Synchronous or Alternating?
LTL Black-Box Checking of Mealy Machines by Combining the LearnLib and LTSmin

Jaco van de Pol(✉) and Jeroen Meijer

Formal Methods and Tools, University of Twente, Enschede, The Netherlands
{j.c.vandepol,j.j.g.meijer}@utwente.nl

Abstract. Mealy machines transduce inputs to outputs, based on finite memory. They are often used to model reactive systems. The requirements on their behaviour can be specified by formulas in Linear-time Temporal Logic. We will study two interpretations of LTL for Mealy machines: the synchronous semantics, where inputs and outputs occur simultaneously; and the alternating semantics, where inputs and outputs strictly alternate. We define and study Mealy-robust LTL properties, which are insensitive to which of these interpretations is chosen.

The motivating application is in the context of black-box checking: Given the interface to some reactive system, one would like to test that a particular LTL property holds. To this end, we combine active automata learning with model checking into sound black-box checking. Here the LTL properties are already checked on intermediate hypotheses, in order to speed up the learner. Finally, we perform an experiment on the Mealy machines provided by the RERS challenge (Rigorous Examination of Reactive Systems). We investigate how many LTL properties from the RERS challenges in 2016 and 2017 are actually robust.

1 Introduction

The problem of black-box checking is to verify (or test) that a reactive system satisfies a number of properties. Here the system is provided as black-box, accessible through its input/output interface only. Assuming that the reactive system has a finite number of states, it can be modeled by a Mealy machine [10]. The properties on the system's behaviour can be specified in Linear-time Temporal Logic (LTL [13]). A solution to the black-box checking problem can be obtained by combining active automata learning [1,6,17] and model checking [2].

A realisation of the black-box checking approach is provided by integrating LearnLib[1] [7,15] with LTSmin[2] [9]. The yearly RERS challenge[3] (Rigorous Evaluation of Reactive Systems [5,8]) provides an excellent testbed

[1] The LearnLib: https://learnlib.de.
[2] LTSmin: `ltsmin.utwente.nl` and https://github.com/utwente-fmt/ltsmin.
[3] RERS challenge: http://rers-challenge.org.

T. Margaria et al. (Eds.): Steffen Festschrift, LNCS 11200, pp. 417–430, 2019.
https://doi.org/10.1007/978-3-030-22348-9_24

for checking properties on reactive systems. The combination of LearnLib + LTSmin has been applied on the problems in the RERS Challenge 2017, as reported in [11].

1.1 Motivating Context: Sound Black-Box Checking

The sound approach to black-box checking is illustrated in Fig. 1. A naive approach to black-box checking would be to first learn a Mealy machine that models the System Under Learning (SUL) and to subsequently check the properties on that automaton. Learning proceeds according to the Angluin-style active automata-learning paradigm [1,6], adapted to Mealy machines in [17]. Here the learner proceeds by performing I/O sequences (membership queries $\in$) on the system. When it believes that it has complete information, it generates a hypothesis automaton (H). This hypothesis is validated on the system using a

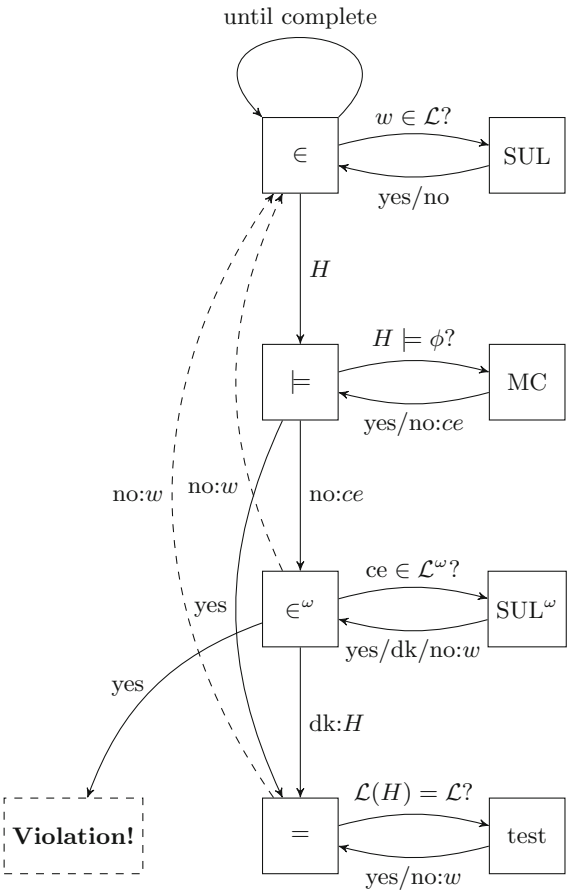

Fig. 1. Sound black-box checking procedure, adapted and simplified from [11]

model-based tester (=); this is an expensive step, involving many queries. This procedure is iterated as long as testing reveals a counterexample (w). The final hypothesis can be used to check the LTL properties, for instance using the Nested Depth-First Search algorithm [3].

A smarter approach to black-box checking [12] applies the model checker (MC) already on the intermediate hypotheses ($\models$). If the model checker provides a counter-example (ce) to the property, this is tested on the reactive system. If the counter-example can be simulated on the system, we found a violation of the property. If not, a prefix of the counter example (w) is provided to the learner, saving one expensive test-procedure.

A complication is that the counter-example provided by the model checker is an infinite path, presented by a lasso xy^ω. In principle, one can only check finite unrollings xy^n on the system. However, this yields an unsound method, unless one knows an upperbound on the number of states of the reactive system.

A sound approach to black-box checking was proposed in [11]. We adapt the check for infinite words ($\in^\omega$), by assuming that one can additionally save states and check their equivalence. So we test the word and save intermediate states $x(s_0)y(s_1)y(s_2), \ldots, y(s_n)$. As soon as we find that $s_k = s_j$ for some $0 \leq k < j \leq n$, we definitely know that xy^ω is a valid counterexample, and report a violation. If the path cannot be continued, we have a found a finite prefix w for the learner. Otherwise, we don't know if xy^ω holds, and we proceed to the tester.

The adapted procedure is sound, in the sense that it only reports true violations. However, it may miss some violations, so it is incomplete: First, the final hypothesis may still not reflect all system behaviour. Second, the model checker may have detected a lasso that could not be confirmed within the bound. Note that the state recording facility could in principle be used for a full model check.

1.2 Problem Statement and Contribution

This paper, dedicated to Bernhard Steffen on the occasion of his 60th birthday, is devoted to taking a closer look at the precise LTL semantics for Mealy machines. In particular, we study the difference between the *synchronous* semantics and the *alternating* semantics. The RERS organisers clearly stipulate that LTL properties are interpreted in the *alternating* semantics, i.e. interpreted over alternating traces of the form $i, o, i, o, \ldots$ However, in the first RERS attempt in 2012 [14], LTSmin used the *synchronous* semantics, interpreted over synchronous traces of the form $i/o, i/o, \ldots$

Surprisingly, this discrepancy leads to only very few wrong answers. When applying sound black-box learning (Fig. 1) to the first 4 problems of the RERS 2017 challenge, with 100 LTL formulas each, we detect the following number of LTL violations for the alternating, resp. synchronous semantics. So there is only a 0.5% deviation! We will show a deviating LTL property from RERS 2017 in Example 4.

We will try to explain this, by studying the class of *Mealy-robust LTL properties*, which are insensitive to choosing the synchronous or alternating LTL seman-

Semantics	Problem 1	Problem 2	Problem 3	Problem 4
Alternating	52	46	54	69
Synchronous	50	46	54	69

tics. In Sect. 2, we formally define (partial) Mealy machines, the synchronous and alternating semantics of LTL properties, and the set of Mealy-robust LTL properties. Section 3 studies Robust LTL in more detail; we restrict ourselves to properties in LTL\X. To this end, we need to introduce a number of finer distinctions (α-, α^1-, σ and σ^1-robustness). A Prolog program summarises *and* automates the derivation rules for robustness. Its correctness depends on the lemmas proved in Appendix A. Section 4 performs a small experiment on the problems of the RERS challenge, checking how many of them we can detect to be robust. Finally, we conclude with some problems left for future research.

2 Preliminaries: LTL Interpretations for Mealy Machines

A (partial) Mealy machine $M = (S, s_0, I, O, \delta)$ consists of a finite set of states S, initial state $s_0 \in S$, nonempty finite disjoint sets of input symbols (I) and output symbols (O), and a partial transition function $\delta : S \times I \hookrightarrow O \times S$. An example is provided in Fig. 2. We will distinguish its *synchronous traces* and *alternating traces*. In a synchronous trace, inputs and outputs happen simultaneously, for example $a/x, a/y, a/x, a/y, b/z, b/z, \ldots$. In an alternating trace, inputs and outputs happen in strict alternation, as in $a, x, a, y, a, x, a, y, \ldots$.

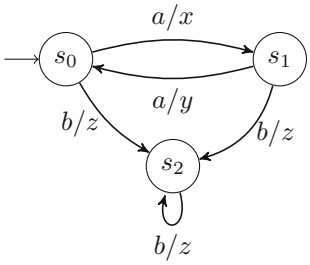

Fig. 2. Mealy machine M with $I = \{a, b\}$ and $O = \{x, y, z\}$.

An (infinite) sequence over A is a function $\mathbb{N} \to A$. Given a sequence π, we write π_i for the i-th element (so $\pi = \pi_0, \pi_1, \ldots$) By π^i we denote the suffix $\pi^i, \pi^{i+1}, \pi^{i+2}, \ldots$.

We formally define the set of synchronous traces $Tr_s = \mathbb{N} \to I \times O$ over I and O, and the set of alternating traces $Tr_a = \pi : \mathbb{N} \to I \cup O$, with $\pi_i \in I \iff \pi_{i+1} \in O$. The latter can be split in Tr_{ai} (starting with an input: $\pi_0 \in I$) and Tr_{ao} (starting with an output: $\pi_0 \in O$). With $Tr_s(M)$ (resp. $Tr_a(M)$) we denote the synchronous (resp. alternating) traces that start in s_0 and follow transitions in M. Note that $Tr_a(M) \subseteq Tr_{ai}$. For $\pi \in Tr_{ai}$, we write $\sigma(\pi)$ for the corresponding synchronous trace. For $\pi \in Tr_s$, $\alpha(\pi)$ denotes the corresponding alternating trace. Note that $\sigma = \alpha^{-1}$ forms a bijection between Tr_s and Tr_{ai}. However, traces in Tr_{ao} still arise as suffixes.

Example 1. Let M be the Mealy machine in Fig. 2. Define the synchronous trace $\pi := a/x, a/y, a/x, a/y, \ldots$ and the alternating trace $\rho := a, x, a, y, b, z, b, z, \ldots$ Indeed, $\pi \in Tr_s(M)$ and $\rho \in Tr_a(M)$. In particular, $\rho \in Tr_{ai}$ and $\rho^1 \in Tr_{ao}$. Finally, $\alpha(\pi) = a, x, a, y, a, x, a, y, \ldots$, while $\sigma(\rho) = a/x, a/y, b/z, b/z, \ldots$ However, $\sigma(\rho^1)$ is not defined.

The distinction between synchronous and alternating traces may seem a small technical detail, but it does have a crucial impact on the corresponding LTL semantics. Let us first define action-based LTL formulas with atomic properties in $I \cup O$, using the following grammar (U is *until*; X is *next*):

$$\Phi ::= I \mid O \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \,\mathsf{U}\, \Phi \mid \mathsf{X}\Phi$$

We permit the usual abbreviations $\phi \vee \psi := \neg(\neg\phi \wedge \neg\psi)$, $\mathsf{F}\,\phi := \mathsf{true}\,\mathsf{U}\,\phi$ (future), $\mathsf{G}\,\phi := \neg\,\mathsf{F}\,\neg\phi$ (globally), $\phi\,\mathsf{R}\,\psi := \neg(\neg\phi\,\mathsf{U}\,\neg\psi)$ (release), and $\phi\,\mathsf{WU}\,\psi := (\phi\ \mathsf{U}\ \psi) \vee (\mathsf{G}\,\phi)$ (weak until). We also introduce conveniently defined atomic properties: $\mathsf{false} := i_0 \wedge \neg i_0$ (with $i_0 \in I$ arbitrary), $\mathsf{true} := \neg\mathsf{false}$, $\mathsf{input} := \bigvee_{i \in I} i$, $\mathsf{output} := \bigvee_{o \in O} o$, which hold for none, all, all input, and all output actions, respectively.

Next, we define the LTL semantics over both synchronous and alternating traces, i.e. $\vDash \subseteq (Tr_s \cup Tr_a) \times \Phi$, by induction over ϕ:

$$\begin{aligned}
\pi \vDash i &\iff \pi_0 \in \{i/o, i\}, \text{ for some } o \in O \\
\pi \vDash o &\iff \pi_0 \in \{i/o, o\}, \text{ for some } i \in I \\
\pi \vDash \neg\phi &\iff \pi \nvDash \phi \\
\pi \vDash \phi \wedge \psi &\iff \pi \vDash \phi \text{ and } \pi \vDash \psi \\
\pi \vDash \phi\,\mathsf{U}\,\psi &\iff \exists j : (\forall k < j : \pi^k \vDash \phi) \text{ and } \pi^j \vDash \psi \\
\pi \vDash \mathsf{X}\,\phi &\iff \pi^1 \vDash \phi
\end{aligned}$$

We are now ready to define the synchronous and alternating semantics of LTL. Note that the following definition discards finite executions of the Mealy machine, even when they lead to a deadlock. The motivation for handling *partial* Mealy machines, but ignoring finite traces, is simply to abide to the rules of the RERS challenge.

Definition 2 (LTL semantics). *For a Mealy machine M and LTL formula ϕ, we define:*

- $M \vDash_s \phi$ *if and only if for all synchronous traces* $\pi \in Tr_s(M)$, $\pi \vDash \phi$
- $M \vDash_a \phi$ *if and only if for all alternating traces* $\pi \in Tr_a(M)$, $\pi \vDash \phi$

The *alternating* semantics is the official LTL semantics of the RERS challenge. Indeed, it supports the intuition that the current state and the input determine, so should precede, the next state and the output. However, when mapping this to a standard LTS for model checking, one typically introduces an "intermediate state" between an input and its subsequent output, which seems unnatural and superfluous. This would lead to $|S| \cdot |I|$ extra states, which makes model checking less efficient. The *synchronous* semantics avoids introducing intermediate states, so it would lead to a more efficient model checking procedure. This is in particular useful when applying brute-force white-box model checking to the RERS problems, for which LTSmin has traversed state spaces of over 5.10^9 states and 5.10^{10} transitions [14], but it is also convenient in the black-box checking scenario.

So the main question is: when is using the synchronous semantics justified? We will call LTL properties that are insensitive to choosing the synchronous or alternating semantics *Mealy-robust*.

Definition 3 (Mealy-Robust LTL properties). *We call LTL formula* ϕ Mealy-robust *if for all Mealy machines* M*, it holds that* $M \vDash_s \phi \iff M \vDash_a \phi$.

Example 4. Property 2 of Problem 1 of the RERS challenge 2017 is:

```
(false R (! ((oY & ! iC) & (true U iC)) | (! oU U (iB | iC))))
```

In standard notation: $\mathsf{G}(\neg o_Y \vee i_C \vee (\mathsf{G}\,\neg i_C) \vee (\neg o_U \,\mathsf{U}\, (i_B \vee i_C)))$. This happens to be one of the examples from the Introduction (Sect. 1) where the alternating and synchronous semantics differ, so it is not robust.

Example 6 will introduce some simpler robust and non-robust formulas.

3 Mealy-Robust LTL Properties

We will now investigate the following question: *Which LTL formulas* ϕ *are Mealy-robust?* We start by defining a number of fine-grained robustness notions on paths. Subsequently, we will prove preservation of robustness by LTL operators. This will yield a procedure to identify a class of robust LTL properties.

3.1 Robustness Notions

Note that to prove robustness of ϕ, we can focus on the robustness for individual paths. We need preservation in two directions, leading to the notions of α- and σ-robustness. However, for the alternating semantics we also need to consider the situation between an input and output action. Hence the notions of α^1- and σ^1-robustness, which consider traces that start with an output action.

Definition 5 (Robustness w.r.t. paths).

- ϕ *is* α*-robust if* $\forall \pi \in Tr_s : \pi \vDash \phi \implies \alpha(\pi) \vDash \phi$
- ϕ *is* σ*-robust if* $\forall \pi \in Tr_{ai} : \pi \vDash \phi \implies \sigma(\pi) \vDash \phi$
- ϕ *is* α^1*-robust if* $\forall \pi \in Tr_s : \pi \vDash \phi \implies \alpha(\pi)^1 \vDash \phi$
- ϕ *is* σ^1*-robust if* $\forall \pi \in Tr_{ai} : \pi^1 \vDash \phi \implies \sigma(\pi) \vDash \phi$
- ϕ *is input-universal* (ι) *if* $\forall \pi \in Tr_{ai} : \pi \vDash \phi$

The last notion states that a property holds universally on traces starting with input (for instance: $\neg o_1$ is input-universal). This will sometimes be needed to "fill the gap" between two outputs.

Example 6. Recall the traces $\pi \in Tr_s(M)$ and $\alpha(\pi) \in Tr_{ai}(M)$ of Example 1: $\pi = a/x, a/y, a/x, a/y, \ldots$ and $\alpha(\pi) = a, x, a, y, a, x, a, y, \ldots$. Let $\phi := \mathsf{G}(x \,\mathsf{U}\, y)$. Clearly, $\pi \vDash \phi$, but $\alpha(\pi) \nvDash \phi$, since in the first alternating action "a", neither x nor y holds. So ϕ is not α-robust.

On the other hand, let $\psi = \mathsf{G}(\neg z \,\mathsf{U}\, y)$. Then both $\pi \vDash \psi$ and $\alpha(\pi) \vDash \psi$. Indeed, it will turn out that ψ is robust.

3.2 Robustness Preservation by LTL\X Operators

We will now first check how robustness is preserved by the Boolean connectives conjunction and disjunction, and establish a duality for negation. Subsequently, we will discuss robustness of the atomic properties (cf. Fig. 3). Finally, we will investigate the robustness properties of the until-operator. Robustness of the neXt-operator is left for future research.

	α	σ	α^1	σ^1	ι	M
i	✓	✓	✗	✓	✗	✓
$\neg i$	✓	✓	✓	✗	✗	✓
o	✗	✓	✓	✓	✗	✗
$\neg o$	✓	✗	✓	✓	✓	✗
false	✓	✓	✓	✓	✗	✓
true	✓	✓	✓	✓	✓	✓
input	✓	✓	✗	✓	✓	✓
output	✗	✓	✓	✓	✗	✗

Fig. 3. $\alpha/\alpha^1/\sigma/\sigma^1$-robustness, input-universality and Mealy-robustness for atomic formulas.

All lemmas in this section are summarised in Fig. 4, in the form of a Prolog program. This program can actually be run. For a formula P, if `robust(P)` succeeds, then robustness is guaranteed. However, if the query fails, the property may still be robust. We do not claim that our derivation rules are complete. The soundness of all rules is proved in detail in Appendix A.

First, we establish that α-robustness and σ-robustness are dual, and so are α^1- and σ^1-robustness. Next, all notions of robustness are preserved by $\wedge$ and $\vee$. Also, we claim compositionality of input-universality for $\wedge$ and $\vee$.

Lemma 7 (Boolean connectives).

1. *ϕ is α-robust, if and only if $\neg\phi$ is σ-robust.*
2. *ϕ is α^1-robust, if and only if $\neg\phi$ is σ^1-robust.*
3. *If ϕ, ψ are $\alpha/\sigma/\alpha^1/\sigma^1$-robust, then so are $\phi \wedge \psi$ and $\phi \vee \psi$.*
4. *If ϕ and ψ is input-universal, then so is $\phi \wedge \psi$. If either of ϕ or ψ is input-universal, then also $\phi \vee \psi$ is.*

Next, we check the robustness of atomic properties and their negations. These results are also tabulated in Fig. 3. We also show whether the atomic properties are input-universal, and Mealy-robust.

Lemma 8 (Robustness of atomic properties).

1. *Let $i \in I$. Then i and $\neg i$ are α-robust and σ-robust.*
2. *Let $o \in O$. Then o and $\neg o$ are α^1-robust and σ^1-robust.*
3. *Let $i \in I$. Then i is σ^1-robust and $\neg i$ is α^1-robust.*
4. *Let $o \in O$. Then o is σ-robust and $\neg o$ is α-robust.*
5. *Let $o \in O$. Then $\neg o$ is input-universal.*
6. *Let $i \in I$. Then i and $\neg i$ are Mealy-robust.*

Obviously, o is not α-robust, since it holds in $i/o, \pi$ but not in $i, o, \alpha(\pi)$. Similarly, i is not α^1-robust, since it holds in $i/o, \pi$, but not in $o, \alpha(\pi)$.

```
:- op(500,xfx,[until,and,or]).        % define infix LTL operators
:- op(450,fy,not).                    % define prefix LTL operators
:- op(400,fx,[in,out]).               % define atomic input, output

robust(X) :- alpha(X), sigma(X).      % main predicate

alpha(X) :- member(X,[false,true,input,in _]).
alpha(not X) :- sigma(X).
alpha(X and Y) :- alpha(X), alpha(Y).
alpha(X or Y) :- alpha(X), alpha(Y).
alpha(X until Y) :- alpha(X), alpha1(X), alpha(Y).
alpha(X until Y) :- alpha1(X), iota(X), alpha1(Y).

sigma(X) :- member(X,[false,true,input,output,in _,out _]).
sigma(not X) :- alpha(X).
sigma(X and Y) :- sigma(X), sigma(Y).
sigma(X or Y) :- sigma(X), sigma(Y).
sigma(X until Y) :- sigma(X), sigma(Y), sigma1(Y).
sigma(X until Y) :- sigma1(X), sigma(Y), sigma1(Y).

alpha1(X) :- member(X,[false, true, output, out _]).
alpha1(not X) :- sigma1(X).
alpha1(X and Y) :- alpha1(X), alpha1(Y).
alpha1(X or Y) :- alpha1(X), alpha1(Y).
alpha1(X until Y) :- alpha(X), alpha1(X), iota(X), alpha1(Y).

sigma1(X) :- member(X,[false,true,input,output,in _,out _]).
sigma1(not X) :- alpha1(X).
sigma1(X and Y) :- sigma1(X), sigma1(Y).
sigma1(X or Y) :- sigma1(X), sigma1(Y).
sigma1(X until Y) :- sigma1(X), sigma(Y), sigma1(Y).

iota(X) :- member(X,[true,not false,input,not out _]).
iota(not not X) :- iota(X).
iota(X and Y) :- iota(X), iota(Y).
iota(X or Y) :- iota(X) ; iota(Y).
iota(X until Y) :- iota(Y).
```

Fig. 4. Prolog program for deriving robustness of LTL properties. This program can be viewed as a summary of Lemmas 7–11.

π	i_0/o_0	i_1/o_1	i_2/o_2	i_3/o_3	$\cdots$
$\pi \vDash \phi \,\mathsf{U}\, \psi$	ϕ	ϕ	ϕ	ψ	$\cdots$

$\alpha(\pi)$	i_0	o_0	i_1	o_1	i_2	o_2	i_3	o_3	$\cdots$
Case 1:	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ψ	?	$\cdots$
Case 2:	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	$\boldsymbol{\phi}$	ψ	$\cdots$

Lemma 9

π	i_0	o_0	i_1	o_1	i_2	o_2	i_3	o_3	$\cdots$
Case 1:	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ψ	?	$\cdots$
Case 2:	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ψ	$\cdots$

$\sigma(\pi)$	i_0/o_0	i_1/o_1	i_2/o_2	i_3/o_3	$\cdots$
$\sigma(\pi) \vDash \phi \,\mathsf{U}\, \psi$	ϕ	ϕ	ϕ	ψ	$\cdots$

Lemma 10

Fig. 5. Illustration of subcases of Lemmas 9 and 10

We now get to the main lemmas, first providing the criteria for the α-robustness of until-formulas (cf. Fig. 5, left):

Lemma 9 (α-Robustness of Until-formulas).

1. *Let ϕ be α-robust and α^1-robust; let ψ be α-robust. Then $\phi \mathsf{U} \psi$ is α-robust.*
2. *Let ϕ be α^1-robust and input-universal; let ψ be α^1-robust. Then $\phi \mathsf{U} \psi$ is α-robust and α^1-robust.*

We will now apply Lemmas 8 and 9 to derive α-robustness of basic Until formulas (see also Fig. 6). From Lemma 8 we obtain that $\neg i$ and $\neg o$ are α- and α^1-robust. Also, we obtain that i, $\neg i$ and $\neg o$ are α-robust ($\forall i \in I, o \in O$). Hence, by Lemma 9, Case 1, the first six shapes below are α-robust. Furthermore, by Lemma 8, $\neg i$, o and $\neg o$ are α^1-robust. Note that $\neg o$ is input-universal, since $i\pi \vDash \neg o$ ($\forall o \in O, i \in I, \pi \in Tr_{ao}$). Hence by Lemma 9, Case 2, we obtain that last three shapes below are α-robust. In total this gives 7 α-robust shapes. Only the last three are guaranteed to be α^1-robust.

$$\neg i_1 \mathsf{U}\, i_2 \mid \neg i_1 \mathsf{U} \neg i_2 \mid \neg i_1 \mathsf{U} \neg o_2 \mid \neg o_1 \mathsf{U}\, i_2 \mid \neg o_1 \mathsf{U} \neg i_2 \mid \neg o_1 \mathsf{U} \neg o_2 \mid \neg o_1 \mathsf{U}\, o_2$$

Recall that this means that whenever a synchronous trace π satisfies one of those formulas, the corresponding alternating trace $\alpha(\pi)$ satisfies it as well. The last three are even satisfied by $\alpha(\pi)^1$.

We continue with σ-robustness of Until-formulas (cf. Fig. 5, right).

Lemma 10. *σ-Robustness of Until-formulas*

1. *Let ϕ be σ-robust. Let ψ be both σ-robust and σ^1-robust. Then $\phi \mathsf{U} \psi$ is σ-robust.*
2. *Let ϕ be σ^1-robust. Let ψ be both σ-robust and σ^1-robust. Then $\phi \mathsf{U} \psi$ is σ-robust and σ^1-robust.*

Note that if ϕ is just σ-robust and ψ is both σ- and σ^1-robust, it is not necessary that $\phi \mathsf{U} \psi$ is σ^1-robust. For instance, take $\pi = o_1, i_2, o_2, \ldots \in Tr_{ao}$. Then $\pi \vDash \neg i_1 \mathsf{U}\, o_1$. However, we don't have $i_1/o_1, i_2/o_2, \ldots \vDash \neg i_1 \mathsf{U}\, o_1$.

Since i, $\neg i$, o and $\neg o$ are all σ- or σ^1-robust, and since only i and o are σ^1-robust, we obtain the following 8 σ-robust basic Until formula shapes. Since $\neg i$ is not σ^1-robust, only the last six are also σ^1-robust.

$$\neg i_1 \mathsf{U}\, i_2 \mid \neg i_1 \mathsf{U}\, o_2 \mid i_1 \mathsf{U}\, i_2 \mid i_1 \mathsf{U}\, o_2 \mid o_1 \mathsf{U}\, i_2 \mid o_1 \mathsf{U}\, o_2 \mid \neg o_1 \mathsf{U}\, i_2 \mid \neg o_1 \mathsf{U}\, o_2$$

Lemma 11 (Input-universal Until). *If ψ is input-universal, then $\phi \mathsf{U} \psi$ is input-universal.*

Theorem 12 (Correctness). *If the Prolog program in Fig. 4 derives the goal* `robust(`ϕ`)`*, then ϕ is Mealy-robust.*

Figure 6 indicates the robustness of basic until formulas without nesting. Here ✓ means that robustness can be proved using previous theorems. ✗ only means that the property cannot be proved, but these might still hold for special cases. The last row deserves some attention: If $o_1 = o_2$, then $\neg o_1 \,\mathsf{U}\, \neg o_2 = \neg o_1$, which is really not σ-robust. However, if $o_1 \neq o_2$, then $\neg o_1 \,\mathsf{U}\, \neg o_2 = \mathsf{true}$, since the first action cannot be both o_1 and o_2, so this is σ-robust. Also, $\forall \pi \in Tr_{ai}$: $\pi \vDash \phi \,\mathsf{U}\, \neg o_2$. So $\phi \,\mathsf{U}\, \neg o_2$ is trivially α-robust. Finally, note that $\mathsf{input}\,\mathsf{U}\,\mathsf{output}$ is input-universal, but not recognized by our derivation rules.

All theorems in this section are proved in Appendix A. The theorems in this section can be turned into derivation rules, as presented in the Prolog program in Fig. 4. Given an LTL property P, it tries to derive `robust(P)` by applying the rules, proving α/σ-robustness where necessary. However, this program may fail on some robust formulas since we don't guarantee completeness. Also, it cannot handle formulas that contain the neXt-operator.

Until	α	α^1	σ	σ^1	ι	M
$i_1 \,\mathsf{U}\, i_2$	✗	✗	✓	✓	✗	✗
$i_1 \,\mathsf{U}\, \neg i_2$	✗	✗	✗	✗	✗	✗
$i_1 \,\mathsf{U}\, o_2$	✗	✗	✓	✓	✗	✗
$i_1 \,\mathsf{U}\, \neg o_2$	✗	✗	✗	✗	✓	✗
$\neg i_1 \,\mathsf{U}\, i_2$	✓	✗	✓	✗	✗	✓
$\neg i_1 \,\mathsf{U}\, \neg i_2$	✓	✗	✗	✗	✗	✗
$\neg i_1 \,\mathsf{U}\, o_2$	✗	✗	✓	✗	✗	✗
$\neg i_1 \,\mathsf{U}\, \neg o_2$	✓	✗	✗	✗	✓	✗
$o_1 \,\mathsf{U}\, i_2$	✗	✗	✓	✓	✗	✗
$o_1 \,\mathsf{U}\, \neg i_2$	✗	✗	✗	✗	✗	✗
$o_1 \,\mathsf{U}\, o_2$	✗	✗	✓	✓	✗	✗
$o_1 \,\mathsf{U}\, \neg o_2$	✗	✗	✗	✗	✓	✗
$\neg o_1 \,\mathsf{U}\, i_2$	✓	✗	✓	✓	✗	✓
$\neg o_1 \,\mathsf{U}\, \neg i_2$	✓	✓	✗	✗	✗	✗
$\neg o_1 \,\mathsf{U}\, o_2$	✓	✓	✓	✓	✗	✓
$\neg o_1 \,\mathsf{U}\, \neg o_2$	✓	✓	✗	✗	✓	✗

Fig. 6. $\alpha/\alpha^1/\sigma/\sigma^1$-robustness and input-universality for simple until-properties. The last column concludes whether the property is robust for all Mealy machines.

4 Experiment: Robustness of RERS Constraints

We applied our Prolog program to the constraints of the sequential LTL properties from the RERS 2016 and 2017 problems [4,8] (after a syntactic transformation). Both years featured 9 problems, with 100 LTL properties each. We first filtered out the properties that contain the X-operator. For each property P, we ran the query `robust(P)` in Prolog. The results are displayed in Fig. 7. Here #\X shows the number of X-free formulas; #R shows the number of LTL formulas proven robust.

	2016		2017	
Problem	#\X	#R	#\X	#R
1	35	9	37	15
2	35	9	30	14
3	34	13	26	8
4	31	11	28	16
5	32	11	28	8
6	35	17	28	6
7	18	9	30	16
8	23	9	28	9
9	25	9	30	12
%	30%	37%	29%	39%

Fig. 7. Experiments on the LTL properties from RERS 2016 and RERS 2017.

Apparently, 30% of the formulas was X-free. From these formulas 38% could be established robust. We conclude that this result only partly explains why applying an "alternating model checker" to the "synchronous RERS problems" resulted in a couple of errors only. Either, some formulas are robust but are not recognized by our

method, or they are not robust, but the Mealy machines corresponding to these problems don't distinguish the two interpretations.

5 Conclusion

We introduced *robustness* of LTL properties, which indicates that they are insensitive to their interpretation over *synchronous* or *alternating* traces. We proved a number of derivation rules for robust properties, implemented them in Prolog, and tested them on the RERS 2016 and 2017 challenges. We found that 38% of the X-free LTL properties could be proven robust.

Of course, the model checker should be correct in all cases. We have solved this in RERS 2017 [11] by transforming the Mealy machine M to an incomplete DFA M', introducing an extra state for each edge, in between an input and the subsequent output. On M' we can apply the standard model checking procedure. For the transformed M', we have: $M \vDash_a \phi \iff M' \vDash \phi$. An alternative procedure could be to transform formula ϕ instead, such that $M \vDash_a \phi \iff M \vDash_s \phi'$. We leave the study of the feasibility of this approach for future research.

Future work also includes a complete (precise) characterisation and decision procedure for robust properties. This would also require a study of the neXt-operator. Maybe previous work on stutter-invariant LTL properties can be useful [16]. Another line would be to extend to input-output systems without strict alternation, like I/O-automata. Finally, it would be interesting to consider the robustness of model checking under general action refinement.

Acknowledgement. The authors are supported by the 3TU.BSR project and the TTW project SUMBAT, grant 13859. We thank Mirja van de Pol for carefully reading a preliminary version of this document. We also profited from the numerous suggestions by the anonymous reviewers. Finally, we thank Bernhard Steffen and his team, for their wonderful work in designing, maintaining and sharing the LearnLib, and organising the RERS challenge series.

A Full Soundness Proofs for Robustness Derivation Rules

Lemma 7 (Boolean connectives)

1. ϕ is α-robust, if and only if $\neg\phi$ is σ-robust.
2. ϕ is α^1-robust, if and only if $\neg\phi$ is σ^1-robust.
3. If ϕ, ψ are $\alpha/\sigma/\alpha^1/\sigma^1$-robust, then so are $\phi \wedge \psi$ and $\phi \vee \psi$.
4. If ϕ and ψ is input-universal, then so is $\phi \wedge \psi$. If either of ϕ or ψ is input-universal, then also $\phi \vee \psi$ is.

Proof. 1. $\Rightarrow$. Let ϕ be α-robust. Let $\pi \in Tr_{ai}$ and assume $\pi \vDash \neg\phi$, so $\pi \nvDash \phi$. Note that $\pi = \alpha(\sigma(\pi))$. By α-robustness and contraposition, $\sigma(\pi) \nvDash \phi$, so $\sigma(\pi) \vDash \neg\phi$. Hence $\neg\phi$ is σ-robust.
$\Leftarrow$: Similar, by noting that for $\pi \in Tr_s$, $\pi = \sigma(\alpha(\pi))$.

2. $\Rightarrow$: Let ϕ be α^1-robust. Let $\pi \in Tr_{ai}$. Assume $\pi^1 \vDash \neg\phi$, so $\pi^1 \nvDash \phi$. Note that $\pi^1 = \alpha(\sigma(\pi))^1$ By α^1-robustness and contraposition, $\sigma(\pi) \nvDash \phi$, so $\sigma(\pi) \vDash \neg\phi$. Hence $\neg\phi$ is σ^1-robust.
   $\Leftarrow$: Similar, by noting that for $\pi \in Tr_s$: $\pi = \sigma(\alpha(\pi))$.
3. Holds obviously for $\phi \wedge \psi$ by inspecting the LTL semantics. It follows for $\phi \vee \psi$ by dualities.
4. Trivial.

Lemma 8 (Robustness of atomic properties)

1. Let $i \in I$. Then i and $\neg i$ are α-robust and σ-robust.
2. Let $o \in O$. Then o and $\neg o$ are α^1-robust and σ^1-robust.
3. Let $i \in I$. Then i is σ^1-robust and $\neg i$ is α^1-robust.
4. Let $o \in O$. Then o is σ-robust and $\neg o$ is α-robust.
5. Let $o \in O$. Then $\neg o$ is input-universal.
6. Let $i \in I$. Then i and $\neg i$ are Mealy-robust.

Proof. 1. Let $i \in I$. Let $\pi = i_0/o_0, \pi'$. Note that $\alpha(\pi) = i_0, o_0, \alpha(\pi')$. Then $\pi \vDash i \iff i = i_0 \iff \alpha(\pi) \vDash i$.
2. Let $o \in O$. Let $\pi = i_0/o_0, \pi'$. Note that $\alpha(\pi)^1 = o_0, \alpha(\pi')$. Then $\pi \vDash o \iff o = o_0 \iff \alpha(\pi)^1 \vDash o$.
3. Let $i \in I$ and $\pi \in Tr_{ao}$. Then $\pi = o, \pi'$, so $\pi \nvDash i$. So i is trivially σ^1-robust, and $\neg i$ is α^1-robust by Lemma 7.
4. Let $o \in O$ and $\pi \in Tr_{ai}$. Assume $\pi = i_0, o_0, \pi'$. Then $\pi \nvDash o$ (since I and O are disjoint). So o is trivially σ-robust. Hence $\neg o$ is α-robust by Lemma 7.
5. Any trace in Tr_{ai} is of the form i, o, π', so $\pi \vDash \neg o$.
6. These formulas are both α- and σ-robust, so they agree on the synchronous and alternating traces from any Mealy machine.

Lemma 9 (α-Robustness of Until-formulas)

1. Let ϕ be α-robust and α^1-robust; let ψ be α-robust. Then $\phi \,\mathsf{U}\, \psi$ is α-robust.
2. Let ϕ be α^1-robust and input-universal; let ψ be α^1-robust. Then $\phi \,\mathsf{U}\, \psi$ is α-robust and α^1-robust.

Proof. 1. (cf. case 1 in Fig. 5, left) Let ϕ be α- and α^1-robust and let ψ be α-robust. Let $\pi \in Tr_s$; assume $\pi \vDash \phi \,\mathsf{U}\, \psi$. Then $\exists j : (\forall k < j : \pi^k \vDash \phi) \wedge \pi^j \vDash \psi$. Note that $\forall k : \alpha(\pi^k) = \alpha(\pi)^{2k}$. By α-robustness of ψ, $\alpha(\pi)^{2j} \vDash \psi$. By α- and α^1-robustness of ϕ, for each $k < j$, $\alpha(\pi)^{2k} \vDash \phi$ and $\alpha(\pi)^{2k+1} \vDash \phi$. Hence, for $j' = 2j$, we obtain: $\exists j' : (\forall k < j' : \alpha(\pi)^k \vDash \phi) \wedge \alpha(\pi)^{j'} \vDash \psi$, so $\alpha(\pi) \vDash \phi \,\mathsf{U}\, \psi$.
2. (cf. case 2 in Fig. 5, left) The proof is similar, but now for $j' = 2j + 1$ we obtain $\alpha(\pi)^{j'} \vDash \psi$. For $k' = 2k < j'$, we derive ϕ because it is input-universal. For $k' = 2k+1 < j'$, we derive ϕ because it is α^1-robust. Hence, $\alpha(\pi) \vDash \phi \,\mathsf{U}\, \psi$. In this case, α^1-robustness follows as well (even if $j = 0$).

Lemma 10 (σ-Robustness of Until-formulas)

1. Let ϕ be σ-robust. Let ψ be both σ-robust and σ^1-robust. Then $\phi \,\mathsf{U}\, \psi$ is σ-robust.

2. Let ϕ be σ^1-robust. Let ψ be both σ-robust and σ^1-robust. Then $\phi \mathsf{U} \psi$ is σ-robust and σ^1-robust.

Proof. We first prove the conclusions on σ-robustness, then σ^1-robustness.

- σ-robustness: Let ϕ be σ- or σ^1-robust; let ψ be σ- and σ^1-robust. Let $\pi \in Tr_{ai}$ be given, with $\pi \vDash \phi \mathsf{U} \psi$. Then $\exists j : (\forall k < j : \pi^k \vDash \phi) \wedge \pi^j \vDash \psi$. Note that $\sigma(\pi)^k = \sigma(\pi^{2k})$. If $j = 2j'$ (case 1 in Fig. 5, right), then $\sigma(\pi)^{j'} \vDash \psi$ because ψ is σ-robust. If $j = 2j' + 1$ (case 2 in Fig. 5, right), then $\sigma(\pi)^{j'} \vDash \psi$ because ψ is σ^1-robust. In both cases, for $k' < j'$, we obtain $\sigma(\pi)^{k'} \vDash \phi$ either from $\pi^{2k'}$ (if ϕ is σ-robust), or from $\pi^{2k'+1}$ (if ϕ is σ^1-robust). So indeed $\sigma(\pi) \vDash \phi \mathsf{U} \psi$.
- σ^1-robustness: Similar, but we start with $\pi \in Tr_{ao}$ with $\pi \vDash \phi \mathsf{U} \psi$. We now need σ^1-robustness of ϕ to infer ϕ at the first state of $\sigma(\pi)$.

Lemma 11. If ψ is input-universal, then $\phi \mathsf{U} \psi$ is input-universal.

Proof. Trivial: If $\pi \vDash \psi$ then $\pi \vDash \phi \mathsf{U} \psi$ (at π_0).

Theorem 12 (Correctness). If the Prolog program in Fig. 4 derives the goal `robust(`ϕ`)`, then ϕ is Mealy-robust.

Proof. Note that α- and σ-robustness imply robustness. All other rules of the program correspond to previous lemmas.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
2. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.): Handbook of Model Checking. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8
3. Courcoubetis, C., Vardi, M.Y., Wolper, P., Yannakakis, M.: Memory-efficient algorithms for the verification of temporal properties. Formal Meth. Syst. Des. **1**(2/3), 275–288 (1992)
4. Geske, M., Jasper, M., Steffen, B., Howar, F., Schordan, M., van de Pol, J.: RERS 2016: parallel and sequential benchmarks with focus on LTL verification. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 787–803. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_59
5. Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D., Pasareanu, C.S.: Rigorous examination of reactive systems - the RERS challenges 2012 and 2013. STTT **16**(5), 457–464 (2014)
6. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: a redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 307–322. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_26
7. Isberner, M., Howar, F., Steffen, B.: The open-source LearnLib. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 487–495. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_32
8. Jasper, M., et al.: The RERS 2017 challenge and workshop (invited paper). In: 24th ACM SIGSOFT IS SPIN on Model Checking of Software (SPIN 2017), pp. 11–20 (2017)

9. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: high-performance language-independent model checking. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 692–707. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_61
10. Mealy, G.H.: A method for synthesizing sequential circuits. Bell Syst. Tech. J. **34**(5), 1045–1079 (1955)
11. Meijer, J., van de Pol, J.: Sound black-box checking in the LearnLib. In: Dutle, A., Muñoz, C., Narkawicz, A. (eds.) NFM 2018. LNCS, vol. 10811, pp. 349–366. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77935-5_24
12. Peled, D.A., Vardi, M.Y., Yannakakis, M.: Black box checking. J. Automata Lang. Comb. **7**(2), 225–246 (2002)
13. Pnueli, A.: The temporal logic of programs. In: 18th AS on Foundations of Computer Science (FOCS 1977), pp. 46–57 (1977)
14. van de Pol, J., Ruys, T.C., te Brinke, S.: Thoughtful brute-force attack of the RERS 2012 and 2013 challenges. STTT **16**(5), 481–491 (2014)
15. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: LearnLib: a framework for extrapolating behavioral models. STTT **11**(5), 393–407 (2009)
16. Ben Salem, A.E., Duret-Lutz, A., Kordon, F., Thierry-Mieg, Y.: Symbolic model checking of stutter-invariant properties using generalized testing automata. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 440–454. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_38
17. Steffen, B., Howar, F., Merten, M.: Introduction to active automata learning from a practical perspective. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 256–296. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21455-4_8