The Complexity of Counting Problems for Linear Temporal Logic

Masterarbeit

Fabian Müller
Matrikel-Nr. 2672340

15. März 2016

Erstprüfer: Heribert Vollmer
Zweitprüfer: Arne Meier

Institut für Theoretische Informatik
Leibniz Universität Hannover

Contents

1 Introduction

Linear Temporal Logic (LTL) was first introduced by Pnueli [Pnu77]. It was used for reasoning about the behavior and the properties of parallel programs and concurrent systems.

Sistla and Clarke showed that satisfiability and model checking are #PSPACE-complete for full LTL (allowing all temporal operators and Boolean functions), and drop to NP-complete when restricting the set of temporal operators [SC85]. Bauland et al. generalized that result by showing the complexity of satisfiability for LTL-formulae with different subsets of temporal operators and Boolean functions [BSS+08]. The results range from polynomial time solvable if only monotone Boolean functions ($\vee$,$\wedge$) are allowed, over NP-complete only using $x \wedge \neg y$ and the temporal operators F and X, to #PSPACE-complete if allowing any additional temporal operator.

While in classical complexity theory, classes of decision problems are considered, counting complexity theory is about classes of functions. But these fields are closely connected: Most counting classes consist of problems that are defined by the number of accepting paths of a Turing machine.

This thesis is about the counting of models for Linear Temporal Logic formulae. We need to clarify about which kind of models we are going to talk. An LTL-formula has either zero or infinitely many models, so we will consider bounded models. In particular we will consider so-called k-word-models. Informally, these are models consisting of k worlds with edges between world i and $i+1$ (for $1 \leq i < k$) and an back edge from world k to one previous world. The problem we will analyze asks how many k-word-models for a specific LTL-formula exist. We will show the complexity of that problem for different sets of Boolean functions and temporal operators. Torfah and Zimmermann showed that this problem is #PSPACE-complete if k is coded binary and #P-complete if k is coded unary [TZ14]. In Section 3.1 we will have a look at the proof of the #PSPACE result and give a much simpler proof for the #P one. With the latter we also get a large number of additional results for different restricted versions of this problem. Moreover, we show #P-completeness for two restricted versions of the problem with unary coded k that are not covered by the previous mentioned result. Afterwards we have a look at the tractable cases. We will show for some additional fragments that they can be computed in polynomial time and therefore belong to the class FP.

2 Preliminaries

Before we can start with the results we need a short introduction to some basics. We want to follow the results of Torfah and Zimmermann about counting k-word-models for LTL-formulae. Therefore, we need to define Linear Temporal Logic and k-word-models. Furthermore, we need a short introduction to counting complexity to state the results. After that, we want to show results for fragments of this problem, that is, with restricted sets of temporal operators and Boolean functions. For the latter it is expedient to become familiar with Boolean clones and Post's lattice.

2.1 Boolean Functions

We start with a short introduction to Boolean functions. We later want to talk about LTL-formulae with different sets of Boolean functions and temporal operators. The easiest way to handle these sets of Boolean functions is to use Post's lattice. We start by giving the basic definitions.

A *Boolean function* f is a mapping from $\{0,1\}^n$ to $\{0,1\}$, for $n \in \mathbb{N}$.

Definition 2.1. A set B of Boolean functions is closed under *arbitrary composition* if $\langle B\rangle = B$, where: $f \in \langle B\rangle$ iff $f \in B$ or there are $g \in \langle B\rangle$ and $\{\chi_1, \dots, \chi_n\}$ which are variables or functions from $\langle B\rangle$, such that $f \equiv g(\{\chi_1, \dots, \chi_n\})$ holds.

That means if we have a set of Boolean functions B, then $\langle B\rangle$ contains all Boolean functions we can build using functions from B.

A *projection function* is a function that maps tuples of n elements to one of its elements: $proj_{i,n}(x_1, \dots, x_n) = x_i$, for $1 \leq i \leq n$.

Definition 2.2. A set of Boolean functions B is a *clone* if it contains all projection functions and is closed under arbitrary composition. The smallest clone that contains B is denoted with $[B]$.

We say B' is a *base* of B if $[B'] = B$. A list of the different clones with their standard bases is given in Figure 2. Figure 1 shows inclusions between these clones graphically, this graph is also called Post's lattice.

The clones are named after the properties of the Boolean functions one can build using the functions from the base of that clone. BF stands for Boolean function and contains all Boolean functions, because one can build all Boolean functions using only $\{\land, \neg\}$. R_1-formulae are 1-reproducing, that means $f(1, \dots, 1) = 1$. Analogously R_2-formulae are 0-reproducing and R_0-formulae are both 1-reproducing and 0-reproducing. The M-clones are the monotone clones, for all functions f in these clones and for all tuples $(x_1, \dots, x_n)$, $(y_1, \dots, y_n)$ it holds that:

$$x_1 \leq y_1, \dots, x_n \leq y_n \implies f(x_1, \dots, x_n) \leq f(y_1, \dots, y_n).$$

For example if $f(0,1) = 1$ holds it is not possible that $f(1,1) = 0$ holds as well. The different indices represent the constants 0 and 1 — M contains both, M_1 contains only

1, M_0 only 0 and M_2 none constant — this holds for other clones as well. S-formulae are c-separating ($c \in \{0,1\}$), it holds that if $f(x_1,\dots,x_n) = c$ it follows that $a_i = c$ for $i \in \{1,\dots,n\}$. The E and V-clones contain the Boolean function $\wedge$ respectively $\vee$ and are named after "et" and "vel" the Latin words for "and" and "or". D-formulae are self-dual, that is $f(x_1,\dots,x_n) = \neg f(\neg x_1,\dots,\neg x_n)$. The L stands for linear, that is, all L-formulae can be written in the following way: $f(x_1,\dots,x_n) \equiv c_0 \oplus c_1 \wedge x_1 \oplus \cdots \oplus c_n \wedge x_n$ for $c_i \in \{0,1\}$. The N-clones contain negation and the I-clones the identity function ($f(x) = x$).

2.2 Linear Temporal Logic

In this Section we want to define Linear Temporal Logic. We define structures and when such structures satisfy a LTL-formula. Furthermore, we define k-word-models as a special case of these structures.

Definition 2.3. Let B be a set of Boolean functions and $T \subseteq \{X, G, F, U, W, S\}$. Then the temporal logic LTL(B,T) is defined via the following grammar:

$$\varphi ::= \psi \mid f(\varphi,\dots,\varphi) \mid g(\varphi,\dots,\varphi)$$

where ψ is a propositional formula, $f \in [B]$ and $g \in T$.

We later want to count the satisfying models for a given LTL-formula but first we have to specify what satisfying means in this context.

Definition 2.4. A *structure* $S = (s, P, \sigma)$ consists of an infinite sequence $s = (s_i)_{i\in\mathbb{N}}$ of distinct states, a set of propositions P and a function $\sigma\colon \{s_i \mid i \in \mathbb{N}\} \to 2^P$ which induces an assignment $P \to \{0,1\}$ for each state.

We define what it means, that a structure S satisfies a LTL-formula inductively as follows:

$S, s_i \models x$	iff	$x \in \sigma(s_i)$
$S, s_i \models \varphi_1 \wedge \varphi_2$	iff	$S, s_i \models \varphi_1$ and $S, s_i \models \varphi_2$
$S, s_i \models \neg\varphi_1$	iff	$S, s_i \not\models \varphi_1$
$S, s_i \models X\varphi_1$	iff	$S, s_{i+1} \models \varphi_1$
$S, s_i \models \varphi_1 U \varphi_2$	iff	there is a $k \geq i$ such that $S, s_k \models \varphi_2$ and for every $i \leq j < k, S, s_j \models \varphi_1$
$S, s_i \models \varphi_1 S \varphi_2$	iff	there is a $k \leq i$ such that $S, s_k \models \varphi_2$ and for every $k < j \leq i, S, s_j \models \varphi_1$

It holds for LTL-forumlae φ, ψ that $F\varphi = \top U \varphi$, $G\varphi = \neg F \neg \varphi$ and $\varphi W \psi = \varphi U \psi \vee G\varphi$.

As mentioned in the introduction we want to talk about bounded models, in particular k-word-models. Those models are a special case of the structures we defined above.

Definition 2.5. A *k-word-model* of LTL-formula φ is a pair (u, v) of finite words over 2^P such that: $|uv| = k$ and $(\{s_1,\dots,s_k\}, P, \sigma), s_1 \models \varphi$, where σ assigns the states $s_1,\dots,s_j$ the symbols of the words of the prefix $u_1,\dots,u_j$ and the every state s_{j+l} the symbols of the word $v_{l \bmod k-j+1}$, for $j \geq 1$.

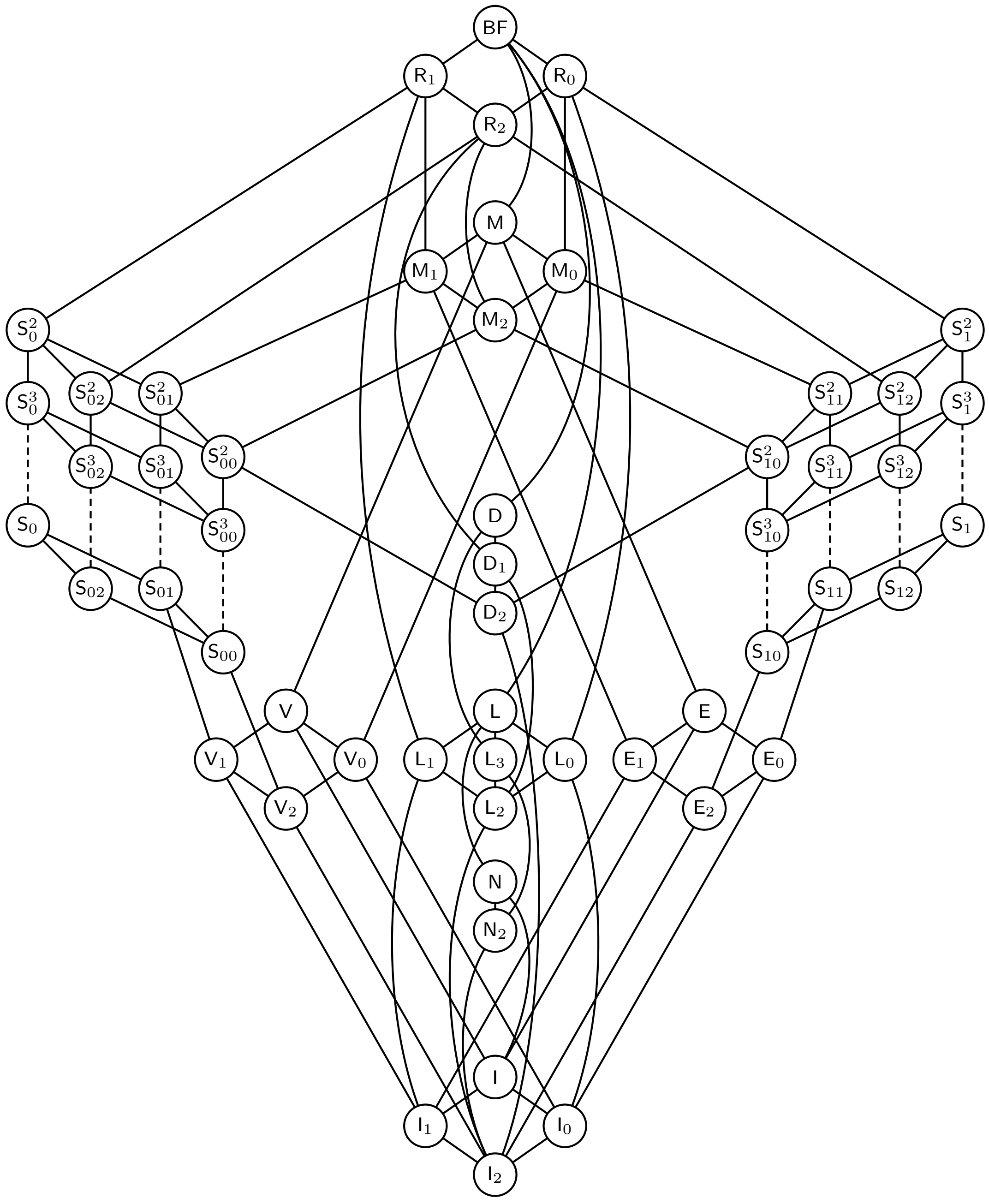

Figure 1: Post's lattice

Clone	Definition	Base	
BF	All Boolean functions	$\{x \land y, \neg x\}$	
R_0	$\{f \in \mathsf{BF} \mid f$ is 0-reproducing$\}$	$\{x \land y, x \oplus y\}$	
R_1	$\{f \in \mathsf{BF} \mid f$ is 1-reproducing$\}$	$\{x \lor y, x \leftrightarrow y\}$	
R_2	$\mathsf{R}_0 \cap \mathsf{R}_1$	$\{x \lor y, x \land (y \leftrightarrow z)\}$	
M	$\{f \in \mathsf{BF} \mid f$ is monotone$\}$	$\{x \land y, x \lor y, 0, 1\}$	
M_0	$\mathsf{M} \cap \mathsf{R}_0$	$\{x \land y, x \lor y, 0\}$	
M_1	$\mathsf{M} \cap \mathsf{R}_1$	$\{x \land y, x \lor y, 1\}$	
M_2	$\mathsf{M} \cap \mathsf{R}_2$	$\{x \land y, x \lor y\}$	
S_0	$\{f \in \mathsf{BF} \mid f$ is 0-separating$\}$	$\{x \to y\}$	
S_0^n	$\{f \in \mathsf{BF} \mid f$ is 0-separating of degree $n\}$	$\{x \to y, \mathrm{T}_2^{n+1}\}$	
S_1	$\{f \in \mathsf{BF} \mid f$ is 1-separating$\}$	$\{x \nrightarrow y\}$	
S_1^n	$\{f \in \mathsf{BF} \mid f$ is 1-separating of degree $n\}$	$\{x \nrightarrow y, \mathrm{T}_n^{n+1}\}$	
S_{02}^n	$\mathsf{S}_0^n \cap \mathsf{R}_2$	$\{x \lor (y \land \neg z), \mathrm{T}_2^{n+1}\}$	
S_{02}	$\mathsf{S}_0 \cap \mathsf{R}_2$	$\{x \lor (y \land \neg z)\}$	
S_{01}^n	$\mathsf{S}_0^n \cap \mathsf{M}$	$\{\mathrm{T}_2^{n+1}, 1\}$	
S_{01}	$\mathsf{S}_0 \cap \mathsf{M}$	$\{x \lor (y \land z), 1\}$	
S_{00}^n	$\mathsf{S}_0^n \cap \mathsf{R}_2 \cap \mathsf{M}$	$\{x \lor (y \land z), \mathrm{T}_2^3\}$	if $n = 2$,
		$\{\mathrm{T}_2^{n+1}\}$	if $n \geq 3$
S_{00}	$\mathsf{S}_0 \cap \mathsf{R}_2 \cap \mathsf{M}$	$\{x \lor (y \land z)\}$	
S_{12}^n	$\mathsf{S}_1^n \cap \mathsf{R}_2$	$\{x \land (y \lor \neg z), \mathrm{T}_n^{n+1}\}$	
S_{12}	$\mathsf{S}_1 \cap \mathsf{R}_2$	$\{x \land (y \lor \neg z)\}$	
S_{11}^n	$\mathsf{S}_1^n \cap \mathsf{M}$	$\{\mathrm{T}_n^{n+1}, 0\}$	
S_{11}	$\mathsf{S}_1 \cap \mathsf{M}$	$\{x \land (y \lor z), 0\}$	
S_{10}^n	$\mathsf{S}_1^n \cap \mathsf{R}_2 \cap \mathsf{M}$	$\{x \land (y \lor z), \mathrm{T}_2^3\}$	if $n = 2$,
		$\{\mathrm{T}_n^{n+1}\}$	if $n \geq 3$
S_{10}	$\mathsf{S}_1 \cap \mathsf{R}_2 \cap \mathsf{M}$	$\{x \land (y \lor z)\}$	
D	$\{f \in \mathsf{BF} \mid f$ is self-dual$\}$	$\{\mathrm{maj}(x, \neg y, \neg z)\}$	
D_1	$\mathsf{D} \cap \mathsf{R}_2$	$\{\mathrm{maj}(x, y, \neg z)\}$	
D_2	$\mathsf{D} \cap \mathsf{M}$	$\{\mathrm{maj}(x, y, z)\}$	
L	$\{f \in \mathsf{BF} \mid f$ is affine$\}$	$\{x \oplus y, 1\}$	
L_0	$\mathsf{L} \cap \mathsf{R}_0$	$\{x \oplus y\}$	
L_1	$\mathsf{L} \cap \mathsf{R}_1$	$\{x \leftrightarrow y\}$	
L_2	$\mathsf{L} \cap \mathsf{R}_2$	$\{x \oplus y \oplus z\}$	
L_3	$\mathsf{L} \cap \mathsf{D}$	$\{x \oplus y \oplus z \oplus 1\}$	
E	$\{f \in \mathsf{BF} \mid f$ is constant or a conjunction$\}$	$\{x \land y, 0, 1\}$	
E_0	$\mathsf{E} \cap \mathsf{R}_0$	$\{x \land y, 0\}$	
E_1	$\mathsf{E} \cap \mathsf{R}_1$	$\{x \land y, 1\}$	
E_2	$\mathsf{E} \cap \mathsf{R}_2$	$\{x \land y\}$	
V	$\{f \in \mathsf{BF} \mid f$ is constant or a disjunction$\}$	$\{x \lor y, 0, 1\}$	
V_0	$\mathsf{V} \cap \mathsf{R}_0$	$\{x \lor y, 0\}$	
V_1	$\mathsf{V} \cap \mathsf{R}_1$	$\{x \lor y, 1\}$	
V_2	$\mathsf{V} \cap \mathsf{R}_2$	$\{x \lor y\}$	
N	$\{f \in \mathsf{BF} \mid f$ is essentially unary$\}$	$\{\neg x, 0, 1\}$	
N_2	$\mathsf{N} \cap \mathsf{D}$	$\{\neg x\}$	
I	$\{f \in \mathsf{BF} \mid f$ is constant or a projection$\}$	$\{id, 0, 1\}$	
I_0	$\mathsf{I} \cap \mathsf{R}_0$	$\{id, 0\}$	
I_1	$\mathsf{I} \cap \mathsf{R}_1$	$\{id, 1\}$	
I_2	$\mathsf{I} \cap \mathsf{R}_2$	$\{id\}$	

Figure 2: The list of all Boolean clones with definitions and bases

We call u the prefix and v the period of a k-word-model (u, v). One can imagine a k-word-model as a Kripke-Structure with k worlds, where there is an edge between world i and $i+1$ for every $i = 0, \dots, k$ and a back edge from world k to a previous world r (r corresponds to the first word of the period). The word i then has to hold in world i. Figure 3 shows a k-word-model represented as a graph.

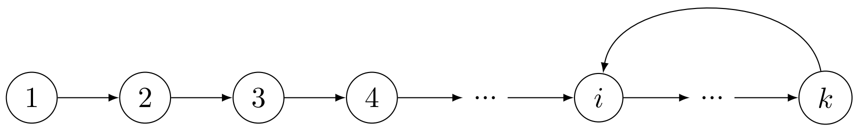

Figure 3: k-world-model

2.3 Counting Complexity

We continue with a brief introduction to counting complexity [AB09]. In this Section we define the counting classes #P, #PSPACE and FP. Furthermore, we finally define the problem #LTLSAT, that is, computing the number of k-word-models to a given LTL-formula. Moreover, we define hardness, completeness and two different reduction types for the classes #P and #PSPACE.

Definition 2.6. A function $f\colon \Sigma^* \to \mathbb{N}$ is in the class #P (resp. #PSPACE), if there is a nondeterministic polynomial time (resp. space) Turing machine M such that $f(w)$ is equal to the number of accepting runs of M on w.

Similar to the complexity class P in classical complexity theory we want to talk about functions that are computable in polynomial time. This class is called FP. Next we will define some counting problems. We start with the problem #MONOTONE 2-SAT which we later need for a reduction.

Problem:	#MONOTONE 2-SAT
Input:	Monotone formula φ in 2-CNF
Output:	Number of satisfying assignments for φ

Monotone refers to the property mentioned in Section 2.1 ($x_1 \leq y_1, \dots, x_n \leq y_n \implies f(x_1, \dots, x_n) \leq f(y_1, \dots, y_n)$), and here means that only the base functions $\{0, 1, \vee, \wedge\}$ of M are allowed.

The next problem is the counterpart to the propositional satisfiability problem in the counting world, namely #SAT.

Problem:	#SAT(B)
Input:	Formula φ over [B]
Output:	Number of satisfying assignments for φ

There are results of Reith that show the complexity of #SAT over every clone of Post's lattice [Rei02]. Some of these results we can later transfer to our LTL counterpart #LTLSAT.

The k-word-model problem over a set of Boolean functions B and $T \subseteq \{X, F, G, U, W, S\}$ is defined as follows:

Problem:	#LTLSAT(B, T)
Input:	Formula $\varphi \in \text{LTL}(B, T), k \in \mathbb{N}$
Output:	Number of k-word-models of φ

These are all the problems we need to show our results. But as we want to show hardness and completeness (for #P and #PSPACE) for several fragments of #LTLSAT, we have to define these terms. We start with defining two different reductions that we use to show hardness results for #P and #PSPACE.

Definition 2.7. Let #A and #B be two counting problems. #A is *parsimonious reducible* to #B ($\leq_1^p$), if there is a polynomial time computable function f, such that for all x:

$$\#\text{A}(x) = \#\text{B}(f(x))$$

Note that a parsimonious reduction is essentially an injective Karp reduction between counting problems. We will need another reduction type later. After defining completeness, it will be clear why it is necessary.

Definition 2.8. Let #A and #B be two counting problems. #A is $\leq_{1\text{-}T}^p$ reducible to #B, if there are two polynomial time computable functions f, g, such that for all x:

$$\#\text{A}(x) = g(\#\text{B}(f(x)), x)$$

We will now define hardness and completeness for the considered classes with respect to the reductions we just introduced.

Definition 2.9. A counting-problem #B is #P-hard with respect to $\leq_1^p$ ($\leq_{1\text{-}T}^p$), if every #A $\in$ #P is reducible to #B.

With the definition of #P-hardness we have everything we need to define #P-completeness.

Definition 2.10. A counting-problem #B is #P-complete w.r.t. $\leq_1^p$ ($\leq_{1\text{-}T}^p$), if it is #P-hard w.r.t. $\leq_1^p$ ($\leq_{1\text{-}T}^p$) and #B $\in$ #P.

Note that it also suffices to give a reduction from a #P-complete problem #B to another problem #A to show hardness for #A. This holds, because both reduction types that we use are transitive.

Analogously to #P we define hardness and completeness for #PSPACE.

Definition 2.11. A counting-problem #B is #PSPACE-hard with respect to $\leq_1^p$ ($\leq_{1\text{-}T}^p$), if every #A $\in$ #PSPACE is reducible to #B.

Definition 2.12. A counting-problem #B is #PSPACE-complete w.r.t. $\leq_1^p$ ($\leq_{1\text{-}T}^p$), if it is #PSPACE-hard w.r.t. $\leq_1^p$ ($\leq_{1\text{-}T}^p$) and #B $\in$ #PSPACE.

The Problem #MONOTONE 2-SAT is #P-complete with respect to $\leq_{1\text{-}T}^p$, which was shown by Valiant [Val79]. Furthermore it is known, that #MONOTONE 2-SAT can not be #P-complete with respect to parsimonious reductions unless $P = NP$. This holds because if #MONOTONE 2-SAT would be #P-complete with respect to parsimonious reductions there would in particular be a parsimonious reduction from #SAT(BF) to #MONOTONE 2-SAT. This would translate to a Karp reduction between the counterparts SAT(BF) and MONOTONE 2-SAT. Hence, MONOTONE 2-SAT would be NP-complete. But because there is a polynomial time algorithm for 2-SAT and therefore also for MONOTONE 2-SAT it would follow that P = NP.

After the introduction to counting complexity, Boolean functions and Linear Temporal Logic we can have a look at the results.

3 Counting Word-Models

We will now present our results for #LTLSAT. We split this Section up between the hard and the tractable cases. Hard means in this context either complete for #P or #PSPACE with respect to any of the defined reductions. Tractable means member of the class FP.

3.1 Hard cases

We will start with the #P- and #PSPACE-complete problems. The first result shows the complexity of full #LTLSAT, that is, with all Boolean functions and temporal operators allowed. We want to differentiate between two cases, one where k is coded binary and one where k is coded unary. After that, we can transfer some results for #SAT to #LTLSAT as mentioned in Section 2.3. The last two results of this Section are pretty similar to each other, they both use only the W temporal operator, they only differ in the allowed Boolean functions.

For full #LTLSAT with binary coded k we show #PSPACE-completeness — we follow the proof of Torfah and Zimmermann [TZ14]. For the #PSPACE-hardness part we map paths of computation trees of Turing machines to k-word-models. We want to make sure that there is exactly one k-word-model for every accepting path of a fixed #PSPACE Turing machine on any input. Therefore, we construct a formula that forces all configurations to be satisfied in the prefix and forces the period to only contain one dummy symbol and be of length one. Furthermore, we ensure that all coded paths in the model have the same length, by repeating the last configuration if needed. This is needed to make sure, that the dummy symbol only occurs within the period.

Theorem 3.1. *#LTLSAT(B, T) is #PSPACE-hard w.r.t. parsimonious reductions ($\leq_1^p$), for $[B] = BF$ and $T = \{X, F, G, U, W, S\}$, if k is coded binary.*

Proof. We show the hardness by giving a generic parsimonious reduction. Let M be a one-tape nondeterministic polynomial space Turing machine. That means that there is a polynomial p, such that M halts on any input x using not more than $p(|x|)$ tape cells. Furthermore M is $c \cdot 2^{p(|x|)}$-time bounded, for some constant c, because there are not more than $O(2^{p(|x|)})$ different configurations possible. We now define an LTL-formula φ_M^x and bound k, that are polynomial in $|x|$ and $|M|$, such that the number of different k-word-models for φ_M^x is equal to the number of accepting paths of M on x.

We assume the following coding for a path of M on input x:

$$\$\ id_0\ \#\ c_0\ \$\ id_1\ \#\ c_1\ \$\ \ldots\ \$\ id_{2^{p(|x|)}}\ \#\ c_{2^{p(|x|)}}\ \$\ (\bot)^\omega,$$

where id_i is the number of the configuration i, c_i the configuration itself and $\#, \$, \bot$ are dummy-symbols that are not part of the alphabet of M. If a path consists of less than $2^{p(|x|)}$ configurations then we repeat the final configuration. By doing that we can choose k such that the period of a k-word-model only consists of the symbol $\bot$.

The formula φ_M^x consists of 6 subformulae that ensure that there is exactly one k-word-model for each accepting path. The subformula φ_{init} ensures that the first configuration

c_0 is the initial configuration, φ_{acc} asserts that the last configuration is an accepting configuration. φ_{config} ensures the consistency of two successive configurations with the transition relation of M. φ_{repeat} ensures that the last configuration is repeated until the maximum length is reached. φ_{loop} forces the period of the k-word-model to only consist of $\perp$ and φ_{id} is used to ensure the correctness of the id's. The formulae can be found in the paper of Torfah and Zimmermann in the appendix section [TZ14]. Note that all the formulae are polynomial-sized compared to the input.

For $k = c \cdot 2^{p(|x|)} \cdot (p(|x|) + 3) + 1$, each accepting path of M on x corresponds to exactly one k-word-model of φ_M^x. Note that the value of k is exponential compared to the input but also is coded binary and therefore only takes polynomial space. Thus, the number of k-word-models of φ_M^x is equal to the number of accepting paths of M on x and the formula φ_M^x can be obtained in polynomial time. $\square$

Having shown #PSPACE-hardness, we now need to show membership to get completeness. For this, we follow the proof of Torfah and Zimmermann [TZ14]. We construct a nondeterministic Turing machine that guesses a k-word-model and verifies it on-the-fly. To make sure, that the Turing machine only uses polynomial space it guesses only one word at a time and verifies it in the corresponding world.

Theorem 3.2. *#LTLSAT$(B, T) \in$ #PSPACE for $[B] = BF$ and $T = \{X, F, G, U, W, S\}$, if k is coded binary.*

Proof. We construct a nondeterministic polynomial space Turing machine M that guesses a k-word-model word by word. That means M stores only one word at a time and uses a counter to guess exactly k words to ensure the space requirement. Furthermore M stores a set C_i of subformulae of the input formula φ for every $0 \leq i \leq k$, such that C_i contains exactly the subformulae that are satisfied in position i of the k-word-model. Again M stores not all of these C_i at once, but for the current, the next and the k-th position, due to the space requirement. The set C_k is guessed by M at the start of the computation and the sets C_i are determined by the following rules:

1. C_i contains all atomic propositions determined by the i-th word of the guessed k-word-model $w(i)$ ($C_i \cap P = w(i)$)

2. All conjunctions, disjunctions and negations can be checked locally for consistency (e.g. $\neg\psi \in C_i$ iff $\psi \notin C_i$)

3. Next formulae are propagated backwards using the following equivalence: $X\psi \in C_i$ iff $\psi \in C_{i+1}$

4. Until formulae are propagated backwards using the following equivalence: $\psi_0 U \psi_1 \in C_i$ iff $\psi_1 \in C_i$ or ($\psi_0 \in C_i$ and $\psi_0 U \psi_1 \in C_{i+1}$)

Note that we do not need to care about other temporal operators because they can be rewritten using X and U. Once M has verified the complete period of the k-word-model it also checks the correctness of the previous guessed C_k, this is done analogously to the verification of the C_i. Now it holds for every subformula ψ of φ that $\psi \in C_i$ iff ψ is satisfied by the "reduced" k-word-model that starts at position i. Therefore, our guessed k-word-model is a model of φ iff $\varphi \in C_0$ iff M accepts. □

Next we want to show #P-completeness for the same problem with unary coded k. First we show the membership in #P. We have to construct a nondeterministic polynomial time Turing machine M, such that the number of accepting paths on input (φ, k) is equal to the number of k-word-models for (φ, k). This can be done by guessing the k-word-model and verifying it in polynomial time. The latter would normally not work in polynomial time, as model checking for LTL is #PSPACE-complete [SC85]. However, we can use an algorithm that is actual used for model checking for CTL-formulae. CTL is an extension of LTL and allows the path quantifiers A and E, where A is the universal quantifier that can be used to quantify over all paths and E is the existential quantifier. It holds that $A\varphi \equiv \neg E \neg \varphi$. In CTL-formuale every temporal operator must be under the scope of such a path quantifier. Given a CTL-formula φ and a structure S one can check whether S satisfies φ in time $O(|S| \cdot |\varphi|)$ (see [CES86, MS03]). Note, that these structures are not the ones we defined in Definition 2.4. We can now use the fact that on models that are essentially paths – like our k-word-models –, model checking for CTL and LTL coincides.

Theorem 3.3. *#LTLSAT$(B, T) \in$ #P for $[B] = BF$ and $T = \{X, F, G, U, W, S\}$, if k is coded unary.*

Proof. Let (φ, k) be the input an let M be a nondeterministic Turing machine that works as follows. M guesses a k-word-model (u, v), with $|uv| = k$ and verifies in polynomial time whether (u, v) satisfies φ by using the algorithm for CTL model checking [CES86]. Hence, for every k-word-model (u, v) of φ there is exactly one accepting path of M on input (φ, k). □

To show #P-hardness for full #LTLSAT with unary coded k one could use the same approach as for the binary coded case. Note, that the paths of #P Turing machines have only polynomial length and therefore can be expressed even with a unary coded k.

However, we use a simpler approach. We use the fact that LTL is an extension of propositional logic. Essentially, propositional logic is equivalent to LTL, if we only allow one world and no successors. This leads to a simple reduction #SAT(B) to #LTLSAT(B, T). We want that for every satisfying assignment for a propositional formula φ we get exactly one k-word-model for our input formula for #LTLSAT(B, T). This can be simply done by setting k to 1 and by that forcing the size of the model to be 1 as well. We do not even have to change the formula.

Theorem 3.4. *For a set of Boolean functions B and $T \subseteq \{X, F, G, U, W, S\}$ #SAT(B) is parsimonious reducible to #LTLSAT(B, T).*

Proof. Let φ be a Boolean formula over a set of propositions P and $f(\varphi) = (\varphi, 1)$. Then for every satisfying assignment θ of φ there is exactly one k-word-model for $(\varphi, 1)$, namely (ϵ, θ). □

With this reduction we can transfer some hardness results that hold for #SAT to #LTLSAT. A full characterization of #SAT is given by Reith [Rei02].

Corollary 3.1. *#LTLSAT(B, T) is #P-hard for $T \subseteq \{X, F, G, U, W, S\}$ and:*

1. *$S_1 \subseteq [B] \subseteq BF$, with respect to $\leq_1^p$*
2. *$S_{02} \subseteq [B] \subseteq R_1, S_{12} \subseteq [B] \subseteq R_1, S_{00} \subseteq [B] \subseteq M$ or $S_{10} \subseteq [B] \subseteq M$, with respect to $\leq_{1\text{-}T}^p$*

Proof. This follows directly from the fact, that #SAT(B) is #P-complete with respect to $\leq_1^p$ (resp. $\leq_{1\text{-}T}^p$) for the mentioned clones [Rei02]. □

Furthermore the problems mentioned in corollary 3.1 are ∈ #P and thus #P-complete, when k is coded unary. This holds because they are fragments of full #LTLSAT and therefore can not be harder.

Now we want to have a look at those clones for which #SAT is computable in polynomial time but #LTLSAT remains #P-hard. Those fragments of #LTLSAT need at least one temporal operator (it would be basically #SAT otherwise). We will now show two results for problems that use the temporal operator W. This operator has the useful property, that one can simulate $\vee$, when the model consists of only one world. We can force such one world models in a reduction by setting k to 1 like in theorem 3.4. With this in mind we have to look at clones B for which #SAT$(B \cup \{\vee\})$ is #P-complete. This is the case for the E-clones ($E \cup \{\vee\} = M$) and the N-clones ($N \cup \{\vee\} = BF$). We start with a reductions from #MONOTONE 2-SAT to #LTLSAT$(E, \{W\})$.

Theorem 3.5. *#LTLSAT$(B, \{W\})$ is #P-hard w.r.t. $\leq_{1\text{-}T}^p$ for $E_2 \subseteq [B] \subseteq E$.*

Proof. We reduce from the #P-complete problem #MONOTONE 2-SAT via the function f. Let φ be an input formula for #MONOTONE 2-SAT and $f(\varphi) = (\varphi', 1)$, where $\varphi = \bigwedge_{i=0}^{m}(y_{i1} \vee y_{i2})$ and $\varphi' = \bigwedge_{i=0}^{m}(y_{i1} W y_{i2})$. We simply replaced every occurrence of $\vee$ by W and forced a singleton model. It follows that for every satisfying assignment for φ there is exactly one k-word-model for φ'. This holds because $y_{i1} W y_{i2}$ is true on a one world model iff y_{i1} or y_{i2} holds in world 1.

□

As mentioned above the next reduction is very similiar to the one given in Theorem 3.5. But this time we can reduce from the problem #SAT(BF), which is #P-complete with respect to parsimonious reduction. Therefore, we show #P-hardness for #LTLSAT$(B, \{W\})$ with respect to parsimonious reductions.

Theorem 3.6. *#LTLSAT$(B, \{W\})$ is #P-hard w.r.t. $\leq_1^p$ for $N_2 \subseteq [B] \subseteq N$*

Proof. The proof works similar to the one presented in Theorem 3.5. This time we reduce from #SAT(BF), which is #P-complete as mentioned before. Let φ be an input formula for #SAT(BF). We again replace every occurrence of $\vee$ in φ with the temporal operator W – which acts like a $\vee$ if the model consists of only one world – and receive the formula φ'. Because $\{\vee, \neg\} = BF$, we are able to express the whole input formula φ this way. Analogously to Theorem 3.5 it follows, that for every satisfying assignment θ for φ there exists exactly one k-word-model for $(\varphi', 1)$, namely (ε, θ). □

Again it holds for the problems from Theorem 3.5 and 3.6 that they are member of the class #P and therefore are #P-complete, if k is coded unary (see Theorem 3.3).

Now we will leave the Section about completeness results and have a to look at some polynomial time results.

3.2 Polynomial time results

For #LTLSAT(B, T) to be computable in polynomial time, B has to be a clone for which #SAT$(B) \in FP$. Furthermore, we only want to look at problems that use a nonempty sets of temporal operators because we would essentially talk about #SAT otherwise. We will see results for the "clone-families" E, V, N and I. #SAT(B) is of course computable in polynomial time for those clones. For the E and V-clones, we give respectively an algorithm that computes the number of k-word-models of a given input formula in polynomial time. The hard part of such an algorithm is to make sure not to count a model twice and hence compute the wrong number. We will start with the results for the E clones.

Theorem 3.7. *#LTLSAT$(B, \{X, G\}) \in FP$, for $[B] \subseteq E$.*

Proof. First we show that there is a normal form for formulae $\varphi \in \text{LTL}(E, \{X, G\})$ and then give an algorithm that computes the number of k-word-models of φ in polynomial time.

Claim. *For every formula $\varphi \in LTL(E, \{X, G\}), \varphi \not\equiv 0$ there exists a formula $\varphi' = \bigwedge_{i=0}^{m} X^i(\varphi_i \wedge G\varphi_i')$ with $\varphi \equiv \varphi'$, where φ_i and φ_i' are conjunctions of propositions.*

Here, "$\equiv$" means that there exist exactly the same amount of k-word-models for both formulae.

Proof of the claim. First of all we can assume that the formula contains no 1 or 0, because $\wedge 1, G1$ and $X1$ do not change the number of models and $\wedge 0, G0$ and $X0$ provide a formula equal to 0. The form of φ' is then achieved by the equivalence of the following formulae:

$$GG\psi \equiv G\psi$$

$$X^i G X^j \psi \equiv X^{i+j} G\psi$$

$$X^i\psi_1 \wedge X^i\psi_2 \equiv X^i(\psi_1 \wedge \psi_2)$$

$$G\psi_1 \wedge G\psi_2 \equiv G(\psi_1 \wedge \psi_2)$$

□

Because the only Boolean operator is $\wedge$, we have to label all the propositions of the formula in their respective worlds. Any other propositions can be labeled arbitrary to get a k-word-model. The Algorithm works as follows: First of all we choose were the period starts, this is done by the first "for" loop. For every world l we store all propositions in $L(l)$. A stored proposition in a world means here, that it is unimportant if that proposition is labeled in world i or not to get a k-word-model. We simply have to update the different $L(l)$'s and count afterwards the number of different assignments to the stored propositions. The lines 6-10 are responsible for updating the $L(l)$'s addressed by the subformulae $X^i\varphi_i$. If $i \leq k$ then we can simply delete these propositions from $L(i)$ (we have to label these in a k-word-model). If $i > k$ then the formula addresses not world i but world $((i-r) \mod (k-r)) + r$ instead, where r is the world where the period starts. For subformulae of the form $X^iG\varphi'_i$ we have to delete every proposition from worlds $\geq i$ respectively $\geq r$ if $i \geq r$.

□

Algorithm 1: Polynomial time algorithm for #LTLSAT$(B, \{X, G\})$

```
   Input  : $\varphi = \bigwedge_{i=0}^{m} X^i(\varphi_i \wedge G\varphi'_i)$, $k \in \mathbb{N}$
 1 $s = 0$
 2 for $r = 1, \ldots, k$ do
 3     for $l = 1, \ldots, k$ do
 4         $L(l) \leftarrow x_1, \ldots, x_n$
 5     end
 6     for $i = 1, \ldots, m$ do
 7         if $i \leq k$ then
 8             delete every $x_j \in \varphi_i$ from $L(i)$
 9         else
10             delete every $x_j \in \varphi_i$ from $L(((i-r) \mod (k-r)) + r)$
11         end
12         if $i \leq r$ then
13             delete every $x_j \in \varphi'_i$ from $L(i), \ldots L(k)$
14         else
15             delete every $x_j \in \varphi'_i$ from $L(r), \ldots L(k)$
16         end
17     end
18     $s = s + \sum_{l=1}^{k} 2^{|L(l)|}$
19 end
   Output: $s$
```

The next result will be about the V-clones and we will allow only X to be used as a temporal operator. As in the previous theorem the main difficulty is to not count any models twice.

Theorem 3.8. *#LTLSAT$(B, \{X\}) \in FP$, for $[B] \subseteq V$.*

Proof. We can assume that the formula is of the form $\varphi = \bigvee_{i=0}^{m} X^i \varphi_i$, where the φ_i are disjunctions of propositions.

The algorithm works as follows. We store a value for every proposition and every world in the array L and for every subformula $X^i\varphi_i$ we store a value of all propositions of φ_i in the array L'. We need two arrays this time to make sure not to count a model twice. Now we count successive the models that satisfy $X^i\varphi_i$ by counting only models that do not have labeled any proposition from a subformula $X^j\varphi_j, j \leq i$ in their respective world.

$\square$

Algorithm 2: Polynomial time algorithm for #LTLSAT$(B, \{X, G\})$

```
Input  : $\varphi = \bigvee_{i=0}^{m} X^i \varphi_i$, $k \in \mathbb{N}$
1  $s = 0$
2  for $r = 1, \dots, k$ do
3      for $l = 1, \dots, k$ do
4          $L(l) \leftarrow x_1, \dots, x_n$
5      end
6      for $i = 1, \dots, m$ do
7          if $i \leq k$ then
8              store every $x_j \in \varphi_i$ in $L'(i)$
9          else
10             store every $x_j \in \varphi_i$ in $L'(((i - r) \mod (k - r)) + r)$
11         end
12     end
13     for $i = 1, \dots, k$ do
14         delete every $x_j \in L'(i)$ from $L(i)$
15         $s' = 1$
16         for $t = 1, \dots, k$ do
17             if $i = t$ then
18                 $s' = s' \cdot (2^{|L'(t)|} - 1) \cdot 2^{L(t)}$
19             else
20                 $s' = s' \cdot 2^{L(t)}$
21             end
22         end
23         $s = s + s'$
24     end
25 end
Output: $s$
```

Now we have a look at those clones that do not allow any binary Boolean functions, i.e. I and N. We saw in Section 3.1 that #LTLSAT$(N, \{W\})$ is already #P-complete, therefore we will also only allow unary temporal operators. In the proof we use the "$\varphi \equiv \psi$" (respectively φ is equivalent to ψ), which means in this context that there exist exactly the same amount of k-word-models for both formulae φ, ψ.

Theorem 3.9. *$\#LTLSAT(B, \{X, G, F\}) \in FP$, for $[B] \subseteq N$.*

Proof. Let φ be the input formula. We show that φ is equivalent to one of the following formulae:

1. GX^iy
2. FX^iy
3. FGX^iy
4. GFX^iy

where y is a proposition or a negated proposition. Note that $\neg F\psi \equiv G\neg\psi$, $\neg G\psi \equiv F\neg\psi$ and $\neg X\psi \equiv X\neg\psi$ in k-word-models. Therefore, we can assume that we only have propositional negation. As it makes no difference for the number of k-word-models whether the proposition is negated or not, we assume w.l.o.g. that y is a positive proposition. Furthermore, it holds that $GFGy \equiv FGy$ and $FGFy \equiv GFy$, which we will proof after we showed, that we can handle the four cases mentioned above.

Case 1: The first formula is easy to handle, for a model to satisfy φ the proposition y has to be labeled in every world $\geq min(i, r)$, where r is the first world of the period. All other (degenerated) propositions can be labeled arbitrarily.

Case 2: It suffices to label y in one world $j \geq i$, in the other world y can be treated like the rest of the propositions. To ensure not to count any model twice we may not label y in a world $j \geq i$ again (when r is fixed).

Case 3: A model satisfies the third type of formula if y is labeled in every world of the period. Whether or not y holds in a world of the prefix does not matter because the F does not address a particular world. Thus, for a fixed r we can count y like a degenerated proposition in the prefix worlds.

Case 4: This type of formula is similar to the second one. The only difference is that every model needs to have y labeled in one of the period worlds because of the G.

Now we have to show that $GFGy \equiv FGy$ and $FGFy \equiv GFy$ holds. We start with $GFGy \equiv FGy$:

We show that every k-word-model that satisfies $GFGy$ also satisfies FGy and vice versa.

"$\Leftarrow$": Let (u, v) be a k-word-model that satisfies FGy. Then, y holds in every world of the period (see 3). Thus, FGy holds in every world i, for $1 \leq i \leq k$ and therefore $GFGy$ holds in world 1.

"$\Rightarrow$": Let (u, v) be a k-word-model that satisfies $GFGy$. That means in every world i, for $1 \leq i \leq k$ the formula FGy is satisfied by (u, v), in particular in world 1.

Lastly we show that $FGFy \equiv GFy$ holds:

"$\Leftarrow$": Let (u, v) be a k-word-model that satisfies GFy. It follows directly that $FGFy$ is also satisfied by (u, v).

"$\Rightarrow$": Let (u, v) be a k-word-model that satisfies $FGFy$. Thus, the formula GFy is satisfied by (u, v) in at least one world i, for $1 \leq i \leq k$. For (u, v) to satisfy GFy in world i the proposition y must hold in at least one world of the period (similar to 4). It follows that GFy is satisfied by (u, v) in world 1.

$\square$

4 Conclusion and Outlook

In this work we analyzed the problem of counting k-word-models for Linear Temporal Logic formulae, namely #LTLSAT. We differentiated between two cases of the problem on the basis of the coding of k. For the case of binary coded k we followed the theorem of Torfah and Zimmermann [TZ14], that states #PSPACE-completeness for full LTL. For the unary coded case we showed various results for different fragments of the problem, starting with transferring lower bounds from #SAT to #LTLSAT. This way, we showed #P-completeness for #LTLSAT for the S-clones, M-clones, R-clones and the clone BF. Furthermore, we showed for the N-clones that the problem #LTLSAT is computable in polynomial time if we only allow unary temporal operators, but becomes #P-complete if we allow the binary operator W. For this case it would be interesting to know if one could come up with a similar reduction using the temporal operator U. However such a reduction would likely be not parsimonious. For the E-clones we had similar results. We showed the membership in FP if we allow the temporal operators X and G and again #P-completeness for the operator W. For the latter it would be again interesting to see a version of this proof using the operator U (this time the reduction type would not change compared to our result as we already used a $\leq^p_{1\text{-}T}$ reduction). We saw an algorithm for #LTLSAT$(V, \{X\})$ that runs in polynomial time and therefore shows membership in FP.

We gave results for most clones of Post's lattice, but many cases remain open. There are many possibilities for #P-complete problems. It would be for example interesting to know whether #LTLSAT$(E, \{F\})$ is #P-complete with respect to any of the given reductions or computable in polynomial time. For the "or"-case — that is #LTLSAT$(V, \{F\})$ — it is very likely that this problem is in the class FP, because one can achieve a rather simple normal form. In addition any results regarding the D- and L-clones would be interesting. As we did not investigate these clones, all cases remain open. At last one could have a look at the $\leq^p_{1\text{-}T}$ reductions. It is entirely possible that some of the problems for which we showed #P-completeness with respect to $\leq^p_{1\text{-}T}$, that they are also #P-complete with respect to parsimonious reductions.

References

[AB09] Arora, S., & Barak, B. (2009). *Computational complexity: a modern approach.* Cambridge University Press.

[BSS+08] Bauland, M., Schneider, T., Schnoor, H., Schnoor, I., & Vollmer, H. (2008). *The complexity of generalized satisfiability for linear temporal logic.* arXiv preprint arXiv:0812.4848.

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

[MS03] Markey, N., & Schnoebelen, P. (2003). *Model checking a path.* In CONCUR 2003-concurrency theory (pp. 251-265). Springer Berlin Heidelberg.

[Pnu77] Pnueli, A. (1977, October). *The temporal logic of programs.* In Foundations of Computer Science, 1977., 18th Annual Symposium on (pp. 46-57). IEEE.

[Rei02] Reith, S. (2002). *Generalized satisfiability problems* (Doctoral dissertation, Julius Maximilian University of Würzburg).

[SC85] Sistla, A. P., & Clarke, E. M. (1985). *The complexity of propositional linear temporal logics.* Journal of the ACM (JACM), 32(3), 733-749.

[Sch02] Schnoebelen, P. (2002). *The Complexity of Temporal Logic Model Checking.* Advances in modal logic, 4(393-436), 35.

[TZ14] Torfah, H., & Zimmermann, M. (2014). *The complexity of counting models of linear-time temporal logic.* arXiv preprint arXiv:1408.5752.

[Val79] Valiant, L. G. (1979). *The complexity of enumeration and reliability problems.* SIAM Journal on Computing, 8(3), 410-421.

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, den March 15, 2016

Fabian Müller