \mathcal{R}_2^{\text{RD}}(w,r)} V_2^{\text{RD}}(w[j, \infty), \mathbf{ff}) = \min\{0\} = 0$. Thus, the bits b'_1 and b'_2 inducing $V^{\text{RD}}(w, [\cdot] \mathbf{ff})$ are not monotonic, which explains the need to maximize over the b'_i to obtain the semantics of the robust box operator. The traces $(\emptyset\{p\})^\omega$ and $\{p\}\emptyset^\omega$ witness that monotonicity can also be violated for the pairs b'_2, b'_3 and b'_3, b'_4 .

We prove that rLDL has the exponential compilation property. This allows us to solve the model checking and the synthesis problem using well-known and efficient automata-based algorithms. Furthermore, we are able to show that the complexity of these algorithms is asymptotically the same as the complexity of the algorithms for plain LDL and LTL. In the terminology introduced in the introduction, we present a direct translation, i.e., we translate rLDL directly into automata.

Theorem 3. *Let φ be an rLDL formula, $n = |\varphi|$, and $\beta \in \mathbb{B}_4$. There is a non-deterministic Büchi automaton $\mathfrak{B}_{\varphi, \beta}$ with $2^{\mathcal{O}(n \log n)}$ states recognizing the language $\{w \in (2^P)^\omega \mid V^{\text{RD}}(w, \varphi) \succeq \beta\}$.*

In order to obtain the desired Büchi automata, we follow the approach by Faymonville and Zimmermann [12], who presented a bottom-up translation of parametric LDL, an extension of LDL with prompt temporal operators, into alternating parity automata of linear size. These are then translated into Büchi automata of exponential size. Here, we do not have to deal with prompt operators, but instead with the consequences of the five-valued semantics. Formally, we show that for every rLDL formula φ and every $\beta \in \mathbb{B}_4$, there is an alternating parity automaton $\mathfrak{A}_{\varphi, \beta}$ with $\mathcal{O}(|\varphi|)$ states recognizing the language $\{w \in (2^P)^\omega \mid V^{\text{RD}}(w, \varphi) \succeq \beta\}$.

As alternating parity automata are closed under union, intersection, and complementation, we directly obtain constructions for robust disjunction and conjunction, as these are defined with respect to the order of the truth values. Furthermore, even the robust semantics of implication and negation can be expressed using union, intersection, and complementation of automata. Thus, the only interesting cases are formulas of the form $\langle r \rangle \varphi$ and $[r] \varphi$. Faymonville and Zimmermann showed that one can translate r (which may contain tests) into an equivalent non-deterministic finite automaton with tests, i.e., states may be marked with formulas and the semantics of the automaton takes these into account.

Fix some $\beta \in \mathbb{B}_4$. One can take an automaton \mathfrak{A}_r for r , an alternating automaton $\mathfrak{A}_{\varphi, \beta}$ for φ , and an alternating automaton $\mathfrak{A}_{\theta, \beta}$ for each test θ occurring in r , and combine them into an alternating automaton for $\langle r \rangle \varphi$ with respect to β that works as follows. It simulates \mathfrak{A}_r and spawns a copy of $\mathfrak{A}_{\theta, \beta}$ each time a state marked by the test θ is traversed. Furthermore, the acceptance condition is chosen such that the simulation has to be stopped at some accepting state of \mathfrak{A}_r , which implies that the prefix read so far is an r -match of degree β . Additionally, when stopping the simulation of \mathfrak{A}_r , we additionally spawn a copy of $\mathfrak{A}_{\varphi, \beta}$ to check that this r -match is φ -satisfying of degree β . Altogether, the resulting automaton checks that there is an r -match of degree β that is φ -satisfying of degree β , i.e., it is indeed equivalent to $\langle r \rangle \varphi$ with respect to β .

The construction for a formula of the form $[r] \varphi$ relies on dual arguments, but is more involved due to the case distinctions in the definition of the robust semantics of the box operator. Using standard arguments about infinite languages of finite words, one can show that each of the conditions on $\mathcal{R}_i^{\text{RD}}$ used in the case distinctions can be checked by an automaton obtained from \mathfrak{A}_r . Furthermore, one can construct alternating automata checking that all (almost all, infinitely many, some) r -matches of some degree β are φ -satisfying of degree β by dualizing the construction for diamond formulas sketched above. Here, one heavily relies on alternation and the parity acceptance condition allowing to express finiteness and infiniteness properties. Finally, the case distinctions themselves can be implemented using the closure properties of alternating automata. We present the full construction in the full version [19].

Furthermore, as it is done for the similar construction for PLDL [12], one can show that the automata can indeed be constructed efficiently: the non-deterministic Büchi automaton $\mathfrak{B}_{\varphi, \beta}$ can be constructed

on-the-fly in polynomial space, which is crucial to obtain a model checking algorithm with polynomial space requirements.

4.1 Expressiveness

In this section, we compare the expressiveness of rLDL to that of rLTL(\Box, \Diamond) and LDL. Following Tabuada and Neider [23] we focus on the fragment rLTL(\Box, \Diamond) without next, until and release operators. While the next and until operator could be added easily, the robust semantics of the release operator is incompatible with our definition of the robust box operator. It turns out as expected, that rLDL subsumes rLTL(\Box, \Diamond). Conversely, every rLDL formula φ can be translated into four LDL formulas $\varphi_1, \dots, \varphi_4$ that encode φ in the following sense: We have $V_i^{\text{RD}}(w, \varphi) = V^{\text{D}}(w, \varphi_i)$ for every w .

Theorem 4. *Both rLTL(\Box, \Diamond) and LDL can be embedded into rLDL.*

As LTL is a syntactic fragment of LDL, we immediately obtain that LTL can be embedded into rLDL and, thus, rLDL inherits the lower bounds of LTL.

Our next theorem states that LDL and rLDL are of equal expressiveness. The direction from LDL to rLDL was shown in Theorem 4, hence we focus on the other one. Following Tabuada and Neider [23], we construct for every rLDL formula φ four LDL formulas $\varphi_1, \dots, \varphi_4$ encoding φ as explained above. The construction relies on Theorem 3, unlike the analogous result translating robust LTL directly into LTL [23].

Theorem 5. *LDL and rLDL are equally expressive and the translations are effective.*

In general, translating an rLDL formula into an equivalent LDL formula incurs a triply-exponential blow-up when using the translation described in the proof. On a more positive note, the resulting LDL formula is test-free, i.e., it does not contain tests in its guards. We leave the question of whether there are non-trivial lower bounds on the translation for future work. For the special case of translating rLTL(\Box, \Diamond) into LTL mentioned above, there is only a linear blowup. This translation was presented by Tabuada and Neider [23], but they only claimed an exponential upper bound. However, closer inspection shows that it is linear if the size of formulas is measured in the number of distinct subformulas, not the length of the formula.

4.2 Model Checking and Synthesis

Theorem 5 immediately provides solutions for typical applications of rLDL, such as model checking and synthesis, by reducing the problem from the domain of rLDL to that of LDL. However, the price to pay for this approach is a triply-exponential blow-up in the size of the resulting LDL formula, which is clearly prohibitive for any real-world application. For this reason, we now develop more efficient model checking and synthesis techniques that are based on our direct translation of rLDL into automata (Theorem 3).

We begin with the rLDL model checking problem, which is defined as follows.

Problem 3. *Let φ be an rLDL formula, \mathcal{S} a transition system, and let $\beta \in \mathbb{B}_4$. Does $V^{\text{RD}}(\lambda(\rho), \varphi) \succeq \beta$ hold true for all paths $\rho \in \Pi_{\mathcal{S}}$?*

The exponential compilation property (see Theorem 3) and standard on-the-fly techniques for checking emptiness of exponentially-sized Büchi automata [25] yield a PSPACE upper bound on the complexity of Problem 3. The matching lower bound follows from the subsumption of LDL shown above, as model checking LDL is PSPACE-complete.

Theorem 6. *rLDL model checking is PSPACE-complete.*

Similar to model checking, the translation from rLDL formulas to automata provides us with an effective means to synthesize reactive controllers from rLDL specifications, i.e., for the following problem, where an rLDL game now has the form (G, φ, β) and Player 0 wins a play if and only if its trace w satisfies $V^{\text{RD}}(w, \varphi) \geq \beta$.

Problem 4. *Let \mathcal{G} be an rLDL game and v a vertex. Determine whether Player 0 has a winning strategy for \mathcal{G} from v and compute a finite-state winning strategy if so.*

Theorem 3 provides a straightforward way to solve Problem 4 by reducing it to solving classical parity games (again, see [14, Chapter 2] for an introduction to parity games) while the lower bound follows from the subsumption of LDL.

Theorem 7. *Solving rLDL games is 2EXPTIME-complete.*

5 Towards Robust and Prompt Linear Dynamic Logic

In the previous sections, we studied robust LDL, i.e., we combined robustness and increased expressiveness, and robust Prompt-LTL, i.e., we combined robustness and quantitative operators. The third combination of two aspects, i.e., quantitative operators and increased expressiveness, has been studied before [12]. For all three resulting logics, model checking and synthesis have the same complexity as for plain LTL.

Here, we consider the combination of all three extensions, obtaining the logic rPrompt-LDL, robust Prompt-LDL. The syntax is obtained by adding the prompt diamond operator $\langle \cdot \rangle_{\mathbf{p}} \varphi$ to LDL, by restricting negations to atomic formulas, and by disallowing implications. Here, r is a guard as in rLDL, which may contain tests, i.e., formulas of rPrompt-LDL. Similarly, the semantics is defined as expected, i.e., it is obtained by extending the semantics of rLDL with a bound k for the prompt diamond operator $\langle \cdot \rangle_{\mathbf{p}} \varphi$. Now, its semantics requires the existence of a φ -satisfying r -match within the next k steps. Formal definitions are as expected and presented in the full version [19].

Example 4. *Consider the formula $[\cdot((\neg t)^*; t; (\neg t)^*; t)^*] \langle \cdot \mathbf{tt}^* \cdot \rangle_{\mathbf{p}} s$ and interpret t as the tick of a clock and s as a synchronization. Then, the formula intuitively expresses that every other tick of the clock is followed after a bounded number of steps (not ticks!) by a synchronization.*

More formally, the different degrees of satisfaction of φ express the following possibilities, with respect to a given bound k : (i) every even clock tick is followed by a synchronization within k steps; (ii) almost every even clock tick is followed by a synchronization within k steps; (iii) infinitely many even clock ticks are followed by a synchronization within k steps; (iv) there is at least one even clock tick that is followed by a synchronization within k steps.

This property can neither be expressed in (robust) LDL nor in (robust) Prompt-LTL. Also note that unlike for the similar formula from Example 1, the last two possibilities are not trivial, as we now only consider positions with an even clock tick and not all positions.

In the previous sections, we have seen two approaches to translating robust logics into Büchi automata, the direct and the reduction-based one. Both are extensions of translations originally introduced by Tabuada and Neider for robust LTL. The former one translates a formula of a robust logic directly into an equivalent Büchi automaton while the latter one first translates a formula of a robust logic into an equivalent classical (non-robust) logic, for which a translation into equivalent Büchi automata is already known. For robust LTL, both approaches are applicable [23] and yield Büchi automata of exponential

size. Here, out of necessity, we apply both approaches: for robust LDL, we present a direct translation while we present a reduction-based approach for robust Prompt-LTL. Let us quickly elaborate the reasons for this.

First, consider the reduction-based approach for robust LTL, which translates a formula φ of robust LTL and a truth value $\beta \succ 0000$ into an LTL formula φ_β that captures φ with respect to β . To this end, the formula φ_β implements the intuitive meaning of the robust semantics for the always operator, e.g., we have $(\Box p)_{1111} = \Box p$, $(\Box p)_{0111} = \Diamond \Box p$, $(\Box p)_{0011} = \Box \Diamond p$, and $(\Box p)_{0001} = \Diamond p$.

Trying to apply this approach to the rLDL formula $\varphi = [\cdot r \cdot] p$, say for $\beta = 0111$, would imply using a formula of the form $\langle \cdot r_0 \cdot \rangle [\cdot r_1 \cdot] p$ where r_0 and r_1 are obtained by “splitting” up r . It captures the robust semantics of φ with respect to β on some trace w by expressing that there is an r_0 -match j such that every r_1 -match in $w[j, \infty)$ is p -satisfying with degree β . Thus, r_0 and r_1 have to be picked such that the r_1 -matches in $w[j, \infty)$ as above correspond exactly to the r -matches in w . Further, to obtain a translation of optimal complexity, r_0 and r_1 have to be of polynomial size in $|r|$. It is an open problem whether such a splitting is always possible, in particular in the presence of tests in r and guards with only finitely many r -matches.

Secondly, recall that the direct approach to robust LTL translates a formula φ of rLTL(\Box, \Diamond) into a Büchi automaton that captures φ with respect to all $\beta \in \mathbb{B}_4$ (by considering five initial states, one for each β). Trying to apply this approach to robust Prompt-LTL requires using a more general automaton model that is able to capture the quantitative nature of the prompt diamond operator while still yielding a model checking and a synthesis algorithm with the desired complexity. To the best of our knowledge, no such translation from Prompt-LTL to automata has been presented in the literature, which would be a special case of our construction here.

Thus, according to the state-of-the-art, the direct approach is the only viable one for robust extensions of LDL while the reduction-based approach is the only viable one for robust extensions of Prompt-LTL. This leaves us with no viable approach for rPrompt-LDL.

Nevertheless, in the full version [19], we present a fragment of rPrompt-LDL and a reduction-based translation to Prompt-LDL for it. The fragment is obtained by disallowing tests in guards and requiring them to always have infinitely many matches. For such formulas, one can translate the guard into a deterministic finite automaton (without tests) and then use this automaton to “split” r . However, this involves multiple exponential blowups and hence does not prove that the fragment has the exponential compilation property. Nonetheless, this translation shows that both model checking and synthesis are decidable for this fragment. The decidability of these problems for full rPrompt-LDL is left for further research and seemingly requires new approaches.

6 Conclusion

We addressed the problems of verification and synthesis with robust, expressive, and quantitative linear temporal specifications. Inspired by robust LTL, we have first developed robust extensions of the logics LDL and Prompt-LTL, named rLDL and rPrompt-LTL, respectively. Then, we combined rLDL and rPrompt-LTL into a third logic, named rPrompt-LDL, which has the expressiveness of ω -regular languages and allows robust reasoning about timing bounds.

For rLDL and rPrompt-LTL, we have shown how to solve the model checking and synthesis problem relying on the exponential compilation property. Hence, all these problems are not harder than those for plain LTL. The situation for the combination of all three basic logics, i.e., for rPrompt-LDL, is less encouraging. In the full version [19], we show the problems to be decidable for an important fragment, but

due to a blowup of the formulas during the reduction, we (most likely) do not obtain optimal algorithms. Decidability for the full logic remains open.

In future work, we aim to determine the exact complexity of the model checking and synthesis problem for (full) rPrompt-LDL. One promising approach is to generalize the translation of rLDL into alternating parity automata. However, this requires a suitable quantitative alternating automata model with strong closure properties that can be transformed into equivalent non-deterministic and deterministic automata.

Another promising direction for further research is to study the semantics for the robust box operator proposed in Footnote 3 on Page 9. In particular, it is open whether the translation into alternating automata can be generalized to this setting without a blowup. Also, we leave open whether full robust LTL, i.e., with until and release, can be embedded into rLDL. As is, the robust semantics of the release operator (see [23]) is not compatible with our robust semantics for rLDL. In future work, we plan to study generalizations of full robust LTL.

Another natural question is whether the techniques developed for rLDL can be applied to a robust version of the Property Specification Language [10].

References

- [1] Rajeev Alur, Kousha Etessami, Salvatore La Torre & Doron Peled (2001): *Parametric Temporal Logic for “Model Measuring”*. *ACM Trans. Comput. Log.* 2(3), pp. 388–407, doi:10.1145/377978.377990.
- [2] Rajeev Alur, Salvatore La Torre & P. Madhusudan (2003): *Playing Games with Boxes and Diamonds*. In Roberto M. Amadio & Denis Lugiez, editors: *CONCUR 2003, LNCS 2761*, Springer, pp. 127–141, doi:10.1007/978-3-540-45187-7_8.
- [3] Rajeev Alur & Salvatore La Torre (2004): *Deterministic generators and games for LTL fragments*. *ACM Trans. Comput. Log.* 5(1), pp. 1–25, doi:10.1145/963927.963928.
- [4] Christel Baier & Joost-Pieter Katoen (2008): *Principles of Model Checking*. The MIT Press.
- [5] Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger & Barbara Jobstmann (2010): *Robustness in the Presence of Liveness*. In: *CAV 2010, LNCS 6174*, Springer, pp. 410–424, doi:10.1007/978-3-642-14295-6_36.
- [6] Eric Dallal, Daniel Neider & Paulo Tabuada (2016): *Synthesis of safety controllers robust to unmodeled intermittent disturbances*. In: *CDC 2016*, pp. 7425–7430, doi:10.1109/CDC.2016.7799416.
- [7] Giuseppe De Giacomo & Moshe Y. Vardi (2013): *Linear Temporal Logic and Linear Dynamic Logic on Finite Traces*. In Francesca Rossi, editor: *IJCAI, IJCAI/AAAI*. Available at <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997>.
- [8] Alexandre Donzé & Oded Maler (2010): *Robust Satisfaction of Temporal Logic over Real-Valued Signals*. In Krishnendu Chatterjee & Thomas A. Henzinger, editors: *FORMATS 2010, LNCS 6246*, Springer, pp. 92–106, doi:10.1007/978-3-642-15297-9_9.
- [9] Laurent Doyen, Thomas A. Henzinger, Axel Legay & Dejan Nickovic (2010): *Robustness of Sequential Circuits*. In Luís Gomes, Victor Khomenko & João M. Fernandes, editors: *ACSD 2010, IEEE Computer Society*, pp. 77–84, doi:10.1109/ACSD.2010.26.
- [10] C. Eisner & D. Fisman (2006): *A Practical Introduction to PSL*. Integrated Circuits and Systems, Springer, doi:10.1007/978-0-387-36123-9.
- [11] Georgios E. Fainekos & George J. Pappas (2009): *Robustness of temporal logic specifications for continuous-time signals*. *Theor. Comput. Sci.* 410(42), pp. 4262–4291, doi:10.1016/j.tcs.2009.06.021.
- [12] Peter Faymonville & Martin Zimmermann (2017): *Parametric Linear Dynamic Logic*. *Inf. Comput.* 253, pp. 237–256, doi:10.1016/j.ic.2016.07.009.

- [13] Limor Fix (2008): *Fifteen Years of Formal Property Verification in Intel*. In Orna Grumberg & Helmut Veith, editors: *25 Years of Model Checking - History, Achievements, Perspectives*, LNCS 5000, Springer, pp. 139–144, doi:10.1007/978-3-540-69850-0_8.
- [14] Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors (2002): *Automata, Logics, and Infinite Games: A Guide to Current Research*. LNCS 2500, Springer, doi:10.1007/3-540-36387-4.
- [15] Ron Koymans (1990): *Specifying real-time properties with metric temporal logic*. *Real-Time Systems* 2, pp. 255–299, doi:10.1007/BF01995674.
- [16] Orna Kupferman, Nir Piterman & Moshe Y. Vardi (2009): *From Liveness to Promptness*. *Formal Methods in System Design* 34(2), pp. 83–103, doi:10.1007/s10703-009-0067-z.
- [17] Martin Leucker & César Sánchez (2007): *Regular Linear Temporal Logic*. In Cliff B. Jones, Zhiming Liu & Jim Woodcock, editors: *ICTAC 2007*, LNCS 4711, Springer, pp. 291–305, doi:10.1007/978-3-540-75292-9_20.
- [18] Rupak Majumdar & Indranil Saha (2009): *Symbolic Robustness Analysis*. In Theodore P. Baker, editor: *RTSS 2009*, IEEE Computer Society, pp. 355–363, doi:10.1109/RTSS.2009.17.
- [19] Daniel Neider, Alexander Weinert & Martin Zimmermann (2018): *Robust, Expressive, and Quantitative Linear Temporal Logics: Pick any Two for Free (full version)*. arXiv 1808.09028. Available at <http://arxiv.org/abs/1808.09028>.
- [20] Daniel Neider, Alexander Weinert & Martin Zimmermann (2018): *Synthesizing Optimally Resilient Controllers*. In Dan R. Ghica & Achim Jung, editors: *CSL 2018, LIPIcs* 119, Schloss Dagstuhl - LZI, pp. 34:1–34:17, doi:10.4230/LIPIcs.CSL.2018.34.
- [21] Amir Pnueli (1977): *The temporal logic of programs*. In: *FOCS 1977*, IEEE, pp. 46–57, doi:10.1109/SFCS.1977.32.
- [22] Paulo Tabuada, Sina Yamac Caliskan, Matthias Rungger & Rupak Majumdar (2014): *Towards Robustness for Cyber-Physical Systems*. *IEEE Trans. Automat. Contr.* 59(12), pp. 3151–3163, doi:10.1109/TAC.2014.2351632.
- [23] Paulo Tabuada & Daniel Neider (2016): *Robust Linear Temporal Logic*. In Jean-Marc Talbot & Laurent Regnier, editors: *CSL 2016, LIPIcs* 62, Schloss Dagstuhl - LZI, pp. 10:1–10:21, doi:10.4230/LIPIcs.CSL.2016.10.
- [24] Moshe Y. Vardi (2011): *The rise and fall of LTL*. In Giovanna D’Agostino & Salvatore La Torre, editors: *GandALF 2011, EPTCS* 54.
- [25] Moshe Y. Vardi & Pierre Wolper (1994): *Reasoning About Infinite Computations*. *Inf. Comput.* 115(1), pp. 1–37, doi:10.1006/inco.1994.1092.
- [26] Pierre Wolper (1983): *Temporal Logic Can Be More Expressive*. *Information and Control* 56(1/2), pp. 72–99, doi:10.1016/S0019-9958(83)80051-5.
- [27] Martin Zimmermann (2013): *Optimal Bounds in Parametric LTL Games*. *Theor. Comput. Sci.* 493, pp. 30–45, doi:10.1016/j.tcs.2012.07.039.
- [28] Martin Zimmermann (2018): *Parameterized linear temporal logics meet costs: still not costlier than LTL*. *Acta Inf.* 55(2), pp. 129–152, doi:10.1007/s00236-016-0279-9.