Multi-Valued Bounded Model Checking

Jefferson Oliveira Andrade [†] and Takuo Yonezawa[†]

This article describes the implementation of the first (to our best knowledge) multi-valued model-checker which is based on bounded model checking. Reasons to support the originality of this work and foreseen impacts are given bellow.

1. Introduction

Software specification is a challenging task. It is not uncommon to be faced with contradictory specifications or with uncertainty in requirements. Current specification techniques and tools lack the proper treatment of these problems. It would be highly desirable to have tools that allow both (1) more precise creation of systems models *(system requirements)*, and (2) to do more powerful reasoning about the modeled systems *(validate specifications)*. As a first step toward this direction we propose a multi-valued bounded model-checking tool.

We want to be able to create models with incorporate uncertainty and disagreement. This is not possibly done in a natural way using classic 2-valued logics. That leads us to propose the use of multi-valued logics[22),23)], to express both or model (requirements) and the properties (specifications) that we want to check for these models.

Model checking (MC) is a successful technique for formal verification of systems having been successfully applied to many domains, like hardware design verification, protocol verification, robotic control verification, etc.[2),19),26)] Some organizations that make use of it include Intel, Microsoft, NASA and Lucent Technologies.

In early 90's a variation of MC called Symbolic Model Checking (SMC) was proposed.[27)] SMC was based on Binary Decision Diagram (BDD) manipulation and has allowed the verification of systems with more than 10^{20} states. Unfortunately the amount of memory required to store and manipulate BDDs can grow exponentially on the number of state variables. In late 90's a variation of MC called Bounded Model Checking (BMC), based on boolean satisfiability (SAT) checkers, was proposed.[3),4)] SAT procedures do not suffer from the space explosion problem.

This report describes our first attempt to create a tool that combines the expressive power of many-valued logics and the efficiency of Bounded Model Checking. The rest of the report is organized as follows. Section 2 describes further motivations for this work. Section 3 presents related works. Section 4 defines the syntax and semantics of standard 2-valued CTL. Section 5 introduces briefly Bounded Model Checking. Section 6 expands the topics of section 4 and 5 to the multi-valued case. Section 7 describes our prototype implementation. The report ends with our conclusions and states future directions for this 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.[33)] 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

† Department of Computer Science, University of Tsukuba

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. Related work

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[5),6)]. 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[12)] for unrestricted interpretations and MTBDD's[11)] 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.[13)]

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[28)], that later revised their technique to make use of designated values in complete lattices[29)].

Also, model checking algorithms for mv-CTL over multi-valued interpretations featuring different notions of negations were considered Chechik *et al.*[15)].

Regarding, bounded model checking, the original idea has been proposed by Biere, Clarke et al.[3),4),18)]. 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.[31)].

4. Computational Tree Logic – CTL

Model checking can be summarized as an automated technique to verify temporal properties on finite systems.[†] In model checking literature, the standard representation of models for the system we are interested in, is *Kripke structures.* A Kripke structure is a finite transition system. Regardless of the concrete definition language, the system model can be represented by a Kripke Structure.

Definition 4.1 (Kripke structure)**.** A Kripke structure M is a t-uple, $M = \langle S, S_0, R, AP, \mathcal{O}\rangle$. S is the set of states, $S_0 \subseteq S$ is the set of initial states, $R \subseteq (S \times S)$ is a transition relation, and $\mathcal{O} : S \to \wp(AP)$ is a observation, or labeling, function, where AP is the set of atomic propositions and $\wp(AP)$ denotes the power-set over AP.

The properties we want to verify can be classified in two groups:

- *Safety properties* – Regards the fact that some forbidden state is unreachable by any path from any of the initial states.
- *Liveness properties* – If some desirable state will eventually be reached. To specify liveness properties, reachability analysis is not enough, so we use *temporal logics* to specify them.

To express specifications of transition systems, we need to state not only about current state but also about future states, e.g. "Whenever the light is turned on, it will be eventually turned off". The logic commonly used to express such a statement in model checking is CTL (Computational Tree Logic). CTL treats time as a sequence of states (path), and since future need not be determinant in CTL, it forms branching path, or computational tree.

Definition 4.2 (Path on a Kripke structure)**.** Given a Kripke structure $M = \langle S, S_0, R, \mathcal{O}\rangle$, a *path* $\pi : \mathbb{N} \to S$ is a mapping such that $\forall i \in \mathbb{N}.(\pi(i), \pi(i+1)) \in R)$.

- The set of all paths for a Kripke structure M will be denoted Π_M.
- The set of all paths starting at a given state $s \in S$ of a Kripke structure M will be denoted by $\Pi_M(s)$

4.1 Syntax of CTL

The general syntax of CTL is defined as followings:

$$\begin{aligned}\phi, \psi ::= \;& \top \mid \bot \mid p \\ & \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \to \psi \\ & \mid \mathsf{AX}\phi \mid \mathsf{AF}\phi \mid \mathsf{AG}\phi \mid \mathsf{A}[\phi \mathsf{U} \psi] \\ & \mid \mathsf{EX}\phi \mid \mathsf{EF}\phi \mid \mathsf{EG}\phi \mid \mathsf{E}[\phi \mathsf{U} \psi]\end{aligned}$$

[†] Although, there is an increasing interest in model checking for infinite systems.

Where $p \in AP$ is an atomic proposition.

In the remaining of this article, to make the definitions more manageable, we will constrain ourselves on the ECTL fragment of the CTL logic.

Definition 4.3 (ECTL)**.** The ECTL is the language obtained by restricting the CTL formulas from the use of the universal path quantifier, and also, in this text, the use of the implication and the EF operators. So the syntax of ECTL is given by:

$$\begin{aligned}\phi, \psi &::= \top \mid \bot \mid p \\ &\mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \\ &\mid \mathsf{EX}\phi \mid\mid \mathsf{EG}\phi \mid \mathsf{E}[\phi \mathsf{U} \psi]\end{aligned}$$

We stress the fact that this restriction does not affect the expressive power of CTL, since all the remaining operators can be defined in terms of the ECTL operators.

Definition 4.4 (Operator equivalence)**.** The following equivalences are used to define the remaining syntax and semantics of the CTL operators:

$$\begin{aligned}\phi \to \psi &\equiv \neg\phi \vee \psi \\ \mathsf{EF}\phi &\equiv \neg\mathsf{EG}\neg\phi \\ \mathsf{AX}\phi &\equiv \neg\mathsf{EX}\neg\phi \\ \mathsf{AF}\phi &\equiv \neg\mathsf{EF}\neg\phi \\ \mathsf{AG}\phi &\equiv \neg\mathsf{EG}\neg\phi \\ \mathsf{A}[\phi\mathsf{U}\psi] &\equiv \neg\mathsf{E}[\neg\phi\mathsf{U}\neg\psi]\end{aligned}$$

4.2 Semantics of CTL

The meaning of $\top$, $\bot$, AP and the standard logical operators ($\neg$, $\wedge$, $\vee$) remains the same as in classical logic. They assert about the current state.

The temporal operators (AX, AF, AG, $\mathsf{A[U]}$, EX, EF, EG and $\mathsf{E[U]}$) assert about future states. All operators started with A state "for all paths start from current state, ..." while operators started with E state "for some path starts from current state, ...".

AX and EX state about next state, i.e. $\mathsf{AX}\phi$ means "for all paths start from current state, ϕ hold at next state". AF and EF state about sometime in future, i.e. $\mathsf{AF}\phi$ means "for all paths start from current state, ϕ hold at some future state". AG and EG express permanent property of the model, i.e. $\mathsf{AG}\phi$ means "for all paths start from current state, ϕ always hold". AU and EU are AG and EG with a limit, i.e. $\mathsf{A}[\phi\mathsf{U}\psi]$ means "for all paths start from current state, ϕ hold at every state until ψ hold, and ψ hold at some future state.

Definition 4.5 (Semantics of ECTL)**.** Let $M = \langle S, S_0, R, \mathcal{O}\rangle$ be a model for CTL, $s \in S$, and ϕ a CTL formula. The relation $M, s \models \phi$, insists ϕ hold at s in model M, is defined by structural induction on ϕ:

(1) $M, s \models \top$

(2) $M, s \not\models \bot$

(3) $M, s \models p \iff p \in O(s)$

(4) $M, s \models \neg\phi \iff M, s \not\models \phi$

(5) $M, s \models (\phi_1 \wedge \phi_2) \iff (M, s \models \phi_1) \wedge (M, s \models \phi_2)$

(6) $M, s \models (\phi_1 \vee \phi_2) \iff (M, s \models \phi_1) \vee (M, s \models \phi_2)$

(7) $M, s \models \mathsf{EX}\phi \iff \exists_{s_1}.(s, s_1) \in R \wedge M, s_1 \models \phi$

(8) $M, s \models \mathsf{EG}\phi \iff$
$\exists\pi.(s = \pi(1) \wedge \forall_i.(\pi(i), \pi(i+1)) \in R)$
$\wedge \forall i.M, \pi(i) \models \phi$

(9) $M, s \models \mathsf{E}[\phi_1\mathsf{U}\phi_2] \iff$
$\exists_\pi.(s = \pi(1) \wedge \forall_i.(\pi(i), \pi(i+1)) \in R)$
$\wedge \exists_j.(M, \pi(j) \models \phi_2) \wedge \forall_{i<j}.(M, \pi(i) \models \phi_1)$

Having defined the semantics of CTL, the model-checking problem can now be stated as the problem of verifying if $\forall_{s \in S_0}.M, s \models \phi$ holds for a given specification ϕ.

5. Bounded model checking

The basic idea of bounded model checking, is to consider only a finite prefix of a path that may be a witness to an existential model checking problem. We restrict the length of the prefix by some bound k. Although the prefix of a path is finite, it still might represent an infinite path if there is a *back loop* from the last state fo the prefix to any of the previous states.

Definition 5.1 (k-path)**.** Giben the Kripke structure $M = \langle S, S_0, R, AP, \mathcal{O}\rangle$, and $k \in \mathbb{N}_+$, we define a k-path as a sequence $\pi_k = s_0, \ldots, s_k$ of states, such that for all i, $0 \leq i \leq k$, $(s_i, s_{i+1}) \in R$. We also define Π_k as the set of all k-paths of M.

Definition 5.2 ((k,l)-loop)**.** Giben the Kripke structure $M = \langle S, S_0, R, AP, \mathcal{O}\rangle$, for $l \leq k$ we call a path π a (k,l)-loop in M if $(\pi(k), \pi(l)) \in R$ and $\pi = u.v^\omega$ with $u = (\pi(0), \ldots, \pi(l-1))$ and $v = (\pi(l), \ldots, \pi(k))$. We cal π a k-loop if there exists $k \geq l \geq 0$ for which π is a (k,l)-loop.

Definition 5.3 (k-model)**.** Given a Kripke structure $M = \langle S, S_0, R, AP, \mathcal{O}\rangle$, we define the *k-model of* M as a structure $M_k = \langle S, S_0, \Pi_k, \mathcal{O}\rangle$

5.1 Semantics of bounded CTL

Definition 5.4 (Bounded Semantics)**.** Let M be a Kripke structure $\langle S, S_0, R, AP, \mathcal{O}\rangle$, $k \in \mathbb{N}$ be a bound, and ϕ and ψ be CTL formulas. $M_k, s \models \phi$ denotes that ϕ is true at state s of

M under a bounded interpretation with bound k, or equivalently, that ϕ is true at state s of the k-model M_k. The relation $\models$ is inductively defined as follows:

$$
\begin{aligned}
&M_k, s \models p \iff p \in O(s)\\
&M_k, s \models \neg p \iff p \notin O(s)\\
&M_k, s \models \phi \wedge \psi \iff M_k, s \models_k \psi \wedge\\
&\qquad M_k, s \models_k \phi\\
&M_k, s \models \phi \vee \psi \iff M_k, s \models_k \psi \vee\\
&\qquad M_k, s \models_k \phi\\
&M_k, s \models \mathsf{EX}\phi \iff\\
&\qquad \exists_{\pi\in\Pi_k}.(\pi(0) = s \wedge\\
&\qquad\qquad M_k, \pi(1) \models p)\\
&M_k, s \models \mathsf{EG}\psi \iff\\
&\qquad \exists_{\pi\in\Pi_k}.(\pi(0) = s \wedge\\
&\qquad\qquad \forall_{0\le i\le k}.M_k, \pi(i) \models p)\\
&M_k, s \models \mathsf{E}[\phi \mathsf{U} \psi] \iff\\
&\qquad \exists_{\pi\in\Pi_k}.(\pi(0) = s \wedge\\
&\qquad\qquad \exists_{0\le j\le k}.(M_k, \pi(j) \models \psi \wedge\\
&\qquad\qquad\qquad \forall_{0\le i\le j}.M_k, \pi(i) \models \phi))
\end{aligned}
$$

Definition 5.5 (Validity for Bounded Semantics). A CTL formula ϕ is valid in a k-model M_k of M, denoted by $M \models_k \phi$ iff $\forall_{s\in S_0}.M_k, s \models \phi$.

Bellow we present an important result about the completeness of the proposed bounded semantics of CTL. For a proof of this result, we direct the interested reader to the work of Penczek *et al.*[31]).

Theorem 5.1. *Given a Kripke structure $M = \langle S, S_0, R, \mathcal{O}\rangle$, a CTL formula ϕ and a bound $k = |M|$, then $M \models \phi$ iff $M \models_k \phi$.*

It is important to mention that it is often the case that, if $M \models \phi$, then there exists $k < |M|$ such that $M \models_k \phi$. This observation can explain the high efficiency of BMC for a great number of cases.

5.2 The BMC algorithm for CTL

In this section we present a general BMC method for CTL. For this we firs define the notion of sub-model of a k-model.

Definition 5.6. Let $M_k = \langle S, S_0, \Pi_k, \mathcal{O}\rangle$ be a k-model of a Kripke structure M, and let $\Pi'_k \subseteq \Pi_k$ be a subset of the k-paths of M_k. We define *States*$(\Pi'_k) = \{s \in S | \exists_{\pi\in\Pi_k}.\exists_{i\le k}.\pi(i) = s\}$, as the set of all states belonging to a k-path within Π'_k.

Definition 5.7 (Sub-model). Given a k-model $M_k = \langle S, S_0, \Pi_k, \mathcal{O}\rangle$ of a Kripke structure M, we call a structure $M'_k = \langle S', S_0, \Pi'_k, \mathcal{O}'\rangle$ a *sub-model* of M_k if $\Pi'_k \subseteq \Pi_k$, $S \subseteq$ *States*(Π'_k) and $\mathcal{O}' = \mathcal{O}|_{S'}$.

The bounded semantics of CTL over *sub-models* is defined after that for k-models.

Algorithm 1 BMC method for CTL

```
Require: A Kripke structure $M = \langle S, S_0, R, \mathcal{O}\rangle$, and a CTL formula $\psi$.
Ensure: The validity check of $\psi$ in $M$.
 1: procedure BMC($M, \psi$)
 2:     let $\phi \leftarrow \neg\psi$
 3:     for $k \leftarrow 1..|M|$ do
 4:         Let $M_k$ be a $k$-model of $M$
 5:         Let $\mathcal{M}^*_k \leftarrow \{M'_k \mid M'_k = \langle S', S_0, \Pi'_k, \mathcal{O}'\rangle \wedge |\Pi'_k| \le f_k(\phi)\}$
 6:         Let $[\![M^\phi]\!]_k \leftarrow$ TRANSLATE($\mathcal{M}^*_k$)
 7:         Let $[\![M, \phi]\!]_k \leftarrow [\![M^\phi]\!]_k \wedge [\![\phi]\!]_{M_k}$
 8:         Check the SATisfiability of $[\![M, \phi]\!]_k$
 9:     end for
10: end procedure
```

5.3 Translating BMC to SAT

Definition 5.8 (Translation of a CTL formula). We denote by $[\![\phi]\!]^s_k$ the propositional translation of CTL formula ϕ at state s.

$$
\begin{aligned}
[\![p]\!]^s_k &:= p(s)\\
[\![\neg p]\!]^s_k &:= \neg p(s)\\
[\![\phi \wedge \psi]\!]^s_k &:= [\![\phi]\!]^s_k \wedge [\![\psi]\!]^s_k\\
[\![\phi \vee \psi]\!]^s_k &:= [\![\phi]\!]^s_k \vee [\![\psi]\!]^s_k\\
[\![\mathsf{EX}\phi]\!]^s_k &:= \bigvee_{s'\in S'} \left(R(s, s') \wedge [\![\phi]\!]^{s'}_k \right)\\
[\![\mathsf{EG}\phi]\!]^s_k &:= \bigvee_{0\le i\le k} \begin{pmatrix} H(s, \pi(i))\\ \wedge \bigvee_{l=0}^{k} L_{k,i}(l)\\ \wedge \bigwedge_{j=0}^{k} [\![\phi]\!]^{\pi(j)}_k \end{pmatrix}\\
[\![\mathsf{E}[\phi \mathsf{U} \psi]]\!]^s_k &:= \bigvee_{0\le i\le k} \begin{pmatrix} H(s, \pi(i))\\ \wedge \bigvee_{j=0}^{k} L_{k,i}(l)\\ \wedge \bigwedge_{j=0}^{k} [\![\phi]\!]^{\pi(j)}_k \end{pmatrix}
\end{aligned}
$$

6. Multi-valued Model-checking

Normal model-checking receives as input a specification composed by a Kripke structure and a temporal logic description of the system's properties. The specification is verified against Kripke structure to see if it holds or not. On the other hand, *multi-valued model-checking* receives an extension of the Kripke structure called *multi-valued Kripke structure*, and the description of the properties is expressed on *multi-valued temporal logic.* Then the specification is verified against multi-valued Kripke structure.

6.1 Boolean Algebras

This section introduces many-valued structures uses as logical domains of interpretation for formulas in multi-valued logics, like mv-CTL, that we use in our work.

Definition 6.1 (Lattice)**.** A *lattice* is a partially ordered set $\mathcal{L} = (L, \sqsubseteq)$ such that for any two elements $x, y \in L$ it is defined:

- Their greatest lower bound $(x \sqcap y)$, called *meet*;
- Their lowest upper bound $(x \sqcup y)$, called *join*;

Alternatively a lattice can be defined in terms of the meet and join operations, so we can speek of a lattice as a structure $\mathcal{L} = (L, \sqcap, \sqcup)$ that satisfies the following laws:

$$\left.\begin{array}{r} x \sqcap y = y \sqcap y \\ x \sqcup y = y \sqcup y \end{array}\right\} \textit{commutative laws}$$

$$\left.\begin{array}{r} (x \sqcap y) \sqcap z = x \sqcap (y \sqcap z) \\ (x \sqcup y) \sqcup z = x \sqcup (y \sqcup z) \end{array}\right\} \textit{associative laws}$$

$$\left.\begin{array}{r} x \sqcup (x \sqcap y) = x \\ x \sqcap (x \sqcup y) = x \end{array}\right\} \textit{absorption laws}$$

$$\left.\begin{array}{r} x \sqcup x = x \\ x \sqcap x = x \end{array}\right\} \textit{identity laws}$$

Definition 6.2 (Quasi-Boolean Algebra)**.** We call a *Quasi-boolean Algebra*, the structure $\mathcal{B} = \langle B, \sqcap, \sqcup, \sim, \top, \bot \rangle$ where $\langle B, \sqcap, \sqcup \rangle$ is a distributive lattice, $\top$ and $\bot$ are the least and the greatest elements and $\sim$ is a unary operation on B such that for every $x \in B$ there exists a unique element $\sim x \in B$ satisfying the following laws for all $x, y \in B$:

$$\left.\begin{array}{r} \sim (a \sqcap b) = \sim a \sqcup \sim b \\ \sim (a \sqcup b) = \sim a \sqcap \sim b \end{array}\right\} \textit{De Morgan}$$

$$\sim\sim a = a \qquad \textit{involution}$$

$$a \sqsubseteq b \iff \neg a \sqsupseteq \neg b \qquad \textit{anti-monotonic}$$

We also define the relation $\sqsubseteq$ by $a \sqsubseteq b$ iff $a \sqcap b = a$.

Definition 6.3 (Boolean Algebra)**.** A *boolean algebra* is a quasi-boolean algebra $\mathcal{B}$ with the additional condition that for every element $x \in B$:

$$x \sqcap \sim x = \bot \qquad \textit{Law of Non-Contradiction}$$

$$x \sqcup \sim x = \top \qquad \textit{Law of Excluded Middle}$$

Figure 1 illustrates a boolean lattice with 16 elements, we say it is an order 4 boolean lattice.

Having introduced the necessary theoretical background, we proceed to define multi-valued sets, relations and Kripke structures which are going to be the foundation of the multi-valued model checking. We define the operations of complement, intersection and backward image over multi-valued sets, e.e. sets that the membership function takes values over a lattice. In this section, the presentation of the material combines results from various sources[7),10),29)] since a common ground for the subject is not yet established.

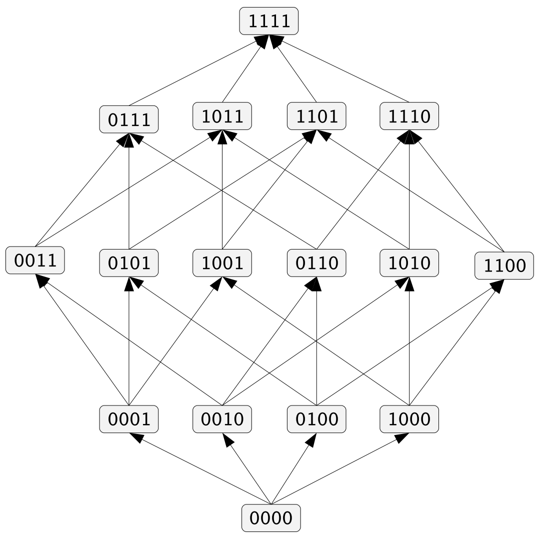

Fig. 1 An order 4 boolean lattice.

6.2 Multi-valued sets and multi-valued relations

In the classic notion of sets, the membership of an object to a set is determined by the set's membership function. Assume that we have a set of object S and that we wanto to define a subset S' of S suc that every object in S' satisfies a property H. Let the membership function of S' be $h : S \to \{\bot, \top\}$ with definition: $h(s) = \top(\textit{true})$ if s satisfies property H and $h(s) = \bot(\textit{false})$ otherwise. Then the collection of objects of S that constitute the subset S' is denoted by $\{s \in S \mid h(s)\}$, which implies that for all the objects $s \in S'$ it is the case that $h(s) = \top$. For example, consider S to be $\mathbb{N}$ and the membership function to be $h(s) = (s \leq 5)$, then the set $S' = \{s \in \mathbb{N} \mid s \leq 5\}$ is $S' = \{0, 1, 2, 3, 4, 5\}$.

The multi-valued sets, denoted by mv-sets, are a straightforward extension of the classical sets. The characteristic function of an mv-set takes values over a lattice instead of the classical 2-valued Boolean set. Intuitively, when the characteristic function is multi-valued it express the degree that an object belongs to the mv-set.

Definition 6.4 (Multi-values set)**.** Let $\mathcal{L} = \langle L, \sqcap, \sqcup \rangle$ be a lattice and S be a set of objects, them a *multi-valued set*, denoted by $\mathbb{S}$, is a total function $\mathbb{S} : S \to L$.

As mv-sets are, in fact, functions, $\mathbb{S}(x)$ denotes the degree of membership of x in $\mathbb{S}$. Next, we will define the operations of union ($\cup_L$), intersection ($\cap_L$), set-inclusion ($\subseteq_L$) and equality for mv-sets using the lattice join and meet operations.

Definition 6.5 (mv-union, mv-intersection, mv-set inclusion, mv-equality). Let $\mathcal{L} = \langle L, \sqcap, \sqcup \rangle$ be a lattice, then we define:

mv-Intersection:

$$(\mathbb{S} \cap_L \mathbb{S}')(x) := \mathbb{S}(x) \sqcap \mathbb{S}'(x)$$

mv-Union:

$$(\mathbb{S} \cup_L \mathbb{S}')(x) := \mathbb{S}(x) \sqcup \mathbb{S}'(x)$$

mv-Set inclusion:

$$(\mathbb{S} \subseteq_L \mathbb{S}') := \forall_x.(\mathbb{S}(x) \sqsubseteq \mathbb{S}'(x))$$

mv-Equality:

$$(\mathbb{S} =_L \mathbb{S}') := \forall_x.(\mathbb{S}(x) = \mathbb{S}'(x))$$

Definition 6.6 (mv-complement, De Morgan, mv-antimonotonicity). Let $\mathcal{B} = \langle B, \sqcap, \sqcup, \sim, \bot, \top \rangle$ be an algebra, then the multi-valued set will be the total function $\mathbb{S} : S \to B$. When the values of the set are over an albebra B, then we denote the mv operations of Definition 6.5 using the subscript B. Now, we can define the mv-set complement operation using the algebra's complement operation and, also, derive the De Morgan laws:

mv-Complement:

$$\overline{\mathbb{S}}(x) :=\sim \mathbb{S}(x)$$

De Morgan:

$$\overline{\mathbb{S} \cap_B \mathbb{S}'} := \overline{\mathbb{S}} \cup_B \overline{\mathbb{S}'}$$
$$\overline{\mathbb{S} \cup_B \mathbb{S}'} := \overline{\mathbb{S}} \cap_B \overline{\mathbb{S}'}$$

mv-Antimonotonicity:

$$\mathbb{S} \subseteq_L \mathbb{S}' := \overline{\mathbb{S}'} \subseteq_B \overline{\mathbb{S}}$$

Note that all the above definitions actually follow the definitions for the algebraization of the classical 2-valued logic. Hence, in the special case where the algebra is over the the lattice $\mathcal{L}_2$ we get the classical 2-valued set theory. Now that we have established the notion of mv-sets, we proceed to define multi-valued relations, or mv-relations. Defining mv-relations is important as they are necessary for defining Kripke structures with multi-valued transition relations.

Definition 6.7 (Multi-valued relations). A *multi-valued relation* $\mathbb{R}$ on sets S and T over a lattice $\mathcal{L}$ is a function $\mathbb{R} : S \times T \to L$.

6.3 Multi-valued Kripke structure

The extension of the classical notion of Kripke structures to the multi-valued ones (mv-Kripke structures) is straightforward. Note that some authors perform multi-valued model checking on mv-Kripke structures where the predicates take values from an mv-algebra[7),13)], but they keep the transition relation defined over $\mathcal{B}_2$, while others consider also mv-transition relations. In the following we will denote the multi-valued Kripke structures my M.

Definition 6.8 (Multi-Valued Kripke Structure). We call the t-uple $M = \langle S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathcal{L} \langle$ a *Multi-Valued Kripke Structure* with components defined as follows:

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

We assume $\mathbb{R}$ to be total (as it is usually the case), i.e. $Dom(\mathbb{R}) = S \times S$, and it is defined even for the cases where for $s, t \in \mathbb{R}$ we have $\mathbb{R}(s, t) = \bot$.

Also, for the multi-valued case we need to slightly modify the definition of path sets.

Definition 6.9 (Paths on mv-Kripke structures). Let $M = \langle S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathcal{L} \rangle$ be a mv-Kripke structure, the for each $s \in S$:
$\Pi_M^{all}(s) := \{\pi : \mathbb{N} \to S \mid (\pi(0) = s \wedge \forall_{i \in \mathbb{N}}.(\mathbb{R}(\pi(i), \pi(i+1)) \neq \bot)$

6.4 Multi-valued CTL – mvCTL

The syntax of mv-CTL is basically the same as the 2-valued CTL, except that beyond $\bot$ and $\top$ we also accept the other lattice values $l \in L$ as mv-CTL literal formulas. We now move to the definition of the mv-CTL semantics.

Definition 6.10 (mv-CTL Semantics). Given a mv-Kripke structure $M = \langle S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathcal{L} \rangle$, we denote by $||\phi||_M^s : S \to L$ the degree that a state $s \in S$ satisfies a specification ϕ in the mv-Kripke structure (model) M. The the multi-valued semantics of the core operators of mv-CTL are defined as shown in **Fig. 2**.

6.5 mv-CTL Bounded Semantics

Definition 6.11 (mv-path). Given a mv-Kripke structure $M = \langle S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathcal{L} \rangle$, we call $\Pi_M(s)$ the set of *mv-paths* of M, provided:

$$\Pi_M(s) = \{\pi : \mathbb{N} \to S \mid \pi(0) = s \wedge \\ \forall_{i \in \mathbb{N}}.(\mathbb{R}(\pi(i), \pi(i+1)) \sqsubseteq \bot)\}$$

Definition 6.12 (mv-loop-path). Given a mv-

$$
\begin{aligned}
||l||^s_M &:= l \text{ for } l \in L\\
||p||^s_M &:= \mathbb{O}(s,p) \text{ for } p \in AP\\
||\neg\phi||^s_M &:= \sim ||\phi||^s_M\\
||\phi \vee \psi||^s_M &:= ||\phi||^s_M \sqcup ||\psi||^s_M\\
||\mathsf{EX}\phi||^s_M &:= \bigsqcup_{s' \in S} \left(R(s,s') \sqcap ||\phi||^{s'}_M\right)\\
||\mathsf{EG}\phi||^s_M &:= ||\phi||^s_M \sqcap \bigsqcup_{s' \in S} \left(R(s,s') \sqcap ||\mathsf{EG}\phi||^{s'}_M\right)\\
||\mathsf{E}[\phi\mathsf{U}\psi]||^s_M &:= ||\psi||^s_M \sqcup \left(||\psi||^s_M \sqcap \bigsqcup_{s' \in S} \left(R(s,s') \sqcap ||\mathsf{E}[\phi\mathsf{U}\psi]||^{s'}_M\right)\right)
\end{aligned}
$$

Fig. 2 Semantic for mv-CTL.

Kripke structure $M = \langle S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathcal{L}\rangle$ be, we call the set of *mv-loop-paths* of M, provided:

$$
\begin{aligned}
\Pi^{k,l}_M(s) = \{\pi : \mathbb{N} \to S \mid \pi \in \Pi_M(s) \wedge\\
\exists_{0 \le l \le k}.(\mathbb{R}(\pi(l), \pi(k)) \sqsubseteq \bot)\}
\end{aligned}
$$

Definition 6.13 (Bounded Semantics). Let M be a mv-Kripke structure $\langle S, S_0, \mathbb{R}, \mathbb{O}, \mathcal{L}\rangle$, $k \in \mathbb{N}$ be a bound, and ϕ and ψ be CTL formulas. $M_k, s \models \phi$ denotes that ϕ is true at state s of M under a bounded interpretation with bound k, or equivalently, that ϕ is true at state s of the k-model M_k. The relation $\models$ is inductively defined as shown in **Fig. 3**.

7. Prototype Implementation

On this section we describe our prototype implementation of multi-valued bounded model checking. This first attempt is far from complete and must be seen as a work in progress. For this prototype two decisions were made in order to reduce the time frame necessary for the implementation. First we decided to work with *boolean algebras*. Although we aim for more general logic structures, boolean algebras provided both, a flexible and expressive language to express specifications, and a direct path for a fast implementation. Secondly we decide to translate our models to an underlying tools, instead of implementing the low level BMC algorithm directly. This strategy, namely, to translate a multi-valued model-checking problem to a standard 2-valued one, was already explored by Konikowska *et al.*[28],[29], but we make note that we use a slightly different approach in our method.

We have implemented our prototype over NuSMV[16] and our method can be summarized as follows. We encode a boolean algebra with 2^n values in a vector of n bits. An extension of the NuSMV syntax was 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 `#`$d_1 \dots d_n$ 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 29), except that the set of designated values is not bound to the model, but to individual specifications.

The role of the special 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. 4**, where the transitions of the model are labeled with 3-bit lattice values, indicating 3 composing layers.

Figure 5 illustrate the use of the special

$$
\begin{aligned}
||l||_M^{s,k} &:= l \quad \text{for } l \in L \\
||p||_M^{s,k} &:= \mathbb{O}(s,p) \quad \text{for } p \in AP \\
||\neg\phi||_M^{s,k} &:= \sim ||\neg\phi||_M^{s,k} \\
||\phi \vee \psi||_M^{s,k} &:= ||\phi||_M^{s,k} \sqcup ||\psi||_M^{s,k} \\
||\mathsf{EX}\phi||_M^{s,k} &:= \begin{cases} \bigsqcup_{\pi \in \Pi_M(s)} \left(R(\pi(0),\pi(1)) \sqcap ||\phi||_M^{\pi(1),k-1} \right) & \text{if } k > 0 \\ \bot & \text{otherwise} \end{cases} \\
||\mathsf{EG}\phi||_M^{s,k} &:= \begin{cases} \bigsqcup_{\pi \in \Pi_M^{k,l}(s)} \left(\bigsqcap_{i=0}^{k-1} \left(R(\pi(i),\pi(i+1)) \sqcap ||\phi||_M^{\pi(i),k-i} \right) \right) & \text{if } k > 0 \\ \bot & \text{otherwise} \end{cases} \\
||\mathsf{E}[\phi \mathsf{U} \psi]||_M^{s,k} &:= \bigsqcup_{\pi \in \Pi_M^{k,l}(s)} \left(\bigsqcap_{i=0}^{k-1} \left(\bigsqcup_{j=0}^{k} \left(||\psi||_M^{\pi(j),k-j} \sqcap \bigsqcup_{i}^{j-1} R(\pi(i),\pi(i+1)) \sqcap ||\phi||_M^{\pi(i),k-i} \right) \right) \right)
\end{aligned}
$$

Fig. 3 Bounded semantic for mv-CTL.

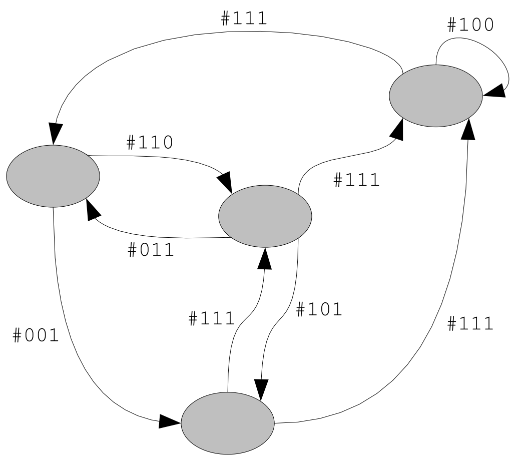

Fig. 4 Model with multiple viewpoints incorporated.

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 2. 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. Algorithms 3, 4 and 5 deal with translation of general expressions, conditional expressions and specifications, respectively.

The translation for specifications, in Algorithm 5, take in account that a specification is satisfiable if it's interpretation issues a value greater or equal the lattice value associated with it. So, in translating the specification to 2-valued CTL we construct a boolean expression that grants this meaning.

Algorithm 2 Translate the multi-valued model to a 2-valued one.

```
Require: Multi-valued model $M_n$, where
    $M_k = \langle V_n, I_n, T_n, \Phi_n, \mathcal{L}_n \rangle$.
Ensure: The corresponding 2-valued model
    $M_2 = \langle V_2, I_2, T_2, \Phi_2 \rangle$.
 1: function TranslateModel($M_n$)
 2:     let $\langle V_2, I_2, T_2, \Phi_2 \rangle = \langle [], [], [], [] \rangle$
 3:     for all $v \in V_n$ do
 4:         let $V_2 = V_2 \cup [\mathsf{Var}(v,i) | 1 \le i \le n]$
 5:     end for
 6:     for all $e \in I_n$ do
 7:         let $[e_1, \ldots, e_m] = \mathsf{TrItExpr}(e)$
 8:         let $I_2 = I_2 \cup [e_1, \ldots, e_m]$
 9:     end for
10:     for all $e \in T_n$ do
11:         let $[e_1, \ldots, e_m] = \mathsf{TrItExpr}(e)$
12:         let $T_2 = T_2 \cup [e_1, \ldots, e_m]$
13:     end for
14:     for all $q \in \Phi_n$ do
15:         let $[\phi_1, \ldots, \phi_n] = \mathsf{TrItSpec}(q)$
16:         let $\Phi_2 = \Phi_2 \cup [\phi_1, \ldots, \phi_n]$
17:     end for
18:     return $\langle V_2, I_2, T_2, S_2 \rangle$
19: end function
```

8. Conclusion and Future Work

We have shown that the application of Bounded Model Checking to systems models build over multi-valued structures based on boolean algebras is possible, and that the trans-

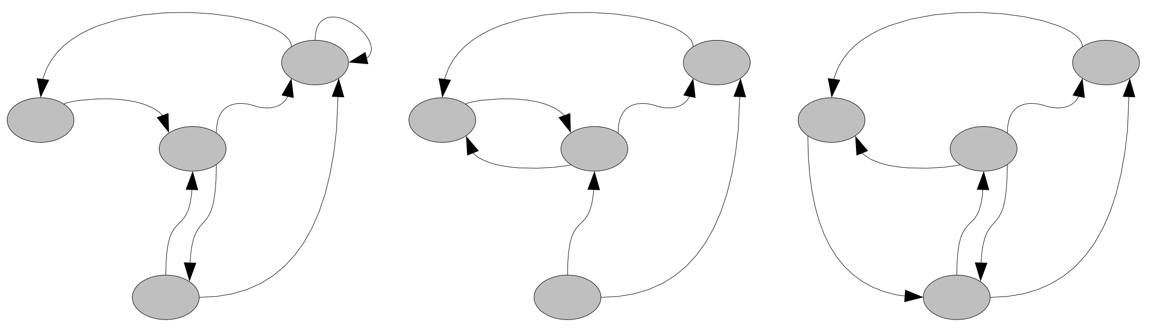
(a) $TR(100) (b) $TR(010) (c) $TR(001)

Fig. 5 Three different *"slices"* of the original model.

Algorithm 3 Translate a multi-valued expression to a list of 2-valued ones.

```
Require: A valid multi-valued expression $e$, over a boolean lattice of order $n$ .
Ensure: A list of $m \leq n$ 2-valued expressions.
function TrltExpr($e$)
    if $e$ is a variable then
        return $[\mathsf{Var}(e,i) | 1 \leq i \leq n]$
    else if $e$ is a lattice value then
        return $[\mathsf{Bit}(e,i) | 1 \leq i \leq n]$
    else if $e$ is like $\langle a \odot b \rangle$ then
        let $[a_1, \ldots, a_m] = \mathsf{TrltSpec}(a)$
        let $[b_1, \ldots, b_m] = \mathsf{TrltSpec}(b)$
        return $[\langle a_i \odot b_i \rangle | 1 \leq i \leq m]$
    else if $e$ is like $\langle \odot a \rangle$ then
        let $[a_1, \ldots, a_m] = \mathsf{TrltSpec}(a)$
        return $[\langle \odot a_i \rangle | 1 \leq i \leq m]$
    else if $e$ is a conditional then
        return $\mathsf{TrltCond}(e)$
    end if
end function
```

Algorithm 4 Translate conditional multi-valued expressions.

```
Require: A multi-valued conditional expression $C_n = [c_1, \ldots, c_l]$, composed by a sequence of $l$ clauses. Each clause $c_i = \langle e_{ti}, e_{ri}$ is a pair composed by a test expression $e_{ti}$ and a result expression $e_{ri}$.
Ensure: A sequence $[C_{2_1}, \ldots, C_{2_n}]$ of 2-valued conditional expressions, translating $C_n$.
function TrltCond($C$)
    let $[C_{2_1}, \ldots, C_{2_n}] = [[], \ldots, []]$
    for all $\langle e_{t_i}, e_{r_i} \rangle \in C_n$ do
        let $[e_{t_{i_1}}, \ldots, e_{t_{i_l}}] = \mathsf{TrltSpec}(e_{t_i})$
        let $[e_{r_{i_1}}, \ldots, e_{r_{i_l}}] = \mathsf{TrltSpec}(e_{r_i})$
        for $j \leftarrow 1..n$ do
            $C_{2_i} \leftarrow C_{2_i} \cup [\langle e_{t_{i_j}}, e_{r_{i_j}} \rangle]$
        end for
    end for
    return $[C_{2_1}, \ldots, C_{2_n}]$
end function
```

lation of the multi-valued models to 2-valued models is very straightforward. Furthermore, we have shown that this approach leads to a very suitable method of dealing with incosistent viewpoints over specifications and allows the construction of composed models from this different viewpoints.

We aim to keep the work on this line of investigation. Topics that can be further investigated our tool and future, improved, versions include:

(1) Possible means for software engineers to express requirements in a more precise way, expressing the uncertainties or disagreements about them. The possibility to find counter-examples will later make possible to demonstrate, if necessary, that some of these disagreements must be solved and/or some uncertainties must be dismissed to grant the validity of the specifications.

(2) The capacity of bounded model checking to efficiently find counter-examples and the possibility that multi-valued temporal logic provides to express disagreement will certainly provide powerful tools to reason about model refinement.

(3) The possibility to replace parts of a model for a "macro-state" with all logic variables having value "undetermined" and check the resulting model against the specification can lead to new techniques model abstraction and partial model ver-

Algorithm 5 Translation for multi-valued CTL specifications.

Require: A specification $q = \langle \phi, l \rangle$. That is a pair of a multi-valued CTL formula ϕ and a lattice value l that states the *acceptable degree* for the specification.
Ensure: The composed 2-valued CTL formula that translates q.

```
function TrltSpec($q$)
    let $[\phi_1, \ldots, \phi_n] = \mathsf{TrltExpr}(\phi)$
    for all $i \leftarrow 1..n$ do
        let $q_i = \langle\rangle$
        for all $j \leftarrow 1..(i-1)$ do
            if $l^i = 1$ then
                let $q_i = \langle q_i \wedge \phi_i \rangle$
            end if
        end for
    end for
    return $\langle q_1 \vee \ldots \vee q_n \rangle$
end function
```

ification.

References

1) H.R. Andersen. An introduction to binary decision diagrams. Lecture notes, 1997.
2) B.Beard, M.Bidoit, A.Finkel, F.Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, and P.McKenzie. *Systems and software verification: model-checking techniques and tools.* Springer-Verlag, 2001.
3) A.Biere, A.Cimatti, E.Clarke, O.Strichman, and Y.Zhu. Bounded model checking, 2003.
4) A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without bdds. *Lecture Notes in Computer Science*, 1579:193–207, 1999.
5) G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Computer Aided Verification*, pages 274–287, 1999.
6) G. Bruns and P. Godefroid. Generalized model checking: Reasoning about partial state spaces. *Lecture Notes in Computer Science*, 1877:168+, 2000.
7) G. Bruns and P. Godefroid. Model checking with multi-valued logics, 2003.
8) R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.
9) R.E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
10) M.Chechik, B.Devereaux, S.Easterbrook, and A. Gurfinkel. Multi-valued symbolic model-checking. *ACM Transaction on Software Engineering and Methodology*, 2(4):371–408, 2003.
11) M. Chechik, B. Devereaux, S. Easterbrook, Y.C. Lai, and V.Petrovykh. Eficient multiple-valued model-checking using lattice representations. *Lecture Notes in Computer Science*, 2154:441–455, 2001.
12) M.Chechik, B.Devereux, and S.Easterbrook. Implementing a multi-valued symbolic model checker. In *Proceedings of 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 2031 of *Lecture Notes in Computer Science*, pages 404–419. Springer, 2001.
13) M. Chechik, B. Devereux, and A. Gurfinkel. Model-checking infinite state-space systems with fine-grained abstractions using SPIN. *Lecture Notes in Computer Science*, 2057:16+, 2001.
14) M.Chechik, A.Gurfinkel, B.Devereux, A.Lai, and S.Easterbrook. Data structures for symbolic multi-valued model-checking. *Form. Methods Syst. Des.*, 29(3):295–344, 2006.
15) M. Chechik and W. MacCaull. Ctl model-checking over logics with nonclassical negations, 2003.
16) A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking, 2002.
17) E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Computational challenges in bounded model checking. *Software Tools for Technology Transfer (STTT)*, 7(2):174–183, April 2005.
18) E.M. Clarke, A.Biere, R.Raimi, and Y.Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
19) E.M. Clarke, O.Grumberg, and D.A. Peled. *Model Checking.* The MIT Press, 1999.
20) M. W. M. Cong Liu, Andreas Kuehlmann. Cama: A multi-valued satisfiability solver. In *International Conference on Computer Aided Design*, pages 326 – 333. IEEE/ACM, November 2003.
21) G. E. Fainekos. An introduction to multi-valued model checking. Technical report, Department of Computer and Information Science, University of Pennsylvania, 2005.
22) M.C. Fitting. Many-valued modal logics. *Fundamenta Informaticae*, XV:235–254, 1991.
23) M.C. Fitting. Many-valued modal logics II. In *Proc.LFCS'92.* Springer-Verlag, 1992.

24) A. Gurfinkel. Multi-valued symbolic model-checking: fairness, counter-examples, running time. Master's thesis, Department of Computer Science, University of Toronto, 2003.
25) A.Gurfinkel and M.Chechik. Generating counterexamples for multi-valued model-checking, 2003.
26) M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridg University Press, Cambridg, UK, 2nd edition, 2004.
27) J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.
28) B. Konikowska and W. Penczek. Reducing model checking from multi-valued ctl* to ctl*, 2002.
29) M.C. Konikowska. On designated values in multi-valued ctl* model checking, 2004.
30) H.C. Li, S.Krishnamurthi, and K.Fisler. Modular verification of open features using three-valued model checking. *Automated Software Engg.*, 12(3):349–382, 2005.
31) W. Penczek, B. Wo'zna, and A. Zbrzezny. Bounded model checking for the universal fragment of ctl, 2002.
32) F.Somenzi. Binary decision diagrams, 1999.
33) G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical Report 7007.011, National Institute of Standards and Technology – NIST, May 2002.
34) G.Winskel. *The formal semantics of programming languages: an introduction.* MIT Press, 1993.