The Blow-Up in Translating LTL to Deterministic Automata

Orna Kupferman and Adin Rosenberg

School of Computer Science and Engineering,
Hebrew University, Jerusalem 91904, Israel. {orna,adinr}@cs.huji.ac.il

Abstract. The translation of LTL formulas to nondeterministic automata involves an exponential blow-up, and so does the translation of nondeterministic automata to deterministic ones. This yields a $2^{2^{O(n)}}$ upper bound for the translation of LTL to deterministic automata. A lower bound for the translation was studied in [KV05a], which describes a $2^{2^{\Omega(\sqrt{n})}}$ lower bound, leaving the problem of the exact blow-up open. In this paper we solve this problem and tighten the lower bound to $2^{2^{\Omega(n)}}$.

1 Introduction

The logic LTL (linear temporal logic) [Pnu81] is used for the specification of ongoing behaviors of reactive systems. Such behaviors can be specified also using highly expressive second-order logics, but LTL offers two important advantages. First, writing formulas in temporal logic is simpler. Second, decision procedures for temporal logic are of elementary complexity. These advantages have made temporal logic, and in particular LTL, useful in practice.

The key to the elementary complexity of the decision procedures for temporal logics is their elementary translation to automata on infinite objects. In contrast, the translation of monadic second order logic formulas to automata is nonelementary [Büc62,Rab69]. In particular, given an LTL formula ψ of length n, it is possible to translate ψ to a nondeterministic Büchi word automaton (NBW, for short) with at most $2^{O(n)}$ states [VW94]. The translation of LTL to NBW has been a subject of extensive research, studying its theoretical complexity, optimizations, and performance in practice (c.f., [GPVW95,EH00,SB00,GO01]).

NBWs are strictly more expressive than deterministic Büchi word automata (DBWs, for short): a language $L \subseteq \Sigma^\omega$ can be recognized by a DBW iff there is a language $R \subseteq \Sigma^*$ such that for every word $w \in \Sigma^\omega$, we have that $w \in L$ iff w has infinitely many prefixes in R [Lan69]. All ω-regular languages, however, and therefore also all LTL formulas, can be translated to deterministic word automata with richer acceptance conditions, like Rabin or parity [Saf88,Pit06]. Such a translation is part of several decision procedures for LTL (e.g., synthesis and control [PR89]), algorithms for translating LTL to other logics (e.g., LTL to alternation-free μ-calculus [KV05a] and to general μ-calculus [Dam94]), as well as decision procedures for other logics (e.g., satisfiability for CTL* [ES84]). The

blow-up that the translation involves plays a role even in algorithms that avoid determinization [KV05b,Kup06].

Recall that the translation of LTL to NBW involves an exponential blow-up. Determinization of NBWs also involves an exponential blow-up [Saf88,Pit06], yielding a doubly-exponential upper bound for the translation of LTL to deterministic automata. The doubly-exponential upper bound holds both for deterministic automata with rich acceptance conditions as well as for DBWs. We note, however, that the translation of LTL to DBW, when it exists, can avoid Safra's determinization and is much simpler [BK09]. In [KV05a], Kupferman and Vardi studied a lower bound for the translation. They described a family of languages $L_1, L_2, \ldots$ such that L_n is can be specified by an LTL formula of length $O(n^2)$ yet the smallest DBW for it needs at least 2^{2^n} states. This implies a $2^{2^{\Omega(\sqrt{n})}}$ lower bound for the translation, leaving the problem of the exact tight bound open.

In this paper we solve this problem. We first describe a family of languages $L_1, L_2, \ldots$ such that L_n can be specified by an LTL formula of length $O(n \log n)$ yet the smallest DBW for it needs at least 2^{2^n} states. The languages L_n are defined with respect to an alphabet of a constant size (6 letters). We then show that moving to an alphabet of size $O(n)$ we can tighten the lower bound further and describe a family $L_1, L_2, \ldots$ such that L_n can be specified by an LTL formula of length $O(n)$ yet the smallest DBW for it needs at least 2^{2^n} states. This implies a $2^{2^{\Omega(n)}}$ lower bound for the translation, matching the known upper bound. As in [KV05a], the languages we use are DBW-recognizable. By [KPB94], if a deterministic Rabin automaton (DRW, for short) recognizes a language that is DBW-recognizable, then a DBW for it can be defined on top of the same structure. It follows that our results imply a tight $2^{2^{\Theta(n)}}$ bound for the translation of LTL to both DBW and DRW.

2 Preliminaries

2.1 Linear temporal logic

The logic *LTL* is a linear temporal logic [Pnu81]. Formulas of LTL are constructed from a set AP of atomic propositions using the usual Boolean operators and the temporal operators X ("next time") and U ("until"). Formally, an LTL formula over AP is defined as follows:

- *true*, *false*, or p, for $p \in AP$.
- $\neg\psi_1$, $\psi_1 \wedge \psi_2$, $X\psi_1$, or $\psi_1 U \psi_2$, where ψ_1 and ψ_2 are LTL formulas.

The logic LTL is used for specifying on-going behaviors of reactive systems. Consider a computation $\pi = \pi_0, \pi_1, \pi_2, \ldots$, where for every $j \geq 0$, the set $\pi_j \subseteq AP$ is the set of atomic propositions that hold in the j-th position of π. We denote the suffix $\pi_j, \pi_{j+1}, \ldots$ of π by π^j. We use $\pi \models \psi$ to indicate that an LTL formula ψ holds in the computation π. The relation $\models$ is inductively defined as follows:

- For all π, we have that $\pi \models \mathit{true}$ and $\pi \not\models \mathit{false}$.
- For an atomic proposition $p \in AP$, we have that $\pi \models p$ iff $p \in \pi_0$.
- $\pi \models \neg\psi_1$ iff $\pi \not\models \psi_1$.
- $\pi \models \psi_1 \wedge \psi_2$ iff $\pi \models \psi_1$ and $\pi \models \psi_2$.
- $\pi \models X\psi_1$ iff $\pi^1 \models \psi_1$.
- $\pi \models \psi_1 U \psi_2$ iff there exists $k \geq 0$ such that $\pi^k \models \psi_2$ and $\pi^i \models \psi_1$ for all $0 \leq i < k$.

Each LTL formula ψ over AP defines a language $L(\psi) \subseteq (2^{AP})^\omega$ of the computations that satisfy ψ. Formally, $L(\psi) = \{\pi \in (2^{AP})^\omega | \pi \models \psi\}$.

We denote the size of an LTL formula φ by $|\varphi|$ and we use the following abbreviations in writing formulas:

- $\vee, \rightarrow$, and $\leftrightarrow$, interpreted in the usual way.
- $F\psi = \mathit{true} U \psi$ ("eventually").
- $G\psi = \neg F \neg \psi$ ("always").

2.2 Automata over infinite words

For a finite alphabet Σ, an infinite word $w = \sigma_1 \cdot \sigma_2 \cdots$ is an infinite sequence of letters from Σ. A property of a system with a set AP of atomic propositions can be viewed as a language over the alphabet 2^{AP}. We have seen in Section 2.1 that LTL can be used in order to define properties. Another way to define properties is using automata.

A *nondeterministic Büchi automaton* over infinite words is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$, where Σ is a finite nonempty alphabet, Q is a finite nonempty set of states, $Q_0 \subseteq Q$ is a nonempty set of initial states, $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function, and $\alpha \subseteq Q$ is an acceptance condition. Intuitively, when an automaton $\mathcal{A}$ runs on an input word over Σ, it starts in one of the initial states, and it proceeds along the word according the transition function. Thus, $\delta(q, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Note that the automaton may be *nondeterministic*, since it may have many initial states and the transition function may specify many possible transitions for each state and letter. The automaton $\mathcal{A}$ is *deterministic* if $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all states $q \in Q$ and letters $\sigma \in \Sigma$.

Formally, a *run* r of $\mathcal{A}$ on an infinite word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is an infinite sequence $q_0, q_1, \ldots$ of states in Q such that $q_0 \in Q_0$, and for all $i \geq 0$, we have $q_{i+1} \in \delta(q_i, \sigma_{i+1})$. Note that a nondeterministic automaton can have many runs on a given input word. In contrast, a deterministic automaton can have at most one run on a given input word. The acceptance condition α determines which runs are accepting. A run r is accepting if it visits some state in α infinitely often. Formally, let $\mathit{inf}(r) = \{q : q_i = q \text{ for infinitely many } i\text{'s }\}$. Then, r is accepting iff $\mathit{inf}(r) \cap \alpha \neq \emptyset$. This is called the Büchi acceptance condition.

We also refer here to the Rabin acceptance condition. The Rabin acceptance condition is richer than the Büchi acceptance condition: $\alpha \subseteq 2^Q \times 2^Q$ is a set of pairs of subsets of states, and a run r satisfies a condition $\alpha =$

$\{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$ iff there is $1 \leq i \leq k$ such that $inf(r) \cap G_i \neq \emptyset$ and $inf(r) \cap B_i = \emptyset$. We are not going to use the Rabin condition and only refer to known results about it. We use NBW, DBW, and DRW to denote nondeterministic Büchi automata, deterministic Büchi automata, and deterministic Rabin automata, respectively.

3 From LTL to DBW

It is shown in [VW94] that every LTL formula ψ can be translated to an NBW $\mathcal{A}_\psi$ of size $2^{O(|\psi|)}$ such that $L(\mathcal{A}_\psi) = L(\psi)$. It is shown in [Saf88] that every NBW with n states can be translated to a deterministic Rabin automaton with $2^{O(n \log n)}$ states. It follows that every LTL formula ψ can be translated to a DRW $\mathcal{A}_\psi$ of size $2^{2^{O(|\psi|)}}$ such that $L(\mathcal{A}_\psi) = L(\psi)$. Moreover, it is shown in [KPB94] that DRWs are *Büchi type*: if a DRW recognizes a language that is DBW-recognizable, then an equivalent DBW can be defined on the same structure. It follows that if $L(\psi)$ is DBW-recognizable, then there is a DBW $\mathcal{A}_\psi$ of size $2^{2^{O(|\psi|)}}$ such that $L(\mathcal{A}_\psi) = L(\psi)$.

3.1 The known lower bound: from $O(n^2)$ to 2^{2^n}

In [KV05a], Kupferman and Vardi studied a lower bound for the translation of LTL to DBW. By the Büchi-typeness of DRWs, the same bound applies for the translation of LTL to DRW. We review their result below.

Theorem 1. *[KV05a] There exists an infinite family of DBW-recognizable languages $L_1, L_2, \ldots$ such that for every $n \geq 1$, the language L_n can be specified by an LTL formula of length $O(n^2)$, and every DBW that recognizes L_n has at least 2^{2^n} states.*

Proof: Let $\Sigma = \{a, b, \#, \$\}$. We define the family of languages as follows:

$$L_n = \{(a + b + \#)^* \cdot \# \cdot w \cdot \# \cdot (a + b + \#)^* \cdot \$ \cdot w \cdot \#^\omega \mid w \in \{a, b\}^n\}.$$

For $n \geq 1$, we use the term n-block to refer to a word in $(a + b)^n$. It is not hard to see that L_n contains exactly all words in which some n-block w appears both after the single \$, with a $\#^\omega$ tail after it, and before the \$, where it is surrounded by #'s.

For every n, the language L_n can be specified by the LTL formula

$$\psi_n = [(\neg\$)U(\$ \wedge \overbrace{X((a \vee b) \wedge X((a \vee b) \wedge \ldots X((a \vee b)}^{n} \wedge XG\#\overbrace{)\ldots))}^{n})] \wedge$$
$$F[\# \wedge \textstyle\bigwedge_{1 \leq i \leq n}((X^i a \wedge G(\$ \rightarrow X^i a)) \vee (X^i b \wedge G(\$ \rightarrow X^i b))) \wedge X^{n+1}\#].$$

The first clause of the formula asserts that there is exactly one \$ in the word, followed by an n-block and an infinite tail of #'s. The second clause asserts that there exists a position in which # is true and the i-th letter from this position,

for $1 \leq i \leq n$, agrees with the i-th letter after the \$. Also, the $(n+1)$-th letter from this position is #. Clearly, the length of ψ_n is quadratic in n. Note that the quadratic blow-up arises from the need to repeat n checks, where the check for the i-th letter requires a subformula of length $O(i)$.

By [CKS81], the smallest deterministic automaton on finite words that accepts L_n (omitting the $\#^\omega$ suffix) has at least 2^{2^n} states. The same argument can be used to prove that the smallest DBW that accepts L_n has at least 2^{2^n} states: reaching the \$, the DBW should remember the set of n-blocks that have appeared, surrounded by #'s, before. □

Note that, for simplicity, [KV05a] assumes that the atomic propositions over which the LTL formulas are defined are mutually exclusive (that is, at each moment, exactly one proposition holds). Since the number of atomic propositions is fixed, this can be achieved by adding a conjunction of a fixed size that enforces it.

3.2 Improvement # 1: from $O(n \log n)$ to 2^{2^n} with a fixed alphabet

In the proof above, the LTL formula checks that for some n-block w appearing before the \$, the i-th letter after the \$ matches the i-th letter of w. Each of these checks is done using a subformula of length $O(i)$, and a check is required for all $1 \leq i \leq n$, leading to an overall formula of a quadratic length. In this section we consider a variant of L_n in which each letter in the n-blocks is prefixed by the binary encoding of its position in the block. This enables each of the checks to be specified by an LTL formula of length $O(\log n)$, resulting in a formula of length $O(n \log n)$ for all positions. The formulas should also assert that each letter is indeed prefixed by the encoding of its position, but this can be done by a conjunction of length $O(n \log n)$, leading to an entire formula of length $O(n \log n)$. Formally, we have the following.

Theorem 2. *There exists a family of DBW-recognizable languages $L_1, L_2, \ldots$ over a 6-letter alphabet, such that for every n, the language L_n can be defined by an LTL formula of length $O(n \log n)$, and every DBW that recognizes L_n has at least 2^{2^n} states.*

Proof: Let $\Sigma = \{a, b, \#, \$, 0, 1\}$. We first introduce some notations.

- For $n \geq 1$ and $1 \leq i \leq n$, let $k = \lceil \log n \rceil$ and $b_{n,i}$ be the k-bit binary encoding of $i - 1$. For example, $b_{8,4} = 011$ and $b_{12,11} = 1010$.
- Let $b_{n,i}[j]$ denote the j-th bit in $b_{n,i}$.

We are going to define L_n as the language of words consisting of a sequence of n-blocks, separated by #, followed by a \$, a copy of some n-block, and an infinite tail of #'s. Each n-block must be well-formatted; that is, rather than being a simple word in $(a+b)^n$, it is now a subword of length $n(k+1)$, consisting of n letters in $\{a, b\}$, with the i-th letter, for $1 \leq i \leq n$, being prefixed by $b_{n,i}$. Thus, each bit in the n-block is "labeled" by its position in the block. These

labels allow an LTL formula to efficiently verify that the n-block following the letter \$ is indeed a copy of one of the n-blocks appearing before the letter \$.

We define L_n as an intersection of two languages, S_n and R_n. The language S_n contains all words that have the proper format; i.e., the word is a sequence of n-blocks, separated by #'s, followed by \$, another n-block, and an infinite tail of #'s. The language R_n contains all words in which some n-block surrounded by #'s appear before and after a single \$. Formally, let $r_n = b_{n,1} \cdot (a+b) \cdot b_{n,2} \cdot (a+b) \cdots b_{n,n} \cdot (a+b)$. Then,

$$
\begin{aligned}
S_n &= \# \cdot (r_n \cdot \#)^* \cdot \$ \cdot r_n \cdot \#^\omega \\
R_n &= \bigcup_{w \in r_n} \Sigma^* \cdot \# \cdot w \cdot \# \cdot \Sigma^* \cdot \$ \cdot \Sigma^* \cdot w \cdot \Sigma^\omega \\
L_n &= S_n \cap R_n.
\end{aligned}
$$

We now turn to define the LTL formula ψ_n that specifies L_n. As in [KV05a], we assume that the atomic propositions are mutually exclusive. As there, since the number of atomic propositions is fixed, this can be enforced by a fixed-length subformula. For $1 \leq i \leq n$, we let $\varphi_{n,i}$ assert that the current position starts with $b_{n,i}$. Formally,

$$\varphi_{n,i} = b_{n,i}[1] \wedge X(b_{n,i}[2] \wedge X(b_{n,i}[3] \wedge \ldots \wedge X(b_{n,i}[k]) \ldots)).$$

We now define the LTL formula ψ_n as the conjunction of the following clauses:

$$\# \wedge X(\varphi_{n,1}) \tag{1}$$

$$\wedge\, G(\bigwedge_{i=1}^{n-1} (\varphi_{n,i} \rightarrow X((a \vee b) \wedge X\varphi_{n,i+1}))) \tag{2}$$

$$\wedge\, G(\varphi_{n,n} \rightarrow X((a \vee b) \wedge X(\# \wedge X(\varphi_{n,1} \vee \$ \vee G\#)))) \tag{3}$$

$$\wedge\, (\neg\$)U(\$ \wedge X^{n\cdot(k+1)}G\#) \tag{4}$$

$$\wedge\, F(\# \wedge (\bigwedge_{i=1}^{n} (\varphi_{n,i} \rightarrow X^k \bigvee_{\sigma \in \{a,b\}} (\sigma \wedge F(\$ \wedge F(\varphi_{n,i} \wedge X^k\sigma)))))U\#). \tag{5}$$

Clause (1) asserts that the word begins correctly. Clause (2) asserts that the n-blocks are well formed. Clause (3) asserts that the n-blocks are separated by #'s, and right after them starts a new n-block, or there is a \$, or a $\#^\omega$ tail. Clause (4) asserts that the first \$ symbol is followed by a subword of length $n(k+1)$ after which a $\#^\omega$ tail starts. Clause (5) asserts that there exists a string w, surrounded by #'s, with each letter appearing again after a \$ symbol. Note that Clauses (1) through (4) assert that the word is in S_n, while Clause (5), given that Clauses (1)-(4) hold, adds the requirement that the input is in R_n.

Since $|\varphi_{n,i}| = O(k)$ and $k = \lceil \log n \rceil$, it follows that $|\psi_n| = O(n \log n)$.

Since L_n is of the form $L'_n \cdot \#^\omega$ for a regular language L'_n, a DBW recognizing it can easily be constructed by adding a transition from the accepting state of a DFW accepting L'_n to a one state DBW that accepts the $\#^\omega$ tail.

It is left to prove that every DBW that recognizes L_n must have at least 2^{2^n} states. Assume by contradiction that there exists a DBW $\mathcal{A}$ that recognizes L_n and has less than 2^{2^n} states. For $0 \leq i \leq 2^n - 1$, let w_i be the n-block that corresponds to the the i-th word in $(a+b)^n$, say, according to a lexicographic order. For every $S = \{i_1, i_2, \ldots, i_k\} \subseteq \{0, 1, \ldots, 2^n - 1\}$, let $p_S = \#w_{i_1}\#w_{i_2}\#\ldots\#w_{i_k}\#\$$. Let q_S be the state that $\mathcal{A}$ visits after reading p_S. Since $\mathcal{A}$ has less than 2^{2^n} states, there must be two distinct sets $S, S' \subseteq \{0, 1, \ldots, 2^n - 1\}$ such that $q_S = q_{S'}$. Since $S \neq S'$, there must be $0 \leq i \leq 2^n - 1$ that distinguishes them. Without loss of generality, assume that $i \in S \setminus S'$. Since $i \in S$, it follows that $p_S \cdot w_i\#^\omega \in L_n$, and the run of $\mathcal{A}$ on $p_S \cdot w_i\#^\omega$ is accepting. Therefore, the run on $p_{S'} \cdot w_i\#^\omega$ is accepting as well. However, $i \notin S'$, so $p_{S'} \cdot w_i\#^\omega \notin L_n$, which leads to a contradiction. □

3.3 Improvement #2: from $O(n)$ to 2^{2^n} with a linear alphabet

In the proof above, in a well-formatted n-block, each letter a or b was prefixed by the binary encoding of its position, which is of length $\lceil \log n \rceil$. By using an alphabet of linear size, we can use the alphabet in order to encode the position of the a's and the b's. This will allow the LTL formula to check the matching of each letter in the n-block by a formula of a fixed length. Checking for all letters can then be done by a formula of a linear length. In addition, we have to check that the letters we use indeed encode the positions, which again can be done by a formula of a linear length. Formally, we have the following.

Theorem 3. *There exists a family of DBW-recognizable languages $L_1, L_2, \ldots$ such that for every n, the language L_n can be specified by an LTL formula of length $O(n)$, and every DBW that recognizes L_n has at least 2^{2^n} states.*

Proof: First we define the alphabet Σ_n. Let $\Sigma'_n = \{1, 2, \ldots, n\} \times \{a, b\}$, and $\Sigma_n = \Sigma'_n \cup \{\#, \$\}$. For clarity, we use the symbols a_i and b_i for $\langle i, a\rangle$ and $\langle i, b\rangle$, respectively.

Next, we define L_n. Intuitively, L_n is again the language of words consisting of n-blocks, separated by #'s, followed by a \$ symbol, a copy of some n-block, and an infinite $\#^\omega$ tail. Now however, the n-blocks are well-formatted in a different way: for each $1 \leq i \leq n$, the i-th letter is a_i or b_i. Thus, again each occurrence of a and b is "labeled" by its position in the n-block. These labels allow an LTL formula to efficiently check that the n-block following the \$ symbol is indeed a copy of one of the n-blocks appearing before the \$.

Again, we define L_n as an intersection of S_n and R_n, which are defined exactly as in the proof of Theorem 2, only with $r_n = (a_1 + b_1) \cdot (a_2 + b_2) \cdots (a_n + b_n)$. Thus,

$$
\begin{aligned}
S_n &= \# \cdot (r_n \cdot \#)^* \cdot \$ \cdot r_n \cdot \#^\omega \\
R_n &= \bigcup_{w \in r_n} \Sigma_n^* \cdot \# \cdot w \cdot \# \cdot \Sigma_n^* \cdot \$ \cdot \Sigma_n^* \cdot w \cdot \#^\omega \\
L_n &= S_n \cap R_n
\end{aligned}
$$

Finally, we define an LTL formula ψ_n that recognizes L_n. Again, we assume that the atomic propositions are mutually exclusive. A naive way to enforce this is by a conjunction disabling all pairs of atomic propositions to hold simultaneously. This, however, would result in a formula quadratic in n, and is thus too long. As we shall see below, the fact the formula ψ_n forces the letters # or \$ to appear between n-block can be used in order to specify mutual exclusiveness with a formula of linear length. Now, ψ_n is a conjunction of the following clauses.

$$
\begin{aligned}
&\#\wedge X(a_1 \vee b_1 \vee \$)\\
&\wedge G(\bigwedge_{i=1}^{n-1}((a_i \vee b_i) \rightarrow X(a_{i+1} \vee b_{i+1})))\\
&\wedge G((a_n \vee b_n) \rightarrow X(\# \wedge X(a_1 \vee b_1 \vee \$ \vee G\#)))\\
&\wedge (\neg\$)U(\$ \wedge X((a_1 \vee b_1) \wedge X^n G\#))\\
&\wedge F(\# \wedge X(((\bigwedge_{i=1}^{n}(a_i \wedge F(\$ \wedge Fa_i)) \vee (b_i \wedge \ F(\$ \wedge Fb_i))))U\#))
\end{aligned}
$$

The structure of the clauses is similar to these used in the proof of Theorem 2. Clearly, $|\psi_n| = O(n)$. Also, the language L_n is DBW-recognizable, and the proof that the minimal DBW that recognizes L_n has at least 2^{2^n} states is identical to the previous one, with the present format of n-blocks.

It is left to show that we can enforce mutual exclusion using a formula of linear size. We use the following formula:

$$
\begin{aligned}
&G((\# \vee \$) \rightarrow \neg \bigvee_{i=1}^{n}(a_i \vee b_i))\\
&\wedge G((\# \rightarrow \neg\$))\\
&\wedge G(\bigwedge_{i=1}^{n}(a_i \rightarrow \neg b_i)).
\end{aligned}
$$

The formula guarantees that the atomic propositions # and \$ are mutually exclusive to all other atomic propositions, and that for all $1 \leq i \leq n$, the atomic propositions a_i and b_i are mutually exclusive. We claim that this, together with ψ_n, implies that x_i and y_j are also mutually exclusive, for all $x, y \in \{a, b\}$ and $1 \leq i < j \leq n$. Assume by contradiction that there is some n-block such that both x_i and y_j hold in a position k in the n-block. By the formula ψ_n, the fact that y_j holds in position k implies that # holds in position $k+n-j+1$. Also, the fact that x_i holds in position k and $i < j$ implies that $a_{i+n-j+1} \vee b_{i+n-j+1}$ also holds in position $k+n-j+1$. This, however, contradicts the mutual exclusiveness of # with $a_{i+n-j+1}$ and $b_{i+n-j+1}$, so we are done. □

4 Discussion

We tightened the lower bound in the blow-up involved in the translation of LTL formulas to deterministic Büchi automata from $2^{2^{\Omega(\sqrt{n})}}$ to $2^{2^{\Omega(n)}}$. Interestingly,

we had to distinguish between the case the set of atomic propositions is fixed and the case it is not. This is interesting, as the known translations with which the upper bound for the blow-up is proven do not try to take advantage of a fixed alphabet. Indeed, given an LTL formula ψ of length n, its translation goes through a nondeterministic Büchi automaton with $2^{O(n)}$ states, which is then determinized to a Büchi automaton with $2^{2^{O(n)}}$ states. A more careful analysis of the constants hiding in the $O()$ notations reveals that one can actually take advantage of the fixed alphabet.

In [BKR10], the authors define the class of *ordered alternating automata*. In ordered automata, the non-accepting states of the automaton are ordered, and transitions between non-accepting states must respect the order. LTL formulas can be translated to ordered alternating automata. Unlike general alternating automata, for which removal of alternation involves that break-point construction and a 3^n blow up [MH84], alternation of ordered automata (as well as very weak alternating automata, which are a special case of ordered automata [GO01]) can be removed with only an $n2^n$ blow-up. Moreover, it is shown in [BKR10] that for ordered automata with m letters, alternation can be removed with a 2^{m+n} blow-up, in a construction that makes use of the fact that the break-point construction can be based on subsets of letters rather than subsets of states. Our results here motivate further study of constructions that explicitly refer to the set of letters. It may well be that the lower bound described here for the case of an alphabet of a constant size is tight, and that efforts should now be directed at improving the upper bound for this setting.

Acknowledgement We thank the anonymous reviewers for helpful comments.

References

[BK09] U. Boker and O. Kupferman. Co-ing Büchi made tight and helpful. In *Proc. 24th IEEE Symp. on Logic in Computer Science*, pages 245–254, 2009.

[BKR10] U. Boker, O. Kupferman, and A. Rosenberg. Alternation Removal in Büchi Automata. In *Proc. 37th International Colloquium on Automata, Languages and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 76–87. Springer, 2010.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[Dam94] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[EH00] K. Etessami and G.J. Holzmann. Optimizing Büchi automata. In *11th Int. Conf. on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2000.

[ES84] E.A. Emerson and A. P. Sistla. Deciding branching time logic. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 14–24, 1984.

[GO01] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Proc 13th Int. Conf. on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

[GPVW95] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In P. Dembiski and M. Sredniawa, editors, *Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, 1995.

[KPB94] S.C. Krishnan, A. Puri, and R.K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *Algorithms and Computations*, volume 834 of *Lecture Notes in Computer Science*, pages 378–386. Springer, 1994.

[Kup06] O. Kupferman. Avoiding determinization. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 243–254, 2006.

[KV05a] O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

[KV05b] O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

[Lan69] L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[Pit06] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

[Rab69] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.

[SB00] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proc 12th Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2000.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.