Electronic Notes in Theoretical Computer Science 119 (2005) 83–101 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Bounded Model Checking with SNF, Alternating Automata, and Bu¨chi Automata

# Daniel Sheridan[1](#_bookmark0)

*School of Informatics University of Edinburgh Edinburgh, UK*

Abstract

Model checking of LTL formulæ is traditionally carried out by a conversion to Bu¨chi 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 Bu¨chi automata are used from both a theoretical and an experimental perspective.

*Keywords:* Bounded model checking, SNF, LTL, Bu¨chi automata, Alternating automata

# Introduction

Before the introduction of bounded model checking in 1999 [[1](#_bookmark10)], 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 au- tomaton for a given LTL formula has been extensive and varied. There is liter- ature giving improvements to the original “GPVW” conversion algorithm [[12](#_bookmark21)] including simplifying the LTL before conversion, and the automaton after conversion (eg, [[7](#_bookmark16)]) as well as the conversion itself. Some recent work [[10](#_bookmark19),[11](#_bookmark20)]

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

1571-0661 © 2005 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2004.12.024

proposes the use of alternating automata (AA) as an intermediate represen- tation 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 Bu¨chi automaton.

Bounded model checking (BMC) has traditionally taken a different ap- proach: the original paper [[1](#_bookmark10)] gives an encoding from LTL directly to propo- sitional logic. Being defined recursively on the structure of the formula this (na¨ıvely) appears to be exponential size in the number of states, although with careful treatment [[5](#_bookmark14)] the result is polynomial size. An alternative encoding [[9](#_bookmark18)] 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](#_bookmark15)]. The only experimental comparison [[5](#_bookmark14)] 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-propo- sitional conversion and the conversions via automata as “syntactic” versus “semantic” [[5](#_bookmark14)], we demonstrate in this paper the close correspondence between SNF and alternating automata and their conversion procedures from LTL. We review the use of Bu¨chi 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.

# Background

* 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 *abω*: 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 *lLk* =˙ (*π*(*k*) = *π*(*l*)) [2](#_bookmark1) . 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 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 *π*(*k*) and *π*(*l*) rather than the transition as used in the original presentation [[1](#_bookmark10)]

and the value of *l* simultaneously.

* 1. *The Separated Normal Form*

SNF [[8](#_bookmark17)] is a clause-like normal form based on the Separation Theorem of Gabbay, with the general form G *i*(*Pi* → *Fi*) where *Pi* → *Fi*, called *rules* are restricted to (writing *p* and *f* for propositional formulæ)

Initial rules of the form start → *f* where start holds only in the initial state of each path

Global invariant rules *p* → *f* with no temporal operator

Global step rules *p* → X *f*

Global eventuality rules *p* → 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](#_bookmark2) to the initial formula set {start → *f* } [[9](#_bookmark18)]. The transformations introduce new

variables identified by the syntax *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 *ψ*(G *f* ) to say that G *f* occurs in *ψ*, while *ψ*(*g*) stands for the formula obtained by substituting every occurrence of G *f* with *g* in *ψ*; similarly for the other temporal operators.

* 1. *Bu¨chi Automata*

We cover Bu¨chi automata only briefly here; we direct the interested reader to the tutorial paper by Wolper [[17](#_bookmark24)].

Definition 2.1 A Bu¨chi automaton B is defined by the tuple ⟨*Q,* Σ*, δ,I,T* ⟩ where *Q* is the set of states; Σ is the alphabet of transition labels; *δ* is the transition function *Q* → 22Σ×*Q*; *I* ⊆ *Q* is the set of initial states; *T* ⊆ *Q* is the set of accepting states.

Note that we use 2Σ 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 Bu¨chi automaton is a path through the automaton; it is accept- ing if the states in *T* are visited an infinite number of times. That is,

Definition 2.2 A *run* of a Bu¨chi automaton B with respect to a word *u*0*u*1 *...*

∈ Σ*ω* is a sequence of states in *q*0*q*1 *...* ∈ *Qω* with *q*0 ∈ *I* and ∀*i*∃*αi* ⟨*αi, qi*+1⟩ ∈

Snf[X]({*ϕ* → *ψ*(X *f* )}∪ Γ) =˙

Snf ({*ϕ* → *ψ*(F *f* )}∪ Γ) =˙

⎧*ϕ* → *ψ*(X *f* )⎫ ∪ Γ

⎩ X *f* → X *f* ⎭

⎧ ⎫

⎨ ⎬

⎨*ϕ* → *ψ*(F *f* )⎬ ∪ Γ

[F]

⎩ F *f* → F *f* ⎭

Snf[G]

({*ϕ* → *ψ*(G *f* )}∪ Γ) =˙ ⎧⎨

*ϕ* → *ψ*(*f* ∧ XG *f*)

⎫⎬ ∪ Γ

⎧⎪⎪⎨

⎩XG *f* → X (*f* ∧ XG *f* )⎭

Snf[U]({*ϕ* → *ψ*(*f* U *g*)}∪ Γ) =˙

X(*f* U *g*) → X (*g* ∨ (*f* ∧ X(*f* U *g*)))

*ϕ* → *ψ*(*g* ∨ (*f* ∧ X(*f* U *g*)))

⎫⎪⎪⎬

∪ Γ

Snf[R]

({*ϕ* → *ψ*(*f* R *g*)}∪ Γ) =˙

⎪⎪⎩

⎨

⎧

*ϕ* → F *g*

*ϕ* → *ψ*(*g* ∧ (*f* ∨ X(*f* R *g*)))

⎪⎪⎭

⎬ ∪ Γ

⎫

⎩X(*f* R *g*) → X (*g* ∧ (*f* ∨ X(*f* R *g*)))⎭

Fig. 1. The transformation function for SNF.

*δ*(*qi*) such that *ui* ∈ *αi*. A run is *accepting* if infinitely many states in the run are members of *T* .

A *generalised* Bu¨chi automaton (GBA) has a set of accepting sets T ⊆ 2*Q*; each set must be visited infinitely often for acceptance. A GBA may be reduced to a classical Bu¨chi automaton but incurs a linear blowup of *O*(|T |).

* 1. *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 ∧*,* ∨ and states. Alternating automata are exponentially more succinct than Bu¨chi automata.

There are two presentations of LTL to automata conversion via alternating

automata. We follow the slightly unconventional presentation by Gastin and Oddoux [[11](#_bookmark20)]: 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](#_bookmark19)] 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-Bu¨chi accepting

condition, while Fritz uses a Bu¨chi condition. We can disregard this: for the alternating automata under consideration, a Bu¨chi condition *F* ⊆ *Q* is equivalent to the co-Bu¨chi condition *Q* \ *F* .

Definition 2.3 An *alternating co-Bu¨chi automaton* A is defined by the tuple

⟨*Q,* Σ*, δ,I,F* ⟩ where *Q* is the set of states; Σ is the alphabet of transition labels; *δ* is the transition function *Q* → 22Σ×2*Q* ; *I* ⊆ 2*Q* is the set of initial combinations of states; *F* ⊆ *Q* is the set of final states

As for the Bu¨chi automaton definition above, the transition labels are from 2Σ; 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,* ±) deter- mined by the transitions, such that ∀*q* ∈ *Q,* ∀ ⟨*α, q*'⟩ ∈ *δ*(*q*)*, q*' ± *q*. That is, transitions are only permitted from a state to a lower or equal state. The re-

sult of this restriction is that the only loops in very weak co-Bu¨chi alternating automaton (VWAA) are self-loops.

Definition 2.4 A run *σ* of a VWAA on a word *u*0*u*1 *...* ∈ Σ*ω* is a labelled DAG ⟨*V, E, λ*⟩ with *V* partitioned into levels *Vi*, *V* = *i*∈N *Vi* and *E* ⊆ *i*∈N *Vi*×*Vi*+1. *λ* : *V* → *Q* labels the vertices of the graph with states of the au- tomaton. *Vi* may be seen as a multiset of elements of *Q*. The graph is related

to the word and the automaton by *λ*(*V*0) ∈ *I* and ∀*v* ∈ *Vi,* ∃ ⟨*λ*(*v*)*, α, s*'⟩ ∈ *δ*(*λ*(*v*))*.ui* ∈ *α* ∧ *s*' = *λ*(*E*(*v*)) A run is accepting if every infinite branch of *σ* has only a finite number of nodes with labels in *F* .

* + 1. *LTL to VWAA Conversion*

We report here the conversion procedure given by Gastin and Oddoux. The set operator ⊗ constructs the conjunctions of two sets of disjunctive normal form transitions: *X* ⊗ *Y* = {⟨*α*1 ∩ *α*2*, e*1 ∧ *e*2⟩ | ⟨*α*1*, e*1⟩ ∈ *X,* ⟨*α*2*, e*2⟩ ∈ *Y* }. The

overbar operator *ψ*¯ converts *ψ* to a set-style disjunctive normal form represen-

tation: a set of conjunctions of atomic propositions or temporal subformulæ. For an LTL formula *ϕ* over atomic propositions *P* , the VWAA A*ϕ* =

⟨*Q,* Σ*, δ,I,F* ⟩ is given by

¬*p*

G (*p* → F *q*)

*q*



G (*p* → F *q*)

\*

\* F *q*

*q*

\*

T

F *q* G (*p* → F *q*)

F *q* G (*p* → F *q*)

F *q* F *q* G (*p* → F *q*)

T T G (*p* → F *q*)

Fig. 2. Example alternating automaton (left) and part of a run over the input *σ* = {*p*}{}{*p*}{*pq*} ···

(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)
* Σ = 2*P* ; *I* = *ψ*¯; *F* is the set of formulæ of the form *ψ*1 U *ψ*2 or F *ψ*1
* *δ* is defined as

*δ*(T) = {⟨Σ*,* T⟩}

*δ*(*p*) = {⟨{*a* ∈ Σ | *p* ∈ *a*}*,* T⟩}

*δ*(¬*p*) = {⟨{*a* ∈ Σ | *p* ∈*/ a*}*,* T⟩} *δ*(X *ψ*) = {⟨Σ*, e*⟩ | *e* ∈ *ψ*¯} *δ*(F *ψ*) = ∆(*ψ*) ∪ ({⟨Σ*,* F *ψ*⟩})

*δ*(G *ψ*) = ∆(*ψ*) ⊗ {⟨Σ*,* G *ψ*⟩})

*δ*(*ψ*1 U *ψ*2) = ∆(*ψ*2) ∪ (∆(*ψ*1) ⊗ {⟨Σ*, ψ*1 U *ψ*2⟩})

*δ*(*ψ*1 R *ψ*2) = ∆(*ψ*2) ⊗ (∆(*ψ*1) ∪ {⟨Σ*, ψ*1 R *ψ*2⟩})

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

∆(*ψ*) = *δ*(*ψ*) if *ψ* ∈ *Q*

∆(*ψ*1 ∧ *ψ*2) = ∆(*ψ*1) ⊗ ∆(*ψ*2)

∆(*ψ*1 ∨ *ψ*2) = ∆(*ψ*1) ∪ ∆(*ψ*2)

We give an example VWAA corresponding to the LTL formula G (*p* → F *q*) in Figure [2](#_bookmark3) along with a sample run.

* + 1. *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*, *Ci* ⊆ *Q*. A sequence of configurations over a word *u*0*u*1 *...* ∈ Σ*ω* is accepting if there exists a set of edges *E* partitioned into *Ei* ⊆ *Ci* × *Ci*+1 such that ∀*q* ∈ *Ci*∃ ⟨*α, q*'⟩ ∈ *δ*(*q*)*.ui* ∈ *α*∧*q*' ⊆ *Ei*(*q*) and every path *q*0*q*1 *...* such that *qi*+1 ∈ *Ei*(*qi*) contains only finitely many occurrences of the members of *F* .

For example, consider the run in Figure [2](#_bookmark3). The corresponding sequence of configurations is

{G (*p* → F *q*)}

{F *q,* G (*p* → F *q*)}

{F *q,* G (*p* → F *q*)}

{F *q,* G (*p* → F *q*)}

{G (*p* → 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 *Ci* with acceptance described by edges *Ei* may be directly translated into the DAG *i*∈N *Ci, i*∈N *Ei,I* 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 *Ci* = *v*∈*Vi λ*(*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 *Ci*. This is assured by examination of the DAG, since every element of each *Vi* must be followed by an accepting sequence of edges.

* + 1. *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 *Ci* with a superset of *Ci* (similarly *Vi*) provided that successive configurations (lev- els 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.

# Bounded Model Checking Encodings

Having discussed three representations of LTL formulæ suited to model check- ing we now turn to the way that these representations can be used for bounded model checking. The encoding of Bu¨chi automata was discussed by de Moura et al. [[6](#_bookmark15)] as well as Clarke et al. [[5](#_bookmark14)]. The use of SNF for bounded model check- ing was the subjection of a paper by Frisch et al. [[9](#_bookmark18)]. 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](#_bookmark10)] has the potential to simplify the resulting formula [3](#_bookmark5) considerably:

*M, f* )*k* := *M*)*k* ∧ enc*c*(*f, k*) ∧

enc*n*(*f, k*) ∨

*k*

*l* =0

(*l*L*k* ∧ enc*l*(*f, k, l*))

where enc*c,* enc*n,* and enc*l* denote the common, finite, and loop encodings as described below.

* 1. *Bounded Model Checking with Bu¨chi Automata*

We present a variation on the encoding of de Moura et al. [[6](#_bookmark15)], 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 Bu¨chi automata: the complexity of checking multiple acceptance sets is much lower than the overhead of conversion to classical Bu¨chi automata.

All paths accepted by a Bu¨chi automaton are infinite — formulæ with finite counterexamples such as F *φ* are encoded with a trivial infinite loop. The finite prefix case is therefore never accepting, and we deduce that enc*n*(*f, k*) = ⊥.

Given a generalised Bu¨chi automaton representing LTL formula *f* , B*f* =

⟨*Q,* Σ*, δ,I,* T ⟩, we encode the current state *q* ∈ *Q* as a base two integer in the range 0 *...* |*Q*|− 1: there is a one-to-one mapping *ϵ* ⊆ *Q* ×{*i* | 0 ≤ *i <* |*Q*|− 1}. That is, for each state *i*, we have a set of propositional variables *qn*(*i*), 0 ≤ *n <* [log2(|*Q*|)| and we write *q*)*i* for the assertion that the bit pattern *q*0*q*1 *...*

3 The formula given is derived from the usual BMC formulation as given in Biere et al. [[1](#_bookmark10)]. We write *M* )k for the en coding of the model, lLk for the constraint that the path is a

*k*-*l*-loop, but we omit the 0≤l<k ¬lLk non-loop constraint as suggested by [[3](#_bookmark11)]

is the base two representation of *ϵ*(*q*). For Bu¨chi automata representing LTL, Σ is the set of propositions in that model; the encoding of elements *a* ∈ Σ 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*B*f* (*i, k*) = ( *s*) ∧ *a*) ∧ *s* )

*i i* ' *i*+1

⟨*s,α,s*'⟩∈*δ a*∈*α*

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

*I*B (*k*) = *s*)0

*f*

*s*∈*I*

Finally, we encode the acceptance sets. The Bu¨chi acceptance condition is that each member of T is visited infinitely often. As we have ruled out finite path prefixes, we know that all paths being considered are of the form

*abω*. If we assert as part of the loop encoding that the corresponding paths in the Bu¨chi automaton follow the same pattern, we can simply require that representatives from each acceptance set appear in the loop (ie, in *b*):

*k*

*F*B (*k, l*) = *s*)

*i*

*f*

*T* ∈T *i*=*l s*∈*T*

Thus we have

*k*−1

enc*c*(*f, k*) = *I*B*f* (*k*) ∧ *T*B*f* (*i, k*)

*i*=0

enc*n*(*f, k*) = ⊥

enc*l*(*f, k, l*) = *F*B*f* (*k, l*) ∧

[log2(|*Q*|)|−1

*i*=0

*qi*(*l*) ↔ *qi*(*k*)

Although the LTL to Bu¨chi 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*B which is quadratic: *O*(|T |*k*2).

*f*

* 1. *Bounded Model Checking with Alternating Automata*

The encoding of alternating automata is very similar to Bu¨chi 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 conjunc- tions of states.

Given a VWAA representing LTL formula *f* , A*f* = ⟨*Q,* Σ*, δ,I,F* ⟩, we encode the presence of a a state *q* in the *i*th configuration by the variable

*q*(*i*). A configuration is encoded as a conjunction of its members: we write

*C*)*i* = *q*∈*C q*(*i*), with ∅)*i* = ⊥. 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](#_bookmark4). 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 la- belled with a set of sets of atomic propositions: the set of permitted assign- ments to propositions. These can be denoted [4](#_bookmark6) by a conjunction of literals

where *p* ∧ *q* denotes {*a* ∈ Σ | *p* ∈ *a*}∩ {*a* ∈ Σ | *q* ∈ *a*}. We write *α*)*i* for the conjunction of literals representing *α* ∈ 2Σ — this is particularly convenient as the implementation of the LTL to VWAA conversion [[11](#_bookmark20)] produces these

conjunctions directly.

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

*T*A*f* (*i, k*) = *q*(*i*) → *α*) ∧ *s* )

*i* ' *i*+1

*q*∈*Q*

⟨*α,q*'⟩∈*δ*(*q*)

and the initial set of configurations is encoded

*I*A (*k*) = *C*0)0

*f*

*C*0 ∈*I*

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 configura- tion 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 *k*th configuration: this will hold even if the first empty configuration is before *k*.

*P*A*f* (*k*) = ¬*q*(*k*)

*q*∈*Q*

For the loop case, we cannot simply check for an infinite number of oc- currences of the members of *F* as the co-Bu¨chi 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](#_bookmark20)]

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*A*f* (*k, l*) =

*k*

*q*)*i* →

*α*)*i* ∧ *q*')*i*+1

*q*∈*F i*=*l*

⟨*α,q*'⟩∈*δ*(*q*)

*q*∈*/q*'

Thus we have

*k*−1

enc*c*(*f, k*) = *I*A*f* (*k*) ∧ *T*A*f* (*i, k*)

*i*=0

enc*n*(*f, k*) = *P*A*f* (*k*)

enc*l*(*f, k, l*) = *F*A*f* (*k, l*) ∧ *q*(*l*) ↔ *q*(*k*)

*q*∈*Q*

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*A*f* which is quadratic in *k*.

* 1. *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 → *f* ) specify initial conditions:

*I*Ψ*f* (*k*) = *f* )

0

(start→*f* )∈Ψ*f*

Global invariant rules *p* → *f* are constraints on the configurations of indi-

vidual states:

*P*Ψ*f* (*i*) = *p* → *f* )

*i*

(*p*→*f* )∈Ψ*f*

Global step rules (*p* → 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*Ψ*f* (*i, k*) =

(*p*→X *f* )∈Ψ*f*

*T* (*k*) =

*k*

*p*)*i*

→ *f* )*i*+1

*n*

*k*

Ψ*f*

(*p*→X *f* )∈Ψ*f*

*k*

*p*)*k* → ⊥

*l* (*k, l*) =

*T*

Ψ*f*

(*p*→X *f* )∈Ψ*f*

*p*)*k* → *f* )*l*

Global eventuality rules (*p* → F *f* ) are superficially similar to acceptance conditions but can be interpreted more directly — as in [[1](#_bookmark10)]. For a finite

prefix, this is simply a disjunction over states; for a *k*-loop of the form *abω*, evaluating F during *b* is equivalent to evaluating it at the start of *b*.

*Fn* (*k*) =

Ψ*f*

*k*

*p*)*i* → *f* )*j*

*k*

*i*=0 (*p*→F *f* )∈Ψ*f*

*k*

*j*=*i*

*l* (*k, l*) =

*F*

Ψ*f*

*k*

*p*)*i* → *f* )*j*

*k*

*i*=0 (*p*→F *f* )∈Ψ*f*

*j*=min(*i,l*)

Thus we have

enc*c*(*f, k*) = *I*Ψ*f* (*k*) ∧

*k*

*P*Ψ*f* (*i*) ∧

*i*=0

*k*−1

*T*Ψ*f* (*i, k*)

*i*=0

enc*n*(*f, k*) = *Tn* (*k*) ∧ *Fn* (*k*)

Ψ*f* Ψ*f*

enc*l*(*f, k, l*) = *T l* (*k, l*) ∧ *F l* (*k, l*)

Ψ*f*

Ψ*f*

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*.

* + 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 F [[9](#_bookmark18)]. Using the bound operator, which holds only

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

⎧⎪⎪⎨

'

[F]

X F *f* → X (*f* ∨ X F *f*)

Snf

({*ϕ* → F(F *f* )}∪ Γ) =˙

*ϕ* → *ψ*(*f* ∨ XF *f* )

⎫⎪⎪⎬

∪ Γ

⎪⎪⎩

bound → ¬XF *f*

⎪⎪⎭

This direct approach affects the length of the counterexample: evaluating 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 *k x* ∨ *n*−1 *x* ≡

*i*=*n*

*i*=*l*

*k*

*i*=*l*

*x* for *l < n* ≤ *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 F *f* is not a state variable but rather is a simple propositional variable, and

this is reflected in the encoding.

⎧⎪ *ϕ* → *ψ*(*f* ∨ XF *f* ∨ F *f* ) ⎫⎪

''

[F]

⎪⎨F *f* → AtLoop(*f* ∨ X F *f* )⎪⎬ ∪ Γ

Snf

({*ϕ* → F(F *f* )}∪ Γ) =˙

This gives us the encoding

⎪⎪⎩

XF *f* → X (*f* ∨ XF *f*)

bound → ¬X F *f*

⎪⎪⎭

enc' (*f, k*) = enc*c*(*f, k*) ∧

*c*

*f* )*k*

(bound→*f* )∈Ψ*f*

enc' (*f, k*) = *Tn* (*k*) ∧

*n*

Ψ*f*

(*p*→AtLoop(*f* ))∈Ψ*f*

*p* → ⊥

enc'(*f, k, l*) = *T l* (*k, l*) ∧

*l*

Ψ*f*

*p* → *f* )*l*

(*p*→AtLoop(*f* ))∈Ψ*f*

For an alternative presentation of this approach, see Cimatti et al. [[4](#_bookmark13)].

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.

# SNF versus Automata

We have examined two established methods of encoding LTL for bounded model checking and introduced a third: the encoding via alternating au- tomata. We now clarify the relationships and relative advantages of the en- codings.

* 1. *SNF and Alternating Automata*

The configuration view of alternating automata makes it apparent that Fix- point 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 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 cor- responds 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-Bu¨chi acceptance set. The difficulty of checking the co- Bu¨chi acceptance condition are sidestepped by the start-of-loop projection introduced in Section [3.3.1](#_bookmark7). 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 *Lk*, 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](#_bookmark20)] and simulation [[10](#_bookmark19)] reductions, some of which do not project directly to SNF; the advantages of these have the potential to outweigh the drawbacks of the encoding.

* 1. *SNF and Alternating Automata versus Bu¨chi Automata*

Most of the encoding issues discussed above apply equally to Bu¨chi 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 consider- ably. 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 Bu¨chi 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 *abi*. For example, consider the word *xx*(*abb*)*ω*, which is recognised in this form by the specification *F* (*b* ∧ *F* (*a*)) using the direct or SNF encodings, but which must be expanded to *xxabb*(*abb*)*ω* to be

recognised by the automata-based encodings.

* 1. *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 Bu¨chi and co-Bu¨chi 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.

* 1. *Empirical Results*

To demonstrate some of the differences between the approaches we give a se- lection of experimental results comparing a variety of BMC encodings. The existing encodings, the original BMC encoding [[1](#_bookmark10)] (marked “Orig” in the re- sults), the SNF encoding and its refinement [[9](#_bookmark18)] (“SNF” and “FIX”) are com- pared against Bu¨chi automata, in this case the Etessami and Holzmann [[7](#_bookmark16)] procedure (“TMP”), and the VWAA produced by the tool from Gastin and Oddoux [[11](#_bookmark20)] 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](#_bookmark23)] with the specifications given in Frisch et al. [[9](#_bookmark18)], at several different bounds to illustrate

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Enc. | *k* | Size | Vars | Time | Size | Vars | Time | Size | Vars |  | Time |
|  | | Access | ibility | (4,2) | Overt | aking 1 | (5,5) | Overt | aking | 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](#_bookmark12)] with an improved CNF conversion [[16](#_bookmark25)]; timings were made in the SAT solver zChaff [[15](#_bookmark26)].

Table [1](#_bookmark8) 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 Bu¨chi automata produced by TMP: a par- ticularly 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](#_bookmark9)). Here we see that the Bu¨chi 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.

# 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 Bu¨chi 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](#_bookmark19)] 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-Bu¨chi condition, further improving the performance. A possible alternative technique for encoding the co-Bu¨chi condition is to adapt the new linear-space encoding of Latvala et al. [[13](#_bookmark22)].

# Acknowledgements

I would not have begun to investigate these encoding methods without the helpful comments from the reviewers for CHARME 2003. I am also indebted to Dr Paul Jackson for extensive discussion and input on the topics in this paper.

# 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.

1. 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.
2. 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.
3. 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).
4. 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.
5. 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.
6. Etessami, K. and G. J. Holzmann, *Optimizing Bu¨chi 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.

1. Fisher, M., *A resolution method for temporal logic*, in: *Proceedings of the Twelfth International Joint Conference on Artiﬁcial Intelligence (IJCAI)* (1991), pp. 99–104.
2. 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.

1. Fritz, C., *Constructing Bu¨chi automata from linear temporal logic using simulation relations for alternating Bu¨chi 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.
2. Gastin, P. and D. Oddoux, *Fast LTL to Bu¨chi 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.
3. 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.
4. 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).
5. 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.
6. 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.
7. 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> .
8. 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.