PAPER *Special Section on Formal Approach*

Efficient Multi-Valued Bounded Model Checking for LTL over Quasi-Boolean Algebras*

Jefferson O. ANDRADE[†a)] ***and*** **Yukiyoshi KAMEYAMA**[††b)], ***Members***

SUMMARY Multi-valued Model Checking extends classical, two-valued model checking to multi-valued logic such as Quasi-Boolean logic. The added expressivity is useful in dealing with such concepts as incompleteness and uncertainty in target systems, while it comes with the cost of time and space. Chechik and others proposed an efficient reduction from multi-valued model checking problems to two-valued ones, but to the authors' knowledge, no study was done for multi-valued bounded model checking. In this paper, we propose a novel, efficient algorithm for multi-valued bounded model checking. A notable feature of our algorithm is that it is not based on reduction of multi-values into two-values; instead, it generates a single formula which represents multi-valuedness by a suitable encoding, and asks a standard SAT solver to check its satisfiability. Our experimental results show a significant improvement in the number of variables and clauses and also in execution time compared with the reduction-based one.

key words: *multi-valued model checking, bounded model checking, quasi-boolean logic*

1. Introduction

Model Checking is an automatic technique for verifying temporal properties of finite transition systems [2]. Multi-valued model checking extends it by using multi-valued logic [3], [4] instead of two-valued one. Multi-valuedness gives us a more natural way to express such concepts as incompleteness, uncertainty, authenticity, capability, and many others, and has been proved useful in various fields of verification.

This extra expressivity, however, comes at the cost of performance and/or space, and many researchers have investigated this problem. Among others, Chechik et al. [5] proposed (1) an embedding of Quasi-Boolean algebra (Quasi-Boolean logic) into Boolean algebra, and (2) Multi-valued Decision Diagrams (MDD) as an extension of Binary Decision Diagrams (BDD) [6]. By combining them, they obtained an efficient symbolic model checker for Quasi-Boolean logics. Nakajima [7] applied their technique to verify a non-trivial real-world software specification.

This paper investigates *Bounded Model Checking* (BMC) [8] for multi-valued logic. In particular, we propose a novel algorithm for multi-valued BMC problems where both models and specifications are defined over Quasi-Boolean algebras.

The fundamental problem to build such an algorithm is that the standard reduction method is quite inefficient, where reduction means a way to represent a multi-valued formula by many two-valued ones (which are called slices). Intead of generating many slices, our algorithm generates a single two-valued formula that represents these slices with clever encoding. Based on our prototype implemenetation, our algorithm shows a significant improvement in the number of variables and clauses and also in runtime compared with reduction-based one.

The rest of this paper is organized as follows. Section 2 introduces a motivating example, and Sect. 3 introduces basic concepts. Section 4 describes our algorithms, and Sect. 5 shows our experiments with a prototype implementation. Finally, Sect. 6 provides our conclusion.

2. Example

As a motivating example, we introduce a simple message relay system inspired by Nakajima's example [7]. Figure 1 shows our model for the system, where IDLE etc. are states, received=F etc. are the values of the variables in each state. The model is similar to a Kripke structure, however

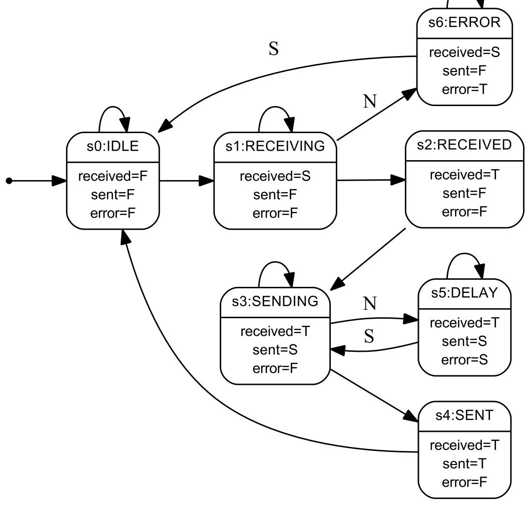


Fig. 1 A multi-valued model for a simple message relay system.

Manuscript received July 18, 2011.
Manuscript revised November 13, 2011.

[†]The author is with the Department of Informatics, Campus Serra, Federal Institute of Espírito Santo, Brazil.
[††]The author is with the Department of Computer Science, University of Tsukuba, Tsukuba-shi, 305–8573 Japan.

[*]This paper is a revised and extended version of the paper presented at International Symposium on Automated Technology for Verification and Analysis, October 20–23, 2008, Seoul, Korea [1].

a) E-mail: joandrade@ifes.edu.br
b) E-mail: kameyama@acm.org

DOI: 10.1587/transinf.E95.D.1355

some transitions and values are annotated with an element of Quasi-Boolean algebra such as T (true), S (should), N (should not), and F (false). Unannotated transitions are assumed to have T as its annotation.

Let us explain the model. Initially, the system is IDLE. It can remain IDLE or transit to RECEIVING. After the message has been RECEIVED, the system starts SENDING. After transmission is completed, the system returns to IDLE. The system *should not* present any DELAY on the transmission, but it is possible (not prohibited) that DELAY occurs in exceptional situations. If there is a delay, the system *should* eventually resume transmission, but we cannot say it *must* resume transmission, since the environment may not allow it. The system *should not* lose the received data, but it is not impossible that an ERROR occurs. After data loss, the system *should* return to IDLE.

We want to specify the following properties on this system using Linear-time Temporal Logic (LTL):

1. All messages received must be sent eventually.
   $\mathsf{G}(\text{received} \to \mathsf{F}\,\text{sent})$
2. The system must always resume from an error.
   $\mathsf{G}(\text{error} \to \mathsf{F}(\neg\text{error} \wedge \neg\text{received} \wedge \neg\text{sent}))$
3. The system should not begin a transmission unless a message has been received.
   $\mathsf{G}((\neg\text{received} \wedge \neg\text{sent}) \vee \mathsf{F}(\text{received}\ \mathsf{R}\ \neg\text{sent}))$

Note that the truth values of the formulas above may be N or S, and therefore they are actually formulas of multi-valued logic[†].

The method in this paper allows one to verify these properties even if the model is expressed over a Quasi-Boolean Algebra.

3. Basic Definitions and Results

Classical model checking algorithms are defined for Kripke structures and specifications written as temporal logic formulas. In multi-valued model checking, Kripke structures and temporal logic are extended so that the domain of truth values is a multi-valued one. Fitting [3], [4] showed that many of the desired properties in multi-valued logic are obtained if the truth values form a complete lattice. In addition, to preserve the relationship between logic operators and their meaning, we require the domain to satisfy the distributivity laws (for conjunction and disjunction) and De Morgan's laws (for complement). Boolean Algebra satisfies these conditions, but they are too restrictive for practical purposes. Quasi-Boolean Algebra is a minimal structure that satisfies these conditions, and is widely used in various areas which need multi-valued logic.

In this section, we review its definition and application to multi-valued model checking.

3.1 Quasi-Boolean Algebra

A *lattice* $\mathcal{L} = \langle L, \sqsubseteq \rangle$ is a partially ordered set in which any two elements x and y have a least upper bound (join, denoted by $x \sqcup y$), and a greatest lower bound (meet, denoted by $x \sqcap y$). In this paper, we consider only finite lattices, namely, the set L of elements should be finite. A finite lattice has the greatest element (denoted by $\top$) and the least element (denoted by $\bot$).

Alternatively, a lattice may be defined as an algebraic structure with the underlying set L and operations $\sqcap$ and $\sqcup$, i.e., $\mathcal{L} = \langle L, \sqcap, \sqcup \rangle$.

Definition 1 (Distributive Lattices). *We say a lattice $\mathcal{L} = \langle L, \sqsubseteq \rangle$ is a* distributive lattice *iff the following properties hold for all $x, y, z \in L$.*

$$\begin{array}{ll} x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z) & \multirow{2}{*}{} \\ x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z) & \textit{Distributivity} \end{array}$$

It is easy to see that every finite distributive lattice is a *complete lattice*, that is, the least upper bound and the greatest lower bound for any set of elements exist.

Definition 2 (Quasi-Boolean Algebra). *Given a finite distributive lattice $\mathcal{L} = \langle L, \sqcap, \sqcup \rangle$,* Quasi-Boolean algebra *(QBA) is a tuple $Q_L = \langle L, \sqcap, \sqcup, \neg \rangle$, where $\neg$ is a unary operator called* quasi-boolean complement *for which the following properties hold for all $x, y \in L$.*

$$\begin{array}{rl} \neg(x \sqcap y) = \neg x \sqcup \neg y & \\ \neg(x \sqcup y) = \neg x \sqcap \neg y & \textit{De Morgan laws} \\ \neg\neg a = a & \textit{Involution} \\ x \sqsubseteq y \textit{ iff } \neg y \sqsubseteq \neg x & \textit{Anti-monotonicity} \end{array}$$

Figure 2 shows some QBA of practical interest. Figure 2 (a) shows a 3-valued logic with truth values F (false), U (unknown) and T (true), where the negation of U is U itself. Figure 2 (b) extends it to $n+1$ values with a total order. Figures 2 (c) and 2 (d) are QBAs obtained by the products $\mathcal{L}_2 \times \mathcal{L}_2$ and $\mathcal{L}_3 \times \mathcal{L}_3$, resp. Figure 2 (e) shows a QBA used in the example of the previous section. It has six truth values T, F, N ("should not"), DK ("don't know"), DC ("don't care"), and S ("should"). It is interesting to note that negation of S is N, while negation of DK is DK itself.

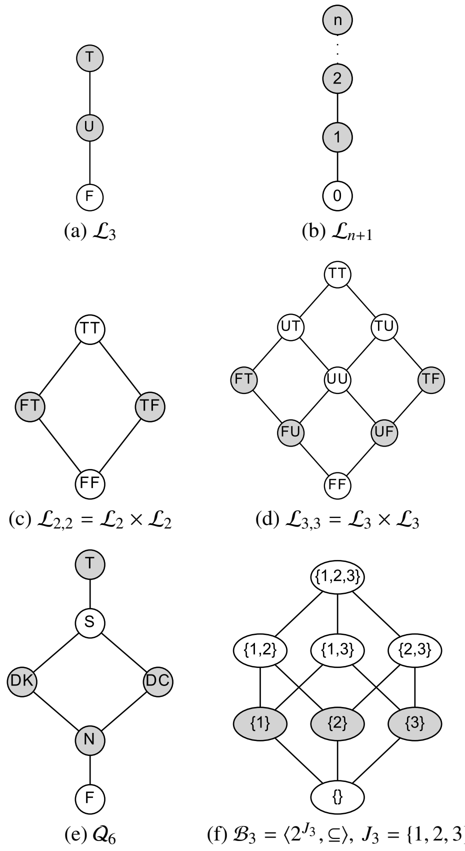
All Boolean Algebras (BA) are QBAs. Figure 2 (c) and Fig. 2 (f) are Boolean algebras with 2^2 elements, and 2^3 elements, resp, called order-2 and order-3 Boolean Algebras.

Chechik et al. [5] embeds QBAs into BAs using the notion of join-irreducible elements defined below.

Definition 3 (Join-Irreducible Element). *Let $Q = \langle L, \sqcap, \sqcup, \neg \rangle$ be a QBA. An element j of L is* join-irreducible *iff j is not the least element $\bot$, and for any $a, b \in L$, $a \sqcup b = j$ implies $a = j$ or $b = j$.*

Definition 4 (Embedding of QBA). *Let $Q = \langle L, \sqcap, \sqcup, \neg \rangle$ be a QBA, and $\mathcal{J}_Q$ be the set of join-irreducible elements in Q. The embedding* $\mathsf{em} : L \to 2^{\mathcal{J}_Q}$ *is defined by* $\mathsf{em}(\ell) = \{j \in \mathcal{J}_Q \mid j \sqsubseteq \ell\}$.

[†]The multi-valued logic used here will be later shown in Fig. 2 (e).



(a) $\mathcal{L}_3$ (b) $\mathcal{L}_{n+1}$ (c) $\mathcal{L}_{2,2} = \mathcal{L}_2 \times \mathcal{L}_2$ (d) $\mathcal{L}_{3,3} = \mathcal{L}_3 \times \mathcal{L}_3$ (e) $\mathcal{Q}_6$ (f) $\mathcal{B}_3 = \langle 2^{J_3}, \subseteq \rangle$, $J_3 = \{1, 2, 3\}$

Fig. 2 Examples of lattices of practical interest. The gray nodes denote *join-irreducible elements* of each lattice.

For instance, in the 3-valued lattice $\mathcal{L}_3$, em maps F, U, and T, to $\{\}$ $\{\mathsf{U}\}$, and $\{\mathsf{U}, \mathsf{T}\}$, resp.

The embedding em is injective, and preserves the greatest and least elements, meet, and join where we regard $2^{\mathcal{J}_Q}$ as a Boolean Algebra ordered by set inclusion. The embedding is not necessarily surjective, and may not preserve negation. For $\mathcal{L}_3$, negation of U is U, but the complement of $\{\mathsf{U}\}$ in $2^{\mathcal{J}_Q}$ is $\{\top\}$, which is not in the image of em.

The embedding gives us a way of representing an element of QBA as a bit sequence. Let $j_1, \ldots, j_n$ be an enumeration of $\mathcal{J}_Q$ (we fix the order of these elements.) We represent $\ell \in L$ by a bit-sequence of length n, where the ith bit of the bit-sequence is 1 iff $j_i \sqsubseteq \ell$, for $1 \leq i \leq n$. The bit-sequence is written as, for instance, $\sharp 110$.

3.2 Multi-Valued Model Checking

To perform multi-valued model checking (mvMC) based on QBA, we need to extend Kripke structures and LTL to their multi-valued counterparts.

Let $Q = \langle L, \sqcap, \sqcup, \neg \rangle$ be a QBA throughout this subsection. *Multi-valued Kripke structures* [3] extend 2-valued one as follows.

Definition 5 (Multi-Valued Kripke Structure)**.** *A* multi-valued Kripke structure *(*mv*KS) over Q is a tuple $\mathcal{M} = \langle S, \mathcal{I}, \mathcal{R}, \mathcal{AP}, \mathcal{V} \rangle$ such that:*

- *S is a finite set of states.*
- *$\mathcal{I} : S \to L$ gives the initial states.*
- *$\mathcal{R} : S \times S \to L$ gives the transition relation.*
- *$\mathcal{AP}$ is a finite set of atomic propositions.*
- *$\mathcal{V} : S \times \mathcal{AP} \to L$ is the valuation function for an atom at a state.*

Note that the codomain of $\mathcal{I}$, $\mathcal{R}$ and $\mathcal{V}$ is L by which multi-valuedness may be introduced. For instance, $\mathcal{I}(s_0) = \mathsf{U}$ means that it is undefined (unspecified) that s_0 is an initial state or not. We say an *mv*KS $\mathcal{M}$ is *total* if $(\bigsqcup_{s \in S} \mathcal{I}(s)) = \top$, and, for all $s \in S$, $(\bigsqcup_{s' \in S} \mathcal{R}(s, s')) = \top$. Following [2], [8], [9], we assume that every *mv*KS is total throughout the present paper.

Definition 6 (Path)**.** *A* path *over a* mv*KS is a mapping $\pi : \mathbb{N} \to S$ where $\mathbb{N}$ is the set of natural numbers. The suffix π^j of a path π denotes the path such that $\pi^j(i) = \pi(i + j)$ for $i \geq 0$.*

To express specifications of systems, we use *mv*LTL, a multi-valued extension of *Linear-time Temporal Logic*, which was first introduced in [10].

Definition 7 (Formulas of *mv*LTL)**.** *Given a* mv*KS $\mathcal{M}$ based on a QBA Q, a formula in* mv*LTL is defined as follows:*

$$\phi, \psi ::= \ell \mid p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \mathsf{X}\,\phi \mid \mathsf{F}\,\phi \mid \phi\,\mathsf{U}\,\psi$$

where $\ell \in L$ (an element of QBA), and $p \in \mathcal{AP}$ (an atomic proposition in $\mathcal{M}$).

We may introduce other temporal operators such as G and R using X, F, and U.

Definition 8 (Semantics of *mv*LTL)**.** *Let $\mathcal{M}$ be a mvKS over Q as before, π be a path on $\mathcal{M}$, and ϕ be a* mv*LTL formula. We define the interpretation of ϕ w.r.t. π in $\mathcal{M}$ (written $\pi \models \phi$) as an element of Q as follows:*

$$\begin{array}{lcl}
\pi \models \ell & \overset{\text{def}}{=} & \ell \textit{ for } \ell \in L \\
\pi \models p & \overset{\text{def}}{=} & \mathcal{V}(\pi(0), p) \textit{ for } p \in \mathcal{AP} \\
\pi \models \neg\phi & \overset{\text{def}}{=} & \neg(\pi \models \phi) \\
\pi \models \phi \wedge \psi & \overset{\text{def}}{=} & (\pi \models \phi) \sqcap (\pi \models \psi) \\
\pi \models \phi \vee \psi & \overset{\text{def}}{=} & (\pi \models \phi) \sqcup (\pi \models \psi) \\
\pi \models \mathsf{X}\,\phi & \overset{\text{def}}{=} & \pi^1 \models \phi \\
\pi \models \mathsf{F}\,\phi & \overset{\text{def}}{=} & (\pi \models \phi) \sqcup (\pi^1 \models \mathsf{F}\,\phi) \\
\pi \models \phi\,\mathsf{U}\,\psi & \overset{\text{def}}{=} & (\pi \models \psi) \sqcup \big((\pi \models \phi) \sqcap (\pi^1 \models \phi\,\mathsf{U}\,\psi)\big)
\end{array}$$

Note that the interpretation does not necessarily give a boolean value to each *mv*LTL formula. Rather, it gives a truth value as an element of Q, as indicated by the use of $\sqcap$ and $\sqcup$ in the right-hand side of the definition.

The interpretation above is a straightforward extension of the standard, boolean semantics of LTL formulas except one point: the "definition" for F and U may be circular when π^1 is identical to π. This circularity can be easily avoided by using the least fixpoint[†], since the right-hand sides of the "definitions" are monotone, and the least fixpoints exist for any set of elements, as our domain is a complete lattice.

A counter-intuitive fact about the QBA-based multi-valued logic is that logical equivalence is not the same as equality. As an example, in the three-valued logic $\mathcal{L}_3$, the value of $\mathsf{U} \leftrightarrow \mathsf{U}$ (under the standard definition of $\leftrightarrow$ in terms of $\sqcup$ etc.) is not T, but U.

We define the semantics of a *mv*LTL formula ϕ w.r.t. $\mathcal{M}$ by:

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

where $\mathcal{W}$ is defined by:

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

$\mathcal{W}(\pi)$ is the "weight" of the path π, which represents the degree of "being a path in $\mathcal{M}$". In the definition, we explicitly take the greatest lower bound.

Finally we state the Multi-Valued Model Checking problem.

Definition 9 (*mv*MC problem)**.** *The multi-valued model checking problem is to decide if* $(\mathcal{M} \models \phi) = \top$ *holds. If it holds, we say that* ϕ *is valid in* $\mathcal{M}$.

It immediately follows that, to check the validity of ϕ, we only have to check if there exists a path π (a *counterexample* of ϕ) such that the value of $(\neg \mathcal{W}(\pi)) \sqcup (\pi \models \phi)$ is not $\top$. By taking negation of both sides, we check if $\mathcal{W}(\pi) \sqcap (\pi \models \neg\phi)$ is not $\bot$. The last check can be done using the bit-sequence representation of QBA elements explained earlier; we only have to check if there is 1 in the bit sequence, which can be represented by disjunction of all the bits.

In the next section, we present an algorithm to perform such a check in the context of bounded model checking.

3.3 Generalized Queries

In some applications, we need to check, for $\ell \in \mathcal{L}$, a more general form $(\mathcal{M} \models \phi) \sqsupseteq \ell$. For instance, the specification $\mathsf{G}(\text{received} \to \mathsf{F}\,\text{sent})$ of the example in Sect. 2 does not always hold since the transmission may be delayed infinitely in some exceptional case. However, the specification *should* hold for normal cases, and we would like, then, to check if:

$$(\mathcal{M} \models \mathsf{G}(\text{received} \to \mathsf{F}\,\text{sent})) \sqsupseteq \mathsf{S}$$

We can cope with the generalized form as follows. A counterexample of the query $(\mathcal{M} \models \phi) \sqsupseteq \ell$ is a path π such that

$$(\mathcal{W}(\pi) \sqcap (\pi \models \neg\phi)) \not\sqsubseteq \neg\ell$$

It is easy to check if a certain value v is not equal to or smaller than $\neg\ell$. For instance, suppose the underlying QBA is embedded into order-5 BA (with 2^5 elements), and $\neg\ell$ is mapped to the bit sequence $\sharp 10100$ by this embedding. To check $v \not\sqsubseteq \neg\ell$, we only have to check if at least one of the 2nd, 4th and 5th bits of v is 1, which can be represented by disjunction.

In summary, we can cope with the generalized queries with no extra cost, and hence we will ignore the generalized queries in the subsequent sections.

4. Algorithms

This section discusses several possible algorithms for multi-valued bounded model checking, and present our algorithm.

4.1 Review of Classical Bounded Model Checking

Figure 3 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 $x_0, x_1, \ldots, x_k$) which expresses a k-bounded model of $\neg\phi$. More precisely, $f(x_0, x_1, \ldots, x_k)$ holds if and only if $x_0, x_1, \ldots, x_k$ is either a finite path or a 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.
3. Finally a SAT solver decides if the CNF is satisfiable or not. If it is satisfiable, there is a counterexample of length k. 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 [11].

4.2 Overview of Multi-Valued BMC

The goal of the algorithms is to encode a multi-valued bounded model checking problem as a boolean satisfiability problem, which can be solved by the state-of-the-art

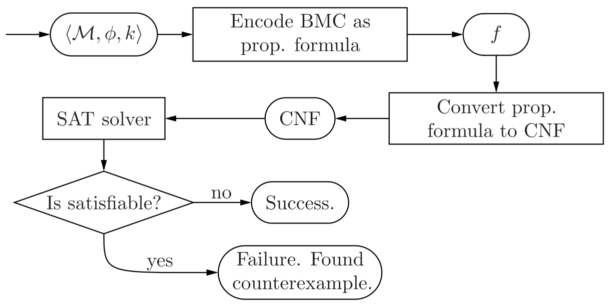

⟨M, φ, k⟩
Encode BMC as prop. formula
f
Convert prop. formula to CNF
CNF
SAT solver
Is satisfiable?
no
Success.
yes
Failure. Found counterexample.

Fig. 3 Diagram for classical Bounded Model Checking.

[†]If we directly define the semantics of G, we should use the greatest fixpoint.

SAT solver. We must convert multi-valued entities (models and formulas) to two-valued ones at some point of our algorithm, and the embedding of QBA into BA in the previous section makes such a conversion possible. The question is when we convert multi-valued ones to two-valued ones, and we have several possibilities as follows. (They are illustrated in Fig. 4.)

1. *Naïve algorithm* — route $1 \to 3 \to 6 \to 7$ — This method converts the *mv*KS and *mv*LTL specification to n 2-valued ones, and then solves the corresponding n instances of 2-valued BMC problems separately. This method is the easiest way, but is applicable only when the QBA is a BA, in which case each bit in the bit sequence has no dependency.
2. *Reduction-based algorithm* — route $2 \to 4 \to 6 \to 7$ — This method encodes the *mv*BMC problem as a multi-valued propositional formula θ and then reduces the formula θ to n 2-valued propositional formulas representing each bit of θ. We then take the conjunction of the n 2*v*-propositional formulas as the translation of the original *mv*BMC model and proceed with the BMC procedure as we would do in classical BMC. This approach was investigated for multi-valued boolean logic in [10] where we reported results of some experiments.
3. *Direct algorithm* — route $2 \to 5 \to 7$ — This method is similar to the reduction-based one, except that it does not actually convert (or reduce) QBA to BA, which takes time and space. Rather, we generate a single formula which *represents* the conversion, and the size of the generated formula has comparative length as the formula for each bit, rather than the conjunction of all such formulas.

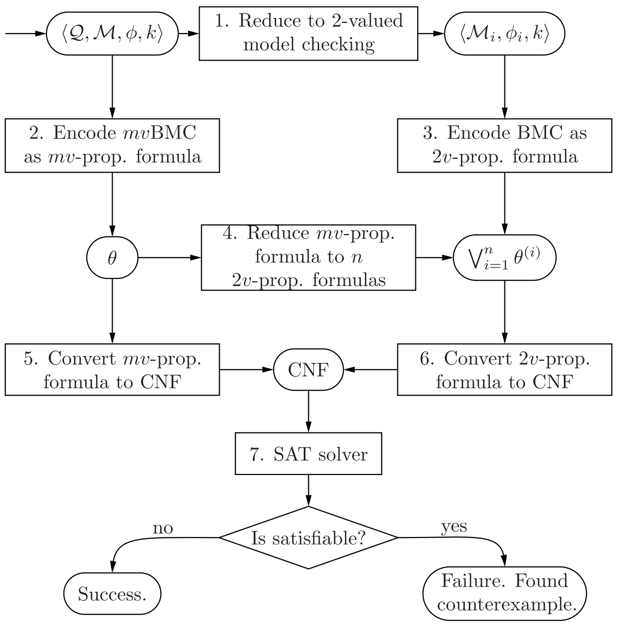

$\langle Q, \mathcal{M}, \phi, k\rangle$
1. Reduce to 2-valued model checking
$\langle \mathcal{M}_i, \phi_i, k\rangle$
2. Encode mvBMC as mv-prop. formula
3. Encode BMC as $2v$-prop. formula
θ
4. Reduce mv-prop. formula to n $2v$-prop. formulas
$\bigvee_{i=1}^n \theta^{(i)}$
5. Convert mv-prop. formula to CNF
CNF
6. Convert $2v$-prop. formula to CNF
7. SAT solver
Is satisfiable?
no
yes
Success.
Failure. Found counterexample.

Fig. 4 The possible paths to the *mv*BMC depicted as an extended flowchart diagram. Q is a QBA, $\mathcal{M}$ is the given model, ϕ is the *mv*LTL specification to verify and k is the bound. The numbers in the processes are for identification purpose only.

There is yet another possible algorithm; we translate the encoded *mv*BMC problem (i.e. θ) to a multi-valued clause form, *mv*CNF, and then pass it to a multi-valued SAT solver such as CAMA [12] and CMV-SAT-1 [13]. However, these multi-valued SAT solvers rely on the assumption that the set of logic values is totally ordered, which is not generally the case for QBA. Therefore, we do not consider this algorithm in this paper.

Naïve conversion was studied in the context of multi-valued CTL symbolic model checking [10], [14], but it is much less efficient than the others, and therefore we concentrate on the reduction-based and direct algorithms.

We note that while the embedding of QBA to BA does not preserve negation, it is not a problem in our development, since we first convert, using De Morgan laws, every multi-valued formulas to the formulas in Negation Normal Form (NNF), in which negation may appear only in front of atomic propositions. This point will be explained later.

Both algorithms for *mv*BMC, reduction-based and direct, share the first step, step 2 in Fig. 4, also referenced as *bounded model generation*. Our algorithm for bounded model generation is an extension of the 2-valued case [8] and is the same one that we previously published for BA [1].

To illustrate the general idea of the *bounded model generation* algorithm we present a simple example of its application below.

Example 1. *Let $\mathcal{M}$ be the model presented in Fig. 1, over the logic Q_6, the bound $k = 2$ and also let the negation of the specification, already in NNF, be $\psi = \mathsf{F}sent$. We represent the transition relation of the model $\mathcal{M}$ by $\mathcal{R}(x_i, x_j)$*[†]*. So, following the definitions given in [1], the bounded model generation outputs the following multi-valued formula:*

$$[\![\mathcal{M}, \psi]\!]_2 = [\![\mathcal{M}]\!]_2 \wedge [\![\mathsf{F}sent]\!]_2 \tag{1}$$

$$\begin{aligned}[\![\mathcal{M}]\!]_2 &= \mathcal{I}'(x_0) \wedge \bigwedge_{i=0}^{1} \mathcal{R}'(x_i, x_{i+1}) \\ &= \mathcal{I}'(x_0) \wedge \mathcal{R}'(x_0, x_1) \wedge \mathcal{R}'(x_1, x_2) \end{aligned} \tag{2}$$

$$\begin{aligned}[\![\mathsf{F}sent]\!]_2 &= \left(\neg\left(\bigvee_{l=0}^{2} \mathcal{R}'(x_2, x_l)\right) \wedge [\![\mathsf{F}sent]\!]_2^0\right) \\ &\quad \vee \left(\bigvee_{l=0}^{2} \left(\mathcal{R}'(x_2, x_l) \wedge {}_l[\![\mathsf{F}sent]\!]_2^0\right)\right) \\ &= \left(\neg\left(\bigvee_{l=0}^{2} \mathcal{R}'(x_2, x_l)\right) \wedge \left(\bigvee_{j=0}^{2} sent_j\right)\right) \\ &\quad \vee \left(\bigvee_{l=0}^{2} \left(\mathcal{R}'(x_2, x_l) \wedge \bigvee_{j=0}^{2} sent_j\right)\right) \end{aligned} \tag{3}$$

In the above set of equations, $\mathcal{I}'$ and $\mathcal{R}'$ are mv-propositional formulas that encode the set of initial states

[†]Due to space constraints, we do not unroll the transition relation.

and the transition relation of model $\mathcal{M}$ resp. Equation (1) *is the general form of the encoding for the bounded model checking problem. Equation* (2) *is the encoding for the model and its transition relation. Equation* (3) *is the encoding for the given specification. We used $sent_j$ as a shorthand for $\mathcal{V}'(x_j, sent)$, i.e., the valuation of the atomic proposition sent in the state x_j.*

We have not specified how we represent $\mathcal{I}'$, $\mathcal{R}'$, and $\mathcal{V}'$, which depends on how we encode state variables $x_0, x_1, \cdots, x_k$. For this, we take a simple method that encodes each state by a binary encoding as follows: let m be the smallest integer not smaller than $\log_2 |S|$ (where $|S|$ is the number of states). We introduce two-valued propositional variables $y_0, y_1, \cdots, y_{m-1}$, and represent each state by conjunction of positive or negative literals for these variables. For instance, the state 3 is represented by $y_0 \wedge y_1 \wedge \neg y_2 \wedge \cdots \wedge \neg y_{m-1}$. There are more efficient ways for this encoding in the case of the two-valued BMC, for instance, representing a state by a conjunction of atomic propositions. However, it may not work for some multi-valued models, so we take this simple encoding. In summary, the formula output by BMC generation contains two-valued propositional variables $y_0, y_1, \cdots, y_{m-1}$, which we also call state variables.

4.3 Reduction-Based Algorithm

The Reduction-based Algorithm takes the route $2 \to 4 \to 6 \to 7$ in Fig. 4 and we explain steps 4 and 6 in this section.

Algorithm 1 shows a top-level description of the reduction-based *mv*BMC procedure. The call to function boundedModel, generates a multi-valued propositional formula which expresses a bounded model of length k. It slightly extends the classical algorithm in the literature [8] for two-valued bounded model checking. The resulting formula $[\![\mathcal{M}, \neg\phi]\!]_k$ contains state variables $x_0, x_1, \ldots, x_k$, and expresses that $x_0, x_1, \ldots, x_k$ forms a counterexample of the given specification. We have proved that our *mv*BMC algorithm is correct for an arbitrary finite Boolean algebra [10].

Algorithm 1 Reduction-based *mv*BMC

```
1:  function redMvBmc($\mathcal{Q}$,$\mathcal{M}$,$\phi$,$k$)
2:      $k' \leftarrow 0$
3:      while $k' < k$ do
4:          $k' \leftarrow k' + 1$
5:          $\psi \leftarrow$ nnf($\neg\phi$)
6:          $\theta \leftarrow$ boundedModel($\mathcal{Q}, \mathcal{M}, \psi, k'$)
7:          $n \leftarrow |\mathcal{J}_\mathcal{Q}|$
8:          $\theta' \leftarrow \bigvee_{i=0}^{n} \theta^{(i)}$        ▹ $\theta'$ is a 2v-prop. formula.
9:          cnf $\leftarrow$ boolToCnf($\theta'$)
10:         $S \leftarrow$ satSolver(cnf)
11:         if $S \neq \emptyset$ then        ▹ Found a counterexample.
12:             return ⟨false, $S$⟩
13:         end if
14:     end while
15:     return ⟨true, $\emptyset$⟩        ▹ Property $\phi$ holds.
16: end function
```

4.3.1 Reduce *mv*-Prop Formula to n 2*v*-Prop Formulas (Step 4)

Line 8 is central in Algorithm 1, since it is where the reduction from *mv*-propositional to 2*v*-propositional is performed. Each formula $\theta^{(i)}$ represents one bit (or one layer) of the original *mv*-formula, i.e., the *mv*-formula θ being reduced with respect to a single join-irreducible element of $\mathcal{Q}$.

The process of obtaining $\theta^{(i)}$ is not straightforward as in the BA case [10], since the embedding does not preserve negation. Table 1 shows an example for the negation in QBA $\mathcal{L}_3$.

Since the variables in the formula we are treating are two-valued variables only, we can take each slice (each bit) of the formula as follows.

Definition 10 (*mv*-Propositional Slicing)**.** *Given an mv-propositional formula θ over a QBA $\mathcal{Q}$, we define the 2-valued propositional formula $\theta^{(i)}$ inductively as follows:*

$$\textit{For literals:}\ \begin{cases} l^{(i)} & \overset{\text{def}}{=} & b_i(l) \\ (\neg l)^{(i)} & \overset{\text{def}}{=} & b_i(\neg l) \\ y^{(i)} & \overset{\text{def}}{=} & y \\ (\neg y)^{(i)} & \overset{\text{def}}{=} & \neg y \end{cases}$$

$$\textit{For composite formulas:}\ \begin{cases} (\circ\,\phi)^{(i)} & \overset{\text{def}}{=} & \circ(\phi^{(i)}) \\ (\phi\ \circ\ \psi)^{(i)} & \overset{\text{def}}{=} & \phi^{(i)} \circ \psi^{(i)} \end{cases}$$

where y is a two-valued propositional variable (state variable), $b_i : L \to \{0, 1\}$ is a function that maps a lattice value to its "bit" of order i, assuming a binary encoding for lattice values. The symbol $\circ$ is used as a placeholder for logical operators such as conjunction.

4.3.2 Convert 2*v*-Prop Formula to CNF (Step 6)

The call to boolToCnf in Algorithm 1 represents the conversion from 2*v*-propositional to CNF. In order to generate as small CNFs as possible, we apply the structure-preserving CNF conversion [15] to share as many subformulas as possible.

Although sharing subformulas is performed in the last step, the space-complexity of Reduction-based Algorithm is not very good, since we generate n formulas corresponding to n bits (which again corresponds to each join-irreducible elements), and take the disjunction of them.

4.3.3 Correctness of Reduction-Based Algorithm

Theorem 1. *The reduction-based algorithm is sound and complete, namely, given a mvMC problem $\langle \mathcal{Q}, \mathcal{M}, \phi \rangle$ and*

Table 1 Bitmap representation of lattice values for QBA $\mathcal{L}_3$ and their respective negations.

Lattice Value	Repr.	Negation	Neg. Repr.
F	♯00	T	♯11
U	♯10	U	♯10
T	♯11	F	♯00

a bound k, the algorithm generates a counterexample of length up to k if and only if $\mathcal{M} \models \phi$ *has a counterexample of length up to k.*

Proof. The proof of this theorem consists of verification of each part of the algorithm. Each step is rather straightforward and can be proved by simple reasoning, and thus omitted.

□

It is easy to see that, even for Multi-Valued Model Checking, there is a completeness threshold c in the sense of [11], for instance, the "diameter" of *mv*KS can be such a threshold. It follows that, by repeating the reduction-based algorithm for $k = 1, 2, \cdots, c$, we can verify the given specification.

4.4 Direct Algorithm

We now come to Direct Algorithm which we think is the best among the algorithms. It takes route $2 \to 5 \to 7$. In this section we cover step 5.

The Direct Algorithm has been developed based on Reduction-based Algorithm with one additional idea: rather than *generating* all the sliced formulas, we introduce propositional variables to *represent* each slice (bit), and leave the decision to the SAT solver, as to which slice must actually be generated. Besides this point, Direct Algorithm is the same as Reduction-based one; we use $\mathsf{neg}_Q(p, i)$ for negation of atomic propositions, and we share as many sub-expressions as possible during the CNF generation.

To understand the key idea, let us first see an example.

Example 2. *Consider algebra* $\mathcal{B}_3$ *(Fig. 2 (f)). We introduce two propositional variables* q_0 *and* q_1 *to represent each slice number as a binary number. Actually we only have 3 slices (one for each join-irreducible element of* $\mathcal{B}_3$*), so we will not use all possible combinations of* q_0 *and* q_1*. For instance, the 0th slice is* $(\neg q_0 \wedge \neg q_1)$*, the 1st is* $(q_0 \wedge \neg q_1)$ *and so on*[†]*. A lattice element* $\sharp 110$ *has the bit 1 in the 0th and 1st 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* $(x = s_0) \wedge \sharp 110$ *is represented by* $(x = s_0) \wedge \neg q_1$*. This translation increases the size of the resulting formula, compared to the original mv-formula, much less than the Reduction-based algorithm does.*

Note that the CNF generation is done without any reduction from *mv*-propositional formulas to 2-valued ones.

4.4.1 Convert *mv*-Prop Formula to CNF (Step 5)

A high level description of our approach to *mv*BMC is presented in Algorithm 2. This algorithm differs from the reduction-based one in the call to function mvPropToCnf, which converts a multi-valued propositional formula to a 2-valued CNF, which is key to the algorithm's efficiency and is described in Definition 12.

Algorithm 2 Direct Algorithm for *mv*BMC

```
 1: function mvbmc($Q$,$\mathcal{M}$,$\phi$,$k$)
 2:     $k' \leftarrow 0$
 3:     while $k' < k$ do
 4:         $k' \leftarrow k' + 1$
 5:         $\psi \leftarrow$ nnf($\neg\phi$)
 6:         $\theta \leftarrow$ boundedModel($Q, \mathcal{M}, \psi, k'$)
 7:         cnf $\leftarrow$ mvPropToCnf($Q, \theta$)
 8:         $S \leftarrow$ satSolver(cnf)
 9:         if $S \neq \emptyset$ then                ▹ Found a counterexample.
10:             return $\langle$false$, S\rangle$
11:         end if
12:     end while
13:     return $\langle$true$, \emptyset\rangle$              ▹ Property $\phi$ holds.
14: end function
```

Definition 11 (Representation of Lattice Values)**.** *For a QBA* Q *where* $n = |\mathcal{J}_Q|$*, 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:*

$$Q_p(i) = \begin{cases} q_p & \text{if the } p^{th} \text{ bit of } i \text{ is } 1 \\ \neg q_p & \text{otherwise} \end{cases}$$

  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.*
- *For an element* ℓ *of* Q*, we define*

$$\mathrm{Rep}(\ell) = \bigvee_{i \in \mathrm{Ones}(\ell)} R(i)$$

  where $\mathrm{Ones}(\ell) = \{i \mid \ell\text{'s } i^{th} \text{ bit is } 1\}$.

It is possible to use circuit minimization techniques to simplify the above representation[††]. Note also that, we need only $\lceil \log_2(n) \rceil$ propositional variables to represent the bits, where $n = |\mathcal{J}_Q|$.

Definition 12 (Direct CNF Conversion)**.** *Let f be a multi-valued propositional formula over a QBA* Q*, and* $h = \lceil \log_2(n) \rceil$*, where* $n = |\mathcal{J}_Q|$*.*

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. (exclude spurious bits)*
4. *Apply the structure-preserving conversion to* f'*.*

4.4.2 Correctness of Direct Algorithm

The algorithm above is guaranteed to be correct [10]. For simplicity we assume that $n = 2^h$ for some natural number h where $n = |\mathcal{J}_Q|$.

[†] q_0 corresponds to the least significant bit.

[††] Our prototype implements the Quine-McCluskey minimization algorithm.

Theorem 2. *The direct algorithm is also sound and complete.*

Proof. We can prove this theorem by reducing it to the previous theorem as follows: Suppose Direct Algorithm generates the formula ψ in CNF. Then, by definition, for each i less than n, the formula:

$$\psi \wedge Q_0(i) \wedge Q_1(i) \wedge \cdots \wedge Q_{h-1}(i)$$

is equivalent to the i-th sliced formula generated by Reduction-based Algorithm. Hence, for some truth-assignment for $q_0, q_1, \cdots, q_{h-1}$, ψ is satisfiable iff the disjunction of all the sliced formulas generated by Reduction-based Algorithm is satisfiable. It follows that, soundness and completeness of Reduction-based Algorithm imply those of Direct Algorithm.

□

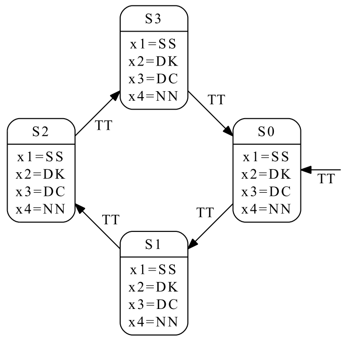
Since Direct Algorithm is designed as a refinement of Reduction-based Algorithm, we can naturally expect the former's performance is better than the latter's. We will confirm this observation in the next section.

5. Experiments and Discussion

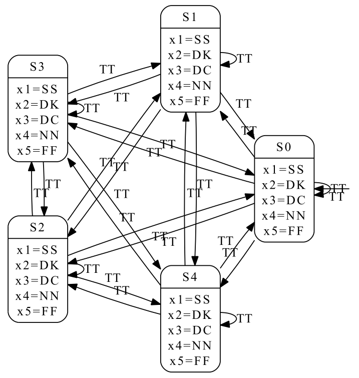
We have built a prototype that implements both the *reduction-based algorithm* and the *direct algorithm* in Sect. 4. We implemented the prototype in the Scala programming language over the Java Virtual Machine version 1.6. Algorithm 2 shows the direct *mv*BMC algorithm, and Algorithm 1 shows the reduction-based version. Note that not only CNFs but also the representation of lattice elements are cached and shared, hence they are processed only once. For SAT solving, we used SAT4J [16] as an embedded SAT solver. We have executed the tests on a machine with 4.0 GiB of RAM and an Intel Core i5 M430 (2.27 GHz) processor running Ubuntu Linux 10.04 operating system.

The models used in our experiments belong to four different groups.

1. Systems modeled after "real world" examples, taken from the literature:
   a. The coffee machine system (CMS) [17].
   b. The message relay system (MRS) [7].
2. Systems that form a simple loop, that we called L_n systems. All L_n systems have exactly n states and n variables on each state. Figure 5 (a) shows the graphic representation for the L_4 system. Transitions are 2-valued, with the truth values $\top$ or $\bot$ only. The valuation function is given by $\mathcal{V}(s_i, x_j) = \text{tord}(Q, i + j)$, where tord is a selection function that obeys an arbitrary total order over the elements of Q. Then $\mathsf{F}\, x_i$ is valid in this model if $|Q| \geq n$, and not valid otherwise, in which case there exists a counterexample of length $k \leq |Q| - 1$.
3. System that form a completed connected graph, i.e., where we have transitions from each state to all the other states in the system, that we called T_n systems. All T_n systems have exactly n variables on every state and the weight of every transition is assigned according to an arbitrary order from the QB algebra over which the system is defined. Figure 5 (b) shows the T_5 system.



(a) Loop model L_4.



(b) Total model T_5.

Fig. 5 Example of the artificial test models used to collect statistics of the prototype.

The reason we introduced the artificial systems, L_n and T_n, is that we needed models that could scale on a parameter. As can be seem by their description the two artificial models we presented are scalable in the parameter n what makes possible to obtain the statistical data we needed.

We have run tests with both algorithms for a number of L_n and T_n instances. Of course, due to space limitations we can not show all these experimental results here, but we selected one instance of each of these systems to illustrate their behavior. Since we want to compare the two real world systems, CMS and MRS, with the two artificial ones we choose instances L_7 and T_7 since their size is approximately equal to the size of the coffee machine and message relay systems.

Figure 6 shows the evolution of the four selected systems number of CNF variables as a function of k for both algorithms, and Fig. 7 shows the number of CNF clauses also as a function of k. As can be seen, for all systems the growth of the curves for the direct algorithm is much slower than the growth for the reduction-based algorithm, with a much more accentuated reduction for the number of CNF clauses. We call attention to the fact that even if the reduction in the number of variables is approximately linear, the absolute value of this reduction can be significant, with a maximum of more than 10^4 variables for model MRS in the

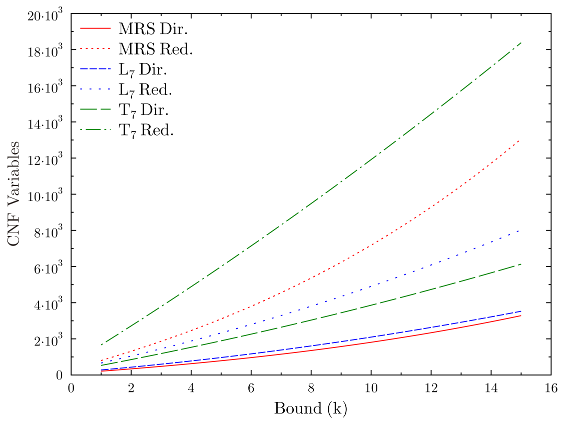


Fig. 6 Average number of CNF variables.

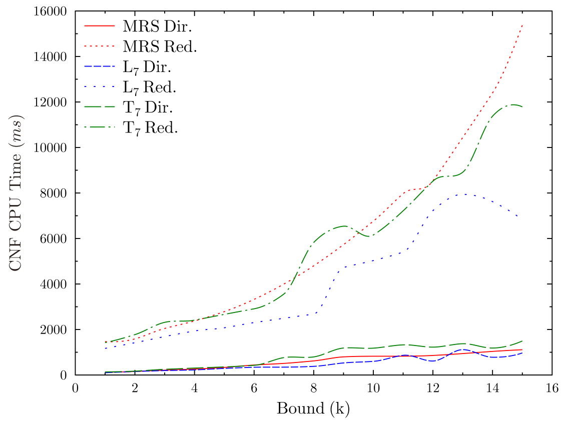


Fig. 8 Average *mv*-prop→CNF conversion runtime.

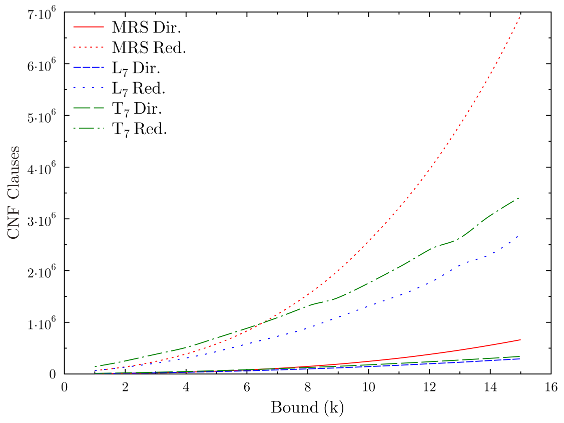


Fig. 7 Average number of CNF clauses.

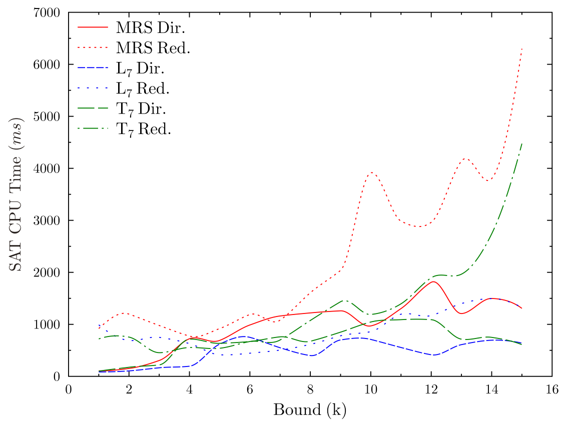


Fig. 9 Average SAT solver runtime.

data shown. Even more impressive was the reduction in the number of CNF clauses. Where the maximum difference between the reduction-based and the direct algorithm reached about 6.5 million clauses.

Maybe of greater interest is the behavior of both algorithms regarding their running time. For that matter, since the bounded model generation phase is common to both algorithms, we concentrated on the CNF conversion time. Figure 8 show the CNF conversion CPU time of the example model for both algorithms. As can be seen, the runtime evolution it is much less consistent than that of the number of CNF variables or clauses, but it is still clearly visible that the direct algorithm has a much lower runtime cost than the reduction-based one. Possible explanations for the inconsistency of the evolution of the CNF conversion runtime are the fact that the number of shared sub-expressions is not linear in k and the action of the garbage collector of the Java Virtual Machine which we could not isolate during the collection of data.

Figure 9 shows the SAT solver CPU running time for the selected models. As we observed for CNF conversion, the SAT solver running time does not express a consistent behavior when compared to the number of CNF variables or CNF clauses. This is a known issue and we do not elaborate on the topic. What we do want to call attention for is the fact that, regardless of the inconsistency of the SAT solver runtime evolution, in all tested models, the running time for the direct version of was much smaller than that for the reduction based version.

Since our experiments are for relatively small models and our implementation may be sub-optimal we cannot generalize this results at the moment. But we consider these results very encouraging.

6. Conclusion

In this article we presented an extension of classical LTL model-checking to reasoning with quasi-boolean algebras and described the notion of multi-valued Kripke structures. We also presented a translation of the LTL *mv*BMC problem over quasi-boolean algebras to a SAT problem. We then presented two algorithms for LTL *mv*BMC based on this translation, namely the reduction-based algorithm and the direct algorithm.

As part of our experiments we built a prototype that implements both algorithms, and our experiments confirm our claims that for the direct algorithm the number of CNF variables and clauses generated by the translation is linear in the size of the problem instance. We also observed a significant reduction in the runtime for the CNF conversion phase in favor of the direct algorithm. This behavior is much more efficient than the one presented by the reduction-based algorithm in both runtime and space.

As for continuing our research, we expect the method described here to be useful for dealing with model abstraction. Also in this line of research, we have already started some theoretical work on the generalization of queries for multi-valued bounded model checking. We foresee that this generalization technique will also be useful in the context of model abstraction, so we surely hope to conduct investigations in this area. One other path of future work is to investigate the impact of optimization techniques known for classical bounded model checking for the multi-valued case.

References

[1] J.O. Andrade and Y. Kameyama, "A direct algorithm for multi-valued bounded model checking," ATVA 2008: 6th International Symposium on Automated Technology for Verification and Analysis, Lect. Notes Comput. Sci., pp.80–94, Springer-Verlag, 2008.

[2] E.M. Clarke, O. Grumberg, and D.A. Peled, Model Checking, The MIT Press, 1999.

[3] M.C. Fitting, "Many-valued modal logics," Fundamenta Informaticae, vol.XV, pp.235–254, 1991.

[4] M.C. Fitting, "Many-valued modal logics II," Proc. LFCS'92, Springer-Verlag, 1992.

[5] M. Chechik, B. Devereaux, S. Easterbrook, Y.C. Lai, and V. Petrovykh, "Efficient multiple-valued model-checking using lattice representations," Lect. Notes Comput. Sci., vol.2154, pp.441–455, 2001.

[6] D.M. Miller, "Multiple-valued logic design tools," Proc. 23rd International Symposium on Multiple-Valued Logic, pp.2–11, Sacramento, CA, USA, IEEE Computer Society, May 1993.

[7] S. Nakajima, "Behavioural analysis of component framework with multi-valued transition system," APSEC '02, p.217, Washington, DC, USA, IEEE Computer Society, 2002.

[8] A. Biere, E.M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," Design Automation Conference, pp.317–320, 1999.

[9] M. Chechik, B. Devereaux, S. Easterbrook, and A. Gurfinkel, "Multi-valued symbolic model-checking," ACM Trans. Softw. Eng. Methodol., vol.2, no.4, pp.371–408, 2003.

[10] J.O. Andrade and Y. Kameyama, "An algorithm for bounded multi-valued model checking," Proc. 4th Symposium on Science and Technology for System Verification (DSW 2007), pp.11–20, Nagoya, Japan, Japan Society for Software Science and Technology, 2007.

[11] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," TACAS'99, Lect. Notes Comput. Sci., vol.1579, pp.193–207, 1999.

[12] C. Liu, A. Kuehlmann, and M.W. Moskewicz, "CAMA: A multi-valued satisfiability solver," Proc. 2003 IEEE/ACM International Conference on Computer-Aided Design, ICCAD '03, pp.326–333, Washington, DC, USA, IEEE Computer Society, 2003.

[13] S. Jain, E. O'Mahony, and M. Sellmann, "A complete multi-valued SAT solver," Proc. 16th International Conference on Principles and Practice of Constraint Programming, CP'10, pp.281–296, Berlin, Heidelberg, Springer-Verlag, 2010.

[14] A. Gurfinkel and M. Chechik, "Multi-valued model checking via classical model checking," CONCUR 2003 – Concurrency Theory, 14th International Conference, ed. R.M. Armadio and D. Lugiez, Lect. Notes Comput. Sci., vol.2761, pp.263–277, Marseille, France, Springer, Sept. 2003.

[15] D. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," J. Symbolic Computation, vol.2, pp.293–304, 1986.

[16] D. Le Berre and A. Parrain, "The SAT4J library, release 2.2," JSAT Journal on Satisfiability, Boolean Modeling and Computation, vol.7, pp.59–64, 2010.

[17] M. Chechik, B. Devereaux, and S. Easterbrook, "Implementing a multi-valued symbolic model checker," TACAS'01, Lect. Notes Comput. Sci., vol.2031, pp.404–419, Springer, 2001.

Jefferson O. Andrade received the Diploma of Engineer in Computer Engineering and Master of Informatics degrees from the Federal University of Espirito Santo, Brazil, in 1995 and 2001, resp. He is currently pursuing the Ph.D. in Engineering at the Graduate School of Systems and Information Engineering in the University of Tsukuba. He worked with software development companies in Brazil from 1995 to 2003 and he is a lecturer at the Federal Institute of Education, Science and Technology of Espirito Santo since 2003. His research interests include formal methods in software engineering, verification and validation of software and logical methods in artificial intelligence. He is a member of ACM and SBC.

Yukiyoshi Kameyama received the B.Sc. and M.Sc. degrees from the University of Tokyo in 1985 and 1987, resp, and the Ph.D. degree from Kyoto University in 1996. He worked as a faculty at Tohoku University and Kyoto University from 1987 to 2001, and he is a professor at the University of Tsukuba. His research interests include programming logic and software verification. He is a member of ACM, JSSST and IPSJ.