Saarland University
Faculty of Mathematics and Computer Science

Bachelor's Thesis

Learning Temporal Properties from Few, Positive Examples

Author
Angelina Göbl

Advisor
Frederik Schmitt

Reviewers
Prof. Bernd Finkbeiner, Ph.D.
Prof. Dr. Hazem Torfah

Submitted: 30th November 2023

Eidesstattliche Erklärung Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 30$^{\text{th}}$ November, 2023

Abstract

Specification mining refers to the problem of automatically inferring formal specifications from data. In contrast to traditional methods, which require both positive and negative examples, we focus on using positive data only, as negative examples are often hard to obtain. The main challenge is presented by the unrestricted search space being caused by using positive examples only. Existing algorithms to mine specifications for temporal properties with only positive trace data require an upper bound on the size of the resulting properties in order to avoid overly specific solutions. In addition, a large amount of examples needs to be provided. We, on the other hand, introduce a statistical learning approach to alleviate the need to provide such a bound by using the Minimum Description Length (MDL) principle for model selection. We focus on mining LTL formulas. Using MDL, we define ranking functions consisting of a simplicity and specificity score. The simplicity score is based on the size of the formula, while the specificity score reasons about the number of models fulfilling the formula. We experience, that our statistical learning approach is able to mine both liveness and safety properties with less data than related work.

Acknowledgements

First of all, I would like to thank Prof. Bernd Finkbeiner for offering me the opportunity to write this thesis. Furthermore, I am grateful for all the support and help of my advisor Frederik over the last months. In addition, I want to thank Prof. Bernd Finkbeiner and Prof. Hazem Torfah for reviewing this thesis. Last but not least, a special thanks goes to my family and friends for supporting me.

Contents

Chapter 1

Introduction

Formal specifications are the key component for formal verification. They are used to precisely describe the desired behavior of a system. However, creating a complete set of formal properties manually and maintaining it over a longer period of time with changes in the system is challenging. In addition, it is easy to miss parts of the specification. Hence, we are interested in generating formal specifications.

Specification mining refers to the problem of inferring formal specifications automatically from trace data, i.e. learning properties of a system in the form of formal specifications. In formal verification, specification mining finds application in model checking and runtime-monitoring [9]. Model checking is a technique that given a system and a formal specification automatically checks, whether the specification is fulfilled. Maintaining the full set of formal specifications is challenging. However, specification mining can be utilized to generate specifications, which enables the application of model checking. Besides its usage in formal verification of reactive systems, specification mining is applied for bug detection and to facilitate interpretability and testing [28].

In the context of specification mining, there is several work taking both positive and negative examples as input into account, which is mentioned in more detail in Chapter 6. However, in many cases, negative examples are hard to obtain. In safety-critical areas, like self-driving cars, producing negative examples means to, for example, hit pedestrians. In addition, for systems, that cannot be fully analyzed due to their size, only traces representing executions can be extracted. Furthermore, if there is only access to a black-box implementation, again only the extraction of positive examples is possible. Therefore, we will focus on using positive examples only for specification mining.

However, working with only positive examples results in an unrestricted search space, ranging from accepting everything to accepting only the set of traces given as examples. Accepting everything results in a simple, yet insufficient specification, as no information of the given data can be inferred from the specification. On the other hand, a specification accepting only the set of traces given as examples is undesirable as well. It is a restrictive specification, however, it is extremely complex, modeling every small detail present in the data. In both settings, the specification does not model the data well.

Specifications, which are too permissive are referred to as overgeneralized. These specifications are overly generic and therefore lacking information. In contrast, there are specifications, which are too restrictive. These are referred to as undergeneralized specifications as they focus too much on the details in the data and are not able to capture the relationships present in the data as they fail to abstract from these details. Over- and undergeneralized specifications both fail to capture the relevant information in the data and are not able to model the relationships between the given examples. The challenge in mining specifications from positive data only lies in learning specifications, which avoid both over- and undergeneralization. Existing algorithms to mine specifications for temporal properties with only positive traces are based on language minimality to avoid undergeneralization. In addition, to prevent overgeneralization, these algorithms require an upper bound on the size of the resulting properties. This avoids overly specific solutions [9, 28]. Here, Roy et al. [28] approached LTL_f specification mining by selecting the most restrictive LTL_f formula out of their search space. Their search space was restricted to the complete set of LTL formulas with a size of at most the predefined upper bound.

We present an approach, which alleviates the need of a bound by using statistical learning. To be more precise, we use the Minimum Description Length (MDL) principle [27] for model selection, as it automatically avoids over- and underfitting and hence, avoids both over- and undergeneralization. MDL was already successfully applied for mining regular expressions from positive examples [24].
Using the MDL approach, we search for specifications, that are simple yet restrictive, i.e. where the specification can be encoded in a short manner and the trace data can be reproduced using the specification without much additional information needed. Therefore, we define the MDL ranking score by using a combination of a simplicity and a specificity term. The simplicity of a specification is measured by the length of the specification. The specificity of a specification measures the restrictiveness of the property. The more restrictive the property, the less additional information is needed to reconstruct the trace data. Hence, we refer to this score as specificity score.

In this work, we focus on mining Linear Temporal Logic (LTL) [25] specifications. LTL is a linear-time specification language, which was introduced in 1977 by Amir Pnueli. The specification mining problem is further divided into two settings. On the one hand, we look into LTL_f specification mining, on the other hand, we deal with traditional LTL. For LTL_f, we distinguish between existential and universal semantics to reason about finite traces. For LTL as well as LTL_f specification mining, sampled LTL formulas represent our search space. The simplicity score is based on the size of the specification, while the specificity score focuses on model counting. For LTL_f, we focus on finite model counting, more precisely on the number of finite traces of a specific length fulfilling the property according to the respective semantics. For LTL, we focus on the asymptotic density and the density of a bound of the LTL formula as defined by Finkbeiner and Torfah [10]. With our approach, we are able to mine a range of liveness and safety properties using only 50 traces. In order to mine the complete benchmark, we incorporate 1,000 traces. In comparison, Ehlers et al. [9] evaluate their approach based on up to 50,000 traces and Roy et al. [28] included 10,000 traces for evaluation. Compared to related work, we need less data to mine these properties. Therefore, we conclude, that our statistical learning approach for specification mining performs well in a setting with few examples.

Chapter 2

Background

In this chapter, we introduce the statistical learning background as well as specification languages and basic definitions, which will be relevant throughout this thesis.

2.1 Statistical Learning

2.1.1 Model Selection, Over- and Undergeneralization

First, we introduce *model selection*.

Definition 2.1.1 (Model Selection)**.** Given some data D, let $\mathcal{H}$ be the set of hypotheses, so the set of models, describing the data D. *Model Selection* refers to the task of selecting one hypothesis $H \in \mathcal{H}$ out of the set of all different model candidates $\mathcal{H}$. The models are evaluated based on model comparison criteria.

In general, model comparison is known to be a difficult task [20]. We experience, that the model, which fits our data the best often does not correspond to the best model. The model fitting the data the best is typically too complex. We refer to such overly complex models as *undergeneralized*. They are not able to model the important relationships present in the data. This is due to the fact, that these models focus too much on details and fail to abstract from these details. Therefore, these models generalize poorly and are often referred to as *overfitted*. On the other hand, focusing on simplicity quickly leads to *overgeneralization*. The models are too permissive, lack information and hence, are unable to represent the given data efficiently as they are overly simple. This behavior is typically referred to as *underfitting*. Overgeneralized as well as undergeneralized models both fail to capture the relevant information in the data. They do not display the regularities in the data and are hence, unable to model relationships present in the data. As a consequence, over- and undergeneralized models are not desirable.

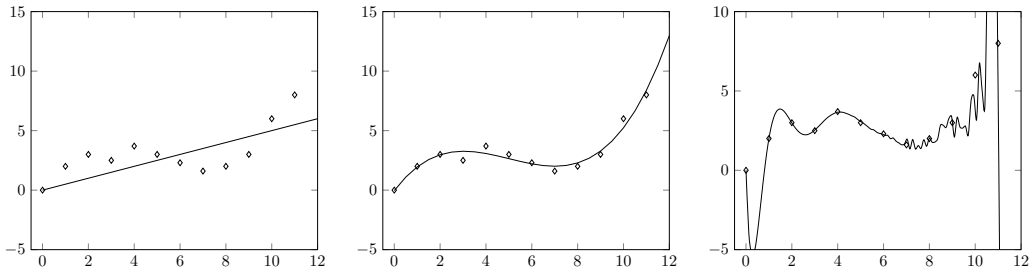

Figure 2.1: polynomials with degree 1 (on the left), 3 (in the middle) and 12 (on the right)

Example 2.1 (Model Selection)**.** Now we look at an example for model selection. Let the data D be the points (x, y) as depicted in Figure 2.1. Imagine, we want to learn the relationship between the x- and y-values and hence find a fitting polynomial. Therefore, let $\mathcal{H}$ be the space of the three hypotheses which are represented in Figure 2.1. We experience, that the linear regression model, which is depicted by the leftmost polynomial, results in a straight line, which does not model the data well as it is overly simple. This polynomial represents an overgeneralized model. On the other hand, there exist polynomials, where all points lay on the function. One example is shown by the rightmost polynomial, which has a degree of 12. This corresponds to a polynomial with the least error, however, it is overly complex. This model is undergeneralized. The model depicted in the middle depicts a polynomial, which is compared to the one on the right, rather simple, but still has a small nonzero error. This polynomial has a degree of 3. Out of these three polynomials, the model in the middle represents the best, as it is fairly simple but still has a good fit.

Therefore, a model selection criteria should be chosen which avoid over- as well as undergeneralization. Next, we will look into a model selection principle, which avoids over- as well as undergeneralization automatically.

2.1.2 Minimum Description Length Principle

The *Minimum Description Length (MDL) principle* is a model selection principle, which was introduced by Jorma Rissanen in *Modeling by the shortest data description* [27] in 1978. The main idea of MDL is to minimize the amount of information necessary to describe the data in a lossless manner.

Definition 2.1.2 (Two-Part Data Description)**.** Given some data D, let $\mathcal{H}$ be the set of hypotheses matching the data D. Consider a *two-part data description*, which

1. describes the *hypothesis* $H \in \mathcal{H}$, which is a model and aims to describe regularity

present in the data.

2. describes the *data D given the hypothesis H* and consists of all the information, which is additionally needed to reproduce the data given the hypothesis. In the following, we refer to this part of the description as data details.

The *best hypothesis* H^* of the set of hypotheses $\mathcal{H}$ given the data D corresponds to the shortest data description and is defined as:

$$H^* = \arg\min_{H \in \mathcal{H}} (L(H) + L(D|H)) \tag{2.1}$$

where $L(H)$ and $L(D|H)$ model the code length needed to respectively describe the hypothesis H and the data details. Well-known results from information theory relate the code length $L(x)$ to probabilistic models over the event x [20]. The results state, that the optimal code length is given by the Shannon information content:

$$L(x) = -\log_2(P(x))$$

Conversely, from the optimal code length, the probability is implicitly defined as:

$$P(x) = 2^{-L(x)}$$

$P(H)$ models the *prior* over the hypothesis space. In general, we are interested in short and hence, simple hypotheses. Hence, the higher the probability, the simpler is the hypothesis. Therefore, the prior can be interpreted as the *simplicity* of the hypothesis.

$P(D|H)$ models the *probability density* of the data according to the hypothesis H. The probability density $P(D|H)$ can be interpreted as evidence for the data. The higher the probability, the more precise the hypothesis is modeling the data and hence, the more specific is the hypothesis. Therefore, the probability density $P(D|H)$ corresponds to the *specificity* of the hypothesis.

Consequently, the combination of the hypothesis and data term leads to a trade-off between specificity and simplicity, which prevents over- as well as under-generalization. Due to the connection of over- and under-generalization to under- and overfitting, MDL protects against under- and overfitting.

MDL is closely related to Bayesian model comparison. Besides MDL, *Bayesian model comparison* represents another important model selection principle. The Bayesian model comparison states, that the model maximizing the posterior probability is the best model. The posterior probability is defined as:

$$P(H|D) = \frac{P(H) \cdot P(D|H)}{P(D)}$$

More details on the relationship between MDL and Bayesian model comparison are given in [20].

2.2 Specification Languages

2.2.1 Regular Expressions

A *regular expression* over the alphabet Σ is built from the constants $\emptyset$ and ϵ, the operators concatenation $\cdot$, union $+$ and the Kleene star * as well as from symbols $\sigma \in \Sigma$.

Definition 2.2.1 (Syntax of regular expressions)**.** Let Σ be the alphabet. Regular expressions over the alphabet Σ are inductively defined:

$$\varphi ::= \epsilon \mid \emptyset \mid \sigma \mid \varphi^* \mid \varphi \cdot \varphi \mid \varphi + \varphi$$

with $\sigma \in \Sigma$. The constants ϵ and $\emptyset$ are representing the set $\{\epsilon\}$ and the empty set.

Definition 2.2.2 (Semantics of regular expressions)**.** Let φ be a regular expression. The *language* $\mathcal{L}(\varphi)$ over the alphabet Σ is defined inductively:

- if $\varphi = \epsilon$ then $\mathcal{L}(\varphi) = \mathcal{L}(\epsilon) = \{\epsilon\}$
- if $\varphi = \emptyset$ then $\mathcal{L}(\varphi) = \mathcal{L}(\emptyset) = \emptyset$
- if $\varphi = \sigma$ with $\sigma \in \Sigma$ then $\mathcal{L}(\varphi) = \mathcal{L}(\sigma) = \{\sigma\}$
- if $\varphi = \varphi_1^*$ then $\mathcal{L}(\varphi) = \mathcal{L}(\varphi_1^*) = \{u_1 u_2 ... u_n \mid n \in \mathbb{N}, u_i \in \mathcal{L}(\varphi_1) \text{ for all } 0 \leq i \leq n\}$
- if $\varphi = \varphi_1 \cdot \varphi_2$ then $\mathcal{L}(\varphi) = \mathcal{L}(\varphi_1 \cdot \varphi_2) = \{uv \mid u \in \mathcal{L}(\varphi_1), v \in \varphi_2\}$
- if $\varphi = \varphi_1 + \varphi_2$ then $\mathcal{L}(\varphi) = \mathcal{L}(\varphi_1 + \varphi_2) = \mathcal{L}(\varphi_1) \cup \mathcal{L}(\varphi_2)$

If a language over finite words is definable by a regular expression, it is *regular*.

2.2.2 ω-Regular Expressions

ω-Regular expressions are an extension of regular expressions to infinite words. To be more precise, ω-regular expressions only allow infinite words. Finite words as well as the empty word ϵ are excluded from ω-regular expressions.

An ω-regular expression over the alphabet Σ is build from regular expressions over Σ as well as from the operators union $+$ of two ω-regular expressions, the infinite repetition $^\omega$ over regular expressions as well as the concatenation of a regular with an ω-regular expression.

Definition 2.2.3 (Syntax of ω-regular expressions)**.** Let Σ be the alphabet. Let R be the set of all regular expressions over Σ. ω-regular expressions over the alphabet Σ are inductively defined by

$$\varphi ::= r_1^\omega \mid r \cdot \varphi \mid \varphi + \varphi$$

with $r \in R$ and $r_1 \in R$ with $\epsilon \notin \mathcal{L}(r_1)$.

Definition 2.2.4 (Semantics of ω-regular expressions)**.** Let φ be a ω-regular expression. Let R be the set of all regular expressions over Σ. The language $\mathcal{L}(\varphi)$ over the alphabet Σ is defined inductively:

- if $\varphi = r_1^\omega$ with $r_1 \in R$ then $\mathcal{L}(\varphi) = \mathcal{L}(r_1^\omega) = \mathcal{L}(r_1)^\omega$
  with $L^\omega = \{w_1 w_2 ... \mid w_i \in L \text{ with } |w_i| > 0 \text{ for all } i \in \mathbb{N}\}$ for $L \subseteq \Sigma^*$ with $\epsilon \notin L$
- if $\varphi = r \cdot \varphi_1$ then $\mathcal{L}(\varphi) = \mathcal{L}(r \cdot \varphi_1) = \mathcal{L}(r) \cdot \mathcal{L}(\varphi_1)$
  with $\mathcal{L}_r \cdot \mathcal{L}_\omega = \{\alpha\beta \mid \alpha \in \mathcal{L}_r, \beta \in \mathcal{L}_\omega\}$ for $\mathcal{L}_r \subseteq \Sigma^*$ and $\mathcal{L}_\omega \subseteq \Sigma^\omega$
- if $\varphi = \varphi_1 + \varphi_2$ then $\mathcal{L}(\varphi) = \mathcal{L}(\varphi_1 + \varphi_2) = \mathcal{L}(\varphi_1) \cup \mathcal{L}(\varphi_2)$

If a language over infinite words is definable by a ω-regular expression, it is *ω-regular*.

2.2.3 Linear Temporal Logic

Linear Temporal Logic (LTL) is a widely used specification language for linear-time properties. It was introduced by Amir Pnueli in 1977 [25]. The definitions of this section are based on Baier and Katoen's book *Principles of Model Checking* [3].

An LTL formula is built from its atomic propositions, the boolean connectives conjunction $\wedge$ and negation $\neg$ as well as the temporal operators Next $\bigcirc$ and Until $\mathcal{U}$. The LTL formula $\bigcirc\varphi$ states, that φ must hold at the next step. $\varphi_1 \mathcal{U} \varphi_2$ states, that φ_2 must hold at some point in time, however until this point is reached, φ_1 has to hold.

Definition 2.2.5 (Syntax of LTL Formulas)**.** An *LTL formula* over the set AP of atomic propositions is defined by the grammar

$$\varphi ::= true \mid a \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi \mathcal{U} \varphi$$

with $a \in AP$.

Additionally, there exist boolean and temporal operators, which can be derived by the grammar and are therefore excluded. We allow the temporal operators Finally $\Diamond$ and Globally $\Box$. The formula $\Diamond\varphi$ states, that φ must hold at some point in time, while $\Box\varphi$ states, that φ has to hold for the complete trace. Finally $\Diamond$ and Globally $\Box$ can be derived by the equations

$$\Diamond\varphi = \text{true } \mathcal{U} \varphi \quad \text{and} \quad \Box\varphi = \neg\Diamond\neg\varphi.$$

Besides negation and conjunction, there exist conventional propositional logic opera-

tors like $\vee, \rightarrow, \leftrightarrow, \oplus$, these can be derived by the following equations:

$$\begin{aligned}\varphi_1 \vee \varphi_2 &= \neg(\neg\varphi_1 \wedge \neg\varphi_2)\\ \varphi_1 \rightarrow \varphi_2 &= \neg\varphi_1 \vee \varphi_2\\ \varphi_1 \leftrightarrow \varphi_2 &= (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)\\ \varphi_1 \oplus \varphi_2 &= (\varphi_1 \wedge \neg\varphi_2) \vee (\neg\varphi_1 \wedge \varphi_2)\end{aligned}$$

Definition 2.2.6 (Semantics of LTL Formulas)**.** Let φ be an LTL formula over AP. The *language* over the alphabet 2^{AP} is induced by

$$\mathcal{L}(\varphi) = \{\sigma \in 2^{AP} \mid \sigma \models \varphi\}$$

where $\models$ corresponds to the smallest relation satisfying the following properties:

$\sigma \models \text{true}$	
$\sigma \models a$	iff $a \in \sigma(0)$
$\sigma \models \varphi_1 \wedge \varphi_2$	iff $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$
$\sigma \models \neg\varphi$	iff $\sigma \not\models \varphi$
$\sigma \models \bigcirc\varphi$	iff $\sigma[1,\infty] = \sigma(1)\sigma(2)\sigma(3)... \models \varphi$
$\sigma \models \varphi_1 \,\mathcal{U}\, \varphi_2$	iff $\exists j \geq 0.\ \sigma[j,\infty] \models \varphi_2$ and for all $0 \leq i < j\ \sigma[i,\infty] \models \varphi_1$

The infinite sequences $\sigma \in (2^{AP})^\omega$ are called *traces*. Hence, $\mathcal{L}(\varphi)$ describes the set of all traces satisfying the LTL formula φ.

Definition 2.2.7 (LTL Properties)**.** We now introduce syntactic LTL classes.

- A *bounded-safety* property φ describes a set of infinite words, which are expressable in LTL using only $\bigcirc$ as a temporal operator.
- An *invariant* property φ describes a set of infinite words, which are expressable in LTL by the formula $\Box\psi$, where ψ is a bounded-safety property.
- A *guarantee* property φ describes a set of infinite words, which are expressable in LTL by the formula $\Diamond\psi$, where ψ is a bounded-safety property.
- A *persistence* property φ describes a set of infinite words, which are expressable in LTL by the formula $\Diamond\Box\psi$, where ψ is a bounded-safety property.
- A *response* property φ describes a set of infinite words, which are expressable in LTL by the formula $\Box\Diamond\psi$, where ψ is a bounded-safety property.

2.3 Automata

Next, we define *automata over infinite words*, as they represent one opportunity to express LTL as well as ω-regular properties. The definitions are based on [11].

Definition 2.3.1 (Automata over infinite words)**.** An *automaton over infinite words* is a five-tuple $\mathcal{A} = (\Sigma, Q, I, T, Acc)$, where

- Σ is the finite alphabet
- Q is the finite set of states
- $I \subseteq Q$ is the set of initial states
- $T \subseteq Q \times \Sigma \times Q$ is the set of transitions and
- $Acc \subseteq Q^\omega$ is the accepting condition.

Let $A = (\Sigma, Q, I, T, Acc)$ be an automaton over infinite words and let $\sigma = \alpha_0\alpha_1 \ldots \in \Sigma^\omega$ be an infinite word. A *run* of the infinite word σ on the automaton A is an infinite sequence of $r = q_0q_1 \ldots \in Q^\omega$ such that $q_0 \in I$ and for all $i \in \mathbb{N}$, $(q_i, \alpha_i, q_{i+1}) \in T$. A run $r = q_0q_1 \ldots$ is called *accepting*, if it is contained in the accepting condition, so if $r \in Acc$. An infinite word σ is accepted by A, if there exists an accepting run r of A for σ. The *language recognized* by the automaton A is defined as $\mathcal{L}(A) = \{\sigma \in \Sigma^\omega \mid A \text{ accepts } \sigma\}$

The infinity set $\mathrm{Inf}(r)$ of a run r, captures all states, which are visited infinitely often. It is defined as

$$\mathrm{Inf}(r) = \{q \in Q \mid \forall i \exists j > i . q_j = q\}$$

Definition 2.3.2 (Büchi Automaton)**.** A *Büchi automaton* is an automaton $A = (\Sigma, Q, I, T, \mathrm{B\ddot{U}CHI(F)})$ over infinite words with the accepting condition BÜCHI(F), where $F \subseteq Q$. The Büchi acceptance condition BÜCHI(F) over the set of states $F \subseteq Q$ is defined as

$$\mathrm{B\ddot{U}CHI(F)} = \{r \in Q^\omega \mid \mathrm{Inf}(r) \cap F \neq \emptyset\}.$$

A run of the Büchi automaton is called *accepting* if one state of F is visited infinitely often. F is called the set of accepting states.

Definition 2.3.3 (Parity Automaton)**.** A *parity automaton* is an automaton $A = (\Sigma, Q, I, T, \mathrm{PARITY(c)})$ over infinite words with the accepting condition PARITY(c). The parity acceptance condition PARITY(c) for a coloring function $c : Q \to \mathbb{N}$ is defined by

$$\mathrm{PARITY(c)} = \{r \in Q^\omega \mid \max\{c(q) \mid q \in \mathrm{Inf}(r)\} \text{ is even}\}.$$

A run of the parity automaton is called *accepting*, if of all states, which are visited infinitely often, the highest color of these states is even. If the coloring function of a parity automaton only contains the colors 1 and 2, so if for all $q \in Q$, $c(q) \in \{1, 2\}$, then the parity automaton represents a Büchi automaton. This Büchi automaton has the accepting condition $F = \{q \in Q \mid c(q) = 2\}$. An automaton is called complete if $|(q, \sigma, q') \in T| \geq 1$ for every $q \in Q$ and every $\sigma \in \Sigma$. This means, that for every state and every action, there exists at least one edge to a successor state. An automaton is called *deterministic* if $|I| \leq 1$ and $|(q, \sigma, q') \in T| \leq 1$ for every $q \in Q$ and $\sigma \in \Sigma$, otherwise it is called *nondeterministic*.

A *strongly connected component* (SCC) in the automaton corresponds to a strongly connected component of the automaton-induced graph. We call a component of a graph strongly connected if, for any vertex in the graph, every vertex is reachable. A strongly connected component is called *terminal* if none of the states in the component has a transition leaving the component. Furthermore, an SCC of a parity automaton is called *accepting*, if the highest color of all states of the component is even.

2.4 Linear-Time Properties

The following notions are based on Finkbeiner and Torfah [10].

Definition 2.4.1 (Linear-Time Property)**.** A *linear-time property* φ over the alphabet Σ denotes a set of infinite words $\varphi \subseteq \Sigma^\omega$. We call the elements of φ *models* of φ. The complement set $\bar{\varphi} = \Sigma^\omega \setminus \varphi$ denotes the set of *non-models* of φ.

A word w over the alphabet Σ is called *ultimately periodic*, if it has the form $u \cdot v^\omega$ with $u \in \Sigma^*$ and $v \in \Sigma^+$.

Example 2.2 (Linear-Time Property)**.** Let the linear-time property φ describe the set of infinite words with infinitely many a's and let $AP = \{a, b\}$ be the set of atomic propositions. This property can be expressed in LTL with the formula $\varphi = \square\diamondsuit a$. Now the infinite word $(\{a\}\{\})^\omega$ represents a model of φ, while $\{\}^\omega$ is a non-model.

A *lasso* over the alphabet Σ of length n is given by a pair $(u, v) \in \Sigma^* \times \Sigma^+$ with $|u \cdot v| = n$ and $|v| > 0$, which induces the ultimately periodic word $u \cdot v^\omega$.

Definition 2.4.2 (n-Model)**.** A lasso $(u, v) \in \Sigma^* \times \Sigma^+$ of length n is called an *n-model* for the property φ over the alphabet Σ, if $u \cdot v^\omega \in \varphi$. We refer to n as the bound of the n-model. The set of all n-models of φ induces the language $L_n\varphi$ for the bound n. The complement language $\overline{L_n}\varphi$ for a bound n captures the set of *n-non-models* of φ. The *cardinality* $\#_\varphi(n)$ of a property φ for a bound n denotes the size of the language $L_n(\varphi)$.

Example 2.3 (n-Model)**.** Let the linear-time property φ describe the set of infinite words with infinitely many a's and let $AP = \{a, b\}$ be the set of atomic propositions. The ultimately periodic word $\{\}\{\}(\{a\}\{\}\{a, b\})^\omega$ represents a n-model of φ for $n = 5$. While the word $\{\}\{\}\{a\}(\{\}\{b\})^\omega$ is a n-non-model of φ for $n = 5$.

Definition 2.4.3 (Good-prefix)**.** Given an infinite word $\sigma = \alpha_0 \ldots \in \Sigma^\omega$, the prefix $\alpha_0 \ldots \alpha_i$ is denoted by $\sigma[\ldots i]$. A finite word $w = \alpha_0 \ldots \alpha_i$ is referred to as *good-prefix* for a property φ, if every infinite word σ with $\sigma[\ldots i] = \alpha_0\alpha_1 \ldots \alpha_i$ is a model of φ. $Good(\varphi)$ denotes the set of all good-prefixes for the property φ.

Example 2.4 (Good-Prefix)**.** We look at a few examples of good-prefixes:

- Given the property $\varphi = a \,\mathcal{U}\, b$, a good-prefix is represented by $\{a\}\{a\}\{b\}$.
- Given the property $\varphi = \Diamond a$, a good-prefix is represented by $\{\}\{a\}$.

Definition 2.4.4 (Bad-prefix)**.** A finite word $w = \alpha_0\alpha_1 \ldots \alpha_i$ is referred to as *bad-prefix* for a property φ, if every infinite word σ with $\sigma[\ldots i] = \alpha_0\alpha_1 \ldots \alpha_i$ is a non-model of φ. $Bad(\varphi)$ denotes the set of all bad-prefixes for the property φ.

Example 2.5 (Bad-Prefix)**.** We look at a few examples of bad-prefixes:

- Given the property $\varphi = a \,\mathcal{U}\, b$, a bad-prefix is represented by $\{\}$.
- Given the property $\varphi = \Box a$, a bad-prefix is represented by $\{a, b\}\{a\}\{b\}$.

Definition 2.4.5 (Safety Property)**.** A *safety property* describes a property φ, where every non-model of φ has a bad-prefix.

Definition 2.4.6 (Liveness Property)**.** A *liveness property* describes a property φ, where for every finite word $w = \alpha_0 \ldots \alpha_i$, there exists an infinite word σ with $\sigma[\ldots i] = \alpha_0 \ldots \alpha_i$, such that σ is a model of φ.

Example 2.6 (Safety and Liveness Properties)**.** We now represent safety as well as liveness properties:

- $\Box a$ is a safety property as for every non-model of $\Box a$, there exists a bad-prefix.
- $\Diamond a$ is liveness property. We can extend every finite word with an infinite continuation, which contains an a. This combination of the finite prefix and the infinite continuation represents a model of the property.
- *true* is the only property, which is a liveness and a safety property.
- $a \,\mathcal{U}\, b$ is neither a safety nor a liveness property. The trace $\{a\}^\omega$ is not fulfilling the property, but there does not exist a bad-prefix for this trace, therefore it

cannot be a safety property. It is no liveness property either, since $\emptyset$ is a bad-prefix and hence we are not able to find an extension for the prefix to fulfill $a \,\mathcal{U}\, b$

2.5 Density of Linear-Time Properties

In this section, we focus on the *density* of linear-time properties over the space of ultimately periodic words, which was introduced by Finkbeiner and Torfah [10].

Definition 2.5.1 (Density)**.** The density of a linear-time property φ over an alphabet Σ and a bound n describes the ratio of the cardinality of φ for n to the total number of ultimately periodic words uv^ω of with $|u| + |v| = n$:

$$\nabla_\varphi(n) = \frac{\#_\varphi(n)}{n \cdot |\Sigma|^n}$$

Definition 2.5.2 (Asymptotic density)**.** The asymptotic density of the linear-time property φ corresponds to the value

$$\lim_{n \to \infty} \nabla_\varphi(n)$$

in case the value exists. We denote the asymptotic density by ∇^∞_φ.

The density for φ over n does not always converge for linear-time properties, hence the asymptotic density does not always exist. However, for ω-regular expressions we experience, that the asymptotic density is always convergent. As LTL is strictly less expressive than ω-regular expressions, the asymptotic density converges for LTL as well.

Example 2.7 (Density of Properties)**.** Now we look at the density of some properties:

- $a \,\mathcal{U}\, b$ has an asymptotic density of $\frac{2}{3}$ since the increase in the number of models is twice as high as the increase in the number of non-models.
- $a \wedge \bigcirc b$ has a density equal to $\frac{1}{4}$ for a bound greater than 1.
- For every bounded-safety property φ, which is not equal to true or false, the asymptotic density converges to ϵ, with $\epsilon \in (0, 1)$.
- Every invariant and persistence property φ has an asymptotic density of 0.
- Every guarantee and response property φ has an asymptotic density of 1.

2.6 Discrete-Time Markov Chains

In this section, we introduce *discrete-time Markov chains* (DTMCs). The information is based on Harchol-Balter's book *Introduction to Probability for Computing* [17].

Definition 2.6.1 (Stochastic process)**.** A *stochastic process* describes a sequence of random variables $X_0, X_1, \ldots$ denoted by $\boldsymbol{X} = \{X_n | n \in \mathbb{N}\}$.

Definition 2.6.2 (Markov Property)**.** The *Markov property* expresses the condition that the future state X_{n+1} is given the past states $X_0, X_1, \ldots X_{n-1}$ and the current state X_n, independent of the past states and only dependent on the current state X_n. Intuitively, the Markov property states that the future is, given the present, independent of the past.

Definition 2.6.3 (Discrete-time Markov chains)**.** A *discrete-time Markov chain* (DTMC) describes a stochastic process $\{X_n | n \in \mathbb{N}\}$ with the Markov property. Formally, a stochastic process $\{X_n\}$ is called a Markov chain if for all time steps $n \geq 0$ and all states $i_0, i_1, \ldots, i_{n-1}, i$ and $j \in \mathbb{Z}$ it holds that:

$$
\begin{aligned}
&\boldsymbol{P}\{X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots X_0 = i_0\} \\
=&\boldsymbol{P}\{X_{n+1} = j | X_n = i\} && \text{Markov property} \\
=&P_{ij}
\end{aligned}
$$

P_{ij} represents the transition probability of moving from i to j independent of the time step n and the past states $X_0, X_1, \ldots X_{n-1}$.

Example 2.8 (Markov Chain)**.** We now look at the weather prediction described as a Markov chain in Figure 2.2. The random variables X_i with $i \in \mathbb{N}$ are represented as triple. They describe for each state the probability of being in this state at time step i. The random variable X_0 represents the initial distribution. Possible initial distribution are for example $X_0 = (1.0, 0.0, 0.0)$ and $X_0 = (0.4, 0.3, 0.3)$. The initial distribution $X_0 = (1.0, 0.0, 0.0)$ states, that we start at a time, when it is sunny, while the initial distribution $X_0 = (0.33, 0.33, 0.33)$ states, that the probability of the weather events is equally likely. The Markov chain has the Markov property, as the transitions are only dependent on the current state. To be more precise, if we are, for example, in the state *sunny*, the probability of transitioning to the state *cloudy*, is only dependent on the fact, that we are currently in state *sunny*. Whether we reached the state *sunny* via the state *sunny*, *cloudy* or *rainy* does not influence the probability of the current move.

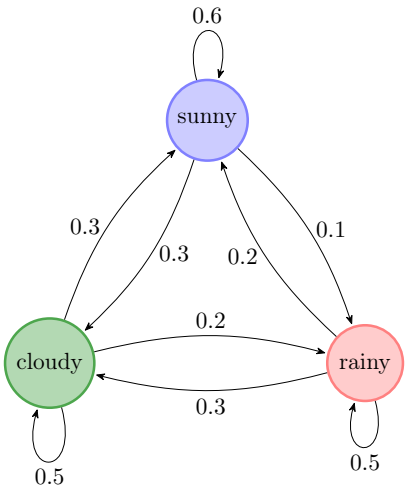

Figure 2.2: Markov Chain for weather forecast

Definition 2.6.4 (Transition Probability Matrix)**.** The *transition probability matrix* if a DTMC is a matrix $\boldsymbol{P}$, where the entry P_{ij} states the probability of moving from state i to state j on the following transition, given i is the current state.

Example 2.9 (Transition Probability Matrix)**.** We now construct the transition probability matrix for Figure 2.2. It is given by:

$$\boldsymbol{P} = \begin{pmatrix} 0.6 & 0.3 & 0.1 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{pmatrix}$$

The values of the first row correspond to the probabilities of moving from the state *sunny* to *sunny*, *cloudy* and *rainy* from left to right. The second row corresponds to the probabilities of moving from state *cloudy* and the last row represents the values for the state *rainy*.

Chapter 3

Linear Temporal Logic Specification Mining

In this chapter, we introduce the Linear Temporal Logic specification mining setting with its challenges. As we focus on learning a specification from positive data only, we have an unrestricted search space. Compared to a setting, that takes both positive and negative data into account, there are more possible specifications describing the given data. Especially, avoiding both over- and undergeneralization is even more important due to the larger search space. Therefore, the challenges are represented by defining the search space as well as a ranking function. Since we are interested in testing and evaluating our approach efficiently, we take an LTL formula as input and sample accepting traces for the property. These traces are taken into account to model the search space. The search space represents the model candidates, which are taken into account for the specification mining task. The ranking function assesses the models of the search space. This model selection is based on the Minimum Description Length principle. Important to notice is that we focus on specification mining for LTL_f as well as traditional LTL. Before we get into the challenges, we first define the semantics for LTL_f.

3.1 Semantics of LTL_f

As we focus on LTL over finite traces (LTL_f), we define the LTL_f semantics in order to evaluate whether an LTL_f formula holds on a trace. Based on the semantics, we define the notion of *non-empty states*, which finds application in the implementation we present in Chapter 4. For the semantics, we focus on the notion of good as well as non-bad-prefixes.

Definition 3.1.1 (Existential Semantics)**.** Given a finite trace u and an LTL_f formula

φ, the LTL_f formula φ holds on the finite trace u, if and only if there exists an infinite word σ with $\sigma[\dots i] = u$ such that σ is a model of φ. To be more precise, the LTL_f formula φ holds on the finite trace u, if and only if u is not a bad-prefix for the LTL formula φ.

Definition 3.1.2 (Universal Semantics)**.** Given a finite trace u and an LTL_f formula φ, the LTL_f formula φ holds on the trace u, if and only if u represents a good-prefix for the LTL formula φ.

To be more precise, this means for the universal semantics, that for a finite trace u with $|u| = i$ every infinite word σ with $\sigma[\dots (i-1)] = u$ is a model of the traditional LTL formula φ.

Important to notice is, that for the existential semantics, liveness properties, for example, $\Diamond a$ or $\Diamond\Box a$ will always evaluate to true. On the other hand, for the universal semantics, properties with conditions on infinitely many positions, for example, $\Box a$ always evaluate to false.

Example 3.1 (LTL_f Semantics)**.** Let $\varphi = \Diamond a$ be the given LTL formula and $tr = \{\}\{\}$ be a finite trace. According to the existential semantics, the formula φ holds for the trace tr, as tr is a non-bad-prefix. To be more precise, the infinite trace $\{\}\{\}(\{a\})^\omega$ represents a model for φ evaluated as a traditional LTL formula and hence, tr cannot be a bad-prefix. On the other hand, for the universal semantics, φ does not hold for tr, as tr does not represent a good-prefix, as for example $\{\}\{\}(\{\})^\omega$ is a non-model for φ.

Now, we look at a way of encoding the semantics via the states of a parity automaton.

Definition 3.1.3 (Non-Empty States)**.** Let φ be an LTL formula and let *sem* be either the existential or the universal semantics. We compute a deterministic parity automaton $\mathcal{A}$ from the LTL formula φ. We call a state s of the automaton a *non-empty state* if every finite path $r = p_0 p_1 \dots p_{i-1}$ of A with length i and $p_{i-1} = s$ is accepted according to the semantics *sem*.

Intuitively speaking, given a DPA and semantics, a state is called a non-empty state, if traces ending in this state are accepted according to the semantics.

Example 3.2 (Non-Empty States)**.** Let $\varphi = a \,\mathcal{U}\, b$ be an LTL formula. A deterministic parity automaton for φ is displayed in Figure 3.1. The state q_0 represents the initial state, and the coloring function is given by

$$c(q) = \begin{cases} 1 & \text{if } q \in \{q_0, q_2\} \\ 2 & \text{if } q = q_1 \end{cases}.$$

For the existential semantics $\{q_0, q_1\}$ represents the set of non-empty states, while for the universal semantics, the non-empty states are given by $\{q_1\}$.

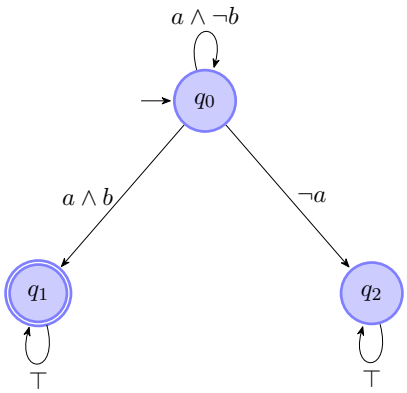

Figure 3.1: DPA for $\varphi = a \,\mathcal{U}\, b$

Example 3.3 (Non-Empty States)**.** Let $\varphi = \Box a$ be an LTL formula. A deterministic parity automaton for φ is displayed in Figure 3.2. The state q_0 represents the initial state, and the coloring function is given by

$$c(q) = \begin{cases} 1 & \text{if } q = q_0 \\ 2 & \text{if } q = q_1 \end{cases}.$$

For the existential semantics $\{q_0\}$ represents the non-empty state, while for the universal semantics, there exists no non-empty state.

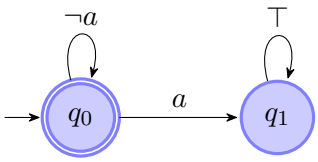

Figure 3.2: DPA for $\varphi = \Box a$

The implementation for the non-empty states is explained in Section 4.1.

Next, we focus on the properties of non-empty states for the existential semantics. For the existential semantics, non-empty states for a property φ in the DPA, correspond to the states, where there exists an infinite path, that is accepted by φ. To be more precise, for every state in the non-empty states, there still exists a satisfying infinite path through the automaton. For properties, that are not satisfiable, there do not exist non-empty states. In addition, accepted finite paths through the automaton only visit non-empty states. As soon, as a finite path visits states, which are not included in the non-empty states, the path is not accepted by the LTL formula in the

existential semantics. Furthermore, there exists no infinite extension for this prefix, which is accepted by the LTL formula. Therefore, we experience, that the initial state of satisfiable specifications has to be part of the non-empty states, as otherwise, we would not be able to reach non-empty states.

Now, we consider the properties of non-empty states for the universal semantics. For the universal semantics, the non-empty states for a property φ, correspond to states, where every outgoing path is accepted by φ. Compared to the existential semantics, there exist properties, e.g. $\square a$, which are satisfiable, but still have no non-empty states in the universal semantics. For these properties, there do not exist traces in the universal semantics. In addition, while for the existential semantics, the initial state is always part of the non-empty states, for the universal semantics, the initial state is never part of the non-empty states, except for LTL formulas being equivalent to *true*. However, as soon as we reach a non-empty state in the universal setting, we will only visit non-empty states.

3.2 Trace Sampling and Search Space

We now look into the trace sampling and the search space generation. First, we focus on the trace sampling process. We start the trace sampling by taking an LTL formula and atomic propositions. After that, we sample the traces with these atomic propositions satisfying the LTL formula. These traces are dependent on the setting in the form of ultimately periodic words or finite words. For the finite words, we sample depending on the semantics, traces which are accepted by the universal or existential semantics.

Representing the search space for an LTL formula given traces is challenging. Therefore, given the atomic propositions, we randomly sample LTL formulas. These formulas together with all their subformulas represent the whole space of candidates. Next, we filter out the formulas, which do not fulfill the sampled traces. After that, we are only left with formulas, which hold on all the traces. These formulas are the final candidates for model selection.

In order to sample traces and create the search space, we focus on the notion of *non-empty states*. The implementation for the trace sampling is explained in Section 4.2 and the implementation for the search space generation in Section 4.3.

3.3 Ranking Function

Defining a suitable ranking function, which avoids over- and undergeneralization, is essential for successful model selection. Therefore, we focus on balancing simplicity and specificity by using the Minimum Description Length (MDL) principle [27]. We

define different ranking functions, however, each consists of one score measuring the specificity and one measuring the simplicity. For specificity, we focus on the restrictiveness of an LTL formula, more precisely on the number of models for the LTL formula. For the simplicity score, we consider the size of the formula.

3.3.1 Simplicity Terms

For the simplicity score, we focus on the notion of size. We denote the size of the LTL formula as follows:

$$size(\varphi) = \begin{cases} 1 & \text{if } \varphi = a, a \in APS \\ 1 & \text{if } \varphi = true \vee \varphi = false \\ 1 + size(\varphi_1) & \text{if } \varphi = \circ\varphi_1 \text{ with } \circ \in \{\neg, \square, \Diamond, \bigcirc\} \\ 1 + size(\varphi_1) + size(\varphi_2) & \text{if } \varphi = \varphi_1 \circ \varphi_2 \text{ with } \circ \in \{\mathcal{U}, \mathcal{W}, \vee, \wedge, \rightarrow, \leftrightarrow\} \end{cases}$$

The different functions modeling the simplicity are presented in the following:

1. **Regular Size as Simplicity Term:** Let φ be an LTL formula, let s be the size of the LTL formula. The simplicity is defined as
$$Size_{Reg}(\varphi) = \frac{1}{s}$$

2. **Linear Size as Simplicity Term:** Let φ be an LTL formula, let s be the size of the LTL formula and let $s_{\max}$ be the maximum size of the all specification candidates. The simplicity is defined as
$$Size_{Lin}(\varphi) = 1 - \frac{s}{s_{max}}$$

3. **Exponential Size as Simplicity Term:** Let φ be an LTL formula and let s be the size of the LTL formula. The simplicity is defined as
$$Size_{Exp}(\varphi) = \frac{1}{2^s}$$

4. **Mapped Size as Simplicity Term:** Let $[x_{\min}, x_{\max}]$ be the range of the specificity score for all specification candidates and let $[s_{\min}, s_{\max}]$ be the range of the size of all candidates. Let φ be an LTL formula chosen from the specification candidates and let s be the size of the LTL formula φ. This score maps the size of the formula, to the range of the the specificity score using a linear function. The smallest formula, so the formula with the minimum size is mapped to the highest probability, while the largest formula is mapped to the lowest probability. The simplicity is defined as
$$Size_{Mapped}(\varphi) = -\frac{x_{\max} - x_{\min}}{s_{\max} - s_{\min}} \cdot s + \left(\frac{x_{\max} - x_{\min}}{s_{\max} - s_{\min}} \cdot s_{\max} \right) + x_{\min}$$

3.3.2 Specificity Terms

For the specificity term, we differentiate between the LTL_f and the LTL setting. For LTL_f, we focus on finite model counting as well as a finite density approach, which are both based on the number of finite models of a specified length for the property. For the LTL setting, we focus on the notion of the density of LTL formulas.

3.3.2.1 Linear Temporal Logic over finite traces

In this subsection, we introduce the approaches for measuring the specificity of an LTL_f formula.

1. **Model Counting Score as Specificity Score:** Let φ be an LTL_f formula and let n be the length of the finite traces, which are provided as examples. We differentiate between the existential semantics indicated with $\exists$ and the universal semantics indicated with $\forall$. Let $m^{\exists}$ be the total number of traces of length n, where φ holds according to the existential semantics. Let $m^{\forall}$ be the total number of traces of length n, where φ holds according to the universal semantics. The specificity score for the different semantics are defined as:

$$MC^{\exists}_{finiteCounting}(\varphi, n) = \frac{1}{m^{\exists}}$$

$$MC^{\forall}_{finiteCounting}(\varphi, n) = \frac{1}{m^{\forall}}$$

2. **Finite Density on Finite Traces** Let φ be an LTL_f formula and let n be the length of the finite traces, which are provided as examples. We differentiate between the existential semantics indicated with $\exists$ and the universal semantics indicated with $forall$. Let $m^{\exists}$ be the total number of traces of length n, where φ holds according to the existential semantics. Let $m^{\forall}$ be the total number of traces of length n, where φ holds according to the universal semantics. The specificity score for the different semantics are defined as:

$$MC^{\exists}_{finiteDensity}(\varphi, n) = 1 - \frac{m^{\exists}}{(2^{AP})^n}$$

$$MC^{\forall}_{finiteDensity}(\varphi, n) = 1 - \frac{m^{\forall}}{(2^{AP})^n}$$

Example 3.4 (LTL_f Specificity Scores)**.** Let $\varphi = \square\neg(a \wedge b)$ be our LTL_f formula over the atomic propositions $AP = \{a, b\}$. Let the existential semantics be our given semantics and let 5 be the length of the traces. For φ, there exist 3^5 models of length 5, as the sets $\{a\}$, $\{b\}$ and $\{\}$ represent all possible assignments of the atomic propositions for every position of the trace. Given the number of finite models of

length 5, we now compute the scores for finite semantics. The score $MC^{\exists}_{finiteCounting}$ for φ is given by:

$$MC^{\exists}_{finiteCounting}(\varphi, 5) = \frac{1}{3^5} = 0.00411423$$

The score $MC^{\exists}_{finiteDensity}$ for φ is given by:

$$MC^{\exists}_{finiteDensity}(\varphi, 5) = 1 - \frac{3^5}{4^5} = 1 - 0.2373 = 0.7627$$

3.3.2.2 Specificity Term: Linear Temporal Logic

Now, we focus on the density of an LTL formula, as LTL reasons about infinite traces.

1. **Density of Bound n as Specificity Score:** Let φ be an LTL formula and let $\nabla_\varphi(n)$ be the density of the LTL formula φ for the bound n. The specificity score is defined as:
$$MC_{density}(\varphi, n) = 1 - \nabla_\varphi(n)$$

2. **Asymptotic Density as Specificity Score:** Let φ be an LTL formula and let ∇^∞_φ be the asymptotic density of the LTL formula. The specificity score is defined as:
$$MC_{asympDensity}(\varphi) = 1 - \nabla^\infty_\varphi$$

Example 3.5 (LTL Specificity Scores)**.** Let $\varphi = \square\neg(a \wedge b)$ again be our LTL formula over the atomic propositions $AP = \{a, b\}$. Let our bound n be 5. For φ, the sets $\{a\}$, $\{b\}$ and $\{\}$ represent all possible assignments of the atomic propositions for every position. Hence, there are $5 \cdot 3^5$ n-models. The score $MC_{density}$ for φ is given by:

$$MC_{density}(\varphi, 5) = 1 - \frac{5 \cdot 3^5}{5 \cdot 4^5} = 0.7627$$

The score $MC_{asympDensity}$ for φ is given by $MC_{asympDensity}(\varphi) = 1$ as φ is a safety property.

Example 3.6 (LTL Specificity Scores for a Liveness Property)**.** Let $\varphi = \diamondsuit\square a$ be our LTL formula over the atomic propositions $AP = \{a, b\}$. Let our bound n be 5. φ has 992 n-models, hence the $MC_{density}$ score of φ is given by:

$$MC_{density}(\varphi, 5) = 1 - \frac{992}{5 \cdot 4^5} = 0.80625$$

The $MC_{asympDensity}$ of φ is given by $MC_{asympDensity}(\varphi) = 1$ as φ is a persistence property. The $MC_{finiteDensity}$ score evaluates to 0, as φ is a liveness property and hence, all finite words represent a prefix for a valid trace.

3.4 The Asymptotic Density

In this section, we focus on the computation of the asymptotic density of an LTL formula φ. Given φ, we produce an equivalent DPA. The density of a deterministic parity automaton is given by the sum of the densities of all accepting terminal strongly connected components [10]. The density of an accepting terminal SCC corresponds to the probability of reaching this terminal accepting SCC. Hence, computing the density can be viewed as a convergence problem of a Markov chain by replacing the label of the automaton, by its probability of taking this edge.

To compute the probability of reaching a terminal SCC of a deterministic parity automaton, we modify the automaton. As we are only interested in whether we end in the specific terminal strongly connected component, but not about the exact state of the component, we replace every terminal SCC with one new state.

Construction 3.5 (Replacement of terminal SCCs)**.** Let $\mathcal{A} = (\Sigma, Q, I, T, \text{PARITY}(c))$ be a deterministic parity automaton with terminal SCCs $scc = \{s_1, \ldots, s_n\}$, where $s_i \subseteq Q$. It holds that $s_j \cap s_k = \emptyset$ for all $j, k \in \{1, 2, \ldots, n\}$ with $j \neq k$. We denote $sccSet = \{q \mid q \in Q :$ there exists a terminal SCC $s_i \in scc$ such that $q \in s_i\}$ as the set of states belonging to a terminal SCCs. We construct the deterministic parity automaton $\mathcal{A}'$ as follows:

- $\Sigma' = \Sigma$
- $Q' = \{q_{Original} \mid q \in Q \setminus sccSet\} \cap \{i_{CSS} \mid i \in \{1, \ldots, n\}$ for $scc = \{s_1, \ldots, s_n\}\}$
- $I' = \begin{cases} \{q_{Original}\} & \text{if } I = \{q\} \text{ with } q \in Q \setminus sccSet \\ \{i_{CSS}\} & \text{if } I = \{q\} \text{ with } q \in sccSet, \text{ where } q \in s_i \text{ with } s_i \in scc \end{cases}$
- $T' = \{(q_{Original}, \sigma, q'_{Original}) \mid (q, \sigma, q') \in T \wedge q, q' \in Q \setminus sccSet\}$
  $\cap \{(i_{CSS}, \sigma, i_{CSS}) \mid$ for all $\sigma \in \Sigma$ and all $i \in \{1, \ldots, n\}$
  where $scc = \{s_1, \ldots, s_n\}\}$
  $\cap \{(q_{Original}, \sigma, i_{CSS}) \mid (q, \sigma, q') \in T, q \in Q \setminus sccSet,$
  $q' \in s_i,$ with $s_i \in scc\}$
- $\text{PARITY}(q) = \begin{cases} \text{PARITY}(q) & \text{if } q \in Q \setminus sccSet \\ \max_{q' \in s_i} \text{PARITY}(q') & \text{if } q \in sccSet, q \in s_i \text{ with } s_i \in scc \end{cases}$

Example 3.7 (DPA Construction)**.** Given the LTL formula $\varphi = a \wedge \Box\Diamond b$. An equivalent DPA is given by Figure 3.3, where the nodes q_0, q_2 and q_3 have the color 1 and q_1 has the color 2. This DPA consists of two terminal SCCs, where the set of the terminal SCC is given by $\{\{q_1, q_2\}, \{q_3\}\}$ First, the two nodes q_1 and q_2 represent an accepting terminal SCC as q_1 has the color 2. This terminal SCC is colored green.

On the other hand, q_3, which is colored in red represents one terminal SCC, however, this terminal SCC is not accepting.
As we are not interested in the state at which we enter the SCC, we reduce the problem of calculating the density by representing each terminal SCC by only one state. We apply Construction 3.5 to obtain the modified DPA for $\varphi = a \wedge \Box\Diamond b$. The modified automaton is given in Figure 3.4. Besides merging the states, this construction adjusts edges as well. We experience, that in Figure 3.3, starting in q_0, there is one edge to q_1 labeled with $a \wedge b$ as well as one edge to q_2 labeled with $a \wedge \neg b$. In addition, q_1 and q_2 represent one terminal SCC. Therefore, the states q_1 and q_2 are combined into one new state with an edge a. With this modified DPA, we can now calculate the probability of entering SCCs.

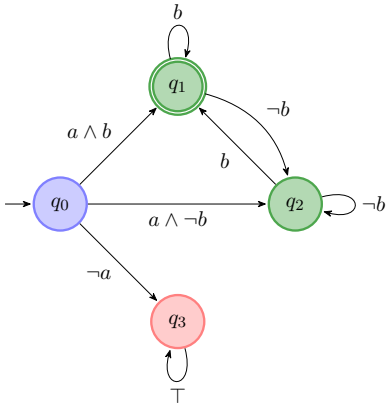

Figure 3.3: DPA for LTL formula $\varphi = a \wedge \Box\Diamond b$.

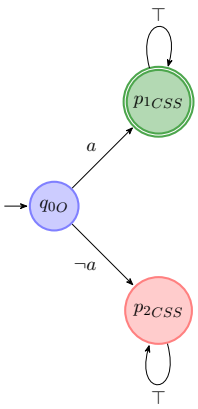

Figure 3.4: DPA for $\varphi = a \wedge \Box\Diamond b$ with SCC modification

To compute the density, we are interested in the probability of reaching terminal SCCs. Hence, given a DPA, we do not focus on edge labels, but on transition probabilities. We can represent the probability of an edge via a transition probability matrix P.

Example 3.8 (Probability Matrix P)**.** Let 3.4 be the given DPA. We construct the matching transition probability matrix as follows:

$$\boldsymbol{P} = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The first row represents the outgoing edges of q_{0O}, the second row of p_{1CSS} and the last row of p_{2CSS}.

Now, we can focus on the notion of the density.

Definition 3.5.1 (Density of DPA)**.** Let $\mathcal{A}$ be a deterministic parity automaton. We compute the modified DPA $\mathcal{A}'$ using the Construction 3.5 in order to represent the terminal strongly connected components by one state each. Let $s_1, s_2 \dots s_m$ be the states of the DPA $\mathcal{A}'$. Let s_i be the initial state and $\boldsymbol{P}$ be the transition probability matrix of $\mathcal{A}'$.

The initial distribution is given by $\boldsymbol{d} = (d_1, d_2, \dots, d_m)$, where

$$d_j = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

The distribution in the limit is given by

$$\mathbf{d}^* = \lim_{n \to \infty} d \cdot P^n$$

If this distribution is converging, we know, that the density of the DPA is converging. So in case, the distribution convergences, the asymptotic density is given by

$$\nabla_{\mathcal{A}} = \nabla_{\mathcal{A}'} = \sum_{i \in accTerSCC} p_i$$

with $\mathbf{d}^* = (p_1, \dots, p_m)$ and $accTerSCC$ being the states, which represent the accepting terminal strongly connected components.

Example 3.9 (Density of DPA)**.** Taking the DPA, from our previous examples presented in Figure 3.4. The distribution in the limit is given by:

$$\mathbf{d}^* = \lim_{n \to \infty} \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}^n = \begin{pmatrix} 0 & 0.5 & 0.5 \end{pmatrix}$$

p_{1CSS} and p_{2CSS} represent the terminal SCCs, however only p_{1CSS} is accepting. The asymptotic density is given by the element of $\mathbf{d}^*$ for state p_{1CSS}. In this case, corresponds to the entry in the middle of the vector. The density of the DPA is 0.5.

Chapter 4

Implementation

In this chapter, we explain the algorithms we implemented for the LTL Specification Mining. The implementation is done in F#. We explain the implementation with the help of F# pseudocode enriched with mathematical operations to improve the readability. The exact implementation is provided in [16]. In general, we distinguish three different settings:

1. LTL_f Specification Mining with existential semantics
2. LTL_f Specification Mining with universal semantics
3. LTL Specification Mining

We evaluate the algorithm by taking an LTL formula, generating traces for this specification and evaluating the approach using the generated traces and a ranking function. For the generation of traces, we take two parameters into account. On the one hand, the number of samples specifies the number of traces we are generating. On the other hand, the length of the traces. For LTL_f, the length specifies the length of the finite traces we are generating. For LTL, the length is taken into account for the length of the lasso. We will go into more detail in 4.2. In addition, there exists a parameter, which specifies the size of our search space. To be more precise, this parameter represents the number of specifications, which are sampled and represent the possible search space. In addition to these sampled formulas, we additionally include all subformulas to expand the search space even further. Ranking functions based on the density of the bound n, take the bound n as a parameter into account.

Next, we show how deterministic parity automata are internally represented and present the implementation of the non-empty states. After that, we focus on the sampling process, the search space generation and algorithms to compute the model counting scores and density.

4.1 Deterministic Parity Automaton

To be more efficient, many computations are based on the deterministic parity automata. For the implementation, we use the FsOmegaLib Library [5] to represent the DPAs internally. The internal representation of a deterministic parity automaton consists of a skeleton, an initial state and a color mapping. The skeleton represents the atomic propositions, the states of the automaton and the edges in the form of a mapping from the source/state to a list of successors. The successors are represented by a DNF of the atomic propositions together with the target state. The color mapping function maps every state to a color.

One important notion in the algorithms are the non-empty states of a DPA as defined in Definition 3.1.3. To compute the non-empty states, we distinguish between the two finite semantics:

4.1.1 Existential Semantics

The function `nonEmptyStatesExistential` computes the existential non-empty states of a deterministic parity automaton. The main idea of this function is to compute all states for which there exists a path to a state with an even color, that allows an accepting loop.
Therefore, we first compute the even colors present in the DPA. After that, we compute for every even color found in the DPA, the states with this even color, which allow self-loops only traversing states with a smaller or equal color. These states are computed using the function `acceptingLoopColor`. We continue by merging all these sets of states found for the different even colors. This set now represents all states with an even color which allow accepting self-loops.
We now use this set to generate the set of non-empty states. To achieve that, we take all states of the DPA, for which there exists a path to a state that allows an accepting loop. To be more precise, we filter the states of the DPA by checking whether there exists a path from this state of the DPA to a state allowing accepting self-loops.

```
let nonExistentialStatesExistential dpa =
    let evenColors =
        {dpa.Color.[s]|s ∈ dpa.States: ∃x ∈ ℕ: dpa.Color.[s] ÷ 2=x}
    let statesWithAcceptingSelfloop =
    ∪col∈evenColors acceptingLoopColor col dpa
    let reachPairs =
        shortestPathBetweenAllPairs dpa.States dpa.Edges
    return {s|s ∈ dpa.States: ∃s' ∈ statesWithAcceptingSelfloop:
            reachPair.ContainsKey(s,s')}
```

```
let acceptingLoopColor col dpa  =
    let statesOfSmallerCol = {s|s ∈ dpa.States: dpa.Color.[s] ≤ col}
    let reachPairs =
        shortestPathBetwAllPairs statesOfSmallerCol dpa.Edges
    let winningStates = {s|s ∈ dpa.States: dpa.Color.[s]=col
                           ∧ reachPairs.ContainsKey(s,s)}
    return winningStates
```

4.1.2 Universal Semantics

For the universal semantics, we need to compute the non-empty states as well. The main idea is to work with the negated DPA. We negate the DPA by increasing the color of each state by one. After that, we compute the non-empty states for the existential semantics of the negated DPA. For each of these existential non-empty states of the negated DPA there exists at least one satisfying path for the negated DPA and hence, at least one not-accepting path in the original DPA.
For the universal semantics, a state is called a non-empty state, if every infinite outgoing path of the state represents an accepting path. Hence, if there exists at least one outgoing path for this state, that is not accepted in the original DPA, this state is not part of the non-empty states of the original DPA. Therefore, the non-empty states of the original DPA are given by the set difference between the DPA states and the non-empty states for the existential semantics of the negated DPA.

```
let nonEmptyStatesUniversal dpa =
    let dpa' = {dpa with Color = {c+1 | c ∈ dpa.Color}}
    let nonWinningStates = nonEmptyStatesExistential dpa'
    return dpa'.States \ nonWinningStates
```

4.2 Sampling Traces

In order to evaluate the approach efficiently, we take an LTL formula as well as atomic propositions to sample traces which are taken into account to evaluate the ranking functions. First, we read the LTL formula and the atomic propositions and compute the deterministic parity automaton for the LTL formula using Spot [7]. As mentioned before, for the trace generation, the length as well as the number of generated traces are given as parameters. The generation of traces differs for every setting, therefore we explain the generation process individually.

4.2.1 LTL_f Specification Mining

We use the function `sampleTraces` to generate finite traces for the existential and universal semantics. First, we compute the non-empty states for the respective semantics and ensure, that there exists at least one reachable non-empty state. After that, we continue to generate all possible paths of a specified length of the automaton. We filter these by checking for each path, whether the last state of this path represents a non-empty state of the automaton. If the last state is part of the non-empty states, we are dealing with a path, whose corresponding traces are accepting and hence, we keep this path. On the other hand, if the last state is not contained in the non-empty states, we filter out this path. At this point, we are only left with paths, where all traces are accepting. Therefore, we now randomly select one of these paths and assign the values of the atomic propositions according to the DNF of the respective edge. Values of atomic propositions not taken into account by the DNF are randomly assigned.

```
let sampleTraces sem dpa length numberOfTraces =
    let nonEmptyStates = computeNonEmptyStates sem dpa
    if nonEmptyStates.isEmpty then fail
    let allPaths = genAllPaths dpa length dpa.Initial
    let candidates =
        { path | path ∈ allPaths: path.Last ∈ nonEmptyStates}
    let traces =  createPaths dpa numberOfTraces candidates
    return traces
```

```
let rec createPaths dpa number candidates =
    match number with
    | 0 -> return []
    | n -> return createTrFromPath dpa (Random.Next candidates)::
           createPath dpa (n-1) candidates
```

```
let rec createTrFromPath dpa path =
    if path.length <= 1 then
        return []
    else
        let edge = dpa.getEdges from=path.First to=path.Second
        let assignment = dpa.getAssignment edge
        return assignment::
           (createTrFromPath dpa path.RemoveFirst)
```

```
let computeNonEmptyStates sem dpa =
    if sem=Universal then
        return nonEmptyStatesUniversal dpa
     else
        return nonEmptyStatesExistential dpa
```

The function `genAllPaths` generates all possible paths through the DPA of the specified length.

4.2.2 LTL Specification Mining

For the LTL specification mining setting, we sample traces in the form of ultimately periodic words $u \cdot v^{\omega}$. The parameters taken into account, are the number of traces and the length. In order to make our set of traces more diverse, we choose the lower bound for the length of both the prefix and the loop randomly for every trace. The function `generateLassoTraces` generates the specified number of traces each consisting of a prefix and a loop. First, we focus on the loop. Therefore, we compute the states with even color, which allow accepting self-loops. Using these states, we create a map consisting of these states and for each state one possible loop. We call this map `acceptingLoops`. Next, we explain how we generate this map. For every state with an even color, which allows accepting self-loop, we generate the set of states with a smaller and equal color. Given this set of states with a smaller and equal color, we compute the edges, which ensure, that we do not visit states, which with a higher color. We generate the loop by randomly walking through the automaton and generating the trace, ensuring we are able to return to the start of the loop. For the length parameter of the loop, we take an arbitrary length between one and the specified length. As we walk randomly for the specified number of steps, we may not end in the state we started the loop. In this case, we take the shortest path to get there. The prefix is similarly generated by randomly moving through the automaton, while still making sure, there exists at least one accepting path. There exists an accepting path as long as we can reach one of the keys of the map of the `acceptingLoops`. For the length of the prefix, we take an arbitrary length between zero and the specified length subtracted by one. After we reach the specified length, we are not necessarily in a state, that allows an accepting loop. Therefore, we take the shortest way to a state, which allows an accepting loop. Hence, the specified length is just the minimum length of the prefix.

```
let rec generateLassoTraces dpa numberOfTraces length=
    if numberOfTraces = 0
        return {}
    else
        let prefix, loop = sampleLassoInDpa length dpa
        return {(prefix,loop)} ∪
            generateLassoTraces dpa (numberOfTraces-1) length
```

```
let  sampleLassoInDpa length dpa =
    let loopMap = generateLoopMap dpa length dpa.States
    let prefix,finalState = generateRandomPath edges=dpa.Edges
        length=(Random.Next length) initial=dpa.Initial
        finalStates=loopMap.Keys
    let loop = loopMap.[finalState]
    return prefix, loop
```

```
let rec generateLoopMap dpa length states=
    match states with
    | [] -> []
    | x::xs -> if dpa.Color[x]%2=0 then
        let statesOfSmallerCol =
            {s|s ∈ dpa.States: dpa.Color.[s] ≤ dpa.Color[x]}
        let reachPairs =
            shortestPathBetwAllPairs statesOfSmallerCol dpa.Edges
        if reachPair.containsKey (s,s) then
            let path,_ = generateRandomPath edges=reachPairs
                    length=(Random.Next length)+1 initial=x
                    finalStates={x}
            return (x, path)::generateLoopMap dpa length xs
        else
            return generateLoopMap dpa length xs
    else
        return generateLoopMap dpa length xs
```

The function `generateRandomPath` with its parameters `length`, `edges`, `initial` and `finalStates` generates a random trace through the automaton, which starts in the `initialState` and ends in one of the `finalStates`. To do so, the function randomly walks for `length` many steps. If the state, it ends in is part of the `finalStates`, it ends and returns the finite trace with its final state. Otherwise, it returns the random

walk concatenated with the shortest path to one of the `finalStates`. In addition, its final state is returned as well.

4.3 Search Space

Given traces and atomic propositions, we can generate the search space. First, we randomly sample a specific number of LTL formulas by using the command line tool *randltl* from Spot [7]. These formulas together with all their subformulas represent the whole space of possible candidates. However, it is not guaranteed, that every formula holds on the traces. Therefore, for every setting, we still have to check, for each formula whether the formula holds on every generated trace. For that, we generate a deterministic parity automaton for every LTL formula and test whether every trace is accepted by the automaton.

```
let searchSpace traces sampledLTL sem=
    if sem=infinite then
        return {ltl|ltl ∈ sampledLTL
                ∧ infiniteCheck(dpaFrom ltl) traces}
    else
        return {ltl|ltl ∈ sampledLTL
                ∧ finiteCheck sem (dpaFrom ltl) traces}
```

We now distinguish the checking for LTL_f and for LTL.

4.3.1 LTL_f check

For the LTL_f setting, we first compute the set of non-empty states for the DPA according to the semantics. Given the set of sampled finite traces, we then traverse for every trace through the automaton and save the state we end in. If for every trace, the last state is included in the non-empty states, this formula represents a candidate of the search space. On the other hand, if at least one trace does not end in a non-empty state, the formula does not represent a candidate of the search space.

```
let finiteCheck sem dpa traces=
    let nonEmptySt =  computeNonEmptyStates sem dpa
    let acceptedTraces = { tr | tr ∈ traces ∧
        (simTraceInDpa dpa tr.prefix).Last ∈ nonEmptyStates}
    return traces=acceptedTraces
```

The function `simTraceInDpa` simulates the trace in the DPA and returns the corresponding path through the DPA.

4.3.2 LTL check

For the LTL setting, given the set of sampled traces in the form of ultimately periodic words $u \cdot v^{\omega}$, we traverse for every trace through the automaton. We start by traversing through the automaton for the prefix and store the state, it ends in. To check the loop, we start in the state, we end in the prefix. Now we need to iterate potentially multiple times through the loop. While traversing the automaton for the loop, we store the current state with the current position of the loop. In every step, we check, whether we already stored the combination of the current state and current position. In case, it is already stored, we found a loop and are now able to determine, whether this trace is accepted by the DPA. We check if it is an accepting trace by searching for the highest color in the loop. If this color is even, the trace is accepted, otherwise, the trace is not accepted. In case, the combination is not stored yet, we store it, take the next step in the loop and try again with the successor state and successor position. This function terminates as the DPA consists of finitely many states.

```
let infiniteCheck dpa traces=
    let acceptedTraces = { tr | tr ∈ traces
       ∧ checkLoop dpa tr.loop visited=[] pos=0
                 state= ((simulateTraceInDpa dpa tr.prefix).Last)
    }
    return acceptedTraces=traces
```

```
let rec checkLoop dpa loop visited position state  =
    if (state, position) ∈ visited then
        let index = findIndex ((=) (state, position)) visited
        let loopPart = visited[index..]
        let maxCol = max {dpa.Color(element.first) |
                          element ∈ loopPart}
        return maxColor % 2 = 0
    else
        let newPos = position+1 % loop.Length
        let newState = dpa.Successor(state, edge=loop[position])
        let newVisited =((state,position)::visited)
        return checkLoop dpa loop newVisited newPos newState
```

4.4 Ranking Function

4.4.1 Simplicity Score

The simplicity scores are based on the notion of the *size* of an LTL formula. The definition of the size is given in Section 3.3.1. The implementation of the individual simplicity scores is straightforward.

4.4.2 Specificity Score

For the specificity score, we need to distinguish between the model counting and the density scores. Besides the computation of the model counting and the density scores, the computation of the specificity scores is straightforward. Hence, we only focus on the density and model counting.

4.4.2.1 Model Counting

For the model counting algorithm, we make use of the structure of the DPA. The function `modelCounting` computes the number of traces of a given length, fulfilling the property of the DPA for the given semantics recursively. In this algorithm, we first compute the non-empty states of the automaton for the DPA and the semantics and construct all possible AP assignments. These assignments are stored in `inputSpace`. Next, we initialize our initial counts. For every state in the DPA, we set the count to one, in case the state is contained in the non-empty states, otherwise, to zero. This score represents the model counting score of traces, with length zero. If a state is contained in the non-empty states, traces ending in these state are accepted. Now, we produce the model counting score based on the initial counts. For that, we pass the counts backward along the automaton. In every step, we iterate through the states of the automaton and compute the new model counting scores for every state. The new model counting score of a state is given by the sum of the previous model counting scores of the successors. We generate the successors by checking for the current state and every AP assignment the successor. If one state is the successor for multiple AP assignments, the model counting score is multiple times added. In general, the model counting score of length i corresponds to the map entry of the initial state of the DPA in the i-th step of the recursion. Hence, the final model counting score is given by the final entry of the initial state.

```
let modelCounting dpa sem length=
    let nonEmptyStates = computeNonEmptyStates sem dpa
    let inputSpace = booleanPowerset (dpa.APs.Length)
    let initCount = {(s,1) | s ∈ dpa.States ∧ s ∈ nonEmptyStates}
                    ∪{(s,0) | s ∈ dpa.States ∧ s ∉ nonEmptyStates}
```

```
    let result = genCount initCount length inputSpace
    return result.[dpa.Initial]
```

```
let rec genCount counts length inputSpace=
    if length=0 then
        return counts
    else
        let currentCounts =
            {(s,sum) | s ∈ dpa.States ∧ sum =
            ∑_{i∈inputSpace} counts[dpa.Successor(state=s,label=i)]}
        return genCount currentCounts (length-1) inputSpace
```

Example 4.1 (Model Counting Score)**.** Let $\varphi = \square a$ be our LTL formula over the atomic propositions $AP = \{a, b\}$. We construct the automaton $\mathcal{A}$, which is equivalent to φ. The automaton is represented in Figure 3.2. q_0 has the color 2, while q_1 has the color 1. The initial count for q_0 is 1 and for q_1 0. We represent the counts for the call `modelCounting dpa existential 5` for every recursion step in the following table. We state the counts at the end of every step.

length	q_0	q_1
start	1	0
5	2	0
4	4	0
3	8	0
2	16	0
1	32	0
0	32	0

We experience, that the score of q_0 doubles in every step, since there are the two assignments $a = 1, b = 1$ and $a = 1, b = 0$, which allow us to stay in q_0. Hence, there are 32 finite models of length 5. For q_1 the score remains 0 since there is no possibility to reach another state than q_1.

4.4.2.2 Asymptotic Density

The computation of the asymptotic density is also based on the DPA. Here, we first construct the new DPA $\mathcal{A}'$ using Construction 3.5 from the given automaton. Using the newly obtained DPA, we can generate the set of states, which represent terminal accepting SCCs. This set is represented by `acceptingTermCssStates`. If the set is empty, the asymptotic density is zero. In case, there exists at least one accepting terminal SCC we need to generate the transition probability of $\mathcal{A}'$ as well as the initial distribution. In case, the Markov convergence is converging in 10,000 time steps, we

generate this converging distribution. The density is then given by the sum of the terminal accepting SCCs over the converging distribution. In case, the Markov chain is not converging, we return -1 to mark, that it is not converging and hence, we do not incorporate this formula in the ranking.

```
let asymptoticDensity dpa =
    let dpa' = SCCconstruction dpa
    let acceptingTermCssStates = {s | s ∈ dpa'.States
        ∧ s is an accepting terminal SCC state}
    if acceptingTermCssStates.Empty then
        return 0
    else
        let edgeProb = probTransitionMatrix dpa'
        let initial = initialDistribution dpa'
        if isConverging initial edgeProb  then
            let stationarityDistribution =  initial*edgeProb^10000
            return ∑_{i∈acceptingTermCssStates} stationarityDistribution[i]
        else
            return -1
```

Example 4.2 (Asymptotic Density)**.** Let $\varphi = \square a$ be our LTL formula over the atomic propositions $AP = \{a, b\}$. We construct the automaton $\mathcal{A}$, which is equivalent to φ. The automaton is represented in Figure 3.2. q_0 has the color 2, while q_1 has the color 1. Construction 3.5 returns the same automaton, as there only exists one terminal SCC with exactly one state. This terminal SCC is given by q_1. However, this terminal SCC is not accepting, as q_1 has the color one. Hence, the asymptotic density is 0.

4.4.2.3 Density of Bound n

The function `density` computes the number of n-models, by iterating through all values for `prefixLength` ≥ 0 and `loopLength` ≥ 0, for which $n =$ `prefixLength` $+$ `loopLength` holds. For every of these combinations of `prefixLength` and `loopLength`, it calls the function `modelScore`, which calculates the number of models for a specific `prefixLength` and `loopLength`.

The function `modelScore` recursively constructs all possible prefixes symbolically and computes their model counting score. The function first checks whether the length of the prefix is already reached. In case it is already reached, it computes the number of loop models for the specified `loopLength`. In case, the length of the prefix is not reached, it recursively sums up over the model counting score for every possible edge. For every edge, the model counting score consists of the product between the number of assignments of the `dnf` of the respective edge and the recursive call for the

`modelScore` with a by one increased `step`. The number of assignments of the `dnf`, which evaluate to true is calculated via the function `countModelOfGuard`.

The function `getNumOfLassos` symbolically counts the number of loops using a constraint. It is a recursive function, which produces all possible paths of possible loops of the length as specified in `loopLength`. The model scores of all these paths are added up to the final model score value. The constraint represents the `dnf` assignment for the respective positions. It starts with the `dnf` assignment *true* at every position for the loop. The constraints are updated in every step for the current position by conjunction of the old constraint of this position with the `dnf` assignment of the edge of the current state. There might be several edges, we look at each individually and take a new instance of the constraint for each. In addition, we track the combinations of the current state and the current position we see when moving through the automaton. As soon as we have seen the combination of current state and current position, we have a loop. First, we check if the loop is accepting. If it is not accepting, we return zero, otherwise, we continue by using the list of the constraints. Since the constraints are represented as `dnf` assignemnts, we can multiply over the `countModelGuard` value of all entries of the constraint. This leaves us with the number of loop models for one path through the automaton.

```
let density n dpa=
    let densityComp dpa prefixLength loopLength=
        if loopLength=0 then
            return 0
        else

            return density dpa (prefixLength+1) (loopLength-1)
                + (modelScore dpa prefixLength loopLength step=0
                current=dpa.Initial)
    densityComp dpa 0 n
```

```
let rec modelScore dpa prefixLength loopLength step current =
    if step=prefixLength then
        let initialContraint = List.init loopLength ( _ -> DNF.true)
        return getNumOfLassos current loopLength dpa pos=0
                constraints=initialContraint visited=[]
    else
        ∑(dnf,ss)∈dpa.Edges.[current] (countModelOfGuard dnf) *
                (modelScore dpa prefixLength loopLength step+1 ss)
```

```
let rec getNumOfLassos current loopLength dpa pos constraints visited=
    if (current, pos) ∈ visited then
        let index = findIndex ((=) (state, pos)) visited
        let loopPart = visited[index..]
        let maxCol = max {dpa.Color(element.first) |
                          element ∈ loopPart}
        if maxColor % 2 = 0  then
            return ∏_{i=0}^{loopLength−1} countModelsOfGuard constraints[i]
        else
            return 0
    else
        let newVisited = visited @ [(current,pos)]
        let newPos = (pos+1) % loopLength
        let ScorePerDNF =
            dpa.Edges.[current] |> map (fun (g, ss) ->
                let newFormula =
                    DNF.constructConjunction [constraints.[current];g]
                if DNF.isSat newFormula then
                    let newContraint = currentConstraints
                        |> mapi (fun i x ->
                            if i = currentPosition then newFormula
                            else x)
                    letNumOfLassos ss loopLength dpa newPos
                        newConstraint newVisited
                else
                    0
            )
        return ∑_{i∈ScorePerDNF} i
```

Chapter 5

Evaluation

In this chapter, we evaluate our algorithms using benchmarks from related work and compare their result to ours. We test the algorithm using different ranking functions. This chapter covers the exact ranking functions.

5.1 Benchmark

As a benchmark, we use on the one hand the benchmark used by Ehlers et al. [9] to evaluate our approach. To get a broader overview of the performance, we additionally incorporate the benchmark used by Roy et al. [28].
Ehlers et al. [9] use a benchmark consisting of six invariants and four liveness properties. The LTL patterns used by Roy et al. [28] are divided into absence, existence, universality and disjunction of common patterns [8]. However, we only use the absence, existence and universality patterns as the disjunction patterns represent disjunctions of the existence patterns. Table 5.1 describes the properties taken into account by Roy et al. and Table 5.2 describes the properties used by Ehlers et al.

Absence	Existence	Universality
$\square(\neg a)$	$\diamondsuit a$	$\square a$
$\square(a \rightarrow \square(\neg b))$	$(\square(\neg a)) \vee (\diamondsuit(a \wedge (\diamondsuit b)))$	$\square(a \rightarrow \square b)$
$\diamondsuit a \rightarrow (\neg b \mathcal{U}\, a)$	$\square(a \wedge ((\neg b) \rightarrow ((\neg b) \mathcal{U}((\neg b) \wedge c))))$	$\diamondsuit a \rightarrow (b \mathcal{U}\, a)$

Table 5.1: Benchmark used by Roy et al.[28]

Property
1. $\square(a \rightarrow (b \vee c \vee d))$
2. $\square(a \rightarrow (b \vee c))$
3. $\square(\bigcirc \neg a \rightarrow (\neg b \leftrightarrow \bigcirc(\neg b)))$
4. $\square(a \rightarrow \neg b)$
5. $\square(a \rightarrow (\neg b \wedge \neg c))$
6. $\square a$
7. $\square(a \rightarrow \diamondsuit b)$
8. $\square((a \wedge b) \rightarrow \bigcirc\diamondsuit(\neg c))$
9. $\square\diamondsuit a$
10. $\square\diamondsuit(\neg a \wedge \neg b)$

Table 5.2: Benchmark used by Ehlers et al.[9]

5.2 Parameters

All the ranking functions are evaluated using the same parameters. For every specification, 50 traces are automatically generated. The parameter for the length of the traces is set to 10. The algorithm is evaluated on a search space of 5000 LTL formulas and their subformulas. The formulas generated by Spot [7] have a length of up to 15. They are generated over the set of atomic propositions consisting of $\{a, b\}$ combined with the atomic propositions present in the respective specification, we are trying to mine. More detailed results for all our ranking functions with the ranking of the Top 5 specifications of the benchmark are displayed in Appendix A. In order to make the results more pleasant to read and better interpretable, instead of giving the value of the specificity, simplicity and ranking score as described in Chapter 3, we plug them into the $-\log_2()$ function. This adaption does not change the result of the ranking as the specification φ^*, which maximizes

$$\varphi^* = \arg\max_{\varphi \in \Phi} score(\varphi) = \arg\max_{\varphi \in \Phi} (Size(\varphi) \cdot MC(\varphi))$$

minimizes

$$\begin{aligned}\varphi^* &= \arg\min_{\varphi \in \Phi} \log_2(score(\varphi)) \\ &= \arg\min_{\varphi \in \Phi} \log_2(Size(\varphi) \cdot MC(\varphi)) \\ &= \arg\min_{\varphi \in \Phi} (-\log_2(Size(\varphi)) - \log_2(MC(\varphi)))\end{aligned}$$

However, the sum over the simplicity and the specificity score using the $\log_2()$ function represents the overall score.

5.3 Existential LTL_f Specification Mining

For the existential setting, we evaluate the following ranking function in more detail.

Existential Ranking Function: Finite Counting and Mapped Size

Simplicity:	$Size_{Mapped}(\varphi)$	*Mapped Size*
Specificity:	$MC^{\exists}_{finiteCounting}(\varphi, 10)$	*Finite Counting*
Ranking:	$Size_{Mapped}(\varphi) \cdot MC^{\exists}_{finiteCounting}(\varphi, 10)$	

Over all the ranking functions, we tested for this setting, the function *Finite Counting and Mapped Size* has the best results. It is able to mine all absence and universality specifications as well as safety properties, in case it does not lead to an overflow. The searched specifications or equivalent specifications are mined on the first position in the ranking. The exact ranking of the Top 5 specifications is displayed in the Appendix A.2. The tables 5.3 and 5.4 represent the position in the ranking for the benchmark. We are able to mine these properties with 50 traces, while related work was evaluated using between 10,000 and 50,000 traces.

In general, by defining the ranking function *Finite Counting and Mapped Size*, with the size function displaying the values linearly, we make the assumption that the property we try to mine is rather short and more importantly restrictive. The function *Finite Counting and Mapped Size* is capable of mining properties, as both, the specificity and the simplicity score have the same range. Hence, they provide a good trade-off between simplicity and specificity. Due to our underlying assumption, we manage to mine restrictive properties, like invariants. However, we fail to mine non-restrictive properties as well as liveness properties. It is important to notice, that we cannot mine any liveness properties like $\Diamond\Box a$, as there do not exist bad-prefixes for counterexamples. In particular, this means, that every finite trace represents a prefix for a liveness property. Hence, the specificity score of liveness properties equals the score for the property *true* for the existential semantics.

Absence	Rank	Existence	Rank	Universality	Rank
$\Box(\neg a)$	1	$\Diamond a$	-	$\Box a$	1
$\Box(a \rightarrow \Box(\neg b))$	1	$(\Box(\neg a)) \vee (\Diamond(a \wedge (\Diamond b)))$	-	$\Box(a \rightarrow \Box b)$	2 (1)
$\Diamond a \rightarrow (\neg b \,\mathcal{U}\, a)$	1	$\Box(a \wedge ((\neg b) \rightarrow ((\neg b)\,\mathcal{U}((\neg b) \wedge c))))$	-	$\Diamond a \rightarrow (b \,\mathcal{U}\, a)$	1

Table 5.3: Ranking positions for Finite Counting and Mapped Size, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al. [28]

Property	Rank
1. $\square(a \rightarrow (b \vee c \vee d))$	overflow
2. $\square(a \rightarrow (b \vee c))$	1
3. $\square(\bigcirc \neg a \rightarrow (\neg b \leftrightarrow \bigcirc(\neg b)))$	1
4. $\square(a \rightarrow \neg b)$	1
5. $\square(a \rightarrow (\neg b \wedge \neg c))$	1
6. $\square a$	1
7. $\square(a \rightarrow \diamondsuit b)$	-
8. $\square((a \wedge b) \rightarrow \bigcirc\diamondsuit(\neg c))$	-
9. $\square\diamondsuit a$	-
10. $\square\diamondsuit(\neg a \wedge \neg b)$	-

Table 5.4: Ranking positions for Finite Counting and Mapped Size, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al. [9]

Besides the ranking function *Finite Counting and Mapped Size*, we evaluated the following functions as well. The Top 5 rankings can be found in the Appendix A.2.

Existential Ranking Function: Finite Counting and Regular Size

Simplicity:	$Size_{Reg}(\varphi)$	*Regular Size*
Specificity:	$MC^{\exists}_{finiteCounting}(\varphi, 10)$	*Finite Counting*
Ranking:	$Size_{Reg}(\varphi) \cdot MC^{\exists}_{finiteCounting}(\varphi, 10)$	

The ranking function *Finite Counting and Regular Size* is not able to mine properties, which are less restricted like $\diamondsuit a \rightarrow b\,\mathcal{U}\,a$ and $\diamondsuit a \rightarrow (\neg b\,\mathcal{U}\,a)$. These properties are not found in the ranking. For both settings, the Top 5 specifications are *true*, $\diamondsuit a$, $\diamondsuit b$, $\diamondsuit(\square b)$ and $\diamondsuit(\neg a)$. However, for the tested invariant properties, the searched property is mined in the Top Two. Important to notice, the function *Finite Counting and Regular Size* is the only function over the whole ranking function space, that is able to mine the property $\square(a \wedge ((\neg b) \rightarrow ((\neg b)\,\mathcal{U}((\neg b) \wedge c))))$. The other functions are only able to mine the subproperty $\square a$. Still, overall it does not perform as well as the ranking function *Finite Counting and Mapped Size*.

Existential Ranking Function: Finite Counting and Linear Size

Simplicity:	$Size_{Lin}(\varphi)$	*Linear Size*
Specificity:	$MC^{\exists}_{finiteCounting}(\varphi, 10)$	*Finite Counting*
Ranking:	$Size_{Lin}(\varphi) \cdot MC^{\exists}_{finiteCounting}(\varphi, 10)$	

The ranking function *Finite Counting and Linear Size*, is similar to *Finite Counting and Regular Size* not able to mine the property $\Diamond a \rightarrow (\neg b \mathcal{U} a)$ in the ranking. However, $\Diamond a \rightarrow b \mathcal{U} a$ is mined on position two. Still, it is capable of mining restrictive properties but is not able to mine less restrictive properties.

Existential Ranking Function: Finite Density and Linear Size

Simplicity:	$Size_{Lin}(\varphi)$	*Linear Size*
Specificity:	$MC^{\exists}_{finiteDensity}(\varphi, 10)$	*Finite Density*
Ranking:	$Size_{Lin}(\varphi) \cdot MC^{\exists}_{finiteDensity}(\varphi, 10)$	

We achieve an almost similar result with the function *Finite Density and Linear Size* as we achieve with *Finite Counting and Regular Size*. We are able to mine both $\Diamond a \rightarrow b \mathcal{U} a$ and $\Diamond a \rightarrow (\neg b \mathcal{U} a)$ on the first position. In general, for all specifications, which are minable in the Top 5 for *Finite Counting and Regular Size*, we can mine them as well with the *Finite Density and Linear Size* in the Top 5. Hence, this ranking function is able to mine restrictive properties. Same as the ranking function *Finite Counting and Regular Size*, it is not able to mine liveness properties for the same reason.

5.4 Universal LTL$_f$ Specification Mining

We experience, that the universal setting is not suited for most of the properties in our benchmark. For the universal semantics, we are in general only able to mine properties, which are completely fulfilled on a finite trace and such that every infinite extension corresponds to a valid word. In particular, we are not able to mine invariants, persistence and response properties.

Hence, we designed a new benchmark in order to evaluate this setting. First, we decided on taking the property $\Diamond a$, as this was already in our other benchmark and is minable using this approach. Inspired by Somenzi and Bloem [29], we have incorperated the specifications $a \mathcal{U} b$, $a \mathcal{U} (b \mathcal{U} c)$ and $\neg(a \mathcal{U} (b \mathcal{U} c))$ in our benchmark. Additionally, we added $(\neg a) \mathcal{U} b$.

We now look deeper into the results for the following ranking function:

Universal Ranking Function: Finite Counting and Mapped Size

Simplicity:	$Size_{Mapped}(\varphi)$	*Mapped Size*
Specificity:	$MC^{\forall}_{finiteCounting}(\varphi, 10)$	*Universal Finite Counting*
Ranking:	$Size_{regular}(\varphi) \cdot MC^{\forall}_{finiteCounting}(\varphi, 10)$	

We experience, that we are capable of mining the properties $a \mathcal{U} b$, $(\neg a) \mathcal{U} b$ as well as $\neg(a \mathcal{U} (b \mathcal{U} c))$ in the Top 2. In comparison, we are not able to mine $\Diamond a$ and $(a \mathcal{U} (b \mathcal{U} c))$ in the Top 5. The exact ranking positions are displayed in Table 5.5. We experience for $(a \mathcal{U} (b \mathcal{U} c))$, that we need more traces in order to mine it as we are coincidentally generating traces, which fulfill properties, which end up higher in the ranking. $\Diamond a$ is not restrictive and similar to $(a \mathcal{U} (b \mathcal{U} c))$ we need more traces to rule out the coincidental similarity between the generated traces. However, the particular ranking function is still not able to mine it with 1000 traces as it has the underlying assumption, that the traces are restrictive. A ranking function like *Finite Density and Exponential* is able to mine it using 1000 traces.

Property	Ranking
$\Diamond a$	-
$a \mathcal{U} b$	1
$(\neg a) \mathcal{U} b$	2
$a \mathcal{U} (b \mathcal{U} c)$	-
$\neg(a \mathcal{U} (b \mathcal{U} c))$	2

Table 5.5: Ranking positions for Finite Counting and Mapped Size, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, our own Benchmark

Besides the mentioned universal ranking function, we tested the following functions as well, their rankings are represented in the Appendix A.1.

Universal Ranking Function: Finite Density and Linear Size

Simplicity:	$Size_{Lin}(\varphi)$	*Linear Size*
Specificity:	$MC^{\forall}_{finiteDensity}(\varphi, 10)$	*Universal Finite Density*
Ranking:	$Size_{Lin}(\varphi) \cdot MC^{\forall}_{finiteDensity}(\varphi, 10)$	

Universal Ranking Function: Finite Density and Exponential Size

Simplicity:	$Size_{Exp}(\varphi)$	*Exponential Size*
Specificity:	$MC^{\forall}_{finiteDensity}(\varphi, 10)$	*Universal Finite Density*
Ranking:	$Size_{Exp}(\varphi) \cdot MC^{\forall}_{finiteDensity}(\varphi, 10)$	

We experience, that the *Finite Density and Exponential* score is the only score, that is able to mine the property $\Diamond a$ out of these three functions, taking 1000 examples into account. However, *Finite Density and Exponential* is not able to mine $\neg(a \mathcal{U} (b \mathcal{U} c))$ in the Top 5, compared to the other functions. In particular, for less specific specifications like $\Diamond a$, *Finite Density and Exponential* is working the best. However, for more

restrictive properties, both *Finite Density and Linear Size* as well as *Finite Counting and Mapped Size* have better results. We experience, that the benchmark properties were in general ranked higher for *Finite Counting and Mapped Size*.

Therefore, depending on the restrictiveness of the property, we need to choose a different ranking function to be able to mine the property. For still restrictive properties like $a \mathcal{U} b$, $a \mathcal{U} (b \mathcal{U} c)$, the universal ranking function *Finite Counting and Mapped Size* is suitable. It finds a trade-off between simplicity and specificity since both scores have the same range. As *Finite Counting and Mapped Size* has the underlying assumption, that the property is rather restrictive, we are not able to mine unrestricted properties like guarantee properties, for example, $\Diamond a$. In order to mine unrestricted properties, we need a different ranking function, for example *Finite Density and Exponential*. In addition, to mine unrestricted properties, we are dependent on a higher amount of traces in order to rule out the coincidental similarity between the generated traces.

5.5 LTL Specification Mining

Our LTL specification mining approach is evaluated on the benchmark defined in Section 5.1. This setting is further divided into ranking functions based on the density of a specific bound and the asymptotic density. In this section, we first discuss the results of the following ranking function in more detail:

Ranking Function Infinite: Density of Bound 5 with Linear Size

Simplicity:	$Size_{Lin}(\varphi)$	*Linear Size*
Specificity:	$MC_{density}(\varphi, 5)$	*Density of Bound*
Ranking:	$Size_{Lin}(\varphi) \cdot MC_{density}(\varphi, 5)$	

This ranking function *Density of Bound 5 with Linear Size* has the best result out of the tested ranking functions. For the benchmark by Ehlers et al. [9], we are able to mine 6 out of 10 in the Top 5 specifications using only 50 traces. These are 4 out of the 6 safety properties as well as 2 out of 4 liveness properties. This is the first of our ranking functions to mine liveness properties.

Using only 50 traces, we are not able to mine the property $\square(a \rightarrow (b \vee c \vee d))$, as the traces generated coincidentally fulfill properties with a lower specificity as well as a lower simplicity score for *Density of Bound 5 with Linear Size*. Important to notice is, that the property $\square(a \rightarrow (b \vee c \vee d))$ takes 4 atomic propositions into account, while most other properties take only two or three. Hence, coincidental similarity is created more easily with only 50 traces. With enough traces, these coincidental properties are ruled out. In addition, the liveness properties $\square(a \rightarrow \Diamond b)$ and $\square((a \wedge b) \rightarrow \bigcirc \Diamond (\neg c))$

Formulas	Density of Bound 5 with Linear Size			
	Formula	Score	Simp	Spec
$\square(a \rightarrow (b \vee c \vee d))$	1. $(d\,\mathcal{W}(\bigcirc(c \wedge d)))$	1.42	0.74	0.68
	2. $(\square((c\,\mathcal{U}\,d)\,\mathcal{W}\,b))$	1.55	0.74	0.82
	3. $(d\,\mathcal{W}(\bigcirc d))$	1.86	0.45	1.42
	4. $(d\,\mathcal{U}(\bigcirc d))$	1.86	0.45	1.42
	5. $(c\,\mathcal{U}\,d)$	1.90	0.32	1.58
$\square(a \rightarrow (b \vee c))$	1. $(\square(\mathbf{a} \rightarrow (\mathbf{b} \vee \mathbf{c})))$	1.77	0.74	1.04
	2. $(\diamondsuit(c \wedge (\bigcirc(\bigcirc c))))$	2.33	0.74	1.59
	3. $(\diamondsuit(b \wedge (\bigcirc c)))$	2.54	0.58	1.95
	4. $((a \rightarrow c)\,\mathcal{U}\,b)$	2.86	0.58	2.28
	5. $(b\,\mathcal{W}(\bigcirc((c\,\mathcal{W}\,0)\,\mathcal{U}(\bigcirc c))))$	2.89	1.32	1.57
$\square(\bigcirc\neg a \rightarrow (\neg b \leftrightarrow \bigcirc(\neg b)))$	1. $(\square(\diamondsuit(a \wedge b)))$	1.71	0.58	1.13
	2. $((\diamondsuit(a \wedge b))\,\mathcal{W}\,0)$	1.86	0.74	1.13
	3. $(\square(\diamondsuit(\neg(a \rightarrow b))))$	1.86	0.74	1.13
	4. $(\square(\diamondsuit(a \wedge (\neg b))))$	1.86	0.74	1.13
	5. $(\square(\diamondsuit(a\,\mathcal{U}(a \wedge b))))$	2.03	0.91	1.13
$\square(\neg(a \wedge b))$	1. $(\square(a \rightarrow (\neg b)))$	0.98	0.58	0.39
	2. $(\square(\neg(\mathbf{a} \wedge \mathbf{b})))$	0.98	0.58	0.39
	3. $(\neg(\diamondsuit(a \wedge b)))$	0.98	0.58	0.39
	4. $(\square(b \rightarrow (\neg a)))$	0.98	0.58	0.39
	5. $(\square((\neg a)\,\mathcal{W}(\neg b)))$	1.13	0.74	0.39
$\square(a \rightarrow \neg(b \vee c))$	1. $(\square(\neg(a \wedge c)))$	0.98	0.58	0.39
	2. $(\square(a \rightarrow (\neg b)))$	0.98	0.58	0.39
	3. $(\square(\neg(a \wedge b)))$	0.98	0.58	0.39
	4. $(\neg(\diamondsuit(a \wedge c)))$	0.98	0.58	0.39
	5. $(\square(\mathbf{a} \rightarrow (\neg(\mathbf{b} \vee \mathbf{c}))))$	1.05	0.91	0.14
$\square a$	1. $(\square\mathbf{a})$	0.25	0.21	0.05
	2. $(a\,\mathcal{W}\,0)$	0.37	0.32	0.05
	3. $(\square(\bigcirc a))$	0.41	0.32	0.08
	4. $(\bigcirc(\square a))$	0.41	0.32	0.08
	5. $(a\,\mathcal{W}(\bigcirc 0))$	0.49	0.45	0.05
$\square(a \rightarrow \diamondsuit b)$	1. $(\diamondsuit((\neg(\bigcirc b)) \wedge (\bigcirc a)))$	2.64	0.91	1.73
	2. $(\diamondsuit(\neg(a \rightarrow b)))$	2.66	0.58	2.08
	3. $(\diamondsuit(a \wedge (\neg b)))$	2.66	0.58	2.08
	4. $(\bigcirc(\diamondsuit(\neg((\neg a)\,\mathcal{U}\,b))))$	2.80	0.91	1.89
	5. $(1\,\mathcal{U}(a \wedge (\neg b)))$	2.81	0.74	2.08
$\square((a \wedge b) \rightarrow \bigcirc\diamondsuit(\neg c))$	1. $(\diamondsuit(\bigcirc(a \wedge b)))$	2.32	0.58	1.73
	2. $(\diamondsuit((\bigcirc b) \wedge (\bigcirc a)))$	2.47	0.74	1.73
	3. $(\diamondsuit(a \wedge b))$	2.52	0.45	2.08
	4. $(\square(\diamondsuit b))$	2.69	0.32	2.37
	5. $((\diamondsuit a)\,\mathcal{U}(a \wedge b))$	2.81	0.74	2.08
$\square(\diamondsuit a)$	1. $(\square(\diamondsuit\mathbf{a}))$	2.69	0.32	2.37
	2. $(\square(\bigcirc(\diamondsuit a)))$	2.82	0.45	2.37
	3. $(\square(1\,\mathcal{U}\,a))$	2.82	0.45	2.37
	4. $(\diamondsuit(\square(\diamondsuit a)))$	2.82	0.45	2.37
	5. $(\bigcirc(\square(\diamondsuit a)))$	2.82	0.45	2.37
$\square(\diamondsuit(\neg(a \vee b)))$	1. $(\square(\diamondsuit(\neg(\mathbf{a} \vee \mathbf{b}))))$	1.86	0.74	1.13
	2. $(\neg(\diamondsuit(\square(a\,\mathcal{U}\,b))))$	2.09	0.74	1.35
	3. $(\square(\neg(\diamondsuit(\square(a\,\mathcal{U}\,b)))))$	2.26	0.91	1.35
	4. $((\diamondsuit(\square(a \vee b))) \rightarrow (\square b))$	2.33	1.10	1.23
	5. $(\square(\diamondsuit(\neg((a \vee b)\,\mathcal{W}(\bigcirc 0)))))$	2.45	1.32	1.13

Table 5.6: Results for Density of Bound 5, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al.[9]

Formulas	Density of Bound 5 with Linear Size			
	Formula	Score	Simp	Spec
Absence				
$\square(\neg a)$	1. $(\neg(\diamondsuit a))$	0.37	0.32	0.05
	2. $(\square(\neg \mathbf{a}))$	0.37	0.32	0.05
	3. $(\square(\neg(\diamondsuit a)))$	0.49	0.45	0.05
	4. $((\neg a)\,\mathcal{W}\,0)$	0.49	0.45	0.05
	5. $(\neg(1\,\mathcal{U}\,a))$	0.49	0.45	0.05
$\square(a \rightarrow \square(\neg b))$	1. $((\neg a)\,\mathcal{W}(\neg(\diamondsuit b)))$	0.87	0.74	0.13
	2. $(\square(\mathbf{a} \rightarrow (\square(\neg \mathbf{b}))))$	0.87	0.74	0.13
	3. $(\square(a \rightarrow (\neg b)))$	0.98	0.58	0.39
	4. $(\neg(\diamondsuit(a \wedge b)))$	0.98	0.58	0.39
	5. $(\square(b \rightarrow (\neg a)))$	0.98	0.58	0.39
$\diamondsuit a \rightarrow (\neg b\,\mathcal{U}\,a)$	1. $((\diamondsuit \mathbf{a}) \rightarrow ((\neg \mathbf{b})\,\mathcal{U}\,\mathbf{a}))$	2.63	0.91	1.72
	2. $((\neg b)\,\mathcal{W}(a \leftrightarrow (\diamondsuit a)))$	2.63	0.91	1.72
	3. $((\diamondsuit a) \rightarrow ((\neg b)\,\mathcal{W}\,a))$	2.63	0.91	1.72
	4. $((b \rightarrow (\bigcirc 0))\,\mathcal{U}(a \leftrightarrow (\diamondsuit a)))$	3.05	1.32	1.72
	5. $(a \vee ((\neg(\diamondsuit a))\,\mathcal{W}(\neg b)))$	3.19	1.10	2.09
Universality				
$\square a$	1. $(\square \mathbf{a})$	0.25	0.21	0.05
	2. $(a\,\mathcal{W}\,0)$	0.37	0.32	0.05
	3. $(\square(\bigcirc a))$	0.41	0.32	0.08
	4. $(\bigcirc(\square a))$	0.41	0.32	0.08
	5. $(a\,\mathcal{W}(\bigcirc 0))$	0.49	0.45	0.05
$\square(a \rightarrow \square b)$	1. $(\square(\mathbf{a} \rightarrow (\square \mathbf{b})))$	0.72	0.58	0.13
	2. $((\neg a)\,\mathcal{W}(\square b))$	0.72	0.58	0.13
	3. $(\square(a \rightarrow b))$	0.84	0.45	0.39
	4. $((\neg a)\,\mathcal{W}(\bigcirc(\square b)))$	0.92	0.74	0.18
	5. $(\square(a \rightarrow (\bigcirc(\square b))))$	0.92	0.74	0.18
$\diamondsuit a \rightarrow (b\,\mathcal{U}\,a)$	1. $((\diamondsuit \mathbf{a}) \rightarrow (\mathbf{b}\,\mathcal{U}\,\mathbf{a}))$	2.46	0.74	1.72
	2. $(((\diamondsuit a) \rightarrow ((\neg a) \leftrightarrow b))\,\mathcal{W}\,a)$	3.05	1.32	1.72
	3. $((\square b) \vee (\square(\diamondsuit(\neg a))))$	3.32	0.91	2.41
	4. $(\square((\square a) \rightarrow b))$	3.33	0.58	2.75
	5. $(\square(a \rightarrow (\diamondsuit b)))$	3.33	0.58	2.75
Existence				
$\diamondsuit a$	1. $(\diamondsuit(b \wedge (\bigcirc b)))$	2.13	0.58	1.54
	2. $(\bigcirc(\diamondsuit(a \wedge b)))$	2.32	0.58	1.73
	3. $(\bigcirc(1\,\mathcal{U}(a \wedge b)))$	2.47	0.74	1.73
	4. $(\diamondsuit(a \wedge b))$	2.52	0.45	2.08
	5. $(\diamondsuit(a \wedge (\bigcirc b)))$	2.54	0.58	1.95
$\square(\neg a) \vee \diamondsuit(a \wedge (\diamondsuit b))$	1. $(\square(a \rightarrow (\diamondsuit b)))$	3.33	0.58	2.75
	2. $(\bigcirc(\diamondsuit(\bigcirc(\bigcirc b))))$	3.38	0.58	2.80
	3. $(\bigcirc(\bigcirc(\bigcirc(\diamondsuit b))))$	3.38	0.58	2.80
	4. $(\diamondsuit(b \wedge (a \leftrightarrow (\bigcirc(\bigcirc a)))))$	3.39	1.10	2.29
	5. $((\diamondsuit b)\,\mathcal{W}(\neg(\diamondsuit a)))$	3.48	0.74	2.75
$(\square(a \wedge ((\neg b) \rightarrow ((\neg b)\,\mathcal{U}(\neg b \wedge c)))))$	1. $(\square a)$	0.25	0.21	0.05
	2. $(a\,\mathcal{W}\,0)$	0.37	0.32	0.05
	3. $(\square(\bigcirc a))$	0.41	0.32	0.08
	4. $(\bigcirc(\square a))$	0.41	0.32	0.08
	5. $((\square a)\,\mathcal{W}\,0)$	0.49	0.45	0.05

Table 5.7: Results for Density of Bound 5, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al.[28]

are difficult to mine, as these properties are the most unrestrictive properties out of the benchmark.

Important to notice is, that by using 100 traces, we are able to mine 7 out of 10 traces and with 1000 traces we can mine all specifications. In comparison, Ehlers et al. [9] evaluated their approach using 50,000 traces. The ranking result for 50 traces is presented in Figure 5.6. Compared to the setting with finite traces, we are able to mine the liveness properties $\square\diamondsuit a$ as well as $\square\diamondsuit \neg(a \vee b)$ using the density for bound five. Important to notice is, that these properties have an asymptotic density of 1, and are hence, more difficult to mine.

For the benchmark used by Roy et al. [28], we are able to mine again all absence and universality properties in the Top 5 ranking using only 50 traces. By incorporating 1,000 traces, we are able to mine 8 out of the 9 properties. For the last property, the subproperty $\square a$ as well as equivalent specifications represent the Top 5 specifications for this specification mining problem. Still, we need fewer examples than Roy et al. [28] to mine the properties. The ranking result for 50 traces is presented in Figure 5.7.

We experience, that the finite approaches ranked the invariant and restrictive properties higher in the ranking. However, compared to the finite setting, we are now able to mine all the properties of the benchmark of Ehlers et al. [9] if we incorporate more traces, for example, 1,000 traces. Ehlers et al.[9] evaluated their approach using up to 50,000 traces to be able to mine these properties. Compared to Roy et al. [28], we are able to mine 8 out of the 9 properties when including 1,000 traces. Important to notice is, that we are capable of mining even liveness properties with an asymptotic density of 1, even with only 50 traces.

We now consider the results of a ranking function based on the asymptotic density.

Ranking Function Infinite: Density of Bound 10 with Linear Size

Simplicity:	$Size_{Lin}(\varphi)$	*Linear Size*
Specificity:	$MC_{density}(\varphi, 10)$	*Density of Bound*
Ranking:	$Size_{Lin}(\varphi) \cdot MC_{density}(\varphi, 10)$	

This ranking function *Density of Bound 10 with Linear Size* is harder to compare since formulas with at least 3 atomic propositions timed out. As a timeout, we chose 30 minutes. We experience, that the function *Density of Bound 10 with Linear Size* produces results approximating the scores for the ranking function based on the asymptotic density. Therefore, we experience, that the function is able to mine $\square\diamondsuit a$ in the ranking, however, it is not able to mine $\square\diamondsuit \neg(a \vee b)$ anymore. Compared to

the density for bound 5 it is able to mine $\square(a \rightarrow (b \wedge c \wedge d))$ in the ranking. Still, the ranking function *Density of Bound 5 with Linear Size* produces overall better results. Detailed results are presented in the Appendix A.3 as well as A.4.

Ranking Function Infinite: Asymp. Density with Linear Size

Simplicity:	$Size_{Lin}(\varphi)$	*Linear Size*
Specificity:	$MC_{asympDensity}(\varphi)$	*Asymp. Density*
Ranking:	$Size_{Lin}(\varphi) \cdot MC_{density}(\varphi)$	

With the function *Asymp. Density with Linear Size*, we are able to mine most of the safety, absence and universality properties. For the safety properties from the benchmark by Ehlers et al. [9], we are able to mine one specification more compared to the ranking function *Density of Bound 5 with Linear Size*, the ranking positions are however quite similar. The ranking positions for the benchmarks are displayed in Table 5.8 and 5.9. However, with the ranking function based on the asymptotic density we are not able to mine liveness properties, as they have an asymptotic density of 1. All liveness properties, which still satisfy the traces have the same density score of 1 and hence, that same ranking value of 0. This corresponds to the value ∞ for the table entries as $-\log_2(0) = \infty$.

To be more precise, the ranking function based on the asymptotic density is able to mine safety properties. However, compared to the function *Density of Bound 5 with Linear Size*, we are not able to mine liveness properties since all liveness properties have a ranking score of 0. Therefore, the ranking function *Density of Bound 5 with Linear Size* produces overall better results.

Absence	Rank	Existence	Rank	Universality	Rank
$\square(\neg a)$	2	$\diamondsuit a$	-	$\square a$	1
$\square(a \rightarrow \square(\neg b))$	-	$(\square(\neg a)) \vee (\diamondsuit(a \wedge (\diamondsuit b)))$	-	$\square(a \rightarrow \square b)$	4
$\diamondsuit a \rightarrow (\neg b \mathcal{U}\, a)$	1	$\square(a \wedge ((\neg b) \rightarrow ((\neg b)\mathcal{U}((\neg b) \wedge c))))$	-	$\diamondsuit a \rightarrow (b \mathcal{U}\, a)$	1

Table 5.8: Ranking for Asymp. Density with Linear Size, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al. [28]

Property	Rank
1. $\square(a \rightarrow (b \vee c \vee d))$	2
2. $\square(a \rightarrow (b \vee c))$	1
3. $\square(\bigcirc \neg a \rightarrow (\neg b \leftrightarrow \bigcirc(\neg b)))$	5
4. $\square(a \rightarrow \neg b)$	2
5. $\square(a \rightarrow (\neg b \wedge \neg c))$	-
6. $\square a$	1
7. $\square(a \rightarrow \diamondsuit b)$	-
8. $\square((a \wedge b) \rightarrow \bigcirc\diamondsuit(\neg c))$	-
9. $\square\diamondsuit a$	-
10. $\square\diamondsuit(\neg a \wedge \neg b)$	-

Table 5.9: Ranking for Asymp. Density with Linear Size, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al. [9]

5.6 Conclusion

Comparison of the Settings. To wrap it up, we looked into three settings. With the ranking functions of the existential LTL$_f$ setting, we are able to mine all safety properties of the benchmark in the first position of the ranking, however, we are not able to mine liveness properties. For the universal setting, we are in general only able to mine properties with good-prefixes. We might need more examples here, especially if the property is less restrictive. For the infinite setting, we experience, that we are able to mine safety properties. In addition, we are able to even mine liveness properties, like $\square\diamondsuit a$ and $\square\diamondsuit \neg(a \vee b)$ which have an asymptotic density of 0 and are hence rather unrestrictive properties.

Comparison to Ehlers et al. [9]. With the function *Density of Bound 5 with Linear Size* we experience, that we are able to mine 6 out of the 10 properties using only 50 traces. To mine the missing four properties, we require 1,000 traces. In comparison, Ehlers et al. evaluated their approach using up to 50,000 examples. They used traces with a prefix length of 0 to 4 and loop length of 1 to 4, while we sampled traces with a lower bound for the length of 0 to 9 for the prefix and 1 to 10 for the loop. For the safety properties, we experience, that we have even better results using finite traces and evaluating the properties with the existential ranking function *Finite Counting and Mapped Size*. Here we are able to mine all safety properties on position 1.

Comparison to Roy et al. [28]. With the function *Density of Bound 5 with Linear Size*, we are able to mine the universality and absence properties with 50 traces. For the existential properties, we notice, that we need many more examples, as these properties are not restrictive and taking only a few examples leads to coincidental

connections. Here, we are however able to mine $\Diamond a$ as well as $\Box(\neg a) \vee \Diamond(a \wedge \Diamond b)$ using 1,000 examples. In comparison, Roy et al. took 10,000 finite traces of length 10.

Chapter 6

Related Work

Specification mining describes the process of learning a formal specification from given data. Learning algorithms for specification mining can be divided into multiple settings.

Interactive Query Learning Algorithms. Early work is based on interactive query learning algorithms. Here, Angluin introduced the interactive, polynomial learning algorithm $L*$, which learns an unknown regular language using membership and equivalence queries [1]. $L*$ is based on a minimally adequate teacher representing the unknown regular set, which answers these membership and equivalence queries. Angluin's approach was extended by Maler and Pnueli [21] to a subclass of the ω-regular languages, which can be recognized by a deterministic Büchi and a deterministic co-Büchi automaton. Another extension by Angluin and Fisman [2], provides an algorithm with three variations, which extends $L*$ to the full class of regular ω-languages.

Specification Mining for Signal Temporal Logic. Signal Temporal Logic (STL) [22] is an extension of Linear Temporal Logic (LTL) [25]. It was introduced by Maler and Nickovic in 2004 and is used to express properties of continuous real-valued signals. It is widely used for the analysis of programs in cyber-physical systems. There exist several approaches for mining STL specification. Bartocci et al. [4], provide an overview of the variety of approaches by covering influential techniques and comparing them. For Metric Temporal Logic, a special case of STL, specification mining has been recently explored by Raha et al. [26].

Specification Mining from Positive and Negative Examples. Several work on specification mining has been proposed using both positive and negative examples as input data. In this context, Gold has shown the important property of specification mining for deterministic finite automata (DFAs), that finding a consistent DFA of a

fixed size is NP-complete [15]. Despite this result, there still exist DFA identification algorithms, some of them are summarized in [6]. In addition, a variety of specification mining algorithms is based on SAT-solver, e.g. Heule and Verwer introduced an algorithm for identification of DFAs [18] as well as Neider and Gavran, who stated two algorithms for learning LTL [23]. Both algorithms introduced by Neider and Gavran are SAT-based, however, the second algorithm extends the first algorithm with a learning algorithm for decision trees.

Specification Mining from Positive Examples Only. Since negative examples are often hard to obtain, there is interest in algorithms receiving positive examples only as input. We divided this category of algorithms further into smaller subcategories.

Template Based Specification Mining. In the context, where only positive examples are provided e.g. in the form of a log of traces, specification mining is often approached via templates. For LTL specification mining, Lemieux et al. [19] introduced the tool *Texada*. It takes a log file containing various traces and a user-defined LTL formula, where atomic propositions are replaced with variables. Texada outputs the set of all matching LTL formulas, with atomic propositions replacing the variables. Gabel and Su provided different algorithms for template based specification mining of regular languages. Their tool Javert [12] learns complex temporal properties from execution traces only, hence no user-given template is needed. They provided a symbolic algorithm [13], which relies on binary decision diagrams, as well as an online algorithm [14].

Bound-based Specification Mining of Positive Examples Only. A range of algorithms working with positive examples only makes use of a bound in order to restrict the size of the mined specification and to deal with the unrestricted search space. In this context, properties in LTL $\cap$ ACTL can be learned from ultimately periodic words uv^ω [9] in the form of universal very-weak automata (UVWs). The learning algorithm depends on the tightness value n, which states, that all simple chains of states of the UVW have a length of at most n. Roy et al. [28] proposed an algorithm for specification mining for linear temporal logic on finite traces (LTL_f) in 2023. Their approach focuses on the notion of language minimality in combination with restricting the size of the LTL_f formula to an upper bound n. It is a SAT-based algorithm, that translates the LTL_f specification learning problem to propositional logic and is solved by a SAT-solver.

Specification Mining via Statistical Learning. Besides the template-based and bound-based specification mining algorithms, there exist algorithms relying on statistical learning. Pertseva et al. proposed an algorithm [24] for mining regular expressions from few, positive examples, which is based on minimum description length (MDL) learning [27]. MDL learning is applied in order to examine the quality of the regular

expression and to find a trade-off between specificity and simplicity. In addition, it makes use of Version Space Algebras (VSAs) to represent the space of all possible regular expressions. The best regular expression corresponds to the cheapest path of the VSA through a DAG.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we introduced algorithms to mine LTL specifications. We differentiate between LTL_f specification mining and traditional LTL specification mining. On the one hand, LTL_f deals with finite traces. We distinguish existential and universal semantics to reasons about finite traces. On the other hand, LTL reasons about infinite traces. We approach the specification mining problem by using ranking functions based on the Minimum Description Length principle. These ranking functions are divided into a specificity and a simplicity score. For the simplicity score, we focus on the notion of size of an LTL formula and for the specificity, we consider model counting. We provide the theoretical ideas with the algorithms and the implementation. We evaluate the approach based on benchmarks used by Roy et al. [28] and Ehlers et al. [9]. With our approach, we are able to mine many safety properties and liveness properties with only 50 traces. By incorporating 1,000 traces, we are able to learn all properties used by Ehlers et al. [9]. Therefore, we are able to learn the respective properties with a lot less data than Ehlers et al. [9] and Roy et al. [28]. We conclude, that the statistical learning approach performs well in the context of mining specifications with few examples.

7.2 Future Work

LTL - Search Space. The main limitation of our approach is the restricted search space. While approaches from Roy et al. [28] and Ehlers et al. [9] are both SAT-based and take the whole LTL search space into account, our approach only uses randomly generated formulas. In order to be able to mine a greater variability of LTL formulas, we are interested in expanding the search space. Especially, since the idea of ranking

formulas has the potential to be successful in a larger search space.

LTL - Ranking Function. In this thesis, we focused for the simplicity function only on variations to define the size of the LTL formula. It would be interesting to look at different notions of simplicity. Here, a probabilistic context-free grammar could be used to reason about the simplicity of a formula. For the specificity, another idea would be to use the cardinality of an LTL formula for a fixed length instead of the density for the fixed length. We have seen, that for the finite traces, the best ranking function was based on the finite cardinality of the formula instead of the finite version of the density. Hence, it would be interesting to evaluate whether the cardinality is suitable for the specificity score.

ω-Regular Expressions. Pertseva et al. [24] apply MDL learning for regular expressions while representing the search space for regular expressions via Version Space Algebras. Using this search space construction, they are able to mine the best property from a logically defined structure. For future work, it would be interesting to work on constructing a search space for ω-regular expressions. In addition, Pertseva et al. computed the best regular expressions by using a compositional formulation of the MDL approach. They assigned each edge of the VSA a weight. The weight of the regular expression corresponds to the sum of the weight of its atomic components. It would be interesting to apply the same approach for ω-regular expressions. One idea to represent the simplicity is using a probabilistic context-free grammar as Pertseva et al. did. For the specificity, future work could focus on approaches to represent a specificity score via a compositional approach.

Bibliography

[1] Dana Angluin. Learning regular sets from queries and counterexamples. **Information and Computation**, 75(2):87–106, 1987. ISSN 0890-5401. doi: https://doi.org/10.1016/0890-5401(87)90052-6. URL https://www.sciencedirect.com/science/article/pii/0890540187900526.

[2] Dana Angluin and Dana Fisman. Learning regular omega languages. **Theoretical Computer Science**, 650:57–72, 2016. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2016.07.031. URL https://www.sciencedirect.com/science/article/pii/S0304397516303760. Algorithmic Learning Theory.

[3] Christel Baier and Joost-Pieter Katoen. **Principles of Model Checking**. The MIT Press, 2008. ISBN 026202649X. URL http://www.amazon.com/Principles-Model-Checking-Christel-Baier/dp/026202649X%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D026202649X.

[4] Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Nickovic. Survey on mining signal temporal logic specifications. **Information and Computation**, 289:104957, 2022. ISSN 0890-5401. doi: https://doi.org/10.1016/j.ic.2022.104957. URL https://www.sciencedirect.com/science/article/pii/S0890540122001122.

[5] Raven Beutner. **FSOmegaLib**. URL https://github.com/ravenbeutner/FsOmegaLib.

[6] Colin de la Higuera. A bibliographical study of grammatical inference. **Pattern Recognition**, 38(9):1332–1348, 2005. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2005.01.003. URL https://www.sciencedirect.com/science/article/pii/S0031320305000221. Grammatical Inference.

[7] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, An-

toine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In **Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)**, volume 13372 of **Lecture Notes in Computer Science**, pages 174–187. Springer, August 2022. doi: 10.1007/978-3-031-13188-2_9.

[8] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In **Proceedings of the 21st International Conference on Software Engineering**, ICSE '99, page 411–420, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581130740. doi: 10.1145/302405.302672. URL `https://doi.org/10.1145/302405.302672`.

[9] Rüdiger Ehlers, Ivan Gavran, and Daniel Neider. Learning properties in ltl ∩ actl from positive examples only. **2020 Formal Methods in Computer Aided Design (FMCAD)**, pages 104–112, 2020. URL `https://api.semanticscholar.org/CorpusID:221989548`.

[10] Bernd Finkbeiner and Hazem Torfah. The density of linear-time properties. In Deepak D'Souza and K. Narayan Kumar, editors, **Automated Technology for Verification and Analysis**, pages 139–155, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68167-2.

[11] Bernd Finkbeiner, Felix Klein, and Tobias Salzmann. **Automata, Games and Verification Lecture Script**. URL `https://www.react.uni-saarland.de/teaching/automata-games-verification-15/downloads/notes.pdf`.

[12] Mark Gabel and Zhendong Su. Javert: Fully automatic mining of general temporal properties from dynamic traces. In **Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering**, SIGSOFT '08/FSE-16, page 339–349, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781595939951. doi: 10.1145/1453101.1453150. URL `https://doi.org/10.1145/1453101.1453150`.

[13] Mark Gabel and Zhendong Su. Symbolic mining of temporal specifications. In **2008 ACM/IEEE 30th International Conference on Software Engineering**, pages 51–60, 2008. doi: 10.1145/1368088.1368096.

[14] Mark Gabel and Zhendong Su. Online inference and enforcement of temporal properties. In **2010 ACM/IEEE 32nd International Conference on Software Engineering**, volume 1, pages 15–24, 2010. doi: 10.1145/1806799.1806806.

[15] E Mark Gold. Complexity of automaton identification from given data. **Information and Control**, 37(3):302–320, 1978. ISSN 0019-9958. doi: https:

//doi.org/10.1016/S0019-9958(78)90562-4. URL https://www.sciencedirect.com/science/article/pii/S0019995878905624.

[16] Angelina Göbl. **Implementation**. URL https://github.com/AngelinaGoebl/ltllearning.

[17] Mor Harchol-Balter. **Introduction to Probability for Computing**. Cambridge University Press, 2023. doi: 10.1017/9781009309097.

[18] Marijn Heule and Sicco Verwer. Exact dfa identification using sat solvers. pages 66–79, 09 2010. ISBN 978-3-642-15487-4. doi: 10.1007/978-3-642-15488-1_7.

[19] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General ltl specification mining (t). In **2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)**, pages 81–92, 2015. doi: 10.1109/ASE.2015.71.

[20] David J. C. MacKay. **Information Theory, Inference, and Learning Algorithms**. Copyright Cambridge University Press, 2003.

[21] O. Maler and A. Pnueli. On the learnability of infinitary regular sets. **Information and Computation**, 118(2):316–326, 1995. ISSN 0890-5401. doi: https://doi.org/10.1006/inco.1995.1070. URL https://www.sciencedirect.com/science/article/pii/S089054018571070X.

[22] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, **Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems**, pages 152–166, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30206-3.

[23] Daniel Neider and Ivan Gavran. Learning linear temporal properties. **CoRR**, abs/1806.03953, 2018. URL http://arxiv.org/abs/1806.03953.

[24] Elizaveta Pertseva, Mark Barbone, Joey Rudek, and Nadia Polikarpova. Regex+: Synthesizing regular expressions from positive examples. **11TH Workshop on Synthesis**. URL https://par.nsf.gov/biblio/10336574.

[25] Amir Pnueli. The temporal logic of programs. In **FOCS 1977**, pages 46–57. IEEE Computer Society, 1977. doi: 10.1109/SFCS.1977.32. URL https://doi.org/10.1109/SFCS.1977.32.

[26] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, Daniel Neider, and Guillermo A Pérez. Synthesizing efficiently monitorable formulas in metric temporal logic. **arXiv preprint arXiv:2310.17410**, 2023.

[27] J. Rissanen. Modeling by shortest data description. **Automatica**, 14(5):465–471, 1978. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(78)90005-5. URL `https://www.sciencedirect.com/science/article/pii/0005109878900055`.

[28] Rajarshi Roy, Jean-Raphaël Gaglione, Nasim Baharisangari, Daniel Neider, Zhe Xu, and Ufuk Topcu. Learning interpretable temporal properties from positive examples only, 2023.

[29] Fabio Somenzi and Roderick Bloem. Efficient büchi automata from ltl formulae. volume 1855, 05 2000. ISBN 978-3-540-67770-3. doi: 10.1007/10722167_21.

Appendix A

Appendix

A.1 Universal Setting

Formulas	Finite Density and Linear Size				Finite Density and Exponential Size				Finite Counting and Mapped Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
$\Diamond a$	1. $(\bigcirc((\Diamond b)\,\mathcal{U}(\neg a)))$	8.15	0.74	7.42	1. $(\Diamond b)$	12.00	2.00	10.00	1. $(\bigcirc((\Diamond(\bigcirc(\Diamond b)))\,\mathcal{U}(a \leftrightarrow (\Diamond(\Box b)))))$	39.98	20.00	19.98
	2. $(\bigcirc((\Diamond(\bigcirc(\Diamond b)))\,\mathcal{U}(a \leftrightarrow (\Diamond(\Box b)))))$	8.32	1.91	6.42	2. $(\Diamond a)$	12.00	2.00	10.00	2. $((\bigcirc(\Diamond b))\,\mathcal{W}(a \leftrightarrow (\Box((\Diamond a) \rightarrow b))))$	39.98	20.00	19.99
	3. $(\neg(a\,\mathcal{W}(\neg(1\,\mathcal{U}\,b))))$	8.32	0.91	7.42	3. $(\bigcirc(\Diamond a))$	12.00	3.00	9.00	3. $(\bigcirc((\Diamond b)\,\mathcal{U}(\neg a)))$	39.98	19.99	19.99
	4. $((((\Diamond a)\,\mathcal{U}\,b)\,\mathcal{U}(\neg a))$	8.32	0.91	7.42	4. $(\Diamond(\bigcirc a))$	12.00	3.00	9.00	4. $(\neg(a\,\mathcal{W}(\neg(1\,\mathcal{U}\,b))))$	39.98	19.99	19.99
	5. $((\bigcirc(\Diamond b))\,\mathcal{W}(a \leftrightarrow (\Box((\Diamond a) \rightarrow b))))$	8.51	1.91	6.61	5. $((\Diamond a)\,\mathcal{W}\,b)$	12.30	4.00	8.30	5. $((((\Diamond a)\,\mathcal{U}\,b)\,\mathcal{U}(\neg a))$	39.98	19.99	19.99

Table A.1: Results for the universal setting, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 1000, own benchmark

Formulas	Finite Density and Linear Size				Finite Density and Exponential Size				Finite Counting and Mapped Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
$\Diamond a$	1. $((b\,\mathcal{W}(\bigcirc(\bigcirc b)))\,\mathcal{W}(\neg(\bigcirc a)))$	3.44	1.32	2.12	1. $(\Diamond((\neg a)\wedge b))$	9.15	5.00	4.15	1. $((b\,\mathcal{W}(\bigcirc(\bigcirc b)))\,\mathcal{W}(\neg(\bigcirc a)))$	39.45	19.82	19.62
	2. $(((b\to a)\,\mathcal{U}\,a)\to a)$	3.49	0.91	2.58	2. $(\Diamond(\neg(b\to a)))$	9.15	5.00	4.15	2. $(((b\to a)\,\mathcal{U}\,a)\to a)$	39.51	19.77	19.74
	3. $((a\,\mathcal{U}\,b)\,\mathcal{W}(\bigcirc(a\leftrightarrow(\bigcirc 0))))$	3.92	1.32	2.60	3. $(\neg(\Box(b\to a)))$	9.15	5.00	4.15	3. $((((\Box a)\,\mathcal{W}(a\,\mathcal{W}\,b))\,\mathcal{W}(\neg(\bigcirc a)))\,\mathcal{W}\,b)$	39.53	19.91	19.62
	4. $(((\bigcirc a)\,\mathcal{W}\,b)\,\mathcal{W}(\bigcirc(\neg a)))$	4.08	1.10	2.98	4. $(((b\to a)\,\mathcal{U}\,a)\to a)$	9.58	7.00	2.58	4. $((a\,\mathcal{U}\,b)\,\mathcal{W}(\bigcirc(a\leftrightarrow(\bigcirc 0))))$	39.56	19.82	19.74
	5. $((\neg b)\,\mathcal{U}((\neg a)\vee(\bigcirc b)))$	4.10	1.10	3.00	5. $(\Diamond(\neg(a\vee(\neg b))))$	10.15	6.00	4.15	5. $((((\Box a)\,\mathcal{W}(a\,\mathcal{W}\,b))\,\mathcal{W}(\neg(\bigcirc a)))$	39.59	19.85	19.74
$a\,\mathcal{U}\,b$	1. $(a\,\mathcal{U}\,b)$	1.91	0.32	1.58	1. $(a\,\mathcal{U}\,b)$	4.58	3.00	1.58	1. $(a\,\mathcal{U}\,b)$	38.90	19.48	19.42
	2. $(a\,\mathcal{W}\,b)$	1.91	0.32	1.58	2. $(a\,\mathcal{W}\,b)$	4.58	3.00	1.58	2. $(a\,\mathcal{W}\,b)$	38.90	19.48	19.42
	3. $(a\,\mathcal{W}(a\,\mathcal{U}\,b))$	2.17	0.58	1.58	3. $(a\vee b)$	5.00	3.00	2.00	3. $(a\,\mathcal{W}(a\,\mathcal{U}\,b))$	38.97	19.56	19.42
	4. $((a\,\mathcal{U}\,b)\,\mathcal{W}\,b)$	2.17	0.58	1.58	4. $((\neg b)\to a)$	6.00	4.00	2.00	4. $((a\,\mathcal{U}\,b)\,\mathcal{W}\,b)$	38.97	19.56	19.42
	5. $(a\,\mathcal{W}(a\,\mathcal{W}\,b))$	2.17	0.58	1.58	5. $((\neg a)\to b)$	6.00	4.00	2.00	5. $(a\,\mathcal{W}(a\,\mathcal{W}\,b))$	38.97	19.56	19.42
$(\neg a)\,\mathcal{U}\,b$	1. $((\neg a)\,\mathcal{W}\,b)$	2.03	0.45	1.58	1. $(a\to b)$	5.00	3.00	2.00	1. $((\neg a)\,\mathcal{W}\,b)$	38.94	19.52	19.42
	2. $((\neg a)\,\mathcal{U}\,b)$	2.03	0.45	1.58	2. $((\neg a)\,\mathcal{W}\,b)$	5.58	4.00	1.58	2. $((\neg a)\,\mathcal{U}\,b)$	38.94	19.52	19.42
	3. $((a\leftrightarrow b)\,\mathcal{W}\,b)$	2.17	0.58	1.58	3. $((\neg a)\,\mathcal{U}\,b)$	5.58	4.00	1.58	3. $((a\leftrightarrow b)\,\mathcal{W}\,b)$	38.97	19.56	19.42
	4. $(b\mathcal{U}(a\leftrightarrow b))$	2.17	0.58	1.58	4. $((\neg a)\vee b)$	6.00	4.00	2.00	4. $(b\mathcal{U}(a\leftrightarrow b))$	38.97	19.56	19.42
	5. $(b\,\mathcal{W}(a\leftrightarrow b))$	2.17	0.58	1.58	5. $((a\leftrightarrow b)\,\mathcal{W}\,b)$	6.58	5.00	1.58	5. $(b\,\mathcal{W}(a\leftrightarrow b))$	38.97	19.56	19.42
$a\mathcal{U}(b\mathcal{U}\,c)$	1. $(b\mathcal{U}\,a)$	1.91	0.32	1.58	1. $(b\mathcal{U}\,a)$	4.58	3.00	1.58	1. $(b\mathcal{U}\,a)$	58.90	29.48	29.42
	2. $(a\,\mathcal{W}\,b)$	1.91	0.32	1.58	2. $(a\,\mathcal{W}\,b)$	4.58	3.00	1.58	2. $(a\,\mathcal{W}\,b)$	58.90	29.48	29.42
	3. $(b\,\mathcal{W}\,a)$	1.91	0.32	1.58	3. $(b\,\mathcal{W}\,a)$	4.58	3.00	1.58	3. $(b\,\mathcal{W}\,a)$	58.90	29.48	29.42
	4. $(a\,\mathcal{U}\,b)$	1.91	0.32	1.58	4. $(a\,\mathcal{U}\,b)$	4.58	3.00	1.58	4. $(a\,\mathcal{U}\,b)$	58.90	29.48	29.42
	5. $((\neg c)\,\mathcal{U}\,b)$	2.03	0.45	1.58	5. $(c\to b)$	5.00	3.00	2.00	5. $((\neg c)\,\mathcal{U}\,b)$	58.94	29.52	29.42
$\neg(a\mathcal{U}(b\mathcal{U}\,c))$	1. $(\neg(b\,\mathcal{W}\,c))$	1.03	0.45	0.58	1. $(\neg c)$	3.00	2.00	1.00	1. $(\neg(a\,\mathcal{W}(b\,\mathcal{W}\,c)))$	56.13	28.30	27.83
	2. $(\neg(a\,\mathcal{W}\,c))$	1.03	0.45	0.58	2. $(\neg(b\,\mathcal{W}\,c))$	4.58	4.00	0.58	2. $(\neg(a\mathcal{U}(b\mathcal{U}\,c)))$	56.13	28.30	27.83
	3. $(\neg(b\mathcal{U}\,c))$	1.03	0.45	0.58	3. $(\neg(a\,\mathcal{W}\,c))$	4.58	4.00	0.58	3. $((a\,\mathcal{W}(b\mathcal{U}\,c))\to(\Diamond(\bigcirc 0)))$	56.51	28.68	27.83
	4. $(\neg(a\,\mathcal{W}(b\,\mathcal{W}\,c)))$	1.10	0.74	0.36	4. $(\neg(b\mathcal{U}\,c))$	4.58	4.00	0.58	4. $(\neg(b\,\mathcal{W}\,c))$	56.51	28.09	28.42
	5. $(\neg(a\mathcal{U}(b\mathcal{U}\,c)))$	1.10	0.74	0.36	5. $(c\leftrightarrow(\Box b))$	5.00	4.00	1.00	5. $(\neg(a\,\mathcal{W}\,c))$	56.51	28.09	28.42

Table A.2: Results for the universal setting, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, own benchmark

A.2 Existential Setting

Formulas	Finite Counting and Regular Size				Finite Counting and Mapped Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
$\square(a \to (b \vee c \vee d))$	overflow	overflow	overflow	overflow	overflow	overflow	overflow	overflow
$\square(a \to (b \vee c))$	1. 1	30.00	0.00	30.00	1. $(\square(a \to (b \vee c)))$	56.59	28.51	28.07
	2. $(\square(a \to (b \vee c)))$	30.66	2.58	28.07	2. 1	58.07	28.07	30.00
	3. $(\Diamond a)$	31.00	1.00	30.00	3. $((a \to c)\,\mathcal{U}\,b)$	58.09	28.41	29.68
	4. $(\Diamond c)$	31.00	1.00	30.00	4. $((a \to c)\,\mathcal{W}\,b)$	58.09	28.41	29.68
	5. $(\Diamond b)$	31.00	1.00	30.00	5. $(\Diamond a)$	58.15	28.15	30.00
$\square(\bigcirc \neg a \to (\neg b \leftrightarrow \bigcirc(\neg b)))$	1. $(\square((\bigcirc(\neg a)) \to ((\neg b) \leftrightarrow (\bigcirc(\neg b)))))$	19.72	3.46	16.26	1. $(\square((\bigcirc(\neg a)) \to ((\neg b) \leftrightarrow (\bigcirc(\neg b)))))$	34.09	17.82	16.26
	2. 1	20.00	0.00	20.00	2. $(\square(b \to (b\,\mathcal{W}\,a)))$	35.95	16.84	19.11
	3. $(\Diamond a)$	21.00	1.00	20.00	3. $(\square(b\,\mathcal{W}((\neg b)\,\mathcal{U}\,a)))$	36.10	16.99	19.11
	4. $(\Diamond b)$	21.00	1.00	20.00	4. $(\square((\neg b)\,\mathcal{W}(b\,\mathcal{U}\,a)))$	36.10	16.99	19.11
	5. $(\Diamond(\bigcirc b))$	21.58	1.58	20.00	5. 1	36.26	16.26	20.00
$\square(\neg(a \wedge b))$	1. $(\square(\neg(a \wedge b)))$	18.17	2.32	15.85	1. $(\square(\neg(a \wedge b)))$	32.15	16.30	15.85
	2. $(\square(a \to (\neg b)))$	18.17	2.32	15.85	2. $(\square(a \to (\neg b)))$	32.15	16.30	15.85
	3. $(\square(b \to (\neg a)))$	18.17	2.32	15.85	3. $(\square(b \to (\neg a)))$	32.15	16.30	15.85
	4. $(\neg(\Diamond(a \wedge b)))$	18.17	2.32	15.85	4. $(\neg(\Diamond(a \wedge b)))$	32.15	16.30	15.85
	5. $((\neg(a \wedge b))\,\mathcal{W}\,0)$	18.43	2.58	15.85	5. $((\neg(a \wedge b))\,\mathcal{W}\,0)$	32.29	16.44	15.85
$\square(a \to \neg(b \vee c))$	1. $(\square(a \to (\neg(b \vee c))))$	26.03	2.81	23.22	1. $(\square(a \to (\neg(b \vee c))))$	47.24	24.02	23.22
	2. $(\neg(\Diamond(a \wedge c)))$	28.17	2.32	25.85	2. $(\neg(\Diamond(a \wedge c)))$	49.55	23.70	25.85
	3. $(\square(a \to (\neg b)))$	28.17	2.32	25.85	3. $(\square(a \to (\neg b)))$	49.55	23.70	25.85
	4. $(\square(\neg(a \wedge b)))$	28.17	2.32	25.85	4. $(\square(\neg(a \wedge b)))$	49.55	23.70	25.85
	5. $(\square(\neg(a \wedge c)))$	28.17	2.32	25.85	5. $(\square(\neg(a \wedge c)))$	49.55	23.70	25.85
$\square a$	1. $(\square a)$	11.00	1.00	10.00	1. $(\square a)$	20.11	10.11	10.00
	2. $(a\,\mathcal{W}\,0)$	11.58	1.58	10.00	2. $(a\,\mathcal{W}\,0)$	20.22	10.22	10.00
	3. $(\square(a\,\mathcal{W}\,0))$	12.00	2.00	10.00	3. $(\square(a\,\mathcal{W}\,0))$	20.35	10.35	10.00
	4. $(a\,\mathcal{W}(\square a))$	12.00	2.00	10.00	4. $(a\,\mathcal{W}(\square a))$	20.35	10.35	10.00
	5. $(\neg(\Diamond(\neg a)))$	12.00	2.00	10.00	5. $(\neg(\Diamond(\neg a)))$	20.35	10.35	10.00

Table A.3: Results for the existential semantics, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al.[9]

Formulas	Finite Counting and Regular Size				Finite Counting and Mapped Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
$\square(a \to \diamond b)$	1. 1	20.00	0.00	20.00	1. $((1 \,\mathcal{U}\, b) \,\mathcal{W}(b \to (\bigcirc a)))$	40.00	20.00	20.00
	2. $(\diamond b)$	21.00	1.00	20.00	2. $(\square(1 \,\mathcal{U}((\bigcirc 0) \to (\diamond(\square(b \,\mathcal{W}(a \leftrightarrow b)))))))$	40.00	20.00	20.00
	3. $(\diamond a)$	21.00	1.00	20.00	3. $(\diamond(\neg(1 \,\mathcal{U}\, b)))$	40.00	20.00	20.00
	4. $(\neg(\bigcirc 0))$	21.58	1.58	20.00	4. $(\diamond(\square(\bigcirc(\square a))))$	40.00	20.00	20.00
	5. $(\neg(\square b))$	21.58	1.58	20.00	5. $(\bigcirc(\diamond(\square(\bigcirc(\square(\bigcirc(\neg(1 \,\mathcal{U}(\square b)))))))))$	40.00	20.00	20.00
$\square((a \wedge b) \to \bigcirc\diamond(\neg c))$	1. 1	30.00	0.00	30.00	1. $((1 \,\mathcal{U}(\neg(c \to (\neg(a \leftrightarrow c))))) \,\mathcal{W}(\square a))$	60.00	30.00	30.00
	2. $(\diamond b)$	31.00	1.00	30.00	2. $(\square((\diamond c) \,\mathcal{U}(\neg(\square(\bigcirc b)))))$	60.00	30.00	30.00
	3. $(\diamond a)$	31.00	1.00	30.00	3. $(\diamond(\bigcirc(\neg c)))$	60.00	30.00	30.00
	4. $(\diamond c)$	31.00	1.00	30.00	4. $(\diamond(\square((\neg b) \,\mathcal{W}(\diamond((\square c) \to (\diamond a))))))$	60.00	30.00	30.00
	5. $(1 \,\mathcal{U}\, b)$	31.58	1.58	30.00	5. $(\bigcirc(1 \,\mathcal{U}(c \,\mathcal{W}(a \vee c))))$	60.00	30.00	30.00
$\square(\diamond a)$	1. 1	20.00	0.00	20.00				
	2. $(\diamond b)$	21.00	1.00	20.00	1. $((1 \,\mathcal{U}\, b) \,\mathcal{W}(b \to (\bigcirc a)))$	40.00	20.00	20.00
	3. $(\diamond a)$	21.00	1.00	20.00	2. $(\square(1 \,\mathcal{U}((\bigcirc 0) \to (\diamond(\square(b \,\mathcal{W}(a \leftrightarrow b)))))))$	40.00	20.00	20.00
	4. $(\neg(\bigcirc 0))$	21.58	1.58	20.00	3. $(\diamond(\neg(1 \,\mathcal{U}\, b)))$	40.00	20.00	20.00
	5. $(\neg(\square b))$	21.58	1.58	20.00	4. $(\diamond(\square(\bigcirc(\square a))))$	40.00	20.00	20.00
	...	...	...	...	5. $(\bigcirc(\diamond(\square(\bigcirc(\square(\bigcirc(\neg(1 \,\mathcal{U}(\square b)))))))))$	40.00	20.00	20.00
	10. $(\square(\diamond a))$	21.58	1.58	20.00				
$\square(\diamond(\neg(a \vee b)))$	1. 1	20.00	0.00	20.00	1. $((1 \,\mathcal{U}\, b) \,\mathcal{W}(b \to (\bigcirc a)))$	40.00	20.00	20.00
	2. $(\diamond a)$	21.00	1.00	20.00	2. $(\square(1 \,\mathcal{U}((\bigcirc 0) \to (\diamond(\square(b \,\mathcal{W}(a \leftrightarrow b)))))))$	40.00	20.00	20.00
	3. $(\diamond b)$	21.00	1.00	20.00	3. $(\diamond(\neg(1 \,\mathcal{U}(\bigcirc(\bigcirc a)))))$	40.00	20.00	20.00
	4. $(\diamond(\bigcirc a))$	21.58	1.58	20.00	4. $(\diamond(\square(\bigcirc(\square((\diamond(\bigcirc 0)) \to b)))))$	40.00	20.00	20.00
	5. $(\bigcirc(\diamond b))$	21.58	1.58	20.00	5. $(\bigcirc(\diamond(\square(\diamond a))))$	40.00	20.00	20.00

Table A.4: Results for the existential semantics, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al.[9]

Formulas	Finite Counting and Regular Size				Finite Counting and Mapped Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
Absence								
$\square(\neg a)$	1. $(\square(\neg a))$	11.58	1.58	10.00	1. $(\square(\neg a))$	20.22	10.22	10.00
	2. $(\neg(\diamond a))$	11.58	1.58	10.00	2. $(\neg(\diamond a))$	20.22	10.22	10.00
	3. $(\square(\neg(\diamond a)))$	12.00	2.00	10.00	3. $(\square(\neg(\diamond a)))$	20.35	10.35	10.00
	4. $((\neg a)\,\mathcal{W}\,0)$	12.00	2.00	10.00	4. $((\neg a)\,\mathcal{W}\,0)$	20.35	10.35	10.00
	5. $(\neg(1\,\mathcal{U}\,a))$	12.00	2.00	10.00	5. $(\neg(1\,\mathcal{U}\,a))$	20.35	10.35	10.00
$\square(a \to \square(\neg b))$	1. $(\square(a \to (\square(\neg b))))$	15.17	2.58	12.58	1. $(\square(a \to (\square(\neg b))))$	25.80	13.22	12.58
	2. $((\neg a)\,\mathcal{W}(\neg(\diamond b)))$	15.17	2.58	12.58	2. $((\neg a)\,\mathcal{W}(\neg(\diamond b)))$	25.80	13.22	12.58
	3. $((\neg a)\,\mathcal{W}(\square(b \leftrightarrow (b \wedge (\bigcirc 0)))))$	15.91	3.32	12.58	3. $((\neg a)\,\mathcal{W}(\square(b \leftrightarrow (b \wedge (\bigcirc 0)))))$	26.64	14.06	12.58
	4. $(\bigcirc((\neg a)\,\mathcal{W}(\neg(\diamond b))))$	16.27	2.81	13.46	4. $(\bigcirc((\neg a)\,\mathcal{W}(\neg(\diamond b))))$	26.85	13.39	13.46
	5. $(\square(\bigcirc(\neg(a \wedge (a\,\mathcal{W}\,b)))))$	17.72	3.00	14.72	5. $(\square(\bigcirc(\neg(a \wedge (a\,\mathcal{W}\,b)))))$	28.30	13.58	14.72
$\diamond a \to (\neg b\,\mathcal{U}\,a)$	1. 1	20.00	0.00	20.00	1. $((\diamond a) \to ((\neg b)\,\mathcal{U}\,a))$	39.06	19.64	19.42
	2. $(\diamond a)$	21.00	1.00	20.00	2. $((\neg b)\,\mathcal{W}(a \leftrightarrow (\diamond a)))$	39.06	19.64	19.42
	3. $(\diamond b)$	21.00	1.00	20.00	3. $((\diamond a) \to ((\neg b)\,\mathcal{W}\,a))$	39.06	19.64	19.42
	4. $(\diamond(\square b))$	21.58	1.58	20.00	4. $((b \to (\bigcirc 0))\,\mathcal{U}(a \leftrightarrow (\diamond a)))$	39.14	19.72	19.42
	5. $(\diamond(\neg a))$	21.58	1.58	20.00	5. $((b \to a) \leftrightarrow (\diamond a))$	39.19	19.60	19.59
Universality								
$\square a$	1. $(\square a)$	11.00	1.00	10.00	1. $(\square a)$	20.11	10.11	10.00
	2. $(a\,\mathcal{W}\,0)$	11.58	1.58	10.00	2. $(a\,\mathcal{W}\,0)$	20.22	10.22	10.00
	3. $(\square(a\,\mathcal{W}\,0))$	12.00	2.00	10.00	3. $(\square(a\,\mathcal{W}\,0))$	20.35	10.35	10.00
	4. $(a\,\mathcal{W}(\square a))$	12.00	2.00	10.00	4. $(a\,\mathcal{W}(\square a))$	20.35	10.35	10.00
	5. $(\neg(\diamond(\neg a)))$	12.00	2.00	10.00	5. $(\neg(\diamond(\neg a)))$	20.35	10.35	10.00
$\square(a \to \square b)$	1. $((\neg a)\,\mathcal{W}(\square b))$	14.91	2.32	12.58	1. $((\neg a)\,\mathcal{W}(\square b))$	25.65	13.07	12.58
	2. $(\square(a \to (\square b)))$	14.91	2.32	12.58	2. $(\square(a \to (\square b)))$	25.65	13.07	12.58
	3. $((\neg a)\,\mathcal{U}(\neg(\diamond(\neg b))))$	15.39	2.81	12.58	3. $((\neg a)\,\mathcal{U}(\neg(\diamond(\neg b))))$	25.97	13.39	12.58
	4. $((a \leftrightarrow (\bigcirc 0))\,\mathcal{W}(\square b))$	15.39	2.81	12.58	4. $((a \leftrightarrow (\bigcirc 0))\,\mathcal{W}(\square b))$	25.97	13.39	12.58
	5. $(\square(a \leftrightarrow (a \wedge (\square b))))$	15.39	2.81	12.58	5. $(\square(a \leftrightarrow (a \wedge (\square b))))$	25.97	13.39	12.58
$\diamond a \to (b\,\mathcal{U}\,a)$	1. 1	20.00	0.00	20.00	1. $((\diamond a) \to (b\,\mathcal{U}\,a))$	39.02	19.60	19.42
	2. $(\diamond a)$	21.00	1.00	20.00	2. $(((\diamond a) \to ((\neg a) \leftrightarrow b))\,\mathcal{W}\,a)$	39.14	19.72	19.42
	3. $(\diamond b)$	21.00	1.00	20.00	3. $((\bigcirc(\neg a))\,\mathcal{U}(b\,\mathcal{U}\,a))$	39.22	19.64	19.59
	4. $(\diamond(\square b))$	21.58	1.58	20.00	4. $((a \leftrightarrow b) \to (b \leftrightarrow (1\,\mathcal{U}\,a)))$	39.31	19.72	19.59
	5. $(\diamond(\neg a))$	21.58	1.58	20.00	5. $(((b \to a)\,\mathcal{U}\,a) \to a)$	39.38	19.64	19.74

Table A.5: Results for the existential semantics, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al.[28]

Formulas	simpleRegex				regexScoreRegulateSize			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
Existence								
$\Diamond a$	1. 1	20.00	0.00	20.00	1. $((1 \mathcal{U} b) \mathcal{W} (b \rightarrow (\bigcirc a)))$	40.00	20.00	20.00
	2. $(\Diamond a)$	21.00	1.00	20.00	2. $(\Box(1 \mathcal{U}((\bigcirc 0) \rightarrow (\Diamond(\Box(b \mathcal{W}(a \leftrightarrow b)))))))$	40.00	20.00	20.00
	3. $(\Diamond b)$	21.00	1.00	20.00	3. $(\Diamond(\neg(1 \mathcal{U}(\bigcirc(\bigcirc a)))))$	40.00	20.00	20.00
	4. $(\Diamond(\bigcirc a))$	21.58	1.58	20.00	4. $(\Diamond(\Box(\bigcirc(\Box((\Diamond(\bigcirc 0)) \rightarrow b)))))$	40.00	20.00	20.00
	5. $(\neg(\bigcirc 0))$	21.58	1.58	20.00	5. $(\bigcirc(\Diamond(\Box(\bigcirc(\Box(\bigcirc(\neg(1 \mathcal{U}(\Box b)))))))))$	40.00	20.00	20.00
$\Box(\neg a) \vee \Diamond(a \wedge (\Diamond b))$	1. 1	20.00	0.00	20.00	1. $((1 \mathcal{U} b) \mathcal{W} (b \rightarrow (\bigcirc a)))$	40.00	20.00	20.00
	2. $(\Diamond a)$	21.00	1.00	20.00	2. $(\Box(1 \mathcal{U}((\bigcirc 0) \rightarrow (\Diamond(\Box(b \mathcal{W}(a \leftrightarrow b)))))))$	40.00	20.00	20.00
	3. $(\Diamond b)$	21.00	1.00	20.00	3. $(\Diamond(\neg(1 \mathcal{U}(\bigcirc(\bigcirc a)))))$	40.00	20.00	20.00
	4. $(\Diamond(\bigcirc a))$	21.58	1.58	20.00	4. $(\Diamond(\Box(\bigcirc(\Box((\Diamond(\bigcirc 0)) \rightarrow b)))))$	40.00	20.00	20.00
	5. $(\bigcirc(\Diamond b))$	21.58	1.58	20.00	5. $(\bigcirc(\Diamond(\Box(\Diamond a))))$	40.00	20.00	20.00
$(\Box(a \wedge ((\neg b) \rightarrow ((\neg b) \mathcal{U}(\neg b \wedge c)))))$	1. $(\Box a)$	21.00	1.00	20.00	1. $(\Box a)$	38.09	18.09	20.00
	2. $(a \mathcal{W} 0)$	21.58	1.58	20.00	2. $(a \mathcal{W} 0)$	38.21	18.21	20.00
	3. $(\Box(a \wedge ((\neg b) \rightarrow ((\neg b) \mathcal{U}(c \wedge (\neg b))))))$	21.69	3.70	17.99	3. $(\Box(a \mathcal{W} 0))$	38.33	18.33	20.00
	4. $(\Box(a \mathcal{W} 0))$	22.00	2.00	20.00	4. $((\Box a) \mathcal{W} 0)$	38.33	18.33	20.00
	5. $((\Box a) \mathcal{W} 0)$	22.00	2.00	20.00	5. $(\neg(\Diamond(\neg a)))$	38.33	18.33	20.00
$(\Box((\neg b) \rightarrow ((\neg b) \mathcal{U}(c \wedge \neg b))))$	1. 1	30.00	0.00	30.00	1. $(\Box((\neg b) \rightarrow ((\neg b) \mathcal{U}(c \wedge (\neg b)))))$	57.09	29.10	27.99
	2. $(\Diamond b)$	31.00	1.00	30.00	2. 1	57.99	27.99	30.00
	3. $(\Diamond a)$	31.00	1.00	30.00	3. $(\Diamond b)$	58.07	28.07	30.00
	4. $(\Diamond c)$	31.00	1.00	30.00	4. $(\Diamond a)$	58.07	28.07	30.00
	5. $(\Box((\neg b) \rightarrow ((\neg b) \mathcal{U}(c \wedge (\neg b)))))$	31.45	3.46	27.99	5. $(\Diamond c)$	58.07	28.07	30.00

Table A.6: Results for the existential semantics, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al.[28]

Formulas	Finite Counting and Linear Size				Finite Density and Linear Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
$\Box(a \to (b \vee c \vee d))$	overflow	overflow	overflow	overflow	overflow	overflow	overflow	overflow
$\Box(a \to (b \vee c))$	1. $(\Box(a \to (b \vee c)))$	28.81	0.74	28.07	1. $(\Box(a \to (b \vee c)))$	1.18	0.74	0.44
	2. 1	30.10	0.10	30.00	2. $((a \to c)\,\mathcal{U}\,b)$	2.91	0.58	2.32
	3. $(\Diamond a)$	30.21	0.21	30.00	3. $((a \to c)\,\mathcal{W}\,b)$	2.91	0.58	2.32
	4. $(\Diamond c)$	30.21	0.21	30.00	4. $((a \to c)\,\mathcal{U}(b \leftrightarrow (\Diamond a)))$	3.42	1.10	2.32
	5. $(\Diamond b)$	30.21	0.21	30.00	5. $(((a \to c)\,\mathcal{W}\,b)\,\mathcal{U}(\bigcirc c))$	3.51	1.10	2.41
$\Box(\bigcirc \neg a \to (\neg b \leftrightarrow \bigcirc(\neg b)))$	1. $(\Box((\bigcirc(\neg a)) \to ((\neg b) \leftrightarrow (\bigcirc(\neg b)))))$	18.17	1.91	16.26	1. $(\Box(b \to (b\,\mathcal{W}\,a)))$	1.85	0.74	1.12
	2. $(\Box(b \to (b\,\mathcal{W}\,a)))$	19.84	0.74	19.11	2. $(\Box((\bigcirc(\neg a)) \to ((\neg b) \leftrightarrow (\bigcirc(\neg b)))))$	2.02	1.91	0.11
	3. $(\Box(b\,\mathcal{W}((\neg b)\,\mathcal{U}\,a)))$	20.01	0.91	19.11	3. $(\Box(b\,\mathcal{W}((\neg b)\,\mathcal{U}\,a)))$	2.02	0.91	1.12
	4. $(\Box((\neg b)\,\mathcal{W}(b\,\mathcal{U}\,a)))$	20.01	0.91	19.11	4. $(\Box((\neg b)\,\mathcal{W}(b\,\mathcal{U}\,a)))$	2.02	0.91	1.12
	5. 1	20.10	0.10	20.00	5. $((\bigcirc 0)\,\mathcal{W}(\Box((\neg b)\,\mathcal{W}(b\,\mathcal{U}\,a))))$	2.70	1.58	1.12
$\Box(\neg(a \wedge b))$	1. $(\Box(\neg(a \wedge b)))$	16.43	0.58	15.85	1. $(\Box(\neg(a \wedge b)))$	0.67	0.58	0.08
	2. $(\Box(a \to (\neg b)))$	16.43	0.58	15.85	2. $(\Box(a \to (\neg b)))$	0.67	0.58	0.08
	3. $(\Box(b \to (\neg a)))$	16.43	0.58	15.85	3. $(\Box(b \to (\neg a)))$	0.67	0.58	0.08
	4. $(\neg(\Diamond(a \wedge b)))$	16.43	0.58	15.85	4. $(\neg(\Diamond(a \wedge b)))$	0.67	0.58	0.08
	5. $((\neg(a \wedge b))\,\mathcal{W}\,0)$	16.59	0.74	15.85	5. $((\neg(a \wedge b))\,\mathcal{W}\,0)$	0.82	0.74	0.08
$\Box(a \to \neg(b \vee c))$	1. $(\Box(a \to (\neg(b \vee c))))$	24.13	0.91	23.22	1. $(\neg(\Diamond(a \wedge c)))$	0.67	0.58	0.08
	2. $(\neg(\Diamond(a \wedge c)))$	26.43	0.58	25.85	2. $(\Box(a \to (\neg b)))$	0.67	0.58	0.08
	3. $(\Box(a \to (\neg b)))$	26.43	0.58	25.85	3. $(\Box(\neg(a \wedge b)))$	0.67	0.58	0.08
	4. $(\Box(\neg(a \wedge b)))$	26.43	0.58	25.85	4. $(\Box(\neg(a \wedge c)))$	0.67	0.58	0.08
	5. $(\Box(\neg(a \wedge c)))$	26.43	0.58	25.85	5. $((b \to (\neg a))\,\mathcal{W}\,0)$	0.82	0.74	0.08
$\Box a$	1. $(\Box a)$	10.21	0.21	10.00	1. $(\Box a)$	0.21	0.21	0.00
	2. $(a\,\mathcal{W}\,0)$	10.32	0.32	10.00	2. $(a\,\mathcal{W}\,0)$	0.32	0.32	0.00
	3. $(\Box(a\,\mathcal{W}\,0))$	10.45	0.45	10.00	3. $(\Box(\bigcirc a))$	0.32	0.32	0.00
	4. $(a\,\mathcal{W}(\Box a))$	10.45	0.45	10.00	4. $(\bigcirc(\Box a))$	0.32	0.32	0.00
	5. $(\neg(\Diamond(\neg a)))$	10.45	0.45	10.00	5. $(\Box(a\,\mathcal{W}\,0))$	0.45	0.45	0.00

Table A.7: Results for the existential semantics, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al.[9]

Formulas	Finite Counting and Linear Size				Finite Density and Linear Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
$\Box(a \rightarrow \Diamond b)$	1. 1	20.10	0.10	20.00	1. $((1 \,\mathcal{U}\, b) \,\mathcal{W}(b \rightarrow (\bigcirc a)))$	∞	1.10	∞
	2. $(\Diamond b)$	20.21	0.21	20.00	2. $(\Box(1 \mathcal{U}((\bigcirc 0) \rightarrow (\Diamond(\Box(b \,\mathcal{W}(a \leftrightarrow b)))))))$	∞	2.91	∞
	3. $(\Diamond a)$	20.21	0.21	20.00	3. $(\Diamond(\neg(1 \,\mathcal{U}\, b)))$	∞	0.58	∞
	4. $(\neg(\bigcirc 0))$	20.32	0.32	20.00	4. $(\Diamond(\Box(\bigcirc(\Box a))))$	∞	0.58	∞
	5. $(\neg(\Box b))$	20.32	0.32	20.00	5. $(\bigcirc(\Diamond(\Box(\bigcirc(\Box(\bigcirc(\neg(1 \mathcal{U}(\Box b)))))))))$	∞	1.91	∞
$\Box((a \wedge b) \rightarrow \bigcirc\Diamond(\neg c))$	1. 1	30.10	0.10	30.00	1. $((1 \mathcal{U}(\neg(c \rightarrow (\neg(a \leftrightarrow c))))) \,\mathcal{W}(\Box a))$	∞	2.32	∞
	2. $(\Diamond b)$	30.21	0.21	30.00	2. $(\Box((\Diamond c) \mathcal{U}(\neg(\Box(\bigcirc b)))))$	∞	1.10	∞
	3. $(\Diamond a)$	30.21	0.21	30.00	3. $(\Diamond(\bigcirc(\neg c)))$	∞	0.45	∞
	4. $(\Diamond c)$	30.21	0.21	30.00	4. $(\Diamond(\Box((\neg b) \,\mathcal{W}(\Diamond((\Box c) \rightarrow (\Diamond a))))))$	∞	1.91	∞
	5. $(1 \,\mathcal{U}\, b)$	30.32	0.32	30.00	5. $(\bigcirc(1 \mathcal{U}(c \,\mathcal{W}(a \vee c))))$	∞	1.10	∞
$\Box(\Diamond a)$	1. 1	20.10	0.10	20.00	1. $((1 \,\mathcal{U}\, b) \,\mathcal{W}(b \rightarrow (\bigcirc a)))$	∞	1.10	∞
	2. $(\Diamond b)$	20.21	0.21	20.00	2. $(\Box(1 \mathcal{U}((\bigcirc 0) \rightarrow (\Diamond(\Box(b \,\mathcal{W}(a \leftrightarrow b)))))))$	∞	2.91	∞
	3. $(\Diamond a)$	20.21	0.21	20.00	3. $(\Diamond(\neg(1 \,\mathcal{U}\, b)))$	∞	0.58	∞
	4. $(\neg(\bigcirc 0))$	20.32	0.32	20.00	4. $(\Diamond(\Box(\bigcirc(\Box a))))$	∞	0.58	∞
	5. $(\neg(\Box b))$	20.32	0.32	20.00	5. $(\bigcirc(\Diamond(\Box(\bigcirc(\Box(\bigcirc(\neg(1 \mathcal{U}(\Box b)))))))))$	∞	1.91	∞
$\Box(\Diamond(\neg(a \vee b)))$	1. 1	20.10	0.10	20.00	1. $((1 \,\mathcal{U}\, b) \,\mathcal{W}(b \rightarrow (\bigcirc a)))$	∞	1.10	∞
	2. $(\Diamond a)$	20.21	0.21	20.00	2. $(\Box(1 \mathcal{U}((\bigcirc 0) \rightarrow (\Diamond(\Box(b \,\mathcal{W}(a \leftrightarrow b)))))))$	∞	2.91	∞
	3. $(\Diamond b)$	20.21	0.21	20.00	3. $(\Diamond(\neg(1 \mathcal{U}(\bigcirc(\bigcirc a)))))$	∞	0.91	∞
	4. $(\Diamond(\bigcirc a))$	20.32	0.32	20.00	4. $(\Diamond(\Box(\bigcirc(\Box((\Diamond(\bigcirc 0)) \rightarrow b)))))$	∞	1.32	∞
	5. $(\bigcirc(\Diamond b))$	20.32	0.32	20.00	5. $(\bigcirc(\Diamond(\Box(\Diamond a))))$	∞	0.58	∞

Table A.8: Results for the existential semantics, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al.[9]

Formulas	Finite Counting and Linear Size				Finite Density and Linear Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
Absence								
$\Box(\neg a)$	1. $(\Box(\neg a))$	10.32	0.32	10.00	1. $(\Box(\neg a))$	0.32	0.32	0.00
	2. $(\neg(\Diamond a))$	10.32	0.32	10.00	2. $(\neg(\Diamond a))$	0.32	0.32	0.00
	3. $(\Box(\neg(\Diamond a)))$	10.45	0.45	10.00	3. $(\Box(\neg(\Diamond a)))$	0.45	0.45	0.00
	4. $((\neg a)\,\mathcal{W}\,0)$	10.45	0.45	10.00	4. $((\neg a)\,\mathcal{W}\,0)$	0.45	0.45	0.00
	5. $(\neg(1\,\mathcal{U}\,a))$	10.45	0.45	10.00	5. $(\neg(1\,\mathcal{U}\,a))$	0.45	0.45	0.00
$\Box(a \to \Box(\neg b))$	1. $(\Box(a \to (\Box(\neg b))))$	13.32	0.74	12.58	1. $(\Box(a \to (\neg b)))$	0.67	0.58	0.08
	2. $((\neg a)\,\mathcal{W}(\neg(\Diamond b)))$	13.32	0.74	12.58	2. $(\Box(b \to (\neg a)))$	0.67	0.58	0.08
	3. $((\neg a)\,\mathcal{W}(\Box(b \leftrightarrow (b \wedge (\bigcirc 0)))))$	14.17	1.58	12.58	3. $(\neg(\Diamond(a \wedge b)))$	0.67	0.58	0.08
	4. $(\bigcirc((\neg a)\,\mathcal{W}(\neg(\Diamond b))))$	14.37	0.91	13.46	4. $(\Box(\neg(a \wedge b)))$	0.67	0.58	0.08
	5. $(\Box(\bigcirc(\neg(a \wedge (a\,\mathcal{W}\,b)))))$	15.82	1.10	14.72	5. $(\Box(a \to (\Box(\neg b))))$	0.75	0.74	0.01
$\Diamond a \to (\neg b\,\mathcal{U}\,a)$	1. 1	20.10	0.10	20.00	1. $((\Diamond a) \to ((\neg b)\,\mathcal{U}\,a))$	2.50	0.91	1.59
	2. $(\Diamond a)$	20.21	0.21	20.00	2. $((\neg b)\,\mathcal{W}(a \leftrightarrow (\Diamond a)))$	2.50	0.91	1.59
	3. $(\Diamond b)$	20.21	0.21	20.00	3. $((\Diamond a) \to ((\neg b)\,\mathcal{W}\,a))$	2.50	0.91	1.59
	4. $(\Diamond(\Box b))$	20.32	0.32	20.00	4. $((b \to a) \leftrightarrow (\Diamond a))$	2.74	0.74	2.00
	5. $(\Diamond(\neg a))$	20.32	0.32	20.00	5. $((b \to (\bigcirc 0))\,\mathcal{U}(a \leftrightarrow (\Diamond a)))$	2.91	1.32	1.59
Universality								
$\Box a$	1. $(\Box a)$	10.21	0.21	10.00	1. $(\Box a)$	0.21	0.21	0.00
	2. $(a\,\mathcal{W}\,0)$	10.32	0.32	10.00	2. $(a\,\mathcal{W}\,0)$	0.32	0.32	0.00
	3. $(\Box(a\,\mathcal{W}\,0))$	10.45	0.45	10.00	3. $(\Box(\bigcirc a))$	0.32	0.32	0.00
	4. $(a\,\mathcal{W}(\Box a))$	10.45	0.45	10.00	4. $(\bigcirc(\Box a))$	0.32	0.32	0.00
	5. $(\neg(\Diamond(\neg a)))$	10.45	0.45	10.00	5. $(\Box(a\,\mathcal{W}\,0))$	0.45	0.45	0.00
$\Box(a \to \Box b)$	1. $((\neg a)\,\mathcal{W}(\Box b))$	13.17	0.58	12.58	1. $(\Box(a \to b))$	0.53	0.45	0.08
	2. $(\Box(a \to (\Box b)))$	13.17	0.58	12.58	2. $((\neg a)\,\mathcal{W}(\Box b))$	0.59	0.58	0.01
	3. $((\neg a)\,\mathcal{U}(\neg(\Diamond(\neg b))))$	13.49	0.91	12.58	3. $(\Box(a \to (\Box b)))$	0.59	0.58	0.01
	4. $((a \leftrightarrow (\bigcirc 0))\,\mathcal{W}(\Box b))$	13.49	0.91	12.58	4. $(\Box((\neg a)\,\mathcal{U}\,b))$	0.67	0.58	0.08
	5. $(\Box(a \leftrightarrow (a \wedge (\Box b))))$	13.49	0.91	12.58	5. $(\Box((\neg a) \vee b))$	0.67	0.58	0.08
$\Diamond a \to (b\,\mathcal{U}\,a)$	1. 1	20.10	0.10	20.00	1. $((\Diamond a) \to (b\,\mathcal{U}\,a))$	2.33	0.74	1.59
	2. $((\Diamond a) \to (b\,\mathcal{U}\,a))$	20.15	0.74	19.42	2. $((\bigcirc(\neg a))\,\mathcal{U}(b\,\mathcal{U}\,a))$	2.91	0.91	2.00
	3. $(\Diamond a)$	20.21	0.21	20.00	3. $(((\Diamond a) \to ((\neg a) \leftrightarrow b))\,\mathcal{W}\,a)$	2.91	1.32	1.59
	4. $(\Diamond b)$	20.21	0.21	20.00	4. $((a \leftrightarrow b) \to (b \leftrightarrow (1\,\mathcal{U}\,a)))$	3.32	1.32	2.00
	5. $(\Diamond(\Box b))$	20.32	0.32	20.00	5. $(((b \to a)\,\mathcal{U}\,a) \to a)$	3.49	0.91	2.58

Table A.9: Results for the existential semantics, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al.[28]

Formulas	Finite Counting and Linear Size				Finite Density and Linear Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
Existence								
$\Diamond a$	1. 1	20.10	0.10	20.00	1. $((1 \mathcal{U} b) \mathcal{W} (b \rightarrow (\bigcirc a)))$	∞	1.10	∞
	2. $(\Diamond a)$	20.21	0.21	20.00	2. $(\Box(1 \mathcal{U}((\bigcirc 0) \rightarrow (\Diamond(\Box(b \mathcal{W}(a \leftrightarrow b)))))))$	∞	2.91	∞
	3. $(\Diamond b)$	20.21	0.21	20.00	3. $(\Diamond(\neg(1 \mathcal{U}(\bigcirc(\bigcirc a)))))$	∞	0.91	∞
	4. $(\Diamond(\bigcirc a))$	20.32	0.32	20.00	4. $(\Diamond(\Box(\bigcirc(\Box((\Diamond(\bigcirc 0)) \rightarrow b)))))$	∞	1.32	∞
	5. $(\neg(\bigcirc 0))$	20.32	0.32	20.00	5. $(\bigcirc(\Diamond(\Box(\bigcirc(\Box(\bigcirc(\neg(1 \mathcal{U}(\Box b)))))))))$	∞	1.91	∞
$\Box(\neg a) \vee \Diamond(a \wedge (\Diamond b))$	1. 1	20.10	0.10	20.00	1. $((1 \mathcal{U} b) \mathcal{W} (b \rightarrow (\bigcirc a)))$	∞	1.10	∞
	2. $(\Diamond a)$	20.21	0.21	20.00	2. $(\Box(1 \mathcal{U}((\bigcirc 0) \rightarrow (\Diamond(\Box(b \mathcal{W}(a \leftrightarrow b)))))))$	∞	2.91	∞
	3. $(\Diamond b)$	20.21	0.21	20.00	3. $(\Diamond(\neg(1 \mathcal{U}(\bigcirc(\bigcirc a)))))$	∞	0.91	∞
	4. $(\Diamond(\bigcirc a))$	20.32	0.32	20.00	4. $(\Diamond(\Box(\bigcirc(\Box((\Diamond(\bigcirc 0)) \rightarrow b)))))$	∞	1.32	∞
	5. $(\bigcirc(\Diamond b))$	20.32	0.32	20.00	5. $(\bigcirc(\Diamond(\Box(\Diamond a))))$	∞	0.58	∞
$(\Box(a \wedge ((\neg b) \rightarrow ((\neg b) \mathcal{U}(\neg b \wedge c)))))$	1. $(\Box a)$	20.21	0.21	20.00	1. $(\Box a)$	0.21	0.21	0.00
	2. $(a \mathcal{W} 0)$	20.32	0.32	20.00	2. $(a \mathcal{W} 0)$	0.32	0.32	0.00
	3. $(\Box(a \mathcal{W} 0))$	20.45	0.45	20.00	3. $(\Box(\bigcirc a))$	0.32	0.32	0.00
	4. $((\Box a) \mathcal{W} 0)$	20.45	0.45	20.00	4. $(\bigcirc(\Box a))$	0.32	0.32	0.00
	5. $(\neg(\Diamond(\neg a)))$	20.45	0.45	20.00	5. $(\Box(a \mathcal{W} 0))$	0.45	0.45	0.00
$(\Box((\neg b) \rightarrow ((\neg b) \mathcal{U}(c \wedge \neg b))))$	1. $(\Box((\neg b) \rightarrow ((\neg b) \mathcal{U}(c \wedge (\neg b)))))$	29.89	1.91	27.99	1. $(\Box((\neg b) \rightarrow ((\neg b) \mathcal{U}(c \wedge (\neg b)))))$	2.32	1.91	0.41
	2. 1	30.10	0.10	30.00	2. $(c \mathcal{U}((\bigcirc b) \rightarrow b))$	3.74	0.74	3.00
	3. $(\Diamond b)$	30.21	0.21	30.00	3. $((c \mathcal{W} b) \mathcal{W}(\bigcirc(\neg b)))$	3.91	0.91	3.00
	4. $(\Diamond a)$	30.21	0.21	30.00	4. $(a \mathcal{W}((c \mathcal{W} b) \mathcal{W}(\bigcirc(\neg b))))$	5.32	1.32	4.00
	5. $(\Diamond c)$	30.21	0.21	30.00	5. $((\bigcirc c) \mathcal{W}(((\bigcirc(\bigcirc b)) \mathcal{W} c) \rightarrow c))$	6.17	1.58	4.59

Table A.10: Results for the existential semantics, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al.[28]

A.3 Infinite: Density of Bound

Formulas	Density of Bound 5 with Linear Size				Density of Bound 10 with Linear Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
$\square(a \rightarrow (b \vee c \vee d))$	1. $(d\,\mathcal{W}(\bigcirc(c \wedge d)))$	1.42	0.74	0.68	overflow	overflow	overflow	overflow
	2. $(\square((c\,\mathcal{U}\,d)\,\mathcal{W}\,b))$	1.55	0.74	0.82				
	3. $(d\,\mathcal{W}(\bigcirc d))$	1.86	0.45	1.42				
	4. $(d\,\mathcal{U}(\bigcirc d))$	1.86	0.45	1.42				
	5. $(c\,\mathcal{U}\,d)$	1.90	0.32	1.58				
$\square(a \rightarrow (b \vee c))$	1. $(\square(a \rightarrow (b \vee c)))$	1.77	0.74	1.04	timeout	timeout	timeout	timeout
	2. $(\diamondsuit(c \wedge (\bigcirc(\bigcirc c))))$	2.33	0.74	1.59				
	3. $(\diamondsuit(b \wedge (\bigcirc c)))$	2.54	0.58	1.95				
	4. $((a \rightarrow c)\,\mathcal{U}\,b)$	2.86	0.58	2.28				
	5. $(b\,\mathcal{W}(\bigcirc((c\,\mathcal{W}\,0)\,\mathcal{U}(\bigcirc c))))$	2.89	1.32	1.57				
$\square(\bigcirc\neg a \rightarrow (\neg b \leftrightarrow \bigcirc(\neg b)))$	1. $(\square(\diamondsuit(a \wedge b)))$	1.71	0.58	1.13	1. $(\square(b \rightarrow (b\,\mathcal{W}\,a)))$	1.77	0.74	1.03
	2. $((\diamondsuit(a \wedge b))\,\mathcal{W}\,0)$	1.86	0.74	1.13	2. $(\square((\neg b)\,\mathcal{W}(b\,\mathcal{U}\,a)))$	1.88	0.91	0.97
	3. $(\square(\diamondsuit(\neg(a \rightarrow b))))$	1.86	0.74	1.13	3. $(\square(b\,\mathcal{W}((\neg b)\,\mathcal{U}\,a)))$	1.88	0.91	0.97
	4. $(\square(\diamondsuit(a \wedge (\neg b))))$	1.86	0.74	1.13	4. $(\square((\bigcirc(\neg a)) \rightarrow ((\neg b) \leftrightarrow (\bigcirc(\neg b)))))$	2.00	1.91	0.10
	5. $(\square(\diamondsuit(a\,\mathcal{U}(a \wedge b))))$	2.03	0.91	1.13	5. $(\square(\diamondsuit(a \wedge b)))$	2.41	0.58	1.82
$\square(\neg(a \wedge b))$	1. $(\square(a \rightarrow (\neg b)))$	0.98	0.58	0.39	1. $(\square(a \rightarrow (\neg b)))$	0.67	0.58	0.08
	2. $(\square(\neg(a \wedge b)))$	0.98	0.58	0.39	2. $(\square(\neg(a \wedge b)))$	0.67	0.58	0.08
	3. $(\neg(\diamondsuit(a \wedge b)))$	0.98	0.58	0.39	3. $(\neg(\diamondsuit(a \wedge b)))$	0.67	0.58	0.08
	4. $(\square(b \rightarrow (\neg a)))$	0.98	0.58	0.39	4. $(\square(b \rightarrow (\neg a)))$	0.67	0.58	0.08
	5. $(\square((\neg a)\,\mathcal{W}(\neg b)))$	1.13	0.74	0.39	5. $(\square((\neg a)\,\mathcal{W}(\neg b)))$	0.82	0.74	0.08
$\square(a \rightarrow \neg(b \vee c))$	1. $(\square(\neg(a \wedge c)))$	0.98	0.58	0.39	timeout	timeout	timeout	timeout
	2. $(\square(a \rightarrow (\neg b)))$	0.98	0.58	0.39				
	3. $(\square(\neg(a \wedge b)))$	0.98	0.58	0.39				
	4. $(\neg(\diamondsuit(a \wedge c)))$	0.98	0.58	0.39				
	5. $(\square(a \rightarrow (\neg(b \vee c))))$	1.05	0.91	0.14				
$\square a$	1. $(\square a)$	0.25	0.21	0.05	1. $(\square a)$	0.21	0.21	0.00
	2. $(a\,\mathcal{W}\,0)$	0.37	0.32	0.05	2. $(a\,\mathcal{W}\,0)$	0.32	0.32	0.00
	3. $(\square(\bigcirc a))$	0.41	0.32	0.08	3. $(\square(\bigcirc a))$	0.32	0.32	0.00
	4. $(\bigcirc(\square a))$	0.41	0.32	0.08	4. $(\bigcirc(\square a))$	0.32	0.32	0.00
	5. $(a\,\mathcal{W}(\bigcirc 0))$	0.49	0.45	0.05	5. $(a\,\mathcal{W}(\bigcirc 0))$	0.45	0.45	0.00

Table A.11: Results for Density of Bound 5 and 10, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al.[9], timeout after 30min

Formulas	Density of Bound 5 with Linear Size				Density of Bound 10 with Linear Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
$\Box(a \rightarrow \Diamond b)$	1. $(\Diamond((\neg(\bigcirc b)) \wedge (\bigcirc a)))$	2.64	0.91	1.73	1. $(a \leftrightarrow (\neg(b \mathcal{U}(\neg a))))$	3.49	0.91	2.58
	2. $(\Diamond(\neg(a \rightarrow b)))$	2.66	0.58	2.08	2. $((a \wedge b) \rightarrow (\bigcirc a))$	3.74	0.74	3.00
	3. $(\Diamond(a \wedge (\neg b)))$	2.66	0.58	2.08	3. $(\Box(\neg(\Box b)))$	3.77	0.45	3.32
	4. $(\bigcirc(\Diamond(\neg((\neg a) \mathcal{U} b))))$	2.80	0.91	1.89	4. $(\neg(\Diamond(\Box b)))$	3.77	0.45	3.32
	5. $(1 \mathcal{U}(a \wedge (\neg b)))$	2.81	0.74	2.08	5. $(\Box(\Diamond(\neg b)))$	3.77	0.45	3.32
$\Box((a \wedge b) \rightarrow \bigcirc\Diamond(\neg c))$	1. $(\Diamond(\bigcirc(a \wedge b)))$	2.32	0.58	1.73	timeout	timeout	timeout	timeout
	2. $(\Diamond((\bigcirc b) \wedge (\bigcirc a)))$	2.47	0.74	1.73				
	3. $(\Diamond(a \wedge b))$	2.52	0.45	2.08				
	4. $(\Box(\Diamond b))$	2.69	0.32	2.37				
	5. $((\Diamond a) \mathcal{U}(a \wedge b))$	2.81	0.74	2.08				
$\Box(\Diamond a)$	1. $(\Box(\Diamond a))$	2.69	0.32	2.37	1. $(\Box(\Diamond a))$	3.65	0.32	3.32
	2. $(\Box(\bigcirc(\Diamond a)))$	2.82	0.45	2.37	2. $(\Box(\bigcirc(\Diamond a)))$	3.77	0.45	3.32
	3. $(\Box(1 \mathcal{U} a))$	2.82	0.45	2.37	3. $(\Box(1 \mathcal{U} a))$	3.77	0.45	3.32
	4. $(\Diamond(\Box(\Diamond a)))$	2.82	0.45	2.37	4. $(\Diamond(\Box(\Diamond a)))$	3.77	0.45	3.32
	5. $(\bigcirc(\Box(\Diamond a)))$	2.82	0.45	2.37	5. $(\bigcirc(\Box(\Diamond a)))$	3.77	0.45	3.32
$\Box(\Diamond(\neg(a \vee b)))$	1. $(\Box(\Diamond(\neg(a \vee b))))$	1.86	0.74	1.13	1. $((\Diamond(\Box a)) \rightarrow (\neg b))$	5.06	0.74	4.32
	2. $(\neg(\Diamond(\Box(a \mathcal{U} b))))$	2.09	0.74	1.35	2. $((\Diamond(\Box(\bigcirc a))) \rightarrow (\neg a))$	5.23	0.91	4.32
	3. $(\Box(\neg(\Diamond(\Box(a \mathcal{U} b)))))$	2.26	0.91	1.35	3. $((\bigcirc(\Diamond(\bigcirc(\Box b)))) \rightarrow (\neg b))$	5.42	1.10	4.32
	4. $((\Diamond(\Box(a \vee b))) \rightarrow (\Box b))$	2.33	1.10	1.23	4. $(\Box(\Diamond(a \vee b)))$	5.49	0.58	4.91
	5. $(\Box(\Diamond(\neg((a \vee b) \mathcal{W}(\bigcirc 0)))))$	2.45	1.32	1.13	5. $(\Box(\Diamond(a \rightarrow b)))$	5.49	0.58	4.91

Table A.12: Results for Density of Bound 5 and 10, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al.[9], timeout after 30min

Formulas	Density of Bound 5 with Linear Size			
	Formula	Score	Simp	Spec
$\square(a \rightarrow (b \vee c \vee d))$	1. $(d\,\mathcal{W}(\bigcirc(c \wedge d)))$	1.42	0.74	0.68
	2. $(d\,\mathcal{U}(\bigcirc d))$	1.86	0.45	1.42
	3. $(d\,\mathcal{W}(\bigcirc d))$	1.86	0.45	1.42
	4. $(c\,\mathcal{U}\,d)$	1.90	0.32	1.58
	5. $(c\,\mathcal{W}\,d)$	1.91	0.32	1.59
$\square(\bigcirc \neg a \rightarrow (\neg b \leftrightarrow \bigcirc(\neg b)))$	1. $(\square((\bigcirc(\neg a)) \rightarrow ((\neg b) \leftrightarrow (\bigcirc(\neg b)))))$	2.37	1.91	0.46
	2. $(\square((\neg b)\,\mathcal{W}(b\,\mathcal{U}\,a)))$	2.51	0.91	1.60
	3. $(\square(b \rightarrow (b\,\mathcal{W}\,a)))$	2.60	0.74	1.86
	4. $(\diamondsuit(\neg(a \rightarrow b)))$	2.66	0.58	2.08
	5. $(\diamondsuit(a \wedge (\neg b)))$	2.66	0.58	2.08
$\square(a \rightarrow \diamondsuit\, b)$	1. $(\diamondsuit(a \wedge (\neg b)))$	2.66	0.58	2.08
	2. $(\diamondsuit(\neg(a \rightarrow b)))$	2.66	0.58	2.08
	3. $(1\,\mathcal{U}(a \wedge (\neg b)))$	2.81	0.74	2.08
	4. $(\diamondsuit(\neg((\neg a)\,\mathcal{U}\,b)))$	2.96	0.74	2.23
	5. $(b\mathcal{U}(\diamondsuit(\neg(a \rightarrow b))))$	2.98	0.91	2.08
$\square((a \wedge b) \rightarrow \bigcirc\diamondsuit(\neg c))$	1. $(\square(c\mathcal{U}(\diamondsuit(a \leftrightarrow b))))$	3.27	0.91	2.37
	2. $(\neg(\diamondsuit(\bigcirc(\square((\neg a) \leftrightarrow b)))))$	3.47	1.10	2.37
	3. $(\square(b \rightarrow (\neg(\square c))))$	3.48	0.74	2.75
	4. $(\square((\square c) \rightarrow (\neg b)))$	3.48	0.74	2.75
	5. $(\diamondsuit((\neg b)\,\mathcal{W}(a \wedge b)))$	3.60	0.91	2.69

Table A.13: Results for Density of Bound 5, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 100, Benchmark as used by Ehlers et al.[9]

Formulas	Density of Bound 5 with Linear Size			
	Formula	Score	Simp	Spec
$\square(a \rightarrow (b \vee c \vee d))$	1. $(d\,\mathcal{W}((\bigcirc c)\,\mathcal{W}\,b))$	2.92	0.74	2.18
	2. $(\square(a \rightarrow ((b \vee c) \vee d)))$	2.96	1.10	1.86
	3. $(\diamondsuit(b\,\mathcal{W}(c \wedge d)))$	3.12	0.74	2.39
	4. $(\square(c \vee (\diamondsuit d)))$	3.33	0.58	2.75
	5. $(\bigcirc(\bigcirc(\diamondsuit(\bigcirc d))))$	3.38	0.58	2.80
$\square(\bigcirc \neg a \rightarrow (\neg b \leftrightarrow \bigcirc(\neg b)))$	1. $(\square((\bigcirc(\neg a)) \rightarrow ((\neg b) \leftrightarrow (\bigcirc(\neg b)))))$	2.37	1.91	0.46
	2. $(\square(b \rightarrow (b\,\mathcal{W}\,a)))$	2.60	0.74	1.86
	3. $(b \vee (\bigcirc(b \rightarrow a)))$	3.74	0.74	3.00
	4. $(b \rightarrow (\bigcirc(a \vee b)))$	3.74	0.74	3.00
	5. $((b\,\mathcal{W}\,a)\,\mathcal{W}(\neg b))$	3.83	0.74	3.09
$\square(a \rightarrow \diamondsuit b)$	1. $(\square(a \rightarrow (\diamondsuit b)))$	3.33	0.58	2.75
	2. $(\square((\diamondsuit b)\,\mathcal{W}(\neg a)))$	3.48	0.74	2.75
	3. $((\diamondsuit b)\,\mathcal{W}(\neg(\diamondsuit a)))$	3.48	0.74	2.75
	4. $((\diamondsuit b)\,\mathcal{W}(\neg(\bigcirc(\diamondsuit a))))$	3.77	0.91	2.86
	5. $(\square((\neg(\bigcirc a))\,\mathcal{W}(\diamondsuit b)))$	3.77	0.91	2.86
$\square((a \wedge b) \rightarrow \bigcirc\diamondsuit(\neg c))$	1. $(\square((a \wedge b) \rightarrow (\bigcirc(\diamondsuit(\neg c)))))$	4.39	1.32	3.07
	2. $((\square(\bigcirc c)) \rightarrow c)$	5.91	0.58	5.32
	3. $((\square c) \leftrightarrow (\bigcirc(\square c)))$	6.06	0.74	5.32
	4. $((\square(\bigcirc c)) \leftrightarrow (c\,\mathcal{W}\,0))$	6.23	0.91	5.32
	5. $((\neg c) \rightarrow (\diamondsuit(1\,\mathcal{U}(\bigcirc(\neg c)))))$	6.64	1.32	5.32

Table A.14: Results for Density of Bound 5, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 1000, Benchmark as used by Ehlers et al.[9]

Formulas	Density of Bound 5 with Linear Size				Density of Bound 10 with Linear Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
Absence								
$\square(\neg a)$	1. $(\neg(\diamond a))$	0.37	0.32	0.05	1. $(\neg(\diamond a))$	0.32	0.32	0.00
	2. $(\square(\neg a))$	0.37	0.32	0.05	2. $(\square(\neg a))$	0.32	0.32	0.00
	3. $(\square(\neg(\diamond a)))$	0.49	0.45	0.05	3. $(\square(\neg(\diamond a)))$	0.45	0.45	0.00
	4. $((\neg a)\,\mathcal{W}\,0)$	0.49	0.45	0.05	4. $((\neg a)\,\mathcal{W}\,0)$	0.45	0.45	0.00
	5. $(\neg(1\,\mathcal{U}\,a))$	0.49	0.45	0.05	5. $(\neg(1\,\mathcal{U}\,a))$	0.45	0.45	0.00
$\square(a \rightarrow \square(\neg b))$	1. $((\neg a)\,\mathcal{W}(\neg(\diamond b)))$	0.87	0.74	0.13	1. $(\square(a \rightarrow (\neg b)))$	0.67	0.58	0.08
	2. $(\square(a \rightarrow (\square(\neg b))))$	0.87	0.74	0.13	2. $(\neg(\diamond(a \wedge b)))$	0.67	0.58	0.08
	3. $(\square(a \rightarrow (\neg b)))$	0.98	0.58	0.39	3. $(\square(b \rightarrow (\neg a)))$	0.67	0.58	0.08
	4. $(\neg(\diamond(a \wedge b)))$	0.98	0.58	0.39	4. $(\square(\neg(a \wedge b)))$	0.67	0.58	0.08
	5. $(\square(b \rightarrow (\neg a)))$	0.98	0.58	0.39	5. $((\neg a)\,\mathcal{W}(\neg(\diamond b)))$	0.74	0.74	0.01
$\diamond a \rightarrow (\neg b\,\mathcal{U}\,a)$	1. $((\diamond a) \rightarrow ((\neg b)\,\mathcal{U}\,a))$	2.63	0.91	1.72	1. $((\diamond a) \rightarrow ((\neg b)\,\mathcal{U}\,a))$	2.50	0.91	1.59
	2. $((\neg b)\,\mathcal{W}(a \leftrightarrow (\diamond a)))$	2.63	0.91	1.72	2. $((\neg b)\,\mathcal{W}(a \leftrightarrow (\diamond a)))$	2.50	0.91	1.59
	3. $((\diamond a) \rightarrow ((\neg b)\,\mathcal{W}\,a))$	2.63	0.91	1.72	3. $((\diamond a) \rightarrow ((\neg b)\,\mathcal{W}\,a))$	2.50	0.91	1.59
	4. $((b \rightarrow (\bigcirc 0))\,\mathcal{U}(a \leftrightarrow (\diamond a)))$	3.05	1.32	1.72	4. $((b \rightarrow (\bigcirc 0))\,\mathcal{U}(a \leftrightarrow (\diamond a)))$	2.91	1.32	1.59
	5. $(a \vee ((\neg(\diamond a))\,\mathcal{W}(\neg b)))$	3.19	1.10	2.09	5. $(a \vee ((\neg(\diamond a))\,\mathcal{W}(\neg b)))$	3.10	1.10	2.00
Universality								
$\square a$	1. $(\square a)$	0.25	0.21	0.05	1. $(\square a)$	0.21	0.21	0.00
	2. $(a\,\mathcal{W}\,0)$	0.37	0.32	0.05	2. $(a\,\mathcal{W}\,0)$	0.32	0.32	0.00
	3. $(\square(\bigcirc a))$	0.41	0.32	0.08	3. $(\square(\bigcirc a))$	0.32	0.32	0.00
	4. $(\bigcirc(\square a))$	0.41	0.32	0.08	4. $(\bigcirc(\square a))$	0.32	0.32	0.00
	5. $(a\,\mathcal{W}(\bigcirc 0))$	0.49	0.45	0.05	5. $(a\,\mathcal{W}(\bigcirc 0))$	0.45	0.45	0.00
$\square(a \rightarrow \square b)$	1. $(\square(a \rightarrow (\square b)))$	0.72	0.58	0.13	1. $(\square(a \rightarrow b))$	0.53	0.45	0.08
	2. $((\neg a)\,\mathcal{W}(\square b))$	0.72	0.58	0.13	2. $(\square(a \rightarrow (\square b)))$	0.59	0.58	0.01
	3. $(\square(a \rightarrow b))$	0.84	0.45	0.39	3. $((\neg a)\,\mathcal{W}(\square b))$	0.59	0.58	0.01
	4. $((\neg a)\,\mathcal{W}(\bigcirc(\square b)))$	0.92	0.74	0.18	4. $(\square((\neg a)\,\mathcal{U}\,b))$	0.66	0.58	0.08
	5. $(\square(a \rightarrow (\bigcirc(\square b))))$	0.92	0.74	0.18	5. $(\square((\neg a) \vee b))$	0.67	0.58	0.08
$\diamond a \rightarrow (b\,\mathcal{U}\,a)$	1. $((\diamond a) \rightarrow (b\,\mathcal{U}\,a))$	2.46	0.74	1.72	1. $((\diamond a) \rightarrow (b\,\mathcal{U}\,a))$	2.33	0.74	1.59
	2. $(((\diamond a) \rightarrow ((\neg a) \leftrightarrow b))\,\mathcal{W}\,a)$	3.05	1.32	1.72	2. $(((\diamond a) \rightarrow ((\neg a) \leftrightarrow b))\,\mathcal{W}\,a)$	2.91	1.32	1.59
	3. $((\square b) \vee (\square(\diamond(\neg a))))$	3.32	0.91	2.41	3. $((a \leftrightarrow b) \rightarrow (b \leftrightarrow (1\,\mathcal{U}\,a)))$	3.32	1.32	2.00
	4. $(\square((\square a) \rightarrow b))$	3.33	0.58	2.75	4. $(((b \rightarrow a)\,\mathcal{U}\,a) \rightarrow a)$	3.49	0.91	2.58
	5. $(\square(a \rightarrow (\diamond b)))$	3.33	0.58	2.75	5. $(b \vee (\bigcirc(a \rightarrow b)))$	3.74	0.74	3.00

Table A.15: Results for Density of Bound 5 and 10, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al.[28]

Formulas	Density of Bound 5 with Linear Size				Density of Bound 10 with Linear Size			
	Formula	Score	Simp	Spec	Formula	Score	Simp	Spec
Existence								
$\Diamond a$	1. $(\Diamond(b \wedge (\bigcirc b)))$	2.13	0.58	1.54	1. $(\Diamond(b \wedge (\bigcirc b)))$	3.61	0.58	3.03
	2. $(\bigcirc(\Diamond(a \wedge b)))$	2.32	0.58	1.73	2. $(\Box(\Diamond b))$	3.65	0.32	3.32
	3. $(\bigcirc(1\,\mathcal{U}(a \wedge b)))$	2.47	0.74	1.73	3. $((\Diamond b)\,\mathcal{W}\,0)$	3.77	0.45	3.32
	4. $(\Diamond(a \wedge b))$	2.52	0.45	2.08	4. $(\Box(\bigcirc(\Diamond b)))$	3.77	0.45	3.32
	5. $(\Diamond(a \wedge (\bigcirc b)))$	2.54	0.58	1.95	5. $(\bigcirc(\Box(\Diamond b)))$	3.77	0.45	3.32
$\Box(\neg a) \vee \Diamond(a \wedge (\Diamond b))$	1. $(\Box(a \rightarrow (\Diamond b)))$	3.33	0.58	2.75	1. $(\Box(a \rightarrow (\Diamond b)))$	4.27	0.58	3.69
	2. $(\bigcirc(\Diamond(\bigcirc(\bigcirc b))))$	3.38	0.58	2.80	2. $((\Diamond b)\,\mathcal{W}(\neg(\Diamond a)))$	4.42	0.74	3.69
	3. $(\bigcirc(\bigcirc(\bigcirc(\Diamond b))))$	3.38	0.58	2.80	3. $(\Box((\Diamond b)\,\mathcal{W}(\neg a)))$	4.42	0.74	3.69
	4. $(\Diamond(b \wedge (a \leftrightarrow (\bigcirc(\bigcirc a)))))$	3.39	1.10	2.29	4. $(\Box((\neg(\bigcirc a))\,\mathcal{W}(\Diamond b)))$	4.70	0.91	3.79
	5. $((\Diamond b)\,\mathcal{W}(\neg(\Diamond a)))$	3.48	0.74	2.75	5. $((\Diamond b)\,\mathcal{W}(\neg(\bigcirc(\Diamond a))))$	4.70	0.91	3.79
$(\Box(a \wedge ((\neg b) \rightarrow ((\neg b)\,\mathcal{U}(\neg b \wedge c)))))$	1. $(\Box a)$	0.25	0.21	0.05				
	2. $(a\,\mathcal{W}\,0)$	0.37	0.32	0.05				
	3. $(\Box(\bigcirc a))$	0.41	0.32	0.08	timeout	timeout	timeout	timeout
	4. $(\bigcirc(\Box a))$	0.41	0.32	0.08				
	5. $((\Box a)\,\mathcal{W}\,0)$	0.49	0.45	0.05				

Table A.16: Results for Density of Bound 5 and 10, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al.[28], timeout after 30 min

Formulas	Density of Bound 5 with Linear Size			
	Formula	Score	Simp	Spec
Existence				
$\Diamond a$	1. $((\Diamond b)\,\mathcal{U}\,a)$	4.59	0.45	4.15
	2. $((1\,\mathcal{U}\,b)\,\mathcal{U}\,a)$	4.73	0.58	4.15
	3. $(\Diamond a)$	5.21	0.21	5.00
	4. $(1\,\mathcal{U}\,a)$	5.32	0.32	5.00
	5. $(\neg(\Box(\neg a)))$	5.45	0.45	5.00
$\Box(\neg a) \vee \Diamond(a \wedge (\Diamond b))$	1. $((\Box(\neg a)) \vee (\Diamond(a \wedge (\Diamond b))))$	5.47	1.32	4.15
	2. $((\neg a)\,\mathcal{W}(\Diamond b))$	5.63	0.58	5.05
	3. $((\Diamond a) \rightarrow (\Diamond b))$	5.63	0.58	5.05
	4. $((\Diamond b) \vee (\Box(\neg a)))$	5.78	0.74	5.05
	5. $((1\,\mathcal{U}\,a) \rightarrow (\Diamond b))$	5.78	0.74	5.05
$(\Box(a \wedge ((\neg b) \rightarrow ((\neg b)\,\mathcal{U}(\neg b \wedge c)))))$	1. $(\Box a)$	0.25	0.21	0.05
	2. $(a\,\mathcal{W}\,0)$	0.37	0.32	0.05
	3. $(\bigcirc(\Box a))$	0.41	0.32	0.08
	4. $(\Box(\bigcirc a))$	0.41	0.32	0.08
	5. $(a \wedge (\Box a))$	0.49	0.45	0.05

Table A.17: Results for Density of Bound 5, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 1000, Benchmark as used by Roy et al.[28], timeout after 30min

A.4 Infinite: Asymptotic Density

Formulas	Asymp. Density with Linear Size			
	Formula	Score	Simp	Spec
$\square(a \to (b \vee c \vee d))$	1. $(\square((c\,\mathcal{U}\,d)\,\mathcal{W}\,b))$	0.74	0.74	0.00
	2. $(\square(a \to ((b \vee c) \vee d)))$	1.10	1.10	0.00
	3. $(\square(\bigcirc(\bigcirc(d\,\mathcal{W}((\bigcirc c)\,\mathcal{W}\,b)))))$	1.32	1.32	0.00
	4. $(d\,\mathcal{W}(\bigcirc(c \wedge d)))$	1.42	0.74	0.68
	5. $(c \vee (\square(d\,\mathcal{W}\,c)))$	1.74	0.74	1.00
$\square(a \to (b \vee c))$	1. $(\square(a \to (b \vee c)))$	0.74	0.74	0.00
	2. $((a \to c)\,\mathcal{U}\,b)$	2.91	0.58	2.32
	3. $((a \to c)\,\mathcal{W}\,b)$	2.91	0.58	2.32
	4. $(b\,\mathcal{W}(\bigcirc((c\,\mathcal{W}\,0)\,\mathcal{U}(\bigcirc c))))$	2.91	1.32	1.58
	5. $(b \vee (c\,\mathcal{W}(\bigcirc c)))$	3.15	0.74	2.42
$\square(\bigcirc\neg a \to (\neg b \leftrightarrow \bigcirc(\neg b)))$	1. $(\square(b \to (b\,\mathcal{W}\,a)))$	0.74	0.74	0.00
	2. $(\square((\neg b)\,\mathcal{W}(b\,\mathcal{U}\,a)))$	0.91	0.91	0.00
	3. $(\square(b\,\mathcal{W}((\neg b)\,\mathcal{U}\,a)))$	0.91	0.91	0.00
	4. $((\bigcirc 0)\,\mathcal{W}(\square((\neg b)\,\mathcal{W}(b\,\mathcal{U}\,a))))$	1.58	1.58	0.00
	5. $(\square((\bigcirc(\neg a)) \to ((\neg b) \leftrightarrow (\bigcirc(\neg b)))))$	1.91	1.91	0.00
$\square(\neg(a \wedge b))$	1. $(\square(a \to (\neg b)))$	0.58	0.58	0.00
	2. $(\square(\neg(a \wedge b)))$	0.58	0.58	0.00
	3. $(\neg(\Diamond(a \wedge b)))$	0.58	0.58	0.00
	4. $(\square(b \to (\neg a)))$	0.58	0.58	0.00
	5. $(\Diamond(\square(b \to (\neg a))))$	0.74	0.74	0.00
$\square(a \to \neg(b \vee c))$	1. $(\square(\neg(a \wedge c)))$	0.58	0.58	0.00
	2. $(\square(a \to (\neg b)))$	0.58	0.58	0.00
	3. $(\square(\neg(a \wedge b)))$	0.58	0.58	0.00
	4. $(\neg(\Diamond(a \wedge c)))$	0.58	0.58	0.00
	5. $(\square(\bigcirc(\neg(a \wedge b))))$	0.74	0.74	0.00
$\square a$	1. $(\square a)$	0.21	0.21	0.00
	2. $(\square(\bigcirc a))$	0.32	0.32	0.00
	3. $(\bigcirc(\square a))$	0.32	0.32	0.00
	4. $(a\,\mathcal{W}\,0)$	0.32	0.32	0.00
	5. $(\Diamond(\square a))$	0.32	0.32	0.00

Table A.18: Results for Asymptotic Density, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al.[9]

Formulas	Asymp. Density with Linear Size			
	Formula	Score	Simp	Spec
$\Box(a \to \Diamond b)$	1. $(a \leftrightarrow (\neg(b \mathcal{U}(\neg a))))$	3.49	0.91	2.58
	2. $((a \wedge b) \to (\bigcirc a))$	3.74	0.74	3.00
	3. $((a \wedge b) \to (b \wedge (\bigcirc a)))$	4.10	1.10	3.00
	4. $(((\bigcirc(\Box a)) \mathcal{U}(b \to (\neg a))) \mathcal{W}(\bigcirc a))$	4.91	1.91	3.00
	5. $((\neg(a \wedge b)) \vee ((\neg(\bigcirc a)) \to (\Box a)))$	4.91	1.91	3.00
$\Box((a \wedge b) \to \bigcirc\Diamond(\neg c))$	1. $((\bigcirc a) \mathcal{W}(c \mathcal{U} a))$	3.00	0.74	2.26
	2. $((\bigcirc a) \mathcal{U}(c \mathcal{U} a))$	3.00	0.74	2.26
	3. $(((\neg a) \mathcal{U}(a \leftrightarrow b)) \mathcal{W} c)$	3.20	1.10	2.10
	4. $(c \mathcal{W}((\bigcirc a) \mathcal{W} a))$	3.32	0.74	2.58
	5. $((\bigcirc c) \mathcal{W}(b \to a))$	3.54	0.74	2.81
$\Box(\Diamond a)$	1. $((b \to a) \vee (\bigcirc(\neg a)))$	3.91	0.91	3.00
	2. $((\neg b) \vee (\bigcirc(a \to (\bigcirc b))))$	4.10	1.10	3.00
	3. $((\Box(\bigcirc(\bigcirc((\neg a) \mathcal{U}(\Diamond(\Box b)))))) \mathcal{W}(\Diamond a))$	∞	2.32	∞
	4. $(\bigcirc(\Diamond(\Box((\Diamond(\neg a)) \mathcal{W}(\Box a)))))$	∞	1.32	∞
	5. $((\Diamond a) \vee (\neg(\Box b)))$	∞	0.74	∞
$\Box(\Diamond(\neg(a \vee b)))$	1. $(\bigcirc(\bigcirc((\bigcirc b) \vee (\neg(b \wedge (\bigcirc a))))))$	4.58	1.58	3.00
	2. $((\bigcirc(a \leftrightarrow b)) \to (\bigcirc(a \mathcal{U}(\neg b))))$	5.17	1.58	3.58
	3. $(a \vee ((\Diamond(\Box(\neg b))) \mathcal{W}((\neg a) \mathcal{W}(\neg b))))$	5.91	2.32	3.58
	4. $((\Diamond(\neg(\Box b))) \mathcal{W}((1 \mathcal{U} b) \mathcal{W}(\neg(a \wedge b))))$	∞	2.91	∞
	5. $(\bigcirc(\neg(b \leftrightarrow (\neg b))))$	∞	0.74	∞

Table A.19: Results for Asymptotic Density, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Ehlers et al.[9]

Formulas	Asymp. Density with Linear Size			
	Formula	Score	Simp	Spec
Absence				
$\Box(\neg a)$	1. $(\neg(\Diamond a))$	0.32	0.32	0.00
	2. $(\Box(\neg a))$	0.32	0.32	0.00
	3. $(\Box(\neg(\bigcirc a)))$	0.45	0.45	0.00
	4. $(\Diamond(\Box(\neg a)))$	0.45	0.45	0.00
	5. $(\Box(\neg(\Diamond a)))$	0.45	0.45	0.00
$\Box(a \rightarrow \Box(\neg b))$	1. $(\Box(a \rightarrow (\neg b)))$	0.58	0.58	0.00
	2. $(\neg(\Diamond(a \wedge b)))$	0.58	0.58	0.00
	3. $(\Box(b \rightarrow (\neg a)))$	0.58	0.58	0.00
	4. $(\Box(\neg(a \wedge b)))$	0.58	0.58	0.00
	5. $((\neg a)\,\mathcal{W}(\neg(\Diamond b)))$	0.74	0.74	0.00
$\Diamond a \rightarrow (\neg b\,\mathcal{U}\,a)$	1. $((\Diamond a) \rightarrow ((\neg b)\,\mathcal{U}\,a))$	2.49	0.91	1.58
	2. $((\neg b)\,\mathcal{W}(a \leftrightarrow (\Diamond a)))$	2.49	0.91	1.58
	3. $((\Diamond a) \rightarrow ((\neg b)\,\mathcal{W}\,a))$	2.49	0.91	1.58
	4. $((b \rightarrow (\bigcirc 0))\,\mathcal{U}(a \leftrightarrow (\Diamond a)))$	2.91	1.32	1.58
	5. $(a \vee ((\neg(\Diamond a))\,\mathcal{W}(\neg b)))$	3.10	1.10	2.00
Universality				
$\Box a$	1. $(\Box a)$	0.21	0.21	0.00
	2. $(\Box(\bigcirc a))$	0.32	0.32	0.00
	3. $(\bigcirc(\Box a))$	0.32	0.32	0.00
	4. $(a\,\mathcal{W}\,0)$	0.32	0.32	0.00
	5. $(\Diamond(\Box a))$	0.32	0.32	0.00
$\Box(a \rightarrow \Box b)$	1. $(\Box(a \rightarrow b))$	0.45	0.45	0.00
	2. $(\Box((\neg a)\,\mathcal{U}\,b))$	0.58	0.58	0.00
	3. $(\bigcirc(\Box(a \rightarrow b)))$	0.58	0.58	0.00
	4. $(\Box(a \rightarrow (\Box b)))$	0.58	0.58	0.00
	5. $(\Box((\neg a) \vee b))$	0.58	0.58	0.00
$\Diamond a \rightarrow (b\,\mathcal{U}\,a)$	1. $((\Diamond a) \rightarrow (b\,\mathcal{U}\,a))$	2.32	0.74	1.58
	2. $(((\Diamond a) \rightarrow ((\neg a) \leftrightarrow b))\,\mathcal{W}\,a)$	2.91	1.32	1.58
	3. $((a \leftrightarrow b) \rightarrow (b \leftrightarrow (1\,\mathcal{U}\,a)))$	3.32	1.32	2.00
	4. $(((b \rightarrow a)\,\mathcal{U}\,a) \rightarrow a)$	3.49	0.91	2.58
	5. $((a \vee b) \vee (\neg(\Box((\Diamond a)\,\mathcal{W}\,0))))$	3.58	1.58	2.00

Table A.20: Results for Asymp. Density, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al.[28]

Formulas	Asymp. Density with Linear Size			
	Formula	Score	Simp	Spec
Existence				
$\Diamond a$	1. $(a \rightarrow (a \,\mathcal{U}\, b))$	4.17	0.58	3.58
	2. $((\bigcirc(a \vee b)) \,\mathcal{W}(a \rightarrow b))$	4.68	1.10	3.58
	3. $(a \rightarrow ((a \,\mathcal{U}\, b) \leftrightarrow (a \vee (\bigcirc b))))$	5.17	1.58	3.58
	4. $((\Box(1 \,\mathcal{U}(\Diamond b))) \,\mathcal{W}\, 0)$	∞	0.91	∞
	5. $(\Diamond(a \rightarrow (\Diamond a)))$	∞	0.58	∞
$\Box(\neg a) \vee \Diamond(a \wedge (\Diamond b))$	1. $(((\bigcirc a) \vee (\neg((\neg a) \rightarrow (\Diamond a)))) \,\mathcal{W}(a \vee b))$	5.91	2.91	3.00
	2. $((\Diamond(\bigcirc(\Box b))) \,\mathcal{W}(\Diamond(\bigcirc b)))$	∞	1.10	∞
	3. $((\Box(\bigcirc 0)) \rightarrow (a \vee (\neg b)))$	∞	1.10	∞
	4. $(((\Box a) \rightarrow b) \,\mathcal{U}(a \leftrightarrow (\Diamond a)))$	∞	1.32	∞
	5. $(((\neg a) \vee ((\Box b) \rightarrow a)) \,\mathcal{W}\, a)$	∞	1.32	∞
$(\Box(a \wedge ((\neg b) \rightarrow ((\neg b) \,\mathcal{U}(\neg b \wedge c)))))$	1. $(\Box a)$	0.21	0.21	0.00
	2. $(\Box(\bigcirc a))$	0.32	0.32	0.00
	3. $(\Diamond(\Box a))$	0.32	0.32	0.00
	4. $(\bigcirc(\Box a))$	0.32	0.32	0.00
	5. $(a \,\mathcal{W}\, 0)$	0.32	0.32	0.00

Table A.21: Results for Asymp. Density, **Parameters:** NumberOfSamples = 5000, Length = 10, NumberOfTraces = 50, Benchmark as used by Roy et al.[28]