Regular Vacuity

Doron Bustan[2], Alon Flaisher[1], Orna Grumberg[1], Orna Kupferman[3⋆], and Moshe Y. Vardi[2⋆⋆]

[1] Technion Haifa
[2] Rice University
[3] Hebrew University

Abstract. The application of model-checking tools to complex systems involves a nontrivial step of modelling the system by a finite-state model and a translation of the desired properties into a formal specification. While a positive answer of the model checker guarantees that the model satisfies the specification, correctness of the modelling is not checked. Vacuity detection is a successful approach for finding modelling errors that cause the satisfaction of the specification to be trivial. For example, the specification "every request is eventually followed by a grant" is satisfied vacuously in models in which requests are never sent. In general, a specification φ is satisfied vacuously in a model M if φ has a subformula ψ that does not affect the satisfaction of φ in M, where "does not affect" means we can replace ψ by a universally quantified proposition. Previous works focus on temporal logics such as LTL, CTL, and CTL*, and reduce vacuity detection to standard model checking.

A major feature of recent industrial property-specification languages is their regular layer, which includes regular expressions and formulas constructed from regular expressions. Our goal in this work is to extend vacuity detection to such a regular layer of linear-temporal logics. We focus here on RELTL, which is the extension of LTL with a regular layer. We define when a regular expression does not affect the satisfaction of an RELTL formula by means of universally quantified intervals. Thus, the transition to regular vacuity takes us from monadic quantification to dyadic quantification. We argue for the generality of our definition and show that regular-vacuity detection is decidable, but involves an exponential blow-up (in addition to the standard exponential blow-up for LTL model checking). This suggests that, in practice, one may need to work with weaker definitions of vacuity or restrict attention to specifications in which the usage of regular events is constrained. We discuss such weaker definitions, and show that their detection is not harder than standard model checking. We also show that, under certain polarity constraints, even general regular-vacuity detection can be reduced to standard model checking.

1 Introduction

Model-checking tools are successfully used for checking whether systems have desired properties [9]. The application of model-checking tools to complex systems involves

[⋆] Supported in part by BSF grant 9800096, and by a grant from Minerva.

[⋆⋆] Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, EIA-0086264, and ANI-0216467, by BSF grant 9800096, by Texas ATP grant 003604-0058-2003, and by a grant from the Intel Corporation.

a nontrivial step of modelling the system by a finite-state mathematical model, and translation of the desired properties into a formal specification. When the model does not satisfy the specification, model-checking tools accompany a negative answer with a counterexample, which may point to a real error in the system [8]. It is often the case, however, that there is an error in the modelling of the system and/or in the formal specification. Such errors may not be detected when the answer of the model-checking tool is positive: while a positive answer does guarantee that the model satisfies the specification, the answer to the real question, namely, whether the system has the desired properties, may be different.

The realization of this unfortunate situation has led to the development of several *sanity checks* for formal verification. The goal of these checks is to detect errors in the modelling of the system and the properties. Sanity checks in industrial tools are typically simple, often ad hoc, tests, such as checking for enabling conditions that are never enabled [20]. A more systematic approach is based on *vacuity detection*. Intuitively, a specification is satisfied vacuously in a model if it is satisfied in some non-interesting way. For example, the LTL specification $\theta = \textsf{globally}\,(req \rightarrow \textsf{eventually}\,grant)$ ("every request is eventually followed by a grant") is satisfied vacuously in a model with no requests. While vacuity checking cannot ensure that whenever a model satisfies a formula, the model is correct, it does capture inconsistencies between the model and the verified property. Being automatic, vacuity checking avoids hidden false assumptions made by the verifier, and thus it is more likely to capture modelling and specification errors.

Several years of experience in practical formal verification have convinced the verification group in IBM Haifa Research Laboratory that vacuity is a serious problem [5]. To quote from [5]: "Our experience has shown that typically 20% of specifications pass vacuously during the first formal-verification runs of a new hardware design, and that vacuous passes always point to a real problem in either the design or its specification or environment." The first formal treatment of vacuity is described in [5]. Consider a model M satisfying a specification φ. A subformula ψ of φ *does not affect* (the satisfaction of) φ in M if M also satisfies all formulas obtained by modifying ψ. In the example above, the subformula $grant$ does not affect θ in a model with no requests. Now, M satisfies φ vacuously if φ has a subformula that does not affect φ in M. A general method for vacuity detection was presented in [19], who showed that when all the occurrences of ψ in φ are of a *pure polarity* (that is, they are either all under an even number of negations (positive polarity), or all under an odd number of negations (negative polarity)), then ψ does not affect φ iff M satisfies the formula obtained from φ by the single extreme modification of ψ (to $\mathbf{true}$ in case ψ has a negative polarity, and to $\mathbf{false}$ otherwise). This observation reduces vacuity detection to model checking. The usefulness of vacuity analysis is also demonstrated via several case studies in [22]. For more recent work on vacuity checking, see [16, 15].

As shown in [19], the method described there can be used when subformulas of φ are of a *mixed polarity*. In practice, however, one often needs to cope with mixed polarity. For example, the subformula ψ has a mixed polarity in formulas of the (commonly seen) form $\textsf{globally}\,(\psi \rightarrow \theta) \wedge \textsf{eventually}\,\psi$. In fact, industrial-strength property-specification languages such as Sugar [4], ForSpec [3], and the recent standards PSL 1.01

and SVA 3.1a [1] contain operators in which even a single occurrence of ψ may not have a pure polarity (e.g., ψ XOR θ or $\psi \leftrightarrow \theta$).

Once we allow subformulas of a mixed polarity, there is a need to re-examine the definition of when ψ does not affect φ in M. Indeed, it is only in the pure-polarity case that the various modifications of ψ may be restricted to the single extreme modification. Such a re-examination was done in [2], who considered vacuity detection for LTL specifications. While the modifications to ψ in [5] are *syntactic*; i.e., M has to satisfy all formulas $\varphi[\psi \leftarrow \psi']$, namely formulas obtained from φ by substituting ψ by an LTL formula ψ', Armoni et al. argued that a right definition is one in which the modifications to ψ are *semantic*; i.e., M has to satisfy the formula $(\forall x)\varphi[\psi \leftarrow x]$, obtained by substituting ψ by a universally quantified proposition[1]. Gurfinkel et al further extend this definition to CTL* in [15] arguing that it is more robust than other definitions. . It is shown in [2] that, under such a semantic interpretation, vacuity detection of LTL formulas can still be reduced to LTL model checking. A tool used at Intel for vacuity detection is also described in [2].

As mentioned earlier, the work in [2] was motivated by the need to extend vacuity detection to recent industrial property-specification languages, which are significantly richer syntactically and semantically than LTL. A major feature of these languages, which does not exist in LTL, is a *regular layer*, which includes regular expressions and formulas constructed from regular expressions. The regular layer does not only add to the expressive power of the specification language s.t. it can express the whole ω-regular spectrum, but it also seemed to be more intuitive to hardware engineers. For some languages like SVA 3.1a, the only way to express temporal properties is using regular expressions.

As an example of the use of the regular layer, consider the ForSpec formula e seq θ, where e is a regular expression and θ is a formula, asserts that some e sequence is followed by θ, and the ForSpec formula e $\mathsf{triggers}$ θ, asserts that all e sequences are followed by θ. Our goal in this paper is to extend vacuity detection to such a regular layer of linear-temporal logics. Rather than treat the full complexity of industrial languages, we focus here on RELTL, which is the extension of LTL with a regular layer. Thus, we need to define, and then check, the notion of ”does not affect,” not only for subformulas but also for regular expressions. We refer to the latter as *regular vacuity*. As an example, consider the property $\varphi = \mathsf{globally}\ ((req \cdot (\neg ack)^* \cdot ack)\ \mathsf{triggers}\ grant)$, which says that a grant is given exactly one cycle after the cycle in which a request is acknowledged. Note that if $(\neg ack)^* \cdot ack$ does not affect the satisfaction of φ in M (that is, replacing $(\neg ack)^* \cdot ack$ by any other sequence of events does not cause M to violate φ), we can learn that acknowledgments are actually ignored: grants are given, and stay on forever, immediately after a request. Such a behavior is not referred to in the specification, but can be detected by regular vacuity. Note that if the same regular expression appears in the left-hand side of both seq and $\mathsf{triggers}$ formulas or on both sides of a $\mathsf{triggers}$ formula, then this expression has mixed polarity.

[1] A model M satisfies a formula $(\forall x)\varphi(x)$ if φ is satisfied in all computations π that differ from a computation of M only in the label of the proposition x. Note that different occurrences of a state in π may have different x labels.

In order to understand our definition for regular vacuity, consider a formula φ over a set AP of atomic propositions. Let Σ be the set of Boolean functions over AP, and let e be a regular expression over Σ appearing in φ. The regular expression e induces a language – a set of finite words over Σ. For a word $w \in \Sigma^\omega$, the regular expression e induces a set of intervals [3]: these intervals define subwords of w that are members in the language of e. By saying that e does not affect φ in M, we want to capture the fact that we could modify e, replace it with any other regular expression, and still M satisfies φ. As has been the case with propositional vacuity, there is no known algorithmic approach to handle such syntactic modifications in the presence of regular expressions of mixed polarity. Accordingly, as in [2], we follow a semantic approach to modifications of e, where "does not affect" is captured by means of universal quantification. Thus, in RELTL vacuity there are two types of elements we need to universally quantify to check vacuity. First, as in LTL, in order to check whether an RELTL subformula ψ, which is not a regular expression, affects the satisfaction of φ, we quantify universally over a proposition that replaces ψ. In addition, checking whether a regular expression e that appears in φ affects its satisfaction, we need to quantify universally over intervals. Thus, while LTL vacuity involved only *monadic* quantification (over the sets of points in which a subformula may hold), regular vacuity involves *dyadic* quantification (over intervals – sets of pairs of points, in which a regular expression may hold). In Section 3.2, we discuss two weaker alternative definitions: a restriction of the universally quantified intervals to intervals of the same duration as e, and an approximation of the dyadic quantification over intervals by monadic quantification over the Boolean events referred to in the regular expressions. As discussed there, the definition in terms of dyadic quantification is the most general one.

The transition from monadic to dyadic quantification is very challenging. Indeed, while monadic second-order logics are often decidable [7, 23], this is not the case for dyadic second-order logics. For example, while monadic second-order theory of one successor is decidable [7], this is not the case for the dyadic theory [6]. The main result of this work is that regular vacuity is decidable. We show that the automata-theoretic approach to LTL [27] can be extended to handle dyadic universal quantification. Unlike monadic universal quantification, which can be handled with no increase in computational complexity [2], the extension to dyadic quantification involves an exponential blow-up (in addition to the standard exponential blow-up of handling LTL [25]), resulting in an EXPSPACE upper bound, which should be contrasted with a PSPACE upper bound for RELTL model checking. Our NEXPTIME-hardness lower bound, while leaving a small gap with respect to the upper bound, shows that an exponential overhead on top of the complexity of RELTL model checking seems inevitable. The above results suggest that, in practice, one may need to restrict attention to specifications in which regular expressions are of pure polarity. We show that under this assumption, the techniques of [19] can be extended to regular vacuity, which can then be reduced to standard model checking. In addition, for specifications of mixed polarity, the two weaker definitions we suggest for regular vacuity can also be checked in PSPACE – like standard RELTL model checking.

2 RELTL

The linear temporal logic RELTL extends LTL with a regular layer. We consider LTL in a positive normal form, where formulas are constructed from atomic propositions and their negations by means of Boolean ($\wedge$ and $\vee$) and temporal (next, until, and its dual release) connectives. For details, see [21]. Let AP be a finite set of atomic propositions, and let $\mathcal{B}$ denote the set of all Boolean functions $b : 2^{AP} \rightarrow \{\mathbf{false}, \mathbf{true}\}$ (in practice, members of $\mathcal{B}$ are expressed by Boolean expressions over AP). Consider an infinite word $\pi = \pi_0, \pi_1, \ldots \in (2^{AP})^{\omega}$. For integers $j \geq i \geq 0$, and a language $L \subseteq \mathcal{B}^*$, we say that $\pi_i, \ldots, \pi_{j-1}$ *tightly satisfies* L, denoted $\pi, i, j| \equiv L$, if there is a word $b_0 \cdot b_1 \cdots b_{j-1-i} \in L$ such that for all $0 \leq k < j - i$, we have that $b_k(\pi_{i+k}) = \mathbf{true}$. Note that when $i = j$, the interval $\pi_i, \ldots, \pi_{j-1}$ is empty, in which case $\pi, i, j| \equiv L$ iff $\epsilon \in L$.

The logic RELTL contains two regular modalities: $(e$ seq $\varphi)$ and $(e$ triggers $\varphi)$, where e is a regular expression over the alphabet $\mathcal{B}$, and φ is an RELTL formula. Intuitively, $(e$ seq $\varphi)$ asserts that some interval satisfying e is followed by a suffix satisfying φ, whereas $(e$ triggers $\varphi)$ asserts that all intervals satisfying e are followed by a suffix satisfying φ. Note that the seq and triggers connectives are essentially the "diamond" and "box" modalities of PDL [11]. Formally, let π be an infinite word over 2^{AP} then,[2]

- $\pi, i \models (e$ seq $\varphi)$ if for some $j \geq i$, we have $\pi, i, j| \equiv L(e)$ and $\pi, j \models \varphi$.
- $\pi, i \models (e$ triggers $\varphi)$ if for all $j \geq i$ such that $\pi, i, j| \equiv L(e)$, we have $\pi, j \models \varphi$.

In the automata-theoretic approach to model checking, we translate temporal logic formulas to automata [27]. A *nondeterministic generalized Büchi word automaton* (NGBW, for short) is a tuple $A = \langle \Sigma, S, \delta, S_0, \mathcal{F} \rangle$, where Σ is a finite alphabet, S is a finite set of states, $\delta : S \times \Sigma \rightarrow 2^S$ is a transition function, $S_0 \subseteq S$ is a set of initial states, and $\mathcal{F} \subseteq 2^S$ is a set of sets of accepting states. A run ρ of A is an infinite sequence of states in S that starts in a state in S_0 and obeys δ. Let $inf(\rho) \subseteq S$ denote the set of states that are visited infinitely often in ρ. Since the run is infinite and S is finite, it must be that $inf(\rho)$ is not empty. An NGBW A accepts an infinite word π if it has an infinite run ρ over π such that for every $F \in \mathcal{F}$, we have $inf(\rho) \cap F \neq \emptyset$. The full definition of NGBW is given in the full version. We now describe a translation of RELTL formulas to NGBW. The translation can be viewed as a special case of the translation of ETL to NGBW [27] (see also [17]), but we need it as a preparation for our handling of regular vacuity.

Theorem 1. *Given an RELTL formula φ over AP, we can construct an NGBW A_φ over the alphabet 2^{AP} such that $L(A_\varphi) = \{\pi | \pi, 0 \models \varphi\}$ and the size of A_φ is exponential in φ.*

[2] In industrial specification languages such as ForSpec and PSL the semantics is slightly different. There, it is required that the last letter of the interval satisfying $L(e)$ overlaps the first letter of the suffix satisfying ψ. In the full version we describe a linear translation between these two semantics.

Proof: The translation of φ goes via an intermediate formula ψ in the temporal logic ALTL. The syntax of ALTL is identical to the one of RELTL, only that regular expressions over $\mathcal{B}$ are replaced by nondeterministic finite word automata (NFW, for short) over 2^{AP}. The adjustment of the semantics is as expected: let $\pi = \pi_0, \pi_1, \ldots$ be an infinite path over 2^{AP}. For integers i and j with $0 \leq i \leq j$, and an NFW Z with alphabet 2^{AP}, we say that $\pi_i, \ldots, \pi_{j-1}$ *tightly satisfies* $L(Z)$, denoted $\pi, i, j, \models L(Z)$, if $\pi_i, \ldots, \pi_{j-1} \in L(Z)$. Then, the semantics of the seq and $\mathsf{triggers}$ modalities are as in RELTL, with $L(Z)$ replacing $L(e)$.

A regular expression e over the alphabet $\mathcal{B}$ can be linearly translated to an equivalent NFW Z_e with a single initial state [18]. To complete the translation to ALTL, we need to adjust the constructed NFW to the alphabet 2^{AP}. Given the NFW $Z_e = \langle \mathcal{B}, Q, \Delta, q_0, W \rangle$, let $Z'_e = \langle 2^{AP}, Q, \Delta', q_0, W \rangle$, where for every $q, q' \in Q$, and $a \in 2^{AP}$, we have that $q' \in \Delta'(q, a)$ iff there exists $b \in \mathcal{B}$ such that $q' \in \Delta(q, b)$ and $b(a) = \mathbf{true}$. It is easy to see that for all π, i, and j, we have that $\pi, i, j \models L(e)$ iff $\pi, i, j \models L(Z'_e)$. Let ψ be the ALTL formula obtained from φ by replacing every regular expression e in φ by the NFW Z'_e. It follows that for every infinite word π and $i \geq 0$, we have that $\pi, i \models \varphi$ iff $\pi, i \models \psi$.

It is left to show that ALTL formulas can be translated to NGBW. Let ψ be an ALTL formula. For a state $q \in Q$ of an NFW Z, we use Z^q to denote Z with initial state q. Using this notation, ALTL formulas of the form $(Z'_e \ \mathsf{seq}\ \varphi)$ and $(Z'_e \ \mathsf{triggers}\ \varphi)$ now become $(Z'^{q_0}_e \ \mathsf{seq}\ \varphi)$ and $(Z'^{q_0}_e \ \mathsf{triggers}\ \varphi)$. The closure of ψ, denoted $cl(\psi)$, is the set $\{\xi | \xi$ is a subformula of $\psi\} \cup \{(Z^{q'} \ \mathsf{seq}\ \xi) | (Z^q \ \mathsf{seq}\ \xi)$ is a subformula of ψ and q' is a state of $Z^q\} \cup \{(Z^{q'} \ \mathsf{triggers}\ \xi) | (Z^q \ \mathsf{triggers}\ \xi)$ is a subformula of ψ and q' is a state of $Z^q\}$. Let $seq(\psi)$ denote the set of seq formulas in $cl(\psi)$. A subset $C \subseteq cl(\psi)$ is *consistent* if the following hold: (1) if $p \in C$, then $\neg p \notin C$, (2) if $\varphi_1 \wedge \varphi_2 \in C$, then $\varphi_1 \in C$ and $\varphi_2 \in C$, and (3) if $\varphi_1 \vee \varphi_2 \in C$, then $\varphi_1 \in C$ or $\varphi_2 \in C$.

Given ψ, we define the NGBW $A_\psi = \langle 2^{AP}, S, \delta, S_0, \mathcal{F} \rangle$, where $S \subseteq 2^{cl(\psi)} \times 2^{seq(\psi)}$ is the set of all pairs (L_s, P_s) such that L_s is consistent, and $P_s \subseteq L_s \cap seq(\psi)$. Intuitively, when A_ψ reads the point i of π and is in state (L_s, P_s), it guesses that the suffix $\pi_i, \pi_{i+1}, \ldots$ of π satisfies all the formulas in L_s. In addition, as explained below, the set P_s keeps track of the seq formulas in L_s whose eventuality needs to be fulfilled. Accordingly, $S_0 = \{(L_s, \emptyset) \in S : \psi \in L_s\}$.

Before we describe the transition function δ, let us explain how subformulas of the form $(Z^q \ \mathsf{seq}\ \psi)$ and $(Z^q \ \mathsf{triggers}\ \psi)$ are handled. In both subformulas, something should happen after an interval that tightly satisfies Z^q is read. In order to "know" when an interval $\pi_i, \pi_{i+1}, \ldots \pi_{j-1}$ tightly satisfies Z^q, the NGBW A_ψ simulates a run of Z^q on it. The seq operator requires a single interval that tightly satisfies Z^q and is followed by a suffix satisfying ψ. Accordingly, A_ψ simulates a single run, which it chooses nondeterministically. For the $\mathsf{triggers}$ operator, the requirement is for every interval that tightly satisfies Z^q. Accordingly, here A_ψ simulates all possible runs of Z^q. Formally, $\delta : (S \times 2^{AP}) \rightarrow 2^S$ is defined as follows: $(L_t, P_t) \in \delta((L_s, P_s), a)$ iff the following conditions are satisfied:

- For all $p \in AP$, if $p \in L_s$ then $p \in a$, and if $\neg p \in L_s$ then $p \notin a$.
- If $(\mathsf{next}\ \varphi_1) \in L_s$, then $\varphi_1 \in L_t$.
- If $(\varphi_1 \ \mathsf{until}\ \varphi_2) \in L_s$, then either $\varphi_2 \in L_s$, or $\varphi_1 \in L_s$ and $(\varphi_1 \ \mathsf{until}\ \varphi_2) \in L_t$.

- If $(\varphi_1 \textsf{ release } \varphi_2) \in L_s$, then $\varphi_2 \in L_s$ and either $\varphi_1 \in L_s$, or $(\varphi_1 \textsf{ release } \varphi_2) \in L_t$.
  Let $Z = \langle 2^{AP}, Q, \Delta, q_0, W\rangle$ be an NFW.
- If $(Z^q \textsf{ seq } \psi) \in L_s$, then (a) $q \in W$ and $\psi \in L_s$, or (b) $(Z^{q'} \textsf{ seq } \psi) \in L_t$ for some $q' \in \Delta(q, a)$.
- If $(Z^q \textsf{ triggers } \psi) \in L_s$, then (a) if $q \in W$, then $\psi \in L_s$, and (b) $(Z^{q'} \textsf{ triggers } \psi) \in L_t$ for all $q' \in \Delta(q, a)$.
- If $P_s = \emptyset$, then $P_t = L_t \cap seq(\varphi)$. Otherwise, for every $(Z^q \textsf{ seq } \psi) \in P_s$, we have that (a) $q \in W$ and $\psi \in L_s$, or (b) $(Z^{(q')} \textsf{ seq } \psi) \in P_t \cap L_t$ for some $q' \in \Delta(q, a)$.

Finally, the generalized Büchi acceptance condition is used to impose the fulfillment of $\textsf{until}$ and $\textsf{seq}$ eventualities. Thus, $\mathcal{F} = \{\Phi_1, \ldots, \Phi_m, \Phi_{seq}\}$, where for every $(\varphi_i \textsf{ until } \psi_i) \in cl(\varphi)$, we have a set $\Phi_i = \{(L_s, P_s) \in S | \psi_i \in L_s$ or $(\varphi_i \textsf{ until } \psi_i) \notin L_s\}$, and in addition we have the set $\Phi_{seq} = \{(L_s, P_s) \in S | P_s = \emptyset\}$. As in [27], we count on the fact that as long as a seq formula has not reached its eventuality, then some of its derivations appear in the successor state. In addition, whenever P_s is empty, we fill it with new seq formulas that need to be fulfilled. Therefore, the membership of Φ_{seq} in $\mathcal{F}$ guarantees that the eventualities of all seq formulas are fulfilled. The correctness of the construction is proved in the full version. □

The exponential translation of RELTL formulas to NGBW implies a PSPACE model-checking procedure for it [27]. A matching lower bound is immediate from LTL being a fragment of RELTL [25]. Hence the following theorem.

Theorem 2. *The model-checking problem for RELTL is PSPACE-complete.*

3 Regular Vacuity

3.1 A general definition

As discussed in Section 1, we follow the semantic approach to vacuity [2]. According to this approach, a subformula ψ of an RELTL formula φ *does not affect* φ in a model M if M satisfies $(\forall x)\varphi[\psi \leftarrow x]$, where $\varphi[\psi \leftarrow x]$ is the result of replacing in φ all the occurrences of the subformula ψ with the variable x. Thus, ψ is replaced by a universally quantified *propositional variable*. Unlike a subformula ψ, which defines a set of points in a path π (those that satisfy ψ), a regular expression e defines a set of intervals (that is, pairs of points) in π (those that tightly satisfy e). Accordingly, we are going to define "does not affect" for regular expressions by means of universally quantified *interval variables*. For that, we first define QRELTL, which is a technical extension of RELTL; it extends RELTL by universal quantification over a single interval variable.

Recall that the regular expressions of RELTL formulas are defined with respect to the alphabet $\mathcal{B}$ of Boolean expressions over AP. Let y be the interval variable, and let φ be an RELTL formula whose regular expressions are defined with respect to the alphabet $\mathcal{B} \cup \{y\}$. Then $(\forall y)\varphi$ and $(\exists y)\varphi$ are QRELTL formulas. For example, $(\forall y) \textsf{ globally } [(y \textsf{ seq } \psi) \wedge (ab^* \textsf{ triggers } \neg\psi)]$ is a well-formed QRELTL formula, while $\psi \vee [(\exists y)(y \textsf{ seq } \psi)]$ is not.

We now define QRELTL semantics. Let $I = \{(i,j) | \, i,j \in \mathbb{N}, j \geq i\}$ be a set of all (natural) intervals. An *interval set* is a set $\beta \subseteq I$. The interval variable y ranges over interval sets and is associated with β. Thus, $(i,j) \in \beta$ means that y is satisfied over an interval of length $j - i$ that starts at i. For a universally (existentially) quantified formula, satisfaction is checked with respect to every (some) interval set β. We first define when a word $\hat{\pi} = \pi_i \ldots \pi_{j-1}$ over 2^{AP} tightly satisfies, with respect to β, a language L over $\mathcal{B} \cup \{y\}$. Intuitively, it means we can partition $\hat{\pi}$ to sub-intervals that together correspond to a word w in L. Note that since some of the letters in w may be y, the sub-intervals may be of arbitrary (possibly 0) length, corresponding to intervals in β. Formally, we have the following.

Definition 1. *Consider a language $L \subseteq (\mathcal{B} \cup \{y\})^*$, an infinite path π over 2^{AP}, indices i and j with $i \leq j$, and an interval set $\beta \subseteq I$. We say that $\pi_i, \ldots, \pi_{j-1}$ and β* tightly satisfy *L, denoted $\pi, i, j, \beta |\!\!\equiv L$ iff there is $w \in L$ such that either $w = \epsilon$ and $i = j$, or $w = w_0, w_1, \ldots, w_n$ and there is a sequence of integers $i = l_0 \leq l_1 \leq \cdots \leq l_{n+1} = j$ such that for every $0 \leq k \leq n$, the following conditions hold:*

- *If $w_k \in \mathcal{B}$, then $w_k(\pi_{l_k}) = \mathbf{true}$ and $l_{k+1} = l_k + 1$.*
- *If $w_k = y$, then $(l_k, l_{k+1}) \in \beta$.*

For example, if $AP = \{p\}$, $\beta = \{(3,3),(3,4)\}$, and $\pi = \{\{p\}, \emptyset\}^\omega$, then $\pi, 2, 4, \beta |\!\!\equiv \{p \cdot y\}$ since $p(\{p\}) = \mathbf{true}$ and $(3,4) \in \beta$. Also, $\pi, 2, 4, \beta |\!\!\equiv \{p \cdot y \cdot \neg p\}$, since $p(\{p\}) = \mathbf{true}$, $(3,3) \in \beta$, and $\neg p(\emptyset) = \mathbf{true}$. Note that when the required w does not contain y, the definition is independent of β and coincides with tight satisfaction for languages over $\mathcal{B}$.

The semantics of the RELTL subformulas of a QRELTL formula is defined inductively as in RELTL, only with respect to an interval set β. In particular, for the seq and $\mathsf{triggers}$ modalities, we have

- $\pi, i, \beta \models (e \; \mathsf{seq} \; \varphi)$ iff for some $j \geq i$, we have $\pi, i, j, \beta |\!\!\equiv L(e)$ and $\pi, j, \beta \models \varphi$.
- $\pi, i, \beta \models (e \; \mathsf{triggers} \; \varphi)$ iff for all $j \geq i$ s.t. $\pi, i, j, \beta |\!\!\equiv L(e)$ we have $\pi, j, \beta \models \varphi$.

In addition, for QRELTL formulas, we have

- $\pi, i \models (\forall y)\varphi$ iff for every interval set $\beta \subseteq I$, we have $\pi, i, \beta \models \varphi$.
- $\pi, i \models (\exists y)\varphi$ iff there exists an interval set $\beta \subseteq I$, such that $\pi, i, \beta \models \varphi$.

An infinite word π over 2^{AP} satisfies a QRELTL formula φ, denoted $\pi \models \varphi$, if $\pi, 0 \models \varphi$. A model M satisfies φ, denoted $M \models \varphi$, if all traces of M satisfy φ.

Definition 2. *Consider a model M. Let φ be an RELTL formula that is satisfied in M and let e be a regular expression appearing in φ. We say that e* does not affect *φ in M iff $M \models (\forall y)\varphi[e \leftarrow y]$. Otherwise, e affects φ in M. Finally, φ is regularly vacuous in M if there exists a regular expression e that does not affect φ.*

As an example for regular vacuity, consider the property

$$\varphi = \; \mathsf{globally} \; ((req \cdot \mathbf{true} \cdot \mathbf{true}) \; \mathsf{triggers} \; ack)$$

which states that an ack is asserted exactly three cycles after a req. When φ is satisfied in a model M, one might conclude that all requests are acknowledged, and with accurate timing. However, the property is also satisfied in a model M that keeps ack high at all times. Regular vacuity of φ with respect to $(req \cdot \mathbf{true} \cdot \mathbf{true})$ will be detected by showing that the QRELTL formula $(\forall y)\varphi[(req \cdot \mathbf{true} \cdot \mathbf{true}) \leftarrow y]$ is also satisfied in M. This can direct us to the erroneous behavior.

In the previous example we considered regular vacuity with respect to the entire regular expression. Sometimes, a vacuous pass can only be detected by checking regular vacuity with respect to a subexpression. Consider the property

$$\varphi = \textsf{globally } ((req \cdot (\neg ack)^* \cdot ack) \textsf{ triggers } grant)$$

which states that when an ack is asserted sometime after req, then $grant$ is asserted one cycle later. Regular vacuity on the subexpression $((\neg ack)^* \cdot ack)$ can detect that ack is actually ignored, and that $grant$ is asserted immediately after req and remains high. On the other hand, regular vacuity would not be detected on the regular expression $e = (req \cdot (\neg ack)^* \cdot ack)$, as it does affect φ. This is because φ does not hold if e is replaced by an interval $(0, j)$, in which req does not hold in model M.

3.2 Alternative definitions

In this section we describe two alternative definitions for "does not affect" and hence also for regular vacuity. We argue that the definitions are weaker, in the sense that a formula that is satisfied vacuously with respect to Definition 2, is satisfied vacuously also with respect to the alternative definitions, but not vice versa (i.e., it may declare vacuity when the general definition does not.) On the other hand, as we discuss in Section 5, vacuous satisfaction with respect to the alternative definitions is computationally easier to detect.

Regular vacuity modulo duration Consider a regular expression e over $\mathcal{B}$. We say that e is of *duration* d, for $d \geq 0$, if all the words in $L(e)$ are of length d. For example, $a \cdot b \cdot c$ is of duration 3. We say that e is of a *fixed duration* if it is of duration d for some $d \geq 0$. Let $e = a \cdot b \cdot c$ and let $\varphi = e \textsf{ triggers } \psi$. The property φ states that if the computation starts with the Boolean events a, b, and c, then ψ should hold at time 3. Suppose now that in a model M, the formula ψ does not hold at times 0,1, and 2, and holds at later times. In this case, φ holds due to the duration of e, regardless of the Boolean events in e. According to Definition 2, e affects φ (e.g., if $\beta = \{(0, 1)\}$). On the other hand, e does not affect φ if we restrict the interval variable y to intervals of length 3. Thus, e does not affect the truth of φ in M modulo its duration iff φ is still true when e is replaced by an arbitrary interval of the *same* duration (provided e is of a fixed duration). Formally, for a duration d, let $I_d = \{(i, i + d) : i \in \mathbb{N}\}$ be the set of all natural intervals of duration d. The logic *duration-QRELTL* is a variant of QRELTL in which the quantification of y is parametrized by a duration d, and y ranges over intervals of duration d. Thus, $\pi, i \models (\forall_d y)\varphi$ iff for every interval set $\beta \subseteq I_d$, we have $\pi, i, \beta \models \varphi$, and dually for $(\exists_d y)\varphi$.

Definition 3. *Consider a model M. Let φ be an RELTL formula that is satisfied in M and let e be a regular expression of duration d appearing in φ. We say that e* does not affect *φ in M* modulo duration *iff $M \models (\forall_d y)\varphi[e \leftarrow y]$. Finally, φ is regularly vacuous in M modulo duration if there exists a regular expression e of a fixed duration that does not affect φ modulo duration.*

We note that instead of requiring e to have a fixed duration, one can restrict attention to regular expressions of a finite set of durations (in which case e is replaced by intervals of the possible durations); in particular, regular expressions of a bounded duration (in which case e is replaced by intervals shorter than the bound). As we show in Section 5, vacuity detection for all those alternative definitions is similar.

Regular vacuity modulo expression structure Consider again the formula $\varphi = e \text{ triggers } \psi$, for $e = a \cdot b \cdot c$. The formula φ is equivalent to the LTL formula $\varphi' = a \rightarrow X(b \rightarrow X(c \rightarrow X\psi))$. If we check the vacuity of the satisfaction of φ' in a system M, we check, for each of the subformulas a, b, and c whether they affect the satisfaction of φ'. For that, [2] uses universal monadic quantification. In regular vacuity modulo expression structure we do something similar – instead of replacing the whole regular expression with a universally quantified dyadic variable, we replace each of the Boolean functions in $\mathcal{B}$ that appear in the expression by a universally quantified monadic variable (or, equivalently, by a dyadic variable ranging over intervals of duration 1). Thus, in our example, φ passes vacuously in the system M described above, as neither a, b, nor c affect its satisfaction. Formally, we have the following[3].

Definition 4. *Consider a model M. Let φ be an RELTL formula that is satisfied in M and let e be a regular expression appearing in φ. We say that e* does not affect *φ in M* modulo expression structure *iff for all $b \in \mathcal{B}$ that appear in e, we have that $M \models (\forall_1 y)\varphi[b \leftarrow y]$. Finally, φ is regularly vacuous in M modulo expression structure if there exists a regular expression e that does not affect φ modulo expression structure.*

4 Algorithmic Aspects of Detecting Vacuity

In this section we study the complexity of the regular-vacuity problem. As discussed in Section 3, vacuity detection can be reduced to model checking of a QRELTL formula of the form $(\forall y)\varphi$. We describe an automata-based EXPSPACE solution to the latter problem, and conclude that regular vacuity is in EXPSPACE. Recall that we saw in Section 2 that RELTL model checking is in PSPACE. As shown in [2], vacuity detection for LTL is not harder than LTL model checking, and can be solved in PSPACE. In the full version we show that regular vacuity is NEXPTIME-hard. Thus, while the precise complexity of regular vacuity is open, the lower bound indicates that an exponential overhead on top of the complexity of RELTL model checking seems inevitable.

We describe a model-checking algorithm for QRELTL formulas of the form $(\forall y)\varphi$. Recall that in the automata-theoretic approach to LTL model checking, one constructs,

[3] Note that Definition 4 follows the semantic approach of [2]. A syntactic approach, as the one taken in [5, 19], would result in a different definition, where Boolean functions are replaced by different Boolean functions.

given an LTL formula φ, an automaton $A_{\neg\varphi}$ that accepts exactly all paths that do not satisfy φ. Model checking is then reduced to the emptiness of the product of $A_{\neg\varphi}$ with the model M [27]. For a QRELTL formula $(\forall y)\varphi$, we need to construct an automaton $A_{(\exists y)\neg\varphi}$, which accepts all paths that do not satisfy $(\forall y)\varphi$. Since we considered RELTL formulas in a positive normal form, the construction of $\neg\varphi$ has to propagate the negation inward to φ's atomic propositions, using De-Morgan laws and dualities. In particular, $\neg(e \ \mathsf{seq}\ \varphi) = (e \ \mathsf{triggers}\ \neg\varphi)$ and $\neg(e \ \mathsf{triggers}\ \varphi) = (e \ \mathsf{seq}\ \neg\varphi)$. It is easy to see that the length of $\neg\varphi$ in positive normal form is linear in the length of φ.

Theorem 3. *Given an existential QRELTL formula $(\exists y)\varphi$ over AP, we can construct an NGBW A_φ over the alphabet 2^{AP} such that $L(A_\varphi) = \{\pi|\pi, 0 \models (\exists y)\varphi\}$, and the size of A_φ is doubly exponential in φ.*

Proof: The translation of $(\exists y)\varphi$ goes via an intermediate formula $(\exists y)\psi$ in the temporal logic QALTL. The syntax of QALTL is identical to the one of QRELTL, only that regular expressions over $\mathcal{B} \cup \{y\}$ are replaced by NFW over $2^{AP} \cup \{y\}$. The closure of QALTL formulas is defined similarly to the closure of ALTL formulas. The adjustment of the semantics is similar to the adjustment of RELTL to ALTL described in Section 2. In particular, the adjustment of Definition 1 to languages over the alphabet $2^{AP} \cup \{y\}$ replaces the condition "if $w_k \in \mathcal{B}$ then $w_k(\pi_{l_k}) = \mathbf{true}$ and $l_{k+1} = l_k + 1$" there by the condition "if $w_k \in 2^{AP}$, then $w_k = \pi_{l_k}$ and $l_{k+1} = l_k + 1$" here.

Given a QRELTL formula $(\exists y)\varphi$, its equivalent QALTL formula $(\exists y)\psi$ is obtained by replacing every regular expression e in $(\exists y)\varphi$ by Z'_e, where Z'_e is as defined in Section 2. Note that the alphabet of Z'_e is $2^{AP} \cup \{y\}$. It is easy to see that for all π, i, j, and β, we have that $\pi, i, j, \beta \models L(e)$ iff $\pi, i, j, \beta \models L(Z'_e)$. Thus, for every word π and $i \geq 0$, we have that $\pi, i \models (\exists y)\varphi$ iff $\pi, i \models (\exists y)\psi$.

The construction of the NGBW A_φ from $(\exists y)\psi$ is based on the construction presented in Section 2. As there, when A_φ reads π_i and is in state (L_s, P_s), it guesses that the suffix $\pi_i, \pi_{i+1} \ldots$ satisfies all the subformulas in L_s. Since, however, here A_φ needs to simulate NFWs with transitions labelled by the interval variable y, the construction here is more complicated. While a transition labelled by a letter in 2^{AP} corresponds to reading the current letter π_i, a transitions labelled by y corresponds to reading an interval $\pi_i, \ldots, \pi_{j-1}$ in β. Recall that the semantics of QALTL is such that $(\exists y)\psi$ is satisfied in π if there is an interval set $\beta \subseteq I$ for which π, β satisfies ψ. Note that triggers formulas are trivially satisfied for an empty β, whereas seq formulas require β to contain some intervals. Assume that A_φ is in point i of π, it simulates a transition labelled y in an NFW that corresponds to a seq formula in L_s, and it guesses that β contains some interval (i, j). Then, A_φ has to make sure that all the NFWs that correspond to triggers formulas in L_s and that have a transition labelled y, would complete this transition when point j is reached. For that, L_s has to be associated with a set of triggers formulas.

Formally, for a set $L_s \subseteq cl(\psi)$, we define $wait(L_s) = \{(Z^{q'} \ \mathsf{triggers}\ \xi)|(Z^q \ \mathsf{triggers}\ \xi) \in L_s$ and $q' \in \Delta(q, y)\}$. Intuitively, $wait(L_s)$ is the set of triggers formulas that are waiting for an interval in β to end. Once the interval ends, as would be enforced by a seq formula, the members of $wait(L_s)$ should hold. Let $seq(\psi)$ and $trig(\psi)$ be the sets of seq and triggers formulas in $cl(\psi)$, respectively. An *obligation* for ψ is a pair

$o \in seq(\psi) \times 2^{trig(\psi)}$. Let $obl(\psi)$ be the set of all the obligations for ψ. Now, to formalize the intuition above, assume that A_φ is in point i and it simulates a transition labelled y in the NFW Z for some $(Z^q \,\mathsf{seq}\, \xi) \in L_s$. Then, A_φ creates the obligation $o = ((Z^q \,\mathsf{seq}\, \xi), wait(L_s))$ and propagates it until the end of the interval.

The NGBW $A_\varphi = \langle 2^{AP}, S, \delta, S_0, \mathcal{F}\rangle$, where the set of states S is the set of all pairs (L_s, P_s) such that L_s is a consistent set of formulas and obligations, and $P_s \subseteq L_s \cap (seq(\psi) \cup obl(\psi))$. Note that the size of A_φ is doubly exponential in φ. The set of initial states is $S_0 = \{(L_s, P_s) | \psi \in L_s, P_s = \emptyset\}$. The acceptance condition is used to impose the fulfillment of until and seq eventualities, and is similar to the construction in Section 2; thus $\mathcal{F} = \{\Phi_1, \ldots, \Phi_m, \Phi_{seq}\}$, where $\Phi_i = \{s \in S | (\varphi_i \,\mathsf{until}\, \xi_i), \xi_i \in L_s$ or $(\varphi_i \,\mathsf{until}\, \xi_i) \notin L_s\}$, and $\Phi_{seq} = \{s \in S | P_s = \emptyset\}$. We define the transition relation δ as the set of all triples $((L_s, P_s), a, (L_t, P_t))$ that satisfy the following conditions:

1. For all $p \in AP$, if $p \in L_s$ then $p \in a$.
2. For all $p \in AP$, if $\neg p \in L_s$ then $p \notin a$.
3. If $(\,\mathsf{next}\, \varphi_1) \in L_s$, then $\varphi_1 \in L_t$.
4. If $(\varphi_1 \,\mathsf{until}\, \varphi_2) \in L_s$, then either $\varphi_2 \in L_s$, or $\varphi_1 \in L_s$ and $(\varphi_1 \,\mathsf{until}\, \varphi_2) \in L_t$.
5. If $(\varphi_1 \,\mathsf{release}\, \varphi_2) \in L_s$, then $\varphi_2 \in L_s$ and either $\varphi_1 \in L_s$, or $(\varphi_1 \,\mathsf{release}\, \varphi_2) \in L_t$.
6. If $(Z^q \,\mathsf{seq}\, \xi) \in L_s$, then at least one of the following holds:
   (a) $q \in W$ and $\xi \in L_s$.
   (b) $(Z^{q'} \,\mathsf{seq}\, \xi) \in L_t$ for some $q' \in \Delta(q, a)$.
   (c) $\Delta(q, y) \neq \emptyset$ and $o = ((Z^q \,\mathsf{seq}\, \xi), wait(L_s)) \in L_s$. In this case we say that there is a y-transition from $(Z^q \,\mathsf{seq}\, \xi)$ to o in L_s.

   If conditions a or b hold, we say that $(Z^q \,\mathsf{seq}\, \xi)$ is *strong* in L_s w.r.t. $((L_s, P_s), a, (L_t, P_t))$.
7. If $(Z^q \,\mathsf{triggers}\, \xi) \in L_s$, then the following holds:
   (a) If $q \in W$, then $\xi \in L_s$.
   (b) $(Z^{q'} \,\mathsf{triggers}\, \xi) \in L_t$ for all $q' \in \Delta(q, a)$.
8. For every $(Z^q \,\mathsf{seq}\, \xi) \in P_s$, at least one of the following holds:
   (a) $q \in W$ and $\xi \in L_s$.
   (b) $(Z^{q'} \,\mathsf{seq}\, \xi) \in P_t \cap L_t$ for some $q' \in \Delta(q, a)$.
   (c) $\Delta(q, y) \neq \emptyset$ and $o = ((Z^q \,\mathsf{seq}\, \xi), wait(L_s)) \in P_s$. In this case we say that there is a y-transition from $(Z^q \,\mathsf{seq}\, \xi)$ to o in P_s.

   If conditions a or b hold, we say that $(Z^q \,\mathsf{seq}\, \xi)$ is strong in P_s w.r.t. $((L_s, P_s), a, (L_t, P_t))$.
9. If $o = ((Z^q \,\mathsf{seq}\, \xi), \Upsilon) \in L_s$ then at least one of the following holds:
   (a) For some $q' \in \Delta(q, y)$, we have that $(Z^{q'} \,\mathsf{seq}\, \xi) \in L_s$ and $\Upsilon \subseteq L_s$. In this case we say that there is a y-transition from o to $(Z^{q'} \,\mathsf{seq}\, \xi)$ in L_s.
   (b) $o \in L_t$.

   If condition b holds, we say that o is strong in L_s w.r.t. $((L_s, P_s), a, (L_t, P_t))$.
10. If $o = ((Z^q \,\mathsf{seq}\, \xi), \Upsilon) \in P_s$ then at least one of the following holds:
    (a) For some $q' \in \Delta(q, y)$, we have that $(Z^{q'} \,\mathsf{seq}\, \xi) \in P_s$ and $\Upsilon \subseteq L_s$. In this case we say that there is a y-transition from o to $(Z^{q'} \,\mathsf{seq}\, \xi)$ in P_s.
    (b) $o \in P_t$.

    If condition b holds, we say that o is strong in P_s w.r.t. $((L_s, P_s), a, (L_t, P_t))$.
11. If $P_s = \emptyset$, then $P_t = L_t \cap (seq(\varphi) \cup obl(\varphi))$.

12. If $wait(L_s) \subseteq L_s$, then for every element in $L_s \cap (seq(\varphi) \cup obl(\varphi))$ there exists a path (possibly of length 0) of y transitions to a strong element w.r.t. $((L_s, P_s), a, (L_t, P_t))$. Note that the y-transitions are local in L_s and defined in rules $6, 8, 9, 10$.
13. If $wait(L_s) \subseteq L_s$, then for every element in $P_s \cap (seq(\varphi) \cup obl(\varphi))$ there exists a path (possibly of length 0) of y transitions to a strong element w.r.t. $((L_s, P_s), a, (L_t, P_t))$.

We now explain the role of conditions 12 and 13 of δ. As explained above, for every formula $(Z^q \ \mathsf{seq}\ \xi)$ that should hold at point i, the NGBW A_φ simulates a run of Z^q that should eventually accept an interval of π. Since Z^q has transitions labelled by y, it is possible for Z^q to loop forever in (L_i, P_i) (when $(i, i) \in \beta$). Conditions 12 and 13 force the run of Z^q to eventually reach an accepting state, and prevent such an infinite loop. The correctness of the construction is proved in the full version. □

In the automata-theoretic approach to linear model checking, we translate a formula ψ to an automaton that accepts exactly all the computations that satisfy ψ. While traditional translations use nondeterministic automata (cf., [13]), recent translations go through alternating automata (cf., [12, 26]). Then, the state space of the automaton consists of subformulas of ψ, the construction is considerably simpler, and the intermediate automata are exponentially more succinct. In particular, the translation of RELTL formulas to NGBW described in Theorem 1 can be replaced by a simpler translation, to alternating automata. For vacuity detection, however, we have to use nondeterministic automata. To see why, note that reasoning about the QRELTL formula $(\exists y)\varphi$ involves a guess as to where intervals associated with y end. Therefore, a translation of the formula to an alternating automaton results in an automaton in which the different copies need to coordinate in order to synchronize at the position when y ends. Such a synchronization is impossible for alternating automata.

Given a model M and the NGBW A_φ for $(\exists y)\varphi$, the emptiness of their intersection can be tested in time polynomial or in space polylogarithmic in the sizes of M and A_φ (note that M and A_φ can be generated on the fly) [27]. A path in the intersection of M and A_φ is a witness that e affects φ. It follows that the problem of deciding whether a regular expression e affects φ in M can be solved in EXPSPACE. Since the number of regular expressions appearing in φ is linear in the length of φ, we can conclude with the following upper bound to the regular-vacuity problem. As detailed in the full version, the lower bound follows from a reduction of the exponential bounded-tiling problem to regular vacuity.

Theorem 4. *The regular-vacuity problem for RELTL can be solved in EXPSPACE and is NEXPTIME-hard.*

In Section 5, we analyze the complexity of regular vacuity more carefully and show that the computational bottle-neck is the length of regular expressions appearing in triggers formulas in φ. We also describe a fragment of RELTL for which regular vacuity can be solved in PSPACE.

5 Regular Vacuity in Practice

The results in Section 4 suggest that, in practice, because of the computational complexity of general vacuity checking, one may need to work with weaker definitions of vacuity or restrict attention to specifications in which the usage of regular expressions is constrained. In this section we show that under certain polarity constraints, regular vacuity can be reduced to standard model checking. In addition we show that even without polarity constraints, detection of the weaker definitions of vacuity, presented in Section 3.2, is also not harder than standard model checking.

Specifications of pure polarity Examining industrial examples shows that in many cases the number of trigger formulas that share a regular expression with a seq formula is quite small. One of the few examples that use both describes a clock tick pattern and is expressed by the formula $tick\_pattern = (e \textsf{ seq } \mathbf{true}) \wedge \textsf{ globally } (e \textsf{ triggers } (e \textsf{ seq } \mathbf{true}))$, where e defines the clock ratio, e.g. $e = clock\_low \cdot clock\_low \cdot clock\_high \cdot clock\_high$.

As shown in the previous section, the general case of regular vacuity adds an exponential blow-up on top of the complexity of RELTL model checking. A careful analysis of the state space of A_φ shows that with every set L_s of formulas, we associate obligations that are relevant to L_s. Thus, if L_s contains no seq formula with an NFW that reads a transition labelled y, then its obligation is empty. Otherwise, $wait(L_s)$ contains only trigger formulas that appear in L_s and whose NFWs read a transition labelled y. In particular, in the special case where seq and trigger subformulas do not share regular expressions, we have $|obl(\varphi)| = 0$. For this type of specifications, where all regular expressions have a *pure polarity*, regular vacuity is much easier. Rather than analyzing the structure of A_φ in this special case, we describe here a direct algorithm for its regular-vacuity problem.

We first define *pure polarity* for regular expression. As formulas in RELTL are in positive normal form, polarity of a regular expression e is not defined by number of negations, but rather by the operator applied to e. Formally, an occurrence of a regular expression e is of *positive polarity* in φ if it is on the left hand side of a seq modality, and of *negative polarity* if it is on the left hand side of a triggers modality. The polarity of a regular expression is defined by the polarity of its occurrences as follows. A regular expression e is of *positive polarity* if all occurrences of e in φ are of positive polarity, of *negative polarity* if all occurrences of e in φ are of negative polarity, of *pure polarity* if it is either of positive or negative polarity, and of *mixed polarity* if some occurrences of e in φ are of positive polarity and some are of negative polarity.

Definition 5. *Given a formula φ and a regular expression of pure polarity e, we denote by $\varphi[e \leftarrow \bot]$ the formula obtained from φ by replacing e by $\mathbf{true}^*$, if e is of negative polarity, and by $\mathbf{false}$ if e is of positive polarity.*

We now show that for e with pure polarity in φ, checking whether e effects φ, can be reduced to RELTL model checking:

Theorem 5. *Consider a model M, RELTL formula φ, and regular expression e of pure polarity. Then, $M \models (\forall y)\varphi[e \leftarrow y]$ iff $M \models \varphi[e \leftarrow \bot]$.*

Since the model-checking problem for RELTL can be solved in PSPACE, it follows that the regular-vacuity problem for the fragment of RELTL in which all regular expressions are of pure polarity is PSPACE-complete.

Weaker definitions of regular vacuity In Section 3.2, we suggested two alternative definitions for regular vacuity. We now show that vacuity detection according to these definitions is in PSPACE – not harder than RELTL model checking.

We first show that the dyadic quantification in duration-QRELTL can be reduced to a monadic one. Intuitively, since the quantification in duration-QRELTL ranges over intervals of a fixed and known duration, it can be replaced by a quantification over the points where intervals start. Formally, we have the following:

Lemma 1. *Consider a system M, an RELTL formula φ, a regular expression e appearing in φ, and $d > 0$. Then, $M \models (\forall_d y)\varphi[e \leftarrow y]$ iff $M \models (\forall x)\varphi[e \leftarrow (x \cdot \mathbf{true}^{d-1})]$, where x is a monadic variable.*

Universal quantification of monadic variables does not make model checking harder: checking whether $M \models (\forall x)\varphi$ can be reduced to checking whether there is a computation of M that satisfies $(\exists x)\neg\varphi$. Thus, as detailed in [2], when we construct the intersection of M with the NGBW for $\neg\varphi$, the values for x can be guessed, and the algorithm coincides with the one for RELTL model checking. Since detection of vacuity modulo duration and modulo expression structure are both reduced to duration-QRELTL model checking, Theorem 2 implies the following.

Theorem 6. *The problem of detecting regular vacuity modulo duration or modulo expression structure is PSPACE-complete.*

We note that when the formula is of a pure polarity, no quantification is needed, and e may be replaced, in the case of vacuity modulo duration, by $\mathbf{false}$ or $\mathbf{true}^d$ according its polarity. Likewise, in the case of vacuity modulo expression structure, the Boolean formulas in e may be replaced by $\mathbf{false}$ or $\mathbf{true}$.

6 Concluding Remarks

We extended in this work vacuity detection to a regular layer of linear-temporal logics. We focused here on RELTL, which is the extension of LTL with a regular layer. We defined the notion of "does not affect," for regular expressions in terms of universal dyadic quantification. We showed that regular vacuity is decidable, but involves an exponential blow-up (in addition to the standard exponential blow-up for LTL model checking). We showed that under certain polarity constraints on regular expressions, regular vacuity can be reduced to standard model checking. Our decidability result for dyadic second-order quantification is of independent interest. It suggests that the boundary between decidability and undecidability can be charted at a finer detail than the current monadic/dyadic boundary. A related phenomenon was observed in the context of descriptive complexity theory, see [10, 14].

We suggested two alternative definitions for regular vacuity and showed that with respect to these definitions, even for formulas that do not satisfy the polarity constraints, vacuity detection can be reduced to standard model checking, which makes them of practical interest. The two definitions are weaker than our general definition, in the sense that a vacuous pass according to them may not be considered vacuous according to the general definition. It may seem that working with a more sensitive definition would be an advantage, but experience with vacuity detection in industrial settings shows that flooding users with too many reports of vacuous passes may be counterproductive. Thus, it is difficult to make at this point definitive statements about the overall usability of the weaker definitions, as more industrial experience with them is needed.

References

1. Accellera - www.accellera.org.
2. R. Armon, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. Enhanced vacuity detection for linear temporal logic. In *Proc 15th CAV*, LNCS 2725, 2003.
3. R. Armoni et al. The ForSpec temporal logic: A new temporal property-specification logic. In *8th TACAS*, LNCS 2280, pages 296–211, 2002
4. I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic sugar. In *Proc. 13th CAV*, LNCS 2102, pages 363–367, 2001.
5. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. *FMSD* 18(2):141–162, 2001.
6. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. 1996.
7. J. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 1–12, Stanford, 1962.
8. E. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32nd DAC*, pages 427–432, 1995.
9. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
10. T. Eiter, G. Gottlob, and T. Schwentick. Second-order logic over strings: Regular and non-regular fragments. *Developments in Language Theory*, pages 37–56, 2001.
11. M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *JCSS*, 18:194–211, 1979.
12. P. Gastin and D. Oddoux. Fast LTL to büchi automata translation. In *Proc. 13th CAV*, LNCS 2102, pages 53–65, 2001.
13. R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, August 1995.
14. G. Gottlob, P. G. Kolaitis, and T. Schwentick. Existential second-order logic over graphs: Charting the tractability frontier. *JACM*, 51(2):312–362, 2000.
15. A. Gurfinkel and M. Chechik. Extending extended vacuity. In *5th FMCAD*, LNCS 2212, pages 306–321, 2004.
16. A. Gurfinkel and M. Chechik. How vacuous is vacuous. In *Proc. 10th TACAS*, LNCS 2988, pages 451–466, 2004.
17. J. Henriksen and P. Thiagarajan. Dynamic linear time temporal logic. *Annals of Pure and Applied Logic*, 96(1–3):187–207, 1999.
18. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
19. O. Kupferman and M. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2):224–233, February 2003.
20. R. Kurshan. *FormalCheck User's Manual*. Cadence Design, Inc., 1998.
21. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, Berlin, January 1992.
22. M. Purandare and F. Somenzi. Vacuum cleaning CTL formulae. In *Proc. 14th CAV*, LNCS 2404, pages 485–499, 2002.
23. M. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

24. M. Savelsberg and P. van emde Boas. Bounded tiling, an alternative to satisfiability. In *2nd Frege conference*, pages 354–363. Akademya Verlag, 1984.
25. A. Sistla and E. Clarke. The complexity of propositional linear temporal logic. *Journal ACM*, 32:733–749, 1985.
26. M. Vardi. Nontraditional applications of automata theory. In *Proc. STACS*, LNCS 789, pages 575–597, 1994.
27. M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.