A Direct Algorithm for Multi-Valued Bounded Model Checking

Jefferson O. Andrade and Yukiyoshi Kameyama

Department of Computer Science, University of Tsukuba
`joandrade@logic.cs.tsukuba.ac.jp,kameyama@acm.org`

Abstract. Multi-valued Model Checking is an extension of classical, two-valued model checking with multi-valued logic. Multi-valuedness has been proved useful in expressing additional information such as incompleteness, uncertainty, and many others, but with the cost of time and space complexity. This paper addresses this problem, and proposes a new algorithm for Multi-valued Model Checking. While Chechik et al. have extended BDD-based Symbolic Model Checking algorithm to the multi-valued case, our algorithm extends Bounded Model Checking (BMC), which can generate a counterexample of minimum length efficiently (if any). A notable feature of our algorithm is that it directly generates conjunctive normal forms, and never reduces multi-valued formulas into many slices of two-valued formulas. To achieve this feature, we extend the BMC algorithm to the multi-valued case and also devise a new translation of multi-valued propositional formulas. Finally, we show experimental results and compare the performance of our algorithm with that of a reduction-based algorithm.

1 Introduction

Model Checking [7] is a way to verify (or refute) a temporal specification against a system. Multi-valued Model Checking (mvMC) [5] is an extension of the ordinary model checking with multi-valued logic [10]. Multi-valuedness can be used to express additional information on the system being verified, such as incompleteness, uncertainty, authenticity, capability, and many others, and has been proved useful in various areas of system and software verification.

The extra expressivity comes at a cost: suppose the domain of truth values is a 2^n-valued Boolean algebra. A naive approach is to decompose the multi-valued Kripke structure and specification into n components (or *slices*), each of which constitutes the i-th bit of the original one, and run an ordinary, two-valued model checker n times. Then the execution time would be n times as long as its 2-valued counterpart. In most applications of multi-valued model checking, each component of the Kripke structure is similar to each other, checking these n slices independently is obviously suboptimal to do multi-valued model checking.

To solve this problem, Chechik and others formulated the multi-valued model checking problems with Quasi-Boolean algebra as the domain of truth values [5, 4, 6, 8], and proposed a symbolic model checking algorithm based on multi-valued extension of Binary Decision Diagram (BDD). If several slices of Kripke structure share the same structures, they can be shared as in the case of BDD's.

This paper takes an alternative way: we propose a multi-valued model checking algorithm based on bounded model checking (BMC) [1]. In the context of two-valued model checking, BMC is known to be useful in finding a counterexamples of minimum length (and sometimes verifying specifications) efficiently by making use of the propositional SAT solvers[1]. We consider the BMC-based approach is promising in multi-valued model checking, since, in the translation from propositional formulas to conjunctive normal forms, all common subformulas can be shared, hence the similarity among slices (components) of the multi-valued Kripke structures can be captured automatically. BMC usually finds a counterexample faster than SMC, but verification by BMC is usually slower than SMC, so these two methods are complementary.

We propose a direct algorithm, rather than a reduction-based one. The reduction-based one reduces multi-valued formulas into many bits of two-valued formulas, and uses the ordinary, two-valued model checker. A merit of this approach is that we do not have to invent a new algorithm nor a new tool. A big drawback is bad performance: if the truth domain is a finite Boolean algebra that has 2^n elements (that is, there are n slices), then we must run a two-valued model checker n times. Instead of that we propose an algorithm which keeps the multi-valued formulas as far as possible, and finally generates conjunctive normal forms directly. To achieve this feature we propose a new translation from multi-valued propositional formulas into conjunctive normal forms.

We compare the following three algorithms for multi-valued bounded model checking:

– *Naive algorithm*: we reduce the Kripke structures and the specification formula into n slices, and use the ordinary, two-valued BMC algorithm n times.
– *Reduction-based algorithm*: we generate a multi-valued propositional formula which represents a bounded model, reduce it to sliced (two-valued) propositional formulas, and finally convert them to conjunctive normal forms (CNF).
– *Direct algorithm*: we generate a multi-valued propositional formula which represents a bounded model, and translate this formula directly to CNF.

We have implemented the latter two and compared their performance, since Naive algorithm is far less efficient. The experimental results are encouraging: for CNF generation, Direct algorithm is more efficient in time and space than Reduction-based one, and for SAT solving, their execution time is comparative. Since CNF generation occupies large part of the total execution time, Direct algorithm seems to be preferable. In addition, as the size of the lattice grows, the merit of the direct algorithm becomes larger.

The contribution of this paper can be summarized as follows:

– We formulate Multi-valued Model Checking problems in the context of Bounded Model Checking.

[1] SAT for satisfiability.

– We develop not only a reduction-based algorithm for mvMC but also a direct translation algorithm.
– We compare the efficiency of the direct algorithm and the reduction one using our prototypical implementation, which indicates the direct translation is promising.

The rest of the article is organized as follows. Section 2 introduces multi-valued Kripke structures (mvKS) and the semantics of multi-valued LTL as well as multi-valued model checking. Section 3 introduces translations from multi-valued model checking to multi-valued propositional satisfiability. The subsequent sections 4, 5 and 6 explain the translation algorithms in detail. Section 7 explains our implementation, and gives performance measurement of experimental results. Section 8 gives conclusion and future work.

2 Basics of Multi-Valued Model Checking

We introduce the basic definitions of multi-valued Model Checking (mvMC). The formalization in this section is a slightly extended version of those found in the literature [5, 4, 6, 8], and readers are encouraged to refer to them for in-depth explanation and motivating examples on this topic.

2.1 Lattice as the domain of truth values

Classical model checking works for Kripke structures and specification written as a temporal logic formula. In multi-valued model checking, both Kripke structures and temporal logic are extended to multi-valued ones where the domain of truth values forms a lattice with possibly more than two elements. In this paper we use finite Boolean algebras (with 2^n elements) as the domains of truth values. Although this is a more restrictive choice than those found in the literature (for instance, Chechik et al. studied mvMC on Quasi-Boolean algebras [5]), the focus of this paper is in the algorithmic and implementational aspects. We will discuss the extension to more general lattices briefly in Section 8.

We assume that $\mathcal{L}$ is a finite Boolean algebra with a set L of values and operations $\sqcap$ (meet), $\sqcup$ (join) and $\sim$(negation). An order-n Boolean algebra is the one with 2^n elements for a natural number n, and its element can be represented by n bits (written, for instance, `#1101`). Then lattice operations join and meet can be performed in a bit-wise manner. We write $\top \overset{\text{def}}{=}$ `#11...1` (top) and $\bot \overset{\text{def}}{=}$ `#00...0` (bottom).

2.2 Multi-Valued Kripke Structure

Classical Kripke structures are extended to multi-valued ones as follows.

Definition 1 (Multi-Valued Kripke Structure) *A* Multi-Valued Kripke Structure *(mvKS) over a a lattice $\mathcal{L} = \langle L, \sqcap, \sqcup, \sim \rangle$ is the tuple $\mathcal{M} = (S, \mathcal{I}, \mathcal{R}, \mathcal{AP}, \mathcal{V})$ such that:*

- *S is a finite set of states.*
- *$\mathcal{I} : S \to L$ specifies the initial states with its "degree".*
- *$\mathcal{R} : S \times S \to L$ is the transition relation in the lattice.*
- *$\mathcal{AP}$ is a finite set of atomic propositions.*
- *$\mathcal{V} : S \times \mathcal{AP} \to L$ is the valuation function that determines the truth value of an atom at a state.*

Some authors [3] keep the transition relation defined over a two-valued lattice. Here we gave a general definition for it.

Example 1 (mvKS) *The following data determines an mvKS over order-4 Boolean algebra:*

$$
\begin{aligned}
S &= \{s_0, s_1, s_2, s_3\} \\
\mathcal{I}(x) &= \begin{cases} \#1100 & \text{if } x = s_0 \\ \#0011 & \text{if } x = s_3 \\ \bot & \text{otherwise} \end{cases} \\
\mathcal{R}(x,y) &= \begin{cases} \top & \text{if } (x,y) \in \{(s_0,s_1),(s_1,s_2),(s_3,s_0)\} \\ \#1010 & \text{if } (x,y) = (s_2,s_3) \\ \#0101 & \text{if } (x,y) = (s_2,s_0) \\ \bot & \text{otherwise} \end{cases} \\
\mathcal{AP} &= \{p\} \\
\mathcal{V}(x,p) &= \begin{cases} \top & \text{if } x = s_3 \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

The mvKS in Example 1 is illustrated in the left of Figure 1.

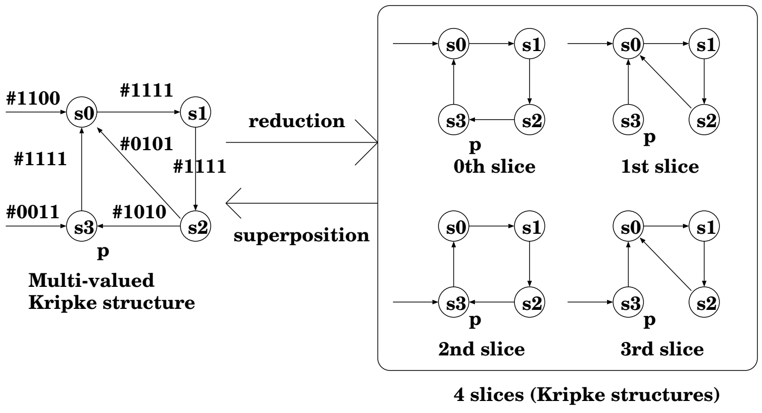

Fig. 1. Multi-Valued Kripke Structure and its Reduction

We can view an mvKS as a superposition of n classical Kripke structures, each of which corresponds to the i-th bit of the lattice. The Kripke structure corresponding to each bit is called a *slice* of the mvKS. In Figure 1, the right figure shows the decomposition (or reduction) of the mvKS into 4 slices. It is apparent that, in the case of order-n Boolean algebra, model checking an mvKS is equivalent to model checking of its n slices (modulo the time and space complexity).

We say an mvKS $\mathcal{M}$ is *total* if $(\bigsqcup_{s \in S} \mathcal{I}(s)) = \top$, and, for all $s \in S$, $(\bigsqcup_{s' \in S} \mathcal{R}(s, s')) = \top$. It is easy to see that an mvKS is total if and only if all its slices are total and have at least one initial state. Following Clarke et al. [7], we assume that every mvKS is total throughout the present paper.

We define a *path* as an infinite sequence of states, namely, a mapping $\pi : \mathbb{N} \to S$ where $\mathbb{N}$ is the set of natural numbers. For a path π, π^j denotes the j-th suffix (path), that is, $\pi^j(i) = \pi(i + j)$ for $i \geq 0$.

2.3 Multi-Valued Linear-time Temporal Logic

We use *Linear-time Temporal Logic* (LTL) as the specification logic with a slight extension to express multi-valuedness over a lattice $\mathcal{L}$. We call this extension mvLTL.

Definition 2 (Formulas of mvLTL) *Let $\mathcal{AP}$ be a set of atomic propositions, and $p \in \mathcal{AP}$ and $\ell \in L$ where $\mathcal{L} = \langle L, \sqcap, \sqcup, \sim \rangle$ is a finite Boolean algebra. An mvLTL formula is defined as follows:*

$$\begin{aligned} \phi, \psi ::= {} & \ell \mid p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \\ & \mid \mathsf{X}\,\phi \mid \mathsf{F}\,\phi \mid \mathsf{G}\,\phi \mid \phi\,\mathsf{U}\,\psi \mid \phi\,\mathsf{R}\,\psi \end{aligned}$$

Definition 3 (Semantics of mvLTL) *Let $\mathcal{M}$ be an mvKS as above, π be a path on $\mathcal{M}$, and ϕ be an mvLTL formula. We define the interpretation of ϕ with respect to π in $\mathcal{M}$, written $(\pi \models \phi)$, as an element of L as follows:*

- $(\pi \models \ell) \stackrel{\text{def}}{=} \ell$ *for* $\ell \in L$.
- $(\pi \models p) \stackrel{\text{def}}{=} \mathcal{V}(\pi(0), p)$ *for* $p \in \mathcal{AP}$.
- $(\pi \models \neg\phi) \stackrel{\text{def}}{=} {\sim}(\pi \models \phi)$.
- $(\pi \models (\phi \wedge \psi)) \stackrel{\text{def}}{=} (\pi \models \phi) \sqcap (\pi \models \psi)$.
- $(\pi \models (\phi \vee \psi)) \stackrel{\text{def}}{=} (\pi \models \phi) \sqcup (\pi \models \psi)$.
- $(\pi \models \mathsf{X}\,\phi) \stackrel{\text{def}}{=} (\pi^1 \models \phi)$.
- $(\pi \models \mathsf{F}\,\phi) \stackrel{\text{def}}{=} \bigsqcup_{i \geq 0}(\pi^i \models \phi)$.
- $(\pi \models \mathsf{G}\,\phi) \stackrel{\text{def}}{=} \bigsqcap_{i \geq 0}(\pi^i \models \phi)$.
- $(\pi \models \phi\,\mathsf{U}\,\psi) \stackrel{\text{def}}{=} \bigsqcup_{i \geq 0}((\pi^i \models \psi) \sqcap (\bigsqcap_{j<i}(\pi^j \models \phi)))$.
- $(\pi \models \phi\,\mathsf{R}\,\psi) \stackrel{\text{def}}{=} \bigsqcap_{i \geq 0}((\pi^i \models \phi) \sqcup (\bigsqcup_{j \leq i}(\pi^j \models \psi)))$.

In the interpretations of temporal operators, we take infinitary meet and join, but they always exist since we are working with finite Boolean algebras.

2.4 Multi-Valued Model Checking Problem

We define the semantics of an mvLTL formula ϕ with respect to $\mathcal{M}$ by:

$$(\mathcal{M} \models \phi) \stackrel{\text{def}}{=} \bigsqcap_{\pi \in \mathbb{N} \to S} ((\sim\mathcal{W}(\pi)) \sqcup (\pi \models \phi))$$

where $\mathcal{W}$ is a mapping from the set of paths to L defined by:

$$\mathcal{W}(\pi) \stackrel{\text{def}}{=} \mathcal{I}(\pi(0)) \sqcap (\bigsqcap_{i \geq 0} \mathcal{R}(\pi(i), \pi(i+1))).$$

$\mathcal{W}(\pi)$ is the weight of the path π, which represents the degree of "being a path in $\mathcal{M}$". In particular, if the first state of the path π is not an initial state ($\mathcal{I}(\pi(0)) = \bot$), then $\mathcal{W}(\pi) = \bot$, hence such a path does not affect the value of $\mathcal{M} \models \phi$.

The *multi-valued model checking problem* is to decide if $(\mathcal{M} \models \phi) = \top$ holds or not. If it holds, ϕ is valid in $\mathcal{M}$.

A *counterexample* of ϕ with respect to $\mathcal{M}$ is a path π such that

$$(\sim\mathcal{W}(\pi)) \sqcup (\pi \models \phi) \neq \top$$

or, equivalently,

$$\mathcal{W}(\pi) \sqcap (\pi \models \neg\phi) \neq \bot.$$

It is easy to see that ϕ is valid *iff* its counterexample does not exist. In the next section, we present an algorithm to find such a counterexample, if it exists.

3 Algorithms of Multi-Valued Bounded Model Checking

We aim to obtain an efficient multi-valued model checker. In the literature, the BDD-based Symbolic Model Checking (SMC) has been extended to the MDD-based one where MDD is a multi-valued extension of BDD [4]. However, as long as the authors know, there has been no attempt to exemplify the multi-valued extension of Bounded Model Checking (BMC), which is the goal of this paper.

3.1 Review of Two-Valued Bounded Model Checking

Figure 2 illustrates the process of classical Bounded Model Checking.

The process can be rephrased in words as follows.

1. Given a Kripke structure, an LTL formula ϕ, and a bound $k > 0$, it generates a propositional formula f (with state variables $x_0, x_1, \dots, x_k$) which expresses a k-bounded model of $\neg\phi$. More precisely, $f(x_0, x_1, \dots, x_k)$ holds if and only if $x_0, x_1, \dots, x_k$ is either a finite path or a "lasso"-shaped looping path such that $\neg\phi$ holds along this path.
2. The formula f is converted to a conjunctive normal form (CNF) since most SAT solvers accept CNF only.

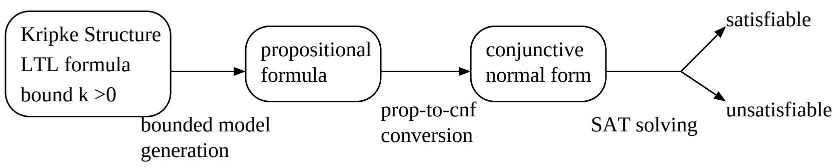

Fig. 2. Process of Bounded Model Checking

3. Finally a SAT solver decides if the CNF is satisfiable or not. If it is satisfiable, there is a counterexample of length k, and otherwise, k is incremented and the same procedure is repeated.

We have to iterate this process only finitely many times, up to the completeness threshold: if there is no counterexample until then, we can conclude that the given specification is verified [1].

3.2 Overview of Multi-Valued Bounded Model Checking

We desire a BMC algorithm for the multi-valued case, but still utilizing state-of-the-art SAT solvers which works for two-valued formulas. Hence, we need to switch from the multi-valued world to the two-valued one at some point in Figure 2. There are three possibilities for this, which are illustrated by Figure 3.

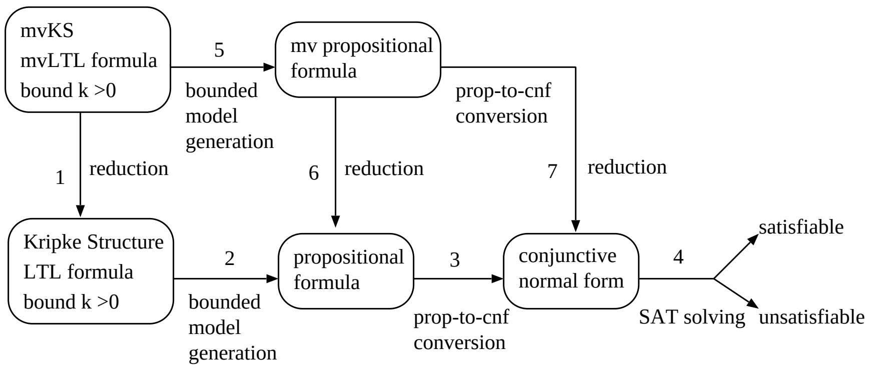

Fig. 3. Process of Multi-Valued Bounded Model Checking

Reflecting the three possibilities, we get three algorithms for Multi-valued Bounded Model Checking:

- *Naive Algorithm* is the route $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$, which first reduces the mvKS and mvLTL formula to two-valued one.

- *Reduction-based Algorithm* is the route $5 \to 6 \to 3 \to 4$, which reduces the output of the mv-formula generated by bounded model generation.
- *Direct Algorithm* is the route $5 \to 7 \to 4$, which keeps the multi-valuedness as far as possible, and directly generates CNF.

Since performance of Naive Algorithm is the worst, we will investigate the latter two where we use or extend the following algorithms and tools that were developed for the two-valued case:

- For bounded model generation, we slightly extend Biere et al's algorithm [1].
- For the conversion from propositional formula to CNF, we extend the algorithm for structure-preserving conversion [12, 1].
- For SAT solving, we use MiniSat solver [9].

4 Bounded Model Generation

The process of multi-valued Bounded Model Generation (Step 5 in Figure 3) is common to Reduction-based and Direct Algorithms. It generates a multi-valued propositional formula which represents a k-bounded model, namely, a counterexample for the given specification of length k.

The algorithm is an extension of the two-valued case [1] and generates a multi-valued propositional formula (mv-propositional formula), which is a propositional formula possibly with lattice elements as propositional constants.

Example 2 *As examples of mv-propositional formula, we define the formulas $\mathcal{I}'$ and $\mathcal{R}'$ which correspond to $\mathcal{I}$ and $\mathcal{R}$ in Example 1.*

$$\begin{aligned}
\mathcal{I}'(x) &\overset{\text{def}}{=} (x = s_0 \land \texttt{\#1100}) \lor (x = s_3 \land \texttt{\#0011}) \\
\mathcal{R}'(x,y) &\overset{\text{def}}{=} (x = s_0 \land y = s_1) \lor (x = s_1 \land y = s_2) \lor (x = s_3 \land y = s_0) \\
&\qquad \lor (x = s_2 \land y = s_3 \land \texttt{\#1010}) \lor (x = s_2 \land y = s_0 \land \texttt{\#0101})
\end{aligned}$$

Definition 4 (Bounded Model Generation) *Let $\mathcal{M}$, ϕ, and k be an mvKS, an mvLTL formula, and a non-negative integer, resp., and $x_0, x_1, \ldots, x_k$ be variables for states. Then we construct a multi-valued propositional formula $[\![\mathcal{M}, \neg\phi]\!]_k$ with free variables $x_0, x_1, \ldots, x_k$ as follows:*

$$\begin{aligned}
[\![\mathcal{M}, \neg\phi]\!]_k &\overset{\text{def}}{=} [\![\mathcal{M}]\!]_k \land [\![\neg\phi]\!]_k, \quad \textit{where} \\
[\![\mathcal{M}]\!]_k &\overset{\text{def}}{=} \mathcal{I}'(x_0) \land \bigwedge_{i=0}^{k-1} \mathcal{R}'(x_i, x_{i+1}) \\
[\![\neg\phi]\!]_k &\overset{\text{def}}{=} \left(\neg\left(\bigvee_{l=0}^{k} \mathcal{R}'(x_l, x_k)\right) \land [\![\neg\phi]\!]_k^0\right) \lor \bigvee_{l=0}^{k} \left(\mathcal{R}'(x_k, x_l) \land {}_l[\![\neg\phi]\!]_k^0\right)
\end{aligned}$$

where $\mathcal{I}'$ and $\mathcal{R}'$ are the mv-propositional formulas representing $\mathcal{I}$ and $\mathcal{R}$ in $\mathcal{M}$, and the mv-formulas $[\![\neg\phi]\!]_k^0$ and ${}_l[\![\neg\phi]\!]_k^0$ are defined in Figure 4.

$$\begin{aligned}
[\![\ell]\!]_k^i &\overset{\text{def}}{=} \ell\\
[\![p]\!]_k^i &\overset{\text{def}}{=} \mathcal{V}'(x_i, p)\\
[\![\neg p]\!]_k^i &\overset{\text{def}}{=} \neg\mathcal{V}'(x_i, p)\\
[\![\phi \wedge \psi]\!]_k^i &\overset{\text{def}}{=} [\![\phi]\!]_k^i \wedge [\![\psi]\!]_k^i\\
[\![\phi \vee \psi]\!]_k^i &\overset{\text{def}}{=} [\![\phi]\!]_k^i \vee [\![\psi]\!]_k^i\\
[\![\mathsf{X}\,\phi]\!]_k^i &\overset{\text{def}}{=} \begin{cases} [\![\phi]\!]_k^{i+1}, & \text{if } i < k\\ \bot, & \text{otherwise}\end{cases}\\
[\![\mathsf{F}\,\phi]\!]_k^i &\overset{\text{def}}{=} \bigvee_{j=i}^{k} [\![\phi]\!]_k^j\\
[\![\mathsf{G}\,\phi]\!]_k^i &\overset{\text{def}}{=} \bot\\
[\![\phi\,\mathsf{U}\,\psi]\!]_k^i &\overset{\text{def}}{=} \bigvee_{j=i}^{k} \left([\![\psi]\!]_k^j \wedge \bigwedge_{n=i}^{j-1} [\![\phi]\!]_k^n\right)\\
[\![\phi\,\mathsf{R}\,\psi]\!]_k^i &\overset{\text{def}}{=} \bigvee_{j=i}^{k} \left([\![\phi]\!]_k^j \wedge \bigwedge_{n=i}^{j} [\![\psi]\!]_k^n\right)
\end{aligned}$$

(a) Translation $[\![\phi]\!]_k^i$ of an mvLTL formula ϕ without a loop.

$$\begin{aligned}
{}_l[\![\ell]\!]_k^i &\overset{\text{def}}{=} \ell\\
{}_l[\![p]\!]_k^i &\overset{\text{def}}{=} \mathcal{V}'(x_i, p)\\
{}_l[\![\neg p]\!]_k^i &\overset{\text{def}}{=} \neg\mathcal{V}'(x_i, p)\\
{}_l[\![\phi \wedge \psi]\!]_k^i &\overset{\text{def}}{=} {}_l[\![\phi]\!]_k^i \wedge {}_l[\![\psi]\!]_k^i\\
{}_l[\![\phi \vee \psi]\!]_k^i &\overset{\text{def}}{=} {}_l[\![\phi]\!]_k^i \vee {}_l[\![\psi]\!]_k^i\\
{}_l[\![\mathsf{X}\,\phi]\!]_k^i &\overset{\text{def}}{=} {}_l[\![\phi]\!]_k^{succ(i)}\\
{}_l[\![\mathsf{F}\,\phi]\!]_k^i &\overset{\text{def}}{=} \bigvee_{j=\min(i,l)}^{k} {}_l[\![\phi]\!]_k^j\\
{}_l[\![\mathsf{G}\,\phi]\!]_k^i &\overset{\text{def}}{=} \bigwedge_{j=\min(i,l)}^{k} {}_l[\![\phi]\!]_k^j\\
{}_l[\![\phi\,\mathsf{U}\,\psi]\!]_k^i &\overset{\text{def}}{=} \bigvee_{j=i}^{k} \left({}_l[\![\psi]\!]_k^j \wedge \bigwedge_{n=i}^{j-1} {}_l[\![\phi]\!]_k^n\right) \vee\\
&\qquad \bigvee_{j=l}^{i-1} \begin{pmatrix} {}_l[\![\psi]\!]_k^j \wedge \bigwedge_{n=i}^{k} {}_l[\![\phi]\!]_k^n \wedge\\ \bigvee_{n=l}^{j-1} {}_l[\![\phi]\!]_k^n \end{pmatrix}\\
{}_l[\![\phi\,\mathsf{R}\,\psi]\!]_k^i &\overset{\text{def}}{=} \bigwedge_{j=\min(i,l)}^{k} {}_l[\![\psi]\!]_k^j \vee\\
&\qquad \bigvee_{j=i}^{k} \left({}_l[\![\phi]\!]_k^j \wedge \bigwedge_{n=i}^{j} {}_l[\![\psi]\!]_k^n\right) \vee\\
&\qquad \bigvee_{j=l}^{i-1} \begin{pmatrix} {}_l[\![\phi]\!]_k^j \wedge \bigwedge_{n=i}^{k} {}_l[\![\psi]\!]_k^n \wedge\\ \bigwedge_{n=l}^{j} {}_l[\![\psi]\!]_k^n \end{pmatrix}
\end{aligned}$$

(b) Translation ${}_l[\![\phi]\!]_k^i$ of an mvLTL formula ϕ for a (l,k)-loop.

Fig. 4. The inductive definition of the translation for an mvLTL formula ϕ in NNF with a bound k and $i \in \mathbb{N}$ with $i \leq k$, where $\mathcal{V}'$ is the formula representing $\mathcal{V}$.

As in the two-valued case, the formula $[\![\mathcal{M}, \neg\phi]\!]_k$ is constructed so that it has a non-$\bot$ value if and only if $x_0, x_1, \ldots, x_k$ form either a finite path along which $\neg\phi$ holds in the bounded semantics, or an (l,k)-loop (the next state of x_k is x_l for some $l \leq k$) along which $\neg\phi$ holds in the standard semantics. Corresponding to the two cases, the mv-formulas $[\![\neg\phi]\!]_k^0$ and ${}_l[\![\neg\phi]\!]_k^0$ represent the semantics of $\neg\phi$ at the state 0. Note that, following Biere et al.[1], Figure 4 defines the translation for ϕ in NNF (Negation Normal Form) only, hence we need to convert $\neg\phi$ to NNF before applying the translation in Figure 4.

The definition above is almost the same as the two-valued one found in the literature[1] except the following points:

– The formulas $\mathcal{I}'$, $\mathcal{R}'$, and $\mathcal{V}'$ are multi-valued propositional formulas, namely, they may contain lattice elements as subformulas.
– However, the atomic formula $x = s_j$ is interpreted by either $\top$ or $\bot$, namely, it essentially remains a two-valued formula.

We say an mv-propositional formula f with free state variables $x_0, \ldots x_k$ is satisfiable if and only if f has a non-$\bot$ value for some assignment of $x_0, \ldots x_k$ to states. Then we have the following theorem.

Theorem 1 (Correctness of Bounded Model Generation) *Let $\mathcal{M}$ be an mvKS and ϕ be an mvLTL formula.*

Soundness *If $[\![\mathcal{M}, \neg\phi]\!]_k$ is satisfiable, then there exists a counterexample of ϕ in $\mathcal{M}$ whose length is k.*
Completeness *If ϕ is not valid in $\mathcal{M}$, then $[\![\mathcal{M}, \neg\phi]\!]_k$ is satisfiable for some k.*

Proof. For each $i < n$ (where n is the order of Boolean algebra), the i-th slice of $[\![\mathcal{M}, \neg\phi]\!]_k$ is identical to the resulting formula of two-valued Bounded Model Generation for the i-th slice of $\mathcal{M}$ and ϕ. Also the totality condition for mvKS implies totality for sliced Kripke structures. Hence the soundness and completeness for the multi-valued version of bounded model generation follows from that for the two-valued version, which was proved, for instance, in [2].

5 Reduction-based Algorithm

Reduction-based Algorithm takes the route $5 \to 6 \to 3 \to 4$ in Figure 3, and we explain Steps 6 and 3 in this section.

Definition 5 (Reduction of mv-Propositional Formula (Step 6)) *Given a multi-valued propositional formula f, we define its i-th slice, $\mathrm{Slice}_i(f)$, as follows:*

$$\mathrm{Slice}_i(\ell) \overset{\text{def}}{=} \begin{cases} \bot & \text{if the } i\text{-th bit of } \ell \text{ is 0} \\ \top & \text{if the } i\text{-th bit of } \ell \text{ is 1} \end{cases}$$

For other cases, $\mathrm{Slice}_i(f)$ is defined homomorphicly, for instance, $\mathrm{Slice}_i(\phi \wedge \psi) = \mathrm{Slice}_i(\phi) \wedge \mathrm{Slice}_i(\psi)$.

For example, for $\mathcal{I}'(x)$ in Example 2, $\mathrm{Slice}_1(\mathcal{I}'(x))$ is $(x = s_0 \wedge \top) \vee (x = s_3 \wedge \bot)$, which is simplified to $(x = s_0)$. Obviously, f is satisfiable (has a non-$\bot$ value) if and only if $\mathrm{Slice}_i(f)$ is satisfiable for some i.

The result of Step 6 is n sliced propositional formulas. We then take their disjunction (since we are interested in satisfiability), and converts it to a CNF. For the Prop-to-CNF Conversion (Step 3), we use the structure-preserving conversion [12, 1].

A few remarks follow.

- All common subformulas will be shared by the conversion, which is especially useful in the context of multi-valued model checking, as we assume that slices of one mvKS are similar to each other.
- However, sharing in Reduction-based Algorithm is not completely satisfactory. Since sharing occurs only among identical subformulas, a small difference in a deeply nested subformula prohibits sharing. Suppose we reduce an mv-formula $(((\texttt{\#10} \wedge a) \vee b) \wedge a) \vee b$ into two slices and convert the disjunction of these slices into CNF. Although the two slices $((a \vee b) \wedge a) \vee b$ and $(b \wedge a) \vee b$ are quite similar to each other, they do not share any subformulas except a and b. In general, if a lattice element exists in a deeply nested subformula of the given mv-formula, sharing does not take place. In such a case, Reduction-based Algorithm cannot generate a small CNF.

6 Direct Algorithm

We explain Step 7 in Figure 3, which is the key step of Direct Algorithm ($5 \to 7 \to 4$). This algorithm is motivated by occasional inefficiency of Reduction-based Algorithm as explained in the previous section. In order to generate as small CNF's as possible, we should share as many subexpressions as possible.

Our idea is simple: rather than *generating* all the sliced formulas, we introduce new propositional variables to *represent* slices, and leave the decision as to which slice should be generated to the SAT solver.

Let us give an example. Example 2 uses the order-4 Boolean algebra, so we introduce two propositional variables q_0 and q_1 to represent each slice number as a binary number. For instance, the 0^{th} slice is $(\neg q_0 \wedge \neg q_1)$, the 1^{st} is $(q_0 \wedge \neg q_1)$ and so on.[2] A lattice element `#1100` has the bit 1 in the 0^{th} and 1^{st} slices, hence it is represented by $(\neg q_0 \wedge \neg q_1) \vee (q_0 \wedge \neg q_1)$, or simply $\neg q_1$. Then an mv-formula $(x = s_0) \wedge$ `#1100` is represented by $(x = s_0) \wedge \neg q_1$. Then the whole formula $\mathcal{I}'(x)$ is represented by $((x = s_0) \wedge \neg q_1) \vee ((x = s_3) \wedge q_1)$. This translation increases the size of the resulting formula, compared to the original mv-formula, much less than the Reduction-based algorithm does.

Definition 6 (Representation of Lattice Values) *For an order-n Boolean algebra $\mathcal{L}$, let $h = \lceil \log_2(n) \rceil$ and $q_0, \ldots, q_{h-1}$ be propositional variables.*

- *For a natural number i such that $0 \leq i < n$, we define $Q_p(i)$ for $0 \leq p < h$ by:*
$$Q_p(i) = \begin{cases} q_p & \text{if the } p\text{-th bit of } i \text{ is 1} \\ \neg q_p & \text{otherwise} \end{cases}$$
and then $R(i)$ is defined as $Q_0(i) \wedge \cdots \wedge Q_{h-1}(i)$. Note that $R(i)$ is the binary representation of i in terms of $q_0, \ldots, q_{h-1}$. For instance, if $h = 5$, then $R(6)$ is $\neg q_0 \wedge q_1 \wedge q_2 \wedge \neg q_3 \wedge \neg q_4$, which is 00110 as a binary number (note that q_0 corresponds to the least significant bit).
- *For an element ℓ of $\mathcal{L}$, we define $\mathrm{Rep}(\ell) = \bigvee_{i \in One(\ell)} R(i)$ where $One(\ell) = \{i \mid \ell\text{'s } i\text{-th bit is } 1\}$.*

Although the above representation is not quite efficient, we can make use of the Karnaugh map technique to simplify it. Note also that, we need only $\lceil \log_2(n) \rceil$ propositional variables to represent slices in the order-n Boolean algebra.

Definition 7 (Direct Algorithm (Step 7)) *Let f be a multi-valued propositional formula over the order-n Boolean algebra, and $h = \lceil \log_2(n) \rceil$.*

1. *Generate h propositional variables $q_0, \ldots, q_{h-1}$.*
2. *Replace any lattice element ℓ in f by $\mathrm{Rep}(\ell)$.*
3. *If $n < 2^h$, let f' be $f \wedge \bigwedge_{n \leq i < 2^h} \neg R(i)$. If $n = 2^h$, let f' be f.*
4. *Apply the structure-preserving conversion [12, 1] to f'.*

[2] Here we assume q_0 corresponds to the least significant bit.

The third step is necessary to exclude spurious slices.

The algorithm above is guaranteed to be correct. For simplicity we assume that $n = 2^h$ for some natural number h.

Theorem 2 (Correctness of Direct Algorithm) *Let f be an mv-propositional formula over the order-n Boolean algebra with $n = 2^h$. If the algorithm for Step 7 generates a CNF c with $q_0, \ldots, q_{h-1}$, then f is satisfiable (in the multi-valued sense) if and only if c is satisfiable for some truth-value assignment for $q_0, \ldots, q_{h-1}$.*

Proof. We have that f is satisfied if and only if its i-th sliced formula is satisfied for some i. Since each slice is represented by a truth-value assignment of q_j's, this is equivalent to the satisfiability of the resulting CNF for some assignment of q_j's.

7 Experimental Results

We have implemented Reduction-based and Direct Algorithms for mvMC, and compared their performance[3]. Our implementation choices are listed below.

– For Reduction-based Algorithm, we compose (fuse) Step 6 and Step 3 into a single recursive function to gain better performance. At the same time, the function simplifies the formula using $\top \wedge f \leftrightarrow f$ etc., and this choice forces us to use the bottom-up variant of the prop-to-CNF conversion.
– For Direct Algorithm, we implement the top-down algorithm in Section 6. Note that not only CNF's but also the representation of lattice elements are cached and shared, hence they are processed only once.
– For SAT solving, we use MiniSat solver version 1.14 using the DIMACS format [11] as its input.

We have implemented these algorithms in the programming language OCaml, and executed them on a machine with 1.0GB memory and Intel Celeron (2.8 GHz) processor running Linux Operating System.

For the target of this experiment, we take a simple mvKS with m states, and the order of Boolean algebra is n. The first model, Model-1, is illustrated

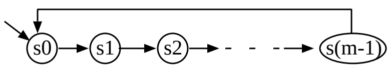

Fig. 5. Model-1

in Figure 5, which shows its initial state (s_0 only) and the transition relation.

[3] In fact, we have implemented many other variants, but here we list only the results of the two most efficient algorithms.

Model-1				CNF gen.		SAT solving		CNF size			
	k	n	m	red.	direct	red.	direct	one bit	red.	direct	result
	14	32	16	2	0	0.0	0.0	5470	11590	6331	unsat.
	15	32	16	2	0	0.0	0.0	5889	12417	6801	sat.
	30	64	32	32	0	0.5	0.1	23276	72380	26569	unsat.
	31	64	32	40	0	0.2	0.0	24129	74817	27521	sat.
	62	128	64	683	7	5.4	0.6	95818	488938	108615	unsat.
	63	128	64	715	8	1.0	0.1	97537	496897	110529	sat.

Table 1. Experimental Results for Model-1.

Model-2				CNF gen.		SAT solving		CNF size			
	k	n	m	red.	direct	red.	direct	one bit	red.	direct	result
	10	128	64	66	1	4.4	8.2	30826	99466	33483	unsat.
	12	128	64	90	2	9.7	11.1	36718	117838	39765	unsat.
	14	128	64	113	2	12.5	14.0	42634	136234	46071	unsat.
	16	128	64	145	2	25.4	12.2	48574	154654	52401	unsat.
	18	128	64	184	2	63.7	22.9	54538	173098	58755	unsat.
	20	128	64	226	2	94.4	28.6	60526	191566	65133	unsat.
	22	128	64	281	3	42.8	14.0	66538	210058	71535	sat.
	24	128	64	329	3	60.9	13.8	72574	228574	77961	sat.

Table 2. Experimental Results for Model-2.

Transitions are essentially two-valued (with the truth values $\top$ or $\bot$ only) so we do not write the lattice elements as their values. The valuation function only is multi-valued:

$$\mathcal{V}(s_i, p) = \texttt{\#0}\cdots\texttt{010}\cdots\texttt{0}$$

with the i-th bit being 1. Then $\mathsf{F}\,p$ is valid in this model if $m \geq n$, and not valid otherwise, in which case there exists a counterexample of length $k = m - 1$. The other two models, Model-2 and Model-3, are small variants of Model-1: in Model-2, $\mathcal{R}(s_i, s_j) = \top$ if and only if $i < j \leq ((i + 3) \bmod m)$, and in Model-3, $\mathcal{R}(s_i, s_j) = \top$ if and only if $i < j \leq ((i + 8) \bmod m)$. In addition, the valuation function in Model-3 is modified to: $\mathcal{V}(s_i, p) = \texttt{\#0}\cdots\texttt{01}\cdots\texttt{10}\cdots\texttt{0}$ with the i-th to $i + 8$-th bits being 1.

Tables 1, 2 and 3 show several experiments for these models. In the table, "red." means Reduction-based Algorithm and "direct" Direct one. The column "CNF gen." shows the execution time (in seconds) before SAT solving (Steps 6 and 3 for Reduction-based one and Step 7 for Direct one), and the column "SAT solving" shows the execution time (in seconds) for SAT solving for their output CNF's. The column "CNF size" shows the number of clauses in the generated CNF[4] where the column "one bit" shows the CNF size of one slice. The column

[4] The number of propositional variables and the overall size of CNF are almost linear in the number of clauses in all cases.

Model-3				CNF gen.		SAT solving		CNF size			
	k	n	m	red.	direct	red.	direct	one bit	red.	direct	result
	10	32	128	95	5	20.5	32.1	142808	138977	138320	unsat.
	12	32	128	124	5	25.9	46.9	164664	169968	165423	unsat.
	14	32	128	157	7	35	81.7	191032	197152	191893	unsat.
	16	32	128	204	9	22.9	0.2	217424	224360	218387	sat.
	18	32	128	289	13	24.1	29.8	243840	251592	244905	sat.

Table 3. Experimental Results for Model-3.

"result" shows the result of SAT solving where "sat." means the algorithm finds a counterexample for the given bound k, the model with m states and the order-n lattice. It should be noted that our implementations show the correct answers for all cases.

We can understand the results as follows.

- Efficient implementation of mv-BMC is possible. Compared with the one-bit case, both algorithms can generate CNF's whose size is much smaller than n times of the CNF size of one-bit case. For instance, the bottom line of the first table shows that Direct Algorithm generated a CNF with 110,529 clauses which is only 13% larger than that generated by the one-bit slice (note that $n = 128$ for this experiment).
- For the comparison between Reduction-based and Direct Algorithms, the latter is better in CNF-generation: its execution time is always much faster (and the difference becomes huge when the order n becomes larger), and the size of generated CNF is smaller or roughly the same.
- The execution time of the SAT solver for the generated CNF is not easily compared between the two methods. Reduction-based one is usually better for smaller n's despite the fact that it generates a bigger CNF. However, the difference is not very big, and Direct method is sometimes better.
- Since CNF generation phase takes longer time than SAT solving, the total execution of Direct Algorithm is better than that of Reduction-based one. Since our experiments are for relatively small models, and our implementation may be suboptimal, we cannot generalize this statement at the moment. However, these results are surely encouraging.

8 Conclusion and Future Directions

We have explored the possibility to obtain an efficient Multi-valued Bounded Model Checker, and for this purpose, we have formalized Reduction-based and Direct Algorithms with correctness guarantee. We have implemented these algorithms (together with other ones) and successfully shown that Multi-Valued Bounded Model Checking is surely possible. Also, we have compared their performance: to the extent we have tested, Direct Algorithm seems to be better than Reduction-based one, but this last point should be examined further.

For future work, we need to introduce into our framework various optimizations found in the context of two-valued BMC. We have already started in investigating some of such optimizations, which are easily integrated to our algorithms. Another obvious thing to do is to extend the lattice to more general one such as Quasi-Boolean algebras, which seems not too difficult, after Chechik and others' work [5]. Experiments with larger, more realistic Kripke structures and finding good application areas are also important.

Acknowledgments. We would like to thank the anonymous referees for insightful comments. The authors were supported in part by JSPS Grant-in-Aid for Scientific Research 20650003.

References

1. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207, 1999.
2. A. Biere, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic Model Checking Using SAT Procedures Instead of BDDs. In *Design Automation Conference*, pages 317–320, 1999.
3. M. Chechik, B. Devereaux, and S. Easterbrook. Implementing a multi-valued symbolic model checker. In *TACAS'01*, volume 2031 of *Lecture Notes in Computer Science*, pages 404–419. Springer, 2001.
4. M. Chechik, B. Devereaux, S. Easterbrook, and A. Gurfinkel. Multi-valued symbolic model-checking. *ACM Transaction on Software Engineering and Methodology*, 2(4):371–408, 2003.
5. M. Chechik, B. Devereaux, S. Easterbrook, Y. C. Lai, and V. Petrovykh. Efficient multiple-valued model-checking using lattice representations. *Lecture Notes in Computer Science*, 2154:441–455, 2001.
6. M. Chechik, A. Gurfinkel, B. Devereaux, A. Lai, and S. Easterbrook. Data structures for symbolic multi-valued model-checking. *Form. Methods Syst. Des.*, 29(3):295–344, 2006.
7. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, 1999.
8. S. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *International Conference on Software Engineering*, pages 411–420, 2001.
9. N. Een and N. Sorensen. The MiniSat homepage, `http://minisat.se/`.
10. M. C. Fitting. Many-valued modal logics. *Fundamenta Informaticae*, XV:235–254, 1991.
11. D. S. Johnson and M. A. Trick, editors. *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series In Discrete Mathematics and Theoretical Computer Science.* AMS, 1996.
12. D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.