Enhanced Vacuity Detection in Linear Temporal Logic

Alon Flaisher

Contents

List of Figures

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. Previous works have focused on detecting vacuity with respect to subformula occurrences in logics such as LTL, CTL, and CTL*. In this work we investigate vacuity detection with respect to subformulas with multiple occurrences in industrial strength specification languages.

The generality of our framework requires us to re-examine the basic intuition underlying the concept of vacuity, which until now has been defined as sensitivity with respect to syntactic perturbation. We study sensitivity with respect to semantic perturbation, which we model by monadic universal quantification. We show that this yields a hierarchy of vacuity notions. We argue that the right notion is that of vacuity defined with respect to traces and provide an algorithm for vacuity detection. As recent industrial property-specification languages feature a regular layer, which includes regular expressions and formulas constructed from regular expressions, we 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 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). Finally, we discuss pragmatic aspects of vacuity checking.

Acknowledgements

Studying for a master degree in computer science while working full time at Intel and expanding the happy family is challenging. I would like to thank the following people, who helped me so much, and made this happen.

My supervisor Prof. Orna Grumberg for being so understanding, open minded, putting the right pressure on me and making this work enjoyable. Thanks for many hours of work at the Technion and Mandarin.

My unofficial supervisor Prof. Moshe Vardi for his guidance and vision, and for sharing his endless knowledge.

Dr. Doron Bustan and Nir Piterman for helping out, taking work to completion and teaching me so much.

The folks at Intel's Haifa development center - Roy Armoni, Limor Fix, Andreas Tiemeyer, Ranan Fraer and others - for introducing me to the world of formal verification and encouraging my studies.

Finally, to my family, for doing "eights in the air" and giving up things just so I can study. My wife Michal, her parents Bella and Kobi and my parents Shosh and Gadi.

Notation and Abbreviations

UQLTL — Universally quantified linear temoral logic
QRELTL — Universally quantified regular expressions linear temoral logic
QALTL — Same as QRELTL, with regular expressions replaced by NFW
x — A propositional variable (associated with σ or α)
y — An interval variable (associated with β)
$\mathcal{T}(M)$ — Set of computations of M
σ — A structure assignment ($\sigma : X \rightarrow 2^S$)
α — A trace assignment ($\alpha : X \rightarrow 2^{\mathbf{N}}$)
β — An interval set ($\beta \subseteq \{(i,j) | \, i,j \in \mathbb{N}, j \geq i\}$)
AP — Set of atomic propositions
φ — A formula (e.g. LTL, RELTL, UQLTL)
ψ — Usually a subformula of φ
M — Model of a system (Kripke structure)
π — A computation (trace) of M
$M, \pi \models \varphi$ — Trace π of M satisfies φ
$M, \pi \not\models \varphi$ — Trace π of M refutes φ
$M, \pi \models_s \varphi$ — Trace π of M structure satisfies φ
$M, \pi \models_t \varphi$ — Trace π of M trace satisfies φ
$M, \pi, i, j \models\!\equiv e$ — Trace π of M tightly satisfies e between i and j
e — A regular expression
Z^q — An NFW Z with an initial state q
NFW — Nondeterministic Finite Word Automaton
$L(Z)$ — Language of the NFW Z
$\mathcal{A}_\varphi$ — An NGBW for φ
NGBW — Nondeterministic Generalized Büchi Word automaton
$\varphi\,[\psi \leftarrow \perp]$ — The formula obtained from φ by replacing ψ by $\mathbf{true}$ or $\mathbf{false}$

Chapter 1

Introduction

Temporal logics, which are modal logics geared towards the description of the temporal ordering of events, have been adopted as a powerful tool for specifying and verifying concurrent systems [Pnu77]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal-logic properties of *finite-state* systems [CE81, CES86, LP85, QS81, VW86]. This derives its significance both from the fact that many synchronization and communication protocols can be modeled as finite-state systems, as well as from the great ease of use of fully algorithmic methods. In temporal-logic *model checking*, we verify the correctness of a finite-state system with respect to a desired behavior by checking whether a labeled state-transition graph that models the system satisfies a temporal logic formula that specifies this behavior (for an in-depth survey, see [CGP99]).

Beyond being fully-automatic, an additional attraction of model-checking tools is their ability to accompany a negative answer to the correctness query with a counterexample to the satisfaction of the specification in the system. Thus, together with a negative answer, the model checker returns some erroneous execution of the system. These counterexamples are very important and can be essential in detecting subtle errors in complex designs [CGMZ95]. On the other hand, when the answer to the correctness query is positive, most model-checking tools provide no witness for the satisfaction of the specification in the system. Since a positive answer means that the system is correct with respect to the specification, this may, a priori, seem like a reasonable policy. In the last few years, however, industrial practitioners have become increasingly aware of the importance of checking the validity of a positive result of model checking. The main justification for suspecting the validity of a positive result is the possibility of errors in the modeling of the system or of the desired behavior, i.e., the specification.

Early work on "suspecting a positive answer" concerns the fact that temporal logic formulas can suffer from *antecedent failure* [BB94]. For example, in verifying a system with respect to the CTL specification $\varphi = AG(req \rightarrow AF grant)$ ("every request is eventually followed by a grant"), one should distinguish between *vacuous satisfaction* of φ, which is immediate in systems in which requests are never sent, and non-vacuous satisfaction, in systems where requests are sometimes sent. Evidently, vacuous satisfaction suggests some unexpected properties of the system, namely the absence of behaviors in which the antecedent of φ is satisfied.

Several years of practical experience in formal verification have convinced the verification group at the IBM Haifa Research Laboratory that vacuity is a serious problem [BBER97]. To quote from [BBER97]: "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 usefulness of vacuity analysis is also demonstrated via several case studies in [PS02]. Often, it is possible to detect vacuity easily by checking the system with respect to hand-written formulas that ensure the satisfaction of the preconditions in the specification [BB94, PP95].

These observations led Beer et al. to develop a method for automatic testing of vacuity [BBER97]. Vacuity is defined as follows: a formula φ is satisfied in a system M vacuously if it is satisfied in M, but some subformula ψ of φ does not *affect* φ in M, which means that M also satisfies $\varphi\,[\psi \leftarrow \psi']$ for all formulas ψ' ($\varphi\,[\psi \leftarrow \psi']$ denotes the result of substituting ψ' for ψ in φ). Beer et al. proposed testing vacuity by means of *witness formulas*. In the example above, it is not hard to see that a system satisfies φ non-vacuously iff it also satisfies $EF req$. In general, however, the generation of witness formulas is not trivial, especially when we are interested in other types of vacuity passes, which are more complex than antecedent failure. While [BBER97] nicely set the basis for a methodology for detecting vacuity in temporal-logic specifications, the particular method described in [BBER97] is quite limited (see also [BBER01]). The type of vacuity passes handled is indeed richer than antecedent failure, yet it is still very restricted. Beer et al. consider the subset w-ACTL of the universal fragment ACTL of CTL. The logic w-ACTL consists of all ACTL formulas in which all the (Boolean or temporal) binary operators have at least one operand that is a propositional formula. Many natural specifications cannot be expressed in w-ACTL.

A general method for detection of vacuity for specifications in CTL* (and hence also LTL, which was not handled by [BBER97]) was presented in [KV99, KV03]. The key idea there is a general method for generating witness formulas. It is shown in [KV03] that instead of replacing a subformula ψ by all subformulas ψ', it suffices to replace it by either $\mathbf{true}$ or $\mathbf{false}$ depending on whether ψ occurs in φ with negative polarity (i.e., under an odd number of negations) or positive polarity (i.e., under an even number of negations). Thus, vacuity checking amounts to model checking witness formulas with respect to all (or some) of the subformulas of the specification φ. It is important to note that the method in [KV03] is for vacuity with respect to subformula occurrences. The key feature of occurrences is that a subformula occurrence has a *pure* polarity (exclusively negative or positive). In fact, it is shown in [KV03] that the method is not applicable to subformulas with mixed polarity (both negative and positive occurrences).

Recent experience with industrial-strength property-specification languages such as ForSpec [AFF$^+$02] suggests that the restriction to subformula occurrences of pure polarity is not negligible. ForSpec, which is a linear-time language, is significantly richer syntactically (and semantically) than LTL. In particular, it has a rich set of arithmetical and Boolean operators. As a result, even subformula occurrences may not have pure polarity, e.g., in the formulas $p \oplus q$ ($\oplus$ denotes exclusive or). While we can rewrite $p \oplus q$ as $(p \wedge \neg q) \vee (\neg p \wedge q)$, it forces the user to think of every subformula occurrence of mixed polarity as two distinct occurrences, which is rather unnatural. Also, a subformula may occur in the specification multiple times, so it need not have a pure polarity even if each occurrence has a pure polarity. For example, if the LTL formula $G(p \rightarrow p)$ holds in a system M then we would expect it to hold vacuously with respect to the subformula p (which has a mixed polarity), though not necessarily with respect to either occurrence of p, because both formulas $G(\mathbf{true} \rightarrow p)$ and $G(p \rightarrow \mathbf{false})$ may fail in M. (Surely, the fact that $G(\mathbf{true} \rightarrow \mathbf{false})$ fails in M should not entail that $G(p \rightarrow p)$ holds in M non-vacuously.) Our first goal in this thesis is to remove the restriction in [KV03] to subformula occurrences of pure polarity, and consider vacuity with respect to subformulas.

The generality of our framework requires us to re-examine the basic intuition underlying the concept of vacuity. As defined, a formula φ is satisfied in a system M vacuously if it is satisfied in M but some subformula ψ of φ does not *affect* φ in M. It is less clear, however, what does "does not affect" means. Intuitively, it means that we can "perturb" ψ without affecting the truth of φ in M. Both [BBER97] and [KV03] consider only syntactic perturbation, but no justification is offered for this decision. We argue that another notion to consider is that of semantic perturbation, where the *truth value* of ψ in M is perturbed arbitrarily.

The first part of this thesis is an examination in depth of this approach. We model arbitrary semantic perturbation by a universal quantifier, which in turn is open to two interpretations (cf. [Kup95]). It turns out that we get two notions of "does not affect" (and therefore also of vacuity), depending on whether universal quantification is interpreted with respect to the system M or with respect to its computation tree. We refer to these two semantics as "structure semantics" and "trace semantics". Surprisingly, the original, syntactic, notion of perturbation falls between the two semantic notions.

We argue then that trace semantics is the preferred one for vacuity checking. Structure semantics is simply too weak, yielding vacuity too easily. Formula semantics is more discriminating, but it is not robust, depending too much on the syntax of the language. In addition, these two semantics yield notions of vacuity that are computationally intractable. In contrast, trace semantics is not only intuitive and robust, but it can be checked easily by a model checker.

In addition to a rich set of arithmetical and Boolean operators, industrial property-specification languages offer a *regular layer*, which includes regular expressions and formulas constructed from regular expressions. For example, 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 triggers θ, asserts that all e sequences are followed by θ. Our second goal in this thesis 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 $\varphi =$ globally $((req \cdot (\neg ack)^* \cdot ack)$ 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.

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 [AFF$^+$02]: 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 φ would be satisfied in M. Once again, we argue that the semantic approach to modifications of e, where "does not affect" is captured by means of universal quantification, is preferred. 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 involves *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). We also discuss two weaker alternative definitions of regular vacuity: a restriction of the universally quantified intervals to intervals of the same duration as e (in case such a duration is well defined), and an approximation of the dyadic quantification over intervals by monadic quantification over the Boolean events referred to in the regular expressions.

In the second part of this thesis we show that regular vacuity is decidable, and that the automata-theoretic approach to LTL [VW94] can be extended to handle dyadic universal quantification. Unlike monadic universal quantification, which can be handled with no increase in computational complexity, the extension to dyadic quantification involves an exponential blow-up (in addition to the standard exponential blow-up of handling LTL [SC85]), resulting in an EXPSPACE upper bound, which should be contrasted with a PSPACE upper bound for RELTL model checking. The NEXPTIME-hardness lower bound [AFF$^+$03](Appendix A), 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.

In the final part of this thesis we address several pragmatic aspects of vacuity checking. We first discuss whether vacuity should be checked with respect to subformulas or subformula occurrences and argue that both checks are necessary. We then discuss how the number of vacuity checks can be minimized and how vacuity results should be reported to the user. We argue that with respect to regular vacuity, 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 [KV03] can be extended to regular vacuity, which can then be reduced to standard model checking. Finally, we describe our experience of implementing vacuity checking in the context of a ForSpec-based model checker. We found vacuity detection useful in detecting wrong assumption (restricting the desired model behavior), detecting bugs in the model and detecting inaccurate properties. In fact, we only consider a verification task to be complete after vacuity analysis.

The thesis summarizes our work on vacuity detection covered in the following papers:

- R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, M. Vardi. Enhanced Vacuity Detection in Linear Temporal Logic. CAV 2003.
- D. Bustan, A. Flaisher, O. Grumberg, O. Kupfreman, M. Vardi. Regular Vacuity. Submitted.

Appendixes A and B prove the correctness of the construction for RELTL and regular vacuity. They are given as appendixes due to significant enhancements performed by Doron Bustan. Appendixes C and D prove lower bound of structure vacuity and regular vacuity. They were writen by Nir Piterman and Orna Kupferman and included for the sake of completeness.

1.1 Related Work

1.1.1 Vacuity Detection

The problem of trivially valid formulas was first noted by Beatty and Bryant [BB94], who termed it antecedent failure. It seems [BBER97] is the first attempt to automatically detect trivial passes under symbolic model checking. In addition to antecedent failure, Beer at el. cover other kinds of trivially true formulas, and call it a *vacuous pass*. They present interesting examples of vacuous passes, such as $AG(p \rightarrow AX(q \rightarrow AXr))$. The formula passes vacuously not only if p never occurs, but also if q never occurs at a next state of p. They define a subset of ACTL, called *w-ACTL* (witness ACTL), for which it is possible to construct a single formula $w(\varphi)$ which detects all vacuous passes of φ. A side affect of their method is that the witness formula which detects trivial passes, also provides an *interesting witness* when false. Examining an interesting witness can provide some confidence that the formal specification accurately reflects the intent of the user.

Beer at el. report that typically 20% of formulas pass vacuously during the first formal verification runs of new hardware designs, and that vacuous passes always point to real problem in either the design, the specification or the environment. They also report that of the formulas which pass *non-vacuously*, examination of the interesting witnesses discovers a problem with approximately 10% of the formulas. Examining such witnesses is orthogonal to vacuity detection.

According to Beer at el. *vacuity* occurs when one of the operands *does not affect* the validity of the formula. Formally, a sub formula ξ does not affect the truth value of φ in model M, if for every formula ξ', the truth value of φ in model M is the same as the truth of φ in model M. Here, φ' is the formula obtained by replacing ξ with ξ' in φ. They say that a formula φ passes vacuously in model M if it passes, and contains a subformula ξ such that ξ does not affect the truth of φ in M.

As mentioned, [BBER97] is restricted to a subset of ACTL called w-ACTL. They claim that in their experience, this subset is sufficient for expressing most of the formulas used by engineers for specification. w-ACTL formulas are ACTL formulas in which for all binary operators at least one of the operands is a propositional formula. For each operator, they define the *important* operand for which vacuity will be detected. They restrict vacuous passes only to cases where the non-affecting subformula is important. For example, in the formula $AG(req \rightarrow AFgrant)$ they check the case where req never happens, but ignore the case in which $AFgrant$ always holds.

According to [BBER97], an *interesting witness* is a path showing one instance of the truth of the formula, on which every important subformula affects the truth of the formula. Beer at el. show how to construct an interesting witness for a w-ACTL formula. They say that a formula w is a *witness formula* of φ, denoted $w(\varphi)$, if for any model M: (1) ($M \models \varphi$ and $M \models w(\varphi)$) iff φ passes vacuously in M. (2) If $M \models \varphi$ and $M \not\models w(\varphi)$ then any cex of $w(\varphi)$ in M is also an interesting witness of φ in M. They show how to construct a witness formula for any w-ACTL formula. Their construction algorithm replaces the smallest important subformula with $\mathbf{false}$.

Kupferman and Vardi [KV99, KV03] extend the work of [BBER97] by presenting a general method for detecting vacuity for specifications in CTL*. Beyond the extension of the method to a highly expressive specification language, they also give a stronger definition of vacuity, in the sense that they check whether all the subformulas of the specification affect its truth value. Given a formula φ and a subformula ψ, they denote by $\varphi\,[\psi \leftarrow \bot]$ the formula obtained from φ by replacing ψ by $\mathbf{true}$ if ψ is of negative polarity and by $\mathbf{false}$ if ψ is of positive polarity. They show that for a subformula occurrence ψ of φ and for every system M, if $M \models \varphi[\psi \leftarrow \bot]$, then for every formula ξ, we have $M \models \varphi[\psi \leftarrow \xi]$. It follows that vacuity detection involves model checking of M with respect to at most $|\varphi|$ formulas, and can be checked in time $O(C_M(|\varphi| \cdot |\varphi|))$, where $C_M(|\varphi|)$ is the complexity of the model checking problem. They show that for φ in CTL, a subformula ψ of φ with *multiple occurrences*, and a system M, the problem of deciding whether ψ does not affect φ in M is co-NP-complete.

Kupferman and Vardi also study the generation of interesting witnesses. Given a formula φ in either LTL or CTL*, they define $witness(\varphi) = \varphi \bigwedge_{\psi \in cl(\varphi)} \neg\varphi[\psi \leftarrow \bot]$. Intuitively, a path π satisfies $witness(\varphi)$ if π satisfies φ and in addition, π does not satisfy the formula $\varphi[\psi \leftarrow \bot]$ for all the subformulas ψ of φ. They show that a counter example for $\neg witness(\varphi)$ in M, is an interesting witness for φ in M. They conclude that for φ in CTL*, the problem of generating an interesting witness for φ in M is PSPACE-complete.

Purandare and Somenzi [PS02] examine the practicality and usefulness of [BBER97, KV03] for CTL. They show that a thorough vacuity check as in [KV03] can be implemented efficiently for CTL, so that the overhead relative to plain model checking is in practice very limited in spite of the worst case complexity. Instead of checking φ and the witness formula generated by various replacements in a sequential fashion, they check φ and all its replacements in a single bottom-up pass over the parse tree or φ. At each node they exploit the relationships between the sets of states satisfying the various formula. According to polarity, the satisfying set of a witness subformula is either a lower bound or an upper bound on the satisfying set of the corresponding subformulas of φ. This allows them to speed up fixpoint computations by accelerating convergence, or simplifying the computation of preimages. They also detect cases in which different replacements lead to an equivalent formula.

Purandare and Somenzi provide several examples of vacuity, including one where thorough vacuity detection is required. They consider the formula $AG(start \wedge valid(x) \wedge valid(y) \rightarrow valid(z))$, where $start$ holds in the first clock of the computation, and $valid()$ tells whether the inputs x and y or the output z are not denormals (this is a floating point multiplier). They report that out of 24 replacements, 20 produce vacuous passes. They revealed that (1) The environment of the model lacks an assignment to $start$; (2) The MSB of the exponent could be incorrect due to overflow during its computation; and (3) The multiplier maintains the invariant $AG\, valid(z)$. These bugs were found because each atomic proposition was replaced separately, and the detection that the antecedent is redundant.

Two recent papers by Gurfinkel and Checkik examine additional aspects of vacuity. In [GC04a] Gurfinkel and Checkik show the relation between vacuity detection and the 3-valued Kleene logic. Simple vacuity detection is exactly the *3-valued model checking* problem. They show generalizations of the vacuity problem to multi-valued model checking, such as four valued-model checking to determine if a formula is vacuous and true, or vacuous and false. The paper deals with subformula occurrences in CTL.

The idea of using multi-valued logic for encoding different degrees of vacuity is also applicable to cases where we want to check vacuity of a formula φ with respect to *several subformulas*, or multiple occurrences of the same formula. They introduce the notion of *mutual vacuity* between different subformulas. Logic values encode different degrees of vacuity, such as "φ is mutually true in a and b, vacuous in c and independent of d, and non vacuous in e".

In [GC04b] Gurfinkel and Checkik relate to the comparison between the three alternative definitions of vacuity in [AFF$^+$03] (see chapter 3), and claim that although [AFF$^+$03] shows that structure and formulas semantics are sensitive to the model and specification language, the robustness of trace semantics is not formally defined. They formalize the notion of robust vacuity and use our quantified temporal logic formulation to extend semantic vacuity to CTL*. Their definition requires a vacuous pass in every model K' that is bisimilar to K. When moving from LTL to CTL* [GC04b] move from traces to trees. They show that vacuity detection for CTL* is expensive (2EXPTIME-complete) , and define fragments of

CTL* for which detecting vacuous satisfaction in not harder than model checking.

Gurfinkel and Checkik also show that vacuity is preserved by abstraction. They show that vacuity detection is more precise than traditional abstract model checking in the sense that sometimes, it is possible to determine that a formula is vacuously satisfied by an abstract model, even if the result of abstract model checking is inconclusive.

1.1.2 Coverage

The notions of coverage and vacuity are closely related. Coverage metrics are based on modifications applied to the *system* (rather than the specification) in order to check which parts of it were actually relevant for the verification process to succeed. Chockler, Kupferman and Vardi [CKV01] suggest several coverage metrics for model checking, and describe two alternative algorithms for finding the uncovered parts of the system under these definitions.

Suspecting the system of containing an error even in the case model checking succeeds, is the basis for both vacuity detection and coverage in temporal logic model checking. Clearly, an erroneous behavior of the system can escape the verification if this behavior is not captured by the specification. Coverage metric techniques are common in simulation based verification [HMA95, HYHD95, DGK96, MAH98, BH99, FAD99]. However, these metrics cannot be applied to model checking as the process of model checking visits all states.

The idea of coverage in temporal logic model checking (coverage) is that a state is uncovered if its labeling is not essential to the success of the model checking process. There are two approaches for defining and developing algorithms for coverage metrics in temporal logic model checking. The first approach, of Katz et al. [KGG99], is based on a comparison of the system with a tableau of the specification. This approach is somewhat strict, as we want specifications to be much more abstract than their implementations, and as sometimes, only part of the design is checked using model checking. A refinement of this approach enables specifying which parts of the model are relevant. The second approach, of Hoskote et al. [HKHZ99], is to define coverage by examining modifications in the system on the satisfaction of the specification. A state w is *q-covered* by φ if the Kripke structure obtained from K by flipping the value of q in w (denoted $\tilde{K}_{w,q}$) no longer satisfies φ. That is, the value of q in w is crucial for the satisfaction of φ in K. By [HKHZ99], a state is covered if it belongs to q-cover(K, φ) for some signal q. This approach resembles vacuity detection, where we examine modifications in the specification on its satisfaction in the system.

Chockler et al. [CKV01] introduce two principals, which they believe should be part of any coverage metric for model checking: a distinction between state-based and logic-based coverage, and a distinction between the system and its environment. The state based approach modifies q in every state of the Kripke structure K. On the other hand, when the system is modeled as a circuit, flipping the value of a signal in a state changes not only the label of the state but also the transitions to and from the state. In the logic-based approach, the value of a signal is fixed to 0,1 or don't care everywhere in the circuit. These two approaches are similar to the structure and trace semantics we examine for vacuity detection in chapter 3. The second principal differentiates between closed and open systems, the later having an interface with the environment. Clearly, there is no point to talk about q-coverage for a signal q that corresponds to an input variable. Similarly, there is no point in checking vacuity with respect to an input variable, as the formula is already satisfied for every possible behavior of the input.

Two alternative definitions for the naive algorithm for coverage, which finds the set of covered states or signals by model checking each of the modified systems, are presented in [CKV01]. The first alternative is a symbolic approach to finding the uncovered parts of the system. Notice that changing q in several states together may have a different affect than changing q in each state alone. The second alternative is an algorithm that makes use of overlaps among the modified systems. Since each modification involves a small change in the original system, there is a great deal of work that can be shared when we model check all the modified systems. Both algorithms work on full CTL.

Chockler et al. also relate to the presentation of output. For a circuit S and a signal q let q-cover(S, φ) denote the set of states q-covered by φ in S. They propose to check at first whether q-cover(S, φ) is empty for some q, before merging all results. An empty set may indicate vacuity in the specification. Another interesting output are computations that contain no covered states or many state that are not covered. Such computations correspond to behaviors of the circuit that are not referred to by the specification. We refer to the presentation of vacuity results in section 8.1.

Finally [CKV01] raise several open issues with respect to coverage metrics to temporal logic model checking. One is incompleteness of the specification vs. redundancies in the system. Another is the feasibility of coverage algorithms, as their complexity is larger than model checking. Chockler, Kupferman, Kurshan and Vardi address coverage metrics from a practical point of view in [CKKV01]. They suggest several definitions of coverage for LTL specifications, and describe two algorithms for computing the parts of the system that are not covered by the specification. The first algorithm is built on top of automata-based model checking, and the second reduces the coverage problem to a model checking problem.

In [CKKV01] the three alternative definition of coverage for LTL specifications are structure coverage (”flipping always”), node coverage (”flipping once”) and tree coverage (”flipping sometimes”). Each approach measures a different sensitivity of the satisfaction of the specification to changes in the system. Chockler at el. use $SC(K, \varphi, q)$, $NC(K, \varphi, q)$, $TC(K, \varphi, q)$ to denote sets of states that are structure $q-$covered, node $q-$covered and tree $q-$covered, respsectively in K. They show that $SC(K, \varphi, q) \subseteq TC(K, \varphi, q)$, $NC(K, \varphi, q) \subseteq TC(K, \varphi, q)$, and that $SC(K, \varphi, q) \nsubseteq NC(K, \varphi, q)$, $NC(K, \varphi, q) \nsubseteq SC(K, \varphi, q)$. The later relation resembles the occurrence-subformula relation described in section 8.2. In vacuity, as in coverage, we cannot prefer one over the other as there are examples where a vacuous pass is only detected when we check vacuity with respect to a subformula, and vice versa.

Chockler at el. show easy implementation for node coverage in the tool COSPAN, which is the engine of FormalCheck, and show that the implementation can be modified in order to handle structure and tree coverage. The implementation is done by introducing two new Boolean variables $flip$ and $flag$, which flip the value of q exactly once when both $flip$ and $flag$ are asserted. The increase in the number of variables is only by 2, thus the complexity remains $O(2^n)$. The complexity for structure and tree coverage is a function of the size of the state space which is at most exponential in the number of state variables. For both tree and structure coverage, Chockler at el. double the number of variables by introducing n new variables that encode the flipped state. Thus the state-space size is $O(2^{2n})$ instead of $O(2^n)$.

1.2 Organization

In the next chapter we give necessary background on automata theory and temporal logic, including UQLTL which augments LTL with quantification over propositional variables. The remainder of the thesis is organized in three parts. Part 1 covers subformula vacuity. We compare three alternative definitions of vacuity and show an efficient algorithm for vacuity detection with respect to trace semantics, which handles subformulas of mixed polarity. Part 2 covers regular vacuity, which is vacuity detection with respect to regular expressions. We introduce RELTL, a temporal logic that extends LTL with regular expressions, define regular vacuity, and provide an algorithm that determines regular vacuity (but involves another exponential blow-up). Part 3 covers pragmatic aspects of both subformula vacuity and regular vacuity. Appendixes A and B prove the correctness of the construction for RELTL and regular vacuity. Appendixes C and D prove lower bound of structure vacuity and regular vacuity.

Chapter 2

Preliminaries

2.1 Automata

Definition 2.1.1 (NFW) *A nondeterministic finite word automaton (NFW) is a tuple* $Z = \langle \Sigma, Q, \Delta, q_0, W \rangle$ *s.t.* Σ *is an alphabet,* Q *is a set of states,* $\Delta : (Q \times \Sigma) \to 2^Q$ *is a transition relation,* q_0 *is a single initial state and* $W \subseteq Q$ *is the set of accepting states.*

Let $\pi = \pi_0, \pi_1, \ldots$ be a finite/infinite word over Σ. For $i \in \mathbb{N}$, let $\pi^i = \pi_i, \pi_{i+1}, \ldots$ denote the suffix of π from its ith letter. A sequence $\xi = q_0, q_1, \ldots q_n$ in Q^* is a *run* of Z over a finite word $\pi \in \Sigma^*$, if q_0 is the initial state, and for every $0 \leq i < n$, we have $q_{i+1} \in \Delta(q_i, \pi_i)$. A run ξ of Z is *accepting* if $q_n \in W$. An NFW Z accepts a word π if Z has an accepting run over π. We use $L(Z)$ to denote the set of words that are accepted by Z. For $q \in Q$, we denote by Z^q the automaton Z with a single initial state q.

Definition 2.1.2 (NGBW) *A* nondeterministic generalized Büchi word automaton *(NGBW) is* $\mathcal{A} = \langle \Sigma, S, S_0, \delta, \mathcal{F} \rangle$*, where* Σ *is a finite set of alphabet letters,* S *is a set of states,* $\delta : S \times \Sigma \to 2^S$ *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 sequence $\rho = s_0, s_1, \ldots$ in S^ω is a *run* of $\mathcal{A}$ over an infinite word $\pi \in \Sigma^\omega$, if $s_0 \in S_0$ and for every $i > 0$, we have $s_{i+1} \in \delta(s_i, \pi_i)$. We use $inf(\rho)$ to denote the set of states that appear infinitely often in ρ. A run ρ of $\mathcal{A}$ is *accepting* if for every $F \in \mathcal{F}$, we have $inf(\rho) \cap F \neq \emptyset$. An NGBW $\mathcal{A}$ accepts a word π if $\mathcal{A}$ has an accepting run over π. We use $L(\mathcal{A})$ to denote the set of words that are accepted by $\mathcal{A}$. For $s \in S$, we denote by $\mathcal{A}^s$ the automaton $\mathcal{A}$ with a single initial state s.

2.2 Temporal Logic

We define the temporal logic LTL over a set of atomic propositions AP in positive normal form. The syntax of LTL is as follows. An atom $p \in AP$ is a formula and so is $\neg p$. If φ_1 and φ_2 are LTL formulas, then so are $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\textsf{next}\ \varphi_1$, $\varphi_1\ \textsf{until}\ \varphi_2$, and $\varphi_1\ \textsf{release}\ \varphi_2$. Each LTL formula φ induces a language $L(\varphi) \subseteq (2^{AP})^\omega$ of exactly all the infinite words that satisfy φ.

The semantics of LTL is defined with respect to an infinite word $\pi \in (2^{AP})^\omega$. We use $\pi, i \models \varphi$ to indicate that the word π^i satisfies the formula φ. The relation $\models$ is defined by induction on the structure of φ as follows.

- For $p \in AP$, we have $\pi, i \models p$ iff $p \in \pi_i$, and $\pi, i \models \neg p$ iff $p \notin \pi_i$.

Let $\varphi, \varphi_1, \varphi_2$ be formulas.

- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$.
- $\pi, i \models \varphi_1 \vee \varphi_2$ iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$.
- $\pi, i \models\ \textsf{next}\ \varphi$ iff $\pi, i+1 \models \varphi$.
- $\pi, i \models \varphi_1\ \textsf{until}\ \varphi_2$ iff there exists $k \geq i$ such that $\pi, k \models \varphi_2$ and for all $i \leq j < k$ we have $\pi, j \models \varphi_1$.
- $\pi, i \models \varphi_1\ \textsf{release}\ \varphi_2$ iff either for some $j \geq i$ $\pi, j \models \varphi_1$ and for every $i \leq k \leq j$ we have $\pi, k \models \varphi_2$, or for every $j \geq i$, $\pi, j \models \varphi_2$.

We use the operator ($\textsf{eventually}\ \varphi$) as a shorthand for ($\mathbf{true}\ \textsf{until}\ \varphi$), and we use the operator ($\textsf{globally}\ \varphi$) as a shorthand for ($\mathbf{false}\ \textsf{release}\ \varphi$). Finally we define *regular expression* over an alphabet Σ. The syntax of regular expressions is as follows. A letter $a \in \Sigma$ is a regular expression. If e_1 and e_2 are regular expressions, then so are $e \cdot e$, $e + e$, and e^*. Each regular expression e induces a language $L(e) \subseteq \Sigma^*$ of exactly all the finite words that satisfy e. The semantics of regular expressions is defined as follows:

- For $a \in \Sigma$, $L(a)$ is the single word a.

For regular expressions e_1 and e_2.

- $L(e_1 \cdot e_2)$ consists of all words formed by concatenating a word in $L(e_1)$ with a word in $L(e_2)$.
- $L(e_1 + e_2)$ is the union of $L(e_1)$ and $L(e_2)$.

- $L(e^*)$ consists of all words formed by concatenating zero or more words from $L(e)$, and includes the empty word ϵ.

Definition 2.2.1 (Occurrence and Subformula Polarity) *An occurrence of formula ψ of φ is of* positive polarity *in φ if it is in the scope of an even number of negations, and of* negative polarity *otherwise. The polarity of a subformula is defined by the polarity of its occurrences as follows. Formula ψ is of* positive polarity *if all occurrences of ψ in φ are of positive polarity, of* negative polarity *if all occurrences of ψ 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 ψ in φ are of positive polarity and some are of negative polarity.*

Given a formula φ and a subformula of pure polarity ψ we denote by $\varphi\,[\psi \leftarrow \bot]$ the formula obtained from φ by replacing ψ by $\mathbf{true}$ if ψ is of negative polarity and by $\mathbf{false}$ if ψ is of positive polarity. Dually, $\varphi\,[\psi \leftarrow \top]$ denotes the formula obtained from φ by replacing ψ by $\mathbf{false}$ if ψ is of negative polarity and by $\mathbf{true}$ if ψ is of positive polarity.

2.3 UQLTL

The logic UQLTL augments LTL with universal quantification over *propositional variables*. Let X be a set of propositional variables and let $x \in X$. The syntax of LTL is extended as follows. If φ is an LTL formula over the extended set of atomic propositions $AP \cup X$, then $\forall x \varphi$ is a UQLTL formula. E.g., $\forall x$ globally $(x \rightarrow p)$ is a legal UQLTL formula, while globally $\forall x\,(x \rightarrow p)$ is not. UQLTL is a subset of *Quantified Propositional Temporal Logic* [SVW85], where all the free variables are quantified universally. In the sequel, we use x to denote a propositional variable. A *closed* formula is a formula with no free propositional variables.

We now give definitions of two semantics for UQLTL formulas. The first is *structure semantics* where a propositional variable is bound to a subset of the states of the Kripke structure. The second is *trace semantics* where a propositional variable is bound to a subset of the locations on the trace. Structure semantics is defined with respect to a Kripke structure $K =< AP, W, R, w_0, L >$, where AP is the set of atomic propositions, W is a set of states, $R \subseteq W \times W$ is the transition relation that must be total (i.e. for every $w \in W$ there exists $w' \in W$ s.t. $R(w, w')$), w_0 is an initial state, and $L : W \rightarrow 2^{AP}$ maps each state to a set of atomic propositions true in this state. A path of K is an infinite sequence

$\pi = w_0, w_1, w_2, \cdots$ of states s.t. for all $i \geq 0$ we have $R(w_i, w_{i+1})$. Let $\mathcal{T}(M)$ denote the set of computations of M.

Let M be a Kripke structure with a set of states S, let $\pi \in \mathcal{T}(M)$, and let X be a set of propositional variables. A *structure assignment* $\sigma : X \to 2^S$ maps every propositional variable $x \in X$ to a set of states in S. We use s_i to denote the ith state along π, and φ to denote UQLTL formulas.

Definition 2.3.1 (UQLTL Structure Semantics) *The relation $\models_s$ is defined inductively as follows:*

- $M, \pi, i, \sigma \models_s x$ *iff* $s_i \in \sigma(x)$.
- $M, \pi, i, \sigma \models_s \forall x \varphi$ *iff* $M, \pi, i, \sigma[x \leftarrow S'] \models_s \varphi$ *for every* $S' \subseteq S$.
- *For any other formula* φ*,* $M, \pi, i, \sigma \models_s \varphi$ *is defined as in LTL.*

A closed UQLTL formula φ is *structure satisfied* at point i of trace $\pi \in \mathcal{T}(M)$, denoted $M, \pi, i \models_s \varphi$, iff $M, \pi, i, \sigma \models_s \varphi$ for some σ (choice is not relevant since φ is closed). A closed UQLTL formula φ is structure satisfied in structure M, denoted $M \models_s \varphi$, iff $M, \pi, 0 \models_s \varphi$ for every trace $\pi \in \mathcal{T}(M)$.

We now define the trace semantics for UQLTL. Let X be a set of propositional variables. A *trace assignment* $\alpha : X \to 2^{\mathbf{N}}$ maps a propositional variable $x \in X$ to a set of natural numbers (points on a path).

Definition 2.3.2 (UQLTL Trace Semantics) *The relation $\models_t$ is defined inductively as follows:*

- $M, \pi, i, \alpha \models_t x$ *iff* $i \in \alpha(x)$.
- $M, \pi, i, \alpha \models_t \forall x \varphi$ *iff* $M, \pi, i, \alpha[x \leftarrow N'] \models_t \varphi$ *for every* $N' \subseteq \mathbf{N}$.
- *For any other formula* φ*,* $M, \pi, i, \sigma \models_t \varphi$ *is defined as in LTL.*

A closed UQLTL formula φ is *trace satisfied* at point i of trace $\pi \in \mathcal{T}(M)$, denoted $M, \pi, i \models_t \varphi$, iff $M, \pi, i, \alpha \models_t \psi$ for some α (choice is not relevant since φ is closed). A closed UQLTL formula φ is trace satisfied in structure M, denoted $M \models_t \varphi$, iff $M, \pi, 0 \models_t \varphi$ for every trace $\pi \in \mathcal{T}(M)$.

We show that trace semantics is stronger than structure semantics in the following sense. Whenever a UQLTL formula holds according to trace semantics it holds according to structure semantics. The opposite is not true. Indeed, a trace assignment can assign a variable different values when the computation visits the same state of M at different point in the trace. We observe that for LTL formulas both semantics are identical. That is, if φ is an LTL formula, then $M \models_s \varphi$ iff $M \models_t \varphi$. We sometimes use $\models$ to denote the satisfaction of an LTL formula, rather than $\models_s$ or $\models_t$.

Theorem 2.3.3 *Given a structure M and a UQLTL formula φ:*

- $M \models_t \varphi \quad \Rightarrow \quad M \models_s \varphi$
- $M \models_s \varphi \quad \not\Rightarrow \quad M \models_t \varphi$

The proof resembles the proofs in [Kup95] for the dual logic EQCTL. Kupferman shows that a structure might not satisfy a formula, although the formula is satisfied by its computation tree.

Proof: Assume in the way of contradiction that $M \models_t \varphi$ but $M \not\models_s \varphi$. Then there exists a structure assignment σ and a trace π such that $M, \pi, 0, \sigma \not\models_s \varphi$. Let $\pi = s_0, s_1, s_2, \ldots$. We build the assignment $\alpha(x) = \{i|\, s_i \in \sigma(x)\}$, which includes point i in the assignment α of a propositional variable x iff s_i is in $\sigma(x)$. Both assignments map all propositional variables in φ to the same truth values along the trace π, thus $M, \pi, 0, \alpha \not\models_t \varphi$. This implies that $M \not\models_t \varphi$, in contradiction with the assumption.

For the other direction, consider the formula $\varphi = \forall x \ \mathsf{globally}\ (x \rightarrow \ \mathsf{next}\ x)$ and a Kripke structure with a single state s_0 that has a self loop.

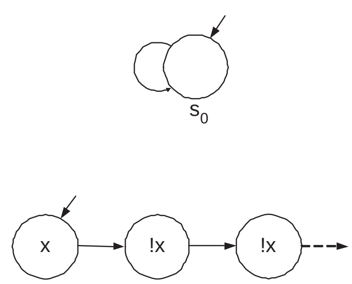

Figure 2.1: $M \models_s \varphi \not\Rightarrow M \models_t \varphi$.

We show that $M, \sigma \models_s \varphi$ for every σ. There are two possible structure assignments, $\sigma(x) = \emptyset$ and $\sigma(x) = s_0$. If $s_0 \in \sigma(x)$, then x is always satisfied and

$M, \sigma \models_s \varphi$. If $s_0 \notin \sigma(x)$, then $X\neg x$ is always satisfied and $M, \sigma \models_s \varphi$. Thus $M \models_s \varphi$.

We now show that $M \not\models_t \varphi$. Notice that M has a single trace π. Consider the trace assignment α that maps x only to the first point along π. That is, $\alpha(x) = \{0\}$. For that assignment $M, \pi, 0, \alpha \not\models_t \varphi$, which implies $M \not\models_t \varphi$.

$\square$

Part I

Subformula Vacuity

Chapter 3

Alternative Definitions of Vacuity

Let ψ be a subformula of φ. We give three alternative definitions of when ψ *does not affect* φ, and compare them. We refer to the definition of [BBER97] as *formula vacuity*. We give two new definitions, *trace vacuity* and *structure vacuity*, according to trace and structure semantics. Notice that we are only interested in the cases where φ is satisfied in the structure.

Intuitively, ψ does not affect φ in M if we can perturb ψ without affecting the truth of φ in M. In previous work, syntactic perturbation was allowed. Using UQLTL we formalize the concept of semantic perturbation. Instead of changing ψ syntactically, we directly change the set of points in a structure or on a trace in which it holds. That is, we replace ψ by a propositional variable that can receive any value (according to the relevant sematic definition).

Definition 3.0.4 (Does Not Affect) *Let φ be a formula satisfied in M where φ and M are both defined over AP. Let ψ be a subformula of φ.*

- ψ does not affect$_f$ φ *in M iff for every LTL formula ξ defined over AP, we have* $M \models \varphi\,[\psi \leftarrow \xi]$ [BBER97].
- ψ does not affect$_s$ φ *in M iff* $M \models_s \forall x \varphi\,[\psi \leftarrow x]$.
- ψ does not affect$_t$ φ *in M iff* $M \models_t \forall x \varphi\,[\psi \leftarrow x]$.

We say that ψ *affects$_f$* φ in M iff it is not the case that ψ does not affect$_f$ φ in M. We say that φ is *formula vacuous* in M, if there exists a subformula ψ such that ψ does not affect$_f$ φ. We define *affects$_s$*, *affects$_t$*, *structure vacuity* and *trace vacuity* similarly.

3.1 Comparing the Alternative Definitions of Vacuity

In the following section we compare the three alternative definitions of vacuity. We show that they are all different. We also argue why trace vacuity is the preferred definition. Notice that we do not restrict a subformula to occur once and it can be of mixed polarity. We show that our three semantics form a hierarchy, with structure semantics being the weakest and trace semantics the strongest.

We show that structure vacuity is weaker than formula vacuity. That is, ψ might affect$_f$ φ in M, but not affect$_s$ φ in M.

Lemma 3.1.1 (Relating Structure and Formula Vacuity) *Let φ be an LTL formula. If ψ does not affect$_f$ φ in M, then it does not affect$_s$ φ in M as well. The reverse implication does not hold.*

In the following proof, we assume that every state has a *unique representation* using the atomic propositions. That is, every state in the structure satisfies a different set of atomic propositions. This is a reasonable assumption for hardware modeling.

Proof: First we prove that if a subformula ψ does not affect$_f$ φ in M, then it does not affect$_s$ φ in M as well. If ψ affects$_s$ φ, then there exists a structure assignment σ and a computation π of M such that $M, \pi, 0, \sigma \not\models_s \varphi\,[\psi \leftarrow x]$. We construct a formula ψ' that behaves like x along π, that is, $M, \pi, i \models \psi'$ iff $M, \pi, i, \sigma \models_s x$. Let $\overline{s}$ be a predicate over AP that is $\mathbf{true}$ only in state $s \in S$. Let ψ' be the disjunction of $\overline{s}$ for all states in $\sigma(x)$. The formula ψ' is well-defined since S is finite. We show that $M, \pi, i, \sigma \models_s x$ iff $M, \pi, i \models \psi'$. If $M, \pi, i, \sigma \models_s x$ then, by the definition of structure semantics, $s_i \in \sigma(x)$. Therefore $\overline{s}_i$ is in the disjunction ψ'. Since $M, \pi, i \models \overline{s}_i$, we have $M, \pi, i \models \psi'$. On the other hand, if $M, \pi, i, \sigma \not\models_s x$ then $s_i \notin \sigma(x)$, and therefore $\overline{s}_i$ is not included in the disjunction ψ'. Since every state is uniquely labeled $M, \pi, i \models \overline{s}_j$ iff $s_j = s_i$. Consequently $M, \pi, i \not\models \psi'$. Thus we have shown that if $M, \sigma \not\models_s \varphi\,[\psi \leftarrow x]$ then $M \not\models \varphi\,[\psi \leftarrow \psi']$.

In the other direction, we construct an LTL formula ψ' that assumes different values when visiting the same state. Let M be the Kripke structure in figure 3.1 and consider the following formula:

$$\varphi = p \vee (\textsf{ globally } \textsf{ eventually } q) \vee (\textsf{ eventually } \textsf{ globally } \neg p)$$

We examine if p affects$_f$ φ. Consider the path $\pi' = s_0, s_1, s_0, s_0, s_0 \ldots$ and let $\psi' = \textsf{ globally } \neg q$. Thus, $\varphi\,[p \leftarrow \psi'] = (\textsf{ globally } \neg q) \vee (\textsf{ globally } \textsf{ eventually } q) \vee$

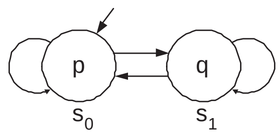

Figure 3.1: Relating structure and formula vacuity.

(eventually globally eventually q). Clearly, $\pi' \not\models \varphi\,[p \leftarrow \psi']$ and p affects$_f$ φ.

On the other hand, for every trace π and every assignment σ, we have $M, \pi, 0, \sigma \models_s \varphi\,[\psi \leftarrow x]$. That is, $M, \pi, 0, \sigma \models_s x \vee($ globally eventually $q) \vee ($ eventually globally $\neg x)$. If $s_0 \in \sigma(x)$ then the disjunct x is satisfied. If $s_0 \notin \sigma(x)$ then for all traces that from some point on remain in s_0 eventually globally $\neg x$ is satisfied, for all other paths, globally eventually q is satisfied. □

We now show that formula vacuity is weaker than trace vacuity. That is, ψ might affect$_t$ φ in M, but not affect$_f$ φ in M.

Lemma 3.1.2 (Relating Trace and Formula) *Let φ be an LTL formula. If ψ does not affect$_t$ φ in M, then ψ does not affect$_f$ φ in M as well. The reverse implication does not hold.*

Proof: We show that if ψ affects$_f$ φ, then it also affects$_t$ φ. If ψ affects$_f$ φ, then there exists a formula ψ' such that $M \not\models \varphi\,[\psi \leftarrow \psi']$. Let π be a trace in M such that $\pi \not\models \varphi\,[\psi \leftarrow \psi']$. Consider the assignment $\alpha(x) = \{i | \, \pi, i \models \psi'\}$. Clearly, $M, \pi, 0, \alpha \not\models_t \varphi\,[\psi \leftarrow x]$, and therefore ψ affects$_t$ φ.

In the other direction, let M be a Kripke structure with a single state labeled by p, with a self-loop. Let $\varphi = (p \rightarrow$ next $p)$. It can be shown that $M \not\models_t \forall x\,\varphi\,[p \leftarrow x]$, thus p affects$_t$ φ. We now show that there cannot exist an LTL formula ψ' such that $M \not\models \varphi\,[p \leftarrow \psi']$. Notice that M has a single trace π, and that $tail(\pi) = \pi$. This means that ψ' is either true along every suffix of π, or ψ' is false along every suffix of π. However $M \not\models \varphi\,[p \leftarrow \psi']$ only if ψ' holds at time zero but fails at time one. □

Which is the most appropriate definition for practical applications? We show that structure vacuity and formula vacuity are sensitive to changes in the design that do not relate to the formula. As an example, consider the formula $\varphi = p \rightarrow$ next p and models M_1 and M_2 in the figure below. In M_2 we add a proposition q

whose behavior is independent of p's behavior. We would not like formulas that relate to p to change their truth value or their vacuity. Both M_1 and its extension M_2 satisfy φ and φ relates only to the proposition p. While p does not affect$_f$ φ in M_1, it does affect$_f$ φ in M_2 (and similarly for affects$_s$). Indeed, the formula $\varphi\,[p \leftarrow q] = q \rightarrow \textsf{next}\, q$ does not hold in M_2. Note that in both models p affects$_t$ φ.

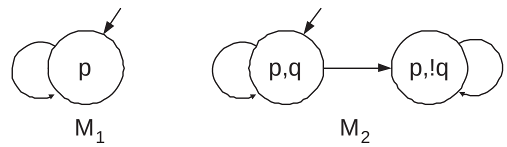

Figure 3.2: Sensitivity of structure and formula vacuity to changes in the design.

Another disadvantage of formula vacuity is that it is *sensitive to the specification language*. That is, a formula passing vacuously might pass unvacuously once the specification language is extended. As an example, consider the following Kripke structure M_1 and the LTL formula $\varphi = \textsf{next}\, q \rightarrow \textsf{next}\ \textsf{next}\, q$. For the

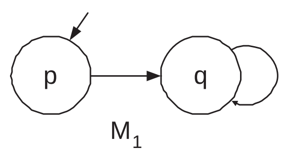

Figure 3.3: Sensitivity of formula vacuity to the specification language.

single trace $\pi \in \mathcal{T}(M_1)$, it holds that $tail(\pi^1) = \pi^1$. Thus, every (future) LTL formula is either true along every suffix of π^1, or is false along every such suffix. This implies that subformula q does not affect$_f$ φ. However, we get an opposite result if the specification language is LTL augmented with the PAST operator [LPZ85]. The PAST operator enables reference to the history of the computation. Formally, if ψ is an LTL formula then $M, \pi, i \models PAST(\psi)$ iff $i > 0$ and $M, \pi, i-1 \models \psi$. Clearly, for every model M we have $M, \pi, 0 \not\models PAST(p)$. In the example, $M_1 \not\models \varphi\,[q \leftarrow PAST(p)]$ since $M_1, \pi, i \models PAST(p)$ iff $i = 1$, thus q affects$_f \varphi$.

To summarize, trace vacuity is preferable since it is less sensitive to changes in the design (as opposed to structure and formula vacuity) and it is independent of the specification language (as opposed to formula vacuity). In structure semantics, the truth value of the propositional variable along the trace depends on the model. Similarly, in formula semantics, the truth value depends on the specification language. On the other hand, trace semantics can assign any value to the propositional variable in every point along the trace. The following lemma shows that changing model M in a way which is irrelevant to a formula φ (as in figure 3.2), does not alter the vacuity of φ in M.

Lemma 3.1.3 *Let $\mathcal{T}(M)|_{AP(\varphi)}$ denote the set of computations in M projected to the set of atomic propositions in φ. Then for every formula φ and model M' s.t. $\mathcal{T}(M')|_{AP(\varphi)} = \mathcal{T}(M)|_{AP(\varphi)}$ we have that φ is trace vacuous in M iff it is trace vacuous in M'.*

Proof: Assume $M \models_t \forall x\varphi\,[\psi \leftarrow x]$ for every subformula ψ of φ, but $M' \not\models_t \forall x\varphi\,[\psi \leftarrow x]$. Then there exists a trace π', a subformula ψ and an assignment α s.t. $M', \pi', 0, \alpha \not\models_t \varphi\,[\psi \leftarrow x]$. However, there also exists a trace $\pi \in \mathcal{T}(M)$ s.t. $\pi|_{AP(\varphi)} = \pi'|_{AP(\varphi)}$. Therefore $M, \pi, 0, \alpha \not\models_t \varphi\,[\psi \leftarrow x]$, which implies that $M \not\models_t \forall x\varphi\,[\psi \leftarrow x]$ in contradiction to the assumption. The other direction is identical. $\square$

Another reasoning for the superiority of trace vacuity is given in chapter 4 (Algorithm and Complexity).

3.2 Comparing the Alternative Definitions of Vacuity under Pure Polarity

In the following section we show that if subformulas are restricted to pure polarity, all the definitions of vacuity coincide. For that, we show that the replacement of subformula ψ by $\perp$ is adequate for vacuity detection according to all three definitions. This result is an extension of the results in [KV03], where only single occurrence was considered.

Lemma 3.2.1 *For every structure M, LTL formula φ and subformula ψ of φ of pure polarity, $M \models_t \varphi\,[\psi \leftarrow \perp]$ iff $M \models_t \forall x\varphi\,[\psi \leftarrow x]$.*

The first direction ($M \models_t \varphi\,[\psi \leftarrow \perp]$ if $M \models_t \forall x\varphi\,[\psi \leftarrow x]$) is immediate. The other direction follows from the claim below. Let θ denote a subformula of φ that may or may not contain the subformula ψ.

Claim 3.2.2 *For every occurrence of θ, every trace $\pi \in \mathcal{T}(M)$ and location i,*

- *if θ is of positive polarity in φ then $M, \pi, i \models \theta\,[\psi \leftarrow \bot]$ implies $M, \pi, i \models_t \forall x\, \theta\,[\psi \leftarrow x]$*
- *if θ is of negative polarity in φ then $M, \pi, i \not\models \theta\,[\psi \leftarrow \bot]$ implies $M, \pi, i \models_t \forall x \neg\theta\,[\psi \leftarrow x]$*

Proof: We prove the claim by induction on the structure of θ. We prove the case that ψ is of positive polarity (i.e. in $\theta\,[\psi \leftarrow \bot]$ the subformula ψ is replaced by $\mathbf{false}$). The case of negative polarity is dual. If ψ is not a subformula of θ the claim follows. Assume that ψ is a subformula of θ.

Let $\theta = p$ for some proposition p. Clearly, also $\psi = p$ and the claim follows. Let $\theta = \theta_1 \wedge \theta_2$. Suppose that the polarity of θ in φ is positive. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then clearly $M, \pi, i \models \theta_1\,[\psi \leftarrow \mathbf{false}]$ and $M, \pi, i \models \theta_2\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \theta_1\,[\psi \leftarrow x]$ and $M, \pi, i \models_t \forall x \theta_2\,[\psi \leftarrow x]$. Clearly, the claim follows. Suppose that the polarity of θ in φ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then either $M, \pi, i \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$ or $M, \pi, i \not\models \theta_2\,[\psi \leftarrow \mathbf{false}]$. Wlog suppose $M, \pi, i \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$. It follows that $M, \pi, i \models_t \forall x \neg\theta\,[\psi \leftarrow x]$.

Let $\theta = \theta_1 \vee \theta_2$. Suppose that the polarity of θ in φ is positive. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then either $M, \pi, i \models \theta_1\,[\psi \leftarrow \mathbf{false}]$ or $M, \pi, i \models \theta_2\,[\psi \leftarrow \mathbf{false}]$. Wlog suppose $M, \pi, i \models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \theta_1\,[\psi \leftarrow x]$. Clearly, the claim follows. Suppose that the polarity of θ in φ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then both $M, \pi, i \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$ and $M, \pi, i \not\models \theta_2\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$ and $M, \pi, i \models_t \forall x \neg\theta_2\,[\psi \leftarrow x]$. It follows that $M, \pi, i \models_t \forall x \neg\theta\,[\psi \leftarrow x]$.

Let $\theta = \neg\theta_1$. Suppose that the polarity of θ in φ is positive. Then the polarity of θ_1 in φ is negative. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then $M, \pi, i \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$. However, $\neg\theta_1\,[\psi \leftarrow x] \equiv \theta\,[\psi \leftarrow x]$ and the claim follows. Suppose that the polarity of θ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then $M, \pi, i \models \theta_1\,[\psi \leftarrow \mathbf{false}]$ and from the induction assumption we know that $M, \pi, i \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$. The claim follows.

Let $\theta = \mathsf{next}\,\theta_1$. Suppose that the polarity of θ is positive. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then $M, \pi, i+1 \models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i+1 \models_t \forall x \theta_1\,[\psi \leftarrow x]$. The claim follows. Suppose that the polarity of θ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then $M, \pi, i+1 \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i+1 \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$. The claim follows.

Let $\theta = \theta_1 U \theta_2$. Suppose that the polarity of θ in φ is positive. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then there exists some $j \geq i$ such that $M, \pi, j \models \theta_2\,[\psi \leftarrow \mathbf{false}]$ and forall $i \leq k < j$ we have $M, \pi, k \models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, j \models_t \forall x \theta_2\,[\psi \leftarrow x]$ and forall $i \leq k < j$ we have $M, \pi, k \models_t \forall x \theta_1\,[\psi \leftarrow x]$. Clearly, the claim follows. Suppose that the polarity of θ in φ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then either forall $j \geq i$ we have $M, \pi, j \not\models \theta_2\,[\psi \leftarrow \mathbf{false}]$ or there exists some $j \geq i$ such that $M, \pi, j \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$ and forall $i \leq k < j$ we have $M, \pi, k \not\models \theta_2\,[\psi \leftarrow \mathbf{false}]$. In the first case, from the induction assumption it follows that forall $j \geq i$ we have $M, \pi, j \models_t \forall x \neg\theta_2\,[\psi \leftarrow x]$. In this case $M, \pi, i \models_t \forall x \neg\theta\,[\psi \leftarrow x]$. In the second case, from the induction assumption it follows that $M, \pi, j \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$ and forall $i \leq k < j$ we have $M, \pi, k \models_t \forall x \neg\theta_2\,[\psi \leftarrow x]$. Again, the claim follows. □

Theorem 3.2.3 *If ψ is of pure polarity in φ then the following are equivalent.*

1. $M, \pi, i \models \varphi\,[\psi \leftarrow \bot]$
2. $M, \pi, i \models_s \forall x\, \varphi\,[\psi \leftarrow x]$
3. *for every formula ξ we have* $M, \pi, i \models \varphi\,[\psi \leftarrow \xi]$
4. $M, \pi, i \models_t \forall x\, \varphi\,[\psi \leftarrow x]$

Proof: As we have shown in Lemmas 3.1.1 and 3.1.2, trace semantics is stronger than formula semantics, and the latter is stronger than structure semantics. Since $M, \pi, i \models_s \forall x\, \varphi\,[\psi \leftarrow x]$ for all structure assignments, including $\sigma(x) = S$ and $\sigma(x) = \emptyset$, we also have 2 ⇒ 1. Thus 4 ⇒ 3 ⇒ 2 ⇒ 1. In the other direction, Lemma 3.2.1 proves that 1 ⇒ 4. □

Intuitively, the fact that a mapping can assign to a propositional variable opposite values in different positions along a trace (or states in a structure) is insignificant. Assigning the value $\bot$ is sufficient, and since the subformula is of pure polarity, $\bot$ is uniquely defined to be constant $\mathtt{true}$ or constant $\mathbf{false}$ throughout the trace. An outcome of Theorem 3.2.3 is that given a subformula ψ of pure polarity in an LTL formula φ, the following are equivalent: (1) ψ does not affect$_f$ φ in M (2) ψ does not affect$_s$ φ in M and (3) ψ does not affect$_t$ φ in M.

Chapter 4

Algorithm and Complexity

In this section we give algorithms for checking vacuity according to the different definitions. As shown in previous sections, in the case of subformulas of pure polarity, the algorithm of [KV03] works for the three, equivalent, definitions. We show that this algorithm, which replaces a subformula by either $\mathbf{true}$ or $\mathbf{false}$ (according to its polarity), cannot be applied to subformulas of mixed polarity. We then study structure and trace vacuity. The question of how to decide formula vacuity remains open.

As shown in the previous section, in the case of subformulas of pure polarity the algorithm of [KV03] applies. We show that this algorithm cannot be applied to subformulas of mixed polarity. Consider the Kripke structure M_2 in Figure 3.2 and the formula $\varphi = p \rightarrow \textsf{next}\, p$. Formula φ is of mixed polarity as the left-hand-side of the implies operator is of negative polarity, while the right-hand-side is of positive polarity (φ can also be written as $\neg p \vee \textsf{next}\, p$). Clearly, $M_2 \not\models_s \forall x \varphi\, [p \leftarrow x]$ (with the structure assignment $\sigma(x)$ including only the initial state), $M_2 \not\models_f \varphi\, [p \leftarrow q]$, and $M_2 \not\models_t \forall x \varphi\, [p \leftarrow x]$ (with the trace assignment $\alpha(x) = \{0\}$). Hence, p affects φ according to all three definitions. On the other hand, $M \models \varphi\, [p \leftarrow \mathbf{false}]$ and $M \models \varphi\, [p \leftarrow \mathbf{true}]$. We conclude that the algorithm of [KV03] cannot be applied to subformulas of mixed polarity.

We now solve trace vacuity. As mentioned, given an LTL formula φ, a model $M = \langle AP, S, S_0, R, L\rangle$ that satisfies φ, and a subformula ψ, we check whether ψ affects$_t$ φ in M by a reduction to model checking. We want to model check the UQLTL formula $\varphi' = \forall x\, \varphi\, [\psi \leftarrow x]$ on M. If $M \models_t \varphi'$ then ψ does not affect$_t$ φ. If $M \not\models_t \varphi'$ then ψ affects$_t$ φ. The algorithm presented below detects if ψ affects$_t$ φ in M.

The structure M' guesses at every step what the right assignment for the propo-

1.	Compute the polarity of ψ in φ.
2.	If ψ is of pure polarity, model check $M \models \varphi[\psi \leftarrow \bot]$.
3.	Otherwise, construct $M' = \langle AP \cup \{x\}, S \times 2^{\{x\}}, S_0 \times 2^{\{x\}}, R', L \rangle$, where for every $X_1, X_2 \subseteq 2^{\{x\}}$ and $s_1, s_2 \in S$ we have $(s_1 \times X_1, s_2 \times X_2) \in R'$ iff $(s_1, s_2) \in R$.
4.	Model check $M' \models \varphi[\psi \leftarrow x]$. If passed, report "$\psi$ does not affect$_t$ φ", otherwise report "ψ affects$_t$ φ".

Figure 4.1: Algorithm for checking if ψ affects$_t$ φ

sitional variable x is. Choosing a path in M' determines the truth values of x along the path. Formally, we have the following claim.

Claim 4.0.4 $M' \models \varphi[\psi \leftarrow x]$ *iff* $M \models_t \forall x\, \varphi[\psi \leftarrow x]$. [1]

Proof: If $M \not\models_t \forall x\, \varphi\,[\psi \leftarrow x]$, then there exists a trace $\pi = s_0, s_1, \ldots$ and a mapping α such that $M, \pi, 0, \alpha \not\models_t \varphi\,[\psi \leftarrow x]$. Let $\overline{x}_i$ be a predicate that is $\mathbf{true}$ iff $i \in \alpha(x)$. The trace $\pi' = (s_0, \overline{x}_0), (s_1, \overline{x}_1) \ldots \in \mathcal{T}(M')$ according to the construction of M'. For every $p \in AP \cup \{x\}$, the truth values of p along π and π' are identical. Thus $M' \not\models \varphi[\psi \leftarrow x]$. The other direction is similar. If $M' \not\models \varphi[\psi \leftarrow x]$, then there exists a path $\pi' = s_0, s_1, \ldots$ in M' such that $M', \pi', 0 \not\models \varphi\,[\psi \leftarrow x]$. According to the construction of M', a corresponding path π also exists in M, apart from the labeling of x. Let α assign the truth values of x along π' for the propositional variable x in M. Since $M, \pi, 0, \alpha \not\models_t \varphi[\psi \leftarrow x]$, we have $M' \models \forall x \varphi[\psi \leftarrow x]$. $\square$

We show that trace vacuity is linear in the structure and PSPACE-complete in the formula.

Theorem 4.0.5 [VW94] *Given a structure M and an LTL formula φ, we can model check φ over M in time linear in the size of M and exponential in φ and in space polylogarithmic in the size of M and quadratic in the length of φ.*

Corollary 4.0.6 *Given a structure M and an LTL formula φ such that $M \models \varphi$, we can decide whether subformula ψ affects$_t$ φ in time linear in the size of M and exponential in φ and in space polylogarithmic in the size of M and quadratic in the length of φ.*

[1] Notice that x is a propositional variable in M, but an atomic proposition in M'.

Recall that in symbolic model checking, the modified structure M' is not twice the size of M but rather includes just one additional variable. The modified formula $\varphi\,[\psi \leftarrow x]$ is at most as long as φ. The corollary follows. In order to check whether φ is trace vacuous we have to check whether there exists a subformula ψ of φ such that ψ does not affect$_t$ φ. Given a set of subformulas $\{\psi_1, \ldots, \psi_n\}$ we can check whether one of these subformulas does not affect$_t$ φ by iterating the above algorithm n times. The number of subformulas of φ is proportional to the size of φ.

Theorem 4.0.7 *Given a structure M and an LTL formula φ such that $M \models \varphi$. We can check whether φ is trace vacuous in M in time $O(|\varphi| \cdot C_M(\varphi))$ where $C_M(\varphi)$ is the complexity of model checking φ over M.*

We show now that unlike trace vacuity, there does not exist an efficient algorithm for structure vacuity. We show that deciding does not affect$_s$ is co-NP-complete in the structure. Notice, that co-NP-complete in the structure is much worse than PSPACE-complete in the formula. Indeed, the size of the formula is negligible when compared to the size of the model. Co-NP-completeness of structure vacuity renders it completely impractical.

Lemma 4.0.8 (Deciding does not affect$_s$) *For φ in LTL, a subformula ψ of φ and a structure M, the problem of deciding whether ψ does not affect$_s$ φ in M is co-NP-complete with respect to the structure M.*

Proof: We show membership in co-NP. We consider the complementary problem of deciding affect$_s$. Consider a formula φ and a structure $M = \langle AP, S, S_0, R, L\rangle$. In order to check whether ψ affects$_s$ φ we have to model check $\forall x \varphi\,[\psi \leftarrow x]$ over M. Guess a subset S' of S and set the structure assignment $\sigma(x) = S'$. Now model check the formula $\varphi\,[\psi \leftarrow x]$ over the structure $M' = \langle AP \cup \{x\}, S, S_0, R, L'\rangle$ where $L'(x) = S'$ and $L'(p) = L(p)$ for $p \neq x$.

In Appendix C we give a reduction from 3CNF satisfiability to deciding affects$_s$. Given a 3CNF formula θ, we construct a structure M_θ and a (fixed) formula φ such that $M_\theta \models \varphi$ and the proposition q affects$_s$ φ in M_θ iff θ is satisfiable. □

The complexity of deciding affects$_f$ is unclear. As shown, in the case of subformulas of pure polarity (or occurrences of subformulas) the algorithm of [KV03] is correct. We have not found either a lower bound or an upper bound for deciding affects$_f$ in the case of mixed polarity.

Part II

Regular Vacuity

Chapter 5

RELTL

5.1 Language Definition

The linear temporal logic RELTL extends LTL with a regular layer. We consider LTL in a positive normal form (see section 2.2). 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: $(e\ \mathsf{seq}\ \varphi)$ and $(e\ \mathsf{triggers}\ \varphi)$, where e is a regular expression over the alphabet $\mathcal{B}$, and φ is an RELTL formula. Intuitively, $(e\ \mathsf{seq}\ \varphi)$ asserts that some interval satisfying e is followed by a suffix satisfying φ, whereas $(e\ \mathsf{triggers}\ \varphi)$ asserts that all intervals satisfying e are followed by a suffix satisfying φ. Note that the seq and $\mathsf{triggers}$ connectives are essentially the "diamond" and "box" modalities of PDL [FL79]. Formally, let π be an infinite word over 2^{AP} then,[1]

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

[1] 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.

5.2 Automata Construction

In the automata-theoretic approach to model checking, we translate temporal logic formulas to automata [VW94]. 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 [VW94] (see also [HT99]), but we need it as a preparation for our handling of regular vacuity.

Theorem 5.2.1 *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.*

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 $\pi, i, j |\equiv L(Z)$, 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 polynomially translated to an equivalent NFW Z_e with a single initial state [HU79]. 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 $\pi, i, j |\equiv L(e)$ iff $\pi, i, j |\equiv L(Z'_e)$. 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 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 $({Z'_e}^{q_0} \ \mathsf{seq}\ \varphi)$ and $({Z'_e}^{q_0} \ \mathsf{triggers}\ \varphi)$. The closure of ψ is defined as follows: $cl(\psi) = \{\xi | \xi$ is a subformula of $\psi\} \cup \{(Z^{q'} \ \mathsf{seq}\ \xi) | (Z^q \ \mathsf{seq}\ \xi) \in cl(\psi)$ and q' is a state of $Z^q\} \cup \{(Z^{q'} \ \mathsf{triggers}\ \xi) | (Z^q \ \mathsf{triggers}\ \xi) \in cl(\psi)$ 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 $(Z^q \text{ seq } \psi)$ and $(Z^q \text{ triggers } \psi)$ 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 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, $\delta : (S \times 2^{AP}) \to 2^S$ 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 (next $\varphi_1) \in L_s$, then $\varphi_1 \in L_t$.
- If $(\varphi_1 \text{ until } \varphi_2) \in L_s$, then either $\varphi_2 \in L_s$, or $\varphi_1 \in L_s$ and $(\varphi_1 \text{ until } \varphi_2) \in L_t$.
- If $(\varphi_1 \text{ release } \varphi_2) \in L_s$, then $\varphi_2 \in L_s$ and either $\varphi_1 \in L_s$, or $(\varphi_1 \text{ release } \varphi_2) \in L_t$.

Let $Z = \langle 2^{AP}, Q, \Delta, q_0, W\rangle$ be an NFW.

- If $(Z^q \text{ seq } \psi) \in L_s$, then either (a) $q \in W$ and $\psi \in L_s$, or (b) $(Z^{q'} \text{ seq } \psi) \in L_t$ for some $q' \in \Delta(q, a)$.
- If $(Z^q \text{ triggers } \psi) \in L_s$, then (a) if $q \in W$, then $\psi \in L_s$, and (b) $(Z^{q'} \text{ triggers } \psi) \in L_t$ for all $q' \in \Delta(q, a)$.
- If $P_s = \emptyset$, then $P_t = L_t \cap seq(\varphi)$. Otherwise, for every $(Z^q \text{ seq } \psi) \in P_s$, we have that either (a) $q \in W$ and $\psi \in L_s$, or (b) $(Z^{(q')} \text{ seq } \psi) \in P_t \cap L_t$ for some $q' \in \Delta(q, a)$.

Finally, the generalized Büchi acceptance condition is used to impose the fulfillment of until and seq eventualities. Thus, $\mathcal{F} = \{\Phi_1, \ldots, \Phi_m, \Phi_{seq}\}$, where for every $(\varphi_i \text{ until } \psi_i) \in cl(\varphi)$, we have a set $\Phi_i = \{(L_s, P_s) \in S | \psi_i \in L_s$ or $(\varphi_i \text{ until } \psi_i) \notin L_s\}$, and in addition we have the set $\Phi_{seq} = \{(L_s, P_s) \in S | P_s =$

$\emptyset\}$. As in [VW94], 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 appendix A. $\square$

The exponential translation of RELTL formulas to NGBW implies a PSPACE model-checking procedure for it [VW94]. A matching lower bound is immediate from LTL being a fragment of RELTL [SC85]. Hence the following theorem.

Theorem 5.2.2 *The model-checking problem for RELTL is PSPACE-complete.*

In Section 7, we elaborate on the construction described here in order to solve the regular vacuity problem.

Chapter 6

Regular Vacuity Definition

In section 3.1 we compared alternative definitions of vacuity detection and concluded that vacuity detection with respect to trace semantics is preferable. However, we did not handle vacuity of regular expressions, and it is not clear, a priori, when a regular expression affects an RELTL formula. In this chapter we follow the semantic approach to vacuity, i.e. replace the regular expression by a universally quantified variable, but also consider two alternative definitions to regular vacuity.

6.1 A General Definition

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 the temporal logic QRELTL, which 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, $(\forall y)$ $\mathsf{globally}$ $[(y\ \mathsf{seq}\ \psi) \wedge (ab^*\ \mathsf{triggers}\ \neg\psi)]$ is a well-formed RELTL formula, while $\psi \vee [(\exists y)(y\ \mathsf{seq}\ \psi)]$ 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 quantified formula, satisfaction is checked with respect to every interval set β. For an existentially quantified formula, satisfaction is checked with respect to 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 6.1.1 *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 satisfies *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 6.1.2 *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 $\varphi =$ globally $((req \cdot \mathbf{true} \cdot \mathbf{true})$ triggers $ack)$, which states that an ack is asserted exactly three cycles after a req. When φ is satisfied in a 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 sub-regular expression. Consider the property $\varphi =$ globally $((req \cdot (\neg ack)^* \cdot ack)$ triggers $grant)$, which states that when an ack is asserted sometime after req, then $grant$ is asserted one cycle later. Regular vacuity on the sub-regular expression $((\neg ack)^* \cdot ack)$ 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.

6.2 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 6.1.2, is satisfied vacuously also with respect to the alternative definitions, but not vice versa. On the other hand, as we discuss in Section 9, 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 $\varphi = e$ triggers ψ. 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 6.1.2, 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 6.2.1 *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 9, vacuity detection for all these alternative definitions is similar.

Regular vacuity modulo expression structure Consider again the formula $\varphi = e$ triggers ψ, 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, [AFF$^+$03] 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[1].

Definition 6.2.2 *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 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.*

Note that since vacuity modulo duration/structure of expression replaces the universal quantification on all intervals by a universal quantification over a subset of them, Definitions 6.2.1 and 6.2.2 are weaker than Definition 6.1.2, in the sense that more regular expressions do not affect φ in M according to Definitions 6.2.1 and 6.2.2. Actually, these three definitions form a hierarchy: a vacuous pass w.r.t regular vacuity implies a vacuous pass w.r.t regular vacuity modulo duration, which implies a vacuous pass w.r.t regular vacuity modulo expression structure. The reverse implications do not hold. For example, suppose $(p \rightarrow \textsf{next}\, \psi)$, $(p \rightarrow \textsf{next next}\, \psi)$, $(q \rightarrow \textsf{next}\, \psi)$ and $(q \rightarrow \textsf{next next}\, \psi)$ always hold in model M. That is, ψ holds at the next two cycles after p or q. We check if $((p \vee q) \cdot (p \vee q))$ affects the formula $\varphi = ((p \vee q) \cdot (p \vee q))$ $\textsf{triggers}\ \psi$. As $M \models (\forall x)((x \vee q) \cdot (x \vee q))$ $\textsf{triggers}\ \psi$, and $M \models (\forall x)((p \vee x) \cdot (p \vee x))$ $\textsf{triggers}\ \psi$, we conclude that both p and q do not affect φ in M. Therefore $((p \vee q) \cdot (p \vee q))$ does not affect φ in M, and φ passes vacuously in M w.r.t regular vacuity modulo expression structure. On the other hand, $M \not\models (x \cdot \mathbf{true})$ $\textsf{triggers}\ \psi$ (assuming there is at least one trace in M in which ψ does not hold without being triggered by p or q). Therefore $((p \vee q) \cdot (p \vee q))$ affects φ in M, and φ passes non vacuously w.r.t regular vacuity modulo duration. It is difficult to make at this point definitive statements about the overall usability of the weaker definitions, as more industrial experience is needed.

[1]Note that Definition 6.2.2 follows our semantic approach. A syntactic approach, as the one taken in [BBER01, KV03], would result in a different definition, where Boolean functions are replaced by different Boolean functions.

Chapter 7

Algorithm and Complexity

In this chapter we study the complexity of the regular-vacuity problem. As discussed in Chapter 6, 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 chapter 4 that vacuity detection for LTL is not harder than LTL model checking and can be solved in PSPACE, and saw in chapter 5 that RELTL model checking is in PSPACE. Appendix D shows 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, 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 [VW94]. 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, $\neg(e \textsf{ seq } \varphi) = (e \textsf{ triggers } \neg\varphi)$ and $\neg(e \textsf{ triggers } \varphi) = (e \textsf{ seq } \neg\varphi)$. It is easy to see that the length of $\neg\varphi$ in positive normal form is linear in the length of φ.

Theorem 7.0.3 *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: Similarly to the proof of Theorem 5.2.1, we first translate the formula $(\exists y)\varphi$ to the 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 Chapter 5. In particular, the adjustment of Definition 6.1.1 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 Chapter 5. 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 Chapter 5. 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 $wait(L_s) = \{(Z^{q'} \text{ triggers } \xi) | (Z^q \text{ triggers } \xi) \in L_s \text{ and } q' \in \Delta(q, y)\}$. 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 $(Z^q \text{ seq } \xi) \in L_s$. Then, A_φ creates the obligation $o = ((Z^q \textsf{ seq } \xi), wait(L_s))$ 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 of obligations, and $P_s \subseteq L_s \cap (seq(\varphi) \cup obl(\varphi))$. 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 are similar to the construction is Chapter 5; thus $\mathcal{F} = \{\Phi_1, \ldots, \Phi_m, \Phi_{seq}\}$ where $\Phi_i = \{s \in S | (\varphi_1 \textsf{ until } \varphi_2), \varphi_2 \in L_s$ or $(\varphi_1 \textsf{ until } \varphi_2) \notin L_s\}$, 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. Note that some of these conditions also impose restrictions on the states.

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

   If conditions a or b hold, we say that $(Z^q \textsf{ seq } \xi)$ is *strong* in L_s w.r.t. $((L_s, P_s), a, (L_t, P_t))$.
7. If $(Z^q \textsf{ triggers } \xi) \in L_s$, then the following holds:
   (a) If $q \in W$, then $\xi \in L_s$.
   (b) $(Z^{q'} \textsf{ triggers } \xi) \in L_t$ for all $q' \in \Delta(q, a)$.

8. For every $(Z^q \textsf{ seq } \xi) \in P_s$, at least one of the following holds:
   (a) $q \in W$ and $\xi \in L_s$.
   (b) $(Z^{q'} \textsf{ seq } \xi) \in P_t \cap L_t$ for some $q' \in \Delta(q, a)$.
   (c) $\Delta(q, y) \neq \emptyset$ and $o = ((Z^q \textsf{ seq } \xi), wait(L_s)) \in P_s$. In this case we say that there is a y-transition from $(Z^q \textsf{ seq } \xi)$ to o in P_s.

   If conditions a or b hold, we say that $(Z^q \textsf{ seq } \xi)$ is strong in P_s w.r.t. $((L_s, P_s), a, (L_t, P_t))$.

9. If $o = ((Z^q \textsf{ seq } \xi), \Upsilon) \in L_s$ then at least one of the following holds:
   (a) For some $q' \in \Delta(q, y)$, we have that $(Z^{q'} \textsf{ seq } \xi) \in L_s$ and $\Upsilon \subseteq L_s$. In this case we say that there is a y-transition from o to $(Z^{q'} \textsf{ seq } \xi)$ 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 $o = ((Z^q \textsf{ seq } \xi), \Upsilon) \in P_s$ then at least one of the following holds:
    (a) For some $q' \in \Delta(q, y)$, we have that $(Z^{q'} \textsf{ seq } \xi) \in P_s$ and $\Upsilon \subseteq L_s$. In this case we say that there is a y-transition from o to $(Z^{q'} \textsf{ seq } \xi)$ 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 $P_t = L_t \cap (seq(\varphi) \cup obl)$.

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. $((L_s, P_s), a, (L_t, P_t))$.

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 Appendix B. □

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) [VW94]. 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.

Theorem 7.0.4 *The regular-vacuity problem for RELTL can be solved in EXPSPACE.*

In Section 9, 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.

Alternating Automata Alternating Büchi automata have the same expressive power as non-deterministic Büchi automata. With alternating Büchi automata, the first construction closely resembles the formula, and the automata is exponentially more succinct than a corresponding non-deterministic Büchi automata. At first, we searched for a construction based on alternating Büchi automata, but it was unclear how to synchronize relevant seq and $\mathsf{triggers}$ formulas. While in the construction above an obligation can relate seq and $triggers$ in a single state, alternating Büchi automata have separate states for the seq and $\mathsf{triggers}$ subformulas. The question whether a similar construction can be done for alternating Büchi automata remains open.

Part III

Pragmatic Aspects

Chapter 8

Subformula Vacuity in Practice

In this chapter we give some pragmatic aspects of vacuity detection. We discuss the different options for reporting vacuity. While previous works consider only giving a yes / no answer, we advocate giving the users a *simplified formula* (see below) as well so that they can best understand why the formula passes vacuously. We also check what the relation is between subformulas and occurrences of subformulas, and conclude that in order to get the most thorough vacuity detection both should be accounted for. Guided by these two observations, we show how we can achieve the most thorough vacuity detection while reducing the number of model-checker runs. Finally, we report on our experience using vacuity detection in an industrial setting. All the work in this chapter relates to trace vacuity. Therefore, we remove the subscript describing the type of affect.

8.1 Display of Results

When applying vacuity detection in an industrial setting there are two options. We can either give the user a simple yes/no answer, or we can accompany a positive answer (vacuity) with a simplified formula. Where ψ does not affect φ we supply $\varphi\,[\psi \leftarrow x]$ (or $\varphi\,[\psi \leftarrow \perp]$ where ψ is of pure polarity) as our explanation to the vacuity of φ. When we replace a subformula by a constant, we propagate the constants upwards. For example, if in subformula $\theta = \psi_1 \wedge \psi_2$ we replace ψ_1 by $\mathbf{false}$, then θ becomes $\mathbf{false}$ and we continue propagating this value above θ.

Previous works were interested only in providing a simple yes / no answer. That is, whether the property is vacuous or not. In this case it suffices to check whether the propositions affect the formula [BBER97, KV03]. Suppose that ψ

$$\text{active} := \text{en} \wedge \neg \text{in}\ ; \quad \text{rdy\_active} := \neg\ \text{rdy\_out} \wedge \neg\ \text{active}\ ;$$
$$\text{bsy\_active} := \neg\ \text{bsy\_out} \wedge \neg\ \text{active};$$
$$\text{active\_inactive} := \text{rdy\_active} \wedge \neg\ \text{bsy\_active}\ ;$$
$$\text{two\_consecutive} := G[(\text{reset} \wedge \text{active\_inactive}\ ) \rightarrow X \neg \text{active\_inactive}];$$

Figure 8.1: Vacuous pass

does not affect φ. It follows that if ψ' is a subformula of ψ then ψ' does not affect φ as well. In view of the above, in order to get a yes / no answer only the minimal subformulas of φ (i.e. the atomic propositions that appear in φ) have to be checked. In contrast, when the goal is to give the user feedback on the source of detected vacuity, it is often more useful to check non-minimal subformulas.

Consider for example the formula *two_consecutive* in Figure 8.1. This is an example of a formula that passed vacuously in one of the designs we checked. The reason for the vacuous pass is that one of the signals in *active_inactive* was set to **false** by a wrong environmental assumption. The following is the simplified formula showing that the second occurrence of *active_inactive* does not affect *two_consecutive*.

$$\text{two\_consecutive}\,[\text{active\_inactive}_2 \leftarrow \perp] = \ \textsf{globally}\ \neg(\text{reset} \wedge \text{active\_inactive})$$

From this simplified formula it is straightforward to understand what is wrong with the formula. The simplified formula associated with the occurrence of the proposition *en* under the second occurrence of *rdy_active* (after constant propagation) is as follows. Note that this occurrence of *en* occurs positively in *two_consecutive*.

$$\text{two\_consecutive}\,[\text{en}_2 \leftarrow \perp] = \ \textsf{globally}\ [(\text{reset} \wedge \text{active\_inactive}) \rightarrow X\neg(\neg\text{rdy\_out} \wedge \neg\text{bsy\_active})]$$

Clearly, this report is much less legible. This formula has very little connection to the original formula. Thus, it is preferable to check vacuity of non-minimal subformulas and subformula occurrences.

If we consider the formula as represented by a tree (rather than DAG – directed acyclic graph) then the number of leaves (propositions) is proportional to the number of nodes (subformulas). We apply our algorithm from top to bottom. We check whether the maximal subformulas affect the formula. If a subformula does not affect, there is no need to continue checking below it. If a subformula does affect, we continue and check its subformulas. In the worst case, when all the subformulas affect the formula, the number of model checker runs in order to give the most intuitive counter example is double the size of the minimal set (number of propositions). The yes / no view vs. the intuitive simplified formula view offer a clear tradeoff between minimal number of model checker runs (in the worst case) and giving the user the most helpful information. We believe that the user should be given the most comprehensive simplified formula. In our implementation we check whether **all** subformulas and occurrences of subformulas affect the formula.

8.2 Occurrences vs. Subformulas

In chapter 4 we introduced an algorithm that can determine if a subformula with multiple occurrences affects a formula. Indeed, in most cases it makes sense to check if a subformula affects a formula, as in pratice, all occurrences of the subformula will have the same truth value at a given point in time. Furthermore, sometimes an errornous behavior can only be detected when all subformula occurrences are replaced simultaneously. For example, let $\varphi = \textsf{globally}\ (p \rightarrow p)$. Intuitively, p does not affect φ since every expression (or variable) implies itself. Indeed, according to all definitions p does not affect φ, regardless of the model. However, every occurrence of p may affect φ, as both $\textsf{globally}\ p = \varphi\,[p_1 \leftarrow \bot]$ and $\textsf{globally}\ \neg p = \varphi\,[p_2 \leftarrow \bot]$ may fail (here, p_i denotes the ith occurrence of p).

On the other hand, an errournous behavior might be masked by one (or more) occurences of the subformula. Consider the formula $\varphi = p \wedge \textsf{globally}\ (q \rightarrow p)$. Assume q is always $\mathbf{false}$ in model M because of a buggy assumption. Clearly, the second occurrence of p does not affect φ in M and a vacuous trigger can be detected. However, the subformula p does affect φ in M because of the first occurrence. Every assignment that gives x the value $\mathbf{false}$ at time 0 would falsify the formula $\varphi\,[p \leftarrow x]$. Thus in order to catch the bug, we would have to check vacuity with respect to each occurrence separately. Recall the formula *two_consecutive* in Figure 8.1. The vacuous pass in this case is only with respect to occurrences and not to subformulas.

We believe that a *thorough vacuity-detection* algorithm should detect both subformulas and occurrences that do not affect the examined formula. It is up to the user to decide which vacuity alerts to ignore.

8.3 Minimizing the Number of Checks

As explained above we choose to check whether all subformulas and all occurrences of subformulas affect the formula. Applying this policy in practice may result in many runs of the model checker and may be impractical. In particular, when the formula is represented as a DAG, checking all occurrences involves turning the DAG into a tree. We show that we can reduce the number of subformulas and occurrences for which we check vacuity by analyzing the structure of the formula syntactically.

As mentioned before, if ψ' is a subformula of ψ and ψ does not affect φ then also ψ' does not affect φ. Hence, once we know that ψ does not affect φ, there is no point in checking subformulas of ψ. If ψ affects φ we have to check also the subformulas of ψ. We show that in some cases for ψ' a subformula of ψ we have ψ' affects φ iff ψ affects φ. In these cases there is no need to check direct subformulas of ψ also when ψ affects φ.

Suppose the formula φ is satisfied in M. Consider an occurrence θ_1 of the subformula $\theta = \psi_1 \wedge \psi_2$ of φ. We show that if θ_1 is of positive polarity then ψ_i affects φ iff θ_1 affects φ for $i = 1, 2$. As mentioned, θ_1 does not affect φ implies ψ_i does not affect φ for $i = 1, 2$. Suppose θ_1 affects φ. Then $M \not\models \varphi\,[\theta_1 \leftarrow \mathbf{false}]$. However, $\varphi\,[\psi_i \leftarrow \mathbf{false}] = \varphi\,[\theta_1 \leftarrow \mathbf{false}]$. It follows that $M \not\models \varphi\,[\psi_i \leftarrow \mathbf{false}]$ and that ψ_i affects φ. In the case that θ_1 is of negative (or mixed) polarity the above argument is incorrect. Consider the formula $\varphi = \neg(\psi_1 \wedge \psi_2)$ and a model where ψ_1 never holds. It is straightforward to see that $\psi_1 \wedge \psi_2$ affects φ while ψ_2 does not affect φ.

Similarly consider the subformula $\theta =$ globally ψ_1 and the occurrence θ_1 of θ of negative polarity. We show that θ_1 affects φ iff ψ_1 affects φ. Suppose θ_1 affects φ. Then $M \not\models \varphi\,[\theta_1 \leftarrow \mathbf{true}]$. As before $\varphi\,[\theta_1 \leftarrow \mathbf{true}] = \varphi\,[\psi_1 \leftarrow \mathbf{true}]$. Suppose that θ_1 is of mixed polarity and that θ_1 affects φ. Then $M \not\models \forall x \varphi\,[\theta_1 \leftarrow x]$. However, we can not prove that $M \not\models \forall x \varphi\,[\psi_1 \leftarrow x]$. This is true only if there exists a computation π of M, an assignment α such that for some $i \geq 0$ we have $\alpha(x) = \{i, \ldots\}$ and $\pi, 0, \alpha \not\models \varphi\,[\theta_1 \leftarrow x]$.

From the above discussion it follows that we can analyze the form of the formula φ syntactically and identify occurrences θ_1 such that θ_1 affects φ iff the subformulas of θ_1 affect φ. In these cases it is sufficient to model check the formula $\forall x \varphi\,[\theta_1 \leftarrow x]$. Below the immediate subformulas of θ_1 we have to continue with the same analysis. For example, if $\theta = (\psi_1 \vee \psi_2) \wedge (\psi_3 \wedge \psi_4)$ is of positive polarity and θ affects φ we can ignore $(\psi_1 \vee \psi_2)$, $(\psi_3 \wedge \psi_4)$, ψ_3, and ψ_4. We do have to check ψ_1 and ψ_2. In Table 8.1 we list the operators under which we can

Operator	Polarity	Operands
$\wedge$	+	all
$\vee$	-	all
$\neg$	pure / mixed	all
X	pure / mixed	all
U	pure	second
globally	pure	all
eventually	pure	all

Table 8.1: Operators for which checks can be avoided

apply such elimination. In the polarity column we list the polarities under which the elimination scheme applies to the operator. In the operands column we list the operands that we do not have to check. We stress that below the immediate operands we have to continue applying the analysis.

The analysis that leads to the above table is quite simple. Using a richer set of operators one must use similar reasoning to extend the table. Notice that we distinguish between pure polarity and mixed polarity. As the above table is true for occurrences, mixed polarity is only introduced in cases that the specification language includes operators with no polarity (e.g. $\oplus$, $\leftrightarrow$).

8.4 Implementation and Methodology

We implemented the above algorithms in Intel's formal verification environment. We use the language ForSpec [AFF$^+$02] with the BDD-based model checker Forecast [FKZ$^+$00] and the SAT-based bounded model checker Thunder [CFF$^+$01]. We enable the users to decide whether they want thorough vacuity detection or just to specify which subformulas / occurrences should be checked. In the case of thorough vacuity detection, for every subformula and every occurrence (according to the elimination scheme above) we create one witness formula. The vacuity algorithm amounts to model checking each of the witnesses. Both model checkers are equipped with a mechanism that allows model checking of many properties simultaneously.

The current methodology of using vacuity is applying thorough vacuity on every specification. The users prove that the property holds in the model; then, vacuity of the formula is checked. If applying thorough vacuity is not possible (due to capacity problems), the users try to identify the important subformulas and check these subformulas manually. In our experience, vacuity checks proved to be effective mostly when the pruning and assumptions used in order to enable model checking removed some important part of the model, thus rendering the specification vacuously true. However, vacuity detection also revealed RTL bugs and faulty specifications.

One area where we applied formal verification was a complex power management finite-state-machine (FSM). One set of properties verified correct transition from state to state and included assertions of the following type:

$$assert\,((state = s_i) \wedge cond) \rightarrow \ \mathsf{next}\ state = s_j$$

Vacuity detection reported that several such assertions passed vacuously and that the right-hand-side (the next state) does not affect. In one case, the vacuous pass resulted from an RTL bug which prevented the condition from happening. Therefore, there was no transition from one specific state to another. Another vacuous pass revealed a typo in one of the assumptions, which prevented the FSM from reaching some states. The validator wrote:

$$assume\,(state = s_i) \rightarrow \ \mathsf{next}\ state = (s_j \vee s_k)$$

instead of:

$$assume\,(state = s_i) \rightarrow \ \mathsf{next}\ ((state = s_j) \vee (state = s_k))$$

The erroneous code performed a bit-wise or between s_j and s_k, and as s_j was encoded as binary 111, there were no transitions from s_i to s_k.

Chapter 9

Regular Vacuity in Practice

The results in Section 7 suggest that, in practice, 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 6.2, is also not harder than standard model checking.

9.1 Specifications of Pure Polarity

Examining industrial examples shows that in practice 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 $\mathsf{triggers}$ 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 9.1.1 *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 9.1.2 *Consider a model M, RELTL formula φ, and regular expression e of pure polarity. Then, $M \models (\forall y)\varphi\,[e \leftarrow y]$ iff $M \models \varphi\,[e \leftarrow \bot]$.*

Proof: If $M \models \forall y \varphi\,[e \leftarrow y]$ then $M, \beta \models \varphi\,[e \leftarrow y]$ for every assignment β, including $\beta_\emptyset = \emptyset$ and $\beta_I = I$ (the set of all intervals). $M, \beta_\emptyset \models \varphi\,[e \leftarrow y]$ implies $M \models \varphi\,[e \leftarrow \mathbf{false}]$ since no interval satisfies $\mathbf{false}$. $M, \beta_I \models \varphi\,[e \leftarrow y]$ implies $M \models \varphi\,[e \leftarrow \mathbf{true}^*]$ since every interval satisfies $\mathbf{true}^*$. Thus $M \models \varphi\,[e \leftarrow \bot]$.

The other direction is proved by induction on the structure of φ (given in positive normal form). As regular expression are only used on the left hand side of seq and $\mathsf{triggers}$, the base case and all operators apart from seq and $\mathsf{triggers}$ are immediate.

Let $\varphi = E \,\mathsf{seq}\, \xi$ where E is a RELTL regular expression and e is a sub-regular expression of E. The polarity of e is positive in φ and therefore $\bot \equiv \mathbf{false}$. If $M, \pi, i \models \varphi\,[e \leftarrow \mathbf{false}]$ then there exists some $j \geq i$ s.t. $M, \pi, i, j \models\!\equiv E\,[e \leftarrow \mathbf{false}]$ and $M, \pi, j \models \xi$. Let $b_0, b_1, \ldots, b_{j-1-i}$ be a word in $L(E\,[e \leftarrow \mathbf{false}])$ s.t. $M, \pi, k \models b_{k-i}$ for all $i \leq k < j$. Clearly $b_{k-i} \neq \mathbf{false}$. This implies that $M, \pi, i, j \models\!\equiv E$ regardless of e. Thus $M, \pi, i \models \forall y \varphi\,[e \leftarrow y]$.

Let $\varphi = E \,\mathsf{triggers}\, \xi$ where E is a RELTL regular expression and e is a sub-regular expression of E. The polarity of e is negative in φ and therefore

$\perp \equiv \mathbf{true}^*$. Assume that $M, \pi, i \not\models \forall y E\,[e \leftarrow y]$ triggers ξ. This implies that $M, \pi, i \models \exists y E\,[e \leftarrow y]$ seq $\neg\xi$. Thus $M, \pi, i, j, \beta |\equiv E\,[e \leftarrow y]$ for some $j \geq i$ and interval set β, and $M, \pi, j \models \neg\xi$. By the definition of tight satisfaction there exists a word $w = b_0, b_1, \ldots, b_n$ over $AP \cup \{y\}$ s.t. $M, \pi, i, j, \beta |\equiv w$. Furthermore, if $b_m \in AP$, $0 \leq m \leq n$, then there exists a k s.t. $i \leq k \leq j$ and $M, \pi, k, k+1 |\equiv b_m$. Otherwise $b_m = y$ and $M, \pi, k, k' |\equiv b_m$ for some k' s.t. $i \leq k \leq k' \leq j$. We now show that there exists a word $w' \in L(E\,[e \leftarrow \mathbf{true}^*])$ s.t. $M, \pi, i, j |\equiv w'$. The word w' is equal to w except that every $b_m = y$ is replaced by $k'_m - k_m$ concatenated $\mathbf{true}$ (where $M, \pi, k_m, k'_m |\equiv b_m$). This implies that $M, \pi, i, j |\equiv E\,[e \leftarrow \mathbf{true}^*]$. Since $M, \pi, j \models \neg\xi$ we have $M, \pi, i \models E\,[e \leftarrow \mathbf{true}^*]$ seq $\neg\xi$, which implies $M, \pi, i \not\models E\,[e \leftarrow \mathbf{true}^*]$ triggers ξ. $\square$

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

9.2 Weaker Definitions of Regular Vacuity

In Section 6.2, 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 9.2.1 *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 $(\exists x)\neg\varphi$. As in chapter 4, 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 5.2.2 implies the following.

Theorem 9.2.2 *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 $\mathbf{true}$.

Chapter 10

Conclusion

In this work we investigated vacuity detection with respect to subformulas with multiple occurrences and with respect to regular expressions. We were motivated by the need to extend vacuity detection to industrial-strength property-specification languages such as ForSpec [AFF$^+$02] and Sugar [BBE$^+$01], which is significantly richer syntactically and semantically than LTL.

The generality of our framework required us to re-examine the basic intuition underlying the concept of vacuity, which until now has been defined as sensitivity with respect to syntactic perturbation. We studied sensitivity with respect to semantic perturbation, which we modeled by universal quantification. We showed that with respect to subformula vacuity, this yields a hierarchy of vacuity notions. We argued that the right notion is that of vacuity defined with respect to traces and described an algorithm for vacuity detection.

We then focused 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 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.

Finally, we discussed pragmatic aspects of vacuity detection, showed how the number of checks can be minimized, and how vacuity results should be displayed to the user. We presented examples from industrial designs where vacuity detection revealed both RTL bugs and erroneous assumptions on the environment. As for regular vacuity, 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.

Bibliography

[AFF$^+$02] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In *Proc. 8th Int'l Conf/ on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *LNCS*, pages 296–211, 2002.

[AFF$^+$03] R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M.Y. Vardi. Enhanced vacuity detection in linear temporal logic. In *Computer Aided Verification, 15th Inte.l Conf., (CAV 2003)*, volume 2725 of *LNCS*, pages 368–380, 2003.

[BB94] D. Beaty and R. Bryant. Formally verifying a microprocessor using a simulation methodology. In *Proc. 31st Design Automation Conference*, pages 596–602. IEEE Computer Society, 1994.

[BBE$^+$01] I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic Sugar. In *Proc. 13th Inte'l Conf. on Computer Aided Verification*, volume 2102 of *LNCS*, pages 363–367, 2001.

[BBER97] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. In *Proc. 9th Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 279–290, 1997.

[BBER01] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. *Formal Methods in System Design*, 18(2):141–162, 2001.

[BH99] J.P. Bergmann and M.A. Horowitz. Improving coverage analysis and test generation for large designs. In *IEEE International Conference for Computer-Aided Design*, pages 580–584, November 1999.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic* *of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[CFF+01] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking at an industrial setting. In *Computer Aided Verification, Proc. 13th International Conference*, volume 2102 of *Lecture Notes in Computer Science*, pages 436–453. Springer-Verlag, 2001.

[CGMZ95] E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32nd Design Automation Conf.*, pages 427–432, 1995.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[CKKV01] H. Chockler, O. Kupferman, R.P. Kurshan, and M.Y. Vardi. A practical approach to coverage in model checking. In *Computer Aided Verification, Proc. 13th International Conference*, volume 2102 of *Lecture Notes in Computer Science*, pages 66–78. Springer-Verlag, 2001.

[CKV01] H. Chockler, O. Kupferman, and M.Y. Vardi. Coverage metrics for temporal logic model checking. In *Tools and algorithms for the construction and analysis of systems*, number 2031 in Lecture Notes in Computer Science, pages 528 – 542. Springer-Verlag, 2001.

[DGK96] S. Devadas, A. Ghosh, and K. Keutzer. An observability-based code coverage metric for functional simulation. In *Proceedings of the International Conference on Computer-Aided Design*, pages 418–425, November 1996.

[FAD99] F. Fallah, P. Ashar, and S. Devadas. Simulation vector generation from HDL descriptions for observability enhanced-statement coverage. In *Proceedings of the 36th Design Automation Conference*, pages 666–671, June 1999.

[FKZ+00] R. Fraer, G. Kamhi, B. Ziv, M. Vardi, and L. Fix. Prioritized traversal: efficient reachability analysis for verication and falsification. In *Proc. 12th Conference on Computer Aided Verication*, volume 1855 of *Lecture Notes in Computer Science*, pages 389–402, Chicago, IL, USA, July 2000. Springer-Verlag.

[FL79] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and Systems Sciences*, 18:194–211, 1979.

[GC04a] Arie Gurfinkel and Marsha Chechik. Extending extended vacuity. In *Proceedings of FMCAD'04 (to appear)*, November 2004.

[GC04b] Arie Gurfinkel and Marsha Chechik. How vacuous is vacuous? In *Proceedings of TACAS'04*, volume 2988 of *LNCS*, pages 92–102. Springer, March 2004.

[HKHZ99] Y. Hoskote, T. Kam, P.-H Ho, and X. Zhao. Coverage estimation for symbolic model checking. In *Proc. 36th Design automation conference*, pages 300–305, 1999.

[HMA95] Y. Hoskote, D. Moundanos, and J. Abraham. Automatic extraction of the control flow machine and application to evaluating coverage of verification vectors. In *Proceedings of ICDD*, pages 532–537, October 1995.

[HT99] J.G. Henriksen and P.S. Thiagarajan. Dynamic linear time temporal logic. *Annals of Pure and Applied Logic*, 96(1–3):187–207, 1999.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[HYHD95] R. Ho, C. Yang, M. Horowitz, and D. Dill. Architecture validation for processors. In *Proceedings of the 22nd Annual Symp. on Computer Architecture*, pages 404–413, June 1995.

[KGG99] S. Katz, D. Geist, and O. Grumberg. "Have I written enough properties ?" a method of comparison between specification and implementation. In *10th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 1703 of *Lecture Notes in Computer Science*, pages 280–297. Springer-Verlag, 1999.

[Kup95] O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. In *Computer Aided Verification, Proc. 8th International Conference*, volume 939 of *Lecture Notes in Computer Science*, pages 325–338, Liege, July 1995. Springer-Verlag.

[KV99] O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. In *10th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 1703 of *Lecture Notes in Computer Science*, pages 82–96. Springer-Verlag, 1999.

[KV03] O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. *Software Tools for Technology Transfer*, 4(2):224–233, 2003.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, Brooklyn, June 1985. Springer-Verlag.

[MAH98] D. Moumdanos, J.A. Abraham, and Y.V. Hoskote. Abstraction techniques for validation coverage analysis and test generation. *IEEE Trans. on Computers*, January 1998.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundation of Computer Science*, pages 46–57, 1977.

[PP95] B. Plessier and C. Pixley. Formal verification of a commercial serial bus interface. In *Proc. of 14th Annual IEEE International Phoenix Conference on Computers and Communications*, pages 378–382, March 1995.

[PS02] M. Purandare and F. Somenzi. Vacuum cleaning CTL formulae. In *Proc. 14th Conf. on Computer Aided Verification*, volume 2404 of *LNCS*, pages 485–499, 2002.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1981.

[SC85] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *J. ACM*, 32:733–749, 1985.

[SveB84] M. Savelsberg and P. van emde Boas. Bounded tiling, an alternative to satisfiability. In *2nd Frege conference*, pages 354–363. Akademya Verlag, 1984.

[SVW85] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *Proc. 10th International Colloquium on Automata, Languages and Programming*, volume 194, pages 465–474, Nafplion, July 1985. Lecture Notes in Computer Science, Springer-Verlag.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

Part IV

Appendixes

Appendix A

The Correctness of the Construction for ALTL

Theorem A.0.3 *Let φ be an ALTL formula and let A_φ be its automaton. Then, $L(A_\varphi) = L(\varphi)$.*

First Direction: $L(\varphi) \subseteq L(A_\varphi)$

Definition A.0.4 *Let π be an infinite word, i an index, and $(Z^q \,\mathsf{seq}\, \xi)$ an ALTL formula, s.t. $\pi, i \models (Z^q \,\mathsf{seq}\, \xi)$. We define the* minimum satisfying index *of $\pi, i, (Z^q \,\mathsf{seq}\, \xi)$ denoted $msi(\pi, i, (Z^q \,\mathsf{seq}\, \xi))$ as the minimal index $j \geq i$ such that $\pi, i, j \models\!\!\equiv Z^q$ and $\pi, j \models \xi$.*

The minimal satisfying index determines the first index where the seq formula could be released from its obligation.

Lemma A.0.5 *Let π be in $L(\varphi)$, then $\pi \in L(A_\varphi)$.*

Proof: We construct a fair run $\rho = (L_0, P_0), (L_1, P_1), \ldots$ of A_φ on π. For every $i \geq 0$ we define L_i to be a subset of $cl(\varphi)$ s.t. a subformula φ' of $cl(\varphi)$ is in L_i iff $\pi, i, \models \varphi'$. We define P_i to be a subset of $seq(\varphi) \cap L_i$. The subsets P_i are inductively defined. For $i = 0$, $P_0 = \emptyset$. For $i > 0$ we distinguish between two cases:

1. If $P_{i-1} = \emptyset$, then $P_i = L_i \cap seq(\varphi)$.

2. Otherwise, P_i contains a formula $(Z^{q'}$ seq $\xi)$ iff it is in L_i and there exists a formula $(Z^q$ seq $\xi)$ in P_{i-1} s.t. $msi(\pi, i-1, (Z^q \text{ seq } \xi)) = msi(\pi, i, (Z^{q'} \text{ seq } \xi)) \geq i$ and q' is in $\Delta(q, \pi_{i-1})$.

We need to prove that ρ is a fair run of A_φ on π. Since $\pi, 0 \models \varphi$, we have that L_0 contains φ. The definition of ρ implies that $P_0 = \emptyset$, thus, (L_0, P_0) is an initial state. The following two propositions complete the proof of Lemma A.0.5.

Proposition A.0.6 *For every i, we have that (L_i, P_i) is in $\delta((L_{i-1}, P_{i-1}), \pi_i)$.*

Proof: We show that all the conditions of δ are satisfied:

1. For all $p \in AP$, if $p \in L_{i-1}$, then $p \in \pi_{i-1}$.
2. For all $p \in AP$, if $\neg p \in L_{i-1}$ then $p \notin \pi_{i-1}$
3. If (next $\varphi_1) \in L_{i-1}$, then since $\pi, i-1 \models ($ next $\varphi_1)$ we have that $\pi, i \models \varphi_1$ and thus $\varphi_1 \in L_i$.
4. If $(\varphi_1$ until $\varphi_2) \in L_{i-1}$ then either $\pi, i-1 \models \varphi_2$, in which case $\varphi_2 \in L_{i-1}$, or $\pi, i-1 \models \varphi_1$ and $\pi, i \models (\varphi_1$ until $\varphi_2)$ in which case $\varphi_1 \in L_{i-1}$ and $(\varphi_1$ until $\varphi_2) \in L_i$.
5. If $(\varphi_1$ release $\varphi_2) \in L_{i-1}$ then $\pi, i-1 \models \varphi_2$ and thus $\varphi_2 \in L_{i-1}$ and either $\pi, i-1 \models \varphi_1$ in which case $\varphi_1 \in L_{i-1}$, or $\pi, i \models (\varphi_1$ release $\varphi_2)$ in which case $(\varphi_1$ release $\varphi_2) \in L_i$.
6. If $(Z^q$ seq $\xi) \in L_{i-1}$, then we distinguish between two cases:
   (a) If $\epsilon \in L(Z^q)$ and $\pi, i-1 \models \xi$, then $q \in W$ and ξ is in L_{i-1}.
   (b) Otherwise, $j = msi(\pi, i-1, Z^q \text{ seq } \xi) \geq i$. Since there exists an accepting run of Z^q on $\pi_{i-1}, \pi_i, \ldots, \pi_{j-1}$ and $\pi, j \models \xi$, for the second state q' of the run, we have that $q' \in \Delta(q, \pi_{i-1})$ and $\pi_i, \pi_{i+1}, \ldots, \pi_{j-1}$ is in $L(Z^{q'})$. This implies that $\pi, i \models (Z^{q'}$ seq $\xi)$ and that $msi(\pi, i, Z^{q'} \text{ seq } \xi) = j$. Thus, $(Z^{q'}$ seq $\xi)$ in L_i.
7. If $(Z^q$ triggers $\xi) \in L_{i-1}$ then the following hold:
   (a) If $q \in W$, then $\epsilon \in L(Z^q)$. As $(Z^q$ triggers $\xi) \in L_{i-1}$, then $\pi, i-1 \models (Z^q$ triggers $\xi)$ and thus $\pi, i-1 \models \xi$. This implies that ξ is in L_{i-1}.

(b) For every $q' \in \Delta(q, \pi_{i-1})$ we have that for every $j \geq i$, if $\pi_i, \pi_{i+1}, \ldots \pi_{j-1}$ is in $L(Z^{q'})$, then $\pi_{i-1}, \pi_i, \ldots \pi_{j-1}$ is in $L(Z^q)$, and $\pi, j \models \xi$. Thus $\pi, i \models (Z^{q'}\ \textsf{triggers}\ \xi)$. This implies that $(Z^{q'}\ \textsf{triggers}\ \xi)$ is in L_i for all $q' \in \Delta(q, a)$.

8. If $P_{i-1} = \emptyset$, then $P_i = L_i \cap seq(\varphi)$. Otherwise, for every $(Z^q\ \textsf{seq}\ \xi) \in P_{i-1}$, we distinguish between two cases

   (a) If $\epsilon \in L(Z^q)$ and $\pi, i-1 \models \xi$, then $q \in W$ and ξ is in L_{i-1}.

   (b) Otherwise, $msi(\pi, i-1, Z^q\ \textsf{seq}\ \xi) \geq i$. Since $P_{i-1} \subseteq L_{i-1}$, the formula $(Z^q\ \textsf{seq}\ \xi)$ is in L_{i-1}. By item 6b there exists a formula $(Z^{q'}\ \textsf{seq}\ \xi)$ in L_i s.t. $q' \in \Delta(q, \pi_{i-1})$ and $msi(\pi, i, Z^{q'}\ \textsf{seq}\ \xi) = msi(\pi, i-1, Z^q\ \textsf{seq}\ \xi)$. This implies that $(Z^{q'}\ \textsf{seq}\ \xi)$ is in P_i.

$\square$

Proposition A.0.7 *ρ is a fair run of A.*

Proof: First, we prove that for every $i \leq m$ we have that $inf(\rho) \cap \Phi_i \neq \emptyset$. We prove that for every $j \geq 0$ there exists $k \geq j$ s.t. $\rho_k \in \Phi_i$. Let $\varphi_1\ \textsf{until}\ \varphi_2$ be the until formula which corresponds to Φ_i, we distinguish between two cases:

1. If $\pi, j \not\models (\varphi_1\ \textsf{until}\ \varphi_2)$ then $(\varphi_1\ \textsf{until}\ \varphi_2)$ is not in L_j, thus $\rho_j \in \Phi_i$.

2. Otherwise, $\pi, j \models (\varphi_1\ \textsf{until}\ \varphi_2)$, thus there exists $k \geq j$ s.t. $\pi, k \models \varphi_2$. This implies that φ_2 is in L_k, thus, $\rho_k \in \Phi_i$.

Next, we prove that $inf(\rho) \cap \Phi_{seq} \neq \emptyset$. We prove that for every $j \geq 0$ there exists $k \geq j$ such that $\rho_k \in \Phi_{seq}$. We distinguish between two cases:

1. If $P_j = \emptyset$, then $\rho_j \in \Phi_{seq}$.

2. Otherwise, P_j contains some $\textsf{seq}$ formulas. Let $i = max_{\varphi' \in P_j} msi(\pi, j, \varphi')$. We prove by induction that for every $j \leq k < i$, we have $max_{\varphi' \in P_k} msi(\pi, k, \varphi') = i$.

   - The base case $k = j$ is trivial.

- Assume that $max_{\varphi' \in P_{k-1}} msi(\pi, k-1, \varphi') = i$. Let $(Z^q \textsf{ seq } \xi)$ be a formula in P_{k-1} s.t. $msi(\pi, k-1, (Z^q \textsf{ seq } \xi)) = i$. Since $i > k$, there exists a run $q, q_1, q_2, \ldots q_{i-k-1}$ of length > 1 of Z^q on $\pi_{k-1}, \pi_k, \ldots \pi_{i-1}$, and $\pi, i \models \xi$. This implies that the formula $(Z^{q_1} \textsf{ seq } \xi)$ is in L_k. Since $q, q_1, q_2, \ldots q_{i-k-1}$ is the shortest accepting run of Z^q on a prefix of π^{k-1}, we have $msi(\pi, k, (Z^{q_1} \textsf{ seq } \xi)) = msi(\pi, k-1, Z^q \textsf{ seq } \xi) = i$, and that $q_1 \in \Delta(q, \pi_{k-1})$, thus $(Z^{q_1} \textsf{ seq } \xi) \in P_k$. In addition, for every other formula $\varphi' \in P_k$ there exists a formula φ'' in P_{k-1} s.t. $msi(\pi, k, \varphi') = msi(\pi, k-1, \varphi'') \leq i$. This implies that $max_{\varphi' \in P_k} msi(\pi, k, \varphi') = i$.

Thus, for every formula $(Z^q \textsf{ seq } \xi)$ in P_{i-1}, we have that $msi(\pi, i-1, (Z^q \textsf{ seq } \xi)) = i$. This implies that for every formula $(Z^q \textsf{ seq } \xi)$ in P_{i-1}, we have that $q \in W$ and $\pi, i-1 \models \xi$. This implies that $P_i = \emptyset$, thus $\rho_i \in \Phi_{seq}$. □

Second Direction: $L(A_\varphi) \subseteq L(\varphi)$

Lemma A.0.8 *Let π be in $L(A_\varphi)$, then $\pi \in L(\varphi)$.*

Before we prove the lemma we present a few propositions.

Proposition A.0.9 *Let $\rho = (L_0, P_0,), (L_1, P_1), \ldots$ be a run of A_φ on π. Let $i \geq 0$ be an index s.t. the formula $(\varphi_1 \textsf{ until } \varphi_2)$ is in L_i. Then either for every $j \geq i$, we have $\{(\varphi_1 \textsf{ until } \varphi_2), \varphi_1\} \subseteq L_j$, or there exists $k \geq i$ s.t. $\varphi_2 \in L_k$ and for every $i \leq j < k$, we have $\varphi_1 \in L_j$.*

Proof: We prove with induction on $k \geq i$ that either there exists an index $k' \leq k$ s.t. $\varphi_2 \in L'_k$ and for every $i \leq j < k'$, $\varphi_1 \in L_j$, or for every $i \leq j \leq k$, $\{(\varphi_1 \textsf{ until } \varphi_2), \varphi_1\} \subseteq L_j$.

- Base case: $k = i$ follows trivially from the definition of δ.
- Induction step: assume that the proposition holds for $k-1$ we distinguish between two cases:
    1. If there exists $k' \leq k-1$ s.t. $\varphi_2 \in L'_k$ and for every $i \leq j < k'$, $\varphi_1 \in L_j$, then the lemma holds trivially.
    2. Otherwise, for every $i \leq j \leq k-1$, $\{(\varphi_1 \textsf{ until } \varphi_2), \varphi_1\} \subseteq L_j$. Since $\varphi_2 \notin L_{k-1}$, the definition of δ implies that either $\varphi_2 \in L_k$ or $\{(\varphi_1 \textsf{ until } \varphi_2), \varphi_1\} \subseteq L_k$, in both cases the induction holds for k.

$\square$

Proposition A.0.10 *Let $\rho = (L_0, P_0,), (L_1, P_1), \ldots$ be a run of A_φ on π. Let $i \geq 0$ be an index s.t. the formula $(\varphi_1 \textsf{ release } \varphi_2)$ is in L_i. Then either for every $j \geq i$, we have $\{(\varphi_1 \textsf{ release } \varphi_2), \varphi_2\} \subseteq L_j$, or there exists $k \geq i$ s.t. $\varphi_1 \in L_k$ and for every $i \leq j \leq k$, we have $\varphi_2 \in L_j$.*

Proof: We prove with induction on $k \geq i$ that either there exists an index $k' \leq k$ s.t. $\varphi_1 \in L'_k$ and for every $i \leq j \leq k'$, $\varphi_2 \in L_j$, or for every $i \leq j \leq k$, $\{(\varphi_1 \textsf{ release } \varphi_2), \varphi_2\} \subseteq L_j$.

- Base case: $k = i$ follows trivially from the definition of δ.
- Induction step: assume that the proposition holds for $k - 1$ we distinguish between two cases:
  1. If there exists $k' \leq k - 1$ s.t. $\varphi_1 \in L'_k$ and for every $i \leq j \leq k'$, $\varphi_2 \in L_j$, then the lemma holds trivially.
  2. Otherwise, for every $i \leq j \leq k - 1$, $\{(\varphi_1 \textsf{ release } \varphi_2), \varphi_2\} \subseteq L_j$. Since $\varphi_1 \notin L_{k-1}$, the definition of δ implies that $\{(\varphi_1 \textsf{ until } \varphi_2), \varphi_2\} \subseteq L_k$ thus, the induction holds for k.

$\square$

Proposition A.0.11 *Let $\rho = (L_0, P_0,), (L_1, P_1), \ldots$ be a run of A_φ on π. Let $i \geq 0$ be an index s.t. the formula $(Z^q \textsf{ seq } \xi)$ is in L_i. Then one of the following holds:*

1. *There exists $k \geq i$ s.t. there exists an accepting run $q, q_1, q_2, \ldots q_{k-i}$ of Z^q over $\pi_i, \pi_{i+1}, \ldots, \pi_{k-1}$ s.t. for every $i \leq j \leq k$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in L_j$, and $\xi \in L_k$.*
2. *There exists an infinite run $q, q_1, q_2 \ldots$ of Z^q over $\pi_i, \pi_{i+1}, \ldots$ s.t. for every $j > i$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in L_j$, and either $q_{j-i} \notin W$ or $\xi \notin L_j$.*

Proof: We prove with induction on $k \leq i$ that one of the following conditions holds:

1. There exists an index $i \leq k' \leq k$ s.t. there exists an accepting run $q, q_1, q_2, \ldots q_{k'-i}$ of Z^q over $\pi_i, \pi_{i+1}, \ldots, \pi_{k'-1}$ s.t. for every $i \leq j \leq k'$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in L_j$, and $\xi \in L_{k'}$.

2. There exists a run $q, q_1, q_2 \dots q_{k-i}$ of Z^q over $\pi_i, \pi_{i+1}, \dots \pi_{k-1}$ s.t. for every $i \leq j \leq k$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in L_j$, and either $q_{j-i} \notin W$ or $\xi \notin L_j$.

- Base case: $k = i$ follows trivially from the definition of δ.
- Induction step: assume that the proposition holds for $k - 1$ we distinguish between two cases:
  1. There exists an index $i \leq k' \leq k - 1$ s.t. there exists an accepting run $q, q_1, q_2, \dots q_{k'-i}$ of Z^q over $\pi_i, \pi_{i+1}, \dots, \pi_{k'-1}$ s.t. for every $i \leq j \leq k'$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in L_j$, and $\xi \in L_{k'}$. Then the lemma holds trivially.
  2. Otherwise, there exists a run $q, q_1, q_2 \dots q_{k-i-1}$ of Z^q over $\pi_i, \pi_{i+1}, \dots \pi_{k-2}$ s.t. for every $i \leq j \leq k - 1$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in L_j$, and either $q_{j-i} \notin W$ or $\xi \notin L_j$. If $q_{k-i} \in W$ and $\xi \in L_k$, then there exists an accepting run $q, q_1, q_2, \dots q_{k-i}$ of Z^q over $\pi_i, \pi_{i+1}, \dots, \pi_{k-1}$ and the lemma holds

     Otherwise, the definition of δ implies that L_k contains a formula $(Z^{q_{k-i}} \textsf{ seq } \xi)$ s.t. $q_{k-i} \in \Delta(q_{k-i-1}, \pi_{k-1})$, thus the lemma holds.

$\square$

Proposition A.0.12 *Let $\rho = (L_0, P_0,), (L_1, P_1), \dots$ be a run of A_φ on π. Let $i \geq 0$ be an index s.t. the formula $(Z^q \textsf{ seq } \xi)$ is in P_i. Then one of the following holds:*

1. *There exists $k \geq i$ s.t. there exists an accepting run $q, q_1, q_2, \dots q_{k-i}$ of Z^q over $\pi_i, \pi_{i+1}, \dots, \pi_{k-1}$ s.t. for every $i \leq j \leq k$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in P_j$, and $\xi \in L_k$.*
2. *There exists an infinite run $q, q_1, q_2 \dots$ of Z^q over $\pi_i, \pi_{i+1}, \dots$ s.t. for every $j \geq i$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in P_j$, and either $q_{j-i} \notin W$ or $\xi \notin L_j$.*

The proof of this proposition is similar to the proof of Proposition A.0.11, and thus omitted.

Proposition A.0.13 *Let $\rho = (L_0, P_0,), (L_1, P_1), \dots$ be a run of A_φ on π. Let $i \geq 0$ be an index s.t. the formula $(Z^q \textsf{ triggers } \xi)$ is in L_i. Then for every $j \geq i$ and every run $q, q_1, q_2, \dots q_{j-i}$ of $Z(q)$ on $\pi_i, \pi_{i+1}, \dots, \pi_{j-1}$, we have that the formula $(Z^{q_{j-i}} \textsf{ triggers } \xi)$ is in L_j. Furthermore, if $q_{j-i} \in W$, then $\xi \in L_j$.*

Proof: We prove the proposition by induction on $j \geq i$.

- Base case: $j = i$ follows directly from the definition of δ.

- Assume that the proposition holds for j we prove it for $j+1$. Let $q, q_1, q_2, \ldots q_{j-i+1}$ of $Z(q)$ on $\pi_i, \pi_{i+1}, \ldots, \pi_j$. The induction hypothesis implies that the formula $(Z^{q_{j-i}}$ triggers $\xi)$ is in L_j. Since $q_{j-i+1} \in \Delta(q_{j-i}, \pi_j)$, δ implies that the formula $(Z^{q_{j-i+1}}$ triggers $\xi)$ is in L_{j+1}. If $q_{j-i+1} \in W$, then δ implies that $\xi \in L_{j+1}$.

$\square$

We now prove Lemma A.0.8. Let $\rho = (L_0, P_0,), (L_1, P_1), \ldots$ be a fair run of A_φ on π. We prove with induction over the structure of φ that for every $i \geq 0$ and every subformula φ' we have that φ' in L_i iff $\pi, i \models \varphi'$. Since for every initial state (L, P) of A_φ, we have $\varphi \in L$, we have that $\pi, 0 \models \varphi$.

- Base case: For p and $\neg p$ the definition of the automaton implies that the lemma holds.

- Induction step: Assume that the lemma holds for φ_1, φ_2, and ξ.

  – The consistency of the states of the automaton implies that the lemma holds for the formulas $(\varphi_1 \wedge \varphi_2)$ and $(\varphi_1 \vee \varphi_2)$.

  – Let (next $\varphi_1)$ be a formula in L_i. The definition of δ implies that $\varphi_1 \in L_{i+1}$. The induction hypothesis implies that $\pi, i+1 \models \varphi_1$, thus $\pi, i \models$ (next $\varphi_1)$.

  – Let $(\varphi_1$ until $\varphi_2)$ be a formula in L_i. Proposition A.0.9 implies that either there exists $k \geq i$ s.t. $\varphi_2 \in L_k$ and for every $i \leq j < k$, we have $\varphi_1 \in L_j$ or for every $j \geq i$, we have $\{(\varphi_1$ until $\varphi_2), \varphi_1\} \subseteq L_j$.

    In the first case, the induction hypothesis implies that for all $i \leq j < k$, we have $\pi, j \models \varphi_1$ and that $\pi, k \models \varphi_2$. Thus $\pi, i \models (\varphi_1$ until $\varphi_2)$.

    As for the second case let Φ_l be the fairness set which corresponds to $(\varphi_1$ until $\varphi_2)$. Since ρ is fair, there exists $k \geq i$ such that $\rho_k \in \Phi_l$. Given that $(\varphi_1$ until $\varphi_2) \in L_k$, $\rho_k \in \Phi_l$ implies that $\varphi_2 \in L_k$. Thus, the induction hypothesis implies that for all $i \leq j < k$, we have $\pi, j \models \varphi_1$ and that $\pi, k \models \varphi_2$. Thus $\pi, i \models (\varphi_1$ until $\varphi_2)$.

  – Let $(\varphi_1$ release $\varphi_2)$ be a formula in L_i. Proposition A.0.10 implies that either for every $j \geq i$, we have $\{(\varphi_1$ release $\varphi_2), \varphi_2\} \subseteq L_j$, or there exists $k \geq i$ s.t. $\varphi_1 \in L_k$ and for every $i \leq j \leq k$, we have $\varphi_2 \in L_j$.

The induction assumption implies that either for all $j \geq i$, we have $\pi, j \models \varphi_2$, or there or there exists $k \geq i$ s.t. for all $i \leq j \leq k$, we have $\pi, j \models \varphi_2$ and $\pi, k \models \varphi_1$. Thus, $\pi, i \models (\varphi_1 \textsf{ release } \varphi_2)$.

– Let $(Z^q \textsf{ seq } \xi)$ be a formula in L_i. Proposition A.0.11 implies that one of the following holds

1. There exists $k \geq i$ s.t. there exists an accepting run $q, q_1, q_2, \ldots q_{k-i}$ of Z^q over $\pi_i, \pi_{i+1}, \ldots, \pi_{k-1}$ s.t. for every $i \leq j \leq k$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in L_j$, and $\xi \in L_k$. In this case the induction hypothesis implies that $\pi, k' \models \xi$ thus $\pi, i \models (Z^q \textsf{ seq } \xi)$.
2. There exists an infinite run $q, q_1, q_2 \ldots$ of Z^q over $\pi_i, \pi_{i+1}, \ldots$ s.t. for every $j \geq i$, we have $(Z^{q_{j-i}} \textsf{ seq } \xi) \in L_j$, and either $q_{j-i} \notin W$ or $\xi \notin L_j$.

   Since ρ is fair there exists $k \geq i$ s.t. $P_k = \emptyset$. This implies that $(Z^{q_{k+1-i}} \textsf{ seq } \xi) \in P_{k+1}$. By proposition A.0.12, one of the following should hold:

   (a) There exists $k' \geq k+1$ s.t. there exists an accepting run $q_{k+1-i}, q_{k+2-i}, \ldots q_{k'-i}$ of $Z^{q_{k+1-i}}$ over $\pi_{k+1}, \pi_{k+2}, \ldots, \pi_{k'-1}$ s.t. for every $k+1 \leq j \leq k'$, we have $(Z^{q_{j-k-1}} \textsf{ seq } \xi) \in P_j$, and $\xi \in L_{k'}$. In this case the induction hypothesis implies that $\pi, k \models \xi$. Since the run $q, q_1, q_2, \ldots, q_{k-i}, q_{k+1-i}, \ldots q_{k'-i}$ is accepting, we have $\pi, i \models (Z^q \textsf{ seq } \xi)$.

   (b) There exists an infinite run $q_{k+1-i}, q_{k+2-i}, \ldots$ of Z^q over $\pi_{k+1}, \pi_{k+2}, \ldots$ s.t. for every $j > k+1$, we have $(Z^{q_{j-k-1}} \textsf{ seq } \xi) \in P_j$, and either $q_{j-k} \notin W$ or $\xi \notin L_j$. In this case P_j is empty only finitely many time in ρ, thus ρ is not fair, contradiction.

– Let $(Z^q \textsf{ triggers } \xi)$ be a formula in L_i. Proposition A.0.13 implies that for every $j \geq i$, if Z^q has an accepting run over $\pi_i, \pi_{i+1}, \ldots \pi_{j-1}$, then $\xi \in L_j$. In this case the induction hypothesis implies that $\pi, j \models \xi$. This implies that $\pi, i \models (Z^q \textsf{ triggers } \xi)$.

$\square$

Appendix B

The Correctness of the Construction for QALTL

Theorem B.0.14 $L(A_\varphi) = L((\exists y)\varphi)$.

First Direction: $L((\exists y\varphi)) \subseteq L(A_\varphi)$

We start by extending the definition of msi to obligations. We say that an obligation $o = ((Z^q \textsf{ seq } \xi), \Upsilon)$ *is possible* for π, i, β if there exists an index $j \geq i$ s.t. $\pi, j, \beta \models \Upsilon$ and for some $q' \in \Delta(q, y)$, we have $\pi, j, \beta \models (Z^{q'} \textsf{ seq } \xi)$.

Definition B.0.15 *Let π be a word, i an index, β an interval set, and $(Z^q \textsf{ seq } \xi)$ a formula in $seq(\varphi)$ s.t. $\pi, i, \beta \models (Z^q \textsf{ seq } \xi)$. Then $msi(\pi, i, \beta, (Z^q \textsf{ seq } \xi)) = min(\{j | \pi, i, j, \beta | \equiv L(Z^q) \wedge \pi, j, \beta \models \xi\})$. Let $o = ((Z^q \textsf{ seq } \xi), \Upsilon)$ be an obligation that is possible for π, i, β. We define $msi(\pi, i, \beta, o) = min(\{j | \exists q' \in \Delta(q, y) \exists k \geq i$ s.t. $\pi, k, \beta \models \Upsilon \wedge j = msi(\pi, k, (Z^{q'} \textsf{ seq } \xi))\})$.*

We now present a lemma, which defines the conditions for tight satisfaction of $L(Z^q)$ in terms of states of Z^q.

Lemma B.0.16 *Let Z^q be NFW, let π be an infinite word, let $j \geq i \geq 0$, and let $\beta \subseteq I$ be an interval assignment. Then $\pi, i, j, \beta | \equiv L(Z^q)$ iff at least one of the following holds:*

1. *$j = i$ and $q \in W$.*
2. *$j > i$ and there exists $q' \in \Delta(q, \pi_i)$ s.t. $\pi, i+1, j, \beta | \equiv L(Z^{q'})$.*

3. *There exists some* k, $i \leq k \leq j$ *and a state* $q' \in \Delta(q, y)$ *s.t.* $(i, k) \in \beta$ *and* $\pi, k, j, \beta \models L(Z^{q'})$.

Proof: $\pi, i, j, \beta \models L(Z^q)$ iff there is $w \in L(Z^q)$, $w = x_0, x_1, \ldots, x_n$ and there is a sequence of integers $i_0, i_1, \ldots, i_n, i_{n+1}$, such that $i_0 = i$ and $i_{n+1} = j$. Moreover, for every $0 \leq k \leq n$ the following conditions hold:

- If $x_k \in 2^{AP}$ then $x_k = \pi_{i_k}$ and $i_{k+1} = i_k + 1$.
- If $x_k = y$ then $(i_k, i_{k+1}) \in \beta$.

We partition this condition into three cases:

1. The case where $w = \epsilon$. In this case $j = i$. Since $w = \epsilon \in L(Z^q)$, we have $q \in W$. Thus the first condition of the lemma holds.

2. The case where $|w| > 0$ and $x_0 \in 2^{AP}$. In this case $x_0 = \pi_i$ thus, for some $q' \in \Delta(q, \pi_i)$ we have that $x_1, x_2, \ldots x_n \in L(Z^{q'})$. Furthermore, for every $1 \leq k \leq n$ we have that the following conditions hold:

   - If $x_k \in 2^{AP}$ then $x_k = \pi_{i_k}$ and $i_{k+1} = i_k + 1$.
   - If $x_k = y$ then $(i_k, i_{k+1}) \in \beta$.

   Thus, $\pi, i + 1, j, \beta \models L(Z^{q'})$ and the second condition holds.

3. The case where $|w| > 0$ and $x_0 = y$. For $k = i_1$, we have that $(i, k) \in \beta$, and for some $q' \in \Delta(q, y)$, we have $x_1, x_2, \ldots x_n \in L(Z^{q'})$. Furthermore, for the sequence $i_1, i_2, \ldots, i_{n+1}$ and for every $1 \leq l \leq n$ we have that the following conditions hold:

   - If $x_l \in 2^{AP}$ then $x_l = \pi_{i_l}$ and $i_{l+1} = i_l + 1$.
   - If $x_l = y$ then $(i_l, i_{l+1}) \in \beta$.

   Thus $\pi, k, j, \beta \models L(Z^{q'})$ and the third condition holds. $\square$

Lemma B.0.17 *Let* π *be in* $L((\exists y)\varphi)$*, then* $\pi \in L(A_\varphi)$.

Proof: We construct a fair run ρ of A_φ on π. Let β be an interval set such that $\pi, 0, \beta \models \varphi$. For every $i \geq 0$ we define $L_i = \{\varphi' | \pi, i, \beta \models \varphi'\} \cup \{o | o$ is possible for $\pi, i, \beta\}$. We define P_i to be a subset of L_i. The subsets P_i are inductively defined. For $i = 0$, $P_0 = \emptyset$. For $i > 0$ we distinguish between two cases:

1. If $P_{i-1} = \emptyset$, then $P_i = L_i \cap (seq(\varphi) \cup obl(\varphi))$.

2. Otherwise, P_i contains a formula $(Z^{q'}\ \mathsf{seq}\ \xi)$ iff it is in L_i and $msi(\pi, i, \beta, (Z^{q'}\ \mathsf{seq}\ \xi)) \leq max\{msi(\pi, i-1, \beta, x) | x \in P_{i-1}\}$. P_i contains an obligation formula $o = ((Z^{q}\ \mathsf{seq}\ \xi), \Upsilon)$ iff it is in L_i and $msi(\pi, i, \beta, o) \leq max\{msi(\pi, i-1, \beta, x) | x \in P_{i-1}\}$.

The following two propositions complete the proof of Lemma B.0.17.

Proposition B.0.18 *For every i, we have that (L_i, P_i) is in $\delta((L_{i-1}, P_{i-1}), \pi_{i-1})$.*

Proof: We need to show that all the conditions for the transition relation are fulfilled. The conditions for p, $\neg p$, $\wedge$, $\vee$, until , $\mathsf{release}$, and $\mathsf{triggers}$ are identical to the condition for the automaton defined in the Section 5, and thus, can be proved similarly to Proposition A.0.6. Next, we prove that the other conditions hold as well.

1. If $(Z^{q}\ \mathsf{seq}\ \xi) \in L_{i-1}$, then for some $j \geq i-1$, we have that $\pi, i-1, j, \beta \models\!\equiv L(Z^{q})$ and $\pi, j, \beta \models \xi$. Lemma B.0.16 implies that at least one of the following holds:

   (a) $j = i-1$ and $q \in W$. In this case condition $6a$ of δ is satisfied.

   (b) $j \geq i$ and there exists $q' \in \Delta(q, \pi_{i-1})$ s.t. $\pi, i, j, \beta \models\!\equiv L(Z^{q'})$. This implies that $\pi, i, \beta \models (Z^{q'}\ \mathsf{seq}\ \xi)$, thus, $(Z^{q'}\ \mathsf{seq}\ \xi) \in L_i$, and condition $6b$ of δ is satisfied.

   (c) There exists an index $i-1 \leq k \leq j$ and a state $q' \in \Delta(q, y)$ s.t. $(i, k) \in \beta$ and $\pi, k, j, \beta \models\!\equiv L(Z^{q'})$. Since $i-1 \leq k$ and $\pi, k, \beta \models \xi$, we have that $o = ((Z^{q}\ \mathsf{seq}\ \xi), wait(L_{i-1}))$ is possible in $\pi, i-1, \beta$. Thus $o \in L_{i-1}$ and condition $6c$ of δ is satisfied.

2. If $(Z^{q}\ \mathsf{seq}\ \xi) \in P_{i-1}$, then it is also in L_i. Let $j \geq i-1$ be the minimal index s.t. $\pi, i-1, j, \beta \models\!\equiv L(Z^{q})$ and $\pi, j, \beta \models \xi$. Lemma B.0.16 implies at least one of the following holds:

   (a) $j = i-1$ and $q \in W$. In this case condition $8a$ of δ is satisfied.

   (b) $j \geq i$ and there exists $q' \in \Delta(q, \pi_{i-1})$ s.t. $\pi, i, j, \beta \models\!\equiv L(Z^{q'})$. This implies that $\pi, i, \beta \models (Z^{q'}\ \mathsf{seq}\ \xi)$, and that $msi(\pi, i-1, \beta, (Z^{q}\ \mathsf{seq}\ \xi)) = msi(\pi, i, \beta, (Z^{q'}\ \mathsf{seq}\ \xi)) = j$. Thus, $(Z^{q'}\ \mathsf{seq}\ \xi) \in P_i$, and condition $8b$ of δ is satisfied.

(c) There exists an index $i-1 \leq k \leq j$ and a state $q' \in \Delta(q, y)$ s.t. $(i, k) \in \beta$ and $\pi, k, j, \beta |\equiv L(Z^{q'})$. Since $\pi, k, j, \beta |\equiv L(Z^{q'})$ and $\pi, j, \beta \models \xi$, we have that the obligation $o = ((Z^q \textsf{ seq } \xi), wait(L_{i-1}))$ is possible in $\pi, i-1, \beta$. Furthermore, $msi(\pi, i-1, \beta, o) \leq msi(\pi, i-1, \beta, (Z^q \textsf{ seq } \xi)) = j$. Thus $o \in P_{i-1}$ and condition $8c$ of δ is satisfied.

3. If $o = ((Z^q \textsf{ seq } \xi), \Upsilon) \in L_{i-1}$, then o is possible for $\pi, i-1, \beta$. This implies that there exists an index $j \geq i-1$ s.t. $\pi, j, \beta \models \Upsilon$ and for some $q' \in \Delta(q, y)$, we have $\pi, j, \beta \models (Z^{q'} \textsf{ seq } \xi)$. If $j = i-1$, then condition $9a$ of δ is satisfied. Otherwise, $j \geq i$. This implies that o is possible for π, i, β. Thus, $o \in L_i$ and condition $9b$ of δ is satisfied.

4.

5. If $o = ((y \textsf{ seq } \xi), \Upsilon) \in P_{i-1}$, then $o \in L_{i-1}$, thus, o is possible for $\pi, i-1, \beta$. This implies that $j' = msi(\pi, i-1, \beta, o)$ is defined. Thus, there exists an index $j \geq i$ s.t. $\pi, j, \beta \models \Upsilon$, and for some $q' \in \Delta(q, y)$, we have $\pi, j, \beta \models (Z^{q'} \textsf{ seq } \xi)$, and $msi(\pi, j, \beta, (Z^{q'} \textsf{ seq } \xi)) = j'$. We distinguish between two cases:

   (a) If $j = i-1$, then $\pi, j, \beta \models (Z^{q'} \textsf{ seq } \xi)$ implies that $(Z^{q'} \textsf{ seq } \xi) \in L_{i-1}$. Since $msi(\pi, i-1, \beta, (Z^{q'} \textsf{ seq } \xi)) = msi(\pi, i-1, \beta, o)$, we have that $(Z^{q'} \textsf{ seq } \xi) \in P_{i-1}$, thus, condition $10a$ of δ is satisfied.

   (b) Otherwise, $j \geq i$. This implies that o is possible for π, i, β. Furthermore, $msi(\pi, i-1, \beta, o) = msi(\pi, i, \beta, o)$. Thus, $o \in L_i$ and condition $10b$ of δ is satisfied.

6. The definitions of P_i implies that if $P_{i-1} = \emptyset$, then $P_i = L_i \cap (seq(\varphi) \cup obl(\varphi))$.

7. (Condition 12) Suppose that $wait(L_{i-1}) \subseteq L_{i-1}$. Let $(Z^q \textsf{ seq } \xi)$ be in L_{i-1}. Then, $\pi, i-1, \beta \models (Z^q \textsf{ seq } \xi)$. This implies that for some j we have that $\pi, i-1, j, \beta |\equiv L(Z^q)$ and $\pi, j, \beta \models \xi$. Then, the definition of tight satisfaction implies that there is $w \in L$, $w = x_0, x_1, \ldots, x_n$ and there is a sequence of integers $i_0, i_1, \ldots, i_n, i_{n+1}$, such that $i_0 = i$ and $i_{n+1} = j$. Moreover, for every $0 \leq k \leq n$ the following conditions hold:

   - If $x_k \in 2^{AP}$ then $x_k = \pi_{i_k}$ and $i_{k+1} = i_k + 1$.
   - If $x_k = y$ then $(i_k, i_{k+1}) \in \beta$.

Let $q, q_1, q_n, \ldots q_{n+1}$ be an accepting run of Z^q on w. We distinguish between three cases:

(a) If $w = \epsilon$, then $q \in W$, and $j = i - 1$ thus condition $6a$ of delta is satisfied and $(Z^q \textsf{ seq } \xi)$ is strong in L_{i-1}.

(b) If $j = i_{n+1} = i - 1$, then $w = y^n$ and for every $k \leq n + 1$ we have that $\pi, i - 1, \beta \models (Z^{q_k} \textsf{ seq } \xi)$, and since $wait(L_{i-1}) \subseteq L_{i-1}$, $o_k = ((Z^{q_k} \textsf{ seq } \xi), wait(L_{i-1}))$ is possible in $\pi, i-1, \beta$. This implies that for every $k \leq n+1$ we have that $(Z^{q_k} \textsf{ seq } \xi) \in L_{i-1}$, and $o_k \in L_{i-1}$. This implies for every $k \leq n$ there exists a y transition from $(Z^{q_k} \textsf{ seq } \xi)$ to o_k and from o_k to $(Z^{q_{k+1}} \textsf{ seq } \xi)$. Furthermore, $q_{n+1} \in W$, thus $(Z^{q_{n+1}} \textsf{ seq } \xi)$ is strong in L_{i-1} and the condition holds.

(c) If $j > i - 1$, then let l be the maximal index s.t. $i_l = i_0 = i - 1$. Then, for every $k \leq l$ we have that $\pi, i - 1, \beta \models (Z^{q_k} \textsf{ seq } \xi)$, and since $wait(L_{i-1}) \subseteq L_{i-1}$, $o_k = ((Z^{q_k} \textsf{ seq } \xi), wait(L_{i-1}))$ is possible in $\pi, i - 1, \beta$. This implies that for every $k \leq l$ we have that $(Z^{q_k} \textsf{ seq } \xi) \in L_{i-1}$, and $o_k \in L_{i-1}$. This implies for every $k < l$ there exists a y transition from $(Z^{q_k} \textsf{ seq } \xi)$ to o_k and from o_k to $(Z^{q_{k+1}} \textsf{ seq } \xi)$. It is left to show that the path ends at a strong element of L_{i-1}. We distinguish between two cases:

   i. If $x_l = y$, then $o = ((Z^{q_l} \textsf{ seq } \xi), wait(L_{i-1})) \in L_{i-1}$. Since $i_{l+1} > i - 1$, o is strong in L_{i-1}.

   ii. If $x_l \in 2^{AP}$, then $(Z^{q_l} \textsf{ seq } \xi)$ is strong in L_{i-1}.

8. The proof that condition 13 of δ holds is similar to the proof for condition 12, and thus omitted. □

Proposition B.0.19 *ρ is a fair run of A_φ.*

Proof: The proof that $\Phi_1, \Phi_2, \ldots, \Phi_m$ are satisfied is similar to the proof In Section 5. For $\Phi_{\textsf{seq}}$ we prove that for every i there exists $j \geq i$ s.t. $P_j = \emptyset$. We distinguish between two cases:

1. If $P_i = \emptyset$, then we are done.

2. Otherwise, let $k = max\{l | l = msi(\pi, i, \beta, x)$ where $x \in P_i\}$. Intuitively, we show that the maximum msi k does not grow until P is empty, and that P is eventually empty. We prove by induction on j that for every $j \geq i$ one of the following holds:

(a) There exists $i \leq j' \leq j$ s.t. $P_{j'} = \emptyset$.

(b) $max\{l | l = msi(\pi, j, \beta, x)$ for some element in $P_j\} \leq k$.

- Base case: $j = i$, thus (b) holds trivially.
- Assume that the induction proposition holds for j. We distinguish between two cases:
  (a) There exists $i \leq j' \leq j$ s.t. $P_{j'} = \emptyset$, then the induction proposition holds for $j + 1$ as well.
  (b) $k' = max\{l | l = msi(\pi, j, \beta, x)$ for some element in $P_j\} \leq k$. If $P_{j+1} = \emptyset$, then the induction holds. Otherwise, by construction of ρ, for every element x in P_{j+1}, we have $msi(\pi, j+1, \beta, x) \leq k' \leq k$.

Since msi of index j is greater or equal to j, for some $i < j \leq k + 1$, we have that $P_j = \emptyset$. □

Second Direction: $A_\varphi \subseteq L((\exists y)\varphi)$

Lemma B.0.20 *Let π be in $L(A_\varphi)$, then $\pi \in L((\exists y)\varphi)$.*

In the rest of this section, we prove Lemma B.0.20. Let $\rho = (L_0, P_0), (L_1, P_1), \ldots$ be a fair run of A_φ on π. First we construct an interval set β according to ρ. Then, we prove with induction over the structure of φ that for every $i \geq 0$ and every subformula φ' in L_i we have that $\pi, i, \beta \models \varphi'$. For the rest of this section, we fix π and ρ.

We define β as follows: An interval (i, j) is in β iff the following conditions hold:

1. There exists a formula $(Z^q \textsf{ seq } \xi) \in L_i$, for which condition $6c$ holds.
2. For some $q' \in \Delta(q, y)$ we have that $(Z^{q'} \textsf{ seq } \xi) \in L_j$, and $wait(L_i) \subseteq L_j$.

Lemma B.0.21 *For every formula $(Z^q \textsf{ seq } \xi) \in L_i$ for which conditions $6c$ of δ holds, there exists an index $j \geq i$ s.t. $wait(L_i) \subseteq L_j$ and for some $q' \in \Delta(q, y)$ we have that $(Z^{q'} \textsf{ seq } \xi) \in L_j$.*

Proof: Since condition $6c$ of δ holds, we have $o = ((Z^q \textsf{ seq } \xi), wait(L_i)) \in L_i$. First we prove that for every $j \geq i$ one of the following conditions holds:

1. There exists $i \leq j' \leq j$ s.t. j' satisfies the conditions of the lemma.

2. $o \in L_j$.

- Base case: $j = i$ holds trivially.

- Assume that the induction proposition holds for j. Then, if here exists $i \leq j' \leq j$ s.t. j' satisfies the conditions of the lemma, then the induction holds for $j + 1$ as well. Otherwise, condition $9a$ of δ does not hold for $o \in L_j$. This implies that condition $9b$ does, thus, $o \in L_{j+1}$.

This implies that either there exists an index j that satisfies the conditions of the lemma, in which case the lemma holds, or for every $j \geq i$, we have $o \in L_j$. Since ρ is fair, there exists $k \geq i$ s.t. $P_k = \emptyset$. Then, since $o \in L_{k+1}$, we have that $o \in P_{k+1}$. By the same induction we can prove that either there exists an index $j \geq k + 1$ that satisfies the conditions of the lemma, in which case the lemma holds, or for every $j \geq k + 1$, we have $o \in P_j$, this case however, contradicts the fairness of ρ, thus the lemma holds. $\square$

Lemma B.0.21 implies that β is well defined.

Proposition B.0.22 *Let $(Z^q \ \mathsf{seq}\ \xi)$ be a formula in L_i. Then, for every $j \geq i$, one of the following conditions holds:*

1. *There exists $j' \leq j$ s.t. $\pi, i, j', \beta \models L(Z^q)$, $\xi \in L_{j'}$.*

2. *There exists a word $w = x_0, x_1, \ldots x_n$ over $2^{AP} \cup \{y\}$, a run $q, q_1, \ldots, q_{n+1}$ of Z^q on w, and a sequence $i_0, i_1, \ldots i_{n+1}$ s.t. $i_0 = i$, $i_{n+1} = j$, and for every $0 \leq k \leq n$, we have the following:*

   (a) *If $x_k \in 2^{AP}$, then $x_k = \pi_{i_k}$, $i_{k+1} = i_k + 1$, $(Z^{q_k} \ \mathsf{seq}\ \xi) \in L_{i_k}$, and $(Z^{q_{k+1}} \ \mathsf{seq}\ \xi) \in L_{i_{k+1}}$.*

   (b) *If $x_k = y$, then $q_{k+1} \in \Delta(q_k, y)$, and for every $i_k \leq l \leq i_{k+1}$, we have $o = ((Z^{q_k} \ \mathsf{seq}\ \xi), wait(L_{i_k})) \in L_l$.*

Proof: we prove the proposition by induction on j.

- Base case: $j = i$. Condition 6 of δ implies that one of the following should hold:

   1. $q \in W$ and $\xi \in L_i$. In this case the first condition of the proposition holds for $j' = i$.

2. $(Z^{q'} \textsf{seq}\, \xi) \in L_{i+1}$ for some $q' \in \Delta(q, \pi_i)$. In this case second condition holds for $w = \pi_i$, the run q, q' and the sequence $i, i+1$.
3. If $\Delta(q, y) \neq \emptyset$, and $((Z^q \textsf{seq}\, \xi), wait(L_i)) \in L_i$, then the second condition holds for $w = y$, the run q, q' (for some $q' \in \Delta(q, y)$), and the sequence i, i.

- Induction step: Assume that the proposition holds for j. If condition 1 of proposition holds for j, then it holds for $j + 1$ as well. Otherwise, there exists a word $w = x_0, x_1, \ldots x_n$ over $2^{AP} \cup \{y\}$ a run $q, q_1, \ldots, q_{n+1}$ of Z^q on w, and a sequence $i_0, i_1, \ldots i_{n+1}$ s.t. $i_0 = i$, $i_{n+1} = j$, and for every $0 \leq k \leq n$, we have the following:

  1. If $x_k \in 2^{AP}$, then $q_{k+1} \in \Delta(q_k, \pi_{i_k})$, $i_{k+1} = i_k + 1$, $(Z^{q_k} \textsf{seq}\, \xi) \in L_{i_k}$, and $(Z^{q_{k+1}} \textsf{seq}\, \xi) \in L_{i_{k+1}}$.
  2. If $x_k = y$, then $q_{k+1} \in \Delta(q_k, y)$, and for every $i_k \leq l \leq i_{k+1}$, we have $o = ((Z^{q_k} \textsf{seq}\, \xi), wait(L_{i_k})) \in L_l$.

  We distinguish between four cases:

  1. If $x_n \in 2^{AP}$, and $(Z^{q_{n+1}} \textsf{seq}\, \xi) \in L_j$ is strong in L_j. Condition 6 of δ implies that one of the following should hold:
     (a) $q_{n+1} \in W$ and $\xi \in L_j$. In this case condition 1 of the proposition is satisfied with $j' = j$.
     (b) $(Z^{q'} \textsf{seq}\, \xi) \in L_{j+1}$ for some $q' \in \Delta(q_{n+1}, \pi_j)$. In this case condition 2 of the proposition holds for $w' = w \cdot \pi_j$, the run $q, q_1, \ldots, q_{n+1}, q'$, and the sequence $i_0, i_1, \ldots, i_{n+1}, j+1$.
  2. If $x_n = y$, and $o = ((Z^{q_n} \textsf{seq}\, \xi), wait(L_{i_n})) \in L_j$ is strong in L_j. Then, Condition 9 of δ implies that $o \in L_{j+1}$. In this case the second condition holds for $w' = w$, the run $q, q_1, \ldots, q_{n+1}$, and the sequence $i_0, i_1, \ldots, i_n, i_{n+1} = j + 1$ (note that we remove the old i_{n+1}).
  3. If $x_n \in 2^{AP}$, and $(Z^{q_{n+1}} \textsf{seq}\, \xi)) \in L_j$ is not strong in L_j. Condition 6 of δ implies the following should hold: There exists $q' \in \Delta(q_{n+1}, y)$, and $o = ((Z^{q_n} \textsf{seq}\, \xi), wait(L_j)) \in L_j$. We distinguish between two cases:
     (a) If $wait(L_j) \not\subseteq L_j$, condition $9a$ of δ does not hold for o. This implies that condition $9b$ of δ does hold for o. In this case the second condition holds for $w' = w \cdot y$, the run $q, q_1, \ldots q_{n+1}, q'$ (for some $q' \in \Delta(q, y)$) and the sequence $i_0, i_1, \ldots, i_{n+1}, j + 1$.

(b) If $wait(L_j) \subseteq L_j$, then condition 12 of δ implies that there exists a path of y transitions from $(Z^{q_{n+1}} \textsf{seq}\, \xi)$ to a strong element in L_j. Note that if there is a y transition from $(Z^q \textsf{seq}\, \xi)$ to $o = ((Z^q \textsf{seq}\, \xi), \Upsilon)$, and a y transition from o to $(Z^{q'} \textsf{seq}\, \xi)$, then $q' \in \Delta(q, y)$. This implies that there exists a sequence $q_{n+1}, q_{n+2}, \ldots q_{n+m}$ s.t. for every $n+1 \leq l < n+m$, $q_{l+1} \in \Delta(q_l, y)$, and either $(Z^{q_{n+m}} \textsf{seq}\, \xi)$ is strong, or $o = ((Z^{q_{n+m}} \textsf{seq}\, \xi), wait(L_j))$ is strong. If $(Z^{q_{n+m}} \textsf{seq}\, \xi)$ is strong, then we have the same proof as in item 1 with $w \cdot y^m$, the run $q, q_1 \ldots q_{n+m}$ and the sequence $i_0, i_1, \ldots, i_{n+1}, j, j, \ldots, j, j+1$. If $o = ((Z^{q_{n+m}} \textsf{seq}\, \xi), wait(L_j))$ is strong, we have the same proof as in item 2 with $w \cdot y^m$, the run $q, q_1 \ldots q_{n+m}$, and the sequence $i_0, i_1, \ldots, i_{n+1}, j, j, \ldots, j, j+1$.

4. If $x_n = y$, and $o = ((Z^{q_n} \textsf{seq}\, \xi), wait(L_{i_n})) \in L_j$ is not strong in L_j. Then, Condition 9 of δ implies the following should hold: For some $q' \in \Delta(q_n, y)$, we have that $(Z^{q'} \textsf{seq}\, \xi) \in L_j$ and $\Upsilon \subseteq L_j$. If $(Z^{q'} \textsf{seq}\, \xi)$ is strong, then the proof is as in item 1, otherwise it is as in item 3.

$\square$

Proposition B.0.23 *Let $(Z^q \textsf{seq}\, \xi)$ be a formula in L_i, and let $l \geq i$ be an index s.t. $P_l = \emptyset$. Then, for every $j \geq l+1$, one of the following conditions holds:*

1. *There exists $i \leq j' \leq j$ s.t. $\pi, i, j', \beta \models L(Z^q)$, $\xi \in L'_j$.*

2. *There exists a word $w = x_0, x_1, \ldots x_n$ over $2^{AP} \cup \{y\}$, a run $q, q_1, \ldots, q_{n+1}$ of Z^q on w, and a sequence $i_0, i_1, \ldots i_{n+1}$ s.t. $i_0 = i$, $i_{n+1} = j$, and for every $0 \leq k \leq n$, we have the following:*

   (a) If $x_k \in 2^{AP}$, then $q_{k+1} \in \Delta(q_k, \pi_{i_k})$, $i_{k+1} = i_k + 1$, $(Z^{q_k} \textsf{seq}\, \xi) \in L_{i_k}$, $(Z^{q_{k+1}} \textsf{seq}\, \xi) \in L_{i_{k+1}}$, <u>and if $i_k \geq l+1$, then $(Z^{q_k} \textsf{seq}\, \xi) \in P_{i_k}$, $(Z^{q_{k+1}} \textsf{seq}\, \xi) \in P_{i_{k+1}}$</u>.

   (b) If $x_k = y$, then $q_{k+1} \in \Delta(q_k, y)$, and for every $i_k \leq l \leq i_{k+1}$, we have $o = ((Z^{q_k} \textsf{seq}\, \xi), wait(L_{i_k})) \in L_l$, <u>and if $i_k \geq l+1$, then $o \in P_{i_k}$</u>.

The proof of Proposition B.0.23 is similar to the proof of Proposition B.0.22, and thus omitted. Proposition B.0.23 implies the following Corollary.

Corollary B.0.24 *Let $(Z^q \textsf{seq}\, \xi)$ be a formula in L_i, and let $l \geq i$ be an index s.t. $P_l = \emptyset$. Then, one of the following conditions holds:*

1. *There exists $j \geq i$ s.t. $\pi, i, j, \beta |\equiv L(Z^q), \xi \in L_j$.*

2. *There exists an infinite sequence $i_0, i_1, \ldots$ s.t. $i_0 = i$, and for every $k \geq 0$, we have the following:*

   (a) *If $x_k \in 2^{AP}$ and $i_k \geq l+1$, then $(Z^{q_k} \textsf{ seq } \xi) \in P_{i_k}$, $(Z^{q_{k+1}} \textsf{ seq } \xi) \in P_{i_{k+1}}$.*

   (b) *If $x_k = y$ and $i_k \geq l+1$, then $o \in P_{i_k}$.*

   (c) *For every $j \geq l$ there exists j' s.t. $i_{j'} \geq j$.*

We now complete the proof of Lemma B.0.20. We prove by induction over the structure of φ that for every $i \geq 0$ and every subformula φ' in L_i we have that $\pi, i, \beta \models \varphi'$.

- Base case: For p and $\neg p$ the definition of the automaton implies that the lemma holds.

- Induction step: The induction step for the operators $\wedge, \vee$, $\textsf{until}$, $\textsf{release}$, $\textsf{next}$ is identical to the proof of Lemma A.0.8. We prove the induction step for the $\textsf{seq}$, and $\textsf{triggers}$ operators.

  – Let $(Z^q \textsf{ seq } \xi)$ be a formula in L_i. Since ρ is fair, there exists $l \geq i$ s.t. $P_l = \emptyset$. Then, Corollary B.0.24 implies that one of the following conditions holds:

    1. There exists $j \geq i$ s.t. $\pi, i, j, \beta |\equiv L(Z^q)$, $\xi \in L_j$. In this case $\pi, i, \beta \models (Z^q \textsf{ seq } \xi)$.
    2. There exists an infinite sequence $i_0, i_1, \ldots$ s.t. $i_0 = i$, and for every $k \geq 0$, we have the following:
       (a) If $x_k \in 2^{AP}$ and $i_k \geq l+1$, then $(Z^{q_k} \textsf{ seq } \xi) \in P_{i_k}$, $(Z^{q_{k+1}} \textsf{ seq } \xi) \in P_{i_{k+1}}$.
       (b) If $x_k = y$ and $i_k \geq l+1$, then $o \in P_{i_k}$.
       (c) For every $j \geq l$ there exists j' s.t. $i_{j'} \geq j$.

    In this case for every $j \geq l+1$, we have $P_j \neq \emptyset$, thus ρ is not fair, contradiction.

  – Let $(Z^q \textsf{ triggers } \xi) \in L_i$. We need to prove that for every $j \geq i$ s.t. $\pi, i, j, \beta |\equiv L(Z^q)$, we have $\xi \in L_j$. Suppose that for $j \geq i$ we have that $\pi, i, j, \beta |\equiv L(Z^q)$, then there is $w \in L(Z^q)$, $w = x_0, x_1, \ldots, x_n$

and there is a sequence of integers $i_0, i_1, \ldots, i_n, i_{n+1}$, such that $i_0 = i$ and $i_{n+1} = j$. Moreover, for every $0 \leq k \leq n$ the following conditions hold:

* If $x_k \in 2^{AP}$ then $x_k = \pi_{i_k}$ and $i_{k+1} = i_k + 1$.
* If $x_k = y$ then $(i_k, i_{k+1}) \in \beta$.

Let $q, q_1, q_2, \ldots, q_{n+1}$ be an accepting run of Z^q on w. We prove with induction on $0 \leq k \leq n$ that $(Z^{q_k}$ triggers $\xi) \in L_{i_k}$. Furthermore, if $q_k \in W$, then $\xi \in L_{i_k}$.

* Base case $k = 0$, then $i_0 = i$. By definition $(Z^q$ triggers $\xi) \in L_i$. If $q \in W$, condition $7a$ of δ implies that $\xi \in L_{i_0}$.
* Assume that the lemma holds for k, then $(Z^{q_k}$ triggers $\xi) \in L_{i_k}$. we distinguish between two cases:
  1. If $x_k = y$, then the definition of β implies that for every $j \geq i_k$ s.t. $(i_k, j) \in \beta$, in particular i_{k+1}, we have $(Z^{q_{k+1}}$ triggers $\xi) \in L_{i_{k+1}}$. Condition $7a$ of δ implies that if $q_{k+1} \in W$, then $\xi \in L_{i_{k+1}}$.
  2. If $x_k \in 2^{AP}$, then condition $6b$ of δ implies that $(Z^{q_{k+1}}$ triggers $\xi) \in L_{i_{k+1}}$ and condition $7a$ of δ implies that if $q + k + 1 \in W$, then $\xi \in L_{i_{k+1}}$.

$\square$

Appendix C

Deciding does not affect$_S$ is co-NP-hard

Lemma C.0.25 *For φ in LTL, a subformula ψ of φ and a structure M, the problem of deciding whether ψ does not affect$_s$ φ in M is co-NP-complete with respect to the structure M.*

Proof: We show co-NP-hardness. We consider the complementary problem of deciding affect$_s$. We give a reduction from 3CNF satisfiability. For every 3CNF formula θ we construct a structure M_θ. We give a (fixed) LTL formula φ such that $M_\theta \models \varphi$ and the proposition q affects$_s$ φ in M_θ iff θ is satisfiable. Consider the formula $\varphi' = \forall x \varphi\,[q \leftarrow x]$. By definition, $M_\theta \not\models \forall x \varphi'$ iff there exists an assignment σ such that $M, \sigma \not\models \varphi'$. We construct M_θ so that the set $\sigma(x)$ represents a satisfying assignment to θ.

For every proposition p_i in θ we have a set of states that represent the assignment $p_i = \mathbf{false}$ and a set of states that represent the assignment $p_i = \mathbf{true}$. The formula φ is constructed so that $M, \sigma \models \varphi\,[q \leftarrow x]$ whenever σ chooses for x a set of states that cannot represent a valid assignment to the propositions of θ. For example, if σ chooses for x only some of the states that represent $p_i = \mathbf{false}$ (or $p_i = \mathbf{true}$) or if σ chooses for x some states that represent $p_i = \mathbf{false}$ and some states that represent $p_i = \mathbf{true}$ for some proposition p_i.

For every clause c_i of θ we add one path to M_θ. If the clause c_i uses propositions p_a, p_b, and p_c we create a path linking a state representing proposition p_a to a state representing proposition p_b to a state representing proposition p_c. If p_a appears in c_i positively, we choose a state that represents $p_a = \mathbf{true}$, otherwise we choose a state that represents $p_a = \mathbf{false}$. Similarly for p_b and p_c. This way, if

$\sigma(x)$ is a valid assignment that does not satisfy the clause c_i then all the states on the path of c_i in M_θ are not in $\sigma(x)$.

Let $\theta = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \alpha_{i,j}$ where $\alpha_{i,j}$ is a literal in $\{p_1, \ldots, p_k\} \cup \{\neg p_1, \ldots, \neg p_k\}$. For every proposition p_i the structure M_θ contains $2n$ states. The first n states represent the assignment $p_i = \mathbf{true}$ and the other n states represent the assignment $p_i = \mathbf{false}$. Then for every clause $c_i = \alpha_{i,1} \vee \alpha_{i,2} \vee \alpha_{i,3}$, we create a path that connects the literals in c_i.

Let $M_\theta = \langle \{c, pos, neg, q\}, S, \{s_0\}, R, L\rangle$. The set of states S is the union of the following sets.

- $\{s_0\}$ - the initial state.
- $\{c_{i,1}, c_{i,2} \mid 1 \leq i \leq n\}$ - two clausal states per clause. These states are used in the path that represents clause i to separate the different proposition states.
- $\{p^+_{l,i}, p^-_{l,i} \mid 1 \leq l \leq k \text{ and } 1 \leq i \leq n\}$ - $2n$ propositional states per proposition, n positive and n negative.

The transition relation is the union of the following sets.

- $R_1 = \{(s_0, p^+_{l,1}) \mid 1 \leq l \leq k\}$ - the initial state s_0 is connected to every first positive propositional state $p^+_{l,1}$.
- $R_2 = \{(p^+_{l,i}, p^+_{l,i+1}), (p^-_{l,i}, p^-_{l,i+1}) \mid 1 \leq l \leq k \text{ and } 1 \leq i \leq n-1\}$ - the positive states related to proposition p_l and the negative states related to proposition p_l form chains.
- $R_3 = \{(p^+_{l,n}, p^-_{l,1}), (p^-_{l,n}, p^-_{l,n}) \mid 1 \leq l \leq k\}$ - The last positive state of p_l is connected to the first negative state. The last negative state of p_l is connected to itself.
- For every clause $\theta_i = \beta_1 \cdot p_a \vee \beta_2 \cdot p_b \vee \beta_3 \cdot p_c$ where $\beta_o \in \{+, -\}$ for $o \in \{1, 2, 3\}$ we add the transitions $R_{4,i} = \{(s_0, p^{\beta_1}_{a,i}), (p^{\beta_1}_{a,i}, c_{i,1}), (c_{i,1}, p^{\beta_2}_{b,i}), (p^{\beta_2}_{b,i}, c_{i,2}), (c_{i,2}, p^{\beta_3}_{c,i})\}$ - there is a path connecting the literals of clause c_i according to their polarities. Between every two propositional states there is a clausal state. We refer to this path as a clausal path. The only way to get from one proposition state to another proposition state in one step is by taking transitions in $R_2 \cup R_3$. Notice that the paths that correspond to different clauses do not share transitions.

The labeling is $L(c) = \{c_{i,j}\}$, $L(pos) = \{p^+_{i,l}\}$, $L(neg) = \{p^-_{i,l}\}$, and $L(q) = \emptyset$. In Figure C we have the 'propositional' part of M_θ without the clausal states and transitions. The structure M_θ can be constructed in polynomial time.

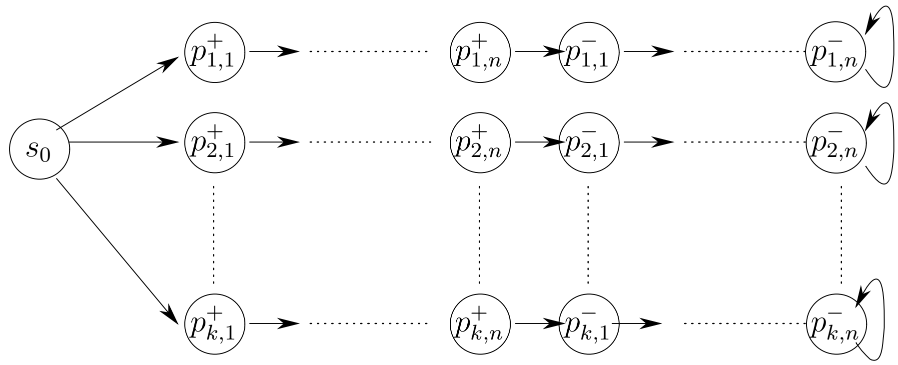

Figure C.1: The structure M_θ

The formula φ is the disjunction of the following formulas.

- $\varphi_1 = F(pos \wedge Xpos \wedge ((q \wedge X\neg q) \vee (\neg q \wedge Xq)))$ - there are two positive states associated with the same proposition (reachable in one step) assigned with different values of q.

- $\varphi_2 = F(neg \wedge Xneg \wedge ((q \wedge X\neg q) \vee (\neg q \wedge Xq)))$ - there are two negative states associated with the same proposition (reachable in one step) assigned with different values of q.

- $\varphi_3 = F(pos \wedge Xneg \wedge ((q \wedge Xq) \vee (\neg q \wedge X\neg q)))$ - the last positive state and the first negative state agree on the assignment of q.

- $\varphi_4 = X(\neg q \wedge X(c \wedge X(\neg q \wedge X(c \wedge X\neg q))))$ - all three literals are not satisfied on a clausal path.

As $L(q) = \emptyset$ the formula φ_3 holds in M and $M_\theta \models \varphi$. We claim that $M_\theta \not\models \forall x\varphi\,[q \leftarrow x]$ iff θ is satisfiable. Indeed, every assignment to x that does not satisfy $\varphi\,[q \leftarrow x]$ must include either all the positive states associated with one proposition or all the negative states associated with one proposition (and not both). Furthermore, as the assignment falsifies $\varphi\,[q \leftarrow x]$ every path associated with some clause must have at least one literal satisfied. Similarly, a satisfying assignment to θ translates to a subset of the states S' assigning $\sigma(x) = S'$ falsifies $\varphi[q \leftarrow x]$. $\square$

In [KV03] Kupferman and Vardi show that deciding affects$_f$ for CTL formulas is NP-complete. They give a reduction from SAT to deciding affects$_f$. In their proof both the structure and the CTL formula depend on the SAT formula. Our proof above can be used to show that for CTL formulas, deciding affects$_f$ is NP-hard in the structure even for a constant formula.

Appendix D

Regular Vacuity Lower Bound

In the *exponential bounded-tiling* problem we are given a fixed set T of tiles, two relations $H, V \subseteq T \times T$, two tiles $t_{init}, t_{fin} \in T$, and an integer n. The goal is to tile a $(2^n \times 2^n)$-square so that horizontal neighbors belong to H, vertical neighbors belong to V, the first tile in the first row is t_{init}, and the first tile in the last row is t_{fin}. Thus, formally, a legal tiling is a function $t : \{0, \ldots, 2^n - 1\}^2 \rightarrow T$ such that the following hold:

- for all $0 \leq i \leq 2^n - 2$ and $0 \leq j \leq 2^n - 1$, we have that $H(t(i, j), t(i+1, j))$,
- for all $0 \leq i \leq 2^n - 1$ and $0 \leq j \leq 2^n - 2$, we have that $V(t(i, j), t(i, j+1))$,
- $t(0, 0) = t_{init}$, and $t(0, 2^n - 1) = t_{fin}$.

The exponential bounded tiling problem is known to be NEXPTIME-hard [SveB84].

Theorem D.0.26 *The regular vacuity problem for RELTL is NEXPTIME-hard.*

Proof: We do a reduction from the exponential bounded tiling problem. Given a tiling system $\mathcal{T} = \langle T, H, V, n, t_{init}, t_{fin} \rangle$, we construct a model $M_{\mathcal{T}}$ of a fixed size and an RELTL formula φ of length $O(n)$ such that $\neg\varphi$ is not regularly vacuous in $M_{\mathcal{T}}$ iff there is a legal tiling t for $\mathcal{T}$.

We encode the tiles in T by a set $AP(T) = \{p_1, \ldots, p_m\}$ of atomic propositions. We define the formula φ over the set $AP = AP(T) \cup \{b, c, d, r\}$ of atomic propositions. The task of the last four atoms will be explained shortly. Since T is fixed, so is AP.

Consider an infinite word π over 2^{AP}. For an atomic proposition $p \in AP$ and a point u in π, we use $p(u)$ to denote the truth value of p at u. That is, $p(u)$ is 1 if p

holds at u and is 0 if p does not hold at u. We divide the word π to blocks of length $2n$. Every block corresponds to a single location in the $(2^n \times 2^n)$-square. Consider a block $u_1, \ldots, u_{2n}$ that corresponds to location $[i, j]$ of the square. We use the point u_1 to encode the tile in location $[i, j]$. Thus, the bit vector $p_1(u_1), \ldots, p_m(u_1)$ encodes the tile $t(i, j)$. We use the atomic proposition b to mark the beginning of the block; that is, b holds on u_1 and fails on $u_2, \ldots, u_{2n}$. This is enforced by the formula φ. Thus, φ contains a conjunct[1]

$$b \wedge (\bigwedge_{1 \leq k \leq 2n-1} \textsf{next}^{\,k} \neg b) \wedge \textsf{globally}\ (b \leftrightarrow \textsf{next}^{\,2n} b).$$

The block $u_1, \ldots, u_{2n}$ also encodes the location of the tile in the square. Since the square is of dimensions $2^n \times 2^n$, this location is a pair $\langle i, j \rangle$, for $0 \leq i, j \leq 2^n - 1$, where i is the column of the tile and j is its row. Encoding the location eliminates the need for exponentially many $\textsf{next}$ operators when we attempt to relate tiles that are vertical neighbors. Encoding is done by the atomic proposition c, called *counter*. Let $c(u_n), \ldots, c(u_1)$ encode i, and $c(u_{2n}), \ldots, c(u_{n+1})$ encode j. Note that, for technical convenience, the least significant bits of the counters are in u_1 and u_{n+1}, and i is encoded before j. A sequence of 2^n blocks corresponds to 2^n tiles and, when starts with $i = 0$, encodes some row j in the square. The values of the counters along this sequence go from $\langle 0, j \rangle$ to $\langle 2^n - 1, j \rangle$, and then start again with $i = 0$, but with an increased j. Thus, the next sequence goes from $\langle 0, j+1 \rangle$ to $\langle 2^n - 1, j+1 \rangle$. The way we encode the counters guarantees that an increase of the counter by one corresponds to either a transition from $\langle i, j \rangle$ to $\langle i+1, j \rangle$, in case $i \neq 2^n - 1$, or to a transition from $\langle 2^n - 1, j \rangle$ to $\langle 0, j+1 \rangle$, otherwise. A proper behavior of the counters is enforced by φ. Since we want the length of φ to be $O(n)$, we need also an atomic proposition d that acts as a "carry" bit. Note that $b \vee d$ holds in a point u_i iff $c(u_i) \neq c(u'_i)$, where u' is the successor block of u. Formally, φ contains the following conjuncts.

1. The counter starts at 0: $\bigwedge_{0 \leq k \leq 2n-1} \textsf{next}^{\,k} \neg c$.

2. The counter is increased properly. Note that as we always want to increase the counter by 1 we take d as a carry to the least significant bit:

    - $\textsf{globally}\ (((b \vee d) \wedge \neg c) \rightarrow (\textsf{next}\ (\neg d) \wedge \textsf{next}^{\,2n} c))$.

[1] Note that the formula is of quadratic length. An equivalent formula of a linear length replaces conjuncts like $\bigwedge_{1 \leq k \leq 2n-1} X^k p$ by $X(p \wedge X(p \wedge \cdots \wedge Xp) \cdots)$. In order to keep the reference to indices clear, we describe here and in the sequel the quadratic version.

- $\textsf{globally}\ ((\neg(b \vee d) \wedge \neg c) \rightarrow (\ \textsf{next}\ (\neg d) \wedge\ \textsf{next}^{\,2n} \neg c))$.
- $\textsf{globally}\ (((b \vee d) \wedge c) \rightarrow (\ \textsf{next}\ d \wedge\ \textsf{next}^{\,2n} \neg c))$.
- $\textsf{globally}\ ((\neg(b \vee d) \wedge c) \rightarrow ((\ \textsf{next}\ \neg d) \wedge\ \textsf{next}^{\,2n} c))$.

Since each location is encoded by a block of length $2n$, the whole tiling is encoded in a finite prefix of π, namely $\pi_0, \ldots \pi_{2n(2^n)^2-1}$. We use the atomic proposition r in order to label this "relevant prefix." More precisely, r holds exactly in this prefix. Thus, φ contains a conjunct

$$(r\ \textsf{until}\ (b \wedge (\bigwedge_{0 \leq k \leq 2n-1} \textsf{next}^{\,k}(r \wedge c)))) \wedge \textsf{globally}\ (b \wedge (\bigwedge_{0 \leq k \leq 2n-1} \textsf{next}^{\,k} c)) \rightarrow\ \textsf{next}^{\,2n}\ \textsf{globally}\ \neg r).$$

Let $t_0 \ldots t_{2^n-1}, t'_0 \ldots t'_{2^n-1}$ be two successive rows of the tiling t. For each i, $0 \leq i \leq 2^n - 1$, we know, given t_i, the possible values for t_{i+1} (these for which $H(t_i, t_{i+1})$, in case $i < 2^n - 1$) and the possible values for t'_i (these for which $V(t_i, t'_i)$). Consistency with H and V gives us a necessary condition for a word to encode a legal tiling. In addition, the tiling should satisfy the edge conditions; it should start with t_{init} and has t_{fin} in position $[0, 2^n - 1]$. For a tile $t \in T$, let $\rho(t)$ be the propositional formula over AP that encodes t. That is, $\rho(t)$ holds in point u_1 of exactly all blocks that encode the tile t. In order to make sure that the edge conditions hold, φ contains the conjunct

$$\rho(t_{init}) \wedge\ \textsf{globally}\ ((b \wedge (\bigwedge_{0 \leq k \leq n-1} \textsf{next}^{\,k} \neg c) \wedge (\bigwedge_{n+1 \leq k \leq 2n} \textsf{next}^{\,k} c)) \rightarrow \rho(t_{fin})).$$

Since the distance between the point where t_i is encoded to the one where t_{i+1} is encoded is exactly $2n$, it is also easy to specify the conditions for horizontal neighbors. Note that the conditions are imposed only when $i \neq 2^n - 1$:

$$\textsf{globally}\ ((b \wedge (\bigvee_{0 \leq k \leq n-1} \textsf{next}^{\,k} \neg c) \wedge \rho(t)) \rightarrow\ \textsf{next}^{\,2n} \bigvee_{t' : H(t,t')} \rho(t')).$$

The difficult part in the reduction is in guaranteeing that the condition for vertical neighbors hold. This is where regular vacuity comes into the picture. They enable us to relate $t[i, j]$ with $t[i, j + 1]$, for all i and j. Let e be a regular expression. Consider the formula ξ_1 below. The formula says that whenever we are in a beginning of a block u corresponding to position $[i, j]$, for some i and j, then the regular expressions e is tightly satisfied only in intervals that end when the block u' that corresponds to position $[i, j + 1]$ starts. To see this, recall that $b \vee d$ holds in a point u_i iff $c(u_i) \neq c(u'_i)$, and note that the formula requires the value of the counter in $u'_{2n}, \ldots, u'_{n+1}$ (that is, the j-coordinate of u') to be greater by 1 than the value in $u_{2n}, \ldots, u_{n+1}$ (the j-coordinate of u), and requires the value of the counter in $u'_n, \ldots, u'_1$ (the i-coordinate of u') to be equal to the value of the counter in $u_n, \ldots, u_1$ (the i-coordinate of u). Note that the requirement is imposed only when u is not in the last row, thus $j \neq 2^n - 1$:

$$\xi_1 = \textsf{globally}\,((b \wedge \textsf{next}^{\,n} \bigvee_{0 \leq k \leq n-1} \textsf{next}^{\,k} \neg c) \rightarrow \bigwedge_{1 \leq k \leq n} \theta_1^k \wedge \theta_2^k \wedge \textsf{next}^{\,n}(\theta_3^k \wedge \theta_4^k)), \text{ where}$$

- $\theta_1^k = (\textsf{next}^{\,k} c) \rightarrow e$ TRIGGERS $\textsf{next}^{\,k} c$,
- $\theta_2^k = (\textsf{next}^{\,k} \neg c) \rightarrow e$ TRIGGERS $\textsf{next}^{\,k} \neg c$,
- $\theta_3^k = (\textsf{next}^{\,k}((c \wedge \neg(b \vee d)) \vee (\neg c \wedge (b \vee d)))) \rightarrow e$ TRIGGERS $\textsf{next}^{\,k} c$, and
- $\theta_4^k = (\textsf{next}^{\,k}((\neg c \wedge \neg(b \vee d)) \vee (c \wedge (b \vee d)))) \rightarrow e$ TRIGGERS $\textsf{next}^{\,k} \neg c$.

Consider now the formula ξ_2 below. The formula says that whenever a block u, not in the last row, starts, there is a block u' in the relevant prefix of π that starts when an interval satisfying e ends, and the blocks u and u' encode tiles that are related by V.

$$\xi_2 = \bigwedge_{t \in T} \textsf{globally}\,((r \wedge b \wedge (\bigvee_{1 \leq k \leq n} \textsf{next}^{\,n+k} \neg c) \wedge \rho(t)) \rightarrow e\ \text{SEQ}(r \wedge \bigvee_{t' : V(t,t')} \rho(t'))).$$

The formula φ contains a conjunct $\xi_1 \wedge \xi_2$, with $e = b$ (in fact any $e \neq \mathbf{true}^{2n2^n}$ will do). Note that for $e = b$, the formula ξ_1 does not hold in a path in which the counters are increased properly.

Let $M_{\mathcal{T}}$ be a Kripke structure that generates all the computations over AP. Thus, $M_{\mathcal{T}} = \langle AP, 2^{AP}, 2^{AP}, 2^{AP} \times 2^{AP}, L \rangle$ with $L(\sigma) = \sigma$. We prove that $\neg\varphi$ is not regularly vacuous in $M_{\mathcal{T}}$ iff there is a legal tiling t for $\mathcal{T}$.

Assume first that there is a legal tiling t for $\mathcal{T}$. Recall that ξ_1 does not hold in a path in which the counters are increased properly. Therefore, φ is not satisfiable, and all the paths of $M_{\mathcal{T}}$ satisfy $\neg\varphi$. We show that all the regular expressions in $\neg\varphi$ affect it, thus $\neg\varphi$ is not regularly vacuous in $M_{\mathcal{T}}$. The single regular expression is $\neg\varphi$ is $e = b$. By the definition of φ, the path π that describes t satisfies $(\exists y).\varphi[e \leftarrow y]$. Indeed, since π describes a legal tiling and since the distance between points where successive points start is $2n2^n$, the interval set β that contains all intervals of length $2n2^n$ is such that $\pi, 0, \beta \models \varphi$. It follows that e affects $\neg\varphi$ in $M_{\mathcal{T}}$, thus $\neg\varphi$ is not regularly vacuous in $M_{\mathcal{T}}$.

For the other direction, assume that $\neg\varphi$ is not regularly vacuous in $M_{\mathcal{T}}$. Since e is the only regular expression in φ, it follows that $M_{\mathcal{T}} \models (\exists y)\varphi[e \leftarrow y]$. Let β be an interval set such that $M_{\mathcal{T}}$ has a path π for which $\pi, 0, \beta \models \varphi$. The conjuncts of φ that are independent of e (that is, all conjuncts except for $\xi_1 \wedge \xi_2$) guarantee that the path π describes a tiling that satisfies the edge conditions and the conditions for horizontal neighbors. By the definition of ξ_1, the path π is such that whenever we are in a beginning of a block u corresponding to position $[i, j]$, for some i and j, then the regular expressions e is tightly satisfied only in intervals that end when the block u' that corresponds to position $[i, j+1]$ starts. Therefore, β contains only intervals of length $2n2^n$. Hence, ξ_2 guarantees that π describes a tiling that also satisfies the conditions for vertical neighbors. Thus, π describes a legal tiling for $\mathcal{T}$, and we are done. $\square$