

# Using Fixpoint Characterisations of LTL for Bounded Model Checking

Daniel Sheridan\*

21st January, 2002

Bounded Model Checking [2] is an approach to the LTL model checking problem which uses an encoding to Boolean satisfiability. The encoding as defined by Biere *et al.* [2] has 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 fixed points of functions [7], used in other model checking systems such as SMV [11].

## 1 Introduction

Model checking is the process of verifying the behaviour of a dynamic system, where a *model* of the implementation of a system is compared with a mathematical *specification* of an intended behaviour of the system. A typical application for model checking is in the field of hardware verification: a model checking tool may be able to find bugs in a hardware design long before the product enters construction and testing.

Bounded model checking (BMC) [2] 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. State of the art Boolean satisfiability solvers can now be used to tackle model checking problems, such as Chaff [12] and Sato [14]. Certain modal operators can be handled by including a check for loops in the state transitions.

Although bounded model checking (BMC) has found some industrial acceptance [5], it suffers from an encoding which produces an exponential number of clauses in the depth of the specification. The present work is concerned with reducing this size explosion, and hence making BMC more accessible.

### 1.1 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

---

\*Artificial Intelligence Group, Department of Computer Science, University of York, England. y010 5DD

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

Figure 1: The Bounded Semantics of LTL

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 in 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  $\pi_i(0) \in I$ ,  $(M, \pi_i) \models f$ .

## 1.2 The Bounded Semantics of LTL

The semantics in Figure 1 are based on the bounded semantics given by Biere et al. [2] and the unbounded semantics given by Clarke et al. [4]. LTL formulæ are *path formulæ*, with semantics given for a path  $\pi$  lying in the model  $M$ . We write  $(M, \pi) \models_k^i f$  if  $f$  holds in the  $i$ th state along  $k$ -bounded path  $\pi^i$ . We abbreviate  $\models_k^0$  as  $\models_k$ .

We say the a path  $\pi$  is a *k-loop* if the  $k$ th state in  $\pi$  is identical to the  $i$ th state, where  $0 \leq i < k$ . Note that we give the semantics here only in terms of paths starting in  $I$ , rather than the more conventional notation of path suffixes.

## 2 The Bounded Model Checking Encoding

The bounded model checking encoding represents a single bounded path  $\pi_{BMC}$  of length  $k$  as a propositional formula, and checks that it violates the bounded semantics (see below) 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

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

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

Table 1: The BMC Encoding for LTL

Biere et al. [2] 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  $\pi$  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  $\pi_{BMC}$  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 semantics (Table 1) that there are two ways of violating each operator in the specification, depending on whether  $\pi_{bmc}$  is a  $k$ -loop; the ‘counterexample’ constraint is therefore a disjunction of the ways in which the specification may be violated.

Given the functions  ${}_lL_k(\pi)$  which holds when  $\pi$  is a  $k$ -loop with  $\pi(k) = \pi(l)$  and  $L_l(\pi) = \bigvee_{l=0}^k {}_lL_k$  which holds when  $\pi$  is any  $k$ -loop, the general translation is defined as<sup>1</sup>:

$$\llbracket M, \pi, f \rrbracket_k := \llbracket M, \pi \rrbracket_k \wedge \left( \neg L_k(\pi) \wedge \llbracket f, \pi \rrbracket_k^0 \right) \vee \bigvee_{l=0}^k ({}_lL_k(\pi) \wedge {}_l\llbracket f, \pi \rrbracket_k^0) \quad (1)$$

$\llbracket M, \pi \rrbracket_k$  denotes the encoding of the transition relation of  $M$  as a constraint on  $\pi$  with bound  $k$ ;  $\llbracket f, \pi \rrbracket_k^i$  and  ${}_l\llbracket f, \pi \rrbracket_k^i$  denote the encoding of the LTL formula  $f$  evaluated along path  $\pi$  at time  $i$ , where  $\pi$  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 [2]; we will not repeat their proofs here.

### 3 Fixpoint Characterisations for LTL

Emerson and Clarke [7] give fixpoint characterisations for CTL; these are easily adapted for LTL. They are given for *unbounded* temporal logic, however the fixpoint characterisations 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  $\mathbf{X}_1$  with semantics

$$(M, \pi) \models_k^i \mathbf{X}_1 f_1 \Leftrightarrow \begin{cases} (M, \pi(i+1)) \models_k f_1 & \text{if } \pi \text{ is a } k\text{-loop} \\ \perp & \text{otherwise} \end{cases}$$

<sup>1</sup>This comes from definition 15 in [2]

Each bounded LTL formula  $f$  is identified with a set of paths in which it holds — the set  $\{\pi \mid (M, \pi) \models_k f\}$ . This allows us to give the following characterisations as the least and greatest fixpoints of functions on sets of paths:

$$\begin{aligned} \mathbf{F} f_1 &= \mu Z. f_1 \vee \mathbf{X} Z & \mathbf{G} f_1 &= \nu Z. f_1 \wedge \mathbf{X}_1 Z \\ [f_1 \mathbf{U} f_2] &= \mu Z. f_2 \vee (f_1 \wedge \mathbf{X} Z) & [f_1 \mathbf{R} f_2] &= \nu Z. f_2 \wedge (f_1 \vee \mathbf{X} Z) \end{aligned}$$

Clarke et al. [4] give proofs for the characterisations for the (unbounded) CTL operators **EG** and **EU**, which may easily be adapted to LTL both for the bounded semantics and for the other operators. We prove the characterisations for the (bounded) operators **G** and **F** in a similar way; the other characterisations may be proven in a similar manner.

**Lemma 2.**  $\tau(Z) = f_1 \wedge \mathbf{X}_1 Z$  is monotonic

*Proof.* To prove that  $P_1 \subseteq P_2 \Rightarrow \tau(P_1) \subseteq \tau(P_2)$ , consider path  $\pi \in \tau(P_1)$ . We know that  $(M, \pi) \models_k f_1 \wedge \mathbf{X}_1 Z$ . From the definition of  $\models_k$ ,  $(M, \pi) \models_k f_1$  and if  $\pi$  is a  $k$ -loop with suffix  $\pi'$ ,  $\pi' \in P_1$ . Since  $P_1 \subseteq P_2$ ,  $\pi' \in P_2$  and hence  $\pi \in \tau(P_2)$ .

If  $\pi$  is not a  $k$ -loop, then  $\tau(P_1) = \tau(P_2) = \emptyset$ . □

**Lemma 3.** Let  $\tau(Z) = f_1 \wedge \mathbf{X}_1 Z$  and let  $\tau^{i_0}(\top)$  be the limit of  $\bigcap_i \tau^i(\top)$ . For all  $\pi \in M$ , if  $\pi \in \tau^{i_0}(\top)$  then  $(M, \pi) \models_k f_1$  and if  $\pi$  is a  $k$ -loop with suffix  $\pi'$  then  $\pi' \in \tau^{i_0}(\top)$ .

*Proof.* Let  $\pi \in \tau^{i_0}(\top)$ . Since  $M$  is finite and  $\tau$  is monotonic,  $\tau(\tau^{i_0}(\top)) = \tau^{i_0}(\top)$ , and so  $\pi \in \tau(\tau^{i_0}(\top))$ . Using the definition of  $\tau$ , we know that  $(M, \pi) \models f_1$  and if  $\pi$  is a  $k$ -loop with suffix  $\pi'$  then  $\pi' \in \tau^{i_0}(\top)$ . □

**Lemma 4.**  $\mathbf{G} f_1$  is a fixpoint of the function  $\tau(Z) = f_1 \wedge \mathbf{X}_1 Z$ .

*Proof.* Consider a  $\pi$  such that  $(M, \pi) \models_k \mathbf{G} f_1$ . From the definition of  $\models_k$ , if  $\pi$  is a  $k$ -loop with suffix  $\pi'$  then  $(M, \pi') \models_k \mathbf{G} f_1$ , and hence  $(M, \pi) \models_k \mathbf{X}_1 \mathbf{G} f_1$ . □

**Lemma 5.**  $\mathbf{G} f_1$  is the greatest fixpoint of the function  $\tau(Z) = f_1 \wedge \mathbf{X}_1 Z$ .

*Proof.* We prove this by contradiction. Suppose we have some  $F$  such that  $F$  is a fixpoint of  $\tau$  and  $\mathbf{G} f_1 \subset F$ . There must be some path  $\pi \in F$  which is not contained in  $\mathbf{G} f_1$ . Since  $F \subseteq (\top)$ , by monotonicity  $F \subseteq \tau^i(\top)$  for any  $i$ . Therefore  $\pi \in \tau^{i_0}(\top)$  and by Lemma 3,  $\pi$  is a  $k$ -loop of states such that each state satisfies  $f_1$ . Thus  $(M, \pi) \models_k \mathbf{G} f_1$ . □

**Lemma 6.**  $\tau(Z) = f_1 \vee \mathbf{X} Z$  is monotonic

*Proof.* As above, let  $P_1 \subseteq P_2$  and consider path  $\pi \in \tau(P_1)$ . Since  $(M, \pi) \models_k f_1 \vee \mathbf{X} Z$ , we deduce that either  $(M, \pi) \models_k f_1$ , hence  $\pi \in \tau(P_2)$ ; or for the suffix  $\pi'$  of  $\pi$ , if  $\pi$  is a  $k$ -loop or  $\pi'$  starts less than  $k$  steps from an initial state, then  $\pi' \in P_1$ . Since  $P_1 \subseteq P_2$ ,  $\pi' \in P_2$  and hence  $\pi \in \tau(P_2)$ .

If  $\pi$  is not a  $k$ -loop and  $\pi'$  is not less than  $k$  steps from an initial state, then  $\tau(P_1) = \tau(P_2) = \emptyset$ . □

**Lemma 7.** Let  $\tau(Z) = f_1 \vee \mathbf{X} Z$  and let  $\tau^{i_0}(\perp)$  be the limit of  $\bigcup_i \tau^i(\perp)$ . For all  $\pi \in M$ , if  $\pi \in \tau^{i_0}(\perp)$  and  $(M, \pi) \not\models_k f_1$  then if  $\pi$  has a suffix  $\pi'$  then  $\pi' \in \tau^{i_0}(\perp)$ .

*Proof.* As above, let  $\pi \in \tau^{i_0}(\perp)$ . By monotonicity,  $\tau(\tau^{i_0}(\perp)) = \tau^{i_0}(\perp)$ , so  $\pi \in \tau(\tau^{i_0}(\perp))$ . Therefore either  $(M, \pi) \models f_1$  or if  $\pi$  has a suffix  $\pi'$ ,  $(M, \pi') \models_k \tau^{i_0}(\perp)$ . □

**Lemma 8.**  $\mathbf{F} f_1$  is a fixpoint of the function  $\tau(Z) = f_1 \vee \mathbf{X} Z$ .

*Proof.* Consider a  $\pi$  such that  $(M, \pi) \models_k \mathbf{F} f_1$ . From the definition of  $\models_k$ , either  $(M, \pi) \models_k f_1$ , or if  $\pi$  has a suffix  $\pi'$  then  $(M, \pi') \models_k \mathbf{F} f_1$  and hence  $(M, \pi) \models_k \mathbf{X} \mathbf{F} f_1$ .  $\square$

**Lemma 9.**  $\mathbf{F} f_1$  is the least fixpoint of the function  $\tau(Z) = f_1 \vee \mathbf{X}Z$ .

*Proof.* As above, we prove this by contradiction. Consider  $F$ , which is a fixpoint of  $\tau$  and  $\mathbf{F} f_1 \supset F$ . There must be some path  $\pi \in \mathbf{F} f_1$  which is not contained in  $F$ . Since  $F \supseteq (\perp)$ , by monotonicity  $F \supseteq \tau^i(\perp)$  for any  $i$ . Therefore  $\pi \notin \tau^{i_0}(\perp)$  and by Lemma 7, there is no state on the path in which  $f_1$  is satisfied. Thus  $(M, \pi) \not\models_k \mathbf{F} f_1$ .  $\square$

## 4 Fixpoint-Based Encodings

Gabbay's Separation Theorem [10] 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, building on METATEM [1], an executable temporal logic, defined a normal form for temporal logic based on the Separation Theorem and gave a series of transformations for reaching it [8]. We have adapted the Separated Normal Form (SNF) here for bounded LTL (based on the adaptation to CTL by Bolotov and Fisher [3]).

SNF takes the same general form as the Separation Theorem; to adapt the form for bounded semantics, we need a path operator that captures the concept of *all reachable states*. For unbounded LTL, the  $\mathbf{G}$  operator is sufficient; here we give the semantics for a modified operator for bounded LTL.

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

$\mathbf{G}_k$  has the same relationship (with regard to fixpoints) with  $\mathbf{X}$  as  $\mathbf{G}$  has with  $\mathbf{X}_1$ . Introducing this operator allows us to restate the general form as

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

Since LTL and CTL have no explicit past-time operators, Bolotov and Fisher [3] introduce the **start** operator, which holds only at the beginning of time.

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

The addition of this single operator is sufficient to allow Fisher's transformations to be used almost unchanged.

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

$$\begin{array}{ll} \mathbf{start} \Rightarrow \bigvee_j l_j & \text{An initial rule} \\ \bigwedge_i l_i \Rightarrow \mathbf{X}_1 \bigvee_j l_j & \text{A global } \mathbf{X}_1\text{-rule} \\ \bigwedge_i l_i \Rightarrow \mathbf{X} \bigvee_j l_j & \text{A global } \mathbf{X}\text{-rule} \\ \bigwedge_i l_i \Rightarrow \mathbf{F} \bigvee_j l_j & \text{A global } \mathbf{F}\text{-rule} \end{array}$$

and 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 [3], or from standard propositional logic.

$$\begin{aligned}
T_G(\{P \Rightarrow \mathbf{G} f\} \cup \Psi) &= \left\{ \begin{array}{l} P \Rightarrow f \wedge x \\ x \Rightarrow \mathbf{X}_1(f \wedge x) \end{array} \right\} \cup \Psi \\
T_U(\{P \Rightarrow f \mathbf{U} g\} \cup \Psi) &= \left\{ \begin{array}{l} P \Rightarrow g \vee (f \wedge x) \\ x \Rightarrow \mathbf{X}(g \vee (f \wedge x)) \\ P \Rightarrow \mathbf{F} g \end{array} \right\} \cup \Psi \\
T_{ren1}(\{P \Rightarrow \mathbf{G} f(\mathbf{F} g)\} \cup \Psi) &= \left\{ \begin{array}{l} P \Rightarrow \mathbf{G} f(x) \\ x \Rightarrow \mathbf{F} g \end{array} \right\} \cup \Psi
\end{aligned}$$

In each of the above transformations, a new variable  $x$  is introduced: the conversion to SNF introduces one variable for each fixpoint characterisation (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).

#### 4.1 Correctness of SNF Transformations

We take the outline of the proof from [9]. 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)<sup>2</sup> as  $\vdash_{\text{QPTL}} f \Leftrightarrow \exists x.T(f)$ . The proof is given for the case that the rule set is a singleton set, since for all transformations,  $T$  is independent of  $\Psi$ . The proofs may easily be extended to non-empty  $\Psi$ . The proof below is for the transformation associated with the  $\mathbf{G}$  operator, but is adaptable to other transformations.

**Lemma 10.** *For sufficiently large  $k$ ,  $(M, \pi) \models_k \mathbf{G} f_1$  if and only if  $(M', \pi) \models_k (x \wedge f_1)$  and  $(M', \pi) \models_k \mathbf{G}_k(x \Leftrightarrow \mathbf{X}_1(x \wedge f_1))$  where  $M \sim^x M'$ .*

*Proof.* Consider the fixpoint expression from Section 3,  $\tau(Z) = f_1 \wedge \mathbf{X}_1 Z$ . We introduce the variable  $x$  such that for all  $n$ ,

$$(M', \pi) \models_k^n x \Leftrightarrow (M', \pi) \models_k^n \mathbf{X}_1 \tau^{k-n}(\text{true})$$

By substituting the definition of  $\tau$  once and the definition of  $x$ , we have  $(M', \pi) \models_k^n x \Leftrightarrow (M', \pi) \models_k^n \mathbf{X}_1(f_1 \wedge x)$  and by reference to the semantics,  $(M', \pi) \models_k \mathbf{G}_k(x \Leftrightarrow \mathbf{X}_1(x \wedge f_1))$

From Lemma 5,  $(M', \pi) \models_k x \Leftrightarrow \mathbf{G} f_1$ , and by unrolling  $\tau$  by one step and substituting the definition of  $x$ , we get  $(M', \pi) \models_k f_1 \wedge x$ .  $\square$

**Theorem 11.** *For any rule  $A$ ,  $\vdash_{\text{QPTL}} A \Leftrightarrow \exists x.T_G(A)$*

*Proof.* We prove each direction independently:

<sup>2</sup>See [13] 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  $p$ .

- $\vdash_{\text{QPTL}} A \Rightarrow \exists x.T_G(A)$

From the definition of SNF, the set of rules  $\{A\}$  represents the expression  $\mathbf{G}_k(P \Rightarrow \mathbf{G} B)$ . We substitute the result of Lemma 10:

$$\begin{aligned} \mathbf{G}_k(P \Rightarrow \mathbf{G} B) &\Rightarrow \exists x.\mathbf{G}_k(x \Leftrightarrow \mathbf{X}_1(x \wedge B)) \wedge \mathbf{G}_k(P \Rightarrow (x \wedge B)) \\ &\Rightarrow \exists x.\mathbf{G}_k((x \Leftrightarrow \mathbf{X}_1(x \wedge B)) \wedge (P \Rightarrow (x \wedge B))) \end{aligned}$$

which implies the set of rules  $\{x \Rightarrow \mathbf{X}_1(x \wedge B), P \Rightarrow x \wedge B\}$ .

- $\vdash_{\text{QPTL}} \exists x.T_G(f) \Rightarrow f$

Starting with the transformed set of rules  $\{x \Rightarrow \mathbf{X}_1(x \wedge B), P \Rightarrow x \wedge B\}$ , and exploiting the corollary of Lemma 10,  $(M', \pi) \models_k (x \wedge f_1) \Leftrightarrow (M', \pi) \models_k \mathbf{G} f_1$

$$\begin{aligned} &\mathbf{G}_k((x \Rightarrow \mathbf{X}_1(x \wedge B)) \wedge (P \Rightarrow (x \wedge B))) \\ \Leftrightarrow &\mathbf{G}_k(x \Rightarrow \mathbf{X}_1(x \wedge B)) \wedge \mathbf{G}_k(P \Rightarrow (x \wedge B)) \\ \Leftrightarrow &\mathbf{G}_k(x \Rightarrow \mathbf{X}_1 \mathbf{G} B) \wedge \mathbf{G}_k(P \Rightarrow (x \wedge B)) \\ \Rightarrow &\mathbf{G}_k((x \Rightarrow \mathbf{X}_1 \mathbf{G} B) \wedge (P \Rightarrow (x \wedge B))) \\ \Rightarrow &\mathbf{G}_k(P \Rightarrow ((\mathbf{X}_1 \mathbf{G} B) \wedge B)) \\ \Rightarrow &\mathbf{G}_k(P \Rightarrow \mathbf{G} B) \end{aligned}$$

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

□

## 4.2 The Fixpoint Normal Form

A key observation to make about SNF is that it treats  $\mathbf{F}$  as a fundamental operation. Since we do not have the same restrictions as the original application domain of SNF (temporal resolution), it seems appropriate to introduce a transformation which uses the fixpoint characterisation of  $\mathbf{F}$  to reduce the set of possible rules even further.

To make the satisfiability meaningful in the context of this transformation when using bounded temporal logic, we introduce a new past-time operator **bound**, which holds only at the temporal bound set on the system. This is used, in effect, as a statement that the event that we are looking for does not occur in the otherwise unconstrained future.

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

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

The set of permissible rules is now reduced to

$$\begin{array}{ll} \mathbf{start} \Rightarrow \bigvee_j l_j & \text{An initial rule} & \bigwedge_i l_i \Rightarrow \mathbf{X}_1 \bigvee_j l_j & \text{A global } \mathbf{X}_1\text{-rule} \\ \mathbf{bound} \Rightarrow \bigvee_j l_j & \text{A final rule} & \bigwedge_i l_i \Rightarrow \mathbf{X} \bigvee_j l_j & \text{A global } \mathbf{X}\text{-rule} \end{array}$$

### 4.3 Correctness of the Fixpoint Normal Form Transformation

Only one new transformation rule is required to convert from SNF to the fixpoint normal form; we show a proof of this rule in the same style as Section 4.1.

**Lemma 12.** *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.* This proceeds in the same way as Lemma 10.  $\square$

**Theorem 13.** *For any rule  $A$ ,  $\vdash_{\text{QPTL}} A \Leftrightarrow \exists x.T_F(A)$*

*Proof.* Proving each direction independently:

- $\vdash_{\text{QPTL}} A \Rightarrow \exists x.T_F(A)$

Substituting Lemma 12,

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

which implies the set of rules  $\{x \Rightarrow \mathbf{X}(x \vee B), \mathbf{bound} \Rightarrow \neg x, P \Rightarrow x \vee B\}$ .

- $\vdash_{\text{QPTL}} \exists x.T_F(f) \Rightarrow f$

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 12,  $(M', s_i) \models_k (x \vee f_1) \Leftrightarrow (M', s_i) \models_k \mathbf{F} f_1$  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  $\{P \Rightarrow \mathbf{F} B\}$ .  $\square$

### 4.4 Encoding the Normal Forms

The methods for using SNF and the fixpoint normal form in the encoding for BMC are identical; we will not differentiate between them in this section.

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

$$\begin{aligned} \mathbf{start} &\Rightarrow \bigvee_j l_j & \mathbf{G}_k\left(\bigwedge_i l_i \Rightarrow \mathbf{X}_1 \bigvee_j l_j\right) \\ \mathbf{bound} &\Rightarrow \bigvee_j l_j & \mathbf{G}_k\left(\bigwedge_i l_i \Rightarrow \mathbf{X} \bigvee_j l_j\right) \\ \mathbf{G}_k\left(\bigwedge_i l_i \Rightarrow \mathbf{F} \bigvee_j l_j\right) & & \end{aligned}$$

$f$	$\llbracket f, \pi \rrbracket_k^0$	$\imath \llbracket f, \pi \rrbracket_k^0$
<b>start</b> $\Rightarrow f_1$	$\llbracket f_1, \pi \rrbracket_k^0$	$\imath \llbracket f_1, \pi \rrbracket_k^0$
<b>bound</b> $\Rightarrow f_1$	$\llbracket f_1, \pi \rrbracket_k^k$	$\imath \llbracket f_1, \pi \rrbracket_k^k$
<b>G<sub>k</sub></b> ( $f_1 \Rightarrow \mathbf{X}_1 f_2$ )	$\perp$	$\bigwedge_{n=0}^k (\imath \llbracket f_1, \pi \rrbracket_k^n \Rightarrow \imath \llbracket f_2, \pi \rrbracket_k^{n+1})$
<b>G<sub>k</sub></b> ( $f_1 \Rightarrow \mathbf{X} f_2$ )	$\bigwedge_{n=0}^{k-1} (\llbracket f_1, \pi \rrbracket_k^n \Rightarrow \llbracket f_2, \pi \rrbracket_k^{n+1})$	$\bigwedge_{n=0}^k (\imath \llbracket f_1, \pi \rrbracket_k^n \Rightarrow \imath \llbracket f_2, \pi \rrbracket_k^{n+1})$
<b>G<sub>k</sub></b> ( $f_1 \Rightarrow \mathbf{F} f_2$ )	$\bigwedge_{n=0}^k (\llbracket f_1, \pi \rrbracket_k^n \Rightarrow \bigvee_{m=n}^k \llbracket f_2, \pi \rrbracket_k^m)$	$\bigwedge_{n=0}^k (\imath \llbracket f_1, \pi \rrbracket_k^n \Rightarrow \bigvee_{m=n}^k \imath \llbracket f_2, \pi \rrbracket_k^{m+1})$

Table 2: The BMC Encoding for SNF-LTL

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 the more efficient encodings 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 [2].

## 4.5 Commentary

We make a number of observations about the propositional form of the SNF and fixpoint based encodings for BMC.

For propositional  $f$ ,  $\llbracket f, \pi \rrbracket_k^i \equiv \imath \llbracket f, \pi \rrbracket_k^i$ , 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  $\pi$  is a loop; in many circumstances, the checks for the looping nature of  $\pi$  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 the optimisation.

The disjunction in the encoding of the global **F**-rule leads to very large clauses – up to  $k$  variables. The alternative of using the transformation in the fixpoint normal form yields a clause or clause set for each consecutive pair of states, in much the same way as the transformation and encoding of the **G** operator.

## 5 Results

The benchmark circuits have been kept deliberately simple as it is the encoding of the specification not the model that differentiates the encodings discussed above. These circuits have the advantage that they can be meaningfully scaled up to provide successively more difficult problems.

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 [6]. The specifications grow with the number of elements in the shift register; in the case of a three element register,

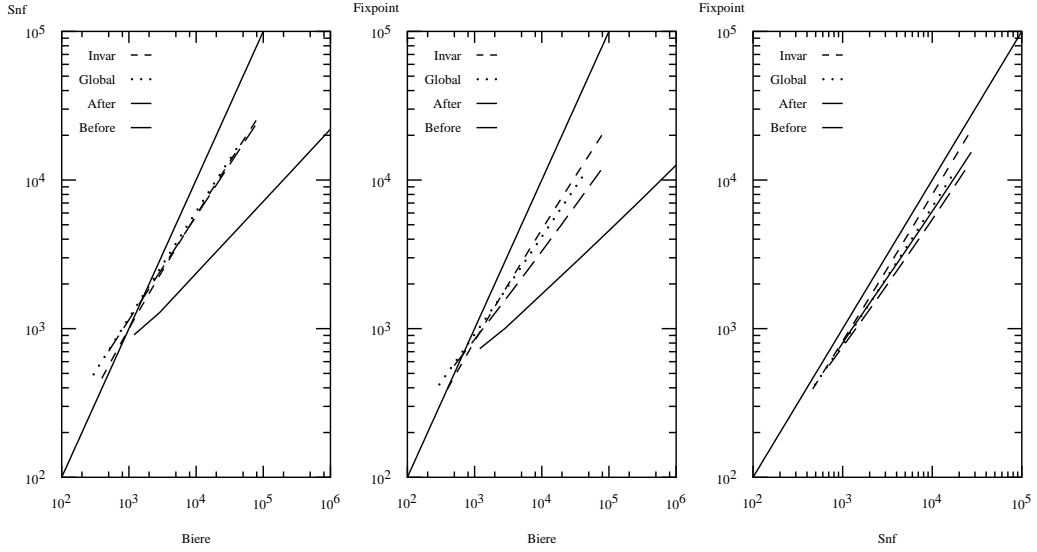


Figure 2: Number of clauses

- 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*  $\triangleq \neg x_0 \wedge \neg x_1 \wedge \neg x_2$  too):  $(((in \wedge empty) \Rightarrow [\neg x_2 \mathbf{U}((x_1 \wedge \neg x_2) \vee \mathbf{G} \neg x_2)]) \mathbf{U}(x_2 \vee \mathbf{G} x_2))$

In addition, we have a simple invariant property which checks that data propagates in the correct order:

- Invariant (depth 1):  $\mathbf{G}(in \Rightarrow \mathbf{X}(x_0 \wedge \mathbf{A}\mathbf{X}(x_1 \wedge \mathbf{X} x_2)))$

We duplicate all of the specifications for variable going low rather than high (for example,  $\mathbf{G}(!in \Rightarrow \mathbf{F}(!x_2))$ ). Note that none of these specifications alone is sufficient for a shift register. For example, the invariant specification holds for a system with all registers stuck on.

There are three encoding procedures, referred to in the graphs as *Biere* (for the original encoding from [2]), *Snf* and *Fixpoint* (for the encodings defined in Section 4). We refer to the four specifications for the shift register as (in order of depth) *Invar*, *Global*, *After*, and *Before*.

## 5.1 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 *Biere*, 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 a polynomial improvement in the number of clauses.

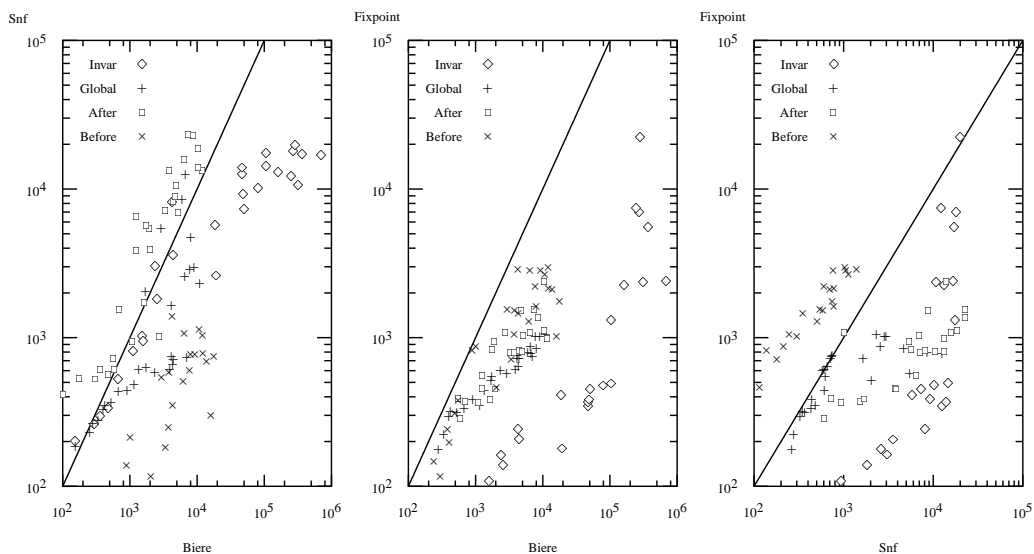


Figure 3: zChaff decisions

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 them. 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. The Global specification has only one occurrence of  $\mathbf{G}$ ; it differs from Invar in the depth of the expression to which it is applied (1 versus 0).

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

## 5.2 zChaff decisions

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]. Timing a computer program is inaccurate, so we take the number of decisions made by zChaff (Figure 3); this has the additional advantage of being repeatable.

The behaviour is far less clear than for the number of clauses; zChaff is a complex system. We can pick out trends, however. Broadly, the Fixpoint encoding almost always results in fewer decisions than the Biere encoding; the Snf encoding does so to a lesser extent, depending on the specification.

For Snf, Invar and Global broadly follow the behaviour predicted by the clause counts: Global exhibits a shallower gradient than Invar, corresponding with its greater depth and indicating a polynomial improvement in the number of decisions. This behaviour is also apparent for Fixpoint, although it is much less pronounced for Global: we see from the Snf vs. Fixpoint graph that Fixpoint only begins to show an advantage

as the problems get harder.

After shows similar behaviour to Global; indeed, the values appear to be parallel for Snf. Fixpoint shows a constant factor improvement over Snf.

The most complex specification, Before, seems to show the least improvement: for both Snf and Fixpoint, the gradient is not significantly less than the equivalence line, indicating a constant factor improvement. Comparing Snf with Fixpoint, we see a polynomial improvement offset by a constant factor; we would expect to see an advantage to Fixpoint for very large problems.

While the performance increases with nesting depth for most of the specifications, the lack of improvement for Before comes as a surprise, considering the more dramatic results in the previous section. The results for Fixpoint are consistently better than Biere, and are more uniform; it also outperforms Snf in all cases except Before.

While above we discounted timings as a value for comparison, it is also possible that the number of decisions is not telling the whole story; the time taken to make each decision depends on a number of factors, including the number and properties of clauses in the system.

## 6 Conclusions

We have described a two new encoding scheme for bounded model checking which builds on the existing encodings and uses fixpoint the 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 polynomial in the size of the problem instance, for both encodings, and also that the improvement in performance in the SAT checker can be polynomial in the size of the problem instance, depending on the specification. Finally, we have demonstrated that extending the SNF transformations with a transformation for the  $\mathbf{F}$  operator results in similar advantages over SNF in most cases.

## References

- [1] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: A framework for programming in temporal logic. In *Proceedings of Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 94–129. Springer-Verlag Inc, June 1989.
- [2] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In W. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems. 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag Inc., July 1999.
- [3] A. Bolotov and M. 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.
- [4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

- [5] F. Copt, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. 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.
- [6] M. Dwyer, G. Avrunin, and J. 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.
- [7] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In J. v. L. 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.
- [8] M. Fisher. A resolution method for temporal logic. In *Proceedings of Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, Aug. 1991.
- [9] M. Fisher and P. 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, Feb. 1992.
- [10] D. 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.
- [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] P. 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, Jan. 1982.
- [14] H. Zhang. SATO: An efficient propositional prover. In M. E. Stickel, editor, *14th International Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 272–275. Springer-Verlag, 1997.