First-Order vs. Second-Order Encodings for LTL_f-to-Automata: An Extended Abstract

Shufang Zhu, Geguang Pu
East China Normal University
Shanghai, China

Moshe Y. Vardi
Rice University
Texas, USA

Abstract

Translating formulas of Linear Temporal Logic (LTL) over finite traces, or LTL_f, to symbolic Deterministic Finite Automata (DFA) plays an important role not only in LTL_f synthesis, but also in synthesis for Safety LTL formulas. The translation is enabled by using MONA, a sophisticated tool for symbolic DFA construction from logic specifications. Recent works used a first-order encoding of LTL_f formulas to translate LTL_f to First Order Logic (FOL), which is then fed to MONA to construct the symbolic DFA. This encoding was shown to perform well, but other encodings have not been studied. Specifically, the natural question of whether second-order (MSO) encoding, which has significantly simpler quantificational structure, can outperform first-order (FOL) encoding remained open.

In this paper we meet this challenge and study MSO encoding for LTL_f formulas. We introduce a specific MSO encoding, and show that this encoding and its simplicity indeed allow more potential than FOL for optimization, thus benefiting symbolic DFA construction. We then show by empirical evaluations that, surprisingly, the FOL encoding outperforms in practice the MSO encodings. We analyze the results and explain how to improve MONA in order to allow the MSO encoding to outperform the FOL encoding.

1 Introduction

Synthesis from temporal specifications [19] is a fundamental problem in Artificial Intelligence and Computer Science [5]. A popular specification is Linear Temporal Logic (LTL) [18]. The standard approach to solving LTL synthesis requires, however, both determinization of automata on *infinite* words and solving *parity games*, both challenging algorithmic problems [14]. Thus a major barrier of temporal synthesis has been algorithmic difficulty. One approach to combating this difficulty is to focus on using fragments of LTL, such as the GR(1) fragment, for which temporal synthesis has lower computational complexity [1].

A new logic for temporal synthesis, called LTL_f, was proposed recently in [4, 5]. The focus there is not on limiting the syntax of LTL, but on interpreting it semantically on *finite* traces, rather than *infinite* traces as in [18]. Such interpretation allows the executions being arbitrarily long, but not infinite. While limiting the semantics to finite traces does not change the computational complexity of temporal synthesis (2EXPTIME), the algorithms for LTL_f are much simpler. The reason is that those algorithms require determinization of automata on *finite* words (rather than *infinite* words), and solving *reachability* games (rather than *parity* games) [5]. Another application, as shown in [23], is that temporal synthesis of *Safety* LTL formulas, a syntactic fragment of LTL expressing *safety properties*, can be reduced to reasoning about finite words (see also [13, 16]). This approach has been implemented in [24] for LTL_f synthesis and in [23] for synthesis of *Safety* LTL formulas, and has been shown to outperform existing temporal-synthesis tools such as Acacia+ [2]. The key algorithmic building block in all these approaches is a translation of LTL_f to *symbolic* Deterministic Finite Automata (DFA).

The method utilized in [23, 24] for such translation of LTL_f to symbolic DFA used an encoding of LTL_f to First-Order Logic (FOL) that captures directly the semantics of temporal connectives, and MONA [10], a sophisticated tool, for symbolic DFA construction from logical specifications. This approach was shown to outperform explicit tools such as SPOT [9], but other encodings have not been studied. However, the FOL encoding relies on a syntax-driven translation that involves an arbitrary alternation of quantifiers, which limits the potential for optimization. Such fact leads us to study translations of LTL_f to Monadic Second Order (MSO) logic of one successor over finite words (called M2L-STR in [12]). Indeed, one possible advantage of using MSO is the simpler quantificational structure such that the MSO encoding requires only a sequence of existential monadic second-order quantifiers followed by a single universal first-order quantifier. Moreover, instead of the syntax-driven translation of FOL encoding, the MSO encoding employs a semantics-driven translation which allows more space for optimization. The natural question arises is whether second-order (MSO) encoding outperforms first-order (FOL) encoding.

To answer this question, we study MSO encodings for LTL_f formulas. We start by introducing an MSO encoding relying on having a second-order variable for each temporal operator appearing in the LTL_f formula and proving the correctness. We then show that this encoding allows more potential for optimization than the FOL encoding, described in [23]. In particular, we study the following optimizations that were introduced in [17, 22]: in the variable form, where

WiL'17, July 06–19,2018, Oxford, UK
2018.

a *Lean* encoding introduces less variables than the standard *Full* encoding; and in the constraint form, where the *Sloppy* encoding allows less constraints than the standard *Fussy* encoding. In addition, we present in this paper the first evaluation of the spectrum of encodings for LTL_f-to-automata from FOL to MSO.

Yet, our empirical evaluations show the superiority of the FOL encoding as a way to get MONA to generate a symbolic DFA, which surprisingly violates the intuition of MSO encoding. We analyze the way MONA handles quantifiers and observe that it processes a full block of like quantifiers one by one, rather than processing the whole block at once. This indicates that the current version of MONA is *not* the best choice for MSO encoding of LTL_f. Therefore with optimized quantifier elimination strategies, MSO encoding may outperform FOL encoding for symbolic DFA construction from LTL_f on MONA. Thus a further contribution of the paper is providing opportunities for the improvement of MONA.

In Section 2 we give preliminaries and notations. Section 3 introduces MSO encoding and corresponding variations in terms of optimizations and proves their correctness. Empirical evaluation results are presented in Section 4. Finally, Section 5 concludes and discusses. The proofs are omitted due to page limit.

2 Preliminaries

2.1 LTL_f Basics

Linear Temporal Logic over *finite traces* (LTL_f) has the same syntax as LTL [4]. Given a set $\mathcal{P}$ of atoms, the syntax of LTL_f formulas is as follows: $\phi ::= \top \mid \bot \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid X\phi \mid \phi_1 U \phi_2$, where $p \in \mathcal{P}$. We use $\top$ and $\bot$ to denote *true* and *false* respectively. X (Next) and U (Until) are temporal operators, whose dual operators are N (Weak Next) and R (Release) respectively, defined as $N\phi \equiv \neg X \neg\phi$ and $\phi_1 R \phi_2 \equiv \neg(\neg\phi_1 U \neg\phi_2)$. The abbreviations (Eventually) $F\phi \equiv \top U \phi$ and (Globally) $G\phi \equiv \bot R \phi$ are defined as usual. We assume standard semantics as defined in [4] and write $\rho \models \phi$ if $\rho \in (2^{\mathcal{P}})^*$ satisfies ϕ, where *trace* $\rho = \rho[0], \rho[1], \ldots, \rho[e]$ is a finite sequence of propositional assignments. Every LTL_f formula can be written in Boolean Normal Form (BNF) or Negation Normal Form (NNF). BNF rewrites the input formula using only $\neg$, $\wedge$, $\vee$, X, and U. NNF pushes negations inwards, introducing the dual temporal operators N and R, until negation is applied only to atoms.

2.2 Symbolic DFA and MONA

We start by defining the concept of symbolic Deterministic Finite Automata (DFA) [24], where a boolean formula is used to represent the transition function of a DFA. A symbolic DFA $\mathcal{F} = (\mathcal{P}, \mathcal{X}, X_0, \eta, f)$ corresponding to an explicit DFA $\mathcal{D} = (2^{\mathcal{P}}, S, s_0, \delta, F)$ is defined as follows: $\mathcal{P}$ is the set of atoms; $\mathcal{X}$ is a set of state variables where $|\mathcal{X}| = \lceil \log_2 |S| \rceil$; $X_0 \in 2^{\mathcal{X}}$ is the initial state corresponding to s_0; $\eta : 2^{\mathcal{X}} \times 2^{\mathcal{P}} \rightarrow 2^{\mathcal{X}}$ is a boolean transition function corresponding to δ; f is the acceptance condition expressed as a boolean formula over $\mathcal{X}$ such that f is satisfied by an assignment X iff X corresponds to a final state $s \in F$.

The MONA tool [10] is an efficient implementation for translating FOL and MSO formulas over finite words into minimized symbolic DFAs. Therefore, in this paper we evaluate FOL and MSO encodings on MONA .

2.3 LTL_f to FOL Encoding

First Order Logic (FOL) encoding of LTL_f translates LTL_f into FOL over finite linear order with monadic predicates. In this paper, we utilize the FOL encoding proposed in [4], where the translation returns an FOL formula $\text{fol}(\phi, 0)$ asserting the truth of LTL_f formula ϕ at point 0 of the linear order. For more details on FOL encoding, we refer to [4]. Given a finite trace ρ, we denote the corresponding finite linear ordered FOL interpretation of ρ by $\mathcal{I}_\rho$. The following theorem guarantees the correctness of FOL encoding of LTL_f .

Theorem 2.1 ([11]). *Let ϕ be an LTL_f formula and ρ be a finite trace. Then $\rho \models \phi$ iff $\mathcal{I}_\rho \models \text{fol}(\phi, 0)$.*

3 MSO Encodings

One immediate drawback of the FOL encoding is that it involves an arbitrary alternation of quantifiers due to the syntax-driven translation. This leads to limited optimization space in the sense that simplifying the translation such that benefiting subsequent symbolic DFA construction. By applying a semantics-driven translation to LTL_f, we obtain an MSO encoding that allows more optimization space. Intuitively speaking, MSO encoding deals with LTL_f formula by interpreting every operator with corresponding subformulas following exactly the semantics of the operator. We now define MSO encodings that translate LTL_f formula ϕ to MSO asserting the truth of ϕ at point 0 of the linear order. The MSO formula is then fed to MONA to produce a symbolic DFA. We first show the basic MSO encoding in Section 3.1, then describe variations of it in the following.

3.1 LTL_f to MSO Encoding

MSO is an extension of FOL that allows quantification over monadic predicates [12]. We first restrict our interest to *monadic structure*. Consider a finite trace $\rho = \rho[0]\rho[1]\cdots\rho[e]$ where for x such that $0 \leq x \leq e$, $\rho[x]$ indicates the set of atoms of $\mathcal{P}$ that are *true* at position x. Then the corresponding *monadic structure* $\mathcal{I}_\rho = (\Delta^I, <, \cdot^I)$ describes ρ as follows. $\Delta^I = \{0, 1, 2, \cdots, last\}$, where $last = e$. The linear order $<$ is defined over Δ^I in the standard way [12]. The notation $\cdot^I$ indicates the set of monadic predicates that describe the atoms of $\mathcal{P}$, where the interpretation of each $p \in \mathcal{P}$ is $p^I = \{x : p \in \rho[x]\}$. Intuitively, p^I is interpreted as the set of positions where p is true in ρ.

We now present MSO encoding of LTL$_f$. For an LTL$_f$ formula ϕ over a set $\mathcal{P}$ of atoms, let $cl(\phi)$ denote the set of subformulas of ϕ. We define atomic formulas as atoms $p \in \mathcal{P}$. For every subformula in $cl(\phi)$ we introduce monadic predicate symbols as follows: for each atomic subformula $p \in \mathcal{P}$, we have a monadic predicate symbol Q_p; for each non-atomic subformula $\theta_i \in \{\theta_1, \ldots, \theta_m\}$, we have Q_{θ_i}. Intuitively speaking, each monadic predicate indicates the positions where the corresponding subformula is true along the linear order.

Let $\mathsf{mso}(\phi)$ be the translation function that given an LTL$_f$ formula ϕ returns a corresponding MSO formula asserting the truth of ϕ at position 0. We define $\mathsf{mso}(\phi)$ as following: $\mathsf{mso}(\phi) = (\exists Q_{\theta_1}) \cdots (\exists Q_{\theta_m})(Q_\phi(0) \wedge (\forall x)(\bigwedge_{i=1}^{m} \mathrm{t}(\theta_i, x)))$, where x indicates the position along the finite linear order. Here $\mathrm{t}(\theta_i, x)$ asserts that the truth of every non-atomic subformula θ_i of ϕ at position x relies on the truth of corresponding subformulas at x such that follows the semantics of LTL$_f$. Therefore, $\mathrm{t}(\theta_i, x)$ is defined as follows:

- If $\theta_i = (\neg\theta_j)$, then $\mathrm{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow \neg Q_{\theta_j}(x))$
- If $\theta_i = (\theta_j \wedge \theta_k)$, then $\mathrm{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow (Q_{\theta_j}(x) \wedge Q_{\theta_k}(x)))$
- If $\theta_i = (X\theta_j)$, then $\mathrm{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow ((x \neq last) \wedge Q_{\theta_j}(x+1)))$
- If $\theta_i = (\theta_j U \theta_k)$, then $\mathrm{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow (Q_{\theta_k}(x) \vee ((x \neq last) \wedge Q_{\theta_j}(x) \wedge Q_{\theta_i}(x+1))))$

In the translation above, the successor function +1 and the variable *last* can be defined in terms of $<$. The following theorem asserts the correctness of the MSO encoding.

Theorem 3.1. *Let ϕ be an LTL$_f$ formula, ρ be a finite trace. Then $\rho \models \phi$ iff $\mathcal{I}_\rho \models \mathsf{mso}(\phi)$.*

3.2 Variations of MSO Encoding

The basic MSO encoding defined in Section 3.1 translates LTL$_f$ to MSO in a naive way, in the sense that introducing a second-order predicate for each non-atomic subformula and employing the $\leftrightarrow$ constraint. Inspired by [17, 22], we define in this section several optimizations to simplify such encoding thus benefiting symbolic DFA construction. We name MSO encoding with different optimizations *variation*. These variations are combinations of three independent components: (1) the Normal Form (choose between BNF or NNF); (2) the Constraint Form (choose between *Fussy* or *Sloppy*); (3) the Variable Form (choose between *Full* or *Lean*). In each component one can choose either of two options to make.

Thus for example, the variation described in Section 3.1 is BNF-*Fussy*-*Full*. Note that BNF-*Sloppy* are incompatible, as described below, and so there are $2^3 - 2 = 6$ viable combinations. We next describe the variations in details.

Constraint Form The translation described in Section 3.1 translating LTL$_f$ formula ϕ to MSO formula $\mathsf{mso}(\phi)$ employs an *iff* constraint, of the form $\mathrm{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow (Q_{\theta_j}(x) \wedge Q_{\theta_k}(x)))$ if $\theta_i = (\theta_j \wedge \theta_k)$ for example. We refer to this as *Fussy* variation. We now introduce *Sloppy* variation, inspired by [22], which allows looser constraints with correctness guarantee that may speed up the symbolic DFA construction.

For LTL$_f$ formulas in NNF, the *Sloppy* variation requires only a single implication constraint $\rightarrow$, yielding $\mathrm{t}_s(\theta_i, x) = (Q_{\theta_i}(x) \rightarrow (Q_{\theta_j}(x) \wedge Q_{\theta_k}(x)))$ for the same example. *Sloppy* variation $\mathsf{mso}_s(\phi)$ returns MSO formula $(\exists Q_{\theta_1}) \cdots (\exists Q_{\theta_m})(Q_\phi(0) \wedge (\forall x)(\bigwedge_{i=1}^{m} \mathrm{t}_s(\theta_i, x)))$, where $\mathrm{t}_s(\theta_i)$ is defined just like $\mathrm{t}(\theta_i)$, replacing the $\leftrightarrow$ by $\rightarrow$. The following theorem asserts the correctness of the *Sloppy* variation.

Theorem 3.2. *Let ϕ be an LTL$_f$ formula in NNF, ρ be a finite trance. Then $\rho \models \phi$ iff $\mathcal{I}_\rho \models \mathsf{mso}_s(\phi)$.*

Sloppy variation cannot be applied to LTL$_f$ formulas in BNF since the $\leftrightarrow$ constraint defined in function $\mathrm{t}(\theta_i)$ is needed only to handle negation correctly. BNF requires a general handling of negation. For LTL$_f$ formulas in NNF, negation is applied only to atomic formulas such that handled implicitly by the base case $\rho, x \models p \leftrightarrow \rho, x \not\models \neg p$. Therefore, translating LTL$_f$ formulas in NNF does not require the $\leftrightarrow$ constraint. For example, consider LTL$_f$ formula $\phi = \neg Fa$ (in BNF), where a is an atom. The corresponding BNF-*Sloppy* variation gives MSO formula $(\exists Q_{\neg Fa})(\exists Q_{Fa})(Q_{\neg Fa}(0) \wedge ((\forall x)((Q_{\neg Fa}(x) \rightarrow \neg Q_{Fa}(x)) \wedge (Q_{Fa}(x) \rightarrow (Q_a(x) \vee ((x \neq last) \wedge Q_{Fa}(x+1)))))))$ via $\mathsf{mso}_s(\phi)$. Consider finite trace $\rho = (a = 0), (a = 1)$, $\rho \models \phi$ iff $\rho \models \mathsf{mso}_s(\phi)$ does not hold since $\rho \not\models \neg Fa$. This happens because $\neg Fa$ requires $(Q_{\neg Fa}(x) \leftrightarrow \neg Q_{Fa}(x))$ as Fa is an non-atomic subformula. Therefore, *Sloppy* variation can only be applied to LTL$_f$ formulas in NNF.

Variable Form So far, we introduced the *Full* variation method, in which each non-atomic subformula in $cl(\phi)$ involves a monadic predicate. We now introduce *Lean* variation, a new variable form, aiming at decreasing the number of quantified monadic predicates. Less quantifiers on monadic predicates could benefit symbolic DFA construction a lot since quantifier elimination in MONA takes heavy cost. The key idea of *Lean* variation is introducing monadic predicates only for atomic subformulas and non-atomic subformulas of the form $\phi_j R \theta_k$ or $\phi_j U \theta_k$ (named as R- or U-subformula respectively).

For non-atomic subformulas that are not U- or R- subformulas, we can construct *second-order terms* using already defined monadic predicates to capture the semantics of them. Function $\mathsf{lean}(\theta_i)$ is defined to give such second-order terms. Intuitively speaking, $\mathsf{lean}(\theta_i)$ indicates the same positions where θ_i is true as Q_{θ_i} does, instead of having Q_{θ_i} explicitly. We use built-in second-order operators in MONA to simplify the definition of $\mathsf{lean}(\theta_i)$. ALIVE is defined to collect all instances along the finite trace. MONA also allows to apply set union, intersection, and difference for second-order terms, as well as the -1 operation (which shifts a monadic predicate backwards by one position). $\mathsf{lean}(\theta_i)$ is defined over the structure of θ_i as following:

- If $\theta_i = (\neg\theta_j)$, then $\mathsf{lean}(\theta_i) = (\mathsf{ALIVE} \backslash \mathsf{lean}(\theta_j))$
- If $\theta_i = (\theta_j \wedge \theta_k)$, then $\mathsf{lean}(\theta_i) = (\mathsf{lean}(\theta_j)\ \mathsf{inter}\ \mathsf{lean}(\theta_k))$

- If $\theta_i = (\theta_j \vee \theta_k)$, then $\text{lean}(\theta_i) = (\text{lean}(\theta_j)$ union $\text{lean}(\theta_k))$
- If $\theta_i = (X\theta_j)$, then $\text{lean}(\theta_i) = ((\text{lean}(\theta_j) - 1) \backslash \{last\})$
- If $\theta_i = (N\theta_j)$, then $\text{lean}(\theta_i) = ((\text{lean}(\theta_j) - 1)$ union $\{last\})$
- If $\theta_i = (\theta_j U \theta_k)$ or $\theta_i = (\theta_j R \theta_k)$, then $\text{lean}(\theta_i) = Q_{\theta_a}$, where Q_{θ_a} is the corresponding monadic predicate.

The following lemma ensures that $\text{lean}(\theta_i)$ keeps the interpretation of each non-atomic subformula $\theta_i \in cl(\phi)$.

Lemma 3.3. *Let ϕ be an LTL$_f$ formula, ρ be a finite trace. Then $\rho, x \models \theta_i$ iff $\text{lean}(\theta_i)(x)$, where x is the position in ρ.*

Finally, we define *Lean* variation based on function $\text{lean}(\phi)$. *Lean* variation $\text{mso}_\lambda(\phi)$ returns MSO formula $(\exists Q_{\theta_1}) \ldots (\exists Q_{\theta_n})$ $(\text{lean}(\phi)(0) \wedge ((\forall x)(\bigwedge_{a=1}^{n} \text{t}_\lambda(\theta_a, x))))$, where n is the number of U- and R- subformulas $\theta_a \in cl(\phi)$, and $\text{t}_\lambda(\theta_a, x)$ is defined as follows: if $\theta_a = (\theta_j U \theta_k)$, then $\text{t}_\lambda(\theta_a, x) = (Q_{\theta_a}(x) \leftrightarrow (\text{lean}(\theta_k)(x) \vee ((x \neq last) \wedge \text{lean}(\theta_j)(x) \wedge Q_{\theta_a}(x+1))))$; if $\theta_a = (\theta_j R \theta_k)$, then $\text{t}_\lambda(\theta_a, x) = (Q_{\theta_a}(x) \leftrightarrow (\text{lean}(\theta_k)(x) \wedge ((x = last) \vee \text{lean}(\theta_j)(x) \vee Q_{\theta_a}(x+1))))$. Then the following theorem guarantees the correctness of *Lean* variation.

Theorem 3.4. *Let ϕ be an LTL$_f$ formula, ρ be a finite trace. Then $\rho \models \phi$ iff $\mathcal{I}_\rho \models \text{mso}_\lambda(\phi)$.*

4 Experimental Evaluation

Having defined MSO encoding with various variations that allow potential optimization, we now evaluate the performance of them by comparing against FOL encoding. We implemented all parsers for LTL$_f$ formulas using C++. The entire framework consists of two steps: the encoding and the symbolic DFA construction. The parser first accepts an LTL$_f$ formula as input, then translates to a corresponding FOL or MSO formula. We feed this formula to MONA [10] to construct the symbolic DFA.

4.1 Experimental Methodology

Benchmarks We performed the experiment in the context of LTL$_f$-to-DFA, thus only satisfiable but not valid formulas are interesting. Therefore we first ran an LTL$_f$ satisfiability checker on LTL$_f$ formulas and their negations to filter the valid or unsatisfiable formulas. We collected 5690 formulas, which consist of two classes of benchmarks: 765 LTL$_f$-specific benchmarks, including 700 scalable LTL$_f$ pattern formulas from [7] and 65 randomly conjuncted common LTL$_f$ formulas from [3, 8, 20] in the style described in [15]; and 4925 LTL-as-LTL$_f$ formulas from [21, 22], since LTL formulas share the same syntax as LTL$_f$.

Experimental Setup As described in Section 3, there are 6 viable combinations (yielding 6 MSO encodings) of 3 independent variations: (1) the Normal Form (BNF or NNF); (2) the Constraint Form (*Fussy* or *Sloppy*); (3) the Variable Form (*Full* or *Lean*). To explore the comparison between FOL and MSO for LTL$_f$-to-DFA translation, for each LTL$_f$ formula, we performed LTL$_f$-to-DFA using all six MSO encodings described in Section 3 and two FOL encodings (BNF and NNF). We ran each formula for every encoding on a node within a high performance cluster. These nodes contain 12 processor cores at 2.2 GHz each with 8GB of RAM per core. Time out was set to be 1000 seconds. Cases that cannot generate the DFA within 1000 seconds generally fail even if the time limit is extended, since in these cases, MONA typically cannot handle the symbolic DFA construction.

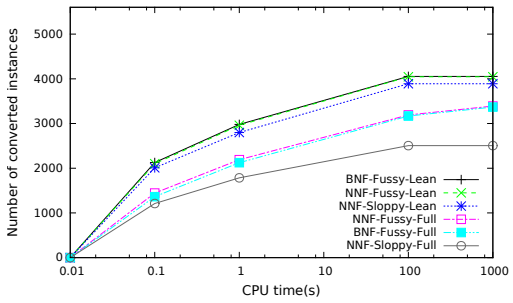

Figure 1. Comparison of 6 MSO encodings

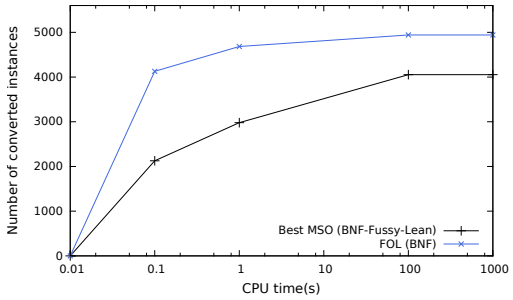

Figure 2. Comparison of FOL and MSO encodings

Correctness The correctness of the implementation of different encodings was evaluated by comparing the DFAs in terms of the number of states and transitions generated from each encoding. No inconsistencies were discovered.

Best FOL We first compared the impact on performance of the two LTL$_f$ normal forms (BNF and NNF) of FOL encoding. It turns out that the normal form does not have a measurable impact on the performance of the first-order encoding. Since FOL-BNF encoding performs slightly better than FOL-NNF, the best FOL encoding refers to FOL-BNF.

4.2 Experimental Results

The experiments were divided into two parts and resulted in two major findings. First we explored the benefits of various variations and showed that the most effective one is the *Lean* variation. Then we compared the best performing MSO encoding against the FOL encoding and showed that FOL encoding, unexpectedly, outperforms the MSO encoding. We report as follows.

Lean variation is more effective for MSO encoding. Figure 1 presents the number of converted instances of each

MSO encoding, where the upper three are all applied with *Lean* variation and the lower ones are with *Full* variation. The choice of BNF or NNF did not have a major impact, and neither did the choice of *Fussy* or *Sloppy*. The one variation that was particularly effective is that of *Lean*. The best-performing MSO encoding was enabled by combination BNF-*Fussy*-*Lean*.

FOL encoding dominates MSO encodings. As presented in Figure 2, FOL encoding shows its superiority over MSO encodings. Thus, the use of second-order logic, even under sophisticated optimization, did not prove its value in terms of performance. This is discussed extensively in next section.

5 Discussion

In this paper we presented the first encodings of LTL$_f$ to symbolic automata based on MSO, and the first comparison of FOL against MSO in the context of LTL$_f$-to-automata translation. We showed that MSO encoding allows a significantly simpler quantificational structure, which requires only a block of existential second-order quantifiers, followed by a single universal first-order quantifier, while FOL encoding involves an arbitrary alternation of quantifiers. Nevertheless, empirical evaluations showed that first-order encoding, in general, outperforms the second-order encodings. This finding contradicts our initial intuition.

One reason for this contradiction might be that MONA is an "aggressive minimizer": after each quantifier elimination, MONA re-minimizes the DFA under construction. Thus, the fact that the MSO encoding starts with a block of existential second-order quantifiers offers no computational advantage, as MONA eliminates the second-order quantifiers one by one, performing computationally heavy minimization after each quantifier. A possible improvement to MONA would enable it to eliminate a whole *block* of quantifiers of the same type (existential or universal) in one operation, involving only one minimization. We conjecture that with such improvement to MONA, the MSO encoding would outperform the FOL encoding. We leave this to future work.

Beyond the possibility of performance gained via second-order encodings, another motivation for studying such encodings is their greater expressivity. The fact that LTL$_f$ has equivalent expressiveness with FOL [11] tells us that its expressiveness is limited. For this reason it is advocated in [4] to use *Linear Dynamic Logic* (LDL$_f$) to specify ongoing behavior. LDL$_f$ is expressively equivalent to MSO, which is more expressive than FOL. Thus, automata-theoretic reasoning for LDL$_f$, for example, reactive synthesis [5], cannot be done via first-order encoding and requires second-order encoding. Similarly, synthesis of LTL$_f$ with incomplete information requires the usage of second-order encoding [6]. We leave this too to future research.

Acknowledgments. Work supported in part by NSF grants CCF-1319459 and IIS-1527668, and by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering", NSFC Projects No. 61572197 and No. 61632005, MOST NKTSP Project 2015BAG19B02. Special thanks to Dror Fried for useful discussions.

References

[1] Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. 2007. Interactive presentation: Automatic hardware synthesis from specifications: a case study. In *DATE*. 1188–1193.

[2] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. 2012. Acacia+, a Tool for LTL Synthesis. In *CAV*.

[3] Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. 2014. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In *AAAI*. 1027–1033.

[4] G. De Giacomo and Moshe Y. Vardi. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*. 854–860.

[5] Giuseppe De Giacomo and Moshe Y. Vardi. 2015. Synthesis for LTL and LDL on Finite Traces. In *IJCAI*. 1558–1564.

[6] Giuseppe De Giacomo and Moshe Y. Vardi. 2016. LTL$_f$ and LDL$_f$ Synthesis under Partial Observability. In *IJCAI*. 1044–1050.

[7] Claudio Di Ciccio, Fabrizio Maria Maggi, and Jan Mendling. 2016. Efficient discovery of Target-Branched Declare constraints. *Inf. Syst.* 56 (2016), 258–283.

[8] Claudio Di Ciccio and Massimo Mecella. 2015. On the Discovery of Declarative Control Flows for Artful Processes. *ACM Trans. Management Inf. Syst.* 5, 4 (2015), 24:1–24:37.

[9] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. 2016. Spot 2.0 — A Framework for LTL and ω-automata Manipulation. In *ATVA*.

[10] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. 1995. Mona: Monadic Second-Order Logic in Practice. In *TACAS*. 89–110.

[11] J.A.W. Kamp. 1968. *Tense Logic and the Theory of Order*. Ph.D. Dissertation. UCLA.

[12] Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. 2000. MONA Implementation Secrets. In *CIAA*. 182–194.

[13] Orna Kupferman and Moshe Y. Vardi. 2001. Model Checking of Safety Properties. *Formal Methods in System Design* 19, 3 (2001), 291–314.

[14] O. Kupferman and Moshe Y. Vardi. 2005. Safraless Decision Procedures. In *FOCS*. 531–540.

[15] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. 2013. LTL Satisfiability Checking Revisited. In *TIME*. 91–98.

[16] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. 1985. The Glory of the Past. In *Logics of Programs*. 196–218.

[17] G. Pan, U. Sattler, and Moshe Y. Vardi. 2003. Optimizing a BDD-Based Modal Solver. In *Proc. 19th Int'l Conf. on Automated Deduction*. 75–89.

[18] Amir Pnueli. 1977. The temporal logic of programs. 46–57.

[19] A. Pnueli and R. Rosner. 1989. On the Synthesis of a Reactive Module. In *POPL*. 179–190.

[20] Johannes Prescher, Claudio Di Ciccio, and Jan Mendling. 2014. From Declarative Processes to Imperative Models. In *SIMPDA 2014*. 162–173.

[21] Kristin Y. Rozier and Moshe Y. Vardi. 2007. LTL Satisfiability Checking. In *Model Checking Software, 14th International SPIN Workshop*. 149–167.

[22] Kristin Y. Rozier and Moshe Y. Vardi. 2011. A Multi-encoding Approach for LTL Symbolic Satisfiability Checking. In *FM*. 417–431.

[23] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. 2017. A Symbolic Approach to Safety LTL Synthesis. In *HVC*. 147–162.

[24] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. 2017. Symbolic LTL$_f$ Synthesis. In *IJCAI*. 1362–1369.