A New Approach to Bounded Model Checking for Branching Time Logics

Rotem Oshman and Orna Grumberg

Technion – Israel Institute of Technology
Department of Computer Science

Abstract. Bounded model checking (BMC) is a technique for overcoming the state explosion problem which has gained wide industrial acceptance. Bounded model checking is typically applied only for linear-time properties, with a few exceptions, which search for a counter-example in the form of a tree-like structure with a pre-determined shape. We suggest a new approach to bounded model checking for universal branching-time logic, in which we encode an arbitrary graph and allow the SAT solver to choose both the states and edges of the graph. This significantly reduces the size of the counter-example produced by BMC.
A dynamic completeness criterion is presented which can be used to halt the bounded model checking when it becomes clear that no counter-example can exist. Thus, verification of the checked property can also be achieved. Experiments show that our approach outperforms another recent encoding for μ-calculus on complex ACTL properties.

1 Introduction

Bounded model-checking (BMC) is a model-checking method that has gained popularity due to the inability of BDD-based symbolic model-checkers to handle large designs. In classical BMC [3], one tries to find a bug of bounded length k. If a bug is not found, the bound is increased until either a bug is found or a pre-determined *completeness threshold* [4] is reached. If the threshold has been reached but no bug has been found, it is concluded that the formula holds in the model. In practice, the threshold is rarely reached, but recent works (e.g., [6]) also describe techniques for SAT-based temporal induction which can be used to prove formulas without reaching the completeness threshold.

BMC has mostly been restricted to linear-time specifications, with a few exceptions ([14], [16]). Most encodings for linear-time logic have a common form, a conjunction of two formulas: one encodes a path starting from an initial state of the model, and the second is property-dependent and constrains the path to be a counter-example to the property being checked.

Bounded model-checking for branching-time logic is a somewhat thornier problem, because it is usually not known in advance what exact shape the counter-example will take. The works that extended the BMC paradigm to universal branching-time logic dealt with this problem in different ways: [14] encodes a property-dependent number of bounded paths, either lasso-shaped or finite, and constrains them to represent a counter-example; [16] encodes a bounded

proof-tree for the negation of the formula, using the local proof rules of [15] for μ-calculus. Both approaches assume a worst-case scenario in the construction of the tree-like structure they encode, with the result that many more states may be encoded than are necessary for the counter-example. For example, to disprove a formula of the form $\psi_1 \vee \psi_2$, both approaches will encode two separate tree-like structures — one for disproving ψ_1 and one for disproving ψ_2. In practice, there may exist counter-examples for both ψ_1 and ψ_2 which share many model states. The counter-examples returned in [14] and [16] are therefore not minimal in the number of states they contain.

In this paper we suggest a new approach to bounded model-checking for universal branching-time logic wherein we encode exactly the states that are necessary for the counter-example. Unlike [14] and [16], we make no assumptions about the structure of the counter-example; we encode k states, where k is the bound, and allow the SAT solver to choose both the states and the edges of the model that will comprise the counter-example. We use the *local constraints* of Namjoshi's proof system for μ-calculus [13] to ensure that the structure represented by the states is a counter-example to the formula being checked. Our approach ensures a *minimal* counter-example, and it avoids representing the same model-state more than once.

We present an encoding for proving existential μ-calculus properties (or falsifying universal properties), using alternating parity tree automata as the specification mechanism. We also present a simplified variation of the encoding for alternation-free existential μ-calculus. The encoding for full existential μ-calculus uses roughly $O(|Q| \cdot k \log k)$ variables, where k is the bound and $|Q|$ is the number of automaton states. The simpler encoding for alternation-free μ-calculus is less compact, requiring $O(|Q| \cdot k^2)$ variables, but it is more explicit and performs better than the more general encoding. The simplified encoding can be extended to handle fairness constraints while still requiring roughly $O(|Q| \cdot k^2)$ variables.

We also describe a dynamic termination criterion which can be used to halt the bounded model-checking by determining that no counter-example comprising $k' > k$ states can exist, where k is the current bound. The idea is similar to the criterion suggested in [16]: we attempt to identify situations where the structure encoded cannot be extended by adding new states (that is, increasing the bound). However, our implementation is quite different, due to the difference between the encodings. Using the termination criterion it is possible to prove and disprove both existential and universal formulas. As is typical for bounded model-checking, the algorithm performs better when proving existential formulas or disproving universal ones than when proving universal formulas or disproving existential ones.

Finally, we present experimental results for the branching-time logic ACTL. Our experiments show that our approach is a good complement to the encoding of [16], especially for complex formulas with a large nesting depth, where our encodings can often disprove formulas that cannot be disproven by the encoding of [16]. Deeply-nested formulas are generated during automatic translation to ACTL from a high-level specification language, e.g., PSL [1], where complex regular expressions translate into deeply-nested ACTL formulas.

2 Preliminaries

2.1 Alternating parity tree automata

A *normal-form alternating parity tree automaton* [17] is a tuple $A = (AP, Q, q_0, \delta, \Omega)$, where AP is a set of atomic propositions, Q is the set of automaton states, and q_0 is the initial state; δ is an alternating transition relation, assigning to each state $q \in Q$ a transition of the form $q_1 \vee q_2$, $q_1 \wedge q_2$, $\Diamond q_1$, $\Box q_1$, p or $\neg p$, where $q_1, q_2 \in Q$ and $p \in AP$; and finally, $\Omega : Q \to \mathbb{N}$ is a partial priority function which represents a *parity acceptance condition* (in [13], the acceptance condition is represented as a partition of Q instead of a priority function). For an automaton state $q \in Q$, $\Omega(q)$ will be called the *priority* of q. The automata we will deal with contain no cycles of priorityless states. We will say that an infinite sequence $\pi = q_0 q_1 \ldots \in Q^\omega$ *satisfies* Ω if the lowest priority $\Omega(q)$ of a state q that has a priority and appears infinitely often in π is even. (An alternative definition, e.g. in [17], requires that the infinite sequence also have an infinite number of states for which the priority is defined. However, automata constructed using the standard translation from μ-calculus have at least one state that has a priority on every cycle in the automaton; all infinite sequences contain an infinite number of states that have a priority. We assume this property in the automaton representing the specification we check.)

Universal μ-calculus properties [11] can be expressed by automata that do not have $\Diamond$-transitions. We will refer to such an automaton as a $\Box$-automaton. Similarly, existential μ-calculus properties can be expressed by $\Diamond$-automata, which have no $\Box$-transitions.

Tree automata run over labeled trees. A *labeled tree* is a pair $T = (N, L)$ where $N \subseteq \mathbb{N}^+$ is a prefix-closed set of tree nodes and $L : N \to 2^{AP}$ is a labeling function. The node ε (the empty word) is the tree root, and there is an edge from node n_1 to node n_2 iff $n_2 = n_1 \cdot i$ for some $i \in \mathbb{N}$. We will use $Succ(n)$ to denote the targets of edges outgoing from n; that is, $Succ(n) = \{n \cdot i \mid i \in \mathbb{N}\} \cap N$.

The acceptance of a tree by an automaton is defined in terms of a two-player infinite game. The game positions are $N \times Q$, and the initial position is (ε, q_0). The player who owns the position (n, q) and the moves available to that player are determined according to δ: player I owns positions (n, q) such that $\delta(q) = q_1 \vee q_2$ or $\delta(q) = \Diamond q_1$; player II owns positions (n, q) such that $\delta(q) = q_1 \wedge q_2$ or $\delta(q) = \Box q_1$. If $\delta(q) = q_1 \vee q_2$ or $\delta(q) = q_1 \wedge q_2$, the available moves are (n, q_1) and (n, q_2); if $\delta(q) = \Diamond q_1$ or $\delta(q) = \Box q_1$, the available moves are (m, q_1) for all $m \in Succ(n)$. A position (n, q) is winning for player I if $\delta(q) = p$ and $p \in L(n)$ or $\delta(q) = \neg p$ and $p \notin L(n)$, and winning for player II if $\delta(q) = p$ and $p \notin L(n)$ or $\delta(q) = \neg p$ and $p \in L(n)$. A play is winning for player I if it is finite and ends in a position that is winning for player I, or if it is infinite and satisfies Ω; otherwise, the play is winning for player II. Strategies are defined as usual: a *strategy* for player x is a partial function mapping finite sequences of configurations to a choice of the next configuration at every position owned by player x. A play is said to be *according to* a strategy for player x if every choice made by player x in the play conforms to the strategy. A strategy is *winning* for player x if player x wins any play she plays according to the strategy. We will

say that an automaton A *accepts* a tree T iff player I has a winning strategy for the game thus described.

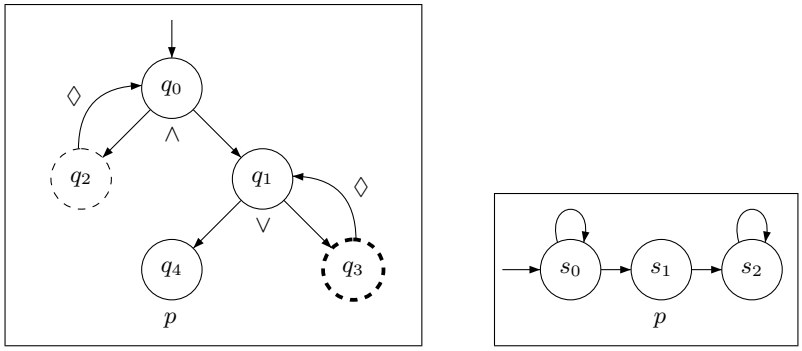
2.2 Kripke structures

To represent finite-state programs, we use *Kripke structures*. A Kripke structure is a tuple $M = (S, s_0, R, L)$, where S is the set of states, s_0 is the initial state, $R \subseteq S \times S$ is a total transition relation and $L : S \to 2^{AP}$ is a labeling function.

Given a Kripke structure M and an alternating parity tree automaton A, we will say that M *satisfies* A iff the computation tree of M is accepted by A. We will say that a model state $s \in S$ satisfies an automaton state q if the computation tree starting from s is accepted by the automaton A' which is identical to A except that q is the initial state of A'.

Theorem 1 ([9], [17]). *For every mu-calculus formula φ there exists an alternating parity tree automaton A_φ such that for every Kripke structure M, M satisfies A_φ iff $M \models \varphi$.*

Example 1. Consider the automaton $A = (\{p\}, \{q_0, q_1, q_2, q_3, q_4\}, q_0, \delta, \Omega)$ shown in Fig.1, where the type of the transition is indicated below each state or on the relevant edge, and Ω is defined only for q_2 and q_3 (shown dashed), which have $\Omega(q_2) = 2$ and $\Omega(q_3) = 1$.



(a) An automaton for $EGEFp$

(b) A Kripke structure satisfying $EGEFp$

Fig. 1. An example automaton and Kripke structure

The automaton is equivalent to the property "there exists a path on which from every state, p is reachable", expressed as $EGEFp$ in the temporal logic ECTL. State q_0 stands for $EGEFp$, and state q_2 stands for $EXEGEFp$, "there exists a successor that satisfies $EGEFp$". Similarly, q_1 stands for EFp and q_3 for $EXEFp$, and state q_4 stands for p. The odd priority for q_3 requires that a winning play only pass through q_3 a finite number of times (since there is no state with a lower even priority), so that eventually the play must transition to q_4, which requires that p be satisfied at the current model state.

2.3 Namjoshi-style temporal proofs

Several proof systems have been suggested for the model checking problem of μ-calculus. Here we focus on the proof system presented by Namjoshi in [13]. The feature which makes it useful for our purposes is that its conditions are local: to verify that a proof is valid, one need only check a series of *local* conditions, which refer at most to a state's immediate successors in the Kripke structure. Other proof systems, such as Stirling's proof rules [15], keep track of states visited along the current proof branch; in Namjoshi's system such "book-keeping" is not required, and ranks are used instead.

In [13], the proof system is presented for automata where the priority function is full. We will present the system from [13] and then explain how it can be extended to the case where the priority function is a partial function.

Let $M = (S, s_0, R, L)$ be a Kripke structure, and let $A = (AP, Q, q_0, \delta, \Omega)$ be a normal-form alternating parity tree automaton with Ω defined for all $q \in Q$. To show that M satisfies A, one must exhibit: (i) for each automaton state $q \in Q$, a predicate I_q, which, intuitively, characterises the set of model states which satisfy q; (ii) non-empty, well-founded sets $W_1, \ldots, W_m$, where m is the number of odd priorities assigned by Ω to states from Q, and pre-orders $\prec_1, \ldots, \prec_m$; (iii) for each automaton state $q \in Q$, a partial rank function $\rho_q : S \to (W, \prec)$, where $W = W_1 \times \ldots \times W_m$ and $\prec$ is the lexicographic order induced by $\prec_1, \ldots, \prec_m$ on W. In this paper, we will assume without loss of generality that $W = \mathbb{N}^k$, with $\prec_i$ the standard order $<$ over $\mathbb{N}$. We will henceforth omit W and simply write $\mathit{\Pi} = (I, \rho)$, where $I = \{I_q \mid q \in Q\}$ is the set of invariants and $\rho = \{\rho_q \mid q \in Q\}$ is the set of rank functions.

We use Invariance and Progress obligations to ensure that player I has a winning strategy for the game induced by A on the computation tree of M: the obligation for automaton states q with $\vee$- or $\Diamond$-transitions represents the move player I must make in positions (s, q) owned by her. Obligations for states q with $\wedge$- or $\Box$-transitions ensure that no matter which move player II makes from a position (n, q), player I will have a winning strategy from the resulting position. In the case of an infinite play, we use ranks to ensure that the play satisfies Ω.

Intuitively, the rank $\rho_q(s)$ represents a commitment regarding the number of times we may pass through states with each odd priority before passing through a state with lower priority in a play from position (n, q), where n is a tree node corresponding to model state s. For example, coordinate 0 of the rank counts the number of times we may pass through states with priority 1 before passing through a state with priority 0. Each time we pass through a state with priority $2i + 1$, coordinates 0 through i decrease lexicographically, and can only increase again when passing through a state q with $\Omega(q) < 2i + 1$. A play according to the strategy induced by the invariants can only pass through a state with an odd priority $2i + 1$ a finite number of times before coordinates $0, \ldots, i$ of the rank reach zero, and then we must pass through a state with lower priority. The lowest priority occurring infinitely often in the play must be even, and player I wins.

This notion is captured by an order $\lhd_q$ over $\mathbb{N}^k$, defined for each $q \in Q$ as follows: $(x_0, \ldots, x_{m-1}) \lhd_q (y_0, \ldots, y_{m-1})$ iff $\Omega(q) = 0$, or $\Omega(q) = 2i, i > 0$ and

$(x_0, \ldots, x_i, 0, \ldots, 0) \preceq (y_0, \ldots, y_i, 0, \ldots, 0)$, or $\Omega(q) = 2i{+}1$ and $(x_0, \ldots, x_i, 0, \ldots, 0) \prec (y_0, \ldots, y_i, 0, \ldots, 0)$. Note that coordinates $i, \ldots, m-1$ are unconstrained when passing through a state with priority $\Omega(q) < 2i+1$, but when passing through states with $\Omega(q) \geq 2i+1$, coordinate i may not increase. When passing through a state with priority $2i+1$, coordinates $0, \ldots, i$ must decrease in lexicographic order.

A valid proof must satisfy the following requirements.

- Consistency: for each $q \in Q$ and $s \in I_q$, $\rho_q(s)$ is defined.
- Initiality: $s_0 \in I_{q_0}$.
- Invariance and Progress: for each $q \in Q$ and $s \in I_q$:
  - If $\delta(q) = p$ then $p \in L(s)$.
  - If $\delta(q) = \neg p$ then $p \notin L(s)$.
  - If $\delta(q) = q_1 \vee q_2$, then either $s \in I_{q_1}$ and $\rho_{q_1}(s) \lhd_q \rho_q(s)$, or $s \in I_{q_2}$ and $\rho_{q_2}(s) \lhd_q \rho_q(s)$.
  - If $\delta(q) = q_1 \wedge q_2$, then $s \in I_{q_1}$ and $\rho_{q_1}(s) \lhd_q \rho_q(s)$, and also $s \in I_{q_2}$ and $\rho_{q_2}(s) \lhd_q \rho_q(s)$.
  - If $\delta(q) = \Diamond q_1$, then there exists $t \in S$ such that $(s,t) \in R$ and $t \in I_{q_1}$ and $\rho_{q_1}(t) \lhd_q \rho_q(s)$.
  - If $\delta(q) = \Box q_1$, then for all $t \in S$ such that $(s,t) \in R$, $t \in I_{q_1}$ and $\rho_{q_1}(t) \lhd_q \rho_q(s)$.

Theorem 2 ([13]). *For every Kripke structure M and automaton A with a full priority function, M satisfies A iff there exists a Namjoshi-style proof showing that M satisfies A.*

Automata resulting from the standard translation for μ-calculus have a *partial* priority function, with infinitely many priorities on every infinite path. For such automata, we would still like the $\lhd_q$ relation to enforce the parity acceptance condition, which now concerns only states that have a priority. Define an extension $\circledcirc_q$ as follows: $x \circledcirc_q y$ iff $\Omega(q)$ is defined and $x \lhd_q y$, or $\Omega(q)$ is undefined and $x = y$. The idea is that priorityless states should simply preserve the rank, keeping it unchanged until the next time we pass through a state with a priority.

Lemma 1. *The proof system obtained by replacing $\lhd_q$ with $\circledcirc_q$ is sound and complete for all automata with no cycles of priorityless states.*

Example 2. Consider the automaton and structure shown in Fig.1. A proof showing that M satisfies A is given by $\Pi = (I, \rho)$, where $I_{q_0} = I_{q_2} = \{s_0\}$ (states that satisfy $EGEFp$ and $EXEGEFp$), $I_{q_1} = I_{q_3} = \{s_0, s_1\}$ (states that satisfy EFp and $EXEFp$), and $I_{q_4} = \{s_1\}$ (the only state labeled with p). The ranks in Π have a single coordinate, representing the length of a path to a state satisfying p. The relation $\circledcirc_q$ is $=$ for all $q \neq q_3$, and for q_3, $\circledcirc_{q_3}$ is $<$. An assignment of ranks that satisfies the proof obligations is $\rho_q(s_0) = 1$, $\rho_q(s_1) = 0$ and $\rho_q(s_2)$ undefined for all $q \in Q$. Note that since we attached the decrease in rank to an automaton state with a $\Diamond$-transition, all the automaton states agree on the rank assigned to each model state. This can be done for all ECTL formulas.

2.4 Notation and terminology

We will let $Q_\Diamond$ denote the set of automaton states q with a transition $\delta(q) = \Diamond q_1$. For automaton states $q \in Q_\Diamond$ and model states $s \in I_q$, it will sometimes be useful to identify the model state (or one of the model states) $t \in S$ which serves to satisfy the Invariance and Progress obligation for s and q in a proof Π. We will refer to t as a *proof successor* for s as required by q.

3 The encodings

We present two encodings to SAT for model checking existential alternating parity tree automata (with no $\Box$-transitions). In both encodings, we search for a counter-example of bounded size k in the Kripke structure, where k is the number of states in the counter-example; the counter-example is represented as an arbitrary graph of unknown structure, and each state in the graph must satisfy certain local obligations to ensure that the graph constitutes a counter-example for the formula in question. The structure of the graph is determined by the local obligations of each state.

3.1 Encoding Namjoshi-style proof obligations

The first encoding we present is a direct translation of the proof obligations of a Namjoshi-style temporal proof to Boolean constraints.

Let $M = (S, s_0, R, L)$ be a Kripke structure. We assume that the initial state and the transition relations are given in the form of propositional formulas I and R respectively, and that the state space S is represented by $\{0,1\}^n$. Also, for each atomic proposition $p \in AP$, we assume a propositional formula $\mathsf{L_p}$ which is true exactly for states $s \in S$ such that $p \in L(s)$. Let $A = (AP, Q, q_0, \delta, \Omega)$ be a $\Diamond$-automaton. To encode the requirements on ranks, we use a set of propositional formulas $\mathsf{LT_q}$ for all $q \in Q$, such that $\mathsf{LT}_q(\sigma_1, \sigma_2)$ holds iff $\sigma_1 \lessdot_q \sigma_2$.

The encoding uses the following variables.

- $u_0, \ldots, u_{k-1}$: vectors representing model states. Each vector comprises n bits.
- x_i^q for each $i = 0, \ldots, k-1$ and $q \in Q$: an indicator variable for the fact that the state assigned to u_i satisfies q.
- ρ_i^q for each $i = 0, \ldots, k-1$: a vector representing the rank $\rho_q(s)$ assigned to u_i by q.

Each rank vector ρ_i^q has m coordinates, where m is the number of odd priorities assigned by Ω to automaton states, and each coordinate j comprises $\log |Q|k$ bits. This is sufficient because if there exists an infinite winning play for player I, then there exists a play that does not pass through an odd-priority state twice before passing through a state with a lower priority. The total number of variables used in the encoding is $O(nk + k|Q| + |Q|mk \log |Q|k)$.

The obligations for a state u_i and an automaton state q are encoded as a Boolean formula of the form $x_i^q \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i$, where $\langle\!\langle \delta(q) \rangle\!\rangle_i$ is defined as follows.

$$\begin{aligned}
&\langle\!\langle p \rangle\!\rangle_i = \mathsf{L_p}(u_i) \\
&\langle\!\langle \neg p \rangle\!\rangle_i = \neg \mathsf{L_p}(u_i) \\
&\langle\!\langle q_1 \wedge q_2 \rangle\!\rangle_i = x_i^{q_1} \wedge \mathsf{LT_q}(\rho_i^{q_1}, \rho_i^q) \wedge x_i^{q_2} \wedge \mathsf{LT_q}(\rho_i^{q_2}, \rho_i^q) \\
&\langle\!\langle q_1 \vee q_2 \rangle\!\rangle_i = (x_i^{q_1} \wedge \mathsf{LT_q}(\rho_i^{q_1}, \rho_i^q) \vee x_i^{q_2}) \wedge (\mathsf{LT_q}(\rho_i^{q_2}, \rho_i^q)) \\
&\langle\!\langle \Diamond q_1 \rangle\!\rangle_i = \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_i, u_j) \wedge x_j^{q_1} \wedge \mathsf{LT_q}(\rho_j^{q_1}, \rho_i^q)\right)
\end{aligned}$$

It is possible to eliminate the indicators x_i^q when $\delta(q) = q_1 \wedge q_2$ or $\delta(q) = q_1 \vee q_2$ by substituting the constraints generated for these formulas anywhere that the indicator appears.

To represent the Initiality requirement, we add the constraint $\mathsf{I}(u_0) \wedge x_0^{q_0}$. The resulting formula is given by

$$\mathsf{PRF}^1_{M,A,k} = \mathsf{I}(u_0) \wedge x_0^{q_0} \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{q \in Q} x_i^q \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i$$

Optimizating the encoding. The encoding presented above is naive, and can be improved in several ways.

First, the encoding suffers from symmetry, which has an adverse effect on the performance of most SAT solvers; the model states $u_1, \ldots, u_{k-1}$ are interchangeable, and the SAT solver is forced to consider many equivalent permutations of the same counter-example before eliminating it. We have found that performance is greatly improved when we break the symmetry by ordering the states $u_1, \ldots, u_{k-1}$, obtaining the formula

$$\mathsf{PRF}^{1'}_{M,A,k} = \mathsf{PRF}^1_{M,A,k} \wedge \bigwedge_{i=1}^{k-2} u_i < u_{i+1}$$

where "$<$" is implemented as the lexicographic order on binary vectors. (u_0 is excluded from the ordering as it is the only state that serves a "special" role: it must be an initial state, and we cannot require that it be smaller than all the other states.)

The way ranks are handled in the encoding can also be improved. By analyzing the structure of the automaton, we can identify sets of automaton states that can share the same rank vectors. For example, if $\delta(q) = q_1 \wedge q_2$ and q does not have a priority, then in a valid proof, $\rho_q(s) = \rho_{q_1}(s) = \rho_{q_2}(s)$ for any model state s. There is no need to encode the rank separately, and instead of having three vectors ρ_i^q, $\rho_i^{q_1}$ and $\rho_i^{q_2}$ all three automaton states can "share" a vector $\rho_i^{\{q,q_1,q_2\}}$. This also simplifies the constraint $\langle\!\langle \delta(q) \rangle\!\rangle_i$, because now we can remove the $\mathsf{LT_q}$ constraint; it is implicit in using the same rank vector.

In particular, for ECTL formulas it is possible to construct automata where *all* the automaton states share a single rank vector ρ_i^Q, greatly simplifying the encoding. For lack of space, we do not elaborate.

Encoding successor states explicitly. In the constraints generated for states $q \in Q_\Diamond$, the transition relation appears k times each. Since the transition relation is often complicated, it is desirable to decrease the number of times it appears. We can do so at the cost of increasing the number of variables, by encoding proof-successors explicitly. For each $q \in Q_\Diamond$ and $i = 0, \ldots, k-1$, we will assign a vector t_i^q to represent the successor required by q if the state assigned to u_i is in I_q. Since we are searching for a proof of size k, t_i^q will be constrained to be one of the states $u_0, \ldots, u_{k-1}$; also, if $t_i^q = u_j$, then we require u_j to be in the appropriate invariant I_{q_1}, where $\delta(q) = \Diamond q_1$, and its rank must behave appropriately. The constraint can now be written as

$$\langle\!\langle \Diamond q_1 \rangle\!\rangle_i = \mathsf{R}(u_i, t_i^q) \wedge \bigvee_{j=0}^{k-1} \left(t_i^q = u_j \wedge x_j^{q_1} \wedge \mathsf{LT_q}(\rho_j^{q_1}, \rho_i^q) \right)$$

In the new encoding, the transition relation appears $k \cdot |Q_\Diamond|$ times instead of $k^2 \cdot |Q_\Diamond|$ times as before. We will also have further use for the information we gain by explicitly encoding proof successors in constructing a dynamic completeness criterion (Section 4).

3.2 Eliminating the use of ranks

Although the previous encoding uses a rather small number of variables, which increases as $O(k \log k)$ with the bound k, the use of ranks can be SAT-unfriendly. The second encoding we present is similar to the first, but using ideas from [10] and [8], we eliminate the use of ranks. The idea is that instead of directly encoding the rank $\rho_q(s)$ for states $s \in I_q$, we will store a subset I_q^σ for each rank σ, containing states $s \in I_q$ that have $\rho_q(s) \leq \sigma$. In the encoding, we will unroll the proof obligations once for each such invariant; when a decrease in rank is called for, we will use the subset that represents the lower rank. This encoding becomes inefficient when the ranks have more than one coordinate, and we will restrict attention to automata that only assign the priorities $1, 2$. Such automata require a single coordinate in the rank, and includes the alternation-free fragment of μ-calculus.

The encoding will use the following variables.

- $u_0, \ldots, u_{k-1}$: vectors representing model states.
- $x_i^{q,t}$ for each $i = 0, \ldots, k-1$, $q \in Q$ and $t = 0, \ldots, mk$ where m is the number of odd-priority automaton states: an indicator variable for the fact that the state assigned to u_i satisfies q and has rank no greater than t.

Our obligations will now take the form $x_i^{q,t} \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i^t$ for $t > 0$, where $\langle\!\langle \delta(q) \rangle\!\rangle_i^t$ is defined by

$$\begin{aligned}
\langle\!\langle p \rangle\!\rangle_i^t &= \mathsf{L_p}(u_i) \\
\langle\!\langle \neg p \rangle\!\rangle_i^t &= \neg \mathsf{L_p}(u_i) \\
\langle\!\langle q_1 \wedge q_2 \rangle\!\rangle_i^t &= x_i^{q_1, r_q(t)} \wedge x_i^{q_2, r_q(t)} \\
\langle\!\langle q_1 \vee q_2 \rangle\!\rangle_i^t &= x_i^{q_1, r_q(t)} \vee x_i^{q_2, r_q(t)} \\
\langle\!\langle \Diamond q_1 \rangle\!\rangle_i^t &= \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_i, u_j) \wedge x_j^{q_1, r_q(t)} \right)
\end{aligned}$$

where $r_q(t) = t$ if $\Omega(q) = 2$ or $\Omega(q)$ is not defined, and $r_q(t) = t - 1$ if $\Omega(q) = 1$. For $t = 0$, we will constrain $x_i^{q,0} \rightarrow$ **false** (or just substitute **false** where the indicator appears).

The optimizations for the previous encoding can be applied here as well. The encoding can be further optimized for ECTL by exploiting the weak structure of the automata along the lines of [8]. Also, although ECTL formulas with fairness cannot always be described by automata that only assign the priorities 1 and 2, it is not difficult to extend the encoding for ECTL to handle fairness, by imitating the way a symbolic model-checker for ECTL handles fairness constraints.

Theorem 3. *Given a Kripke structure M and a $\Diamond$-automaton A, the formulas generated by the encodings of Sections 3.1 and 3.2 are satisfiable iff there exists a proof $\Pi = (I, \rho)$ showing that M satisfies A that contains k states; that is, $\left| \bigcup_{q \in Q} I_q \right| = k$.*

4 A dynamic completeness criterion

Both encodings presented in the previous section provide a way to determine when a Kripke structure does *not* satisfy a $\Box$-automaton A: construct the complement $A_\neg$ for A, choose a bound k, and if a SAT solver returns a satisfying assignment for $\mathsf{PRF}_{M,A_\neg,k}$ then M does not satisfy A. For some formulas there is a known *completeness threshold*, which is a bound on the number of states (usually the length of a path) necessary to disprove the formula. However, the completeness threshold usually depends on both the formula and the model, and in practice it is difficult to compute. Following [16], we are interested in a *dynamic completeness criterion*: a formula $\mathsf{CMP}_{M,A,k}$ that is satisfiable while there is still hope of finding a counter-example, and that becomes unsatisfiable when there is none. Essentially, $\mathsf{CMP}_{M,A,k}$ should encode the fact that it is possible to arrange k states so that they form a "beginning" of a proof that might be extended into a valid proof by adding more states.

To see how $\mathsf{CMP}_{M,A,k}$ should be constructed, consider the situation where we have not found a proof when the bound is k, but there exists a proof Π with $k' > k$ states. Now let Π' be an invalid proof constructed by taking k states of Π, including s_0, and using the invariants and ranks of Π restricted

to these k states. It is easy to see that in Π', the Initiality and Consistency obligations are satisfied. For automaton states $q \in Q \setminus Q_\Diamond$, the Invariance and Progress requirements are satisfied as well. However, Π' must violate some proof obligations, because there is no valid proof of size k. The obligations that are violated are Invariance and Progress obligations for states $q \in Q_\Diamond$ that have $\Diamond$-transitions, and the reason they are violated in Π' is that they rely on some of the states that appear in Π but not in Π'.

We would like to identify situations where no proof fragment Π' matching this description exists, and therefore $\mathsf{CMP}_{M,A,k}$ will encode these requirements:

- Initiality.
- For automaton states $q \in Q \setminus Q_\Diamond$, Invariance and Progress.
- For automaton states $q \in Q_\Diamond$, a weakened version of Invariance and Progress, where the successor required by the obligation for q with $\delta(q) = \Diamond q_1$ is not required to be in the invariant for q_1 unless it is one of the states $u_0, \ldots, u_{k-1}$. This allows the successor to be a new unconstrained state, required only to be distinct from the regular proof states.

The weakened Invariance and Progress requirement identifies cases where adding more states to the proof may yield a valid proof. Assuming the encoding of Section 3.1 with proof successors encoded explicitly, the weakened requirement is given by

$$\langle\!\langle \Diamond q_1 \rangle\!\rangle_i^W = \mathsf{R}(u_i, t_i^q) \wedge \bigwedge_{j=0}^{k-1} \left(t_i^q = u_j \rightarrow \left(x_j^{q_1} \wedge \mathsf{LT_q}(\rho_j^{q_1}, \rho_i^q)\right)\right)$$

where t_i^q is the successor required by $q \in Q_\Diamond$ that has $\delta(q) = \Diamond q_1$ for u_i. Violated Invariance and Progress requirements for states that do not have a $\Diamond$-transition cannot be satisfied directly by adding more states, so they are left unchanged: $\langle\!\langle \delta(q) \rangle\!\rangle_i^W = \langle\!\langle \delta(q) \rangle\!\rangle_i$ for all $q \in Q \setminus Q_\Diamond$.

Additionally, we would like to constrain the SAT solver to use the k proof states "constructively", otherwise a satisfying assignment might encode, e.g., k unreachable states, and satisfy all their proof obligations by adding new and unconstrained states. To this end we will require that all k proof states be distinct from each other, and also that each state except s_0 (represented by u_0) be a proof-successor for some state. This constrains satisfying assignments to encode only "proof-reachable" states — states that are reachable from s_0 by a path in which each state is a proof-successor for the state preceding it.

The formula that encodes all these requirements is given by

$$\mathsf{CMP}^1_{M,A,k} = \mathsf{I}(u_0) \wedge x_0^{q_0} \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{q \in Q} x_i^q \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i^W \wedge \bigwedge_{i \neq j} u_i \neq u_j \wedge$$
$$\wedge \bigwedge_{i=1}^{k-1} \left(\bigvee_{j=0}^{k-1} \bigvee_{q \in Q_\Diamond} u_i = t_j^q \wedge x_j^q \right)$$

Theorem 4. *If there exists a proof for the fact that M satisfies A with $k' \geq k$ states, then $\mathsf{CMP}^1_{M,A,k}$ is satisfiable.*

The scheme for using the dynamic completeness criterion is shown in Alg. 1, which takes as input a $\Diamond$-automaton A and a structure M. For a $\Box$-automaton one constructs the complement, calls the algorithm and returns the opposite answer.

Algorithm 1 BMC using the dynamic completeness criterion

```
for k = 1 to |S| do
  if PRF^{1'}_{M,A,k} is satisfiable then
    return M ⊨ A
  else if CMP^1_{M,A,k} is not satisfiable then
    return M ⊭ A
  end if
end for
return M ⊭ A
```

5 Related work

Many BMC encodings have been suggested for linear-time logics of increasing complexity, among them [7], which handles LTL with past, and [10] and [8], which handle all ω-regular properties. In particular, [10] and [8] apply ideas from the world of symbolic model-checking for branching-time logic to BMC for linear-time. Here we apply similar ideas in their original context of model-checking for branching-time logic.

The first BMC scheme for a branching-time logic, ACTL, was suggested in [14]. This scheme works by explicitly encoding a computation tree of depth k which does not satisfy the formula. Instead of encoding a full tree, it bounds the number of paths needed, based on the structure of the formula. The number of paths is exponential in the nesting depth of the formula (in the worst case).

A more general approach was suggested by Wang in [16], in which the existence of a bounded-depth proof for the negation of a universal μ-calculus property is encoded as a SAT problem. The approach is somewhat similar to ours, but [16] relies on Stirling's local proof rules [15], which keep track of all the states visited along each proof branch, and require directly that the branch be acyclic (for a least fixpoint) or cyclic (for a greatest fixpoint). Different branches of the proof share no information, and a single model state can appear many times in the proof tree.

The encoding of [16] is simple and elegant, but encoding the proof as a tree-like structure allows no sharing of information between different branches of the proof. The disadvantage becomes more acute for complicated formulas, where the proof tree contains many nested subgoals, each of which needs to be justified separately. For disjunction, the encoding of [16] unrolls two separate subproofs,

even though only one of the goals needs to be satisfied; for example, to show that $EFp \vee EFq$ holds ("there exists a path on which we eventually reach p or a path on which we eventually reach q"), two separate paths will be unrolled. In contrast, our encoding enables maximal sharing of information: a model state need never be encoded more than once, and information about the automaton states it satisfies can be used to justify many different subgoals.

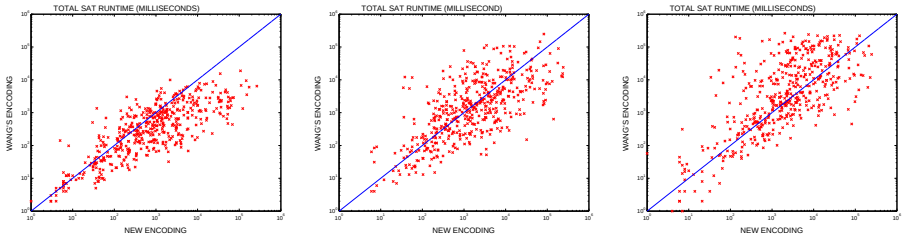
The number of variables used in the encoding of [16] increases exponentially with the bound in the worst case for universal μ-calculus properties, and polynomially for ACTL formulas, with the exponent being the nesting depth of temporal operators in the formula; when the bound increases by one, a full layer is added to the proof tree. Our encoding allows finer control. Although in the worst case it may need to encode the same number of states as the encoding of [16], it can often terminate with a smaller bound and smaller counter-examples. It is not possible to obtain a minimal counter-example from a satisfying assignment to the formula generated in [16], and it is also not possible to determine which parts of the structure returned are relevant to the proof (for example, in the case of conjunction, it is not possible to tell which conjunct was disproved). The bounds used in our encoding and in [16] are incomparable: our bound represents the exact number of states in the counter-example, while the bound in [16] represents the depth of the proof tree. Either method may terminate with a smaller bound than the other.

In [12] it is shown how to solve parity games through a reduction to SAT. This work is closely related to our own, since parity games are equivalent to μ-calculus and to alternating parity tree automata; [12] also uses ranks in a manner similar to ours. However, the encoding of [12] assumes that an explicit representation of the gameboard, which is difficult to compute for the model checking problem, since it means computing the product of the model and the automaton. Also, the encoding represents the entire gameboard at once, and therefore it does not lend itself immediately to bounded model checking.

6 Experimental results

We implemented an ACTL version of our encodings and Wang's encoding from [16] in the NuSMV2 framework [2], and tested their performance on a 3GHz Pentium 4 with 4GB memory, using the ZChaff SAT solver. We used random Kripke structures with 100 states each, and random formulas that were not satisfied in the models. The formulas were of nesting depth $2-5$ of the temporal operators AF, AG and AU. We used a maximal bound of 20, which was never reached in our experiments, and a timeout of 5 minutes. The simplified encoding from Section 3.2 greatly outperformed the general encoding of Section 3.1, probably owing to our naive implementation of ranks, and we present results only for the simplified encoding. Our results for 500 formulas for each nesting depth from 2 to 4 are summarized in Fig. 2 and Table 1.

For nesting depth 2, the encoding of [16] performs better than our encoding (Fig. 2(a)). For nesting depth 3 (Fig. 2(b)) the encodings perform roughly the same, and for nesting depth 4 or greater our encoding performs better than the



(a) Nesting depth 2 (b) Nesting depth 3 (c) Nesting depth 4

Fig. 2. Total SAT runtimes for disproving ACTL formulas of nesting depth 2 – 4

	Nesting depth 2	Nesting depth 3	Nesting depth 4	Nesting depth 5
Wang's encoding	**100%**	**99%**	84%	76%
New encoding	98%	97%	**95%**	**94%**

Table 1. Success rate in disproving ACTL formulas of nesting depth 2 – 5

encoding of [16] (Fig. 2(c)). The counter-examples found by our encoding were generally very small (10 states or less), and the depth of the proof tree encoded in Wang's encoding was often larger. The counter-examples returned by Wang's encoding were larger by an order of magnitude than the examples returned by our encoding for all nesting depths.

7 Conclusion

We have presented a novel approach to bounded model-checking for branching-time logics. We showed two encodings, together with a dynamic termination criterion, which can be used to both prove and disprove specifications in universal or existential branching-time logic. Our experimental results show that for ACTL formulas with a large nesting depth, our encodings perform better than the previous encoding of Wang. We believe that these results will extend to ACTL with fairness and to general μ-calculus formulas.

The approach presented here is applicable to many logics, from ACTL to μ-calculus, and can be extended to use different types of automata as specifications, using ranking functions to represent different acceptance conditions. The use of ranks can also be applied to BMC in linear-time logics, for example by modifying the encoding of [8] for weak alternating Büchi word automata, resulting in encodings that use fewer variables; however, it is not clear that performance will be improved.

The formulas generated in our encodings, and particularly the dynamic completeness criterion, share most of their constraints with the formulas generated in previous iterations. This makes them suitable for incremental SAT, where conflict clauses learned in previous calls to the SAT solver are re-used to help solve the next instance. Performance may also be improved by using encodings

of ranks optimized for SAT, developed in the context of termination checking (e.g., [5]).

References

1. Accellera. *PSL Reference Manual v1.1*, 2004.
2. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *CAV'02*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.
3. Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, 2001.
4. Edmund Clarke, Daniel Kroening, Ofer Strichman, and Joel Ouaknine. Completeness and complexity of bounded model checking. In *VMCAI*, volume 2937 of *LNCS*, pages 85–96, 2004.
5. Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Solving partial order constraints for LPO termination. In *RTA*, volume 4098 of *LNCS*, pages 4–18. Springer, 2006.
6. Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. *Electr. Notes Theor. Comput. Sci.*, 89(4), 2003.
7. Keijo Heljanko, Tommi Junttila, and Timo Latvala. Incremental and complete bounded model checking for full PLTL. In Kousha Etessami and Sriram K. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'2005)*, volume 3576 of *LNCS*, pages 98–111, Edinburgh, Scotland, United Kingdom, July 2005. Springer.
8. Keijo Heljanko, Tommi A. Junttila, Misa Keinänen, Martin Lange, and Timo Latvala. Bounded model checking for weak alternating büchi automata. In *CAV*, volume 4144 of *LNCS*, pages 95–108. Springer, 2006.
9. David Janin and I. Walukiewicz. Automata for the mu-calculus and related results. In *MFCS (Mathematical Foundations of Comuter Science)*, volume 969 of *LNCS*, pages 552–562. Springer, 1995.
10. M. Jehle, J. Johannsen, M. Lange, and N. Rachinsky. Bounded model checking for all regular properties. In A. Biere and O. Strichman, editors, *Proc. 3rd Int. Workshop on Bounded Model Checking, BMC'05*, volume 144 of *Electr. Notes in Theor. Comp. Sc.*, pages 3–18. Elsevier, 2005.
11. Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.
12. M. Lange. Solving parity games by a reduction to SAT. In R. Majumdar and M. Jurdziński, editors, *GDV'05*, 2005.
13. Kedar S. Namjoshi. Certifying model checkers. In *CAV'01*, volume 2102 of *LNCS*, pages 2–13, Paris, France, July 2001.
14. Wojciech Penczek, Bozena Wozna, and Andrzej Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundam. Inf.*, 51(1):135–156, 2002.
15. Colin Stirling and David Walker. Local model checking in the modal mu-calculus. *Theor. Comput. Sci.*, 89(1):161–177, 1991.
16. Bow-Yaw Wang. Proving forall-mu-calculus properties with SAT-based model checking. In *FORTE'05*, volume 3731 of *LNCS*, pages 113–127. Springer, 2005.
17. Thomas Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Soc. Math. Belg.*, 8(2), 2001.