First-Order vs. Second-Order Encodings for LTL$_f$-to-Automata Translation

Shufang Zhu[1], Geguang Pu[1(✉)], and Moshe Y. Vardi[2]

[1] East China Normal University, Shanghai, China
ggpu@sei.ecnu.edu.cn
[2] Rice University, Houston, TX, 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 powerful tool for symbolic, BDD-based, 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 get 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 encoding, which has significantly simpler quantificational structure, can outperform first-order encoding remained open.

In this paper we address this challenge and study second-order encodings for LTL$_f$ formulas. We first introduce a specific MSO encoding that captures the semantics of LTL$_f$ in a natural way and prove its correctness. We then explore is a *Compact* MSO encoding, which benefits from automata-theoretic minimization, thus suggesting a possible practical advantage. To that end, we propose a formalization of symbolic DFA in second-order logic, thus developing a novel connection between BDDs and MSO. We then show by empirical evaluations that the first-order encoding does perform better than both second-order encodings. The conclusion is that first-order encoding is a better choice than second-order encoding in LTL$_f$-to-Automata translation.

1 Introduction

Synthesis from temporal specifications [23] is a fundamental problem in Artificial Intelligence and Computer Science [8]. A popular specification is Linear Temporal Logic (LTL) [24]. The standard approach to solving LTL synthesis requires, however, determinization of automata on *infinite* words and solving *parity games*, both challenging algorithmic problems [17]. 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 [6,8]. 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 [24].

T. V. Gopal and J. Watada (Eds.): TAMC 2019, LNCS 11436, pp. 684–705, 2019.
https://doi.org/10.1007/978-3-030-14812-6_43

Such interpretation allows the executions being arbitrarily long, but not infinite, and is adequate for finite-horizon planning problems. 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) [8]. Another application, as shown in [30], 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 [18,19]). This approach has been implemented in [31] for LTL$_f$ synthesis and in [30] 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 these approaches is a translation of LTL$_f$ to *symbolic* Deterministic Finite Automata (DFA) [30,31]. In fact, translating LTL$_f$ formula to DFA has other algorithmic applications as well. For example, in dealing with safety properties, which are arguably the most used temporal specifications in real-world systems [18]. As shown in [28], model checking of safety properties can benefit from using deterministic rather than nondeterminisic automata. Moreover, in runtime verification for safety properties, we need to generate monitors, a type of which are, in essence, deterministic automata [29]. In [28,29], the translation to deterministic automata is explicit, but symbolic DFAs can be useful also in model checking and monitor generation, because they can be much more compact than explicit DFAs, cf. [31].

The method used in [30,31] for the 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 [13], a powerful tool, for symbolic DFA construction from logical specifications. This approach was shown to outperform explicit tools such as SPOT [12], but encodings other than the first-order one have not yet been studied. This leads us here to study second-order translations of LTL$_f$, where we use Monadic Second Order (MSO) logic of one successor over finite words (called M2L-STR in [16]). Indeed, one possible advantage of using MSO is the simpler quantificational structure that the second-order encoding requires, which is a sequence of existential monadic second-order quantifiers followed by a single universal first-order quantifier. Moreover, instead of the syntax-driven translation of first-order encoding of LTL$_f$ to FOL, the second-order encoding employs a semantics-driven translation, which allows more space for optimization. The natural question arises whether second-order encoding outperforms first-order encoding.

To answer this question, we study here second-order encodings of LTL$_f$ formulas. We start by introducing a specific second-order encoding called MSO encoding that relies on having a second-order variable for each temporal operator appearing in the LTL$_f$ formula and proving the correctness. Such MSO encoding captures the semantics of LTL$_f$ in a natural way and is linear in the size of the formula. We then introduce a so called *Compact* MSO encoding, which captures the tight connection between LTL$_f$ and DFAs. We leverage the fact that

while the translation from LTL_f to DFA is doubly exponential [18], there is an exponential translation from *Past* LTL_f to DFA (a consequence of [5,6]). Given an LTL_f formula ϕ, we first construct a DFA that accepts exactly the reverse language satisfying $models(\phi)$ via *Past* LTL_f. We then encode this DFA using second-order logic and "invert" it to get a second-order formulation for the original LTL_f formula. Applying this approach directly, however, would yield an MSO formula with an exponential (in terms of the original LTL_f formula) number of quantified monadic predicates. To get a more compact formulation we can benefit from the fact that the DFA obtained by MONA from the *Past* LTL_f formula is symbolic, expressed by binary decision diagrams (BDDs) [14]. We show how we can obtain a Compact MSO encoding directly from these BDDs. In addition, we present in this paper the first evaluation of the spectrum of encodings for LTL_f-to-automata from first-order to second-order.

To perform an empirical evaluation of the comparison between first-order encoding and second-order encoding of LTL_f, we first provide a broad investigation of different optimizations of both encodings. Due to the syntax-driven translation of FOL encoding, there is limit potential for optimization such that we are only able to apply different normal forms to LTL_f formulas, which are Boolean Normal Form (BNF) and Negation Normal Form (NNF). The semantics-driven translation of second-order encoding, however, enables more potential for optimization than the FOL encoding. In particular, we study the following optimizations introduced in [21,22]: in the variable form, where a *Lean* encoding introduces fewer variables than the standard *Full* encoding; and in the constraint form, where a *Sloppy* encoding allows less tight constraints than the standard *Fussy* encoding. The main result of our empirical evaluations is the superiority of the first-order encoding as a way to get MONA to generate a symbolic DFA, which answers the question of whether second-order outperforms first-order for LTL_f-to-automata translation.

The paper is organized as follows. In Sect. 2 we provide preliminaries and notations. Section 3 introduces MSO encoding and proves the correctness. Section 4 describes a more compact second-order encoding, called Compact MSO encoding and proves the correctness. Empirical evaluation results of different encodings and different optimizations are presented in Sect. 5. Finally, Sect. 6 offers concluding remarks.

2 Preliminaries

2.1 LTL_f Basics

Linear Temporal Logic over *finite traces* (LTL_f) has the same syntax as LTL [6]. Given a set $\mathcal{P}$ of propositions, 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. Finally, we have standard boolean abbreviations, such as $\vee$ (or) and $\rightarrow$ (implies).

Elements $p \in \mathcal{P}$ are *atoms*. A literal l can be an atom or the negation of an atom. A *trace* $\rho = \rho[0], \rho[1], \ldots$ is a sequence of propositional assignments, where $\rho[x] \in 2^{\mathcal{P}}$ $(x \geq 0)$ is the x-th point of ρ. Intuitively, $\rho[x]$ is the set of propositions that are *true* at instant x. Additionally, $|\rho|$ represents the length of ρ. The trace ρ is an *infinite* trace if $|\rho| = \infty$ and $\rho \in (2^{\mathcal{P}})^{\omega}$; otherwise ρ is *finite*, and $\rho \in (2^{\mathcal{P}})^{*}$. LTL$_f$ formulas are interpreted over finite traces. Given a finite trace ρ and an LTL$_f$ formula ϕ, we inductively define when ϕ is *true* for ρ at point x $(0 \leq x < |\rho|)$, written $\rho, x \models \phi$, as follows:

- $\rho, x \models \top$ and $\rho, x \not\models \bot$;
- $\rho, x \models p$ iff $p \in \rho[x]$;
- $\rho, x \models \neg\phi$ iff $\rho, x \not\models \phi$;
- $\rho, x \models \phi_1 \wedge \phi_2$, iff $\rho, x \models \phi_1$ and $\rho, x \models \phi_2$;
- $\rho, x \models X\phi$, iff $x + 1 < |\rho|$ and $\rho, x + 1 \models \phi$;
- $\rho, x \models \phi_1 U \phi_2$, iff there exists y such that $x \leq y < |\rho|$ and $\rho, y \models \phi_2$, and for all z, $x \leq z < y$, we have $\rho, z \models \phi_1$.

An LTL$_f$ formula ϕ is *true* in ρ, denoted by $\rho \models \phi$, when $\rho, 0 \models \phi$. Every LTL$_f$ formula can be written in Boolean Normal Form (BNF) or Negation Normal Form (NNF) [27]. 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 automaton* [31], where a boolean formula is used to represent the transition function of a Deterministic Finite Automaton (DFA). A symbolic deterministic finite automaton (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 $s \in F$.

We can represent the symbolic transition function η by an indexed family $\eta_q : 2^{\mathcal{X}} \times 2^{\mathcal{P}} \rightarrow \{0, 1\}$ for $x_q \in \mathcal{X}$, which means that η_q can be represented by a binary decision diagram (BDD) [14] over $\mathcal{X} \cup \mathcal{P}$. Therefore, the symbolic DFA can be represented by a sequence of BDDs, each of which corresponding to a state variable.

The MONA tool [13] is an efficient implementation for translating FOL and MSO formulas over finite words into minimized symbolic deterministic automata. MONA represents symbolic deterministic automata by means of *Shared Multi-terminal BDDs* (ShMTBDDs) [3,20]. The symbolic LTL$_f$ synthesis framework of [31] requires standard BDD representation by means of symbolic DFAs as defined above. The transformation from ShMTBDD to BDD is described in [31].

2.3 FOL Encoding of LTL$_f$

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 [6]. We first restrict our interest to *monadic structure*. Consider a finite trace $\rho = \rho[0]\rho[1]\cdots\rho[e]$, the corresponding *monadic structure* $\mathcal{I}_\rho = (\Delta^{\mathcal{I}}, <, \cdot^{\mathcal{I}})$ describes ρ as follows. $\Delta^{\mathcal{I}} = \{0, 1, 2, \cdots, last\}$, where $last = e$ indicating the last point along the trace. The linear order $<$ is defined over $\Delta^{\mathcal{I}}$ in the standard way [16]. The notation $\cdot^{\mathcal{I}}$ indicates the set of monadic predicates that describe the atoms of $\mathcal{P}$, where the interpretation of each $p \in \mathcal{P}$ is $Q_p = \{x \ : \ p \in \rho[x]\}$. Intuitively, Q_p is interpreted as the set of positions where p is true in ρ. In the translation below, $\mathsf{fol}(\theta, x)$, where θ is an LTL$_f$ formula and x is a variable, is an FOL formula asserting the truth of θ at point x of the linear order. The translation uses the successor function $+1$, and the variable *last* that represents the maximal point in the linear order.

- $\mathsf{fol}(p, x) = (Q_p(x))$
- $\mathsf{fol}(\neg\phi, x) = (\neg\mathsf{fol}(\phi, x))$
- $\mathsf{fol}(\phi_1 \wedge \phi_2, x) = (\mathsf{fol}(\phi_1, x) \wedge \mathsf{fol}(\phi_2, x))$
- $\mathsf{fol}(\phi_1 \vee \phi_2, x) = (\mathsf{fol}(\phi_1, x) \vee \mathsf{fol}(\phi_2, x))$
- $\mathsf{fol}(X\phi, x) = ((\exists y)((y = x + 1) \wedge \mathsf{fol}(\phi, y)))$
- $\mathsf{fol}(N\phi, x) = ((x = last) \vee ((\exists y)((y = x + 1) \wedge \mathsf{fol}(\phi, y))))$
- $\mathsf{fol}(\phi_1 U \phi_2, x) = ((\exists y)((x \leq y \leq last) \wedge \mathsf{fol}(\phi_2, y) \wedge (\forall z)((x \leq z < y) \rightarrow \mathsf{fol}(\phi_1, z))))$
- $\mathsf{fol}(\phi_1 R \phi_2, x) = (((\exists y)((x \leq y \leq last) \wedge \mathsf{fol}(\phi_1, y) \wedge (\forall z)((x \leq z \leq y) \rightarrow \mathsf{fol}(\phi_2, z)))) \vee ((\forall z)((x \leq z \leq last) \rightarrow \mathsf{fol}(\phi_2, z))))$

For FOL variables, MONA provides a built-in operator $+1$ for successor computation. Moreover, we can use built-in procedures in MONA to represent the variable *last*. 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 1 ([15]). *Let ϕ be an LTL$_f$ formula and ρ be a finite trace. Then $\rho \models \phi$ iff $\mathcal{I}_\rho \models \mathsf{fol}(\phi, 0)$.*

3 MSO Encoding

First-order encoding was shown to perform well in the context of LTL$_f$-to-automata translation [30], but other encodings have not been studied. Specifically, the natural question of whether second-order (MSO) outperforms first-order

in the same context remained open. MSO is an extension of FOL that allows quantification over monadic predicates [16]. By applying a semantics-driven translation to LTL$_f$, we obtain an MSO encoding that has significantly simpler quantificational structure. This encoding essentially captures in MSO the standard encoding of temporal connectives, cf. [4]. Intuitively speaking, MSO encoding deals with LTL$_f$ formula by interpreting every operator with corresponding subformulas following the semantics of the operator. We now present MSO encoding that translates LTL$_f$ formula ϕ to MSO, which is then fed to MONA to produce a symbolic DFA.

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: $\mathsf{mso}(\phi) = (\exists Q_{\theta_1}) \cdots (\exists Q_{\theta_m})(Q_\phi(0) \wedge (\forall x)(\bigwedge_{i=1}^{m} \mathsf{t}(\theta_i, x))$, where x indicates the position along the finite linear order. Here $\mathsf{t}(\theta_i, x)$ 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 following the semantics of LTL$_f$. Therefore, $\mathsf{t}(\theta_i, x)$ is defined as follows:

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

Consider a finite trace ρ, the corresponding interpretation $\mathcal{I}_\rho$ of ρ is defined as in Sect. 2.3. The following theorem asserts the correctness of the MSO encoding.

Theorem 2. *Let ϕ be an LTL$_f$ formula, ρ be a finite trace. Then $\rho \models \phi$ iff $\mathcal{I}_\rho \models \mathsf{mso}(\phi)$.*

Proof. If ϕ is a propositional atom p, then $\mathsf{mso}(\phi) = Q_p(0)$. It is true that $\rho \models \phi$ iff $\mathcal{I}_\rho \models \mathsf{mso}(\phi)$. If ϕ is an nonatomic formula, we prove this theorem in two directions.

Suppose first that ρ satisfies ϕ. We expand the monadic structure $\mathcal{I}_\rho$ with interpretations for the existentially quantified monadic predicate symbols by setting Q_{θ_i}, the interpretation of subformula θ_i in $\mathcal{I}_\rho$, as the set collecting all

points of ρ satisfying θ_i, that is $Q_{\theta_i} = \{x : \rho, x \models \theta_i\}$. We also have $Q_p = \{x : \rho, x \models p\}$ and denote the expanded structure by $\mathcal{I}_\rho^{mso}$. By assumption, $Q_\phi(0)$ holds in $\mathcal{I}_\rho^{mso}$. It remains to prove that $\mathcal{I}_\rho^{mso} \models \forall x.\mathsf{t}(\theta_i, x)$, for each nonatomic subformula $\theta_i \in cl(\phi)$, which we prove via structural induction over θ_i.

- If $\theta_i = (\neg\theta_j)$, then $\mathsf{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow (\neg Q_{\theta_j}(x)))$. This holds, since $Q_{(\neg\theta_j)} = \{x : \rho, x \not\models \theta_j\}$ and $Q_{\theta_j} = \{x : \rho, x \models \theta_j\}$.
- If $\theta_i = (\theta_j \wedge \theta_k)$, then $\mathsf{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow (Q_{\theta_j}(x) \wedge Q_{\theta_k}(x)))$. This holds, since $Q_{(\theta_j \wedge \theta_k)} = \{x : \rho, x \models \theta_j$ and $\rho, x \models \theta_k\}$, $Q_{\theta_j} = \{x : \rho, x \models \theta_j\}$ and $Q_{\theta_k} = \{x : \rho, x \models \theta_k\}$.
- If $\theta_i = (\theta_j \vee \theta_k)$, then $\mathsf{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow (Q_{\theta_j}(x) \vee Q_{\theta_k}(x)))$. This holds, since $Q_{(\theta_j \vee \theta_k)} = \{x : \rho, x \models \theta_j$ or $\rho, x \models \theta_k\}$, $Q_{\theta_j} = \{x : \rho, x \models \theta_j\}$ and $Q_{\theta_k} = \{x : \rho, x \models \theta_k\}$.
- If $\theta_i = (X\theta_j)$, then $\mathsf{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow ((x \neq last) \wedge Q_{\theta_j}(x+1)))$. This holds, since $Q_{(X\theta_j)} = \{x : \rho, x \models (X\theta_j)\} = \{x : x \neq last$ and $\rho, x+1 \models \theta_j\}$, and $Q_{\theta_j} = \{x : \rho, x \models \theta_j\}$.
- If $\theta_i = (N\theta_j)$, then $\mathsf{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow ((x = last) \vee Q_{\theta_j}(x+1)))$. This holds, since $Q_{(N\theta_j)} = \{x : \rho, x \models (N\theta_j)\} = \{x : x = last$ or $\rho, x+1 \models \theta_j\}$, and $Q_{\theta_j} = \{x : \rho, x \models \theta_j\}$.
- If $\theta_i = (\theta_j U \theta_k)$, then $\mathsf{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow (Q_{\theta_k}(x) \vee ((x \neq last) \wedge Q_{\theta_j}(x) \wedge Q_{\theta_i}(x+1))))$. This holds, since $Q_{(\theta_j U \theta_k)} = \{x : \rho, x \models \theta_j U \theta_k\} = \{x : \rho, x \models \theta_k$ or $x \neq last$ with $\rho, x \models \theta_j$ also $\rho, x+1 \models \theta_i\}$, $Q_{\theta_j} = \{x : \rho, x \models \theta_j\}$, and $Q_{\theta_k} = \{x : \rho, x \models \theta_k\}$;
- If $\theta_i = (\theta_j R \theta_k)$, then $\mathsf{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow (Q_{\theta_k}(x) \wedge ((x = last) \vee Q_{\theta_j}(x) \vee Q_{\theta_i}(x+1))))$. This holds, since $Q_{(\theta_j R \theta_k)} = \{x : \rho, x \models \theta_j R \theta_k\} = \{x : \rho, x \models \theta_k$ with $x = last$ or $\rho, x \models \theta_j$ or $\rho, x+1 \models \theta_i\}$, $Q_{\theta_j} = \{x : \rho, x \models \theta_j\}$, and $Q_{\theta_k} = \{x : \rho, x \models \theta_k\}$.

Assume now that $\mathcal{I}_\rho \models \mathsf{mso}(\phi)$. This means that there is an expansion of $\mathcal{I}_\rho$ with monadic interpretations Q_{θ_i} for each nonatomic subformula $\theta_i \in cl(\phi)$ such that this expanded structure $\mathcal{I}_\rho^{mso} \models (Q_\phi(0) \wedge ((\forall x) \bigwedge_{i=1}^m \mathsf{t}(\theta_i, x)))$. We now prove by induction on ϕ that if $x \in Q_\phi$, then $\rho, x \models \phi$ such that $Q_\phi(0)$ indicates that $\rho, 0 \models \phi$.

- If $\phi = (\neg\theta_j)$, then $\mathsf{t}(\phi, x) = (Q_\phi(x) \leftrightarrow (x \notin Q_{\theta_j}))$. Since $\mathsf{t}(\phi)$ holds at every point x of $\mathcal{I}_\rho^{mso}$, it holds that $x \in Q_\phi$ iff $x \notin Q_{\theta_j}$. It follows by induction that $\rho, x \not\models \theta_j$. Thus, $\rho, x \models \phi$.
- If $\phi = (\theta_j \wedge \theta_k)$, then $\mathsf{t}(\phi, x) = (Q_\phi(x) \leftrightarrow (Q_{\theta_j}(x) \wedge Q_{\theta_k}(x)))$. Since $\mathsf{t}(\phi)$ holds at every point x of $\mathcal{I}_\rho^{mso}$, it follows that $x \in Q_\phi$ iff $x \in Q_{\theta_j}$ and $x \in Q_{\theta_k}$. It follows by induction that $\rho, x \models \theta_j$ and $\rho, x \models \theta_k$. Thus, $\rho, x \models \phi$.
- If $\phi = (\theta_j \vee \theta_k)$, then $\mathsf{t}(\phi, x) = (Q_\phi(x) \leftrightarrow (Q_{\theta_j}(x) \vee Q_{\theta_k}(x)))$. Since $\mathsf{t}(\phi)$ holds at every point x of $\mathcal{I}_\rho^{mso}$, it follows that $x \in Q_\phi$ iff $x \in Q_{\theta_j}$ or $x \in Q_{\theta_k}$. It follows by induction that $\rho, x \models \theta_j$ or $\rho, x \models \theta_k$. Thus, $\rho, x \models \phi$.
- If $\phi = (X\theta_j)$, then $\mathsf{t}(\phi, x) = (Q_\phi(x) \leftrightarrow ((x \neq last) \wedge Q_{\theta_j}(x+1)))$. Since $t(\phi)$ holds at every point x of $\mathcal{I}_\rho^{mso}$, it follows that $x \in Q_\phi$ iff $x \neq last$ and $x+1 \in Q_{\theta_j}$. It follows by induction that $x \neq last$ and $\rho, x \models \theta_j$. Thus, $\rho, x \models \phi$.

- If $\phi = (N\theta_j)$, then $\mathtt{t}(\phi, x) = (Q_\phi(x) \leftrightarrow ((x = last) \vee Q_{\theta_j}(x+1)))$. Since $\mathtt{t}(\phi)$ holds at every point x of $\mathcal{I}_\rho^{mso}$, it follows that $x \in Q_\phi$ iff $x = last$ or $x + 1 \in Q_{\theta_j}$. It follows by induction that $x = last$ or $\rho, x \models \theta_j$. Thus, $\rho, x \models \phi$.
- If $\phi = (\theta_j U \theta_k)$, then $\mathtt{t}(\phi, x) = (Q_\phi(x) \leftrightarrow (Q_{\theta_k}(x) \vee ((x \neq last) \wedge Q_{\theta_j}(x) \wedge Q_\phi(x+1))))$. Since $\mathtt{t}(\phi)$ holds at every point x of $\mathcal{I}_\rho^{mso}$, it follows that $x \in Q_\phi$ iff $x \in Q_{\theta_k}$ or $x \neq last$ with $x \in Q_{\theta_j}$ also $x + 1 \in Q_\phi$. Thus, $\rho, x \models \phi$.
- If $\phi = (\theta_j R \theta_k)$, then $\mathtt{t}(\phi, x) = (Q_\phi(x) \leftrightarrow (Q_{\theta_k}(x) \wedge ((x = last) \vee Q_{\theta_j}(x) \vee Q_\phi(x+1))))$. Since $\mathtt{t}(\phi)$ holds at every point x of $\mathcal{I}_\rho^{mso}$, it follows that $x \in Q_\phi$ iff $x \in Q_{\theta_k}$ with $x = last$ or $x \in Q_{\theta_j}$ or $x + 1 \in Q_\phi$. Thus, $\rho, x \models \phi$. □

4 Compact MSO Encoding

The MSO encoding described in Sect. 3 is closely related to the translation of LTL$_f$ to alternating automata [6], with each automaton state corresponding to a monadic predicate. The construction, however, is subject only to syntactic minimization. Can we optimize this encoding using *automata-theoretic minimization?* In fact, MONA itself applies automata-theoretic minimization. Can we use MONA to produce a more efficient encoding for MONA?

The key observation is that MONA can produce a compact symbolic representation of a non-deterministic automaton (NFA) representing a given LTL$_f$ formula, and we can use this symbolic NFA to create a more compact MSO encoding for LTL$_f$. This is based on the observation that while the translation from LTL$_f$ to DFA is 2-EXP [18], the translation from *past* LTL$_f$ to DFA is 1-EXP, as explained below. We proceed as follows: (1) Reverse a given LTL$_f$ formula ϕ to *Past* LTL$_f$ formula ϕ^R; (2) Use MONA to construct the DFA of ϕ^R, the reverse of which is an NFA, that accepts exactly the reverse language of the words satisfying $models(\phi)$; (3) Express this symbolic DFA in second-order logic and "invert" it to get $\mathcal{D}_\phi$, the corresponding DFA of ϕ.

The crux of this approach, which follows from [5,6], is that the DFA corresponding to the reverse language of an LTL$_f$ formula ϕ of length n has only 2^n states. The reverse of this latter DFA is an NFA for ϕ. We now elaborate on these steps.

4.1 LTL$_f$ to PLTL$_f$

Past Linear Temporal Logic over finite traces, i.e. PLTL$_f$, has the same syntax as PLTL over infinite traces introduced in [24]. Given a set of propositions $\mathcal{P}$, the grammar of PLTL$_f$ is given by:

$$\psi ::= \top \mid \bot \mid p \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid Y\psi \mid \psi_1 S \psi_2$$

Given a finite trace ρ and a PLTL$_f$ formula ψ, we inductively define when ψ is *true* for ρ at step x ($0 \leq x < |\rho|$), written by $\rho, x \models \psi$, as follows:

- $\rho, x \models \top$ and $\rho, x \not\models \bot$;
- $\rho, x \models p$ iff $p \in \rho[x]$;
- $\rho, x \models \neg\psi$ iff $\rho, x \not\models \psi$;
- $\rho, x \models \psi_1 \wedge \psi_2$, iff $\rho, x \models \psi_1$ and $\rho, x \models \psi_2$;
- $\rho, x \models Y\psi$, iff $x - 1 \geq 0$ and $\rho, x - 1 \models \psi$;
- $\rho, x \models \psi_1 S \psi_2$, iff there exists y such that $0 \leq y \leq x$ and $\rho, y \models \psi_2$, and for all z, $y < z \leq x$, we have $\rho, z \models \psi_1$.

A PLTL$_f$ formula ψ is *true* in ρ, denoted by $\rho \models \psi$, if and only if $\rho, |\rho| - 1 \models \psi$. To reverse an LTL$_f$ formula ϕ, we replace each temporal operator in ϕ with the corresponding *past* operator of PLTL$_f$ thus getting ϕ^R. X(Next) and U(Until) correspond to Y(Before) and S(Since) respectively.

We define $\rho^R = \rho[|\rho| - 1], \rho[|\rho| - 2], \ldots, \rho[1], \rho[0]$ to be the reverse of ρ. Moreover, given language $\mathcal{L}$, we denote the reverse of $\mathcal{L}$ by $\mathcal{L}^R$ such that $\mathcal{L}^R$ collects all reversed sequences in $\mathcal{L}$. Formally speaking, $\mathcal{L}^R = \{\rho^R : \rho \in \mathcal{L}\}$. The following theorem shows that PLTL$_f$ formula ϕ^R accepts exactly the reverse language satisfying ϕ.

Theorem 3. *Let $\mathcal{L}(\phi)$ be the language of* LTL$_f$ *formula ϕ and $\mathcal{L}^R(\phi)$ be the reverse language, then $\mathcal{L}(\phi^R) = \mathcal{L}^R(\phi)$.*

Proof. $\mathcal{L}(\phi^R) = \mathcal{L}^R(\phi)$ iff for an arbitrary sequence $\rho \in \mathcal{L}(\phi)$ such that $\rho \models \phi$, it is true that $\rho^R \models \phi^R$. We prove the theorem by the induction over the structure of ϕ. *last* is used to denote the last instance such that $last = |\rho| - 1$.

- Basically, if $\phi = p$ is an atom, then $\phi^R = p$, $\rho \models \phi$ iff $p \in \rho[0]$ such that $p \in \rho^R[last]$. Therefore, $\rho^R \models \phi^R$;
- If $\phi = \neg\phi_1$, then $\phi^R = \neg\phi_1^R$, $\rho \models \neg\phi_1$ iff $\rho \not\models \phi_1$, such that by induction hypothesis $\rho^R \not\models \phi_1^R$ holds, therefore $\rho^R \models \phi^R$ is true;
- If $\phi = \phi_1 \wedge \phi_2$, then $\phi^R = \phi_1^R \wedge \phi_2^R$, $\rho \models \phi$ iff ρ satisfies both ϕ_1 and ϕ_2. By induction hypothesis $\rho^R \models \phi_1^R$ and $\rho^R \models \phi_2^R$ hold, therefore $\rho^R \models \phi^R$ is true;
- If $\phi = X\phi_1$, $\phi^R = Y\phi_1^R$, $\rho \models \phi$ iff suffix ρ' is sequence$\rho[1], \rho[2], \ldots, \rho[last]$ and $\rho' \models \phi_1$. By induction hypothesis, $\rho'^R \models \phi_1^R$ holds, in which case $\rho^R, last - 1 \models \phi_1^R$ is true, therefore $\rho^R \models \phi^R$ holds.
- If $\phi = \phi_1 U \phi_2$, $\rho \models \phi$ iff there exists y such that y ($0 \leq y \leq last$), suffix $\rho' = \rho[y], \rho[y+1], \ldots, \rho[last]$ satisfies ϕ_2. Also for all z such that z ($0 \leq z < y$), $\rho'' = \rho[z], \rho[z+1], \ldots, \rho[last]$ satisfies ϕ_1. By induction hypothesis, $\rho'^R \models \phi_1^R$ and $\rho''^R \models \phi_2^R$ hold, therefore we have $\rho^R, last - y \models \phi_2^R$ and $\forall z. last - y < z \leq last, \rho^R, z \models \phi_1^R$ hold such that $\rho^R \models \phi^R$. The proof is done. □

4.2 PLTL$_f$ to DFA

The DFA construction from PLTL$_f$ formulas relies on MONA as well. Given PLTL$_f$ formula ψ, we are able to translate ψ to FOL formula as input of MONA, which returns the DFA. For PLTL$_f$ formula ψ over $\mathcal{P}$, we construct the corresponding FOL formula with respect to point x by a function $\mathsf{fol_p}(\psi, x)$ asserting the truth of ψ at x. Detailed translation of PLTL$_f$ to FOL is defined below. The translation uses the predecessor function -1, and the predicate *last* referring to the last point along the finite trace.

- $\mathsf{fol_p}(p, x) = (Q_p(x))$
- $\mathsf{fol_p}(\neg\psi, x) = (\neg\mathsf{fol_p}(\psi, x))$
- $\mathsf{fol_p}(\psi_1 \wedge \psi_2, x) = (\mathsf{fol_p}(\psi_1, x) \wedge \mathsf{fol_p}(\psi_2, x))$
- $\mathsf{fol_p}(Y\psi, x) = ((\exists y)((y = x - 1) \wedge (y \geq 0) \wedge \mathsf{fol_p}(\psi, y)))$
- $\mathsf{fol_p}(\psi_1 S \psi_2, x) = ((\exists y)((0 \leq y \leq x) \wedge \mathsf{fol_p}(\psi_2, y) \wedge (\forall z)((y < z \leq x) \rightarrow \mathsf{fol_p}(\psi_1, z))))$

Consider a finite trace ρ, the corresponding interpretation $\mathcal{I}_\rho$ is defined as in Sect. 2.3. The following theorem guarantees the correctness of the above translation.

Theorem 4. [15] *Let ψ be a* PLTL$_f$ *formula, ρ be a finite trace. Then $\rho \models \psi$ iff $\mathcal{I}_\rho \models \mathsf{fol_p}(\psi, last)$, where $last = |\rho| - 1$.*

Proof. We prove the theorem by the induction over the structure of ψ.

- Basically, if $\psi = p$ is an atom, $\rho \models \psi$ iff $p \in \rho[last]$. By the definition of $\mathcal{I}$, we have that $last \in Q_p$. Therefore, $\rho \models \psi$ iff $\mathcal{I}_\rho \models \mathsf{fol_p}(p, last)$ holds;
- If $\psi = \neg\psi$, $\rho \models \neg\psi$ iff $\rho \not\models \psi$. By induction hypothesis it is true that $\mathcal{I}_\rho \not\models \mathsf{fol_p}(\psi, last)$, therefore $\mathcal{I}_\rho \models \mathsf{fol_p}(\neg\psi, last)$ holds;
- If $\psi = \psi_1 \wedge \psi_2$, $\rho \models \psi$ iff ρ satisfies both ψ_1 and ψ_2. By induction hypothesis, it is true that $\mathcal{I}_\rho \models \mathsf{fol_p}(\psi_1, last)$ and $\mathcal{I}_\rho \models \mathsf{fol_p}(\psi_2, last)$. Therefore $\mathcal{I}_\rho \models \mathsf{fol_p}(\psi_1, last) \wedge \mathsf{fol_p}(\psi_2, last)$ holds;
- If $\psi = Y\psi_1$, $\rho \models \psi$ iff prefix $\rho' = \rho[0], \rho[1], \ldots, \rho[last - 1]$ of ρ satisfies $\rho' \models \psi_1$. Let $\mathcal{I}'_\rho$ be the corresponding interpretation of ρ', thus for every atom $p \in \mathcal{P}$, $x \in Q'_p$ iff $x \in Q_p$ where Q'_p is the corresponding monadic predicate of p in $\mathcal{I}'_\rho$. By induction hypothesis it is true that $\mathcal{I}'_\rho \models \mathsf{fol_p}(\psi_1, last - 1)$, therefore $\mathcal{I}_\rho \models \mathsf{fol_p}(Y\psi_1, last)$ holds.
- If $\psi = \psi_1 S \psi_2$, $\rho \models \psi$ iff there exists y such that $0 \leq y \leq last$ and prefix $\rho' = \rho[0], \rho[1], \ldots, \rho[y]$ of ρ satisfies ψ_2 and for all z such that $y < z \leq last$, $\rho'' = \rho[0], \rho[1], \ldots, \rho[z]$ satisfies ψ_1. Let $\mathcal{I}'_\rho$ and $\mathcal{I}''_\rho$ be the corresponding interpretations of ρ' and ρ''. Thus for every atom $p \in \mathcal{P}$ it is true that $x \in Q'_p$ iff $x \in Q_p$, $x \in Q''_p$ iff $x \in Q_p$, where Q'_p and Q''_p correspond to the monadic predicates of p in $\mathcal{I}'_\rho$ and $\mathcal{I}''_\rho$ respectively. By induction hypothesis it is true that $\mathcal{I}'_\rho \models \mathsf{fol_p}(\psi_2, last - y)$ and $\mathcal{I}''_\rho \models \mathsf{fol_p}(\psi_1, last - z)$ hold, therefore $\mathcal{I}_\rho \models \mathsf{fol_p}(\psi_1 S \psi_2, last)$. □

4.3 Reversing DFA via Second-Order Logic

For simplification, from now we use ψ to denote the corresponding PLTL$_f$ formula ϕ^R of LTL$_f$ formula ϕ. We first describe how BDDs represent a symbolic DFA. Then we introduce the Compact MSO encoding that inverts the DFA by formulating such BDD representation into a second-order formula. The connection between BDD representation and second-order encoding is novel, to the best of our knowledge.

As defined in Sect. 2.2, given a symbolic DFA $\mathcal{F}_\psi = (\mathcal{P}, \mathcal{X}, X_0, \eta, f)$ represented by a sequence $\mathcal{B} = \langle B_0, B_1, \ldots, B_{k-1}\rangle$ of BDDs, where there are k variables in $\mathcal{X}$, a run of such DFA on a word $\rho = \rho[0], \rho[1], \ldots, \rho[e-1]$ involves a sequence of states $\xi = X_0, X_1, \ldots, X_e$ of length $e+1$. For the moment if we omit the last state reached on an input of length e, we have a sequence of states $\xi' = X_0, X_1, \ldots, X_{e-1}$ of length e. Thus we can think of the run ξ' as a labeling of the positions of the word with states, which is $(\rho[0], X_0), (\rho[1], X_1), \ldots, (\rho[e-1], X_{e-1})$. At each position with given word and state, the transition moving forward involves a computation over every B_q $(0 \leq q \leq k-1)$. To perform such computation, take the high branch in every node labeled by variable $v \in \{\mathcal{X} \cup \mathcal{P}\}$ if v is assigned 1 and the low branch otherwise.

The goal here is to write a formula $\mathsf{Rev}(\mathcal{F}_\psi)$ such that there is an accepting run over $\mathcal{F}_\psi$ of a given word ρ iff ρ^R is accepted by $\mathsf{Rev}(\mathcal{F}_\psi)$. To do this, we introduce one second-order variable V_q for each $x_q \in \mathcal{X}$ with $0 \leq q \leq k-1$, and one second-order variable N_α for every nonterminal node α in BDDs, u nonterminal nodes in total. The V_q variables collect the positions where x_q holds, and the N_α variables indicate the positions where the node α is visited, when computing the transition. To collect all transitions moving towards accepting states, we have BDD $B'_f = f(\eta(\mathcal{X}, \mathcal{P}))$.

Here are some notations. Let α be a nonterminal node, c be a terminal node in B_q such that $c \in \{0,1\}$ and $d \in \{0,1\}$ be the value of v. For nonterminal node α, we define:

$$\mathsf{Pre}(\alpha) = \{(\beta, v, d) \ : \ \text{there is an edge from } \beta \text{ to } \alpha \text{ labelled by } v = d\}$$
$$\mathsf{Post}(\alpha) = \{(\beta, v, d) \ : \ \text{there is an edge from } \alpha \text{ to } \beta \text{ labelled by } v = d\}$$

For every terminal node c in BDD B_q, we define:

$$\mathsf{PreT}(B_q, c) = \{(\beta, v, d) \ : \ \text{there is an edge from } \beta \text{ to } c \text{ labelled by } v = d \text{ in BDD } B_q\}$$

Also, we use $\in^d$ to denote $\in$ when $d = 1$ and $\notin$ when $d = 0$. For each BDD B_q, $\mathsf{root}(B_q)$ indicates the root node of B_q.

We use these notations to encode the following statements:

(1) At the **last** position, state X_0 should hold since ξ' is being inverted and X_0 is the starting point;

$$\mathsf{Rinit} = (x = last) \rightarrow \left(\bigwedge\nolimits_{0 \leq q \leq k-1, X_0(x_q) = d} x \in^d V_q \right);$$

(2) At position x, if the current computation is at nonterminal node α labeled by v, then (2.a) the current computation must come from a predecessor labeled by v' following the value of v', and (2.b) the next step is moving to the corresponding successor following the value of v;

$$\mathsf{node} = \bigwedge\nolimits_{1\leq\alpha\leq u}(\ \mathsf{PreCon} \wedge \mathsf{PostCon}\ ); \text{where}$$

$$\mathsf{PreCon} = \left(x \in N_\alpha \rightarrow (\bigvee_{(\beta,v',d)\in \mathsf{Pre}(\alpha)} [x \in N_\beta \wedge x \in^d v'])\right)$$

$$\mathsf{PostCon} = \left(\bigwedge_{(\beta,v,d)\in \mathsf{Post}(\alpha)} [x \in N_\alpha \wedge x \in^d v \rightarrow x \in N_\beta]\right).$$

(3) At position x such that $\mathbf{x} > \mathbf{0}$, if the current computation node α moves to a terminal node c of B_q, then the value of x_q at position **x-1** is given by the value of c. Such computations of all $B_q (0 \leq q \leq k-1)$ finish one transition;

$$\mathsf{Rterminal} = \bigwedge_{0\leq q\leq k-1}\left(\bigwedge_{(\beta,v,d)\in \mathsf{PreT}(B_q,c)} [(x > 0 \wedge x \in N_\beta \wedge x \in^d v) \rightarrow (x-1 \in^c V_q)]\right);$$

(4) At the **first** position, the current computation on B'_f has to surely move to terminal 1, therefore terminating the running trace of ξ.

$$\mathsf{Racc} = (x = 0) \rightarrow \left(\bigvee_{(\beta,v,d)\in \mathsf{PreT}(B'_f,1)} [x \in N_\beta \wedge x \in^d v]\right).$$

To get all computations over BDDs start from the root at each position, we have

$$\mathsf{roots} = \bigwedge\nolimits_{0\leq x\leq last} \bigwedge\nolimits_{0\leq q\leq k-1} x \in \mathsf{root}(B_q)$$

$\mathsf{Rev}(\mathcal{F}_\psi)$ has to take a conjunction of all requirements above such that

$$\mathsf{Rev}(\mathcal{F}_\psi) = (\exists V_0)(\exists V_1)\ldots(\exists V_{k-1})(\exists N_1)(\exists N_2)\ldots(\exists N_u)(\forall x)(\mathsf{Rinit} \wedge \mathsf{node} \wedge \mathsf{Rterminal} \wedge \mathsf{Racc} \wedge \mathsf{roots}).$$

Therefore, let $\mathsf{Cmso}(\phi)$ be the translation function that given an LTL$_f$ formula ϕ returns a corresponding second-order formula applying the Compact MSO encoding, we define $\mathsf{Cmso}(\phi) = \mathsf{Rev}(\mathcal{F}_\psi)$ asserting the truth of ϕ at position 0, where ψ is the corresponding PLTL$_f$ formula of ϕ, and $\mathcal{F}_\psi$ is the symbolic DFA of ψ. The following theorem asserts the correctness of the Compact MSO encoding.

Theorem 5. *The models of formula* $\mathsf{Cmso}(\phi)$ *are exactly the words satisfying* ϕ.

Proof. We first have that $\mathcal{L}(\phi) = \mathcal{L}^R(\psi) = \mathcal{L}^R(\mathcal{F}_\psi)$ holds since ψ is the corresponding PLTL$_f$ formula of ϕ and $\mathcal{F}_\psi$ collects exactly the words satisfying ψ. Moreover, $\mathcal{L}(\mathsf{Rev}(\mathcal{F}_\psi)) = \mathcal{L}^R(\mathcal{F}_\psi)$ is true following the construction rules of $\mathsf{Rev}(\mathcal{F}_\psi)$ described above and $\mathsf{Cmso}(\phi) = \mathsf{Rev}(\mathcal{F}_\psi)$. Therefore, $\mathcal{L}(\phi) = \mathcal{L}(\mathsf{Cmso}(\phi))$ holds, in which case the models of formula $\mathsf{Cmso}(\phi)$ are exactly the words satisfying ϕ. □

Notice that the size of $\mathsf{Cmso}(\phi)$ is in linear on the size of the BDDs, which lowers the logical complexity comparing to the MSO encoding in Sect. 3. Moreover, in the Compact MSO encoding, the number of existential second-order symbols for state variables are nevertheless possibly less than that in MSO encoding, but new second-order symbols for nonterminal BDD nodes are introduced. BDDs provide a compact representation, in which redundant nodes are reduced. Such advantages allow Compact MSO encoding to use as few second-order symbols for BDD nodes as possible.

5 Experimental Evaluation

We implemented proposed second-order encodings in different parsers for LTL$_f$ formulas using C++. Each parser is able to generate a second-order formula corresponding to the input LTL$_f$ formula, which is then fed to MONA [13] for subsequent symbolic DFA construction. Moreover, we employed *Syft*'s [31] code to translate LTL$_f$ formula into first-order logic (FOL), which adopts the first-order encoding described in Sect. 2.3.

Benchmarks. We conducted the comparison of first-order encoding with second-order encoding 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, of which 700 are scalable LTL$_f$ pattern formulas from [10] and 65 are randomly conjuncted common LTL$_f$ formulas from [7,11,25] ; and 4925 LTL-as-LTL$_f$ formulas from [26,27], since LTL formulas share the same syntax as LTL$_f$.

Experimental Setup. To explore the comparison between first-order and second-order for LTL$_f$-to-DFA translation, 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 8 GB of RAM per core. Time out was set to be 1000 s. Cases that cannot generate the DFA within 1000 s generally fail even if the time limit is extended, since in these cases, MONA typically cannot handle the large BDD.

5.1 Optimizations of Second-Order Encoding

Before diving into the optimizations of second-order encoding, we first study the potential optimization space of the first-order encoding that translates LTL$_f$ to

FOL. Due to the syntax-driven translation of FOL encoding, we are only able to apply different normal forms, Boolean Norma Form (BNF) and Negation Normal Form (NNF). We compared the impact on performance of FOL encoding with two LTL$_f$ normal forms. 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.

To explore the potential optimization space of the second-order encodings proposed in this paper, we hope to conduct experiments with different optimizations. We name second-order encoding with different optimizations *variations*. We first show optimizations of the MSO encoding described in Sect. 3, then describe variations of the Compact MSO encoding shown in Sect. 4 in the following.

The basic MSO encoding defined in Sect. 3 translates LTL$_f$ to MSO in a natural way, in the sense that introducing a second-order predicate for each non-atomic subformula and employing the $\leftrightarrow$ constraint. Inspired by [22,27], we define in this section several optimizations to simplify such encoding thus benefiting symbolic DFA construction. These variations indicating different optimizations 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 Sect. 3 is BNF-*Fussy*-*Full*. Note that BNF-*Sloppy* are incompatible, as described below, and so there are $2^3 - 2 = 6$ viable combinations of the three components above. We next describe the variations in details.

Constraint Form. We call the translation described in Sect. 3 the *Fussy* variation, in which we translate ϕ to MSO formula $\mathsf{mso}(\phi)$ by employing an *iff* constraint (see Sect. 3). For example:

$$\mathsf{t}(\theta_i, x) = (Q_{\theta_i}(x) \leftrightarrow (Q_{\theta_j}(x) \wedge Q_{\theta_k}(x))) \text{ if } \theta_i = (\theta_j \wedge \theta_k) \tag{1}$$

We now introduce *Sloppy* variation, inspired by [27], which allows less tight constraints that still hold correctness guarantees thus may speed up the symbolic DFA construction. To better reason the incompatible combination BNF-*Sloppy*, we specify the description for different normal forms, NNF and BNF separately.

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

The *Sloppy* variation cannot be applied to LTL$_f$ formulas in BNF since the $\leftrightarrow$ constraint defined in function $\mathsf{t}(\theta_i)$ 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.

The following theorem asserts the correctness of the *Sloppy* variation.

Theorem 6. *Let ϕ be an* LTL$_f$ *formula in* NNF *and ρ be a finite trance. Then $\rho \models \phi$ iff $\mathcal{I}_\rho \models \mathsf{mso}_s(\phi)$.*

The proof here is analogous to that of Theorem 2. The crux here is that the $\leftrightarrow$ in $\mathsf{t}(\theta_i)$ is needed only to handle negation correctly. *Sloppy* encoding, however, is applied only to LTL$_f$ formulas in NNF, so negation can be applied only to atomic propositions, which is handled by the base case $(\neg Q_p(x))$.

Variable Form. In all the variations of the MSO encoding we can get above, we introduced a monadic predicate for each non-atomic subformula in $cl(\phi)$, this is the *Full* variation. We now introduce *Lean* variation, a new variable form, aiming at decreasing the number of quantified monadic predicates. Fewer 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 $\phi_j U \theta_k$ or $\phi_j R \theta_k$ (named as U- or R-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 get 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 using built-in procedures in MONA 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 $\mathsf{lean}(\theta_i) = (\mathsf{lean}(\theta_j)$ inter $\mathsf{lean}(\theta_k))$
- If $\theta_i = (\theta_j \vee \theta_k)$, then $\mathsf{lean}(\theta_i) = (\mathsf{lean}(\theta_j)$ union $\mathsf{lean}(\theta_k))$
- If $\theta_i = (X\theta_j)$, then $\mathsf{lean}(\theta_i) = ((\mathsf{lean}(\theta_j) - 1) \backslash \{last\})$
- If $\theta_i = (N\theta_j)$, then $\mathsf{lean}(\theta_i) = ((\mathsf{lean}(\theta_j) - 1)$ union $\{last\})$
- If $\theta_i = (\theta_j U \theta_k)$ or $\theta_i = (\theta_j R \theta_k)$, then $\mathsf{lean}(\theta_i) = Q_{\theta_a}$, where Q_{θ_a} is the corresponding monadic predicate.

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

Lemma 1. *Let ϕ be an* LTL$_f$ *formula, ρ be a finite trace. Then $\rho, x \models \theta_i$ iff $\mathsf{lean}(\theta_i)(x)$ holds, where x is the position in ρ.*

Proof. Suppose first that $\rho, x \models \theta_i$. We prove this inductively on the structure of θ_i.

- If $\theta_i = \neg\theta_j$, then $\mathsf{lean}(\theta_i) = (\mathsf{ALIVE} \backslash \mathsf{lean}(\theta_j))$. $\mathsf{lean}(\theta_i)(x)$ holds since $\mathsf{lean}(\theta_i) = \{x : x \notin \mathsf{lean}(\theta_j)\}$ and $\mathsf{lean}(\theta_j) = \{x : \rho, x \models \theta_j\}$.
- If $\theta_i = \theta_j \wedge \theta_k$, then $\mathsf{lean}(\theta_i) = (\mathsf{lean}(\theta_j)$ inter $\mathsf{lean}(\theta_k))$. $\mathsf{lean}(\theta_i)(x)$ holds since $\mathsf{lean}(\theta_i) = \{x : x \in \mathsf{lean}(\theta_j)$ and $x \in \mathsf{lean}(\theta_k)\}$, $\mathsf{lean}(\theta_j) = \{x : \rho, x \models \theta_j\}$ and $\mathsf{lean}(\theta_k) = \{x : \rho, x \models \theta_k\}$.
- If $\theta_i = \theta_j \vee \theta_k$, then $\mathsf{lean}(\theta_i) = (\mathsf{lean}(\theta_j)$ union $\mathsf{lean}(\theta_k))$. $\mathsf{lean}(\theta_i)(x)$ holds since $\mathsf{lean}(\theta_i) = \{x : x \in \mathsf{lean}(\theta_j)$ or $x \in \mathsf{lean}(\theta_k)\}$, $\mathsf{lean}(\theta_j) = \{x : \rho, x \models \theta_j\}$ and $\mathsf{lean}(\theta_k) = \{x : \rho, x \models \theta_k\}$.
- If $\theta_i = X\theta_j$, then $\mathsf{lean}(\theta_i) = ((\mathsf{lean}(\theta_j) - 1) \backslash \{last\})$. $\mathsf{lean}(\theta_i)(x)$ holds since $\mathsf{lean}(\theta_i) = \{x : x \neq last$ and $x + 1 \in \mathsf{lean}(\theta_j)\}$, $\mathsf{lean}(\theta_j) = \{x : \rho, x \models \theta_j\}$.
- If $\theta_i = N\theta_j$, then $\mathsf{lean}(\theta_i) = ((\mathsf{lean}(\theta_j) - 1)$ union $\{last\})$. $\mathsf{lean}(\theta_i)(x)$ holds since $\mathsf{lean}(\theta_i) = \{x : x = last$ or $x+1 \in \mathsf{lean}(\theta_j)\}$, $\mathsf{lean}(\theta_j) = \{x : \rho, x \models \theta_j\}$.
- If $\theta_i = \theta_j U \theta_k$ or $\theta_i = \theta_j R \theta_k$, then $\mathsf{lean}(\theta_i) = Q_{\theta_a}$. $\mathsf{lean}(\theta_i)(x)$ holds since $\mathsf{lean}(\theta_i) = Q_{\theta_a} = \{x : \rho, x \models \theta_a\}$, where Q_{θ_a} is the corresponding second-order predicate for formula θ_i.

Assume now that $\mathcal{I}^{mso}_{\rho} \models \mathsf{lean}(\theta_i)(x)$ with given interpretations of second-order predicates. We now prove $\rho, x \models \theta_i$ by induction over the structure on θ_i.

- If $\theta_i = \neg\theta_j$, then $\mathsf{lean}(\theta_i) = (\mathsf{ALIVE} \backslash \mathsf{lean}(\theta_j))$. Since $\mathsf{lean}(\theta_i)(x)$ holds, we also have that $x \in \mathsf{lean}(\theta_i)$ iff $x \notin \mathsf{lean}(\theta_j)$. It follows by induction that $\rho, x \models \theta_i$.
- If $\theta_i = \theta_j \wedge \theta_k$, then $\mathsf{lean}(\theta_i) = (\mathsf{lean}(\theta_j)$ inter $\mathsf{lean}(\theta_k))$. Since $\mathsf{lean}(\theta_i)(x)$ holds, we also have that $x \in \mathsf{lean}(\theta_i)$ iff $x \in \mathsf{lean}(\theta_j)$ and $x \in \mathsf{lean}(\theta_k)$. It follows by induction that $\rho, x \models \theta_i$.
- If $\theta_i = \theta_j \vee \theta_k$, then $\mathsf{lean}(\theta_i) = (\mathsf{lean}(\theta_j)$ union $\mathsf{lean}(\theta_k))$. Since $\mathsf{lean}(\theta_i)(x)$ holds, we also have that $x \in \mathsf{lean}(\theta_i)$ iff $x \in \mathsf{lean}(\theta_j)$ or $x \in \mathsf{lean}(\theta_k)$. It follows by induction that $\rho, x \models \theta_i$.
- If $\theta_i = X\theta_j$, then $\mathsf{lean}(\theta_i) = ((\mathsf{lean}(\theta_j) - 1) \backslash \{last\})$. Since $\mathsf{lean}(\theta_i)(x)$ holds, we also have that $x \neq last$ and $x + 1 \in \mathsf{lean}(\theta_j)$. It follows by induction that $\rho, x \models \theta_i$.
- If $\theta_i = N\theta_j$, then $\mathsf{lean}(\theta_i) = ((\mathsf{lean}(\theta_j) - 1)$ union $\{last\})$. Since $\mathsf{lean}(\theta_i)(x)$ holds, we also have that $x = last$ or $x + 1 \in \mathsf{lean}(\theta_j)$. It follows by induction that $\rho, x \models \theta_i$.
- If $\theta_i = \theta_j U \theta_k$ or $\theta_i = \theta_j R \theta_k$, then $\mathsf{lean}(\theta_i) = Q_{\theta_a}$, where Q_{θ_a} is the corresponding second-order predicate. It follows by induction that $\rho, x \models \theta_i$.

□

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

Theorem 7. *Let ϕ be an* LTL$_f$ *formula, ρ be a finite trace. Then $\rho \models \phi$ iff $\mathcal{I}_\rho \models \mathsf{mso}_\lambda(\phi)$.*

Proof. If ϕ is a propositional atom p, then $\mathsf{mso}_\lambda(\phi) = Q_p(0)$. It is true that $\rho \models \phi$ iff $\mathcal{I}_\rho \models \mathsf{mso}_\lambda(\phi)$. If ϕ is an nonatomic formula, we prove this theorem in two directions.

Suppose first that ρ satisfies ϕ. We expand the monadic structure $\mathcal{I}_\rho$ with interpretations for $Q_{\theta_1}, Q_{\theta_2}, \ldots, Q_{\theta_n}$ by setting $Q_{\theta_a} = \{x : \rho, x \models \theta_a\}$. Let the expanded structure be $\mathcal{I}_\rho^{mso}$. By assumption, $\mathsf{lean}(\phi)(0)$ holds in $\mathcal{I}_\rho^{mso}$. It remains to prove that $\mathcal{I}_\rho^{mso} \models (\forall x)(\bigwedge_{a=1}^{n} \mathsf{t}_\lambda(\theta_a, x))$, for each U or R subformula $\theta_a \in cl(\phi)$.

- If $\theta_a = (\theta_j U \theta_k)$, then $\mathsf{t}_\lambda(\theta_a, x) = (\mathsf{lean}(\theta_a)(x) \leftrightarrow (\mathsf{lean}(\theta_k)(x) \vee ((x \neq last) \wedge \mathsf{lean}(\theta_j)(x) \wedge \mathsf{lean}(\theta_a)(x+1))))$. This holds, since $\mathsf{lean}((\theta_j U \theta_k)) = \{x : \rho, x \models \theta_j U \theta_k\} = \{x : \rho, x \models \theta_k$ or $x \neq last$ with $\rho, x \models \theta_j$ also $\rho, x+1 \models \theta_a\}$, $\mathsf{lean}(\theta_j) = \{x : \rho, x \models \theta_j\}$, and $\mathsf{lean}(\theta_k) = \{x : \rho, x \models \theta_k\}$ with Lemma 1;
- If $\theta_a = (\theta_j R \theta_k)$, then $\mathsf{t}_\lambda(\theta_a, x) = (\mathsf{lean}(\theta_a)(x) \leftrightarrow (\mathsf{lean}(\theta_k)(x) \wedge ((x = last) \vee \mathsf{lean}(\theta_j)(x) \vee \mathsf{lean}(\theta_a)(x+1))))$. This holds, since $\mathsf{lean}((\theta_j R \theta_k)) = \{x : \rho, x \models \theta_j R \theta_k\} = \{x : \rho, x \models \theta_k$ with $x = last$ or $\rho, x \models \theta_j$ or $\rho, x+1 \models \theta_a\}$, $\mathsf{lean}(\theta_j) = \{x : \rho, x \models \theta_j\}$, and $\mathsf{lean}(\theta_k) = \{x : \rho, x \models \theta_k\}$ with Lemma 1.

Assume now that $\mathcal{I}_\rho \models \mathsf{mso}_\lambda(\phi)$. This means that there is an expansion of $\mathcal{I}_\rho$ with monadic interpretations Q_{θ_a} for each element θ_a of U or R subformulas in $cl(\phi)$ such that this expanded structure $\mathcal{I}_\rho^{mso} \models (\mathsf{lean}(\phi)(0)) \wedge ((\forall x)(\bigwedge_{a=1}^{n} \mathsf{t}_\lambda(\theta_a, x)))$. If ϕ is not an R or U subformula, then it has been proven by Lemma 1 that if $x \in \mathsf{lean}(\phi)$, then $\rho, x \models \phi$. We now prove by induction on ϕ that if $x \in Q_{\theta_a}$, then $\rho, x \models \phi$. Since $\mathcal{I}_\rho^{mso} \models (\mathsf{lean}(\phi)(0))$, it follows that $\rho, 0 \models \phi$.

- If $\phi = (\theta_j U \theta_k)$, then $\mathsf{t}_\lambda(\phi, x) = (\mathsf{lean}(\phi)(x) \leftrightarrow (\mathsf{lean}(\theta_k)(x) \vee ((x \neq last) \wedge \mathsf{lean}(\theta_j)(x) \wedge \mathsf{lean}(\phi)(x+1))))$. Since $\mathsf{t}_\lambda(\phi)$ holds at every point x of $\mathcal{I}_\rho^{mso}$, it follows that $x \in \mathsf{lean}(\phi)$ iff $x \in \mathsf{lean}(\theta_k)$ or $x \neq last$ with $x \in \mathsf{lean}(\theta_j)$ also $x+1 \in \mathsf{lean}(\phi)$. Moreover, $\mathsf{lean}(\phi) = Q_{\theta_a}$, where Q_{θ_a} is the corresponding second-order predicate. Thus, by induction hypothesis $\rho, x \models \phi$.
- If $\phi = (\theta_j R \theta_k)$, then $\mathsf{t}_\lambda(\phi, x) = (\mathsf{lean}(\phi)(x) \leftrightarrow (\mathsf{lean}(\theta_k)(x) \wedge ((x = last) \vee \mathsf{lean}(\theta_j)(x) \vee \mathsf{lean}(\phi)(x+1))))$. Since $\mathsf{t}_\lambda(\phi)$ holds at every point x of $\mathcal{I}_\rho^{mso}$, it follows that $x \in \mathsf{lean}(\phi)$ iff $x \in \mathsf{lean}(\theta_k)$ with $x = last$ or $x \in \mathsf{lean}(\theta_j)$ or $x+1 \in \mathsf{lean}(\phi)$. Moreover, $\mathsf{lean}(\phi) = Q_{\theta_a}$, where Q_{θ_a} is the corresponding second-order predicate. Thus, by induction hypothesis $\rho, x \models \phi$. □

Having defined different variations of the MSO encoding, we now provide variations of the Compact MSO encoding described in Sect. 4.

Sloppy Formulation. The formulation described in Sect. 4 strictly tracks the computation over each BDD B_q, which we refer to *Fussy* formulation. That is, for each nonterminal node α, both the forward computation and previous computation must be tracked. This causes a high logical complexity in the formulation.

An alteration to diminish the logical complexity is to utilize a *Sloppy Formulation*, analogous to the *Sloppy* variation described above, that only tracks the forward computation. Since the previous computations are not tracked, none of the computations leading to terminal node 0 of the BDD B'_f enable an accepting condition.

To define the accepting condition of *Sloppy Formulation*, we have

$$\mathsf{Racc_s} = \left(\bigvee_{(\beta,v,d)\in \mathsf{PreT}(B'_f,0)} [x \in N_\beta \wedge x \in^d v] \right) \rightarrow (x \neq 0).$$

Moreover, $\mathsf{node_s}$ only requires $\mathsf{PostCon}$ of node. Therefore, we have

$$\mathsf{node_s} = \bigwedge_{1\leq\alpha\leq u} \mathsf{PostCon}.$$

The second-order formula $\mathsf{Rev_s}(\mathcal{F}_\psi)$ of *Sloppy Formulation* is defined as following:

$$\mathsf{Rev_s}(\mathcal{F}_\psi) = (\exists V_0)\ldots(\exists V_{k-1})(\exists N_1)\ldots(\exists N_u)(\forall x)(\mathsf{Rinit} \wedge \mathsf{node_s} \\ \wedge \mathsf{Rterminal} \wedge \mathsf{Racc_s} \wedge \mathsf{roots}),$$

where $\mathsf{Rinit}, \mathsf{Rterminal}$ and roots are defined as in Sect. 4. Therefore, let $\mathsf{Cmso_s}(\phi)$ be the *Sloppy Formulation* of the Compact MSO encoding, we define $\mathsf{Cmso_s}(\phi) = \mathsf{Rev_s}(\mathcal{F}_\psi)$ asserting the truth of ϕ at position 0, where ψ is the corresponding PLTL$_f$ formula of ϕ, and $\mathcal{F}_\psi$ is the symbolic DFA of ψ. The following theorem asserts the correctness of the *Sloppy Formulation*.

Theorem 8. *The models of formula* $\mathsf{Cmso_s}(\phi)$ *are exactly the words satisfying* ϕ.

The proof here is analogous to that of the *Fussy Formulation*, where the crux is that we define the computation trace on a BDD as a sequence of sets of BDD nodes, instead of just a specific sequence of BDD nodes, see the definition of $\mathsf{node_s}$. Such definition still keeps unambiguous formulation of the symbolic DFA since we have stronger constraints on the accepting condition, as shown in the definition of $\mathsf{Racc_s}$.

5.2 Experimental Results

Having presented different optimizations, we now have 6 variations of the MSO encoding corresponding to specific optimizations, which are BNF-*Fussy-Full*, BNF-*Fussy-Lean*, NNF-*Fussy-Full*, NNF-*Fussy-Full*, NNF-*Sloppy-Full* and NNF-*Sloppy-Lean*. Moreover, we have two variations of the Compact MSO encoding, which are *Fussy* and *Sloppy*. The experiments were divided into two parts and resulted in two major findings. First we explored the benefits of the various optimizations of MSO encoding and showed that the most effective one is that of *Lean*. Second, we aimed to answer the question whether second-order outperforms first-order

in the context of LTL$_f$-to-automata translation. To do so, we compared the best performing MSO encoding and Compact MSO encoding against the FOL encoding and showed the superiority of first-order.

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.

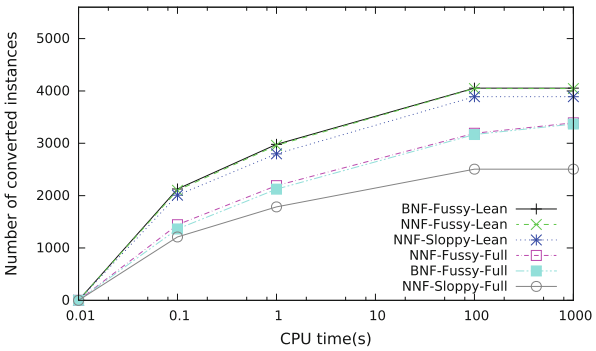

Fig. 1. Comparison over 6 variations of MSO encoding

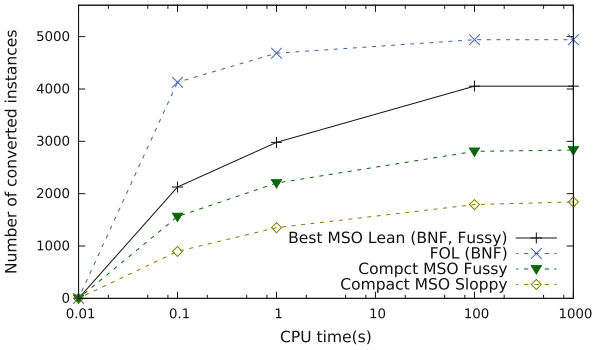

Fig. 2. Overall comparison of FOL, MSO and Compact MSO encodings

Lean Constraint Form Is More Effective in* MSO *Encodings. Fig. 1 presents the number of converted instances of each variation of MSO encoding, where the upper three are all for *Lean* variations and the lower ones are for *Full* variations. The choice of BNF vs NNF did not have a major impact, and neither did the choice of *Fussy* vs *Sloppy*. The one optimization that was particularly effective was that of *Lean* variation. The best-performing MSO encoding was BNF-*Fussy-Lean*. While in the Compact MSO encoding, the *Fussy* variation highly outperforms that of *Sloppy*, as shown in Fig. 2.

First-Order Logic Dominates Second-Order Logic for LTL_f-to-Automata Translation. As presented in Fig. 2, FOL encoding shows its superiority over second-order encodings performance-wise, which are MSO encoding and Compact MSO encoding. Thus, the use of second-order logic, even under sophisticated optimization, did not prove its value in terms of performance. This suggests that nevertheless second-order encoding indicates a much simpler quantificational structure which theoretically leads to more potential space to optimize, it would be useful to have first-order as a better way in the context of LTL_f-to-automata translation in practice.

6 Concluding Remarks

In this paper, we revisited the translation from LTL_f to automata and presented new second-order encodings, MSO encoding and Compact MSO encoding with various optimizations. Instead of the syntax-driven translation in FOL encoding, MSO encoding provides a semantics-driven translation. Moreover, 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. The Compact MSO encoding simplifies further the syntax of the encoding, by introducing more second-order variables. Nevertheless, empirical evaluation showed that first-order encoding, in general, outperforms the second-order encodings. This finding suggests first-order encoding as a better way for LTL_f-to-automata translation.

To obtain a better understanding of the performance of second-order encoding of LTL_f, we looked more into MONA. An interesting observation is that MONA is an "aggressive minimizer": after each quantifier elimination, MONA re-minimizes the DFA under construction. Thus, the fact that the second-order 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. Therefore, 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. Currently, the quantifier-elimination strategy of one quantifier at a time is deeply hardwired in MONA, so the suggested improvement would require a major rewrite of the tool. We conjecture that, with such an extension of MONA, the second-order encodings would have a better performance, but this is left to future work.

Beyond the unrealized possibility of performance gained via second-order encodings, another motivation for studying such encodings is their greater expressivity. The fact that LTL_f is equivalent to FOL [15] shows limited expressiveness of LTL_f. For this reason it is advocated in [6] 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 [8], 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 [9]. We leave this too to future research.

Acknowledgments. Work supported in part by China HGJ Project No. 2017ZX0103 8102-002, NSFC Projects No. 61572197, No. 61632005 and No. 61532019, NSF grants IIS-1527668, IIS-1830549, and by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering". Special thanks to Jeffrey M. Dudek and Dror Fried for useful discussions.

References

1. Bloem, R., Galler, S.J., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Interactive pesentation: automatic hardware synthesis from specifications: a case study. In: DATE, pp. 1188–1193 (2007)
2. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.-F.: Acacia+, a tool for LTL synthesis. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 652–657. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_45
3. Bryant, R.E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput. Surv. **24**(3), 293–318 (1992)
4. Burch, J., Clarke, E., McMillan, K., Dill, D., Hwang, L.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**(2), 142–170 (1992)
5. Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. J. ACM **28**(1), 114–133 (1981)
6. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI, pp. 854–860 (2013)
7. De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on LTL on finite traces: insensitivity to infiniteness. In: AAAI, pp. 1027–1033 (2014)
8. De Giacomo, G., Vardi, M.Y.: Synthesis for LTL and LDL on finite traces. In: IJCAI, pp. 1558–1564 (2015)
9. De Giacomo, G., Vardi, M.Y.: LTL_f and LDL_f synthesis under partial observability. In: IJCAI, pp. 1044–1050 (2016)
10. Di Ciccio, C., Maggi, F.M., Mendling, J.: Efficient discovery of target-branched declare constraints. Inf. Syst. **56**, 258–283 (2016)
11. Ciccio, C.D., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manag. Inf. Syst. **5**(4), 24:1–24:37 (2015)
12. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 – a framework for LTL and ω-automata manipulation. In: ATVA, pp. 122–129 (2016)
13. Henriksen, J.G., et al.: Mona: monadic second-order logic in practice. In: Brinksma, E., Cleaveland, W.R., Larsen, K.G., Margaria, T., Steffen, B. (eds.) TACAS 1995. LNCS, vol. 1019, pp. 89–110. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60630-0_5
14. Akers Jr., S.B.: Binary decision diagrams. IEEE Trans. Comput. **27**(6), 509–516 (1978)
15. Kamp, J.: Tense logic and the theory of order. Ph.D. thesis, UCLA (1968)
16. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA implementation secrets. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 182–194. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44674-5_15

17. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: FOCS, pp. 531–540 (2005)
18. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods Syst. Des. **19**(3), 291–314 (2001)
19. Lichtenstein, O., Pnueli, A., Zuck, L.: The glory of the past. In: Parikh, R. (ed.) Logic of Programs 1985. LNCS, vol. 193, pp. 196–218. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-15648-8_16
20. Biehl, M., Klarlund, N., Rauhe, T.: Mona: decidable arithmetic in practice. In: Jonsson, B., Parrow, J. (eds.) FTRTFT 1996. LNCS, vol. 1135, pp. 459–462. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61648-9_56
21. Pan, G., Sattler, U., Vardi, M.Y.: BDD-based decision procedures for $\mathcal{K}$. In: Voronkov, A. (ed.) CADE 2002. LNCS (LNAI), vol. 2392, pp. 16–30. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45620-1_2
22. Pan, G., Vardi, M.Y.: Optimizing a BDD-based modal solver. In: Baader, F. (ed.) CADE 2003. LNCS (LNAI), vol. 2741, pp. 75–89. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45085-6_7
23. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL, pp. 179–190 (1989)
24. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)
25. Prescher, J., Di Ciccio, C., Mendling, J.: From declarative processes to imperative models. In: SIMPDA 2014, pp. 162–173 (2014)
26. Rozier, K.Y., Vardi, M.Y.: LTL satisfiability checking. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 149–167. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73370-6_11
27. Rozier, K.Y., Vardi, M.Y.: A multi-encoding approach for LTL symbolic satisfiability checking. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 417–431. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21437-0_31
28. Rozier, K.Y., Vardi, M.Y.: Deterministic compilation of temporal safety properties in explicit state model checking. In: Biere, A., Nahir, A., Vos, T. (eds.) HVC 2012. LNCS, vol. 7857, pp. 243–259. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39611-3_23
29. Tabakov, D., Rozier, K.Y., Vardi, M.Y.: Optimized temporal monitors for SystemC. Formal Methods Syst. Des. **41**(3), 236–268 (2012)
30. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: A symbolic approach to safety LTL synthesis. Hardware and Software: Verification and Testing. LNCS, vol. 10629, pp. 147–162. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70389-3_10
31. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: Symbolic LTL$_f$ synthesis. In: IJCAI, pp. 1362–1369 (2017)