A Fixpoint Based Encoding for Bounded Model Checking

Alan Frisch[1], Daniel Sheridan[1], and Toby Walsh[2]

[1] University of York, York, UK
{frisch,djs}@cs.york.ac.uk
[2] Cork Constraint Computation Center, University College Cork, Cork, Ireland
tw@4c.ucc.ie

Abstract. The *Bounded Model Checking* approach to the LTL model checking problem, based on an encoding to Boolean satisfiability, has seen a growth in popularity due to recent improvements in SAT technology. The currently available encodings have certain shortcomings, particularly in the size of the clause forms that it generates. We address this by making use of the established correspondence between temporal logic expressions and the fixpoints of predicate transformers as used in symbolic model checking. We demonstrate how an encoding based on fixpoints can result in improved performance in the SAT checker.

1 Introduction

Bounded Model Checking (BMC) [1] is an encoding to Boolean Satisfiability (SAT) of the LTL model checking problem. The encoding is achieved by placing a bound on the number of time steps of the model that are to be checked against the specification. The resulting Boolean formula contains variables representing the state variables of the model at each state along a path, together with constraints requiring the path to be contained within the model and to violate the specification. The result of the SAT checker is thus a path in the model which is a counterexample to the specification, or failure, which means that no such path exists within the bound.

The encoding of the LTL specification in BMC is defined recursively on the structure of the formula. While for simple specifications this is sufficient, more complex specifications such as bounded existence and response patterns [5] lead to an exponential blowup in the size of the resulting Boolean formula. Recent improvements to the encoding in NuSMV [3] have not removed this restriction.

The fixpoint characterisations of temporal operators [6] have been exploited in other model checking systems such as SMV [11]; we discuss an approach to their use in an encoding of LTL for BMC which produces more compact encodings which can be solved more quickly in the SAT solver.

2 The Bounded Model Checking Encoding

The bounded model checking encoding represents a single bounded path π_{BMC} of length k as a propositional formula, and checks that it violates the bounded

$$
\begin{array}{lll}
(M,\pi) \models_k^i a & \Leftrightarrow a \in L(\pi(i)) \quad \text{for atomic } a \\
(M,\pi) \models_k^i \neg f_1 & \Leftrightarrow (M,\pi) \not\models_k^i f_1 \\
(M,\pi) \models_k^i f_1 \wedge f_2 & \Leftrightarrow (M,\pi) \models_k^i f_1 \text{ and } (M,\pi) \models_k^i f_2 \\
(M,\pi) \models_k^i f_1 \vee f_2 & \Leftrightarrow (M,\pi) \models_k^i f_1 \text{ or } (M,\pi) \models_k^i f_2 \\
(M,\pi) \models_k^i \mathbf{X}\, f_1 & \Leftrightarrow \begin{cases} (M,\pi) \models_k^{i+1} f_1 & \text{if } \pi \text{ is a } k\text{-loop} \\ (M,\pi) \models_k^{i+1} f_1 \wedge i < k & \text{otherwise} \end{cases} \\
(M,\pi) \models_k^i \mathbf{F}\, f_1 & \Leftrightarrow \begin{cases} \exists j, i \le j.(M,\pi) \models_k^j f_1 & \text{if } \pi \text{ is a } k\text{-loop} \\ \exists j, i \le j \le k.(M,\pi) \models_k^j f_1 & \text{otherwise} \end{cases} \\
(M,\pi) \models_k^i \mathbf{G}\, f_1 & \Leftrightarrow \begin{cases} \forall j, i \le j.(M,\pi) \models_k^j f_1 & \text{if } \pi \text{ is a } k\text{-loop} \\ \bot & \text{otherwise} \end{cases} \\
(M,\pi) \models_k^i [f_1\, \mathbf{U}\, f_2] & \Leftrightarrow \begin{cases} \exists j, i \le j.(M,\pi) \models_k^j f_2 \wedge \forall n, i \le n < j.(M,\pi) \models_k^n f_1 & \text{if } \pi \text{ is a } k\text{-loop} \\ \exists j, i \le j \le k.(M,\pi) \models_k^n f_2 \wedge \forall n, i \le n < j.(M,\pi) \models_k^n f_1 & \text{otherwise} \end{cases} \\
(M,\pi) \models_k^i [f_1\, \mathbf{R}\, f_2] & \Leftrightarrow \begin{cases} \exists j, i \le j.(M,\pi) \models_k^j f_1 \wedge \forall n, i \le n \le j.(M,\pi) \models_k^j f_2 & \text{if } \pi \text{ is a } k\text{-loop} \\ \exists j, i \le j \le k.(M,\pi) \models_k^j f_1 \wedge \forall n, i \le n \le j.(M,\pi) \models_k^j f_2 & \text{otherwise} \end{cases}
\end{array}
$$

Fig. 1. The Bounded Semantics of LTL

semantics (Figure 1) of the specification f. That is, that $(M, \pi_{BMC}) \not\models_k f$. We will write the bounded model checking encoding of a problem with bound k as

$$[\![M, \pi, \neg f]\!]_k$$

Biere et al. [1] show how incorporating a check for loops in the transition graph makes bounded model checking complete for sufficiently large bound k. The resulting theorem is paraphrased here.

Theorem 1. *Let f be an LTL formula, M a Kripke structure, and π a path in M. Then $(M, \pi) \models f$ iff there exists $k \in \mathbb{N}$ with $(M, \pi) \models_k f$.*

The encoding is structured as a conjunction of constraints requiring π_{BMC} to be a *valid path in $M* and be a *counterexample of f*. The 'valid path' constraint is a propositional encoding of the transition relation. We can see from the bounded semantics of LTL (Figure 1) that there are two ways of violating each operator in the specification, depending on whether π_{bmc} is a k-loop; the 'counterexample' constraint is therefore a disjunction of the ways in which the specification may be violated.

Given the functions ${}_l\mathrm{L}_k(\pi)$ which holds when π is a k-loop with $\pi(k) = \pi(l)$ and $\mathrm{L}_l(\pi) = \bigvee_{l=0}^{k}\ {}_l\mathrm{L}_k$ which holds when π is any k-loop, the general translation

Table 1. The BMC Encoding for LTL

f	$\llbracket f, \pi \rrbracket_k^i$	${}_l\llbracket f, \pi \rrbracket_k^i$
$\mathbf{G}\, f_1$	$\bot$	$\bigwedge_{j=\min(i,l)}^{k} {}_l\llbracket f_1, \pi \rrbracket_k^j$
$\mathbf{F}\, f_1$	$\bigvee_{j=i}^{k} \llbracket f_1, \pi \rrbracket_k^j$	$\bigvee_{j=\min(i,l)}^{k} {}_l\llbracket f_1, \pi \rrbracket_k^j$
$\mathbf{X}\, f_1$	$i < k \wedge \llbracket f_1, \pi \rrbracket_k^{i+1}$	$\left(i < k \wedge {}_l\llbracket f_1, \pi \rrbracket_k^{i+1}\right) \vee \left(i = k \wedge {}_l\llbracket f_1, \pi \rrbracket_k^{l}\right)$
$f_1\, \mathbf{U}\, f_2$	$\bigvee_{j=i}^{k} (\llbracket f_2, \pi \rrbracket_k^j \wedge \bigwedge_{n=i}^{j-1} \llbracket f_1, \pi \rrbracket_k^n)$	$\bigvee_{j=i}^{k} ({}_l\llbracket f_2, \pi \rrbracket_k^j \wedge \bigwedge_{n=i}^{j-1} {}_l\llbracket f_1, \pi \rrbracket_k^n)$ $\vee \bigvee_{j=l}^{i-1} ({}_l\llbracket f_2, \pi \rrbracket_k^j \wedge \bigwedge_{n=i}^{k} {}_l\llbracket f_1, \pi \rrbracket_k^n \wedge \bigwedge_{n=l}^{j-1} {}_l\llbracket f_1, \pi \rrbracket_k^n)$
$f_1\, \mathbf{R}\, f_2$	$\bigvee_{j=i}^{k} (\llbracket f_1, \pi \rrbracket_k^j \wedge \bigwedge_{n=i}^{j} \llbracket f_2, \pi \rrbracket_k^n)$	$\bigwedge_{j=\min(i,l)}^{k} {}_l\llbracket f_2, \pi \rrbracket_k^j \vee \bigvee_{j=i}^{k} ({}_l\llbracket f_1, \pi \rrbracket_k^j \wedge \bigwedge_{n=i}^{j} {}_l\llbracket f_2, \pi \rrbracket_k^n)$ $\vee \bigvee_{j=l}^{i-1} ({}_l\llbracket f_1, \pi \rrbracket_k^j \wedge \bigwedge_{n=i}^{k} {}_l\llbracket f_2, \pi \rrbracket_k^n \wedge \bigwedge_{n=l}^{j} {}_l\llbracket f_2, \pi \rrbracket_k^n)$

is defined as[3]:

$$\llbracket M, \pi, f \rrbracket_k := \llbracket M, \pi \rrbracket_k \wedge \left(\left(\neg \mathrm{L}_k(\pi) \wedge \llbracket f, \pi \rrbracket_k^0\right) \vee \bigvee_{l=0}^{k} \left({}_l\mathrm{L}_k(\pi) \wedge {}_l\llbracket f, \pi \rrbracket_k^0\right) \right) \tag{1}$$

$\llbracket M, \pi \rrbracket_k$ denotes the encoding of the transition relation of M as a constraint on π with bound k; $\llbracket f, \pi \rrbracket_i^k$ and ${}_l\llbracket f, \pi \rrbracket_i^k$ denote the encoding of the LTL formula f evaluated along path π at time i, where π is a non-looping path and a k-loop to l respectively. These encodings are given in Table 1. Biere et al. show the correctness of some of these encodings in [1]; we will not repeat their proofs here.

3 Exploiting Fixpoints in BMC

The approach that we have taken to making a fixpoint-based encoding for BMC is based on a clause-style normal form for temporal logic. We can then redefine the encoding to specifically take advantage of the properties of the normal form.

3.1 The Separated Normal Form

Gabbay's Separation Theorem [9] states that arbitrary temporal formulæ may be written in the form $\mathbf{G}\left(\bigwedge_i (P_i \Rightarrow F_i)\right)$ where P_i are (strict) past time formulæ and F_i are (non-strict) future time formulæ.

Fisher [7] defines a normal form for temporal logic based on the Separation Theorem and gave a series of transformations for reaching it. The general form of SNF is the same as the separation theorem; the implications $P_i \Rightarrow F_i$ are referred to as *rules*. Since neither LTL nor CTL have explicit past-time operators, Bolotov and Fisher [2] define the **start** operator which holds only at the beginning of time.

$$(M, \pi) \models_k^i \mathbf{start} \Leftrightarrow \pi(i) \in I$$

[3] This comes from definition 15 in [1]

The possible rules are thus

$$\textbf{start} \Rightarrow \bigvee_j l_j \quad \text{An } \textit{initial} \text{ rule} \qquad \bigwedge_i l_i \Rightarrow \mathbf{X} \bigvee_j l_j \quad \text{A } \textit{global } \mathbf{X}\text{-rule}$$

$$\bigwedge_i l_i \Rightarrow \mathbf{F} \bigvee_j l_j \quad \text{A } \textit{global } \mathbf{F}\text{-rule}$$

where l_i and l_j are literals.

The transformation functions $T(\Psi)$ recursively convert a set of rules which do not conform to the normal form into a set of rules which do. To convert any temporal logic formula f to SNF, it is sufficient to apply the transformation rules to the singleton set $\{\textbf{start} \Rightarrow f\}$. For brevity, we do not list the full set of transformations here; in general they are trivially adapted from those in [2], or from standard propositional logic.

$$T_{\mathrm{G}}(\{P \Rightarrow \mathbf{G}\, f\} \cup \Psi) = \begin{Bmatrix} P \Rightarrow f \wedge x \\ x \Rightarrow \mathbf{X}\,(f \wedge x) \end{Bmatrix} \cup \Psi$$

$$T_{\mathrm{U}}(\{P \Rightarrow f\, \mathbf{U}\, g\} \cup \Psi) = \begin{Bmatrix} P \Rightarrow g \vee (f \wedge x) \\ x \Rightarrow \mathbf{X}\,(g \vee (f \wedge x)) \\ P \Rightarrow \mathbf{F}\, g \end{Bmatrix} \cup \Psi$$

$$T_{\mathrm{ren1}}(\{P \Rightarrow \mathbf{G}\, f(\mathbf{F}\, g)\} \cup \Psi) = \begin{Bmatrix} P \Rightarrow \mathbf{G}\, f(x) \\ x \Rightarrow \mathbf{F}\, g \end{Bmatrix} \cup \Psi$$

In each of the above transformations, a new variable x is introduced: the conversion to SNF introduces one variable for each removed operator (in the first two transformations above) in addition to the renaming variables used to flatten the structure of the formula (in the last transformation above).

The transformations to rules are based on the fixpoint characterisations of the LTL operators. All LTL operators can be represented as the fixpoint of a recursive function [6]; the transformations encode the corresponding function as a rule which is required to hold in all states. Only those operators characterised by greatest fixpoints are converted (*always* (**G**) and *weak until* (**W**); *until* (**U**) is converted to *weak until* and *sometime* (**F**) for its transformation), which means that the *sometime* (**F**) operator is left unchanged.

By Tarski's fixpoint theorem [14] we know that a finite number of iterations of a rule is sufficient to find its fixpoint. Thus the instance of the introduced variable at time i holds iff the original operator held at time i. For a formal proof of the correctness of the transformations, see [8].

3.2 Bounded SNF

Although the fixpoint characterisations are given for *unbounded* temporal logic, they are preserved for most of bounded LTL since we have bounded semantics for $\mathbf{X}$. We note that the characterisation of $\mathbf{G}$ is valid if and only if the path is a k-loop; we encapsulate this constraint in the new operator $\mathbf{X_l}$ with semantics

$$(M,\pi) \models_k^i \mathbf{X_l}\, f_1 \Leftrightarrow \begin{cases} (M,\pi(i+1)) \models_k f_1 & \text{if } \pi \text{ is a } k\text{-loop} \\ \perp & \text{otherwise} \end{cases}$$

and modify the transformation accordingly.

The bounded semantics of $\mathbf{G}$ also fails to capture the concept of rules holding in *all reachable states*. We give the semantics for a modified operator $\mathbf{G_k}$ for bounded LTL without the restriction to paths with loops.

$$(M,\pi) \models_k^i \mathbf{G_k}\, f_1 \Leftrightarrow \begin{cases} \forall j, i \leq j.(M,\pi(j)) \models_k f_1 & \text{if } \pi \text{ is a } k\text{-loop} \\ \forall j, i \leq j < k.(M,\pi(j)) \models_k f_1 & \text{otherwise} \end{cases}$$

The correctness of the transformations rely on a sufficient number of instances of the rules occurring. In BMC, this means that the transformations based on fixpoints are correct only when the bound is sufficiently large. It is easy to see, by appealing to the semantics, that the failure mode with an insufficiently large bound is the same as that for the original encoding: no counterexample is found.

Introducing this operator allows us to restate the general form as

$$\mathbf{G_k}\left(\bigwedge_i (P_i \Rightarrow F_i)\right)$$

The rules $P_i \Rightarrow F_i$ are now of the following form:

$\mathbf{start} \Rightarrow \bigvee_j l_j$	An *initial* rule	$\bigwedge_i l_i \Rightarrow \mathbf{X_l} \bigvee_j l_j$	A *global* $\mathbf{X_l}$-rule
$\bigwedge_i l_i \Rightarrow \mathbf{X} \bigvee_j l_j$	A *global* $\mathbf{X}$-rule	$\bigwedge_i l_i \Rightarrow \mathbf{F} \bigvee_j l_j$	A *global* $\mathbf{F}$-rule

with the transformation for the *always* operator being amended to

$$T_{\mathrm{G}}(\{P \Rightarrow \mathbf{G}\, f\} \cup \Psi) = \left\{ \begin{array}{c} P \Rightarrow f \wedge x \\ x \Rightarrow \mathbf{X_l}\,(f \wedge x) \end{array} \right\} \cup \Psi$$

3.3 Encoding Bounded SNF

The distributivity of $\mathbf{G_k}$ follows directly from its semantics; because of the unusual semantics of **start**, this means that any LTL formula may be represented as a conjunction of instances of the following 'universal' rules:

Table 2. The BMC Encoding for SNF-LTL

f	$[\![f,\pi]\!]_k^0$	${}_l[\![f,\pi]\!]_k^0$
$\mathbf{start} \Rightarrow f_1$	$[\![f_1,\pi]\!]_k^0$	${}_l[\![f_1,\pi]\!]_k^0$
$\mathbf{G_k}(f_1 \Rightarrow \mathbf{X_l}\, f_2)$	$\bot$	$\bigwedge_{n=0}^{k} \left({}_l[\![f_1,\pi]\!]_k^n \Rightarrow {}_l[\![f_2,\pi]\!]_k^{n+1}\right)$
$\mathbf{G_k}(f_1 \Rightarrow \mathbf{X}\, f_2)$	$\bigwedge_{n=0}^{k-1} \left([\![f_1,\pi]\!]_k^n \Rightarrow [\![f_2,\pi]\!]_k^{n+1}\right)$	$\bigwedge_{n=0}^{k} \left({}_l[\![f_1,\pi]\!]_k^n \Rightarrow {}_l[\![f_2,\pi]\!]_k^{n+1}\right)$
$\mathbf{G_k}(f_1 \Rightarrow \mathbf{F}\, f_2)$	$\bigwedge_{n=0}^{k} \left([\![f_1,\pi]\!]_k^n \Rightarrow \bigvee_{m=n}^{k} [\![f_2,\pi]\!]_k^m\right)$	$\bigwedge_{n=0}^{k} \left({}_l[\![f_1,\pi]\!]_k^n \Rightarrow \bigvee_{m=\min(n,l)}^{k} {}_l[\![f_2,\pi]\!]_k^m\right)$

$$\mathbf{start} \Rightarrow \bigvee_j l_j \qquad \mathbf{G_k}\left(\bigwedge_i l_i \Rightarrow \mathbf{X_l} \bigvee_j l_j\right)$$

$$\mathbf{bound} \Rightarrow \bigvee_j l_j \qquad \mathbf{G_k}\left(\bigwedge_i l_i \Rightarrow \mathbf{X} \bigvee_j l_j\right)$$

$$\mathbf{G_k}\left(\bigwedge_i l_i \Rightarrow \mathbf{F} \bigvee_j l_j\right)$$

Although it is simple to encode these rules using the standard BMC encodings in Table 1, we can take advantage of the limited nesting depth characteristic of these normal forms to define a more efficient encoding, in the same way as for the depth 1 case in [3] and [13]. The more efficient encodings are given in Table 2. Note that although we make use of the BMC encodings, they are only used for purely propositional formulæ. No further proof of these encodings is required: they are a trivial simplification of those proved in [1].

For propositional f, $[\![f,\pi]\!]_k^i \equiv {}_l[\![f,\pi]\!]_k^i$, so we deduce from Table 2 that this relationship also holds for many cases where f is a rule. The obvious optimisation to make is to introduce an extra constraint to Equation (1) which holds regardless of the whether π is a loop; in many circumstances, the checks for the looping nature of π cancel each other out entirely. While this type of optimisation can be made with the standard BMC encoding, it only occurs where operators are not nested; the renaming effect of SNF simplifies this optimisation.

3.4 The Fixpoint Normal Form

We noted in Section 3.1 that SNF converts only the greatest fixpoint operators, leaving rules containing the *sometime* operator; we see from Table 2 that these rules are the pathological case for this encoding. Converting the *sometime* operator in the same way requires care.

A transformation based directly on the fixpoint characterisation would be

$$T_F(\{P \Rightarrow \mathbf{F}\, f\} \cup \Psi) = \left\{ \begin{array}{c} P \Rightarrow f \vee x \\ x \Rightarrow \mathbf{X}\,(f \vee x) \end{array} \right\} \cup \Psi$$

The problem stems from the disjunction in the second rule. Since we are trying to show satisfiability, it is simple to satisfy each occurrence of the rule by setting the right hand disjunct to true at all time: the rule can always be satisfied. Since we are interested only in the bounded semantics of the operator, it is possible to break this chain at the bound by introducing an extra operator:

$$(M, \pi) \models_k^i \mathbf{bound} \Leftrightarrow i \geq k$$

The transformation is now

$$T_F(\{P \Rightarrow \mathbf{F}\, f\} \cup \Psi) = \left\{ \begin{array}{c} P \Rightarrow f \vee x \\ x \Rightarrow \mathbf{X}\, (f \vee x) \\ \mathbf{bound} \Rightarrow f \vee \neg x \end{array} \right\} \cup \Psi$$

3.5 Correctness of the Fixpoint Normal Form Transformation

We take the outline of the proof from [8]. For a transformation T to preserve the semantics of an arbitrary formula f, we require that

> for all models M and for all LTL formulæ f, $(M, \pi) \models_k f$ iff there exists an M' such that $M \sim^x M'$ and $(M, \pi) \models_k \tau(f)$

where x is a new propositional variable introduced, and $M \sim^x M'$ if and only if M differs from M' in at most the valuation given x. We express this in temporal logic with quantification over propositions (QPTL)[4] as $\vdash_{\mathrm{QPTL}} f \Leftrightarrow \exists x.T(f)$. The proof is given for the case that the rule set is a singleton set, since for all transformations, T is independent of Ψ. The proofs may easily be extended to non-empty Ψ.

Lemma 1. *For sufficiently large k, $(M, \pi) \models_k \mathbf{F}\, f_1$ if and only if $(M', \pi) \models_k (x \vee f_1)$ and $(M', \pi) \models_k \mathbf{G_k}(x \Leftrightarrow \mathbf{X}(x \vee f_1))$ where $M \sim^x M'$.*

Proof. Consider the fixpoint expression $\tau(Z) = f_1 \vee \mathbf{X}\, Z$. We introduce the variable x such that for all n,

$$(M', \pi) \models_k^n x \Leftrightarrow (M', \pi) \models_k^n \mathbf{X}\, \tau^{k-n}(\mathsf{true})$$

By substituting the definition of τ once and the definition of x, we have $(M', \pi) \models_k^n x \Leftrightarrow (M', \pi) \models_k^n \mathbf{X}\, (f_1 \vee x)$ and by reference to the semantics, $(M', \pi) \models_k \mathbf{G_k}(x \Leftrightarrow \mathbf{X}(x \vee f_1))$

From the least fixpoint characterisation[6], $(M', \pi) \models_k x \Leftrightarrow \mathbf{F}\, f_1$, and by unrolling τ by one step and substituting the definition of x, we get $(M', \pi) \models_k f_1 \vee x$.

Theorem 2. *For any rule A, $\vdash_{\mathrm{QPTL}} A \Leftrightarrow \exists x.T_F(A)$*

[4] See [15] for full details; briefly, $(M, i) \models \exists p.A$ iff there exists an M' such that $(M', i) \models A$, and M' and M differ at most in the valuation given to p.

Proof. Proving each direction independently:

- $\vdash_{\mathrm{QPTL}} A \Rightarrow \exists x. T_F(A)$
  Substituting Lemma 1,

$$\begin{aligned}\mathbf{G_k}(P \Rightarrow \mathbf{F}\,B) &\Rightarrow \exists x.\mathbf{G_k}(x \Leftrightarrow \mathbf{X}(x \vee B)) \wedge \mathbf{G_k}(\mathbf{bound} \Rightarrow \neg x) \wedge \mathbf{G_k}(P \Rightarrow (x \vee B)) \\ &\Rightarrow \exists x.\mathbf{G_k}((x \Leftrightarrow \mathbf{X}(x \vee B)) \wedge \mathbf{bound} \Rightarrow \neg x \wedge (P \Rightarrow (x \vee B)))\end{aligned}$$

  which implies the set of rules $\{x \Rightarrow \mathbf{X}(x \vee B),\, \mathbf{bound} \Rightarrow \neg x,\, P \Rightarrow x \vee B\}$.
- $\vdash_{\mathrm{QPTL}} \exists x. T_F(f) \Rightarrow f$
  Starting with the transformed set of rules $\{x \Rightarrow \mathbf{X}(x \vee B),\, \mathbf{bound} \Rightarrow \neg x,\, P \Rightarrow x \vee B\}$, and exploiting the corollary of Lemma 1, $(M', s_i) \models_k (x \vee f_1) \Leftrightarrow (M', s_i) \models_k \mathbf{F}\, f_1$ iff $(M', s_i) \models \mathbf{G_k}(\mathbf{bound} \Rightarrow \neg x)$

$$\begin{aligned}&\mathbf{G_k}((x \Leftrightarrow \mathbf{X}(x \vee B)) \wedge \mathbf{bound} \Rightarrow \neg x \wedge (P \Rightarrow (x \vee B))) \\ \Leftrightarrow\ &\mathbf{G_k}(x \Leftrightarrow \mathbf{X}(x \vee B)) \wedge \mathbf{G_k}(\mathbf{bound} \Rightarrow \neg x) \wedge \mathbf{G_k}(P \Rightarrow (x \vee B)) \\ \Leftrightarrow\ &\mathbf{G_k}(x \Leftrightarrow \mathbf{X}\,\mathbf{F}\,B) \wedge \mathbf{G_k}(\mathbf{bound} \Rightarrow \neg x) \wedge \mathbf{G_k}(P \Rightarrow (x \vee B)) \\ \Rightarrow\ &\mathbf{G_k}((x \Rightarrow \mathbf{X}\,\mathbf{F}\,B) \wedge (P \Rightarrow (x \vee B))) \\ \Rightarrow\ &\mathbf{G_k}(P \Rightarrow ((\mathbf{X}\,\mathbf{F}\,B) \vee B)) \\ \Rightarrow\ &\mathbf{G_k}(P \Rightarrow \mathbf{F}\,B)\end{aligned}$$

  That is, the singleton rule set $\{P \Rightarrow \mathbf{F}\,B\}$.

4 Comparisons

We compare the encodings on an example specifcation $\mathbf{G}\,\mathbf{F}\,f$. This is a reachability specification, with many applications. Before encoding, the specification is negated to

$$\mathbf{F}\,\mathbf{G}\,\neg f \tag{2}$$

We consider only the loop encoding, as the non-loop encoding is $\perp$ for all methods due to the semantics of $\mathbf{G}$.

The original, recursive encoding decomposes in two steps.In the loop case,

$$\begin{aligned}{}_l[\![\mathbf{F}\,\mathbf{G}\,\neg f, \pi]\!]_k^0 &= \bigvee_{i=0}^{k} {}_l[\![\mathbf{G}\,\neg f, \pi]\!]_k^i \\ &= \bigvee_{i=0}^{k} \bigwedge_{j=\min(i,l)}^{k} f(j)\end{aligned}$$

This is a disjunction of conjunctions: the pathological case for conversion to clauses. It is possible to define an more efficient encoding using renamed subformulæ [3], but this approach is difficult to generalise. The size of the formula is $O(k^2)$, hence the cost to build it before CNF conversion is quadratic.

The conversion to SNF yields the following rules[5]

$$\begin{aligned}
\textbf{start} &\Rightarrow \mathbf{F}\, x_1 \\
x_1 &\Rightarrow \neg f \wedge x_2 \\
x_2 &\Rightarrow \mathbf{X_1}\, (\neg f \wedge x_2)
\end{aligned}$$

which encode to the three conjuncts

$$\begin{aligned}
\bigvee_{i=0}^{k} x_1(i) \quad &\wedge \\
\bigwedge_{i=0}^{k} (x_1(i) \Rightarrow \neg f(i) \wedge x_2(i)) \quad &\wedge \\
\bigwedge_{i=0}^{k} (x_2(i) \Rightarrow \neg f(i+1) \wedge x_2(i+1))&
\end{aligned}$$

We have two introduced variables: the first establishes a renaming of the $\mathbf{G}\,\neg f$ subformula, and the second renames each successive step of this subformula. This means that steps are shared between references from the $\mathbf{F}$ operator, leading to a simplification of the problem which is easier to solve as well as being smaller. The encoding corresponds to an ideal renaming of the formula above, but the conversion is performed in linear time, and results in a formula of size $O(k)$. Furthermore, we can show in advance that the encoding of each rule used here is invariant with l, which means that the subformulæ can be factorised out of the disjunction of loops seen in Equation 1.

Finally, we examine the fixpoint normal form conversion. The set of rules corresponding to the specification is

$$\begin{aligned}
\textbf{start} &\Rightarrow x_0 \vee x_1 \\
x_0 &\Rightarrow \mathbf{X}\, (x_0 \vee x_1) \\
\textbf{bound} &\Rightarrow x_1 \vee \neg x_0 \\
x_1 &\Rightarrow \neg f \wedge x_2 \\
x_2 &\Rightarrow \mathbf{X_1}\, (\neg f \wedge x_2)
\end{aligned}$$

[5] Further reduction of the second and third rules is necessary for correct SNF; we disregard this as it makes no difference to the final encoding

which encode to the conjuncts

$$\begin{aligned} x_0(0) \vee x_1(0) \quad &\wedge \\ \bigwedge_{i=0}^{k} (x_0(i) \Rightarrow x_0(i+1) \vee x_1(i+1)) \quad &\wedge \\ x_1(k) \vee \neg x_0(k) \quad &\wedge \\ \bigwedge_{i=0}^{k} (x_1(i) \Rightarrow \neg f(i) \wedge x_2(i)) \quad &\wedge \\ \bigwedge_{i=0}^{k} (x_2(i) \Rightarrow \neg f(i+1) \wedge x_2(i+1)) \end{aligned}$$

The main difference between the SNF encoding and the fixpoint normal form encoding is the omission of the long disjunction in the first conjunct which would be encoded as a single long clause. This is replaced by an array of conjunctions which rename each step in much the same way as for the **G** operator. Although in this case the advantage is dependent on the SAT checker, it is clear that where the **F** operator is nested, similar advantages would be seen as for SNF with the **G** operator.

5 Results

We compare the SNF and Fixpoint encodings with the encoding used in NuSMV version 2.0.2; this version of NuSMV includes several of the optimisations discussed in [3]. For consistency, we have implemented the SNF and Fixpoint encodings as options in NuSMV. All of the experiments have been done using the SAT solver zChaff [12] on a 700MHz Athlon with 256Mb main memory, running Linux.

5.1 Scalability

We observe the difference in the behaviour of the encodings as with increasing problem size by choosing a simple problem that is easy to scale. The benchmark circuits have been kept deliberately simple as it is the encoding of the specification not the model that differentiates the encodings.

A shift register is a storage device of length n which, at each time step, moves the values stored in each of its elements to the next element, reading in a new value to fill the now empty first element. That is, storage elements $x_0 \dots x_{n-1}$ and input in are transformed such that $\forall i, 0 < i < n \cdot (x_i \leftarrow x_{i-1})$ and $x_0 \leftarrow in$. The shift register can be representative of a much more complex step-based process such as a processor pipeline. The specification that the shift register must fulfil will depend on its application; we explore a number of response patterns taken from [4]. The specifications grow with the number of elements in the shift register; in the case of a three element register,

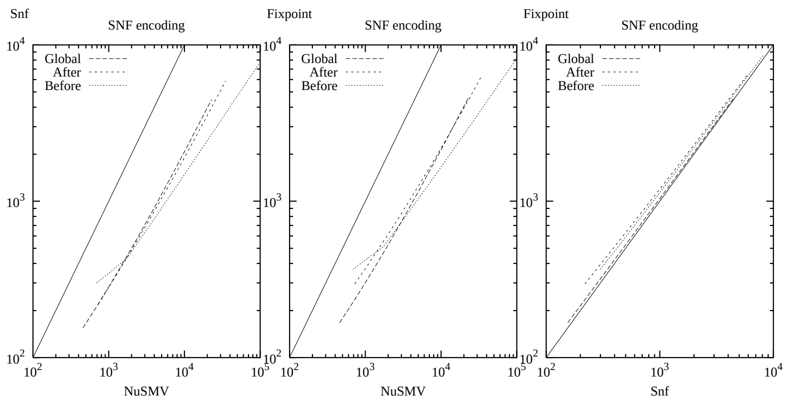

Fig. 2. Number of clauses generated by a shift register model

- Global response (depth 2) — x_2 goes high in response to in: $\mathbf{G}(in \Rightarrow \mathbf{F}\, x_2)$
- After response (depth 3)— x_2 goes high in response to in, after x_1 has gone high: $\mathbf{G}\,(x_1 \Rightarrow \mathbf{G}(in \Rightarrow \mathbf{F}\, x_2))$
- Before response (depth 3)— x_1 goes high in response to in, before x_2 has gone high (this property is only true if all the registers are zero, so we test for $empty \equiv \neg x_0 \wedge \neg x_1 \wedge \neg x_2$ too): $[((in \wedge empty) \Rightarrow [\neg x_2\, \mathbf{U}\,(x_1 \wedge \neg x_2)])\, \mathbf{U}\,(x_2 \vee \mathbf{G}\, x_2)]$

Number of Clauses. We see in Figure 2 that the number of clauses produced by both Snf and Fixpoint grows, in general, less quickly than the number produced by NuSMV, as the length of the register increases. The differing gradients follow the behaviour predicted by the differing depths of the specifications: the slopes decrease with depth indicating an exponential improvement in the number of clauses.

The advantage of the Fixpoint encoding over SNF is dependent upon the number of occurrences of $\mathbf{G}$ in the specification, since this is the only difference between the encodings. We see the greatest advantage for Fixpoint in the After and Before specifications, with two occurrences of $\mathbf{G}$; the first $\mathbf{G}$ in the After specification has a smaller encoding than the second as one of the corresponding rules is an initial rule.

We can conclude that, as far as the number of clauses is concerned, the Fixpoint encoding outperforms Snf and NuSMV in the way that is expected: size and rate of size increase decreasing with the nesting depth and the occurrence of least fixpoint operators.

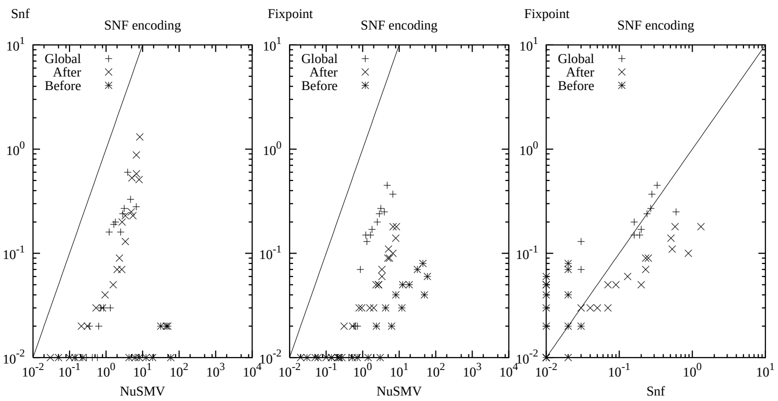

Fig. 3. Time taken by zChaff for a shift register model

zChaff timings. Counting the number of clauses is far from being an effective method of determining the efficiency of an encoding. We also look at one of the current state-of-the-art SAT solvers, zChaff [12].

The behaviour is far less clear than for the number of clauses; zChaff is a complex system. Broadly, the SNF and Fixpoint encodings always results in a shorter runtime than the NuSMV encoding; the Fixpoint encoding outperforms the SNF encoding only for After (for Global, the trend is towards an improvement with larger problems)

We see a clear exponential improvement for certain specifications: the timings for Before with SNF and Fixpoint grow exponentially slower than NuSMV; the Global specification shows the same trend less dramatically. We only see a exponential improvement for After with the Fixpoint encoding: with the SNF encoding, the trend appears to be towards NuSMV being faster.

5.2 Mutual Exclusion

The distributed mutual exclusion circuit from [10] forms a good basis for comparing the performance of different encodings as it meaningfully implements several specifications. We look at three here, applied to a DME of four elements:

- Accessibility: if an element wishes to enter the critical region, it eventually will. We check the accessibility of the first two elements. This specification is correct, so as in [1], we check at a chosen bound to illustrate the timing differences.
- Precedence given token possession: if an element of the DME holds the token, then its requests to enter the critical region are given precedence. We check the converse: if the first element holds the token, the second does not have

Table 3. Timing results in zChaff for the distributed mutual exclusion circuit

Specification	Bound	NuSMV encoding	SNF encoding	Fixpoint encoding	SMV
Accessibility	30	2.65	0.33	0.36	13.13
Accessibility	40	20.93	4.84	4.33	13.13
Priority for 0	14	0.13	0.02	0.02	12.97
Priority for 1	54	14.93	0.44	0.76	15.00
Overtaking depth 1	40	85.73	2.15	1.11	13.96
Overtaking depth 2	40	*	4.92	5.15	14.14

precendence and *vice versa*. Since the token begins at the first element, this is the quicker to prove, with a bound of 14. For the second element, a bound of 54 is required to find the counterexample.

- Bounded overtaking given token posession: if two elements wish to enter the critical region, then the higer priority may enter a given number of times before the other. We check bounded overtaking of one and two entrances. Both specifications are correct so as above we check at a bound of 40. These specifcations are the most complex, including up to four nested *until* operators.

The results are summarised in Table 3 together with the timings for Cadence SMV on CTL representations of the same problems[6]. For the bounded overtaking problems, we note that NuSMV took nearly 10 minutes to generate the formula in the first case, and after 25 minutes had not completed in the second case. In contrast, the time taken to perform the SNF and Fixpoint encodings were insignificant.

While both the SNF and Fixpoint encodings outperform the the NuSMV encoding and the SMV, we do not see a consistent advantage to either. The results for accessibility suggest that Fixpoint scales better with increasing bound, while the results for boudned overtaking suggest that SNF scales better with increasing specification depth.

6 Conclusions

We have described a new encoding scheme for bounded model checking which builds on the existing encodings and uses the fixpoint characterisations of LTL. We have shown that these new encodings are correct, provided that the original bounded model checking encoding is correct. We have demonstrated a reduction in the number of clauses generated by the problem which is exponential in the size of the problem instance, for both encodings, and also that the improvement in performance in the SAT checker can be exponential in the size of the problem instance, depending on the specification. We have demonstrated the advantage

[6] We note that for SMV to terminate in a reasonable time on these problems, it must be started with the `-inc` switch. No similar knowledge of model checker behaviour is needed for BMC

that these encodings give BMC over conventional symbolic model checkers. Finally, we have demonstrated that extending the SNF transformations with a transformation for the **F** operator results in similar advantages over SNF in certain cases.

References

1. Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In W.R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems. 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag Inc., July 1999.
2. Alexander Bolotov and Michael Fisher. A resolution method for CTL branching-time temporal logic. In *Proceedings of the Fourth International Workshop on Temporal Representation and Reasoning (TIME)*. IEEE Press, 1997.
3. Alessandro Cimatti, Marco Pistore, Marco Roveri, and Roberto Sebastiani. Improving the encoding of LTL model checking into SAT. In Agostino Cortesi, editor, *Third International Workshop on Verification, Model Checking and Abstract Interpretation*, volume 2294 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., January 2002.
4. M.B. Dwyer, G.S. Avrunin, and J.C. Corbett. Property Specification Patterns for Finite-State Verification. In M. Ardis, editor, *2nd Workshop on Formal Methods in Software Practice*, pages 7–15, March 1998.
5. M.B. Dwyer, G.S. Avruning, and J.C. Corbett. Patterns in property specifications for finite-state verification. In *21st International Conference on Software Engineering, Los Angeles, California*, May 1999.
6. E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In Jan van Leeuwen J. W. de Bakker, editor, *Automata, Languages and Programming, 7th Colloquium*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer-Verlag Inc, 1980.
7. Michael Fisher. A resolution method for temporal logic. In *Proceedings of Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, August 1991.
8. Michael Fisher and Philippe Noël. Transformation and synthesis in MetateM Part I: Propositional MetateM. Technical Report UMCS-92-2-1, Department of Computer Science, University of Manchester, Manchester M13 9PL, England, February 1992.
9. Dov Gabbay. The declarative past and imperative future. In H. Barringer, editor, *Proccedings of the Colloquium on Temporal Logic and Specifications*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer-Verlag, 1989.
10. A. J. Martin. The design of a self-timed circuit for distributed mutual exclusion. In Henry Fuchs, editor, *Proceedings of the 1985 Chapel Hill Conference on VLSI*, pages 245–260. Computer Science Press, 1985.
11. K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, 1992.
12. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference*, Las Vegas, June 2001.

13. Daniel Sheridan and Toby Walsh. Clause forms generated by bounded model checking. In Andrei Voronkov, editor, *Eighth Workshop on Automated Reasoning*, 2001.
14. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
15. Pierre Wolper. Specification and synthesis of communicating processes using an extended temporal logic. In *Proceeding of the 9th Symposium on Principles of Programming Languages*, pages 20–33, Albuquerque, January 1982.