Linear Completeness Thresholds for Bounded Model Checking*

Daniel Kroening[1], Joël Ouaknine[1], Ofer Strichman[2], Thomas Wahl[1], and James Worrell[1]

[1] Department of Computer Science, Oxford University, UK
[2] Technion, Israel

Abstract. *Bounded model checking* is a symbolic bug-finding method that examines paths of bounded length for violations of a given LTL formula. Its rapid adoption in industry owes much to advances in SAT technology over the past 10–15 years. More recently, there have been increasing efforts to apply SAT-based methods to *unbounded* model checking. One such approach is based on computing a *completeness threshold*: a bound k such that, if no counterexample of length k or less to a given LTL formula is found, then the formula in fact holds over all infinite paths in the model. The key challenge lies in determining sufficiently small completeness thresholds. In this paper, we show that if the Büchi automaton associated with an LTL formula is *cliquey*, i.e., can be decomposed into clique-shaped strongly connected components, then the associated completeness threshold is *linear* in the recurrence diameter of the Kripke model under consideration. We moreover establish that all unary temporal logic formulas give rise to cliquey automata, and observe that this group includes a vast range of specifications used in practice, considerably strengthening earlier results, which report manageable thresholds only for elementary formulas of the form $\mathsf{F}\,p$ and $\mathsf{G}\,q$.

1 Introduction

LTL *bounded model checking* (BMC) [4,3] is a symbolic bug-finding method that searches for lasso-shaped counterexamples to an LTL formula in a given Kripke structure. Within three or four years following its introduction, it was found to have almost entirely replaced BDD-based model checkers in the hardware industry, owing to the fact that many users care more about finding bugs quickly than about formal proofs of their absence, especially as the latter often require vast amounts of memory and time. This major success can be attributed mostly to the impressive advances made in SAT technology over the past 10 to 15 years.

The fundamental approach underpinning BMC is to look for counterexamples, or bugs, of bounded length. As such, an absence of counterexample is inconclusive; a genuine bug could still lurk deeper in the system. For this reason, from the very inception of the technique, researchers have attempted to turn BMC into a

* Supported by the EU FP7 STREP PINCETTE.

complete method with the ability also to guarantee the absence of counterexamples of any length. See, for instance, the original work of Biere *et al.* [4], or the 2008 Turing Award lecture of Ed Clarke [7], in which the problem is described as a topic of active research.

In [4], Biere *et al.* observed that for safety properties of the form $\mathsf{G}\, p$, a *completeness threshold* is given by the *diameter* (longest distance between any two states) of the Kripke structure under consideration: indeed, if no counterexample to $\mathsf{G}\, p$ of length at most the diameter of the system can be found, then no counterexample of any length can possibly exist. Likewise, for liveness properties such as $\mathsf{F}\, q$, the *recurrence diameter* (longest loop-free path) of the Kripke structure can be seen to be an adequate completeness threshold. But the general problem of determining reasonably tight completeness thresholds for arbitrary LTL formulas remains wide open to this day.

Note that the diameter (for safety properties) and the recurrence diameter (for liveness properties) are not merely sound bounds, they are also worst-case tight. In other words, no smaller completeness threshold expressible strictly in terms of the diameters can be achieved. Of course, in any particular situation the least completeness threshold may well be orders of magnitude smaller than the diameter, but determining its value is clearly at least as hard as solving the original model-checking problem in the first place, and we must therefore be content with sound but reasonably tight over-approximations.

In this paper, we describe an efficient technique for obtaining fairly tight, *linear* completeness thresholds for a wide range of LTL formulas, as a function of the diameter and recurrence diameter of any Kripke structure under consideration. All Büchi automata that are *cliquey*, i.e., that can be decomposed into clique-shaped strongly connected components, admit linear completeness thresholds. Moreover, we show that such automata subsume *unary linear temporal logic*, and indeed comprise a wide range of formulas used in practice, including, for example, the vast majority of specifications appearing in Manna and Pnueli's classic text on the specification of reactive and concurrent systems [12].[1] We also show that computing these linear completeness thresholds can be done in time linear in the size of the given Büchi automata. Finally, we exhibit some simple (non-cliquey) Büchi automata, and corresponding LTL formulas, having *superpolynomial* and even *exponential* completeness thresholds.

In the past, researchers have been able to achieve completeness thresholds by studying the *product* structure of the Kripke model and the Büchi automaton corresponding to the specification of interest; see, e.g., [6,1]. Such thresholds are in general incomparable with the ones we present in this paper. Moreover, a significant disadvantage of the earlier approach is that it requires one to investigate a structure which is often much too large and unwieldy to construct, let alone perform any calculations upon. Another benefit of the present approach is that, once the diameter and recurrence diameter of a given Kripke structure are known (or over-approximated), they can be put to use against any

[1] For instance, specifications such as *conditional safety*, *guarantee*, *obligation*, *response*, *persistence*, *reactivity*, *justice*, *compassion*, etc., all fall within our framework.

number of specifications, whereas the earlier approach requires a fresh calculation of the diameters of each of the different product automata, possibly resulting in prohibitive computation costs.

Orthogonal research directions aiming to achieve completeness of bounded model checking include cube enlargement techniques [13], circuit co-factoring [8], induction [16], and Craig interpolation [14].

2 Definitions

By LTL\X we denote standard propositional linear temporal logic [5] without the *next-time* operator X; all the results in this paper also hold if backwards temporal operators are included as well. Every LTL\X formula φ is *invariant under stuttering*, meaning that any two stuttering-equivalent paths either both satisfy or both violate φ (see [5] for a precise definition).[2]

Let AP be a finite set of atomic propositions. A *Kripke structure* is a tuple $M = (S, S_0, R, L)$ with finite set of states S, set of initial states $S_0 \subseteq S$, transition relation $R \subseteq S \times S$, and state-labelling function $L\colon S \to 2^{AP}$, which assigns to each state the set of atomic propositions considered to be true in that state. A *path* through M is a sequence $(\pi_i)_{i=0}^{l}$ ($l \in \mathbb{N} \cup \{\infty\}$) of states such that for $i < l$, $(\pi_i, \pi_{i+1}) \in R$. By $|\pi|$ we denote the number l of edges in π. Thus, if π is finite, its last vertex is $\pi_{|\pi|}$. An infinite path π is *lasso-shaped and k-bounded* if there exist two finite paths u and v such that $\pi = u.v^\omega$ and $|u|+|v| \leq k$. Here, v^ω denotes infinite repetition of v, and u and v^ω are concatenated. For concatenating two paths, we require that the last state of the first path be identical to the first state of the second path. Thus the definition of *k-bounded* implies that $u_{|u|} = v_0 = v_{|v|}$. The concepts of 'reachable' (existence of a connecting path) and 'distance between reachable states' (length of shortest connecting path) are defined in the standard way.

A *(generalised) Büchi automaton* is a tuple $B = (S, S_0, R, L, \mathcal{A})$ with finite set of states S, set of initial states $S_0 \subseteq S$, transition relation $R \subseteq S \times S$, state-labelling function $L\colon S \to \mathcal{B}(AP)$, and family $\mathcal{A} \subseteq 2^S$ of accepting sets of states; here $\mathcal{B}(AP)$ denotes the set of all Boolean combinations of atomic propositions in AP. Note that states (rather than transitions) are labelled, namely by such Boolean combinations of atomic propositions. An infinite path π through B is *accepting* if, for each state set $T \in \mathcal{A}$, π visits a state of T infinitely often.

The *product* of Kripke structure $M = (S, S_0, R, L)$ with Büchi automaton $B = (S', S'_0, R', L', \mathcal{A}')$, denoted $M \times B$, is defined as the Büchi automaton $(S'', S''_0, R'', L'', \mathcal{A}'')$ with

- $S'' = \{(s, s') \in S \times S' \mid L(s) \vDash L'(s')\}$[3]
- $S''_0 = (S_0 \times S'_0) \cap S''$

[2] Many computer scientists, starting with Lamport in the 1980s [11], have argued that high-level specifications of computer systems always ought to be stuttering-invariant.

[3] By $L(s) \vDash L'(s')$, we mean that the Boolean formula $L'(s')$ evaluates to *true* if all variables in $L(s)$ are assigned *true* and all other variables are assigned *false*.

- $R'' = \{((s_1, s'_1), (s_2, s'_2)) \in S'' \times S'' \,|\, (s_1, s_2) \in R \text{ and } (s'_1, s'_2) \in R'\}$
- $L'' \colon S'' \to 2^{AP} \times \mathcal{B}(AP)$ with $L''(s, s') = (L(s), L'(s'))$
- $\mathcal{A}'' = \{(S \times T') \cap S'' \,|\, T' \in \mathcal{A}'\}$.

Note that the labelling functions of M and B determine which states exist (are valid) in the product $M \times B$. There is a transition in the product iff corresponding transitions are present in both components. For our purposes, the labelling of states in the product automaton is irrelevant. Finally, the acceptance set family $\mathcal{A}''$ is derived from that of the Büchi automaton.

The product construction is related to LTL model checking as follows:

Theorem 1 ([9]). *Let M be a Kripke structure and φ an LTL formula. There exists a generalised Büchi automaton $B_{\neg\varphi}$ such that $M \models \varphi$ exactly if $M \times B_{\neg\varphi}$ has no accepting path.*

In figures, we represent Büchi automata as directed graphs. Initial states have an incoming edge without source. Accepting states are drawn as filled discs (our illustrating examples all have a singleton acceptance set family, in other words they are simple Büchi automata), and other states are drawn as hollow circles. In Kripke structures (cf. Figure 4), we depict the label of a state as a set of propositions, omitting the braces {}.

For a Kripke structure M, we write $M \models_k \varphi$ to denote that every lasso-shaped k-bounded path π in M satisfies φ. A *completeness threshold for M and φ* is an integer k such that

$$M \models_k \varphi \quad \Rightarrow \quad M \models \varphi\,.$$

This definition reflects the intuition behind bounded model checking: assuming that there is no counterexample to φ of length at most k, φ should hold in M. We can generalise this definition to Büchi automata as follows: a *completeness threshold for a Kripke structure M and a Büchi automaton B* is any integer k such that, if $M \times B$ has *any* accepting path, then it has a k-bounded lasso-shaped accepting path. With these definitions, an integer k is a completeness threshold for a Kripke structure M and formula φ precisely if it is a completeness threshold for M and $B_{\neg\varphi}$, where $B_{\neg\varphi}$ is the result of translating $\neg\varphi$ into any equivalent generalised Büchi automaton.

The following are key notions in this paper:

Definition 2. *Let M be a Kripke structure. The* ***distance*** *from a state s to a state t is the length of a shortest path from s to t (or ∞ if there is no such path). The* ***diameter*** *of M, denoted $d(M)$, is the largest distance between any two reachable states ('longest shortest path'). The* ***recurrence diameter*** *of M, denoted $rd(M)$, is the length of a longest simple (loop-free) path through M.*

3 Büchi Automata with Linear Completeness Thresholds

Given a Kripke structure and an LTL formula, it is clear that determining the *smallest* completeness threshold is at least as hard as the model-checking problem itself, and is thus not something we are aiming to achieve. Rather, the goal of this paper is to establish a class of LTL formulas admitting completeness thresholds that are *linear* in the diameter and the recurrence diameter of any given Kripke structure. In this section, we first introduce a class of generalised Büchi automata, termed *cliquey*, with this property. We also present an algorithm which, given a cliquey automaton B, produces a symbolic arithmetic expression $ct(d, rd)$ such that, for any Kripke structure M, $ct(d(M), rd(M))$ is a valid completeness threshold for M and B. Moreover, the expression ct is linear in d and rd. In Section 4, we exhibit a class of LTL formulas that have cliquey Büchi representations, namely *unary linear temporal logic* formulas.

Let $sap(B) = \min\{k \mid B \text{ has a lasso-shaped } k\text{-bounded accepting path}\}$, for a non-empty Büchi automaton B. An $sap(B)$-bounded accepting path is called a *shortest accepting path*, SAP for short. If B is empty, let $sap(B) = -\infty$.

Definition 3. *A generalised Büchi automaton B* ***has a linear completeness threshold*** *if there exists $c \in \mathbb{N}$ such that for all Kripke structures M, $sap(M \times B) \leq c \cdot rd(M)$.*

We are now ready to define a class of Büchi automata that admit linear completeness thresholds.

3.1 Cliquey Büchi Automata

The class of automata we consider in this section is characterised by a particular structure of the underlying transition graph:

Definition 4. *A directed graph is* ***cliquey*** *if every maximal strongly connected component (SCC) is a bidirectional clique, i.e. any two nodes of an SCC are connected by an edge in either direction. In particular, every node has a self-loop.*

We say that a generalised Büchi automaton is *cliquey* if its underlying graph structure (ignoring the vertex labelling and the accepting condition) is cliquey. The Büchi automaton in Figure 1 (a) is cliquey, whereas that in (b) is not cliquey. Moreover, we shall see in Section 5.1 that the latter has no equivalent cliquey representation.

Theorem 5. *Every cliquey generalised Büchi automaton admits a linear completeness threshold.*

Proof: Let B be cliquey, M an arbitrary Kripke structure with recurrence diameter rd, and $\Pi = M \times B$. We show $sap(\Pi) \leq c \cdot rd$, for a number c that can be chosen independently of M. If Π has no accepting path, then $sap(\Pi) = -\infty$, so there is nothing to prove. Otherwise, let π be an SAP of Π, and let $C_1, \ldots, C_s$ be the sequence of SCCs of B that π traverses, in this order. We now bound the length of π in each SCC.

Consider a non-final SCC C_i (i.e., $i < s$), and let π^i be the segment of π that traverses C_i (in other words, π^i is a maximal segment of π such that the projection of its states to B is a path in C_i). Suppose $|\pi^i| \geq rd + 2$. The prefix of π^i of length $|\pi^i| - 1$ exceeds M's recurrence diameter rd. Thus we can find two product states of the form (m, b) and (m, b') along this segment. Let (m'', b'') be the successor of (m, b') along π^i (note that (m'', b'') still belongs to π^i):

$$(m, b) \quad \rightsquigarrow \quad (m, b') \rightarrow (m'', b'')\,.$$

From $(m, b') \rightarrow (m'', b'')$ in Π, we conclude $m \rightarrow m''$ in M. Since C_i is a clique, we conclude $b \rightarrow b''$ in B. Hence, $(m, b) \rightarrow (m'', b'')$ in Π. This, however, contradicts the fact that π is an SAP through Π. Therefore, $|\pi^i| \leq rd + 1$.

Consider now the final SCC C_s, and let the family of accepting sets of B be $\mathcal{A} = \{A_1, \ldots, A_n\}$. The segment π^s of π traversing C_s visits each A_i infinitely often. Since each A_i is finite, there exists in fact a fixed state $a_i \in A_i$ in each of them that is visited infinitely often. Segment π^s thus looks like this:

$$(m, b) \quad \rightsquigarrow \quad (m_1, a_1) \quad \rightsquigarrow \quad (m_2, a_2) \quad \rightsquigarrow \quad \ldots \quad \rightsquigarrow \quad (m_n, a_n) \quad \rightsquigarrow \quad (m, b)\,.$$

Using the same argument as in the non-final case, each segment abbreviated by $\rightsquigarrow$ has length at most $rd + 1$ (otherwise, a shorter accepting path could be constructed). As a result, $|\pi^s| \leq (n + 1)(rd + 1)$.

In total, $|\pi| \leq (s - 1)(rd + 1) + (n + 1)(rd + 1)$, clearly inducing a linear completeness threshold, for example with constant $c = 2(s + n)$ (note that s and n are parameters of B and do not depend on M). □

3.2 Computing Completeness Thresholds of Cliquey Automata

The proof in Section 3.1 establishes linearity of the completeness threshold for cliquey Büchi automata. It is, however, very coarse. Among others, the argument ignores the structure of the SCC quotient graph. In the following, we give a higher-order algorithm that takes a cliquey Büchi automaton B as input and returns a function *ct* over two arguments. When supplied with the diameter d and the recurrence diameter *rd* of a Kripke structure M, this function returns a completeness threshold for M and B: $sap(M \times B) \leq ct(d, rd)$. Exploiting the fact that B is cliquey, $ct(d, rd)$ will be linear in d and *rd*.

The algorithm proceeds in two stages. In the first stage, each clique in the SCC quotient graph of B is assigned a cost, as a function of d and *rd*, of traversing it in the product automaton $M \times B$, namely the maximum length a path segment can 'spend' in this clique, given that the path is an SAP. In the second stage, the algorithm traverses the SCC quotient graph, in order to *symbolically* compute respective longest paths from initial cliques to all cliques, using the cost measures computed during the first stage. The result returned by function *ct* is then the maximum path length computed, over all cliques that could potentially serve as the clique visited last along an accepting path.

The Cost of Traversing a Clique. For a generalised Büchi automaton B with accepting sets $A_1, \ldots, A_n$, call a clique C in B *accepting* if for each $i \in \{1, \ldots, n\}$, $C \cap A_i \neq \emptyset$. Such a clique contains a state from each accepting set and is thus eligible as a *final* clique, visited infinitely often, as $M \times B$ is traversed. Further, we say C is *vacuously labelled* if, for each Büchi state b in C, $L(b) = true$. The significance of this condition is that such a Büchi state b can be paired with any Kripke state m: the condition $L(m) \vDash L(b)$ holds for any m. This will be instrumental in redirecting certain non-optimal paths, as we shall see below.

We denote the cost of traversing C as a non-final clique by $\mathit{cost}[C]$, and the cost of traversing C as a final clique by $\mathit{cost}_f[C]$. These values are assigned according to Table 1, depending on whether C is vacuously labelled or not, and whether C is accepting or not. Note that an accepting clique may well be traversed as a non-final clique, whereas a non-accepting clique cannot be traversed as a final clique.

Table 1. Over-approximating the cost of traversing a clique

C vacuously labelled?	C accepting?	$\mathit{cost}[C]$	$\mathit{cost}_f[C]$
no	no	$\mathit{rd}+1$	∞
no	yes	$\mathit{rd}+1$	$(n+1)(\mathit{rd}+1)$
yes	no	d	∞
yes	yes	d	$(n+1)d$

To discuss these figures, consider first the typical case in which C is not vacuously labelled. We have seen in the proof of Theorem 5 in Section 3.1 that any SAP segment within C, visited as a non-final clique, is of length at most $\mathit{rd}+1$; it does not matter here whether C is accepting or not. When visited as the final clique, however, C must be accepting; in this case said segment is of length at most $(n+1)(\mathit{rd}+1)$, as shown in the same proof.

If C is vacuously labelled, we can strengthen the argument from Section 3.1: let $s = (m_0, b_0) \rightsquigarrow (m_1, b_1)$ be a path segment of an SAP whose projection to the B component runs within C. Suppose the length of s is greater than M's diameter d. Then there is a path π from m_0 to m_1 in M with $|\pi| < |s|$. Path π can be used to form a path through C in the product that is shorter than s: pair every state along π with b_0, except the last state m_1, which is paired with b_1 as above. The product states (m, b) thus created *vacuously* satisfy the condition $L(m) \vDash L(b)$, since $L(b) = \mathit{true}$. The B-components of the new path form a path in B, since they run within a clique, with self-loops on all states. These findings contradict the fact that s is a segment of an SAP through $M \times B$. Therefore, $|s| \leq d$, so d is the cost of traversing C as a non-final clique. For traversing C as the final clique (assuming it is accepting), the cost is $(n+1)$ times higher, as before: we apply the diameter argument to each subsegment of s between two accepting states.

The Cost of Traversing the SCC Quotient Graph. The second stage is to 'collect' the costs we have computed per clique in stage 1. Which cliques of B are visited in an actual SAP in $M \times B$ of course depends on M. For our results to hold over any Kripke structure, we determine a *longest* path through the SCC quotient graph. This quotient graph is acyclic, so that the single-source longest path problem can be solved in time linear in the number of quotient edges, by traversing the graph in topological order.

A complication is that, since we do not have the concrete Kripke structure at hand, the costs of moving from clique to clique are given symbolically, by expressions of the form appearing in Table 1. Thus, when comparing the lengths of paths to a particular clique found so far, instead of recording the new length as the numerical maximum of the two given lengths, we record it as the *symbolic maximum* of the two length expressions. The final result reported by the function will thus be an expression involving the parameters d and rd of the unknown Kripke structure, as well as **linear** operators connecting them, such as addition, constant multiplication, and max.

The traversal of the SCC quotient graph is shown in Algorithm 1. It assumes the Büchi automaton has a unique initial clique C_0 (i.e., a clique containing initial states of B); we handle the general case below. The algorithm keeps the cost of traversing a clique, as computed in Table 1, in an array *cost*, and the cost of reaching and traversing a clique in an array *reach*, both as a non-final and final clique (the latter stored in arrays with subscript f). The *reach* values are initialised to 0. For the initial clique, these values are set to the cost to traverse it (Line 4).

Algorithm 1. Maximum length of an SAP in $M \times B$

```
Input: $B$ with initial clique $C_0$
 0: for each clique $C$ do
 1:     initialise $cost[C]$, $cost_f[C]$ as in Table 1
 2:     $reach[C] := reach_f[C] := 0$
 3: end for
 4: $reach[C_0] := cost[C_0]$, $reach_f[C_0] := cost_f[C_0]$
 5: for each clique $C$ of $B$ in a topological order, starting at $C_0$ do
 6:     for each successor clique $D$ of $C$ do
 7:         $reach[D] := \max\{reach[D], reach[C] + cost[D]\}$
 8:         if $D$ is accepting then
 9:             $reach_f[D] := \max\{reach_f[D], reach[C] + cost_f[D]\}$
10:         end if
11:     end for
12: end for
13: return $\max\{reach_f[C] \mid C$ is accepting$\}$
```

The algorithm traverses the cliques C of B in some topological order, starting with C_0, and examines all of C's successor cliques D. Value *reach* is updated to the maximum of its current value and the value obtained by reaching D via C. Value $reach_f$ is updated analogously, but only if D is accepting. After processing all cliques this way, the algorithm returns the maximum of the values $reach_f[C]$ over all accepting cliques.

If B has several initial cliques, the algorithm is performed for each of them in turn; in this case we return the maximum over all values obtained, as the maximum length of an SAP, for any Kripke structure M.

Complexity. The applications of $+$ and max in the algorithm are to be understood as symbolic operations. That is, the result returned in Line 13 is an expression, involving d, rd and the linear operators $+$, max, and constant multiplication (the latter come from the base expressions in Table 1). In the worst case, the expression can contain a number of max applications that is linear in the number of edges of the SCC quotient graph. In many cases, however, the symbolic maximum can be evaluated to one of its arguments. For example, $\max\{rd, d\} = rd$, while $\max\{rd, 2d\}$ cannot be simplified.

The algorithm itself also has linear time complexity, as a (slight modification of the) standard algorithm to compute longest paths in a directed acyclic graph.

4 Linear Temporal Logic and Cliquey Automata

We now turn to temporal-logic model checking, and investigate the relationship with cliquey Büchi automata. In this section, we write bold-face **LTL**, **LTL\X**, and **CL** to denote the set of ω-regular languages that correspond to LTL formulas, LTL\X formulas, and cliquey Büchi automata respectively.

Lemma 6. CL $\subseteq$ LTL: *every cliquey generalised Büchi automaton can be encoded as an equivalent LTL formula.*

Proof: Given a cliquey automaton B, we show how to express its language in LTL. Recall that every strongly connected component of B is a clique, and that an *accepting clique* is one that has a non-empty intersection with each accepting set of B. There are only a finite number of SCC paths that go from a initial clique to an accepting clique. Moreover, for each pair of neighbouring cliques C_i, C_{i+1} along such a path, there is only a finite number of edges from C_i to C_{i+1}. We can therefore write the language of B as a finite union of 'path languages' each of which encodes the words along a path π to an accepting clique such that any successive cliques C_i, C_{i+1} along π are connected by a unique edge.

Each path language can be written as an ω-regular expression over the alphabet 2^{AP}, of the form

$$\underbrace{L(i_0).L(C_0)^*.L(o_0)}_{\text{first clique}}\ \underbrace{L(i_1).L(C_1)^*.L(o_1)}_{\text{second clique}}\ L(i_2)\ \ldots\ L(o_{f-1})\ \underbrace{L(i_f).L(C_f)^\omega}_{\text{final clique}} \quad (1)$$

where, for clique number j, $L(i_j)$ is the labelling of the unique in-point (entry) of C_j coming from C_{j-1}, $L(o_j)$ is the labelling of the unique out-point (exit) of C_j to C_{j+1}, and $L(C_j)$ is the union of the labellings of all Büchi states in C_j. For example, a clique of three states, with the entry state labelled a, exit state labelled b and a third state labelled c, where $a, b, c \in 2^{AP}$, is encoded as a regular expression $a.\{a, b, c\}^*.b$.

Expression (1) can be turned into a star-free ω-regular expression by replacing the subexpressions $L(C_l)^*$ by $\overline{\overline{\emptyset}.\overline{L(C_l)}.\overline{\emptyset}}$, where $\overline{}$ denotes complementation. Being star-free, it is well-known that this expression is equivalent to a suitable LTL formula [10,15]. □

Lemma 7. CL $\not\subseteq$ LTL\X : *there exist cliquey automata that cannot be encoded as LTL\X formulas.*

Proof: Consider the Büchi automaton B in Figure 1 (a). B is cliquey: the two p-labelled states form one SCC, the q-labelled state forms the other SCC, and both are cliques. B does not, however, correspond to any LTL\X formula: B's language contains the word $\{p\}.\{p\}.\{q\}^\omega$, but not the word $\{p\}.\{q\}^\omega$. Since these two words are stuttering equivalent, an encoding of B as an LTL\X formula would violate the stuttering closure of LTL\X. $\square$

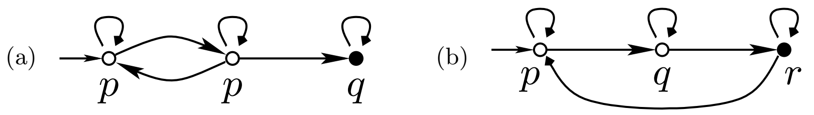

Fig. 1. (a) A cliquey Büchi automaton that does not correspond to any LTL\X formula; (b) A non-cliquey Büchi automaton with linear completeness threshold

Lemma 8. LTL\X $\not\subseteq$ CL : *not all LTL\X formulas have a cliquey automaton encoding.*

Proof: Let $AP = \{p, q, r\}$, and let $p!$ be a short-hand notation for $p \wedge \neg q \wedge \neg r$, and similarly for $q!$ and $r!$. Consider the LTL\X formula

$$\varphi \;=\; p! \;\wedge\; \mathsf{G}\;(\;(p! \Rightarrow (p!\,\mathsf{U}\,q!)) \;\wedge\; (q! \Rightarrow (q!\,\mathsf{U}\,r!)) \;\wedge\; (r! \Rightarrow (r!\,\mathsf{U}\,p!))\;)\;. \quad (2)$$

To prove that φ does not have a cliquey Büchi encoding, we first show:

Property 9. *Any cliquey Büchi automaton over $AP = \{p, q, r\}$ that accepts the word $(\{p\}.\{q\}.\{r\})^\omega$ also accepts some word in $(2^{AP})^*.\{q\}.\{p\}.(2^{AP})^*$.*

Proof: Let B be cliquey and accept $w := (\{p\}.\{q\}.\{r\})^\omega$. We show that B also accepts some word with the substring $\{q\}.\{p\}$. Any path π in B along which w is accepted contains infinitely many states with a label that is satisfied by $\{p\}$. Since B has finitely many states, these states are not all different; let b be a state with such a label that occurs twice along π. Let c be the state following the first occurrence of b; the label of c is satisfied by $\{q\}$. Since c is between two occurrences of b along π, states b and c belong to the same SCC. As B is cliquey, b and c are in fact part of one clique; thus there is an edge from c to b in B. Now consider the path π' that is identical to π, except that one occurrence of the edge $b \to c$ is replaced by the segment $b \to c \to b \to c$. Path π' is a valid path in B. It is also accepting, since we have only added two edges to the accepting path π. Finally, π' accepts a word that contains the substring $\{q\}\{p\}$. $\square$

We can now prove Lemma 8: any automaton B that encodes φ in equation (2) accepts $(\{p\}.\{q\}.\{r\})^\omega$, but does not accept any word with substring $\{q\}.\{p\}$, since a path with such a trace would violate φ. It follows that B is not cliquey. $\square$

We have so far shown that while **CL** is contained in **LTL**, **LTL\X** and **CL** are incomparable. Of particular interest is the intersection of the latter two: LTL\X formulas that have a cliquey representation. To narrow in on this class, consider the *unary temporal logic* fragment of LTL\X, denoted UTL\X, i.e., LTL\X without the *until* operator U (and, if one is using past temporal operators, without the past counterpart of U either). The formulas of UTL\X are built from atomic propositions using Boolean connectives and the unary temporal operators F (*eventually*), and $\overleftarrow{\mathsf{F}}$ (*sometime in the past*)—the dual operators G and $\overleftarrow{\mathsf{G}}$ are derived in the usual way. Formally, UTL\X is defined by the following grammar:

$$\varphi ::= p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathsf{F}\varphi \mid \overleftarrow{\mathsf{F}}\varphi\,,$$

where p is any atomic proposition. Naturally, we denote the associated class of languages **UTL\X**.

We now have:

Lemma 10. $\mathbf{UTL\backslash X} \subseteq \mathbf{CL}$: *every UTL\X formula can be encoded as a generalised cliquey Büchi automaton.*

Proof: We prove this lemma by constructing, for any UTL\X formula φ, a cliquey automaton A_φ that encodes it. Define the *closure* of φ to be the set $cl(\varphi)$ of all subformulas of φ and their negations, where we identify $\neg\neg\psi$ with ψ.

Say that $\mathbf{s} \subseteq cl(\varphi)$ is a *complete type* if (i) for each formula $\psi \in cl(\varphi)$ precisely one of ψ and $\neg\psi$ is a member of $\mathbf{s}$; (ii) $\psi_1 \wedge \psi_2 \in \mathbf{s}$ iff $\psi_1 \in \mathbf{s}$ and $\psi_2 \in \mathbf{s}$; (iii) $\psi \in \mathbf{s}$ implies $\mathsf{F}\psi \in \mathbf{s}$ and $\overleftarrow{\mathsf{F}}\psi \in \mathbf{s}$. Given types $\mathbf{s}$ and $\mathbf{t}$, write $\mathbf{s} \sim \mathbf{t}$ if $\mathbf{s}$ and $\mathbf{t}$ agree on all formulas whose outermost connective is a temporal operator, i.e., $\mathbf{s} \sim \mathbf{t}$ exactly if for all formulas ψ we have $\mathsf{F}\psi \in \mathbf{s}$ iff $\mathsf{F}\psi \in \mathbf{t}$, and $\overleftarrow{\mathsf{F}}\psi \in \mathbf{s}$ iff $\overleftarrow{\mathsf{F}}\psi \in \mathbf{t}$. Write tp_φ for the set of complete types for φ.

An ω-word w over alphabet 2^{AP} naturally extends to an ω-word $\overline{w} = \overline{w}_0\overline{w}_1\ldots$ over alphabet tp_φ, where $\overline{w}_i = \{\psi \in cl(\varphi) \,|\, (w,i) \models \psi\}$.

Recall that a generalised Büchi automaton has a family $\mathcal{A} = \{A_1,\ldots,A_n\}$ of accepting sets such that an accepting run must visit each A_i infinitely often. We define a generalised Büchi automaton $A_\varphi = (S, S_0, R, L, \mathcal{A})$ that accepts $\{w \in (2^{AP})^\omega \,|\, (w,0) \models \varphi\}$. The set of states is $S = tp_\varphi$, with the set S_0 of initial states consisting of those $\mathbf{s} \in tp_\varphi$ such that (i) $\varphi \in \mathbf{s}$ and (ii) $\overleftarrow{\mathsf{F}}\psi \in \mathbf{s}$ only if $\psi \in \mathbf{s}$. The state-labelling function $L\colon S \to \mathcal{B}(AP)$ is defined by $L(\mathbf{s}) = \bigwedge(\mathbf{s} \cap AP) \wedge \bigwedge\{\neg p \,|\, p \in AP \setminus \mathbf{s}\}$. The transition relation R consists of those pairs $(\mathbf{s},\mathbf{t})$ such that

(i) $\overleftarrow{\mathsf{F}}\psi \in \mathbf{t}$ iff either $\psi \in \mathbf{t}$ or $\overleftarrow{\mathsf{F}}\psi \in \mathbf{s}$,
(ii) $\mathsf{F}\psi \in \mathbf{s}$ and $\psi \notin \mathbf{s}$ implies $\mathsf{F}\psi \in \mathbf{t}$, and
(iii) $\neg\mathsf{F}\psi \in \mathbf{s}$ implies $\neg\mathsf{F}\psi \in \mathbf{t}$.

The accepting set family is $\mathcal{A} = \{A_{\mathsf{F}\psi} \,|\, \mathsf{F}\psi \in cl(\varphi)\}$, where $A_{\mathsf{F}\psi} = \{\mathbf{s} \,|\, \psi \in \mathbf{s}$ or $\mathsf{F}\psi \notin \mathbf{s}\}$. This completes the definition of A_φ.

We finally argue that automaton A_φ is cliquey: by the definition of the transition relation of A_φ, states $\mathbf{s}$ and $\mathbf{t}$ are in the same connected component iff $\mathbf{s} \sim \mathbf{t}$. Any two states $\mathbf{s}$ and $\mathbf{t}$ with $\mathbf{s} \sim \mathbf{t}$ are connected by a transition. □

Combining Theorem 5 and Lemma 10 yields one of our main results:

Theorem 11. *Every UTL\X formula admits a linear completeness threshold.*

Finally, one may wonder whether LTL\X formulas that have a cliquey representation are in fact always equivalent to some UTL\X formula. The answer is no, as our next result shows:

Lemma 12. $\mathbf{LTL\backslash X \cap CL \not\subseteq UTL\backslash X}$: *there exist LTL\X formulas that do have a cliquey representation yet are not equivalent to any UTL\X formula.*

Proof (sketch)**:** Let a, b, c be distinct elements of 2^{AP}, and consider the language $L = (a + b + c)^*.a.a^*.b.(a + b + c)^\omega$. L is captured by the LTL\X formula $\mathsf{F}(a \wedge (a \,\mathsf{U}\, b))$, and it is also clear that L is cliquey. Using the results of [17], one can show that this language is inexpressible in UTL (let alone UTL\X). For example, one can compute the syntactic monoid associated with L and invoke the characterisation of syntactic monoids of UTL-definable languages from [17] to obtain the desired result. We omit the details. □

Figure 2 summarises our expressiveness results. All inclusions are strict.

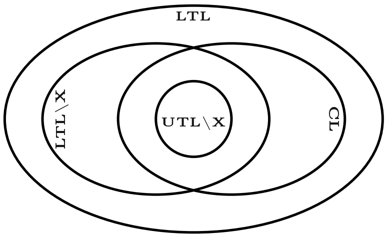

Fig. 2. Relationships among various classes of ω-regular languages

5 Beyond Cliqueyness

Two natural questions arise as to whether cliqueyness is necessary in order to achieve a linear completeness threshold, and whether there actually are any ω-regular languages that fail to have linear completeness thresholds. We answer the first question negatively and the second one positively. In fact, we show that ω-regular languages can be engineered to have completeness thresholds bounded below in the worst case by superpolynomial and even exponential functions of the recurrence diameter of Kripke structures.

5.1 Linear Completeness Thresholds without Cliqueyness

Consider the Büchi automaton B depicted in Figure 1 (b). It is clearly not cliquey and is in fact *semantically* non-cliquey, i.e., not equivalent to any cliquey Büchi automaton. To see this, observe that B accepts the word $w := (\{p\}.\{q\}.\{r\})^\omega$, yet no word with the substring $\{q\}.\{p\}$. By Property 9, B cannot be equivalent to a cliquey automaton.

We claim that B nonetheless has a linear completeness threshold: namely, for any Kripke structure M, $\mathit{sap}(M \times B) \leq \mathit{rd}(M)+1$. Indeed, if an SAP had length greater than $\mathit{rd}(M)+1$, its projection onto M would have to exhibit an 'inner' loop that for some reason could not be cut out. A straightforward case analysis then quickly leads to a contradiction.

5.2 Büchi Automata with Non-linear Completeness Thresholds

On the other hand, not every LTL formula and in fact not every LTL\X formula has a linear completeness threshold. Consider the non-cliquey automaton B in Figure 3, which encodes the LTL\X formula

$$\begin{aligned}\varphi \;=\; p \wedge \neg r \wedge \mathsf{G}\;(\;(p \wedge \neg r) &\Rightarrow (\;(p \wedge \neg r)\;\mathsf{U}\;(q \wedge \neg r)\;) \;\wedge \\ (q \wedge \neg r) &\Rightarrow (\;(q \wedge \neg r)\;\mathsf{U}\;r!\;) \;\wedge \\ r! &\Rightarrow (\;r!\;\mathsf{U}\;(p \wedge \neg r) \vee \mathsf{G}\,r!)\;)\,.\end{aligned}$$

Again, the notation $r!$ is short for $r \wedge \neg p \wedge \neg q$.

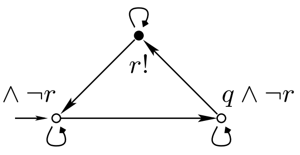

Fig. 3. A non-cliquey Büchi automaton with superpolynomial completeness threshold

To show that B has no linear completeness threshold, we construct a collection $(M_i)_{i=1}^{\infty}$ of Kripke structures such that, for each i, we have $\mathit{sap}(M_i \times B) \geq i/4 \cdot \mathit{rd}(M_i)$.

The construction is depicted in Figure 4. M_i contains i copies of a q-labelled loop, 'q-loop' for short. Each q-loop comprises i states. Consecutive occurrences of the q-loop are connected via an r-labelled state, or r-state for short. The final r-state has a self-loop.

Let us compute M_i's recurrence diameter: a longest loop-free path starts at the first state of the first q-loop (a successor of the first p,q-state), follows that loop around to the first p,q-state, then follows the baseline path—skipping all intermediate q-loops—all the way to the final q-loop. It then enters that loop and follows it to the last of its states (*before* the loop is closed). M_i thus has a recurrence diameter of at most $4i$.

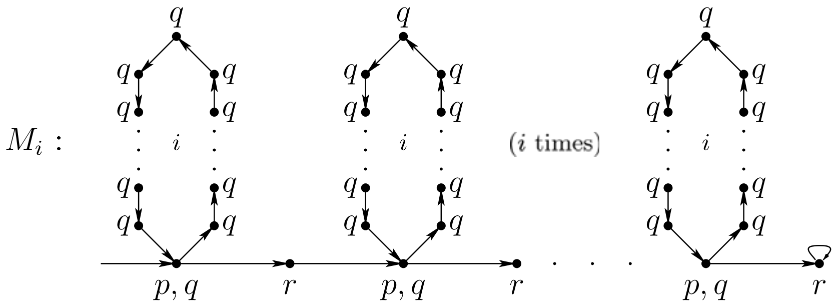

Fig. 4. Kripke structure family $(M_i)_{i=1}^{\infty}$ witnessing a non-linear completeness threshold

An SAP of $\Pi = M \times B$, however, must take **all** q-loops. To see this, consider the initial state of Π, which is labelled $(\{p, q\}, p \wedge \neg r)$. B does not allow an r-state as successor (both possible transitions [one of which is a B-self-loop] require successors satisfying $\neg r$). Thus the joint path must enter the first q-loop. During this loop, B stays in the $(q \wedge \neg r)$-state, up to and including the time when M finishes the loop and arrives back at the p, q-state. At this time the shortest path continues at the state labelled $(\{r\}, r!)$, followed by the state labelled $(\{p, q\}, p \wedge \neg r)$, at which point it is forced into the next q-loop of M_i. Note that, for this path to be accepting, it has to visit an r-state of M_i infinitely often, which is only possible via the self-loop reachable *after* all the q-loops have been taken.

Having to go through i loops each of size i, an SAP of Π has length at least i^2. Combining this with the size of the recurrence diameter of at most $4i$, we see that the completeness threshold for B is at least *quadratic* in the recurrence diameter of Kripke structures. □

It is not difficult to see our family of Kripke structures can in fact be modified to exhibit a *cubic* completeness threshold for our very same automaton B, by modifying the loops slightly and grafting a further additional family of loops onto each of them. In this vein, one sees that completeness thresholds exceeding any given polynomial can in fact be achieved, so that our formula φ and Büchi automaton B have *superpolynomial* completeness threshold.

In fact, even *exponential* completeness thresholds can be achieved for LTL formulas.[4] Consider a family of Kripke structures, each of which resembles a full binary tree, with bidirectional edges between every parent and child. The recurrence diameter of any such structure is the length of a longest loop-free path from one leaf to another, and is therefore logarithmic in the size of the structure. These structures can however be instrumented in such a way that a certain LTL formula forces the unique accepting path to perform a depth-first traversal of the entire tree, resulting in a path of length exponential in the recurrence diameter. To achieve this, atomic propositions are used to keep track of the depth of nodes modulo 3, and further propositions label the root, leaves, and left and right

[4] We are grateful to one of the anonymous referees for this observation.

children accordingly. A traversal of the tree is then orchestrated by requiring that (i) whenever an interior node is entered from above (which is determined by knowledge of the depths modulo 3 of the present node and that of the previous one), then the left child should be visited next; (ii) whenever a non-leaf node is returned to from a left child, then the right child should be visited next; and (iii) whenever a non-leaf node is returned to from a right child, then the parent node should be visited next. Finally, the rightmost leaf is labelled with a special proposition which the formula requires to hold eventually.

6 Concluding Remarks

We have presented a method for calculating fairly tight, linear completeness thresholds for a large class of LTL specifications. The algorithm we propose is highly efficient, running in time linear in the size of the Büchi automaton. Several potential bottlenecks however remain, including the following two:

- Computing the diameter and recurrence diameter of a large Kripke structure can be computationally prohibitive; one possible remedy might be to settle for tractable over-approximations of the diameters, as in [2], in a trade-off which would likely require careful consideration.
- It has often been empirically observed that bounded model checking computations tend not to scale up very well. Since many Kripke structures have deep recurrence diameters (of the order of the total number of states, for example), one can expect that exploring the system to the required depth prove in certain cases to be intractable.

Nonetheless, this is an area of active research in which progress is being made on several fronts. Our hope is that the techniques presented here may prove beneficial not only to practitioners, but also to other researchers whose technology it might potentially complement.

Alongside these practical considerations, two interesting theoretical questions arise: (i) is it decidable whether a given LTL formula (or more generally a given ω-regular language) has a linear completeness threshold; and (ii) is the completeness threshold of an ω-regular language always either linear or superpolynomial? We leave these questions as further research.

References

1. Awedh, M., Somenzi, F.: Proving more properties with bounded model checking. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 96–108. Springer, Heidelberg (2004)
2. Baumgartner, J., Kuehlmann, A., Abraham, J.A.: Property checking via structural analysis. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, p. 151. Springer, Heidelberg (2002)
3. Biere, A., Cimatti, A., Clarke, E., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers 58, 118–149 (2003)

4. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, p. 193. Springer, Heidelberg (1999)
5. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (2000)
6. Clarke, E., Kröning, D., Ouaknine, J., Strichman, O.: Completeness and complexity of bounded model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 85–96. Springer, Heidelberg (2004)
7. Clarke, E.M., Allen Emerson, E., Sifakis, J.: Model checking: Algorithmic verification and debugging. CACM 52(11), 75–84 (2008)
8. Ganai, M., Gupta, A., Ashar, P.: Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In: ICCAD, pp. 510–517 (2004)
9. Gerth, R., Peled, D., Vardi, M., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: PSTV, pp. 3–18 (1995)
10. Kamp, H.: Tense Logic and the Theory of Linear Order. PhD thesis, University of California (1968)
11. Lamport, L.: What good is temporal logic. In: IFIP Congress, pp. 657–668 (1983)
12. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems—Specification. Springer, Heidelberg (1991)
13. McMillan, K.L.: Applying SAT methods in unbounded symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, p. 250. Springer, Heidelberg (2002)
14. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)
15. Schützenberger, M.-P.: On finite monoids having only trivial subgroups. Information and Control 8(2), 190–194 (1965)
16. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 108–125. Springer, Heidelberg (2000)
17. Thérien, D., Wilke, T.: Over words, two variables are as powerful as one quantifier alternation. In: STOC, pp. 234–240 (1998)