Bounded Model Checking with SNF, Alternating Automata, and Büchi Automata

Daniel Sheridan[1]

School of Informatics
University of Edinburgh
Edinburgh, UK

Abstract

Model checking of LTL formulæ is traditionally carried out by a conversion to Büchi automata, and there is therefore a large body of research in this area including some recent studies on the use of alternating automata as an intermediate representation.
Bounded model checking has until recently been apart from this, typically using a direct conversion from LTL to propositional logic. In this paper we give a new bounded model checking encoding using alternating automata and focus on the relationship between alternating automata and SNF. We also explore the differences in the way SNF, alternating, and Büchi automata are used from both a theoretical and an experimental perspective.

Keywords: Bounded model checking, SNF, LTL, Büchi automata, Alternating automata

1 Introduction

Before the introduction of bounded model checking in 1999 [1], LTL model checking was typically performed by converting the formula to an automaton expressing the formula, forming the product with the model automaton, then checking the result for emptiness. Research into producing the smallest automaton for a given LTL formula has been extensive and varied. There is literature giving improvements to the original "GPVW" conversion algorithm [12] including simplifying the LTL before conversion, and the automaton after conversion (eg, [7]) as well as the conversion itself. Some recent work [10,11]

---

[1] Email: d.j.sheridan@sms.ed.ac.uk

proposes the use of alternating automata (AA) as an intermediate representation of the formula. The LTL to AA conversion is linear space so allows for simplifications to be easily performed before the exponential space conversion to a Büchi automaton.

Bounded model checking (BMC) has traditionally taken a different approach: the original paper [1] gives an encoding from LTL directly to propositional logic. Being defined recursively on the structure of the formula this (naïvely) appears to be exponential size in the number of states, although with careful treatment [5] the result is polynomial size. An alternative encoding [9] based directly on the fixpoint characterisations of LTL operators produces an encoding which is quadratic size, but may with care be reduced to linear size in the number of states. The use of LTL to automata conversions as part of bounded model checking was first explicitly suggested by de Moura et al. [6]. The only experimental comparison [5] is very brief and mainly exercises the LTL simplification available in many automata conversion programs.

Although there are grounds for distinguishing between the direct-to-propositional conversion and the conversions via automata as "syntactic" versus "semantic" [5], we demonstrate in this paper the close correspondence between SNF and alternating automata and their conversion procedures from LTL. We review the use of Büchi automata for BMC and give a new encoding to enable direct use of alternating automata. This allows us to compare more closely the use of the SNF encoding with the use of automata, to explore the advantages and disadvantages of each approach. We demonstrate some of these differences with a series of experiments.

2   Background

2.1   Bounded model checking

BMC solves the LTL model checking problem by observing a restricted number of states, k. Infinite counterexamples may be represented by a path of the form ab^ω: a *k-l-loop path* with $k = |ab|$ and $l = |a|$. We constrain a finite sequence of states π to be a k-l-loop by the assertion ${}_l L_k \doteq (\pi(k) = \pi(l))$ [2]. Alternatively we can give finite counterexamples as a *k-prefix path* for some LTL properties. In particular, it is not possible to show to give a counterexample for $\mathbf{F}\,f$ for a k-bounded path. Typically, we verify a model by examining a sequence of k states π interpreted as either a prefix or a loop; we write a disjunction over the k possible interpretations, testing all of the options for the type of path

---

[2] Note that we give an equivalence between $\pi(k)$ and $\pi(l)$ rather than the transition as used in the original presentation [1]

and the value of l simultaneously.

2.2   The Separated Normal Form

SNF [8] is a clause-like normal form based on the Separation Theorem of Gabbay, with the general form $\mathbf{G} \bigwedge_i (P_i \to F_i)$ where $P_i \to F_i$, called *rules* are restricted to (writing p and f for propositional formulæ)

Initial rules of the form $\mathbf{start} \to f$ where $\mathbf{start}$ holds only in the initial state of each path

Global invariant rules $p \to f$ with no temporal operator

Global step rules $p \to \mathbf{X}\, f$

Global eventuality rules $p \to \mathbf{F}\, f$

Transformation from an LTL formula in NNF f to a set of SNF rules is achieved by repeatedly applying the transformation functions in Figure 1 to the initial formula set $\{\mathbf{start} \to f\}$ [9]. The transformations introduce new variables identified by the syntax $\underline{x}$ with x indicating the intuitive meaning of the variable. We write Γ for the subset of formulæ which are not affected by the transformation, φ and ψ for arbitrary LTL formulæ in NNF, and f and g for propositional formulæ. We also write $\psi(\mathbf{G}\, f)$ to say that $\mathbf{G}\, f$ occurs in ψ, while $\psi(g)$ stands for the formula obtained by substituting every occurrence of $\mathbf{G}\, f$ with g in ψ; similarly for the other temporal operators.

2.3   Büchi Automata

We cover Büchi automata only briefly here; we direct the interested reader to the tutorial paper by Wolper [17].

Definition 2.1 A Büchi automaton $\mathcal{B}$ is defined by the tuple $\langle Q, \Sigma, \delta, I, T \rangle$ where Q is the set of states; Σ is the alphabet of transition labels; δ is the transition function $Q \to 2^{2^{\Sigma} \times Q}$; $I \subseteq Q$ is the set of initial states; $T \subseteq Q$ is the set of accepting states.

Note that we use 2^{Σ} in the definition of the transition relation in place of Σ in order to gather transitions that differ only by their actions — this can be a significant optimisation.

A run of a Büchi automaton is a path through the automaton; it is accepting if the states in T are visited an infinite number of times. That is,

Definition 2.2 A *run* of a Büchi automaton $\mathcal{B}$ with respect to a word $u_0 u_1 \ldots$ $\in \Sigma^{\omega}$ is a sequence of states in $q_0 q_1 \ldots \in Q^{\omega}$ with $q_0 \in I$ and $\forall i \exists \alpha_i \, \langle \alpha_i, q_{i+1} \rangle \in$

$$\text{SNF}_{[\mathbf{X}]}(\{\varphi \to \psi(\mathbf{X}\ f)\} \cup \Gamma) \doteq \left\{ \begin{array}{c} \varphi \to \psi(\underline{\mathbf{X}\ f}) \\ \underline{\mathbf{X}\ f} \to \mathbf{X}\ f \end{array} \right\} \cup \Gamma$$

$$\text{SNF}_{[\mathbf{F}]}(\{\varphi \to \psi(\mathbf{F}\ f)\} \cup \Gamma) \doteq \left\{ \begin{array}{c} \varphi \to \psi(\underline{\mathbf{F}\ f}) \\ \underline{\mathbf{F}\ f} \to \mathbf{F}\ f \end{array} \right\} \cup \Gamma$$

$$\text{SNF}_{[\mathbf{G}]}(\{\varphi \to \psi(\mathbf{G}\ f)\} \cup \Gamma) \doteq \left\{ \begin{array}{c} \varphi \to \psi(f \wedge \underline{\mathbf{X}\,\mathbf{G}\ f}) \\ \underline{\mathbf{X}\,\mathbf{G}\ f} \to \mathbf{X}\,(f \wedge \underline{\mathbf{X}\,\mathbf{G}\ f}) \end{array} \right\} \cup \Gamma$$

$$\text{SNF}_{[\mathbf{U}]}(\{\varphi \to \psi(f\ \mathbf{U}\ g)\} \cup \Gamma) \doteq \left\{ \begin{array}{c} \varphi \to \psi(g \vee (f \wedge \underline{\mathbf{X}(f\ \mathbf{U}\ g)})) \\ \underline{\mathbf{X}(f\ \mathbf{U}\ g)} \to \mathbf{X}\,(g \vee (f \wedge \underline{\mathbf{X}(f\ \mathbf{U}\ g)})) \\ \varphi \to \mathbf{F}\ g \end{array} \right\} \cup \Gamma$$

$$\text{SNF}_{[\mathbf{R}]}(\{\varphi \to \psi(f\ \mathbf{R}\ g)\} \cup \Gamma) \doteq \left\{ \begin{array}{c} \varphi \to \psi(g \wedge (f \vee \underline{\mathbf{X}(f\ \mathbf{R}\ g)})) \\ \underline{\mathbf{X}(f\ \mathbf{R}\ g)} \to \mathbf{X}\,(g \wedge (f \vee \underline{\mathbf{X}(f\ \mathbf{R}\ g)})) \end{array} \right\} \cup \Gamma$$

Fig. 1. The transformation function for SNF.

$\delta(q_i)$ such that $u_i \in \alpha_i$. A run is *accepting* if infinitely many states in the run are members of T.

A *generalised* Büchi automaton (GBA) has a set of accepting sets $\mathcal{T} \subseteq 2^Q$; each set must be visited infinitely often for acceptance. A GBA may be reduced to a classical Büchi automaton but incurs a linear blowup of $O(|\mathcal{T}|)$.

2.4 *Alternating Automata*

Alternating automata are a type of tree automaton (runs are described as trees rather than linear traces) combining both deterministic and nondeterministic behaviours: a transition in a nondeterministic automaton leads to a set of states from which one is chosen; a transition in a deterministic tree automaton leads to a successor set. Alternating automata exhibit the combination of these existential and universal behaviours. Although the presentation that we adopt below is one of a nondeterministic choice between conjunctions of states, it can be generalised to arbitrary propositional formulæ over $\wedge, \vee$ and states. Alternating automata are exponentially more succinct than Büchi automata.

There are two presentations of LTL to automata conversion via alternating

automata. We follow the slightly unconventional presentation by Gastin and Oddoux [11]: transitions are from a state to a conjunction of states; each state may have multiple transitions, selected nondeterministically. This effectively encodes a disjunction of conjunctions of states reached from a given state.

The presentation given by Fritz and Wolper [10] is equivalent, but the differences in the definitions lead to larger representations of the automata. An additional difference is that Gastin and Oddoux use a co-Büchi accepting condition, while Fritz uses a Büchi condition. We can disregard this: for the alternating automata under consideration, a Büchi condition $F \subseteq Q$ is equivalent to the co-Büchi condition $Q \setminus F$.

Definition 2.3 An *alternating co-Büchi automaton* $\mathcal{A}$ is defined by the tuple $\langle Q, \Sigma, \delta, I, F \rangle$ where Q is the set of states; Σ is the alphabet of transition labels; δ is the transition function $Q \to 2^{2^{\Sigma} \times 2^{Q}}$; $I \subseteq 2^{Q}$ is the set of initial combinations of states; $F \subseteq Q$ is the set of final states

As for the Büchi automaton definition above, the transition labels are from 2^{Σ}; accepted words are nevertheless from Σ^{ω}.

Alternating automata representing LTL formulæ are known to be *very weak*, which means that there is a partial order on the states $(Q, \sqsubseteq)$ determined by the transitions, such that $\forall q \in Q, \forall \langle \alpha, q' \rangle \in \delta(q), q' \sqsubseteq q$. That is, transitions are only permitted from a state to a lower or equal state. The result of this restriction is that the only loops in very weak co-Büchi alternating automaton (VWAA) are self-loops.

Definition 2.4 A run σ of a VWAA on a word $u_0 u_1 \ldots \in \Sigma^{\omega}$ is a labelled DAG $\langle V, E, \lambda \rangle$ with V partitioned into levels V_i, $V = \bigcup_{i \in \mathbb{N}} V_i$ and $E \subseteq \bigcup_{i \in \mathbb{N}} V_i \times V_{i+1}$. $\lambda : V \to Q$ labels the vertices of the graph with states of the automaton. V_i may be seen as a multiset of elements of Q. The graph is related to the word and the automaton by $\lambda(V_0) \in I$ and $\forall v \in V_i, \exists \langle \lambda(v), \alpha, s' \rangle \in \delta(\lambda(v)).u_i \in \alpha \wedge s' = \lambda(E(v))$ A run is accepting if every infinite branch of σ has only a finite number of nodes with labels in F.

2.4.1 LTL to VWAA Conversion

We report here the conversion procedure given by Gastin and Oddoux. The set operator $\otimes$ constructs the conjunctions of two sets of disjunctive normal form transitions: $X \otimes Y = \{\langle \alpha_1 \cap \alpha 2, e_1 \wedge e_2 \rangle \mid \langle \alpha_1, e_1 \rangle \in X, \langle \alpha_2, e_2 \rangle \in Y\}$. The overbar operator $\bar{\psi}$ converts ψ to a set-style disjunctive normal form representation: a set of conjunctions of atomic propositions or temporal subformulæ.

For an LTL formula φ over atomic propositions P, the VWAA $\mathcal{A}_{\varphi} = \langle Q, \Sigma, \delta, I, F \rangle$ is given by
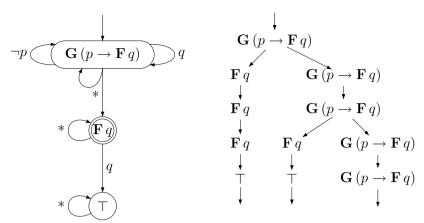
Fig. 2. Example alternating automaton (left) and part of a run over the input $\sigma = \{p\}\{\}\{p\}\{pq\}\cdots$ (right). * indicates the unconstrained transition.

- Q is the set of temporal subformulæ of Q (the set of subformulæ with an LTL operator as the main connective, union the set of atomic propositions)
- $\Sigma = 2^P$; $I = \bar{\psi}$; F is the set of formulæ of the form $\psi_1 \mathbf{U} \psi_2$ or $\mathbf{F} \psi_1$
- δ is defined as

$$\delta(\top) = \{\langle \Sigma, \top \rangle\}$$
$$\delta(p) = \{\langle \{a \in \Sigma \,|\, p \in a\}, \top \rangle\}$$
$$\delta(\neg p) = \{\langle \{a \in \Sigma \,|\, p \notin a\}, \top \rangle\}$$
$$\delta(\mathbf{X}\,\psi) = \{\langle \Sigma, e \rangle \,|\, e \in \bar{\psi}\}$$
$$\delta(\mathbf{F}\,\psi) = \Delta(\psi) \cup (\{\langle \Sigma, \mathbf{F}\,\psi \rangle\})$$
$$\delta(\mathbf{G}\,\psi) = \Delta(\psi) \otimes \{\langle \Sigma, \mathbf{G}\,\psi \rangle\})$$
$$\delta(\psi_1 \mathbf{U} \psi_2) = \Delta(\psi_2) \cup (\Delta(\psi_1) \otimes \{\langle \Sigma, \psi_1 \mathbf{U} \psi_2 \rangle\})$$
$$\delta(\psi_1 \mathbf{R} \psi_2) = \Delta(\psi_2) \otimes (\Delta(\psi_1) \cup \{\langle \Sigma, \psi_1 \mathbf{R} \psi_2 \rangle\})$$

where Δ is the extension of δ to include the propositional subformulæ of φ:

$$\Delta(\psi) = \delta(\psi) \qquad\qquad \text{if } \psi \in Q$$
$$\Delta(\psi_1 \wedge \psi_2) = \Delta(\psi_1) \otimes \Delta(\psi_2)$$
$$\Delta(\psi_1 \vee \psi_2) = \Delta(\psi_1) \cup \Delta(\psi_2)$$

We give an example VWAA corresponding to the LTL formula $\mathbf{G}\,(p \to \mathbf{F}\,q)$ in Figure 2 along with a sample run.

2.4.2 Compact Representation of Runs

The representation of a run of a VWAA as a DAG is problematic as the number of vertices at each level grows without bound. We can reduce the representation of a run by restricting each level to a set rather than a multiset, forming a reduced DAG. We call successive sets *configurations*, $C_i \subseteq Q$. A sequence of configurations over a word $u_0 u_1 \ldots \in \Sigma^\omega$ is accepting if there exists a set of edges E partitioned into $E_i \subseteq C_i \times C_{i+1}$ such that $\forall q \in C_i \exists \langle \alpha, q' \rangle \in \delta(q).u_i \in \alpha \wedge q' \subseteq E_i(q)$ and every path $q_0 q_1 \ldots$ such that $q_{i+1} \in E_i(q_i)$ contains only finitely many occurrences of the members of F.

For example, consider the run in Figure 2. The corresponding sequence of configurations is

$$\{\mathbf{G}\,(p \rightarrow \mathbf{F}\,q)\}$$
$$\{\mathbf{F}\,q, \mathbf{G}\,(p \rightarrow \mathbf{F}\,q)\}$$
$$\{\mathbf{F}\,q, \mathbf{G}\,(p \rightarrow \mathbf{F}\,q)\}$$
$$\{\mathbf{F}\,q, \mathbf{G}\,(p \rightarrow \mathbf{F}\,q)\}$$
$$\{\mathbf{G}\,(p \rightarrow \mathbf{F}\,q)\}$$

This is significantly weaker than the original formulation, but we can show that the languages accepted are equivalent. Firstly, every accepting sequence of configurations C_i with acceptance described by edges E_i may be directly translated into the DAG $\langle \bigcup_{i \in \mathbb{N}} C_i, \bigcup_{i \in \mathbb{N}} E_i, I \rangle$ where I is the identity function on states. In the opposite direction, every accepting DAG can be reduced to an accepting run of configurations given by $C_i = \bigcup_{v \in V_i} \lambda(v)$. We show this sequence is accepting by appealing to an important property of accepting paths: they are both left-append and suffix closed — that is, a suffix of an accepting path is also accepting, as is an accepting path prefixed with a finite number of additional states. This means that the acceptance condition can on configurations can be reduced to the existence of an accepting path from each element of each C_i. This is assured by examination of the DAG, since every element of each V_i must be followed by an accepting sequence of edges.

2.4.3 Superset Property of Runs

Both formulations of runs describe the minimal elements (or multiset) of states at each point in time, but neither requires that the set consists solely of these elements. We may, without changing the language accepted, replace C_i with a superset of C_i (similarly V_i) provided that successive configurations (levels of the tree) can be modified to accommodate the evolution of the extra states while remaining consistent with the definitions of the runs. This is crucial to the encoding described below: we need only constrain the current

configuration to be any superset of that described by the transitions.

3 Bounded Model Checking Encodings

Having discussed three representations of LTL formulæ suited to model checking we now turn to the way that these representations can be used for bounded model checking. The encoding of Büchi automata was discussed by de Moura et al. [6] as well as Clarke et al. [5]. The use of SNF for bounded model checking was the subjection of a paper by Frisch et al. [9]. The approach that we take here to bring the encodings together is to isolate the components of the encoding of the specification into three parts: that which constrains the path in all cases; that which constrains the path only when it is a finite path prefix; and that which constrains the path only when it is a k-loop. The addition of the first constraint to the original approach [1] has the potential to simplify the resulting formula [3] considerably:

$$[\![M, f]\!]_k := [\![M]\!]_k \wedge \mathrm{enc}_c(f, k) \wedge \left(\mathrm{enc}_n(f, k) \vee \bigvee_{l=0}^{k} \left({}_l\mathrm{L}_k \wedge \mathrm{enc}_l(f, k, l) \right) \right)$$

where $\mathrm{enc}_c, \mathrm{enc}_n,$ and enc_l denote the common, finite, and loop encodings as described below.

3.1 Bounded Model Checking with Büchi Automata

We present a variation on the encoding of de Moura et al. [6], making explicit the representation of states in order to avoid the overhead of enforcing mutual exclusion on states. In contrast with other presentations, we use generalised Büchi automata: the complexity of checking multiple acceptance sets is much lower than the overhead of conversion to classical Büchi automata.

All paths accepted by a Büchi automaton are infinite — formulæ with finite counterexamples such as $\mathbf{F}\,\phi$ are encoded with a trivial infinite loop. The finite prefix case is therefore never accepting, and we deduce that $\mathrm{enc}_n(f, k) = \bot$.

Given a generalised Büchi automaton representing LTL formula f, $\mathcal{B}_f = \langle Q, \Sigma, \delta, I, \mathcal{T} \rangle$, we encode the current state $q \in Q$ as a base two integer in the range $0 \ldots |Q| - 1$: there is a one-to-one mapping $\epsilon \subseteq Q \times \{ i \,|\, 0 \leq i < |Q| - 1 \}$. That is, for each state i, we have a set of propositional variables $q_n(i)$, $0 \leq n < \lceil \log_2(|Q|) \rceil$ and we write $[\![q]\!]^i$ for the assertion that the bit pattern $q_0 q_1 \ldots$

---

[3] The formula given is derived from the usual BMC formulation as given in Biere et al. [1]. We write $[\![M]\!]_k$ for the encoding of the model, ${}_l\mathrm{L}_k$ for the constraint that the path is a k-l-loop, but we omit the $\bigwedge_{0 \leq l < k} \neg{}_l\mathrm{L}_k$ non-loop constraint as suggested by [3]

is the base two representation of $\epsilon(q)$. For Büchi automata representing LTL, Σ is the set of propositions in that model; the encoding of elements $a \in \Sigma$ is given as $[\![a]\!]^i$ as for the standard encoding.

The transition relation is encoded as a set of constraints on the originating state, target state, and label. If the transition relation is total, we can write

$$T_{\mathcal{B}_f}(i, k) = \bigvee_{\langle s, \alpha, s' \rangle \in \delta} \bigvee_{a \in \alpha} \left(([\![s]\!]^i \wedge [\![a]\!]^i \wedge [\![s']\!]^{i+1}) \right)$$

The initial set is encoded directly as a disjunction over members of I:

$$I_{\mathcal{B}_f}(k) = \bigvee_{s \in I} [\![s]\!]^0$$

Finally, we encode the acceptance sets. The Büchi acceptance condition is that each member of $\mathcal{T}$ is visited infinitely often. As we have ruled out finite path prefixes, we know that all paths being considered are of the form ab^ω. If we assert as part of the loop encoding that the corresponding paths in the Büchi automaton follow the same pattern, we can simply require that representatives from each acceptance set appear in the loop (ie, in b):

$$F_{\mathcal{B}_f}(k, l) = \bigwedge_{T \in \mathcal{T}} \bigvee_{i=l}^{k} \bigvee_{s \in T} [\![s]\!]^i$$

Thus we have

$$\text{enc}_c(f, k) = I_{\mathcal{B}_f}(k) \wedge \bigwedge_{i=0}^{k-1} T_{\mathcal{B}_f}(i, k)$$

$$\text{enc}_n(f, k) = \bot$$

$$\text{enc}_l(f, k, l) = F_{\mathcal{B}_f}(k, l) \wedge \bigwedge_{i=0}^{\lceil \log_2(|Q|) \rceil - 1} q_i(l) \leftrightarrow q_i(k)$$

Although the LTL to Büchi automaton conversion is exponential in the size of the formula, the encoding above introduces only a linear number of variables. The resulting formula is linear size in the product of the number of transitions and k except for $F_{\mathcal{B}_f}$ which is quadratic: $O(|\mathcal{T}|k^2)$.

3.2 Bounded Model Checking with Alternating Automata

The encoding of alternating automata is very similar to Büchi automata. Since a run is a sequence of configurations rather than states we use one state vari-

able to represent each state; configurations are then represented by conjunctions of states.

Given a VWAA representing LTL formula f, $\mathcal{A}_f = \langle Q, \Sigma, \delta, I, F \rangle$, we encode the presence of a a state q in the ith configuration by the variable $q(i)$. A configuration is encoded as a conjunction of its members: we write $[\![C]\!]^i = \bigwedge_{q \in C} q(i)$, with $[\![\emptyset]\!]^i = \bot$. Note that this constrains the necessary, but not sufficient, members of the configuration, and so describes the smallest configuration that describes the run as discussed in Section 2.4.3. The targets of transitions can be seen as subsets of configurations and are hence encoded in the same way.

For VWAAs derived from LTL formulæ as above, the transitions are labelled with a set of sets of atomic propositions: the set of permitted assignments to propositions. These can be denoted [4] by a conjunction of literals where $p \wedge q$ denotes $\{a \in \Sigma \,|\, p \in a\} \cap \{a \in \Sigma \,|\, q \in a\}$. We write $[\![\alpha]\!]^i$ for the conjunction of literals representing $\alpha \in 2^\Sigma$ — this is particularly convenient as the implementation of the LTL to VWAA conversion [11] produces these conjunctions directly.

As before, the transition relation is given as a series of constraints

$$T_{\mathcal{A}_f}(i, k) = \bigwedge_{q \in Q} \left(q(i) \rightarrow \bigvee_{\langle \alpha, q' \rangle \in \delta(q)} \left([\![\alpha]\!]^i \wedge [\![s']\!]^{i+1} \right) \right)$$

and the initial set of configurations is encoded

$$I_{\mathcal{A}_f}(k) = \bigvee_{C_0 \in I} [\![C_0]\!]^0$$

A VWAA run is accepting if no branch contains an infinite occurrence of elements of F. This can be assured on a k-prefix path if the empty configuration is reached at any point: the *very weak* property means that all successive configurations are also empty and hence no state is visited infinitely often. This also means that we can reduce the check to an empty kth configuration: this will hold even if the first empty configuration is before k.

$$P_{\mathcal{A}_f}(k) = \bigwedge_{q \in Q} \neg q(k)$$

For the loop case, we cannot simply check for an infinite number of occurrences of the members of F as the co-Büchi condition is on paths through the configuration space. That is, an accepting run could consist of an infinite

---

[4] See Remark 2 in Gastin and Oddoux [11]

number of paths each with a finite number of occurrences of an acceptance state. In this case the acceptance state would appear in a configuration within the loop suggesting that the state was visited infinitely often. In fact, we must make use of the *very weak* condition again: the only loops in VWAAs are self-loops, and hence the only paths that visit a state infinitely often must do so by always taking the self-loop transition. By the left-append and prefix closed property of accepting paths, we can deduce that if it is possible to take a non-self-loop transition from an accepting state then that state must be part of an accepting path.

$$F_{\mathcal{A}_f}(k, l) = \bigwedge_{q \in F} \bigvee_{i=l}^{k} \left([\![q]\!]^i \rightarrow \bigvee_{\substack{\langle \alpha, q' \rangle \in \delta(q) \\ q \notin q'}} \left([\![\alpha]\!]^i \wedge [\![q']\!]^{i+1} \right) \right)$$

Thus we have

$$\mathrm{enc}_c(f, k) = I_{\mathcal{A}_f}(k) \wedge \bigwedge_{i=0}^{k-1} T_{\mathcal{A}_f}(i, k)$$

$$\mathrm{enc}_n(f, k) = P_{\mathcal{A}_f}(k)$$

$$\mathrm{enc}_l(f, k, l) = F_{\mathcal{A}_f}(k, l) \wedge \bigwedge_{q \in Q} q(l) \leftrightarrow q(k)$$

This encoding produces a linear number of variables in the size of the LTL formula. The resulting propositional formula is linear in the product of the number of transitions and k, again except for $F_{\mathcal{A}_f}$ which is quadratic in k.

3.3 Bounded Model Checking with SNF

As SNF is a specialisation of LTL we could encode it using the standard BMC method, but we can produce a much better result by considering the structure of rules. Given a set of rules representing an LTL formula f, Ψ_f, we consider each type of rule separately:

Initial rules (start $\rightarrow f$) specify initial conditions:

$$I_{\Psi_f}(k) = \bigwedge_{(\mathbf{start} \rightarrow f) \in \Psi_f} [\![f]\!]^0$$

Global invariant rules $p \rightarrow f$ are constraints on the configurations of individual states:

$$P_{\Psi_f}(i) = \bigwedge_{(p \rightarrow f) \in \Psi_f} [\![p \rightarrow f]\!]^i$$

Global step rules $(p \to \mathbf{X} f)$ connect states with their successors, similar to a transition relation. Above, we included the transition relation in enc_c together with a loop condition on its states in enc_l. However, we can simplify this by isolating the common cases at time $< k$ from the boundary cases which distinguish the behaviour of the finite prefix and k-loop conditions.

$$T_{\Psi_f}(i,k) = \bigwedge_{(p \to \mathbf{X} f) \in \Psi_f} [\![p]\!]_k^i \to [\![f]\!]^{i+1}$$

$$T_{\Psi_f}^n(k) = \bigwedge_{(p \to \mathbf{X} f) \in \Psi_f} [\![p]\!]_k^k \to \bot$$

$$T_{\Psi_f}^l(k,l) = \bigwedge_{(p \to \mathbf{X} f) \in \Psi_f} [\![p]\!]_k^k \to [\![f]\!]^l$$

Global eventuality rules $(p \to \mathbf{F} f)$ are superficially similar to acceptance conditions but can be interpreted more directly — as in [1]. For a finite prefix, this is simply a disjunction over states; for a k-loop of the form ab^ω, evaluating $\mathbf{F}$ during b is equivalent to evaluating it at the start of b.

$$F_{\Psi_f}^n(k) = \bigwedge_{i=0}^{k} \bigwedge_{(p \to \mathbf{F} f) \in \Psi_f} \left([\![p]\!]^i \to \bigvee_{j=i}^{k} [\![f]\!]^j \right)$$

$$F_{\Psi_f}^l(k,l) = \bigwedge_{i=0}^{k} \bigwedge_{(p \to \mathbf{F} f) \in \Psi_f} \left([\![p]\!]_k^i \to \bigvee_{j=\min(i,l)}^{k} [\![f]\!]^j \right)$$

Thus we have

$$\mathrm{enc}_c(f,k) = I_{\Psi_f}(k) \ \wedge \ \bigwedge_{i=0}^{k} P_{\Psi_f}(i) \ \wedge \ \bigwedge_{i=0}^{k-1} T_{\Psi_f}(i,k)$$

$$\mathrm{enc}_n(f,k) = T_{\Psi_f}^n(k) \ \wedge \ F_{\Psi_f}^n(k)$$

$$\mathrm{enc}_l(f,k,l) = T_{\Psi_f}^l(k,l) \ \wedge \ F_{\Psi_f}^l(k,l)$$

As noted above, the size of the SNF representation is linear in the size of the LTL formula; the number of variables in the encoding is therefore linear in the product of k and the size of the formula. The size of the resulting formula using the encoding given above is linear in the product of k and the size of the LTL except for the encoding of eventuality rules which is quadratic in k.

3.3.1   Reduced SNF: the "Fixpoint" form

A further refinement that can be made to SNF in the context of bounded time is the transformation for $\mathbf{F}$ [9]. Using the **bound** operator, which holds only

at time k (at the end of the first occurrence of ab in ab^ω), we can write the transformation

$$\text{SNF}'_{[\mathbf{F}]}(\{\varphi \to \mathcal{F}(\mathbf{F}\, f)\} \cup \Gamma) \doteq \left\{ \begin{array}{c} \varphi \to \psi(f \vee \underline{\mathbf{X}\,\mathbf{F}\, f}) \\ \underline{\mathbf{X}\,\mathbf{F}\, f} \to \mathbf{X}\,(f \vee \underline{\mathbf{X}\,\mathbf{F}\, f}) \\ \mathbf{bound} \to \neg\underline{\mathbf{X}\,\mathbf{F}\, f} \end{array} \right\} \cup \Gamma$$

This direct approach affects the length of the counterexample: evaluating $\mathbf{F}\,x$ in the loop part of the path will only check states up to k, rather than the whole of the loop as expected. The direct encoding approach to this is to always evaluate an eventuality from the start of the loop, since $\bigvee_{i=n}^{k} x \vee \bigvee_{i=l}^{n-1} x \equiv \bigvee_{i=l}^{k} x$ for $l < n \leq k$. The equivalent for RSNF is to consider each eventuality at both the current time and projected to the start of the loop. The latter is explicitly renamed out and the projection asserted by the AtLoop rule given below. This renaming is *time-independent*; that is, the introduced variable $\underline{\mathbf{F}\, f}$ is not a state variable but rather is a simple propositional variable, and this is reflected in the encoding.

$$\text{SNF}''_{[\mathbf{F}]}(\{\varphi \to \mathcal{F}(\mathbf{F}\, f)\} \cup \Gamma) \doteq \left\{ \begin{array}{c} \varphi \to \psi(f \vee \underline{\mathbf{X}\,\mathbf{F}\, f} \vee \underline{\mathbf{F}\, f}) \\ \underline{\mathbf{F}\, f} \to \text{AtLoop}(f \vee \underline{\mathbf{X}\,\mathbf{F}\, f}) \\ \underline{\mathbf{X}\,\mathbf{F}\, f} \to \mathbf{X}\,(f \vee \underline{\mathbf{X}\,\mathbf{F}\, f}) \\ \mathbf{bound} \to \neg\underline{\mathbf{X}\,\mathbf{F}\, f} \end{array} \right\} \cup \Gamma$$

This gives us the encoding

$$\text{enc}'_c(f,k) = \text{enc}_c(f,k) \wedge \bigwedge_{(\mathbf{bound} \to f) \in \Psi_f} [\![f]\!]^k$$

$$\text{enc}'_n(f,k) = T^n_{\Psi_f}(k) \wedge \bigwedge_{(p \to \text{AtLoop}(f)) \in \Psi_f} p \to \bot$$

$$\text{enc}'_l(f,k,l) = T^l_{\Psi_f}(k,l) \wedge \bigwedge_{(p \to \text{AtLoop}(f)) \in \Psi_f} p \to [\![f]\!]^l$$

For an alternative presentation of this approach, see Cimatti et al. [4].

The encoding given above has the number of variables linear in the product of k and the size of the formula as before. The size of the resulting formula is linear in the product of k and the size of the LTL.

4   SNF versus Automata

We have examined two established methods of encoding LTL for bounded model checking and introduced a third: the encoding via alternating automata. We now clarify the relationships and relative advantages of the encodings.

4.1   SNF and Alternating Automata

The configuration view of alternating automata makes it apparent that Fixpoint and AA are nearly equivalent. Step rules in SNF/Fixpoint relate states and their successors to the evolved state of the model, while AA transitions which relates states and their successors to the present state of the model. We can project each SNF variable x created during LTL conversion to a VWAA state $\mathbf{X}\,x$: the set of SNF variables is directly related to the members of the configurations of the VWAA. Furthermore, we can show that SNF step rules created from LTL always have atomic antecedents: a necessary condition to relate step rules to transitions.

The boundary condition used in Fixpoint to represent eventualities corresponds to an assertion that x occurs finitely, not infinitely, often. It is introduced for the same states that, in the alternating automaton conversion, would be in the co-Büchi acceptance set. The difficulty of checking the co-Büchi acceptance condition are sidestepped by the start-of-loop projection introduced in Section 3.3.1. Effectively, all branches of the run are collapsed into one.

In fact, this is the main advantage of SNF over VWAAs: the encoding of the acceptance set is complex and comparatively large for the alternating automaton encoding. There are other advantages: not being a transition system, the variables introduced by SNF are not included in the loopback condition L_k, eliminating the need for the empty-configuration assertion in the finite case. This can even reduce slightly the bound at which counterexamples are found. Alternating automata do benefit from the simplification [11] and simulation [10] reductions, some of which do not project directly to SNF; the advantages of these have the potential to outweigh the drawbacks of the encoding.

4.2   SNF and Alternating Automata versus Büchi Automata

Most of the encoding issues discussed above apply equally to Büchi automata, the exception being the acceptance set which is simpler than the alternating case, although still more complex than the Fixpoint case. The biggest draw-

back for BMC is the requirement for an infinite path. Safety properties with finite counterexamples must still end up in a loop — in both the specification automaton *and the model* which could lengthen the counterexample considerably. In fact, the best choice for simple specifications seems to be the direct encoding: in such a case, the loop constraint could be eliminated altogether.

There are two other loop-related problems with the use of Büchi automata. Firstly, when both the specification and model automata must be in a loop, the length of the loop is the least common multiple of the lengths of the loops in the two automata on their own. This is not an issue for alternating automata because of the weakness property: all loops will be a single state. Secondly, BMC is able to take special advantage of the loopback where a finite counterexample takes the form ab^i. For example, consider the word $xx(abb)^\omega$, which is recognised in this form by the specification $F(b \wedge F(a))$ using the direct or SNF encodings, but which must be expanded to $xxabb(abb)^\omega$ to be recognised by the automata-based encodings.

4.3   Complexity

We noted the complexity of each encoding at the end of its corresponding section. In each case, the encoding produces a linear number of variables and symbols in the size of the original LTL, and in each case there is only a small part of the encoding which produces a quadratic, rather than linear, number of symbols in k: for the automata encodings, it is the Büchi and co-Büchi conditions; for SNF it is the encoding of eventualities. The refinement of SNF can, however, be encoded in a linear number of symbols as described above. No such improvement is immediately obvious for the automata encodings, so specifications including **R** or **G** operators suffer from quadratic growth with these encodings.

4.4   Empirical Results

To demonstrate some of the differences between the approaches we give a selection of experimental results comparing a variety of BMC encodings. The existing encodings, the original BMC encoding [1] (marked "Orig" in the results), the SNF encoding and its refinement [9] ("SNF" and "FIX") are compared against Büchi automata, in this case the Etessami and Holzmann [7] procedure ("TMP"), and the VWAA produced by the tool from Gastin and Oddoux [11] with and without its simplifications ("AA" and "AA-").

To provide a comparison over a range of LTL specifications we fix the model for the experiments, using a distributed mutual exclusion example [14] with the specifications given in Frisch et al. [9], at several different bounds to illustrate

Enc.	k	Size	Vars	Time	Size	Vars	Time	Size	Vars	Time
		Accessibility (4,2)			Overtaking 1 (5,5)			Overtaking 2 (8,8)		
	30	14596	2480	0.35	15737	2511	0.17	16339	2573	0.40
AA	40	19436	3280	1.47	20957	3321	2.02	21759	3403	1.05
	50	24276	4080	4.67	26177	4131	8.45	27179	4233	10.11
	30	15325	2759	0.39	17011	2945	0.35	18065	3069	0.35
AA-	40	20405	3649	1.39	22651	3895	1.40	24055	4059	1.35
	50	25485	4539	5.28	28291	4845	11.87	30045	5049	7.23
	30	14298	2418	0.97	14481	2480	0.23	14814	2573	0.25
SNF	40	19038	3198	0.90	19281	3280	0.75	19724	3403	1.08
	50	23778	3978	4.04	24081	4080	2.76	24634	4233	2.22
	30	14299	2449	0.72	14483	2511	0.21	14816	2604	0.27
FIX	40	19039	3239	0.89	19283	3321	0.96	19726	3444	0.88
	50	23779	4029	4.43	24083	4131	4.17	24636	4284	2.59
	30	14599	2418	0.75	16559	2449	0.42	17898	2480	0.46
TMP	40	19439	3198	4.54	22049	3239	1.43	23828	3280	1.24
	50	24279	3978	4.90	27539	4029	3.54	29758	4080	7.07
	30	15848	2356	0.26	41874	2356	0.37	Encoding time		
Orig	40	21908	3116	1.47	81539	3116	1.92	> 1800 secs		
	50	28368	3876	9.83	142404	3876	17.69			

Table 1

Timings in zChaff for the DME example using three valid specifications. Specifications given as "Name (number of temporal operators, maximum nesting depth)"; "Size" indicates the number of clauses.

scalability. The number and nesting depths of temporal operators appearing in the specifications are reported as pairs of numbers alongside their names in the tables. We used a modified version of NuSMV [2] with an improved CNF conversion [16]; timings were made in the SAT solver zChaff [15].

Table 1 shows the results from verifying three correct specifications. Rather than report the number of states that each automata conversion produces, we report the size of the CNF result. This means that the automaton methods can be directly compared to the SNF and direct encodings.

We observe that as the specifications become more complex, the simplicity of the SNF encoding has an increasing advantage. The alternating automata approach lags close behind the Büchi automata produced by TMP: a particularly interesting result, as the latter includes advanced simulation-based simplification techniques, while the former uses simple transition and state simplifications.

We illustrate the effect of the different encodings on counterexample size by comparing two incorrect specifications with different minimal counterexamples (Table 2). Here we see that the Büchi automaton procedure is slower due to the longer counterexample produced. The other procedures are all comparable

Enc.	k	Time	k	Time
	Priority 1 (4,2)		Priority 2 (4,2)	
AA	14	0.03	53	0.30
AA-	14	0.03	53	0.89
SNF	13	0.02	52	0.49
FIX	13	0.02	52	0.83
TMP	53	3.26	> 200	
Orig	13	0.02	52	1.15

Table 2
Timings in zChaff for the DME example using two invalid specifications. Specifications given as
"Name (number of temporal operators, maximum nesting depth)"

although the VWAA method is slightly faster on the larger example.

5   Conclusions and Future Work

The main advantage of automata based bounded model checking, the high
state of development of the conversion procedures, is balanced by the numer-
ous drawbacks of conversion. We have described how the use o alternating
automata overcomes many of these problems and demonstrated their use for
BMC. A simple alternating automata encoding has been shown to be almost
as effective as a highly developed Büchi automata approach, although both
lag behind the SNF encoding (without any simplification) on many of the
examples given.

This work has indicated several promising directions for further develop-
ment. Simulation-based simplification for alternating automata [10] may im-
prove the performance of the approach, and the close relationship with SNF
could mean that the SNF encoding could also be improved by such simplifi-
cation techniques. This relationship could also yield better encodings for the
co-Büchi condition, further improving the performance. A possible alternative
technique for encoding the co-Büchi condition is to adapt the new linear-space
encoding of Latvala et al. [13].

6   Acknowledgements

References

[1] Biere, A., A. Cimatti, E. Clarke and Y. Zhu, *Symbolic model checking without BDDs*, in: W. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems. 5th International Conference, TACAS'99*, Lecture Notes in Computer Science **1579** (1999), pp. 193–207.

[2] Cimatti, A., E. Clarke, F. Giunchiglia and M. Roveri, NuSMV*: a new Symbolic Model Verifier*, in: N. Halbwachs and D. Peled, editors, *Proceedings of the Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science (1999), pp. 495–499.

[3] Cimatti, A., M. Pistore, M. Roveri and R. Sebastiani, *Improving the encoding of LTL model checking into SAT*, in: A. Cortesi, editor, *Third International Workshop on Verification, Model Checking and Abstract Interpretation*, Lecture Notes in Computer Science **2294** (2002), pp. 196–207.

[4] Cimatti, A., M. Roveri and D. Sheridan, *Bounded verification of past LTL*, in: A. J. Hu and A. K. Martin, editors, *Formal Methods in Computer-Aided Design; 5th International Conference, FMCAD 2004*, Lecture Notes in Computer Science (2004).

[5] Clarke, E. M., D. Kroening, J. Ouaknine and O. Strichman, *Completeness and complexity of bounded model checking*, in: B. Steffen and G. Levi, editors, *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, January 11-13, 2004, Proceedings*, Lecture Notes in Computer Science **2937** (2004), pp. 85–96.

[6] de Moura, L., H. Rueß and M. Sorea, *Lazy theorem proving for bounded model checking over infinite domains*, in: A. Voronkov, editor, *Automated Deduction - CADE-18; 18th International Conference on Automated Deduction*, Lecture Notes in Computer Science **2392** (2002), pp. 438–455.

[7] Etessami, K. and G. J. Holzmann, *Optimizing Büchi automata*, in: C. Palamidessi, editor, *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, Lecture Notes in Computer Science **1877** (2000), pp. 153–167.

[8] Fisher, M., *A resolution method for temporal logic*, in: *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)* (1991), pp. 99–104.

[9] Frisch, A., D. Sheridan and T. Walsh, *A fixpoint based encoding for bounded model checking*, in: M. D. Aagaard and J. W. O'Leary, editors, *Formal Methods in Computer-Aided Design; 4th International Conference, FMCAD 2002*, Lecture Notes in Computer Science **2517** (2002), pp. 238–254.

[10] Fritz, C., *Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata*, in: O. H. Ibarra and Z. Dang, editors, *Implementation and Application of Automata. Eighth International Conference (CIAA 2003)*, Lecture Notes in Computer Science **2759**, Santa Barbara, CA, USA, 2003, pp. 35–48.

[11] Gastin, P. and D. Oddoux, *Fast LTL to Büchi automata translation*, in: G. Berry, H. Comon and A. Finkel, editors, *Proceedings of the 13th Conference on Computer Aided Verification (CAV'01)*, number 2102 in Lecture Notes in Computer Science (2001), pp. 53–65.

[12] Gerth, R., D. Peled, M. Vardi and P. Wolper, *Simple on-the-fly automatic verification of linear temporal logic*, in: P. Dembinski and M. Sredniawa, editors, *Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification*, IFIP Conference Proceedings **38** (1995), pp. 3–18.

[13] Latvala, T., A. Biere, K. Heljanko and T. Junttila, *Simple bounded LTL model checking*, in: A. J. Hu and A. K. Martin, editors, *Formal Methods in Computer-Aided Design; 5th International Conference, FMCAD 2004*, Lecture Notes in Computer Science (2004).

[14] Martin, A. J., *The design of a self-timed circuit for distributed mutual exclusion*, in: H. Fuchs, editor, *Proceedings of the 1985 Chapel Hill Conference on VLSI* (1985), pp. 245–260.

[15] Moskewicz, M., C. Madigan, Y. Zhao, L. Zhang and S. Malik, *Chaff: Engineering an efficient SAT solver*, in: *39th Design Automation Conference*, Las Vegas, 2001, pp. 530–535.

[16] Sheridan, D., *The optimality of a fast CNF conversion and its use with SAT*, Technical Report APES-82-2002, APES Research Group (2004), available from `http://www.dcs.st-and.ac.uk/~apes/apesreports.html`.

[17] Wolper, P., *Constructing automata from temporal logic formulas: A tutorial*, in: *Lectures on Formal Methods in Performance Analysis (First EEF/Euro Summer School on Trends in Computer Science)*, Lecture Notes in Computer Science **2090** (2001), pp. 261–277.