Regular Paper

Optimizing mv-BMC to mv-SAT Conversion

JEFFERSON OLIVEIRA ANDRADE[†1]

Classical, 2-valued logics don't express naturally systems with uncertainty or inconsistency. It has been shown that multi-valued logics, specified over finite lattices, can be efficiently used to model and reason about such systems.[7),8)] For this date, all known approaches to solve the multi-valued symbolic model checking problem use multi-valued decision diagrams (MDD) as a symbolic representation of the model. As for BDDs, MDDs presents a space blow up bottleneck. In this report we describe our results on a technique for using propositional SAT solvers instead of MDD symbolic manipulation procedures, i.e. to use of *bounded model checking* (BMC), for solving the *multi-valued model checking* (MVMC) problem. By using BMC, our procedure aims to avoid the space blow up of MDD, generates counterexamples faster and generates counterexamples of minimal length.

This work extends and improves our previous work developed for the *Advanced Systems Development Research Project* in 2006. A number of additions and improvements are described. We present the semantics of a multi-valued extension of LTL as well as an extended notion of counterexample for the MVMC case and a procedure to reduce an MVMC problem to a propositional satisfiability problem. Finaly we presente preliminary results from our multi-valued BMC prototype implementation.

1. Introduction

It is not uncommon, during software specification, to find contradictory specifications or uncertainty in requirements. Although most of the current specification techniques and tools lack the proper treatment of these problems, some work has been made to address this problem. In the context of automated system verification, the work of Chechik[7),8),11)] and Easterbrook[14)] is distinguished for addressing the problem by employing model checking[13)] over multi-valued logics[16),17)] to reason about distinct viewpoints. Also, the work of Kameyama, Nishizawa and others[23),26)] suggests a direction for employing multi-valued logics to analyse models in an even early stage of the development.

To our knowledge, the most extensive study of applying multi-valued logics to the model checking problem is the work of Chechik, Easterbrook, Gurfinken and others[7),18)]. That work provided strong theoretical results on applying *multi-valued decision diagrams* (MDD) to the on multi-valued symbolic model checking problem and, lead to the development of a multi-valued symbolic model checker, the χChek tool[10)]. On the present work we propose to apply *bounded model checking* (BMC)[1),2)] to the multi-valued model checking problem. In the classical, 2-valued logic context, BMC has been shown to be more efficient on finding counterexamples, generate counter-examples of minimal length and sometimes speed up the verification, when compared to model checking using BDD symbolic manipulation procedures. We expect these features to be preserved in the multi-valued logic context as well.

During the execution of the previous *Advanced Systems Development Research Project* course our goal was to develop a tools capable of creating models that incorporate uncertainty and disagreement, and that provide the means to reason about these models. Since this is not done naturally using 2-valued logics, we turned to the use of *multi-valued logics*[16),17)]. Applying *model checking* for reasoning about the models was a natural development.

On the current project we introduced a number of improvements and expansions over the previous one. The most significant are:

- We developed an original translation from the *multi-valued model checking problem* (MVMC) to *multi-valued propositional satisfiability* (mv-SAT) and from that to two-valued propositional satisfiability (SAT).
- We have changed the underlying logic on which we describe the models and specifications from *Computational Temporal Logic (CTL)* to *Linear-time Temporal Logic (LTL)*. That allowed the development of a much more solid theoretical foundation.
- We refined the specification language making the syntax and semantics much more intuitive and easy to use. The new language also made much easier to employ symbolic model check-

†1 Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba, joandrade@logic.cs.tsukuba.ac.jp

ing optimizations during the "compilation" of the model.
- The prototype was completely rebuild. For the new implementation we have swapped from the object oriented paradigm to the functional one. Using the language F# we have achieved a high performance implementation, but still preserving the expressiveness of a functional language and keeping focus on the algorithms.
- We implemented a package for storing LTL and *finite domain propositional formulas (FDPL)* as *directed acyclic graphs (DAG)*. This data structure made possible to eliminate redundant sub-formulas on construction time with a very small time overhead and a huge (more than 2 orders of magnitude) improvement on memory consumption.

The rest of this report is organized as follows. Section 2 describes further motivations for this work. Section 3 introduces multi-valued Kripke structures (MVKS) and the semantics of mv-LTL as well as multi-valued model checking. Section 4 describes the nave approach to solving the multi-valued model checking problem. It is also in this section that we describe the direct translation from multi-valued model checking problem to multi-valued propositional satisfiability. Section 5 gives an overview of the new (rebuild from scratch) version of our prototype implementation for the multi-valued bounded model checker, as well as some description of the algorithms used. Section 6 introduces a motivational example and gives some results from preliminary experiments with a prototype implementation of our direct translation method. Section 7 discusses other attempts to solve the multi-valued model checking problem found in the literature and discuss our conclusions and directions for future work.

2. Motivation

Large software systems development is still a highly error prone process. According to NIST, errors in software systems cost about 60 billions of dollars every year, only in United States.[25] These errors are due, basically, to two causes:

(1) *The system is not correct*, i.e., the system implementation does not satisfy the specification.

(2) *The system is not adequate*, i.e., the requirements had not been correctly understood and/or represented by the software engineer. One can say that the model created by the engineer incorrectly reflects the system.

This has leaded to a substantial grow in the interest in formal methods in the last few years. Formal methods is a collection of mathematical techniques for specifying and verifying complex hardware and software systems.

As said, a very common problem on specifying software systems is the fact that usually we find uncertainty or disagreement about the requirements; or even worse, we find requirements that are contradictory. It would surely be very useful for a software engineer in this situation to be able to include this uncertainty on the system specification and also create contradictory models and have automatic or semi-automatic ways to refine these contradictory models.

We believe that a tool that allows the creation of models that incorporate uncertainty and disagreement, and also allows the verification of these models against requirements will be extremely useful by the average software engineer.

3. Multi-Valued Model Checking

Model checking can be summarized as an automated technique to verify temporal properties on finite systems.[⋆1] Standard model-checking usually receives as input a system's model described by a Kripke structure (requirements) and a temporal logic description of the system's properties (specifications). The specification is verified against the Kripke structure to see if it holds or not. If the specification does not hold, a counterexample is produced.

On the other hand, *multi-valued model checking* receives as input a *multi-valued Kripke structure* (MVKS), i.e. a multi-valued finite transition system as an extension of the Kripke structure, and the description of the system's properties. Then the specification is verified against MVKS. On this work, from the range of the algebra of truth values and temporal logics available, we will focus on Linear-time Temporal Logic (LTL).

3.1 Boolean Algebra

In this paper, we take Boolean algebras as the many-valued structure used as logical domains of interpretation for formulas in multi-valued logics. We do this in order to focus on algorithmic and implementation issues. Extension to more general structures are left for future work.

Definition 3.1 (Lattice)**.** A *lattice* is a partially ordered set $\mathcal{L} = (L, \sqsubseteq)$ such that, for any two elements $x, y \in L$, the following elements exist:

- Their greatest lower bound $(x \sqcap y)$, called *meet*.

⋆1 There is an increasing interest in model checking for infinite systems, though.

- Their least upper bound $(x \sqcup y)$, called *join*.

A lattice is called *distributive* if it satisfies the distributive laws $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$ and $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$.

Definition 3.2 (Relative Pseudo-Complement)**.** Let $\mathcal{L} = (L, \sqsubseteq)$ be a lattice. If an element $c \in L$ is the greatest member of $\mathcal{L}$ such that $a \sqcap c \sqsubseteq b$, then we call *c the pseudo-complement of a relative to b*, denoted by $a \Rightarrow b$.

For finite distributive lattices the relative pseudo-complement aways exists[16].

Definition 3.3 (Boolean Algebra)**.** A *Boolean Algebra* is a distributive lattice $(L, \sqsubseteq)$ with a maximum element $\top$, a minimum element $\bot$, and a unary operator $\sim$, such that, for any $x \in L$:

$x \sqcap \sim x = \bot$ *Law of Non-Contradiction*
$x \sqcup \sim x = \top$ *Law of the Excluded Middle*

A finite Boolean Algebra has 2^n elements for some natural number n. For a 2^4 valued Boolean Algebra, for example, we say it is an "order 4" Boolean Algebra.

We henceforth assume that $\mathcal{L}$ is a finite Boolean Algebra.

3.2 Multi-Valued Kripke Structure

We define the concepts of multi-valued Kripke structures and paths on a Kripke structure. The extension of the classical notion of Kripke structures to the multi-valued one (MVKS) is straightforward. Note that some authors[3),10)] perform multi-valued model checking on MVKS where the predicates take values from an mv-algebra[3),10)], but they keep the transition relation defined over 2-valued lattice (order 2 Boolean algebra), while others consider also mv-transition relations (for instance,[23)]). On this work, we follow the second, more general, approach.

Definition 3.4 (Multi-Valued Kripke Structure)**.** A *Multi-Valued Kripke Structure* (MVKS) is the tuple $\mathcal{M} = \langle S, S_0, R, \mathcal{AP}, \mathcal{L}, \mathcal{O} \rangle$ with components defined as follows:

- S is a finite set of states.
- $S_0 \subseteq S$, the set of initial states.
- $R : S \times S \to L$ is a function called *mv-transition relation*.
- $\mathcal{AP}$ is a finite set of atomic propositions.
- $\mathcal{L}$ is an mv-algebra (in our case, finite Boolean algebra), defined as $\langle L, \sqcap, \sqcup, \sim, \bot, \top \rangle$.
- $\mathcal{O} : S \times \mathcal{AP} \to L$ is a function that maps a pair $(s, a) \in S \times \mathcal{AP}$ to some $l \in L$.

Definition 3.5 (Path on a Kripke structure)**.** Let $\mathcal{M} = \langle S, S_0, R, \mathcal{AP}, \mathcal{L}, \mathcal{O} \rangle$ be a MVKS. A *path* is a mapping $\pi : \mathbb{N} \to S$. Note that, in MVKS, any sequence of states is a path. Obviously, not all sequences of states are useful.

For a path π, π^j denotes the j-th suffix (path), that is, $\pi^j(i) = \pi(i + j)$ for $i \geq 0$.

When there is no confusion, we may also write a path π as $s_0, s_1, s_2, \ldots$, having the intended meaning that $\pi(0) = s_0, \pi(1) = s_1, \pi(2) = s_2, \ldots$.

Definition 3.6 (Initialized path)**.** We call a path π an *initialized path* iff $\pi(0) \in S_0$. We define the set of paths $\Pi_0 = \{\pi \mid \pi(0) \in S_0\}$.

3.3 Linear-time Temporal Logic

Linear-time Temporal Logic (LTL) is a logic that allows one to refer to the future[13),19)].⋆1 We slightly extend the syntax of LTL formulas so that a lattice value $l \in L$ becomes a formula. We call this extension mv-LTL.

Definition 3.7 (Syntax of mv-LTL)**.** Let $\mathcal{AP}$ be a set of atomic propositions. Let $p \in \mathcal{AP}$ and $l \in L$. The syntax of mv-LTL is defined as follows:

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

$l \in L$ is called a *truth value*.

By convention, to simplify the use of parenthesis, we assume that the unary connectives ($\neg$, X, F and G) bind most tightly. Next come U, R, $\wedge$, $\vee$ and $\rightarrow$, in the order.

The meaning of the standard logical operators ($\neg$, $\wedge$, $\vee$, $\rightarrow$) remains the same as in classical two-valued logic. *They assert about the current state.* The temporal operators (X, F, G, U and R) assert about future states.

Informally, X states about the next state, i.e. $\mathsf{X}\,\phi$ means "ϕ holds at the next state". F states about sometime in future, i.e. $\mathsf{F}\,\phi$ means "ϕ holds at some future state". G expresses a permanent property of the model, i.e. $\mathsf{G}\,\phi$ means "ϕ always holds". U is G with a limit, i.e. $\phi\,\mathsf{U}\,\psi$ means "ϕ holds at every state until ψ holds, and ψ holds at some future state." R is a dual of U.

We then give the semantics of an mv-LTL formula with respect to an MVKS.

Definition 3.8 (Semantics of mv-LTL)**.** Let $\mathcal{M} = \langle S, S_0, R, \mathcal{AP}, \mathcal{L}, \mathcal{O} \rangle$ be an MVKS. Let $\pi = s_0, s_1, s_2, \ldots$ be a path on $\mathcal{M}$. Then, the semantics of an mv-LTL formula is recursively defined by the validity relation $\models$ as follows:

(1) $(\pi \models l) = l$, for $l \in L$.
(2) $(\pi \models p) = \mathcal{O}(\pi(0), p)$, for $p \in \mathcal{AP}$.
(3) $(\pi \models \neg\phi) = \ \sim(\pi \models \phi)$.
(4) $(\pi \models (\phi \wedge \psi)) = (\pi \models \phi) \sqcap (\pi \models \psi)$.
(5) $(\pi \models (\phi \vee \psi)) = (\pi \models \phi) \sqcup (\pi \models \psi)$.
(6) $(\pi \models (\phi \rightarrow \psi)) = ((\pi \models \phi)) \Rightarrow (\pi \models \psi)$.

⋆1 There is an extension of LTL, LTL with Past Operators (PLTL), that allows to refer to the past as well.

(7) $(\pi \models \mathsf{X}\phi) = (\pi^1 \models \phi)$.
(8) $(\pi \models \mathsf{F}\phi) = \bigsqcup_{i\geq 0} \pi^i \models \phi$.
(9) $(\pi \models \mathsf{G}\phi) = \bigsqcap_{i\geq 0} \pi^i \models \phi$.
(10) $(\pi \models \phi \mathsf{U} \psi) = \bigsqcup_{i\geq 0}((\pi^i \models \psi) \sqcap (\bigsqcap_{j<i} \pi^j \models \phi))$.
(11) $(\pi \models \phi \mathsf{R} \psi) = \bigsqcap_{i\geq 0}((\pi^i \models \phi) \sqcup (\bigsqcup_{j\leq i} \pi^j \models \psi))$.

We say that ϕ *is not valid* along π if $\pi \models \phi$ is $\bot$ and that ϕ *is valid* along π if $\pi \models \phi$ is $\top$. Otherwise, for $(\pi \models \phi) = l$ we say that ϕ *has validity "l"* along π.

3.4 Multi-Valued Model Checking Problem

On two-valued LTL model checking the (universal) model checking problem is defined as the problem of verifying if a formula ϕ is valid for all initialized paths of a given model $\mathcal{M}$.[1)]

Definition 3.9 (Model-Checking Problem)**.** Given a Kripke structure $K = \langle S, S_0, \mathcal{AP}, O\rangle$, let $\Pi_0 = \{\pi \,|\pi(0) \in S_0\}$. The (universal) model checking problem, $\mathcal{M} \models \phi$, in LTL is defined as

$$\mathcal{M} \models \phi \stackrel{\text{def}}{=} \forall_{\pi\in\Pi_0}.(\pi \models \phi)$$

and also, according to the semantics of LTL the following equivalence holds.

$$\forall_{\pi\in\Pi_0}.(\pi \models \phi) \equiv \neg(\exists_{\pi\in\Pi_0}.(\pi \models \neg\phi))$$

.

Then, in order to solve the universal model checking problem for a given ϕ we show that the existential model checking problem for $\neg\phi$ has no solution.

Following the 2-valued definition, we define the multi-valued model checking problem as *the problem of verifying, for a MVKS $\mathcal{M}$, if for all initialized paths π, $\pi \models \phi$ is valid for a given specification ϕ.* However, not all initialized paths are useful and it is not the case that a path is either completely useful or completely useless, therefore, we must refine this intuition in some way. Namely, we should take into account the degree of "usefulness" of a path π. Here we take essentially the same definition as[2)3)].

Definition 3.10 (Weight of a Path)**.** Let $\mathcal{M} = \langle S, S_0, R, \mathcal{AP}, \mathcal{L}, \mathcal{O}\rangle$ be an MVKS, and π is a path over $\mathcal{M}$. We define the weight of π as an element of the lattice $\mathcal{L}$ by:

$$Weight(\pi) = \bigsqcap_{i\geq 0} R(\pi(i), \pi(i+1)).$$

.

If $Weight(\pi) = \top$, it is a valid path. If $Weight(\pi) = \bot$, it is an invalid (useless) path. Besides these, there may be paths whose weights are not $\bot$ nor $\top$.

We can now define the validity of an mv-LTL formula *relative to a MVKS $\mathcal{M}$* as follows.

Definition 3.11 (Multi-Valued Validity)**.** Let $\mathcal{M} = \langle S, S_0, R, \mathcal{AP}, \mathcal{L}, \mathcal{O}\rangle$ be an MVKS. The validity of an mv-LTL formula ϕ is defined by:

$$(\mathcal{M} \models \phi) = \bigsqcap_{\pi\in\Pi_0} ((\sim Weight(\pi)) \sqcup (\pi \models \phi))$$

.

Note that this definition coincides with the definition for two-valued validity given in[2)] when $L = \{\bot, \top\}$. Note also that, if $Weight(\pi) = \bot$, then such a path π does not affect the validity of a formula.

The definition above gives the exact truth value of an mv-LTL formula ϕ with respect to a MVKS $\mathcal{M}$, i.e. the exact degree of validity of ϕ w.r.t. $\mathcal{M}$. Rather than computing the exact degree of validity of ϕ, we may want to do more general queries. For instance, we may want to know if $(\mathcal{M} \models \phi) \sqsupseteq l$ for some lattice value $l \in L$. This kind of queries can be reduced to an "exact" query since $(\mathcal{M} \models \phi) \sqsupseteq l$ is true iff $l \Rightarrow (\mathcal{M} \models \phi)$ is $\top$, and this is equivalent to $\mathcal{M} \models (l \rightarrow \phi)$. Hence we only consider the standard specification, and our model checking problem is whether $\mathcal{M} \models \phi$ equals $\top$ or not. Unfortumately this result does not generalize to other relational operations.

3.5 The Notion of Counterexample

Next, we should consider what is a counterexample in this setting. What we want to do is to check if $\mathcal{M} \models \phi$ is $\top$. This is equivalent to check if **for all** path $\pi \in \Pi_0$, the value $\sim Weight(\pi) \sqcup (\pi \models \phi)$ is $\top$. Then a counterexample of this assertion is **any** path $\pi \in \Pi_0$ such that $\sim Weight(\pi) \sqcup (\pi \models \phi) \sqsubset \top$. If such a counterexample exists then $\mathcal{M} \models \phi$ **is not** $\top$.

Definition 3.12 (Counterexample)**.** Let $\mathcal{M} = \langle S, S_0, R, \mathcal{AP}, \mathcal{L}, \mathcal{O}\rangle$ be a MVKS. Let ϕ be an mv-LTL formula to be interpreted as an specification over $\mathcal{M}$. We call a path $\pi \in \Pi_0$ a *counterexample for $\mathcal{M} \models \phi$* iff

$$\sim \mathrm{Weight}(\pi) \sqcup (\pi \models \phi) \sqsubset \top$$

.

When the lattice is Boolean algebra (or quasi-Boolean algebra) we can take negation, so $\neg Weight(\pi) \sqcup (\pi \models \phi) \sqsubset \top$ is equivalent to $Weight(\pi) \sqcap (\pi \models \neg\phi) \sqsupset \bot$. So, in the mv-BMV algorithim we will first negate the specification formula ϕ, and look for some path such that the inequality above holds.

4. Strategies of Multi-Valued Model Checking

As we stated in the introduction, our aim is to explore the possibility of using the BMC technique in multi-valued model checking. For this purpose, the most basic method is "slicing", which essentially

converts one 2^n-valued model checking problem to n 2-valued ones, corresponding to each "bit" of a lattice value.[⋆1]

Assuming that we can extend the two-valued bounded model checking[1)] to the multi-valued case, we can do slicing in various stages:

- We slice the multi-valued model checking problem (MVKS and mv-LTL formula) into n 2-valued model checking problems.
- We slice the multi-valued propositional formula, obtained by translating the multi-valued model checking problem, into n propositional formulas. Or,
- We slice the multi-valued CNF, obtained from the multi-valued propositional formula, into n 2-valued CNF.

We claim that the choice affects the efficiency of mv-bounded model checking, since at each stage we can employ different styles of optimizations. On the following sections we compare the first two approaches indicated above.

4.1 Problem Slicing

What we proposed, as a first attempt, is to encode the logic values of a Boolean Algebra with 2^n values in a vector of n bits. A more interesting interpretation arises when we think of a multi-valued model as a structure representing the overlaying of many different 2-valued models, each 2-valued model corresponding to one layer of the final model. We may think about each "bit" of the logic value as a layer of the model. The idea of an mv-model as a composition of standard model is illustrated in Figure 1, where the transitions of the model are labeled with 3-bit lattice values, indicating 3 composing layers.

Then we slice the model on its many layers and in this way generate many 2-valued models, as show in Figure 2.

One immediate consequence of this view is the possibility to express different viewpoints in the same composite model. Each viewpoint standing for one layer in the model. Of course, combinations of more than one layer are possible.

The definition of slicing for models (MVKS) and specifications (mv-LTL formulas) is very straightforward.

Definition 4.1 (Model Slicing)**.** Given a MVKS $\mathcal{M} = \langle S, S_0, R, \mathcal{AP}, \mathcal{L}, \mathcal{O}\rangle$, we say that the *slice index i of $\mathcal{M}$* is the (standard 2-valued) Kripke structure $\mathcal{M}^{(i)} = \langle S^{(i)}, S_0^{(i)}, R^{(i)}, \mathcal{AP}^{(i)}, \mathcal{O}^{(i)}\rangle$ as defined below:

⋆1 We recall that an element of 2^n-valued Boolean Algebra is represented by n bits.

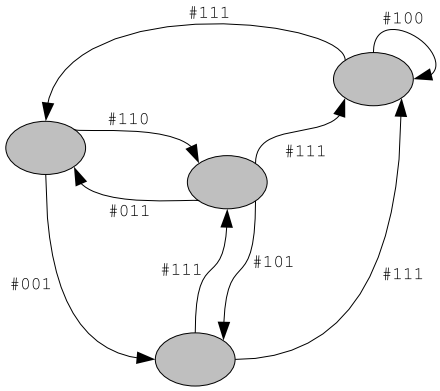

Fig. 1: Model with multiple viewpoints incorporated.

- $S^{(i)} = S$
- $S_0^{(i)} = S_0$
- $R^{(i)} = \{(s,t) \mid b_i(R(s,t)) = 1\}$
- $\mathcal{AP}^{(i)} = \{p^{(i)} \mid p \in \mathcal{AP}\}$
- $\mathcal{O}^{(i)}(s, p^{(i)}) = \begin{cases} \top, & \text{if } b_i(\mathcal{O}(s,p)) = 1 \\ \bot, & \text{otherwise} \end{cases}$

Where $b_i : L \to \{0,1\}$ is a function that maps a lattice value to its "bit" of order i, assuming a binary encoding for lattice values.

Definition 4.2 (Specification Slicing)**.** Given an mv-LTL formula ϕ, we call the *slice index i of ϕ* the (two-valued) LTL formula $\phi^{(i)}$ defined by the inductive definition below:

$$\begin{aligned} l^{(i)} &= b_i(l) \\ p^{(i)} &= p^i \\ (\neg\phi)^{(i)} &= \neg(\phi^{(i)}) \\ (\phi \wedge \psi)^{(i)} &= \phi^{(i)} \wedge \psi^{(i)} \\ (\phi \vee \psi)^{(i)} &= \phi^{(i)} \vee \psi^{(i)} \\ (\phi \to \psi)^{(i)} &= \phi^{(i)} \to \psi^{(i)} \\ (\mathsf{X}\,\phi)^{(i)} &= \mathsf{X}\,\phi^{(i)} \\ (\mathsf{F}\,\phi)^{(i)} &= \mathsf{F}\,\phi^{(i)} \\ (\mathsf{G}\,\phi)^{(i)} &= \mathsf{G}\,\phi^{(i)} \\ (\phi\,\mathsf{U}\,\psi)^{(i)} &= \phi^{(i)}\,\mathsf{U}\,\psi^{(i)} \\ (\phi\,\mathsf{R}\,\psi)^{(i)} &= \phi^{(i)}\,\mathsf{R}\,\psi^{(i)} \end{aligned}$$

.

Where $p^i \in \mathcal{AP}^{(i)}$ and $\mathcal{AP}^{(i)}$ is defined as $\mathcal{AP}^{(i)} = \{p^i \mid p \in \mathcal{AP}\}$.

Definition 4.3 (Problem Slicing)**.** Let $\mathcal{L} = \langle L, \sqsubseteq\rangle$ be a boolean lattice of order n. Given a standard multi-valued model-checking problem $P = \mathcal{M} \models \phi$, where $\mathcal{M}$ is a MVKS, ϕ is a mv-LTL formula, we define $P^{(i)}$, the *slice index i of P*, as the (2-valued) model checking problem, given by the expression below:

$$P^{(i)} = \mathcal{M}^{(i)} \models \phi^{(i)}$$

.

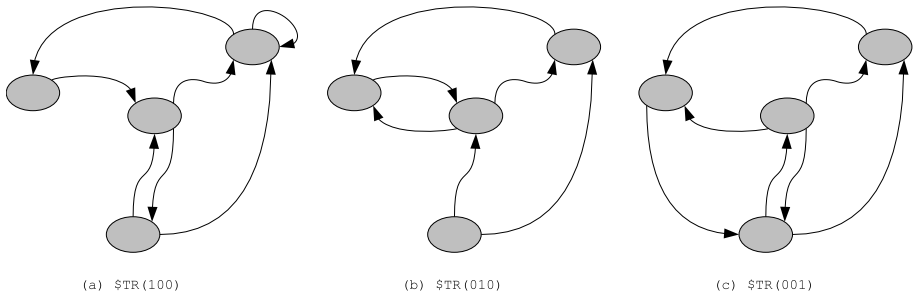

Fig. 2: Three different *"slices"* of the original model.

The following lemma sumarizes the method of naive multi-valued model checking problem slicing.

Lemma 4.1. *For a multi-valued model-checking problem $P = \mathcal{M} \models \phi \geq l$, based on a boolean lattice of order n, the following equivalence holds:*

$$\begin{aligned}\mathcal{M} \models \phi \quad \Leftrightarrow \quad & \#[\mathcal{M}^{(n-1)} \models \phi^{(n-1)}, \\ & \mathcal{M}^{(n-2)} \models \phi^{(n-2)}, \\ & \ldots, \\ & \mathcal{M}^{(0)} \models \phi^{(0)}]\end{aligned}$$

.

4.2 Direct Translation + Slicing

The second approach we present for solving the problem is a direct translation designed following very closely the one proposed in[2], but with the necessary modifications to preserve the multi-valued semantics of models (MVKS) and specifications (mv-LTL formulas).

In the following we always consider that image of the equality relation is restricted to $\{\bot, \top\}$.

Definition 4.4 (Unfolding the Transition Relation)**.** Let $\mathcal{M} = \langle S, S_0, R, \mathcal{AP}, \mathcal{L}, \mathcal{O}\rangle$ be a MVKS. Let $\{s_i | i \in \mathbb{N}\}$ be a set of state variables. We define the mv-propositional formula $[\![\mathcal{M}]\!]_k$ that encodes the transition relation of $\mathcal{M}$ as:

$$[\![\mathcal{M}]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

Where,

$$I(s_i) := \bigvee_{s \in S_0} (s_i = s)$$

and

$$T(s_i, s_j) := \bigvee_{\langle u,v,l\rangle \in R} (s_i = u \wedge s_j = v) \wedge l$$

.

Definition 4.5 (Translation of an mv-LTL formula without a loop)**.** For an mv-LTL formula ϕ, a bound k, $i \in \mathbb{N}$, with $i \leq k$, the translation $[\![\phi]\!]^i_k$ of ϕ for a path without a loop is inductively defined as:

$$\begin{aligned}
[\![l]\!]^i_k &:= l \\
[\![p]\!]^i_k &:= p(s_i) \\
[\![\neg p]\!]^i_k &:= \neg p(s_i) \\
[\![\phi \wedge \psi]\!]^i_k &:= [\![\phi]\!]^i_k \wedge [\![\psi]\!]^i_k \\
[\![\phi \vee \psi]\!]^i_k &:= [\![\phi]\!]^i_k \vee [\![\psi]\!]^i_k \\
[\![\mathsf{X}\,\phi]\!]^i_k &:= \begin{cases} [\![\phi]\!]^{i+1}_k, & \text{if } i < k \\ \bot, & \text{otherwise} \end{cases} \\
[\![\mathsf{F}\,\phi]\!]^i_k &:= \bigvee_{j=i}^{k} [\![\phi]\!]^j_k \\
[\![\mathsf{G}\,\phi]\!]^i_k &:= \bot \\
[\![\phi\,\mathsf{U}\,\psi]\!]^i_k &:= \bigvee_{j=i}^{k} ([\![\psi]\!]^j_k \wedge \bigwedge_{n=i}^{j-1} [\![\phi]\!]^n_k) \\
[\![\phi\,\mathsf{R}\,\psi]\!]^i_k &:= \bigvee_{j=i}^{k} ([\![\phi]\!]^j_k \wedge \bigwedge_{n=i}^{j} [\![\psi]\!]^n_k)
\end{aligned}$$

.

Definition 4.6 (Successor in a loop)**.** Let $k, l, i \in \mathbb{N}$, with $i \leq k$ and $l \leq k$. The successor $succ(i)$ of i in an (k, l)-loop is

$$succ(i) = \begin{cases} i+1, & \text{if } i < k \\ l, & \text{otherwise} \end{cases}$$

.

Definition 4.7 (Translation of an mv-LTL formula for a loop)**.** For an mv-LTL formula ϕ, a bound k, $i, l \in \mathbb{N}$, with $i, l \leq k$, the translation ${}_l[\![\phi]\!]^i_k$ of ϕ for

a (k,l)-loop path is inductively defined as:

$$
\begin{array}{rcl}
{}_l[\![l]\!]^i_k & := & l \\
{}_l[\![p]\!]^i_k & := & p(s_i) \\
{}_l[\![\neg p]\!]^i_k & := & \neg p(s_i) \\
{}_l[\![\phi \wedge \psi]\!]^i_k & := & {}_l[\![\phi]\!]^i_k \wedge {}_l[\![\psi]\!]^i_k \\
{}_l[\![\phi \vee \psi]\!]^i_k & := & {}_l[\![\phi]\!]^i_k \vee {}_l[\![\psi]\!]^i_k \\
{}_l[\![\mathsf{X}\,\phi]\!]^i_k & := & {}_l[\![\phi]\!]^{succ(i)}_k \\
{}_l[\![\mathsf{F}\,\phi]\!]^i_k & := & \bigvee^k_{j=\min(i,l)} {}_l[\![\phi]\!]^j_k \\
{}_l[\![\mathsf{G}\,\phi]\!]^i_k & := & \bigwedge^k_{j=\min(i,l)} {}_l[\![\phi]\!]^j_k \\
{}_l[\![\phi\,\mathsf{U}\,\psi]\!]^i_k & := & \bigvee^k_{j=i} \left({}_l[\![\psi]\!]^j_k \wedge \bigwedge^{j-1}_{n=i} {}_l[\![\phi]\!]^n_k \right) \vee \\
 & & \bigvee^{i-1}_{j=l} \left(\begin{array}{c} {}_l[\![\psi]\!]^j_k \wedge \bigwedge^k_{n=i} {}_l[\![\phi]\!]^n_k \wedge \\ \bigvee^{j-1}_{n=l} {}_l[\![\phi]\!]^n_k \end{array} \right) \\
{}_l[\![\phi\,\mathsf{R}\,\psi]\!]^i_k & := & \bigwedge^k_{j=\min(i,l)} {}_l[\![\psi]\!]^j_k \vee \\
 & & \bigvee^k_{j=i} \left({}_l[\![\phi]\!]^j_k \wedge \bigwedge^{j}_{n=i} {}_l[\![\psi]\!]^n_k \right) \vee \\
 & & \bigvee^{i-1}_{j=l} \left(\begin{array}{c} {}_l[\![\phi]\!]^j_k \wedge \bigwedge^k_{n=i} {}_l[\![\psi]\!]^n_k \wedge \\ \bigwedge^{j}_{n=l} {}_l[\![\psi]\!]^n_k \end{array} \right)
\end{array}
$$

.

Definition 4.8 (Loop Condition)**.** For $k, l \in \mathbb{N}$, let ${}_lL_k := T(s_k, s_l)$, and $L_k = \bigvee^k_{l=0} {}_lL_k$.

Definition 4.9 (General Translation)**.** Let ϕ be an mv-LTL formula, $\mathcal{M}$ an MVKS and $k \in \mathbb{N}$. Then,

$$[\![\mathcal{M}, \phi]\!]_k := [\![\mathcal{M}]\!]_k \wedge [\![\phi]\!]_k$$

Where,

$$[\![\phi]\!]_k := \left(\neg L_k \wedge [\![\phi]\!]^0_k\right) \vee \bigvee_{l=0}^{k} \left({}_lL_k \wedge {}_l[\![\phi]\!]^0_k\right)$$

.

It is important to note that $Weight(\pi)$ is automatically encoded in $[\![\phi]\!]_k$, as ${}_lL_k$ or L_k. So checking the following property for k:

> There is a path π such that $Weight(\pi) \sqcap (\pi \models \neg\phi) \sqsupset \bot$.

Reduces checking the following one:

> There are states $s_0, s_1, \ldots, s_k$ such that $[\![\phi]\!]_k \sqsupset \bot$

By slicing, this final property reduces to checking (for a fixed k):

> There are states $s_0, s_1, \ldots, s_k$ such that there is some slice i for which $b_i([\![\phi]\!]_k) = 1$ (i.e. non-zero).

Hence, we take disjunction of formulas generated by slicing, and check its satisfiability by an ordinary, two-valued SAT-solver.

5. Prototype Implementation

On this section we describe our prototype implementation of multi-valued bounded model checking. Our method can be summarized as follows. We encode a boolean algebra with 2^n values in a vector of n bits. A few syntactical constructions ware defined with the following additional features:

- A statement `lattice boolean(`n`)`, that specify the size of the boolean algebra being used for the model.
- The lattice values are accepted as literals, and specified as $\#d_1 \ldots d_n$ where, $d_{i_{1 \le i \le n}} \in \{0, 1\}$.
- A special predicate `$TR(`l`)`, where l is a lattice value. We will discuss this predicate latter.
- Specifications are defined in relation to a given lattice value. This has the same effect of defining a set of designated values as proposed in 22), except that the set of designated values is not bound to the model, but to individual specifications.

The role of the predicate `$TR(`l`)` is paramount for the mapping of the multi-valued model to a 2-valued one. The basic idea is that `$TR(`l`)` can be used to define the logic values associated with the transitions in the model. In a rouge view, `$TR(`l`)` can be compared to the mv-transition relation $\mathbb{R}$.

Although, the interpretation of `$TR(`l`)` as corresponding to the $\mathbb{R}$ is justifiable, we think that a more interesting interpretation arise when we think of a multi-valued model as a structure representing the overlaying of many different 2-valued models, each 2-valued model corresponding to one layer of the final model, and then we think of `$TR(`l`)` as a selector that specifies on which layers the outermost logic expression applies. The idea of a mv-model as a composition of standard model is illustrated in **Fig. 1**, where the transitions of the model are labeled with 3-bit lattice values, indicating 3 composing layers.

Figure 2 illustrate the use of the special predicate `$TR(`l`)` as a selector of the layers of the mv-model. Each bit in the argument of `$TR(`l`)` selects a different layer of the mv-model. Of course, combinations of more than one layer are possible. One immediate consequence of this view is the possibility to express different viewpoints in a same composite model. Each viewpoint standing for one layer in the model.

The translation of the model, from the multi-valued description, to a standard 2-valued description is outlined by Algorithm 1. Basically, each mv-variable is translated to an array of size n, and each mv-expression is translated to an equivalent list of n expressions.

6. Experimental Results

We have implemented a bounded multi-valued model checker prototype. The prototype translates a MVMC problem description to a *conjunctive nor-*

Algorithm 1 General algorithm for applying MVBMC.

Require: $\mathcal{M}$ a *Model* object; ϕ a mv-LTL formula object; k the bound to be applied.

Ensure: The boolean valuation that represents a counterexample to ϕ in $\mathcal{M}$ if ϕ does not hold; or the empty valuation if ϕ holds.

```
1: procedure MVBMC(M, φ, k, fname)
2:     f ← MVBMCTRANS(M, ¬φ, k)
3:     fs ← SLICE(fs, k)
4:     g ← MKPF(dsj, fs)
5:     BOOL2CNF(g, fname)
6:     RUNSATSOLVER(fname)
7: end procedure
```

Algorithm 2 Translation from multi-valued model to multi-valued finite domain propositional formula.

Require: $\mathcal{M}$ a *Model* object; ϕ a mv-LTL formula object; k the bound to be applied.

Ensure: A multi-valued finite domain propositional formula that encodes the MVBMC problem for a bound k.

```
1: function MVBMCTRANS(M, φ, k)
2:     i_0 ← INSTANCE(M.init, 0)
3:     for i ← 0..(k − 1) do
4:         tr[i] ← INSTANCE(M.trans, i, i + 1)
5:     end for
6:     m ← FOLDL(∨, i_0, tr)
7:     f ← TRSPECNOLOOP(M, φ, k)
8:     g ← TRSPECLOOP(M, φ, k)
9:     h ← (f ∨ g)
10:    return (m ∧ h)
11: end function
```

Algorithm 3 Replace *abstract* model variables by its equivalent *concrete* state variables.

Require: $\mathcal{M}$ — the model data structure; ϕ — a finite domain propositional formula; i — the current state variable index; j — the next state variable index, if not specified its default value is -1.

Ensure: ψ — an *instantiated* finite domain propositional formula equivalente to ϕ, but with all model propositional variables replaced by its corresponding concrete state propositional variables.

```
1: function INSTANCE(M, φ, i, j)
2:     match φ with
3:     | v ⇒
4:         return GETCONCRETEVAR(M, v, i)
5:     | v′, when j < 0 ⇒
6:         error: "Invalid use of next state variable."
7:     | v′, when j ≥ 0 ⇒
8:         return GETCONCRETEVAR(M, v, j)
9:     | n ⇒
10:        return n
11:    | x_1 = x_2 ⇒
12:        y_1 ← INSTANCE(M, x_1, i)
13:        y_2 ← INSTANCE(M, x_2, i)
14:        return y_1 = y_2
15:    | ¬φ_1 ⇒
16:        ψ_1 ← INSTANCE(M, φ_1, i)
17:        return ¬ψ_1
18:    | φ_1 ⊙ φ_2 ⇒
19:        ψ_1 ← INSTANCE(M, φ_1, i)
20:        ψ_2 ← INSTANCE(M, φ_2, i)
21:        return ψ_1 ⊙ ψ_2
22:    end match
23: end function
```

mal formula (CNF) in DIMACS format[20] for satisfiability problems, and uses the MiniSAT SAT solver tool[15] to look for a solution.

6.1 Example

As an example of the use of multi-valued logics for modeling systems, we take the telephone system studied in[14] and[11]. On its original presentation, two separated versions of the same feature were specified and the separated models were merged to reason about which properties are agreed and which are disputed. Figure 3 shows the model of the caller's perspective of the system, and Figure 4 shows the perspective of the callee. The details of the merging procedure are outside the scope of this paper, but are explained in[14]. The merged model is presented in Figure 5.

As can be seen there are disagreement about a number of transitions in the model. Some analysis is necessary if we want be able to tell if these disagreements affect or not the overall design of the system.

6.2 Experiments

For the tests presented here we encode the telephone system shown in Section 6.1 for a 2^2 boolean logic, and extended the model for other 2^n logics, with $1 \leq n \leq 8$. Then we tested each model for increasing values of the bound k. All tests were performed on a Intel Core 2 Duo T7200 (2.0 GHz)

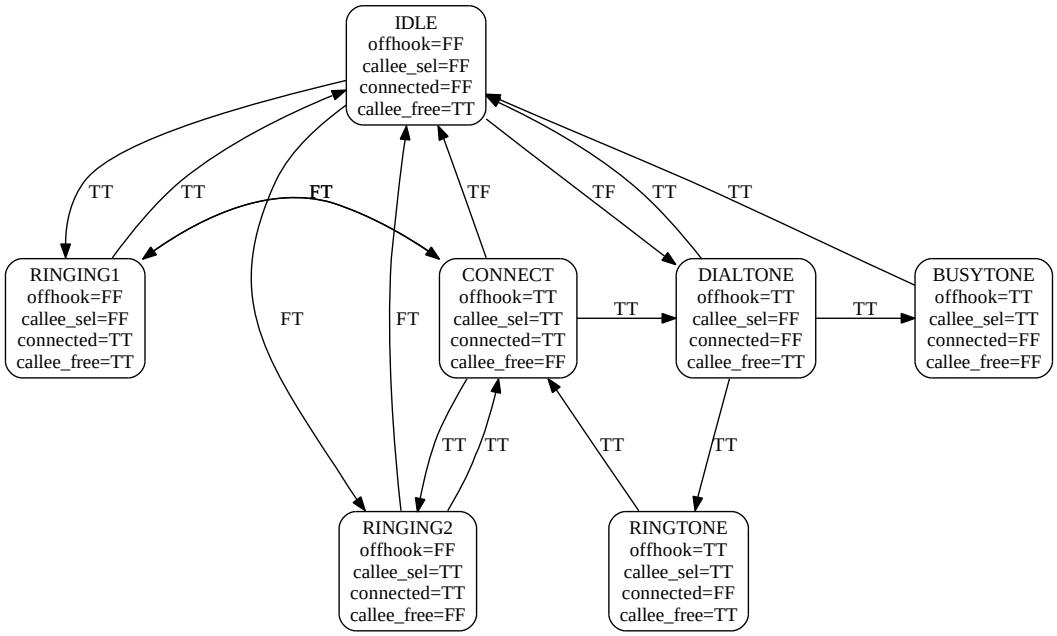

Fig. 5: Combined caller's and callee's persperctives of the telephone system.

Algorithm 4 Retrieve a concrete state propositional variable corresponding to the given abstract model variable

Require: $\mathcal{M}$ — the model data structure; x — a model propositional variable; i — the current state variable index.

Ensure: The concrete state propositional variable correponding to the model propositional variable x.

function GETCONCRETEVAR($\mathcal{M}, k, x, i$)
  nvars $\leftarrow$ length($\mathcal{M}$.vars)
  $x_{\text{pos}} \leftarrow$ index($\mathcal{M}$.vars)
  return $(i \times \text{nvars}) + x_{\text{pos}}$
end function

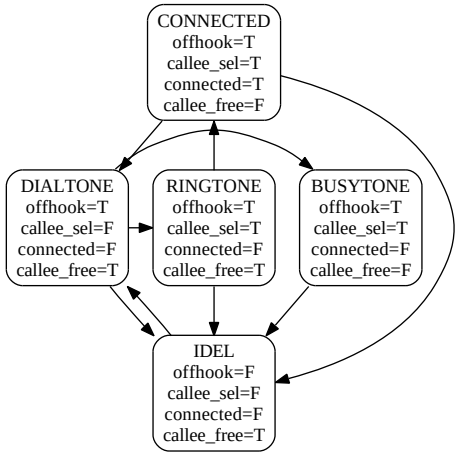

Fig. 3: Caller persperctive of the telephone system.

processor machine with 2.0 GB of RAM.

Figure 6 shows the growth of the number of variables in the CNF generated for each model as a function of k. Figure 7 shows the corresponding growth in the number of clauses in the CNF, also as a function of k. As it is easy to see, both measures (number of variables and number of clauses) seems to grow linearly with k. This behavior may be justified by the particular caracteristic of the propositional formulae generated by our translation. It is easy to note that our approach generate a very specific kind of formulae, by composing formulas that are greatly similar in structure, but each with its own set of variables. Here we remember that we take the disjunction of the propositional formulas generated by slicing the multi-valued formula that encodes the MVMC problem. Also, our implementation uses a direct acyclic graph to represent propositional formulae, this allows us to translate shared sub-formulae only once in the CNF generation.

Since the number of variables and the number of clauses in the CNF grow linearly with k, it is not

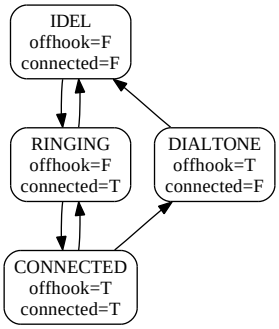

Fig. 4: Callee persperctive of the telephone system.

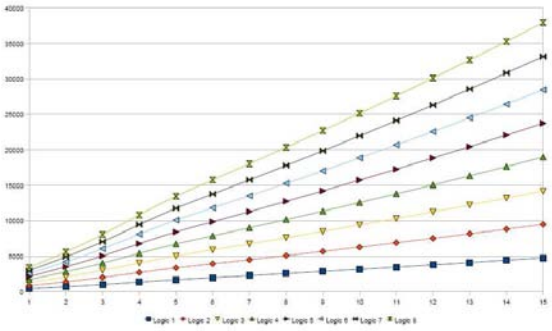

Fig. 6: Number of variables in CNF × k.

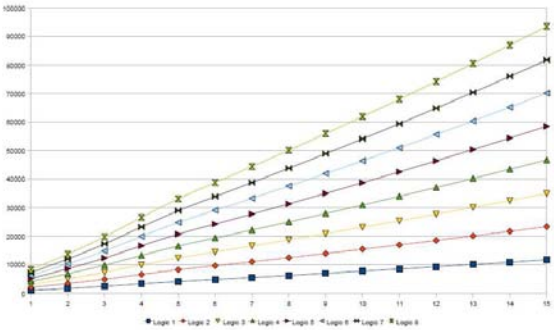

Fig. 7: Number of clauses in CNF × k.

surprising that the time required by our prototype appears to grow polinomialy with k, as shown in Figure 8. Although, we must say that, as for now, we do not have enought evidence to claim that the time complexity in the average case is indeed polinomial, as for the worst case it is known to be explonential.

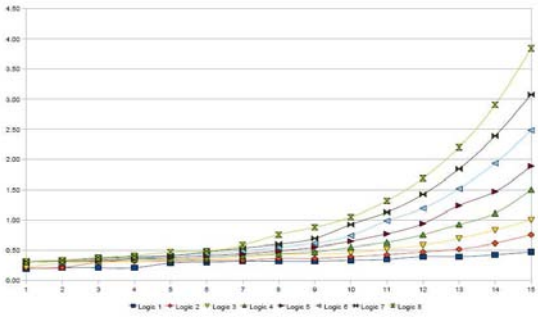

Fig. 8: Time × k, time in milliseconds.

7. Related Work and Conclusion

In recent years a number of researchers has shown interest in the problem of multi-valued model checking, and many distinct approaches has been proposed. The translation of a 3-valued model checking problem for CTL* and the model μ-calculus to a standard model checking problem was defined by Bruns and Godefroid[4),5)]. A restricted version of the problem with a 2-valued transition relation in the model was considered.

Another approach was adopted by Chechik *et al*. A new model checking algorithm for a multi-valued version of CTL was defined exploiting mv-BDD's[9)] for unrestricted interpretations and MTBDD's[8)] for finite distributive quasi-boolean algebras. Still, a model checker for mv-LTL under restrict interpretations (2-valued transition relation and totally ordered sets for the propositions) haves been implemented, based on a translation to (mv-)Büchi automata.[10)]

A translation from a negation-free mv-CTL* to CTL* model checking for model over finite quasi-boolean lattices was shown by Konikowska and Penczek[21)], that later revised their technique to make use of designated values in complete lattices[22)].

Also, model checking algorithms for mv-CTL over multi-valued interpretations featuring different notions of negations were considered Chechik *et al.*[6)].

Regarding, bounded model checking, the original idea has been proposed by Biere, Clarke et al.[1),2),12)]. For the CTL logics, an extension of the BMC method based on SAT procedures to verification of all the properties expressed in ACTL was shown by Penczek et al.[24)].

We have presented a new algorithm for solving the multi-valued model checking problem by ap-

plying bounded model checking, and we state that the proposed translation procedure is sound. This opens the possibility of applying already known optimization techniques for standard BMC in the context of multi-valued model checking. Our preliminary experimental results seems to confirm that the method is efficient and scalable. One more evidence that contributes to this conclusion is shown by Table 1 that show the ration between the time needed for the translation from the MVMC problem to mv-SAT problem. As can be seen, the time consumed by the translation has the same magnitude of the remaining of the process, i.e. the time required for generating the CNF and running the SAT solver. This represents an improvement of orders of magnitude over the naive solution to the problem.

Despite or perliminary results, there are still much ground for further improve this work. At least three points must be addressed in the short term.

(1) We need more comprehensive tests to validate or conjecture that BMC applied to the multi-valued logic context does preserve advantages that it pocesses in the 2-valued context.

(2) We need to improve our prototype so that we can investigate more meaningful examples

(3) We expect to extend our approach to cover more general logics like pseudo-boolean algebras and, maybe, Heyting algebras.

These improvements will provide the means to conduct experiments with realistic applications of multi-valued logics. Many topics can be further investigated, but at the moment we target at the capacity of bounded model checking to efficiently find counterexamples and the possibility that multi-valued temporal logic provides to express disagreement will certainly provide powerful tools to reason about model refinement.[26)]

References

1) Biere, A., Cimatti, A., Clarke, E., Strichman, O. and Zhu, Y.: Bounded model checking (2003).
2) Biere, A., Cimatti, A., Clarke, E. and Zhu, Y.: Symbolic Model Checking without BDDs, *Lecture Notes in Computer Science*, Vol.1579, pp.193–207 (1999).
3) Bruns, G. and Godefroid, P.: Model checking with multi-valued logics (2003).
4) Bruns, G. and Godefroid, P.: Model Checking Partial State Spaces with 3-Valued Temporal Logics, *Computer Aided Verification*, pp.274–287 (1999).
5) Bruns, G. and Godefroid, P.: Generalized Model Checking: Reasoning about Partial State Spaces, *Lecture Notes in Computer Science*, Vol. 1877, pp. 168+ (2000).
6) Chechik, M. and MacCaull, W.: CTL model-checking over logics with nonclassical negations (2003).
7) Chechik, M., Devereaux, B., Easterbrook, S. and Gurfinkel, A.: Multi-Valued Symbolic Model-Checking, *ACM Transaction on Software Engineering and Methodology*, Vol. 2, No. 4, pp. 371–408 (2003).
8) Chechik, M., Devereaux, B., Easterbrook, S., Lai, Y. C. and Petrovykh, V.: Eficient Multiple-Valued Model-Checking Using Lattice Representations, *Lecture Notes in Computer Science*, Vol.2154, pp.441–455 (2001).
9) Chechik, M., Devereux, B. and Easterbrook, S.: Implementing a Multi-Valued Symbolic Model Checker, *Proceedings of 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, Lecture Notes in Computer Science, Vol. 2031, Springer, pp. 404–419 (2001).
10) Chechik, M., Devereux, B. and Gurfinkel, A.: Model-Checking Infinite State-Space Systems with Fine-Grained Abstractions Using SPIN, *Lecture Notes in Computer Science*, Vol. 2057, pp. 16+ (2001).
11) Chechik, M., Gurfinkel, A., Devereux, B., Lai, A. and Easterbrook, S.: Data structures for symbolic multi-valued model-checking, *Form. Methods Syst. Des.*, Vol.29, No.3, pp.295–344 (2006).
12) Clarke, E. M., Biere, A., Raimi, R. and Zhu, Y.: Bounded Model Checking Using Satisfiability Solving, *Formal Methods in System Design*, Vol. 19, No.1, pp.7–34 (2001).
13) Clarke, E. M., Grumberg, O. and Peled, D. A.: *Model Checking*, The MIT Press (1999).
14) Easterbrook, S. and Chechik, M.: A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints, *International Conference on Software Engineering*, pp.411–420 (2001).
15) Een, N. and Sörensson, N.: An Extensible SAT-solver [ver 1.2].
16) Fitting, M.C.: Many-Valued Modal Logics, *Fundamenta Informaticae*, Vol.XV, pp.235–254 (1991).
17) Fitting, M.C.: Many-Valued Modal Logics II, *Proc. LFCS'92*, Springer-Verlag (1992).
18) Gurfinkel, A. and Chechik, M.: Multi-valued model checking via classical model checking, *CONCUR 2003 – Concurrency Theory, 14th International Conference* (Armadio, R. M. and Lugiez, D., eds.), Lecture Notes in Computer Science, Vol.2761, Marseille, France, Springer, pp.263–277 (2003).
19) Huth, M. and Ryan, M.: *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridg University Press, Cambridg, UK, 2nd edition (2004).
20) Johnson, D.S. and Trick, M.A.(eds.): *Cliques, Coloring and Satisfiability: Second DIMACS Implemen-*

k	Logic 1	Logic 2	Logic 3	Logic 4	Logic 5	Logic 6	Logic 7	Logic 8
1	0.416	0.385	0.385	0.316	0.263	0.263	0.300	0.300
2	0.461	0.385	0.333	0.300	0.300	0.286	0.286	0.619
3	0.384	0.316	0.316	0.300	0.333	0.565	0.522	0.542
4	0.461	0.300	0.350	0.545	0.565	0.542	0.520	0.538
5	0.333	0.300	0.333	0.591	0.542	0.538	0.577	0.533
6	0.315	0.286	0.545	0.542	0.538	0.536	0.533	0.548
7	0.300	0.571	0.542	0.577	0.536	0.548	0.559	0.541
8	0.300	0.565	0.519	0.536	0.548	0.514	0.526	0.479
9	0.300	0.565	0.536	0.533	0.543	0.538	0.545	0.500
10	0.571	0.560	0.552	0.571	0.561	0.553	0.492	0.507

Table 1: Ration of the translation time with relation to the total execution time.

tation Challenge, DIMACS Series In Discrete Mathematics and Theoretical Computer Science, Vol.26, AMS (1996).

21) Konikowska, B. and Penczek, W.: Reducing Model Checking from Multi-Valued CTL* to CTL* (2002).
22) Konikowska, M. C.: On Designated Values in Multi-Valued CTL* Model Checking (2004).
23) Nishizawa, K., Kameyama, Y. and Kinoshita, Y.: Simulations of Multi-Valued Models for Modal μ-Calculus, Technical Report AIST01-J00022-68, National Institute of Advanced Industrial Science and Technology (AIST), Japan (2007).
24) Penczek, W., Wo'zna, B. and Zbrzezny, A.: Bounded model checking for the universal fragment of CTL (2002).
25) Tassey, G.: The Economic Impacts of Inadequate Infrastructure for Software Testing, Technical Report 7007.011, National Institute of Standards and Technology – NIST (2002).
26) Tatsumi, Y. and Kameyama, Y.: Towards Modeling-error Detection Using Multi-valued Model Checking, *IPSJ Transactions on Programming*, No. 47 (2006).