

July 2003

SAT Encodings for Model Checking

Thesis Proposal

Daniel Sheridan

Supervisors: Dr Paul Jackson and Dr Kusha Etessami

Laboratory for the Foundations of Computer Science
Division of Informatics
University of Edinburgh

Contents

1	Background	2
1.1	Hypothesis	2
1.2	Rationale	2
1.2.1	Bounded Model Checking	2
1.2.2	Limitations of Symbolic Model Checking	3
1.2.3	SAT technology	3
1.2.4	Temporal Logic Normal Forms	4
1.3	Related Work	4
1.3.1	Existing Bounded Model Checking Literature	4
1.3.2	Non-Bounded SAT-based Model Checking	4
1.3.3	Existing Bounded Model Checking Implementations	5
1.4	Background	5
1.4.1	Model Checking	5
1.4.2	Path Loops	5
1.4.3	Boolean Satisfiability	5
1.4.4	The Bounded Model Checking Encoding	5
1.4.5	Reduced Boolean Circuits	7
1.4.6	The Separated Normal Form	8
2	Progress	10
2.1	Improving Specification Encoding	10
2.1.1	Special Case Encodings	10
2.1.2	Temporal Logic Normal Forms	10
2.1.3	Past Time Logics	13
2.2	General Encoding Improvements	14
2.2.1	Clause Form Conversion	14
2.2.2	Dynamic Step Size Adjustment in Iterative Deepening Search	16
3	Plans	18
3.1	SNF in Context	18
3.1.1	SNF as a renaming strategy	18
3.1.2	SNF as automata	18
3.2	Further work on SNF	19
3.3	General Encoding Improvements	20
3.3.1	Problem Invariants	20
3.4	Integer Encodings	20
3.5	Evaluation	20
3.6	Outline	21

3.7 Proposed Timetable	21
Bibliography	23

Chapter 1

Background

1.1 Hypothesis

The bounded model checking method can be improved in both the time taken to encode and the time taken to verify by considering methods from the field of temporal logic resolution, as well as by considering its interaction with the SAT solver

1.2 Rationale

Bounded model checking (BMC) was developed under influence from a number of areas. It may be considered a variant of symbolic model checking, and it was intended to overcome some of the problems associated with conventional symbolic model checkers. As computers have grown in capacity and speed, algorithms for solving Boolean satisfiability (SAT) have become more viable — their flexibility was demonstrated with encodings of planning. SAT had already been applied to hardware problems such as equivalence checking, and finding a way of applying the technology to the model checking problem was a practical next step. The logic naturally dealt with in BMC is linear temporal logic (LTL), while symbolic model checking deals with computational tree logic (CTL); the development of BMC coincided with a renewed interest in LTL for specifying properties of systems.

Bounded model checking has been developed independently from other LTL model checking methods, and is very different in style from the fixpoint based methods of symbolic model checking for LTL. This leads us towards a possible method of improving BMC — by lifting methods developed for symbolic model checking of CTL and LTL to the bounded context. In particular, the algorithms of symbolic model checking for CTL makes use of the fixpoint characterisation of the temporal operators to break down the problem; symbolic model checking for LTL is achieved by an encoding of the problem to an automaton representation.

1.2.1 Bounded Model Checking

Bounded model checking (BMC) [6] was proposed as a solution to some of the problems of conventional BDD-based symbolic model checking such as space explosion by introducing a temporal bound. The problem can then be encoded as a Boolean formula; the output of a BMC tool is a conjunction of state transition functions and state verification functions. In order to retain the ability to model check infinite temporal properties, BMC considers *loop paths*: paths of a given length, k , in which there is a point l such that $s_l = s_k$. This is sufficient to show, for example, that there is a path along which an eventuality never occurs.

1.2.2 Limitations of Symbolic Model Checking

While BDDs are often a very efficient method for the representation and manipulation of a function, some functions cannot be compactly represented in this way. One particular example, integer multiplication, was recognised by Bryant [9]: he calculates a lower bound of $2^{n/8}$ vertices for an n -bit multiplier. Empirically, he found that an 8-bit multiplier required no more than 5000 vertices, and for $n > 10$, over 100 000 vertices are required. Model checking such circuits is generally impractical because of these huge storage requirements.

The worst case space complexity of BDDs is exponential in the number of variables; this means that the worst case space complexity of symbolic model checkers is also exponential. DLL satisfiability procedures are linear space in the number of clauses; a transition function may be converted to clause form with only a polynomial increase in space [38]. Provided that the specification may be encoded in polynomial space, the worst case space complexity of BMC is only polynomial in the size of the problem (although it may be as bad as exponential space in the size of the bound).

1.2.3 SAT technology

Approaches to solving the Boolean satisfiability problem fall broadly into two classes: incomplete algorithms, such as hill-climbing and simulated annealing, and complete algorithms, characterised by variants of the Davis Putnam algorithm [14, 15]. We will focus on the latter. DP, and its refinement, DPLL, is essentially a search algorithm which traverses the space of variable assignments for a propositional formula. By representing the formula in clause form (that is, conjunctive normal form or CNF) the result of each assignment can be quickly determined: assigning a literal to true eliminates from the problem any clauses with a positive occurrence of that literal, and eliminates any negative occurrences of the literal in any clause. An empty clause implies a contradiction and causes the search to backtrack, while if all clauses are eliminated, the problem is solved.

A number of techniques make this simple search procedure surprisingly efficient. Modern solvers such as Chaff [34] are focussed particularly on the type of structured problem produced by bounded model checkers.

Unit propagation If a clause contains only one literal, it can be satisfied only by assigning it to true.

Conflict learning If the solver explores a branch of the search tree which it finds contains no solutions, it can add a clause to the problem to describe the branch, and so force the avoidance of this path later in the search space.

Non-chronological backtracking

Conflict learning with scheduled lazy deletion Learned conflict clauses may be deleted from the problem if its relevance to the current path is insufficient; when more than a given fraction of its literals are unassigned for the first time it is removed.

Restarts Learning conflict clauses during search means that a large amount of information about the problem is stored. This information was not available when the early decisions in the search tree were made. To give the solver a chance to correct its early mistakes, the solver periodically restarts the search procedure, retaining only the learnt clauses. A degree of randomness introduced into the first few iterations of the decision process helps to avoid following the same paths repeatedly.

Two literal watching If, during solving, all but one of the literals in a clause become false, the remaining literal is forced to one. To avoid an expensive scan of all of the clauses, the solver watches just two literals from the clause. When one of these literals becomes false, the solver will check the rest of the clause, changing the watched literal if unassigned literals remain. This change of observation has a number of side effects during backtracking: the watched literals do not need to change during backtracking, and another assignment to a previously assigned and unassigned variable will be faster as any watching will already have moved off that variable.

1.2.4 Temporal Logic Normal Forms

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 [17] lead to a blowup in the size of the resulting Boolean formula. Recent improvements to the encoding in NuSMV [11] have not removed this restriction.

The fixpoint characterisations of temporal operators [18] have been exploited in other model checking systems such as SMV [32]; 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.

The *Separated Normal Form* [20] originated in the field of temporal resolution. Any temporal operator may be represented as either the least or greatest fixpoint of a next-state function; SNF is based on the least fixpoint characterisations of temporal logic operators. The normal form is a set of rules each consisting of one temporal operator, which are required to hold in all states.

1.3 Related Work

Bounded model checking has inspired a number of researchers to extend and improve it. As well as improving the encoding, researchers have examined the possibility of improving SAT solvers to be more focussed on BMC, and on extending BMC to other temporal logics and other model checking problems. It is also important to be aware of other SAT-based model checking methodologies.

1.3.1 Existing Bounded Model Checking Literature

Cooty et al. [13] discuss the use of BMC at Intel on large, real-world designs (fragments of the Pentium IV Processor) and compare an implementation of BMC using a SAT procedure with one using BDD-based satisfiability testing to give a clearer picture of the relative advantage of SAT. They find that in most cases the SAT-based verifier out-performs even a hand-tuned run of the BDD-based verifier. We note, however, as these results are difficult to repeat as both the programs and the circuits are internal to Intel.

Bounded model checking has been shown to be effective for timed systems [2, 45], by extending the domain of the SAT solver to handle integer inequalities [3, 46, 47].

Extensions to apply BMC to other temporal logics have been studied by Penczek et al., who have looked at the universal fragment of CTL [37], TCTL (in the timed automata domain) [36], and most recently ACTL* [50]. Benedetti et al. considered the extension of LTL to the past¹ by unrolling the transition relation an extra number of times dependant on the depth of past to be considered [5].

Recently, bounded model checking has been extended to a simplified concurrent programming language [27].

1.3.2 Non-Bounded SAT-based Model Checking

While bounded model checking is the dominant SAT-based technique, others have emerged, aiming to remove the limitation that the bound introduces. Inductive techniques for checking invariants were studied by Biere et al. [7] and are fast but, since it considers all adjacent pairs of states, it can be confused by unreachable states in the model which violate the invariant. The technique is improved by Sheeran et al. [43] by extending the length of the trace considered until a connection can be made to an initial state. This method is complete.

Another alternative was proposed by McMillan [33] of using SAT in place of BDDs in a conventional symbolic model checking algorithm. This method uses an adapted SAT solver to compute equivalent formulae.

¹While this does not increase the theoretical expressiveness of the logic, it can make certain specifications more succinct and easily understood

1.3.3 Existing Bounded Model Checking Implementations

The first freely available full implementation of BMC is in version 2.0 of NuSMV [10, 11]. This has been the development platform for novel improvements such as the use of Reduced Boolean Circuits [26] as an intermediate representation for the propositional formulæ. In addition their analysis of the way in which RBCs behave allows them to tailor the encoding process to maximise the amount of sharing within the RBC.

Since version 2.0, VIS [39] has included a simple implementation of BMC which is based directly on Biere et al. [6]. The key difference with this implementation is that clauses are generated directly by the encoding procedure, rather than using an intermediate representation like NuSMV does. This limits the scope for analysis and simplification of the resulting formulæ (the implementation has a hardcoded type of polarity handling — see Section 2.2.1), but guarantees a low overhead implementation of the basic encoding.

Cadence SMV, until recently, used the research implementation `bmc` written by Biere et al. which was restricted to single operator specifications.

1.4 Background

We look at the theoretical basis for the work in Chapter 2.

1.4.1 Model Checking

A model checking problem is a pair $\langle M, f \rangle$ of a model and a temporal logic specification.

A model M is defined as a Kripke structure $\langle S, R, L, I \rangle$ where S is a set of states; $R : S \rightarrow S$ is the transition relation; $L : S \rightarrow \mathcal{P}(AP)$ is the labelling function, marking each state with the set of atomic propositions (AP) that hold in that state; and I is the set of initial states, which may be equal to S . A path $\pi \in M$ is a sequence of states $s_0, s_1, \dots \in M$ such that $\forall i. (s_i, s_{i+1}) \in R$. We write $\pi(i)$ to refer to the i th state along the path.

The *model checking problem for LTL* is to verify that for an LTL formula f , for all paths $\pi_i \in M$ such that $\pi_i(0) \in I$, $(M, \pi_i) \models f$.

1.4.2 Path Loops

We say the a path π is a *k-loop* if for all $i \geq 0$, the $(k + i)$ th state in π is identical to the i th state for some $l, 0 \leq l < k$. If a path is known to be a loop, it is possible to verify the correctness of infinite time specifications such as *always* (\mathbf{G}) by checking just the first k states in the path.

1.4.3 Boolean Satisfiability

Boolean satisfiability (SAT) is the problem of assigning Boolean values to variables in a propositional formula, in such a way as to make the formula evaluate to true (to *satisfy* the formula). For example, for the formula $(a \vee \neg b) \wedge (b \vee \neg c) \wedge (\neg c \vee \neg a)$ can be satisfied by *e.g.* the assignment $a = 1, b = 1, c = 0$.

SAT solvers derived from the Davis-Putnam algorithm [15] require input in clause form (CNF): a conjunction of *clauses*, each of which is a disjunction of literals.

A number of high performance SAT solvers are available, making SAT a convenient ‘black box’ back end for a number of different problems.

1.4.4 The Bounded Model Checking Encoding

The bounded model checking encoding represents k states along a bounded path π_{bmc} together with 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 1.1)

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

Figure 1.1: The Bounded Semantics of LTL

Table 1.1: The BMC Encoding for LTL

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

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.

We write the bounded model checking encoding of a problem with bound k , model M and specification f as

$$\llbracket M, \neg f \rrbracket_k$$

Given the functions $lL_k(\pi)$ which holds when π is a k -loop with $\pi(k) = \pi(l)$ and $L_i(\pi) = \bigvee_{l=0}^k lL_k$ which holds when π is any k -loop, the general translation is defined as²:

$$\llbracket M, f \rrbracket_k := \llbracket M \rrbracket_k \wedge \left(\neg L_k(\pi) \wedge \llbracket f \rrbracket_k^0 \right) \vee \bigvee_{l=0}^k \left(lL_k(\pi) \wedge l\llbracket f \rrbracket_k^0 \right) \quad (1.1)$$

where $\llbracket M \rrbracket_k$ denotes the encoding of the transition relation of M as a constraint on π with bound k ; $\llbracket f \rrbracket_i^k$ and $l\llbracket f \rrbracket_i^k$ 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 1.1. Biere et al. show the correctness of some of these encodings in [6]; we will not repeat their proofs here.

Theorem 1 in Biere et al. [6] states that bounded model checking of this form is complete provided that the bound k is sufficiently large.

1.4.5 Reduced Boolean Circuits

Since BMC is built around propositional formulæ, an efficient data structure is required to store and manipulate them. The implementation of BMC in NuSMV is based around Reduced Boolean Circuits (RBCs) [1].

A *Boolean Circuit* is a representation of propositional formulæ as a DAG. A Reduced Boolean Circuit uses a normal form to maximise subformula sharing. The vertices of an BC consist of the internal nodes $\mathbf{V_I}$ and the leaves $\mathbf{V_L}$. The following properties hold:

- Each $v \in \mathbf{V_I}$ consists of an operator $op(v) \in \{\wedge, \leftrightarrow\}$ and a left and right edge ($left(v), right(v) \in \mathbf{E}$).
- Each $v \in \mathbf{V_L}$ contains a variable $var(v)$.
- Each $e \in \mathbf{E}$ has a sign $sign(e)$ and a target vertex $target(e) \in \mathbf{V}$.

Negation is encoded into the edges by the *sign* attribute. An edge in the BC represents a subformula; it is simple to extract a formula from a BC by reading it as a parse tree.

An RBC has the following additional properties which serve to reduce the number of representations possible for equivalent formulæ:

²This comes from Definition 15 in [6]

- All common subformulæ are shared.
- The constant \top only occurs in single-vertex RBCs.
- For all vertices, $left(v) \neq right(v)$.
- If $op(v) = \leftrightarrow$ then $left(v)$ and $right(v)$ are unsigned.
- There is a total order \sqsubset on BCs such that for all vertices $left(v) \sqsubset right(v)$.

1.4.6 The Separated Normal Form

Gabbay's Separation Theorem [23] states that arbitrary temporal formulæ may be written in the form $\mathbf{G}(\bigwedge_i (P_i \Rightarrow F_i))$ where P_i are (strict) past time formulæ and F_i are (non-strict) future time formulæ.

Fisher [20] 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 [8] define the **start** operator which holds only at the beginning of time.

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

The possible rules are thus

$$\begin{array}{ll} \mathbf{start} \Rightarrow \bigvee_j l_j & \text{An initial rule} \\ \bigwedge_i l_i \Rightarrow \mathbf{X} \bigvee_j l_j & \text{A global } \mathbf{X}\text{-rule} \\ \bigwedge_i l_i \Rightarrow \mathbf{F} \bigvee_j l_j & \text{A global } \mathbf{F}\text{-rule} \end{array}$$

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 $\{\mathbf{start} \Rightarrow f\}$. For brevity, we do not list the full set of transformations here; in general they are trivially adapted from those in [8], or from standard propositional logic.

$$\begin{aligned} T_G(\{P \Rightarrow \mathbf{G} f\} \cup \Psi) &= \left\{ \begin{array}{l} P \Rightarrow f \wedge x \\ x \Rightarrow \mathbf{X}(f \wedge x) \end{array} \right\} \cup T_G(\Psi) \\ T_U(\{P \Rightarrow f \mathbf{U} g\} \cup \Psi) &= \left\{ \begin{array}{l} P \Rightarrow g \vee (f \wedge x) \\ x \Rightarrow \mathbf{X}(g \vee (f \wedge x)) \\ P \Rightarrow \mathbf{F} g \end{array} \right\} \cup T_U(\Psi) \\ T_{\text{ren1}}(\{P \Rightarrow \mathbf{G} f(\mathbf{F} g)\} \cup \Psi) &= \left\{ \begin{array}{l} P \Rightarrow \mathbf{G} f(x) \\ x \Rightarrow \mathbf{F} g \end{array} \right\} \cup T_{\text{ren1}}(\Psi) \end{aligned}$$

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 [18]; 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* (\mathbf{G}) and *weak until* (\mathbf{W}); *until* (\mathbf{U}) is first converted to *weak until* and *sometime* for its transformation) which means that the *sometime* operator remains unchanged.

By Tarski's fixpoint theorem [48] 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 [21].

Chapter 2

Progress

Progress towards the thesis hypothesis has so far concentrated on the encoding of the temporal logic part of the model checking problem. We initially considered optimisations available to special cases of the specification (specifically, single operator specifications), analysing the number of clauses expected from the encoding and exploiting subformula renaming to improve the encoding.

By converting the specification to a normal form such as SNF, we are able to apply renaming more generally, directly to the temporal logic expression. In addition, the conversion to SNF reduces the variety of possible temporal logic expressions, meaning that the encoding to propositional logic can be more focused. We have modified SNF to apply transformations to more temporal operators, and proven the correctness of these new transformations.

Our implementation of the SNF encoding, written as a modification of a publicly available model checker, has been used to demonstrate the advantage of the SNF-based encodings.

2.1 Improving Specification Encoding

We look at how bounded model checking can be improved by focussing on the encoding of the specification, rather than that of the model.

2.1.1 Special Case Encodings

The first attempt that we made at improving the encoding of the specification was to consider particular ways of renaming subformulae. The work is based on that of Nonnengart et al. [35] on choosing renamings based on computing the number of clauses expected from a naïve clause form conversion of a formula¹. We computed the expected number of clauses produced by the bounded model checking encoding as a function of k and used the resulting formula to motivate particular renamings.

While this approach can give good results, it is labour-intensive and is subsumed by more intelligent CNF conversion such as that considered in Section 2.2.1.

2.1.2 Temporal Logic Normal Forms

We demonstrate that an encoding based on the conversion of the temporal logic specification to SNF can overcome the limitations of the recursive nature of the encoding. We also give a simplification of SNF which exploits the greatest fixpoint characterisations in addition to the least fixpoint characterisations, and which reduces rules to relating successive pairs of states, but depends explicitly on the bounded nature of BMC.

¹The clause form generated by the recursive application of the substitution $a \vee (b \wedge c) \rightarrow (a \vee b) \wedge (a \vee c)$.

SNF for BMC

Each rule resulting from the standard SNF transformation is encoded at each state that we wish to consider for bounded model checking. In the non-loop case, this means that the SNF formula is satisfiable only if enough states are considered for the fixpoint characterisations to reach stability; similarly in the loop case, the formula is satisfiable only if the stability is reached within the loop. The result is that the simple encoding below is satisfiable iff a witness exists before the bound.

For a set of rules Ψ , the encoding up to step k is performed by first taking the \Rightarrow operator to be implication and converting the rules to NNF. The specification is then encoded as (non-loop case to the left, loop case to the right):

$$\bigwedge_{i=0..k} \bigwedge_{\psi \in \Psi} \neg \llbracket \psi \rrbracket_i^k \quad \vee \quad \bigvee_{l=0..k} \bigwedge_{i=0..k} \bigwedge_{\psi \in \Psi} \llbracket \psi \rrbracket_i^k$$

where the encoding operator $\llbracket \dots \rrbracket$ is defined as below. Writing a for atomic propositions, f and g for arbitrary PLTL formulae, and $l \in \mathbb{N} \cup \{-\}$ denoting the loop-back point (for $l \in \mathbb{N}$) or the absence of a loop (for $l = -$),

$$\begin{aligned} \llbracket a \rrbracket_k^i &\doteq a(i) \\ \llbracket \neg a \rrbracket_k^i &\doteq \neg a(i) \\ \llbracket f \vee g \rrbracket_k^i &\doteq \llbracket f \rrbracket_k^i \vee \llbracket g \rrbracket_k^i \\ \llbracket f \wedge g \rrbracket_k^i &\doteq \llbracket f \rrbracket_k^i \wedge \llbracket g \rrbracket_k^i \\ \llbracket \text{start} \rrbracket_k^i &\doteq \begin{cases} \top & \text{if } i = 0 \\ \perp & \text{otherwise} \end{cases} \\ \llbracket \mathbf{X} f \rrbracket_k^i &\doteq \begin{cases} \llbracket f \rrbracket_k^{i+1} & \text{if } i < k \\ \llbracket f \rrbracket_k^l & \text{if } i = k \text{ and } l \in \mathbb{N} \\ \perp & \text{otherwise} \end{cases} \\ \llbracket \mathbf{F} f \rrbracket_k^i &\doteq \begin{cases} \bigvee_{j=\min(i,l)}^k \llbracket f \rrbracket_k^j & \text{if } l \in \mathbb{N} \\ \bigvee_{j=i}^k \llbracket f \rrbracket_k^j & \text{otherwise} \end{cases} \end{aligned}$$

Since nesting of temporal operators is dealt with during the transformation to SNF, the encoding above does not suffer from the recursive nature of the encoding in [6]; a temporal operator cannot appear in the argument of another temporal operator, so the result of the encoding is much close to propositional clause form.

The Fixpoint Form

Since SNF converts only the greatest fixpoint operators, it leaves rules containing the \mathbf{F} operator unchanged; unfortunately these rules are the pathological case for this encoding. Since \mathbf{F} is kept to preserve the semantics of the other least fixpoint operators after their removal, its transformation must preserve this least fixpoint behaviour. This is achieved for bounded model checking by restricting the evaluation of the resulting rules to the represented states, by introducing an operator which holds only in the final represented state: **bound**.

$$T_{\mathbf{F}}(\{P \Rightarrow \mathbf{F} f\} \cup \Psi) \doteq \left\{ \begin{array}{l} P \Rightarrow f \vee x \\ x \Rightarrow \mathbf{X}(f \vee x) \\ \mathbf{bound} \Rightarrow f \vee \neg x \end{array} \right\} \cup \Psi$$

The correctness of this transformation for bounded model checking is shown in [22].

The encoding for the Fixpoint Form is the same as for SNF, with the additional encoding for **bound** below. The encoding for \mathbf{F} is no longer needed, as all occurrences are replaced using the transformation above.

Table 2.1: Timing results in zChaff for the MSI cache coherence protocol

Processors	Specification	Bound	NuSMV	SNF	Fixpoint
2	Request A	10	4.40	1.73	1.53
2	Request A	20	19.40	5.82	9.97
2	Request B	10	2.65	3.63	2.69
2	Request B	20	49.78	8.63	16.42
3	Request A	10	13.00	3.03	2.50
3	Request A	20	39.22	8.2	5.79
3	Request B	10	4.60	6.66	5.93
3	Request B	20	54.94	62.11	40.25
3	Request C	10	4.58	6.64	5.91
3	Request C	20	44.8	50.27	37.65

$$\llbracket \mathbf{bound} \rrbracket_k^i \doteq \begin{cases} \top & \text{if } i = k \\ \perp & \text{otherwise} \end{cases}$$

Texas-97 Benchmarks

We examine a number of model checking benchmarks from the *Texas-97* benchmark suite [4]. These benchmarks have been converted from the Blif-mv representation to SMV format by a locally modified version of the VIS model checker [39]. We run these benchmarks at fixed bounds and report the time spent by zChaff.

MSI Cache Coherence Protocol

This is an implementation of a Modified Shared Invalid cache coherence protocol between two or three processors. We examine two of the specifications of behaviour from the benchmark. The results are summarised in Table 2.1.

- Whenever processor A requests the bus, it gets the bus in the next clock cycle. Listed as “Request A” in the results. $\mathbf{G}(bus_reqA \rightarrow \mathbf{X} bus_ackA)$
- Whenever processor B (or C) requests the bus, it gets the bus only when Processor A did not request the bus. Listed as “Request B” or “Request C” in the results. $\mathbf{G}(bus_reqB \rightarrow \mathbf{F} bus_ackB)$

Instruction Fetch Control Module

This is a model of the instruction fetch control module of the experimental TORCH microprocessor developed at Stanford University. Three models are examined; from the text accompanying the benchmark set:

- IFetchControl1: The original instruction module with several assumptions on the environmental signal.
- IFetchControl2: As IFetchControl1 except that the memory stall signal is always low.
- IFetchControl3.v: As IFetchControl1 except that the instruction cache line is assumed to be always valid.

We examine three specifications from the benchmark. The results are summarised in Table 2.2.

- The delayed version of a signal should, in the next state, have the signal’s previous value. Listed as “Delay” in the results. $\mathbf{G}(IStall_s1 \rightarrow \mathbf{X} IStall_s2)$

Table 2.2: Timing results in zChaff for the Instruction Fetch Control Module

Model	Specification	Bound	NuSMV	SNF	Fixpoint
IFetchControl1	Delay	10	0.94	0.45	0.44
IFetchControl2	Delay	10	0.99	0.40	0.40
IFetchControl3	Delay	10	1.29	0.39	0.50
IFetchControl1	Refetch	10	3.69	0.91	0.82
IFetchControl2	Refetch	10	3.30	0.89	0.81
IFetchControl3	Refetch	10	3.74	1.49	1.88
IFetchControl1	WriteCache	10	3.58	1.68	2.47
IFetchControl2	WriteCache	10	2.67	1.65	1.78
IFetchControl3	WriteCache	10	2.78	2.24	1.40

- As above, for the Refetch state. Listed as “Refetch” in the results.

$$\mathbf{G}((\text{PresState}_{s1} = \text{REFETCH}) \rightarrow \mathbf{X}((\text{PrevState}_{s2} = \text{REFETCH})))$$

- *WriteCache*_{s2} becomes one in some paths before *WriteTag*_{s2} becomes one. Listed as “WriteCache” in the results.

$$\neg[\neg \text{WriteTag}_{s2} \mathbf{U} (\text{WriteCache}_{s2} \wedge \neg \text{WriteTag}_{s2})]$$

2.1.3 Past Time Logics

The past LTL construction in NuSMV [5] considers multiple unrollings of the loop to establish the past temporal horizon. This means that the number of states over which the specification must be computed can be much greater than k . We argued in [22] that the time taken to construct the BMC encoding of an LTL formula grows polynomially with the number of states considered, and that the order of the polynomial corresponds to the number of nested temporal operators. The potential advantage gained by using SNF is greatly increased by the large number of states that are considered in past LTL problems.

The extension of the SNF encoding to past is relatively simple as SNF was originally defined over a past-time logic. We add transformations to handle nestings of future and past operators, separating past onto the left hand side of rules, and future onto the right. As the past is considered finite, least fixpoint operators do not present a problem. The result of the encoding includes only stepwise past operators (\mathbf{Z} and \mathbf{Y}) which are readily converted to \mathbf{X} . This means that the encoding given above can be used directly.

In order to stress the ability of the two methods to process past operators and to find short counterexamples, we conceived the Counter(N) problem set: a counter starts at 0, progresses up to N , and then loops back at $N/2$. We evaluate properties of the form $P(i)$, of the form

$$\neg \mathbf{F}(\mathbf{O}((c = N/2) \wedge \mathbf{O}((c = N/2 + 1) \dots \wedge \mathbf{O}(c = N/2 + i) \dots)))$$

The value of i is a measure of the nesting of past operators, while the structure of the property requires that the loop (of length $N/2$) must be traversed backwards several times in order to reach a counterexample.

The results are reported in Table 2.1.3². The direct encoding suffers from the nesting of the property, which influences the past temporal horizon and therefore requires a larger number of virtual unrolls. Most of the time is in fact spent in the generation of the encodings. On the contrary, the encodings are generated efficiently by the SNF-based method, and the time required by the SAT solver is also very limited. SNF-based encodings seem to yield a significant speed up, even if longer paths need to be explored in order to find a counterexample. Notice however that in this problem set the component related to the model is not very significant. Although the ability

²Run times on a Pentium 3, 700MHz, with 512Mb RAM.

Table 2.3: The results for Counter(N). T.O. indicates a runtime exceeding 3600 secs.

N	Counter(16)		Counter(32)		Counter(64)	
	BMC	SNF	BMC	SNF	BMC	SNF
$P(0)$	0.17 8	0.08 8	1.85 16	0.31 16	43.27 32	2.22 32
$P(1)$	16.35 17	0.35 17	1040.08 33	2.44 33	T.O. 34	26.91 65
$P(2)$	324.58 17	1.25 26	T.O. 22	10.76 50		141.12 98
$P(3)$	T.O. 13	3.48 35		33.94 67		550.41 131
$P(4)$		8.32 44		91.08 84		1688.11 164

to construct counterexamples with virtual unroll of the past might be a win, there is clearly a tradeoff between the time that is saved in searching shortened counterexamples compared to the time that is invested in generating more complex encodings.

2.2 General Encoding Improvements

We have considered two particular methods of improving the performance of BMC without a deep analysis of the encoding, but rather by aiming to generate better input for the SAT solve and by reducing the number of times the SAT solver is called.

2.2.1 Clause Form Conversion

One major difference in the implementation of NuSMV as compared to the old BMC program is the use of Reduced Boolean Circuits [1] to store the Boolean formula as it is being constructed. The CNF conversion used in NuSMV is very simple and does not take into account certain details of the structure preserving CNF conversion [38] such as the dependency of the conversion on the polarity of the subformula and not renaming disjunctive subformulae. A formula $a \wedge b$ is converted to the clauses $\{\neg x, a\}, \{\neg x, b\}, \{x, \neg a, \neg b\}$.

As an example, consider a specification of the form $\mathbf{X}(a_0 \wedge \mathbf{X}(a_1 \wedge \dots \mathbf{X}a_n))$. The SNF conversion produces a large number of implications of the form $x_0(i) \rightarrow a(i+1) \wedge x_1(i+1)$ as a result of renaming the nested \mathbf{X} operators. These implications are represented in the RBC as the formula $\neg(\neg x_0(i) \wedge \neg(a(i+1) \wedge x_1(i+1)))$, which is encoded as the clauses

$$\begin{array}{ll}
 \{\neg x_2, a(i+1)\} & \{\neg x_3, \neg x_0(i)\} \\
 \{\neg x_2, x_1(i+1)\} & \{\neg x_3, \neg x_1\} \\
 \{x_2, \neg a(i+1), x_1(i+1)\} & \{x_3, x_0(i), x_1\} \\
 \{\neg x_3\} &
 \end{array}$$

where a more direct conversion would have yielded two clauses and no extra variables:

$$\{\neg x_0(i), a(i+1)\} \qquad \{\neg x_0(i), x_1(i+1)\}$$

We will represent transformations on graphs as a rewriting of a graph together with a list of saved clauses:

$$a \circ b \longrightarrow (c), (\{d, e, f\}, \{g, h, i\})$$

denotes the replacement of the subgraph representing $a \circ b$ with c , and producing the clauses $\{d, e, f\}$ and $\{g, h, i\}$. Transformations are applied in depth-first order, starting at the leaves of the graph.

In the following discussion, sets of clauses are denoted by capital letters and may be combined with set operations; the cross product of two sets of clauses $A \times B$ denotes the clause set obtained by applying the conventional clause form conversion (recursive applications of the transformation $a \vee (b \wedge c) \longrightarrow (a \vee b) \wedge (a \vee c)$) to the disjunction of the CNF expressions for A and B .

$ A $	$ B $	$ A \times B $	$v(A \times B)$	$ x \vee y \wedge (x \Rightarrow A) \wedge (x \Rightarrow B) $	$v(x \vee y \wedge (x \Rightarrow A) \wedge (y \Rightarrow B))$
1	1	1	0	3	2
1	$n > 1$	n	0	$2+n$	2
2	2	4	0	5	2
2	3	6	0	6	2
3	3	9	0	8	2

Table 2.4: CNF conversions for disjunctions. $|X|$ = number of clauses in X ; v = number of auxiliary variables.

The NuSMV RBC CNF Conversion

The original clause form conversion used in NuSMV is based on the structure-preserving clause form conversion [38]. The restrictions of RBCs mean that it is not immediately apparent whether a subgraph has positive or negative polarity; NuSMV overcomes this by renaming all subgraphs in both positive and negative versions:

$$\begin{aligned}
 a \wedge b &\longrightarrow (x_{ab}), (\{\neg x_{ab}, a, b\}, \{x_{ab}, \neg a\}, \{x_{ab}, \neg b\}) \\
 a \leftrightarrow b &\longrightarrow (x_{ab}), (\{\neg x_{ab}, a, b\}, \{\neg x_{ab}, \neg a, \neg b\}, \{x_{ab}, a, \neg b\}, \{x_{ab}, a, \neg b\})
 \end{aligned}$$

Every subformula is renamed, which avoids the exponential size explosion that occurs with the conventional CNF conversion, and it allows the output formula to mimic the structure sharing of the RBC: where a subgraph has more than one incoming edge, the subgraph is encoded only once.

Large chains of disjunctions are common in formulæ generated by BMC. These are particularly inefficiently encoded as each disjunction is represented by a negated conjunction; for a disjunction of $n + 1$ variables, $2n$ clauses are generated with $n + 1$ introduced variables.

The Compact RBC CNF Conversion

We overcome the lack of polarity information by performing an initial depth-first traversal of the graph and marking each node in the graph with the number of incoming vertices with positive and negative polarity (for \leftrightarrow , the polarity is “both”; we increase the counts of both the positive and negative references of the children). This information is then used for renaming: if there are two or more references of the same polarity, then the subformula represented by the vertex is renamed.

The conversion is biased towards generating long clauses. SAT solvers using conflict learning [30] must be able to efficiently handle the large numbers of very long clauses that this technique generates using lazy data structures [29]. This technique also reduces the disadvantage of repeating subclauses where the introduction of extra variables can be avoided (auxiliary variables deepen the search space although we are guaranteed to be able to eliminate them with unit propagation). This results in a number of decisions in the handling of negative polarity conjunctions.

Given a disjunction of two subformulæ a and b , we apply the clause form conversion to the subformulæ giving clause sets A and B . The size of the conversion of $a \vee b$ depends on the technique; Table 2.4 compares the conventional CNF conversion with the structure preserving conversion. The middle pair of columns show the conventional clause form conversion, and the right pair show the structure preserving clause form conversion. We take $|A| = 2, |B| = 3$ as the cut-off point for choosing the conversion method³.

In practice, we take a hybrid option (not shown in the table) of renaming only one child of the disjunction; this avoids introducing one extra variable and an extra clause.

³In fact, we encode $|A| = 2, |B| = 3$ using the conventional method to reduce the number of auxiliary variables; compare with the method in [35] which renames subformulæ even if it results in the same number of clauses being produced.

Transformations

The transformations replace a subtree with a pair of clause sets $X = \langle X^+, X^- \rangle$, for positive and negative polarity; a negation swaps the elements. If no references are made for a particular polarity, the conversion is not made (this condition is omitted from the rules below, for brevity); if there is more than one incoming edge for a particular polarity, a renaming is done. Thus we have several sets of transformation rules, including:

- If no renaming is to be done (that is, the reference counts are ≤ 1 and the number of clauses representing the children is small)

$$\begin{aligned} A \wedge B &\longrightarrow \langle A^+ \cup B^+, A^- \times B^- \rangle, \emptyset \\ a \leftrightarrow b &\longrightarrow \langle (A^+ \times B^+) \cup (A^- \times B^-), (A^+ \times B^+) \cup (A^- \times B^-) \rangle, \emptyset \end{aligned}$$

- If reference counts are > 1

$$\begin{aligned} A \wedge B &\longrightarrow \langle \{\{\neg x\}\}, \{\{x\}\} \rangle, \langle \{\{\neg x\}\} \times (A^+ \cup B^+) \cup (\{\{x\}\} \times A^- \times B^-) \rangle \\ a \leftrightarrow b &\longrightarrow \langle \rangle, \langle \{\{\neg x\}\} \times ((A^+ \times B^+) \cup (A^- \times B^-)) \cup (\{\{x\}\} \times (A^+ \times B^+) \cup (A^- \times B^-)) \rangle \end{aligned}$$

- If the number of clauses representing the children is greater than the cutoff point

$$\begin{aligned} A \wedge B &\longrightarrow \langle A^+ \cup B^+, A^- \times \{\{x\}\} \rangle, \langle \{\{\neg x\}\} \times B^- \rangle \\ a \leftrightarrow b &\longrightarrow \langle (A^+ \times \{\{x\}\}) \cup (A^- \times \{\{\neg x\}\}), (A^+ \times \{\{x\}\}) \cup (A^- \times \{\{\neg x\}\}) \rangle, \\ &\quad \langle \{\{\neg x\}\} \times B^+ \cup (\{\{x\}\} \times B^-) \rangle \end{aligned}$$

In summary, we choose between the conventional clause form conversion and the structure preserving conversion in such a way as to minimise both the clause count and the auxiliary variable count.

2.2.2 Dynamic Step Size Adjustment in Iterative Deepening Search

The correctness of BMC relies on finding a large enough k to ensure completeness of the translation; unfortunately the time taken grows exponentially with k . A typical solution is to repeatedly try to solve the problem with increasing k : in effect, an instance of IDS.

If an iterative procedure such as IDS has the property that solutions at a given iteration are also found at later iterations, it is possible to skip iterations without loss of correctness. We examine the conditions required for skipping to be worthwhile give an algorithm for dynamically adapting the skipping to the behaviour of the search procedure.

We consider the problem f with solution π , written $\pi \models f$. If a solution is found during IDS at depth i , we write $\pi \models_i f$. We write $T(f, i)$ for the time taken for the i th iteration. We make the following simplifying assumptions:

- If f has a solution, this solution may be found by iterative deepening search to some depth k : $\pi \models f \rightarrow \exists k \cdot \pi \models_k f$
- If f has a solution at depth i then it is solvable at all greater depths: $\pi \models_i f \rightarrow \forall j \geq i, \pi \models_j f$
- $T(f, i)$ is monotonically increasing with i : $\forall j \geq i, T(f, j) \geq T(f, i)$

To decide on the size of a step to be taken, we consider the circumstances under which a particular step size will save time overall. Suppose we are currently at depth i during the IDS. It is preferable to solve $\pi \models_{i+\Delta} f$ next rather than the sequence $\forall_{j=i..n} \pi \models_j f$ iff $i + \Delta > n$ and $T(f, i + \Delta) < \sum_{j=i+1}^n T(f, j)$. We choose $i + \Delta$ using a simple heuristic: the first solution of f is equally likely to lie at any depth $k, 0 \leq k < \infty$, so we take $n = i + \lceil \frac{\Delta}{2} \rceil$. While other heuristics are possible, this was found to give sufficiently good performance in testing.

We approximate $T(f, i)$ as an exponential ba^i , which is appropriate for many possible applications including bounded model checking. We determine the a and b using standard statistical methods on the past behaviour of the search, and hence choose a maximum Δ which satisfies the conditions above. We propose the following algorithm for IDS with dynamic step size adjustment.

- Initialise: $a, b \leftarrow \infty$, current depth $i \leftarrow 0$, list of past behaviour $B \leftarrow []$
- Until a solution is found, loop:
 - Solve $\pi \models_i f$, recording the time taken in t
 - Append the pair $\langle i, t \rangle$ to B
 - Use best-fit on B to determine a and b
 - Chose Δ such that $ba^{i+\Delta} < \sum_{j=i+1}^{i+\lceil \frac{\Delta}{2} \rceil} ba^j$
 - $i \leftarrow i + \Delta$

Our preliminary experimental evaluation demonstrates the efficacy of this method on bounded model checking problems; however, other iterative-deepening-style problems must be tried in order to determine the generality of the heuristic chosen.

Chapter 3

Plans

3.1 SNF in Context

The SNF encoding can be related to the two other main approaches to LTL model checking: the standard BMC encoding and the automata-theoretic conversion of [24] and its many refinements.

3.1.1 SNF as a renaming strategy

The BMC encoding and the SNF encoding must be equisatisfiable. In fact, it is clear by inspection that one may be obtained from the other by renaming. For example, the BMC encoding of $\mathbf{G} x$ is $\bigwedge_{i=0..k} x_i$, and by successive renaming we can obtain $x_0 \wedge y_0 \wedge (y_0 \Rightarrow \bigwedge_{i=1..k} x_i)$, then $x_0 \wedge y_0 \wedge (y_0 \Rightarrow x_1 \wedge y_1) \wedge (y_1 \Rightarrow \bigwedge_{i=2..k} x_i)$, and so on. Indeed, this is very similar to a repeated deconstruction of $\mathbf{G} x$ as $x \wedge \mathbf{XG} x$ which can lead to the fixpoint characterisation.

In order to more fully understand the SNF encoding, we must consider its relationship to the BMC encoding. In particular, if the SNF encoding is obtained more quickly and easily than the equivalent renaming of the BMC encoding, it goes some way to justifying the use of SNF.

3.1.2 SNF as automata

Traditional methods for model checking LTL are based on a conversion from the temporal logic to a Büchi automaton [12, 24]. While these techniques are directly applicable to BMC [45], they have not been widely studied. We note that the fixpoint form (Section 2.1.2) is very close to an automaton representation, and in fact we can give a conversion to automata for it. This similarity demands a justification of the use of SNF. An important point to consider here is the closeness of SNF to CNF, and whether it is possible to efficiently convert a Büchi automaton to CNF.

Later work on automata representations of LTL [19, 44] has included preprocessing of the LTL formula to reduce the number and depth of temporal operators; and postprocessing of the automata to reduce the number of transitions and states. The preprocessing should be directly applicable both to SNF and to BMC encodings; while the postprocessing may prove to be adaptable to SNF. In particular, work by Schneider on reducing the number of fairness conditions [41] has strong parallels with work in SNF by Degtyarev et al. on reducing the number of eventualities [16].

3.2 Further work on SNF

The conversion to SNF inevitably introduces a large number of extra variables. Almost as a corollary to Section 3.1.1, we can consider how these variables may be eliminated when they are not needed, in some circumstances resulting in the original BMC encoding. The items below have been implemented, but results from the preliminary evaluation is not promising. Also, the properties which make the transformations possible occur only if past time operators are not considered, though there may be ways of lifting this limitation that we have not yet investigated. This may not be suitable for inclusion in the thesis.

Rule Dependency

Each rule r has exactly one left hand side variable $v(r)$, and a set $S(r)$ of right hand side variables. We say that a rule r_0 is dependent on rule r_1 iff $v(r_0) \in S(r_1)$. We can simplify the encoding by following the dependency graph when encoding to decide which rules to encode in which states. For example, given rules

$$\begin{aligned} \text{start} &\Rightarrow f \wedge x \\ x &\Rightarrow g \\ \text{start} &\Rightarrow g \wedge y \\ y &\Rightarrow \mathbf{X}(g \wedge y) \end{aligned}$$

The conventional encoding would encode the second and fourth rules in all states, and the first and third in the first state only. However, we are only interested in the second rule in the first state, since it depends only on the first rule. By following the dependency graph, we encode the second rule only once, but still make the chain described by the fourth rule.

Implementing this scheme is more difficult than the simple encoding method, but NuSMV includes an efficient DAG implementation which we may be able to use here. This would have the side effect of automatically eliminating duplicated rules.

Rule Chaining

Where a given variable occurs as the left hand side of exactly one rule and in the right hand set of exactly one rule, we may chain the rules together, eliminating variables. Referring to the example rule set above, we see that x is referred to only by the first rule, making it possible to combine the rules to form the

$$\text{start} \Rightarrow f \wedge g$$

By exploiting the dependency graph here, we can make the chaining state-dependent. Both the third and fourth rules refer to variable y , but the sets of states in which they do so are disjoint. This means that the rules are combined to form

$$\text{start} \Rightarrow g \wedge \mathbf{X}(g \wedge \mathbf{X} \dots)$$

The overall effect here is to eliminate any unnecessary introduced variables. The only introduced variables which remain are those necessary to sharing subformulae. It remains to be seen whether this conversion will remove the advantage of SNF: it has been suggested that the increased performance is due, in part, to the breaking up of the specification into constraints covering pairs of states, mirroring the nature of the encoding of the model.

3.3 General Encoding Improvements

3.3.1 Problem Invariants

In most real problems, much of the state space is unreachable from the initial state. A number of methods are available for the removal of some of the unreachable states from SAT encodings of planning [28] which may be adaptable to model checking. A formula describing all of the unreachable states could be unwieldy; constructing an approximation to this formula can make it sufficiently small and simple to incorporate into the encoding while retaining enough expressive power to be useful. Selman and Kautz give a method of constructing a Horn clause approximation to a formula [42] and Williams and Nyack study a planner which uses a related “compilation” scheme to generate an invariant [49]. Rintanen [40] gives an iterative algorithm which incorporates a weakening operation at each iteration to increase the generality of the formula.

For model checking, the established method of producing an expression characterising the reachable states is to use standard BDD techniques to find a fixpoint of the transition relation, starting from the initial state. The resulting BDD can be converted to an RBC for inclusion in the BMC formula. Incorporating a weakening operation into the reachable states construction in the style of [40] or [42] should be particularly beneficial from the BDD point of view: a weaker formula represents more states with less specificity, and hence has fewer nodes.

While work in this area could be promising, it seems unlikely that the time remaining is sufficient to do the topic justice. This will almost certainly not appear in the thesis.

3.4 Integer Encodings

Present implementations of BMC use binary encodings of integers for arithmetic: NuSMV is constructed around a BDD conversion of any arithmetic. There is, however, evidence from the constrain-programming-as-SAT community that binary encodings are inefficient for SAT solvers to deal with and that unary encodings are superior, despite the exponentially higher number of variables involved.

Cumulative unary is a variant of the one bit per value encoding of integers. Consider a set of variables x_1, x_2, \dots representing integer n . The cumulative integer interpretation of a variable x_i is $n > i$. That is, to represent an integer n , we require $x_1 \wedge x_2 \wedge \dots \wedge x_n \wedge \neg x_{n+1} \wedge \neg x_{n+2} \wedge \dots$. The main advantage of this encoding over standard unary is that inequality can be established with a linear, rather than quadratic, number of clauses. In addition, the constraint required to establish the cumulative unary property (conjunction of $x_i \Rightarrow x_{i-1}$) are much simpler than those to establish non-cumulative unary (at-most-one and at-least-one constraints).

Cumulative unary is used by Bailleux and Boufkhad [?] to encode cardinality constraints using a tree of summations of increasing length. They are able to show that their encoding of summation can be solved using unit propagation under certain circumstances (sufficient constraints on the input and output variables to establish a unique result). We generalise their construction here and extend it to other arithmetic operations with a view to using the encoding during bounded model checking.

3.5 Evaluation

Finding good problem sets to evaluate the encodings has consistently been a challenge. The *Texas-97* benchmark suite [4] referred to above includes a variety of interesting problems that we have not yet studied, as does the new VIS test suite [25].

LTL to automata conversions are typically evaluated with a large number of randomly generated formulae together with a number of hand-picked examples. While this method could be used to help evaluate the performance of the LTL encodings discussed, it does not give any indication of the sensitivity of the process to the complexity of the model.

3.6 Outline

Introduction Introduce the topic of the thesis, brief overview and motivation

Literature review

Background theory Introduce the basis for the work to follow

- Temporal logics: LTL and CTL
- Model checking, symbolic model checking
- Bounded model checking
- SAT
- Internals of NuSMV: since NuSMV will be used for all of the evaluation, the methods used to encode the model must be examined.

General Encodings This comes first, as the evaluation of SNF in NuSMV depends on the CNF conversion to be meaningful. In several chapters:

CNF conversion Discussion of tradeoffs and different behaviours in the SAT solver; performance evaluation.

Integer encodings Binary versus unary; consideration of CSP-as-SAT literature; performance evaluations.

Skipping during IDS

Encoding the Specification This also breaks down into several chapters:

Overview of SNF Full discussion of transformations.

SNF for BMC How the encoding works, and how SNF for bounded LTL differs from straight SNF

Fixpoint Form

Evaluation A detailed comparison between SNF, fixpoint, and the traditional encoding: from the point of view of the SAT solver, considering unit propagation times, and from the point of view of a wide variety of benchmark problems.

SNF and Automata Relating SNF to automata-theoretic LTL systems; performance comparison in BMC with automata based encodings.

SNF and CNF Putting SNF in the context of clause form; relating the encoding to renaming of the original encoding; consideration of the unit-propagation behaviour of SNF.

Conclusions

3.7 Proposed Timetable

We aim to submit at the end of the calendar year. In order to achieve this, writing the thesis and investigation of some of the above topics will happen simultaneously. The timetable below includes particular milestones, but throughout the remaining time, work will continue on the benchmark suite which will be used for evaluation at various different stages.

August Start to outline thesis; construct introduction, literature review, background theory sections. Evaluate automata-based encodings, and write section.

September Investigate SNF as a renaming strategy. Write up SNF encodings.

October Evaluate rule dependency and rule chaining. Complete specifications section

November Complete general encodings section

December Proof reading and submission of thesis

Bibliography

- [1] Parosh Aziz Abdulla, Per Bjesse, and Niklas Eén. Symbolic reachability analysis based on SAT-solvers. In S. Graf and M. Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 6th International Conference, TACAS'00*, volume 1785 of *Lecture Notes in Computer Science*, pages 411–425. Springer-Verlag Inc., March 2000.
- [2] Audemard, Cimatti, Kornilowicz, and Sebastiani. Bounded model checking for timed systems. In *IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), LNCS*, volume 22, 2002.
- [3] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani. A SAT based approach for solving formulas over boolean and linear mathematical propositions. In A Voronkov, editor, *Automated Deduction - CADE-18; 18th International Conference on Automated Deduction*, volume 2392, pages 195–210, Copenhagen, Denmark, July 2002.
- [4] Adnan Aziz et al. Examples of HW verification using VIS, 1997. <http://vlsi.colorado.edu/~vis/texas-97/>
- [5] Marco Benedetti and Alessandro Cimatti. Bounded model checking for past LTL. In *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS'03*, Lecture Notes in Computer Science, Warsaw, Poland, April 2003. Springer.
- [6] 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.
- [7] Armin Biere, Edmund Clarke, Richard Raimi, and Yunshan Zhu. Verifying safety properties of a PowerPC[tm] microprocessor using symbolic model checking without BDDs. In N. Halbwachs and D. Peled, editors, *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 60–71. Springer-Verlag Inc, 1999.
- [8] 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.
- [9] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
- [10] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in *Lecture Notes in Computer Science*, pages 495–499, Trento, Italy, July 1999. Springer.

- [11] 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.
- [12] E. Clarke, O. Grumberg, and K. Hamaguchi. Another Look at LTL Model Checking. *Formal Methods in System Design*, 10:47–71, 1997.
- [13] Fady Cooty, Limor Fix, Ranan Fraer, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Y Vardi. Benefits of bounded model checking at an industrial setting. In G Berry, H Comon, and A Finkel, editors, *13th Conference on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 436–453. Springer-Verlag Inc., July 2001.
- [14] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.
- [15] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
- [16] Anatoli Degtyarev, Michael Fisher, and Boris Konev. A simplified clausal resolution procedure for propositional linear-time temporal logic. In *Proceedings of Automated Reasoning with Analytic Tableaux and Related Methods*, volume 2381 of *Lecture Notes in Computer Science*, pages 85–99, Copenhagen, Denmark, July 2002.
- [17] 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.
- [18] 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.
- [19] Kousha Etessami and Gerard J. Holzmann. Optimizing Büchi automata. In Catuscia Palamidessi, editor, *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, volume 1877 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2000.
- [20] Michael Fisher. A resolution method for temporal logic. In *Proceedings of Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, August 1991.
- [21] 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.
- [22] Alan Frisch, Daniel Sheridan, and Toby 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*, volume 2517 of *Lecture Notes in Computer Science*, pages 238–254, Portland, OR, USA, November 2002. Springer.
- [23] Dov Gabbay. The declarative past and imperative future. In H. Barringer, editor, *Proceedings of the Colloquium on Temporal Logic and Specifications*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer-Verlag, 1989.

- [24] Rob Gerth, Doron Peled, Moshe Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In Piotr Dembinski and Marek Sredniawa, editors, *Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, June 1995.
- [25] The VIS Group. VIS verification benchmarks (verilog models). <http://vlsi.colorado.edu/~vis/>.
- [26] Tommi A. Junttila and Ilkka Niemelä. Towards an efficient tableau method for Boolean Circuit Satisfiability Checking. In *Computational logic-CL 2000: First International Conference, London, UK, July 24–28, 2000: proceedings*, volume 1861 of *Lecture Notes in Computer Science and Lecture Notes in Artificial Intelligence*, pages 553–567. Springer-Verlag Inc., 2000.
- [27] Toni Jussila and Ilkka Niemelä. Parallel program verification using BMC. In *Proceedings of Model Checking and Artificial Intelligence*, July 2002.
- [28] Henry A. Kautz and Bart Selman. The role of domain-specific knowledge in the planning as satisfiability framework. In *Artificial Intelligence Planning Systems*, pages 181–189, 1998.
- [29] I. Lynce and J. Marques-Silva. Efficient data structures for fast sat solvers. Technical report, Instituto de Engenharia de Sistemas e Computadores, November 2001.
- [30] João P. Marques Silva and Karem A. Sakallah. Conflict analysis in search algorithms for satisfiability. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, November 1996.
- [31] 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.
- [32] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, 1992.
- [33] Ken L McMillan. Applying SAT methods in unbounded symbolic model checking. In E Brinksma and K Guldstrand Larsen, editors, *Computer Aided Verification; 14th International Conference, CAV 2002*, volume 2404, pages 250–264, Copenhagen, Denmark, July 2002.
- [34] 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.
- [35] Andreas Nonnengart, Georg Rock, and Christoph Weidenbach. On generating small clause normal forms. In Claude Kirchner and Hélène Kirchner, editors, *Fifteenth International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 397–411. Springer-Verlag, 1998.
- [36] Wojciech Penczek, Bożna Woźna, and Andrzej Zbrzezny. Towards bounded model checking for the universal fragment of TCTL. In *Proc. of 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, volume 2469 of *Lecture Notes in Computer Science*, 2002.
- [37] Wojciech Penczek, Bożna Woźna, and Andrzej Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1), May 2003.
- [38] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.

- [39] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. -T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa. VIS: a system for verification and synthesis. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 428–432, New Brunswick, NJ, USA, July/August 1996. Springer Verlag.
- [40] Jussi Rintanen. An iterative algorithm for synthesizing invariants. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and the Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 806–811. AAAI Press, 2000.
- [41] Klaus Schneider. Improving automata generation for linear temporal logic by considering the automaton hierarchy. In *Proceedings 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2250 of *Lecture Notes in Computer Science*, pages 39–54, Havana (Cuba), 2001. Springer-Verlag Inc.
- [42] Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.
- [43] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a SAT-solver. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *Proceedings of Formal Methods in Computer-Aided Design 2000*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2000.
- [44] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 248–263. Springer-Verlag Inc., July 2000.
- [45] Maria Sorea. Bounded model checking for timed automata. In Walter Vogler and Kim Larsen, editors, *Electronic Notes in Theoretical Computer Science*, volume 68. Elsevier, 2003.
- [46] Ofer Strichman. On solving Presburger and linear arithmetic with SAT. In Mark D Aagaard and John W O’Leary, editors, *Formal Methods in Computer-Aided Design, 4th International Conference, FMCAD 2002*, volume 2517 of *Lecture Notes in Computer Science*, pages 160–170, Portland, OR, USA, November 2002. Springer.
- [47] Ofer Strichman, Sanjit A Seshia, and Randal E Bryant. Deciding separation formulas with SAT. In E Brinksma and K Guldstrand Larsen, editors, *Computer Aided Verification; 14th International Conference, CAV 2002*, volume 2404, Copenhagen, Denmark, July 2002.
- [48] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
- [49] Brian C. Williams and P. Pandurang Nayak. A reactive planner for a model-based executive. In M. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1178–1185. Morgan Kaufmann, 1997.
- [50] Bożna Wo’zna and Andrzej Zbrzezny. Reaching the limits for bounded model checking. ICS PAS 958, Warsaw, May 2003.