Multi-Valued Symbolic Model-Checking

MARSHA CHECHIK, BENET DEVEREUX, STEVE EASTERBROOK AND ARIE GURFINKEL
University of Toronto

This paper introduces the concept of multi-valued model-checking and describes an implementation of a multi-valued symbolic model-checker, χChek. Multi-valued model-checking is a generalization of classical model-checking, useful for analyzing models that contain *uncertainty* (lack of essential information) or *inconsistency* (contradictory information often occurring when information is gathered from multiple sources). Multi-valued logics support the explicit modeling of uncertainty and disagreement by providing additional truth values in the logic.

This paper provides a theoretical basis for multi-valued model-checking and discusses some of its applications. A companion paper [Chechik et al., 2001d] describes implementation issues in detail. The model-checker works for any member of a large class of multi-valued logics. Our modeling language is based on a generalization of Kripke structures, where both atomic propositions and transitions between states may take any of the truth values of a given multi-valued logic. Properties are expressed in χCTL, our multi-valued extension of the temporal logic CTL.

We define the class of logics, present the theory of multi-valued sets and multi-valued relations used in our model-checking algorithm, and define the multi-valued extensions of CTL and Kripke structures. We prove correctness by showing that our definition of χCTL preserves the expected properties, and that the model-checking algorithm is guaranteed to terminate. We also address the use of fairness in multi-valued model-checking. Finally, we discuss some applications of the multi-valued model-checking approach.

Categories and Subject Descriptors: []:

General Terms: Documentation, Verification

Additional Key Words and Phrases: CTL, multi-valued logic, model-checking, partiality, inconsistency, fairness, χChek.

1. INTRODUCTION

This paper introduces the concept of multi-valued model checking, and describes an implementation of a multi-valued symbolic model-checker, χChek. χChek is a generalization of an existing symbolic model-checking algorithm [McMillan, 1993], for a multi-valued extension of the temporal logic CTL. The paper describes both a general theory of multi-valued model-checking, and our specific implementation, for which we provide a proof of correctness and an analysis of running time.

Classical model-checking takes a model, M, of a system (expressed as a finite state machine), and a behavioral correctness property, φ, (expressed as a formula in a suitable temporal logic), and determines whether or not the model satisfies the property [Clarke

Authors' address: Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada. Email: {`chechik`, `benet`, `sme`, `arie`}@`cs.toronto.edu`.

et al., 1986], i.e. it returns the value of $M \models \varphi$. Model-checking has been effectively applied to reasoning about correctness of hardware [Clarke et al., 1993], communication protocols [Holzmann, 1991], and software requirements [Atlee and Gannon, 1993]. A number of classical model-checkers are currently used for industrial applications, including SPIN [Holzmann, 1997], SMV [McMillan, 1993], and Murϕ [Dill, 1996]. We give a brief overview of classical CTL model checking in Section 3.

In multi-valued model-checking, the satisfaction relation is multi-valued. For a given property and a given model, the model checker returns one of a number of truth values drawn from a multi-valued logic. In our work, the truth values of the logic must form a (finite) lattice under the truth ordering, where the top and bottom of the lattice can be interpreted as TRUE and FALSE, respectively.

Multi-valued model-checkers have many potential applications, as they support the explicit modeling of uncertainty, disagreement, and relative priority. We provide a brief motivating example in Section 2, in which intermediate values of the logic represent disagreements that arise when we compose two models drawn from different sources. By checking a model in which disagreements are explicitly represented, we can reason about how such disagreements affect various behavioral properties. Some additional applications of multi-valued model-checking are described in Section 8.

The advantages of multi-valued model-checking have been recognized by a number of researchers. For example, 3-valued logic has been used for interpreting results of model-checking with abstract interpretation [Chechik and Ding, 2001; Sagiv et al., 1999], and for analyzing partial models [Bruns and Godefroid, 1999; Bruns and Godefroid, 2000]. Bruns and Godefroid also show that μ-calculus model-checking on 3-valued predicates reduces to classical model-checking. 4-valued logic has been used for reasoning about abstractions of detailed gate or switch-level designs of circuits [Hazelhurst, 1996; Hazelhurst, 1999]. The work described in this paper generalizes these approaches, by providing a model-checker that works for any member of a large class of multi-valued logics.

This paper makes a number of contributions to the study of multi-valued model-checking. First, we identify a useful class of well-behaved multi-valued logics: those in which the truth values form quasi-boolean lattices. This class of logics includes classical boolean logic, as well as the 3- and 4-valued logics described in the literature. We define this class of logics and discuss its properties in Section 4.

Second, we describe a theory of multi-valued sets and multi-valued relations, which we use to provide the core data structures of our symbolic model-checking algorithm. Multi-valued sets are sets whose membership functions are multi-valued. We use a multi-valued set to represent the partition of the state-space over the set of truth values in the logic, induced by a given property. We extend the notion of multi-valued membership to multi-valued relations, which we use to represent the transition relations in our models. Our characterization of multi-valued sets draws on the theory of fuzzy sets, but differs from fuzzy sets in that the set of values is finite. We present the theory of multi-valued sets and relations in Section 5.

Third, we define a multi-valued semantics for CTL, and demonstrate that this semantics preserves the expected properties, including the fixpoint characterizations of the temporal operators. We call the resulting logic χCTL. We provide a model-based semantics for χCTL by extending the notion of Kripke structures, referring to them as χKripke structures, so that both atomic propositions and transitions between states range over values of a multi-valued logic. We present χCTL and χKripke structures in Section 6. We then

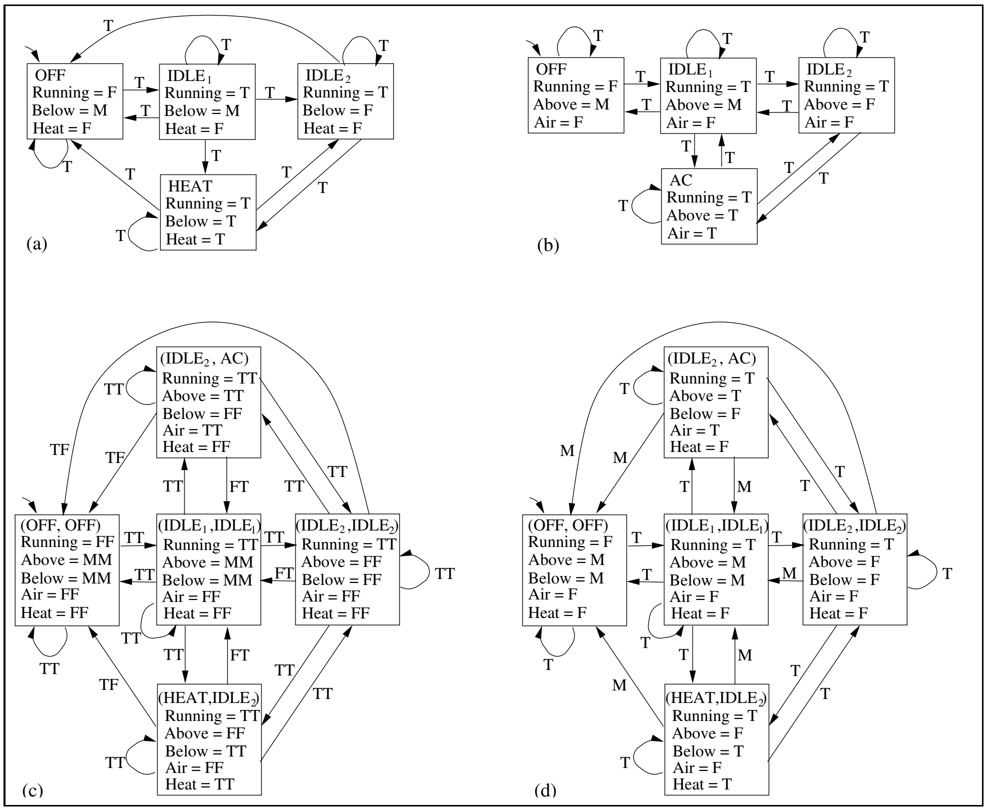

Fig. 1. Models of the thermostat. (a) Heater Aspect; (b) AC Aspect; (c) Thermo$_1$: a combined model over logic **3x3**; (d) Thermo$_2$: a combined model over logic **3**.

prove that for this definition of χCTL, our symbolic model-checking algorithm is correct and is guaranteed to terminate.

Finally, we give a characterization of multi-valued model-checking with fairness. Fairness is used in classical model-checking to simplify modeling, by allowing the user to build a model with more behaviors than is desired, and then to restrict the analysis to just those behaviors that are fair. We argue that fairness conditions in multi-valued model-checking should be boolean-valued and give a formulation of fairness for each χCTL operator in Section 7.

We describe the implementation details and some potential applications of multi-valued model-checking in Section 8. We conclude the paper with a brief discussion of the relationship between our work and other recent work on multi-valued model-checking, and sketch out a research agenda for further work in this area (Section 9).

2. MOTIVATION AND EXAMPLE

Consider the four models for a simple thermostat system given in Figure 1. Figure 1(a) describes an aspect of the system concerned with running a heater if the temperature falls below desired. The system has a switch to turn it off and on (`Running`), one temperature indicator (`Below`), and a variable indicating whether the heater is on (`Heat`). The system starts in state[1] OFF and transitions to $\mathtt{IDLE}_1$ when it is turned on, where it awaits the reading of the temperature indicator. Once the temperature is determined, the system transitions either to $\mathtt{IDLE}_2$ or to HEAT. The value of the temperature indicator is unknown in states OFF and $\mathtt{IDLE}_1$. To model this, we could duplicate these states, assigning Below the value T in one copy and F in the other — the route typically taken by classical model-checkers. Alternatively, we can model the system using the 3-valued logic: T, F and M (MAYBE), assigning Below the value M, as shown in Figure 1(a).

We can check this Heater Aspect model against some properties:

Prop. 1. Can the system transition into $\mathtt{IDLE}_1$ from everywhere?
Prop. 2. Is the heat ever turned on before the temperature falls below desired?
Prop. 3. Is it always possible to turn the system off?

Figure 1(b) describes another aspect of the thermostat system – running the air conditioner. This is similar to the Heater Aspect: it has a variable indicating whether the system is on or off (Running), an indicator (Above) to check whether the temperature is above desired, and a variable indicating whether the air conditioning is on (Air). The system starts in state OFF and transitions to $\mathtt{IDLE}_1$, awaiting the temperature reading. Then it goes to $\mathtt{IDLE}_2$ or to AC. However, this model behaves differently from the Heater Aspect when the system is turned off. Rather than going to state OFF directly, it passes through the transient state $\mathtt{IDLE}_1$ where the necessary reset of the system monitors is performed before turning the system off.

To analyze the interaction of the two aspects, we need to decide how they should be composed. In [Easterbrook et al., 2001] we presented a framework for defining various types of composition. Here we describe a composition in which multi-valued logics allow us to explicitly model and analyze disagreement. We choose to make the powered down state (OFF) and the initialization state ($\mathtt{IDLE}_1$) common across the two aspects. For the remaining states, we form the cross product. However, we also add an environmental assumption that $\mathtt{Above} \wedge \mathtt{Below}$ is FALSE, making the state $(\mathtt{HEAT}, \mathtt{AC})$ unreachable in the composition. Thus, the possible states in the combined model are $(\mathtt{OFF}, \mathtt{OFF})$, $(\mathtt{IDLE}_1, \mathtt{IDLE}_1)$, $(\mathtt{IDLE}_2, \mathtt{IDLE}_2)$, $(\mathtt{IDLE}_2, \mathtt{AC})$, $(\mathtt{HEAT}, \mathtt{IDLE}_2)$.

As a logic for the composition, we choose the product of the individual logics, giving us a 9-valued logic. Values of variables in the merged state are computed as follows: a value of a shared variable is a tuple formed from values of this variable in the original aspects. For example, the value of Running in state $(\mathtt{OFF}, \mathtt{OFF})$ is (T,T) which we write as TT. A value of a variable that is local to one aspect is a tuple where all elements are equal to the value this variable has in the "host" aspect. For example, the value of Below in state $(\mathtt{IDLE}_2, \mathtt{AC})$ is MM because Below has value M in the Heater Aspect and is not present in the AC Aspect. Transitions between states can also have values other than T and F. Transitions between states (s_1, t_1) and (s_2, t_2) are defined as $(R_1(s_1, s_2), R_2(t_1, t_2))$, where $R_i(x, y)$ is the value of the transition between states x and y in system i. For example, the transition between $(\mathtt{IDLE}_2, \mathtt{IDLE}_2)$ and $(\mathtt{IDLE}_1, \mathtt{IDLE}_1)$ is FT because the transition between $\mathtt{IDLE}_2$ and $\mathtt{IDLE}_1$ in the Heater Aspect is F and in the AC Aspect is T. This value models disagreement between the two aspects on the value of the transition. We annotate transitions with their values, omitting FF transitions. The resulting composition, called Thermo_1, is shown in Figure 1(c).

[1] Throughout this paper state labels are capitalized. Thus, HEAT is a state and Heat is a variable name.

We can check Thermo_1 against properties that cannot be expressed in either Heater or AC Aspect alone, e.g.

Prop. 4. Is heat on only if air conditioning is off?
Prop. 5. Is it always true that the heater never runs when the temperature is above desired?

Other compositions are possible, using other multi-valued logics. For example, we can abstract the 9-valued logic to a 3-valued logic by mapping TT to T, FF to F, and everything else to M. The resulting composition, Thermo_2, is shown in Figure 1(d). Logic abstraction is similar to model abstraction, heavily discussed in the literature [Cousot and Cousot, 1999; Visser et al., 2000; Dams, 1996]: abstraction is applied to make the problem more tractable, sometimes at the expense of the ability to draw certain conclusions about the individual models. For example, we cannot conclude from Thermo_2 that the Heater Aspect and the AC Aspect each have a (TRUE) path from $(\texttt{IDLE}_2, \texttt{AC})$ to $(\texttt{IDLE}_1, \texttt{IDLE}_1)$ because all paths between these states go through an M transition. However, in Thermo_1 we know that the path through state $(\texttt{OFF}, \texttt{OFF})$ has value TF, and thus the path is present in the Heater Aspect; we also know that the direct path has value FT, and thus is present in the AC Aspect.

3. CTL MODEL-CHECKING

In this section, we give a brief overview of classical CTL model checking. CTL model-checking is an automatic technique for verifying properties expressed in a propositional branching-time temporal logic called *Computation Tree Logic* (CTL) [Clarke et al., 1986]. A model is a Kripke structure, and properties are evaluated on a tree of infinite computations produced by the model. The standard notation $M, s \models \varphi$, indicates that a formula φ holds in a state s of a model M. If a formula holds in the initial state, it is considered to hold in the model.

A Kripke structure consists of a set of states S, a transition relation $R \subseteq S \times S$, an initial state $s_0 \in S$, a set of atomic propositions A, and a labeling function $I : S \to 2^A$. R must be total, i.e, $\forall s \in S,\ \exists t \in S$, s.t, $(s, t) \in R$. Finite computations are modeled by adding a self-loop to the final state of the computation. For each $s \in S$, the labeling function provides a list of atomic propositions which hold in the state S.

CTL is defined as follows:

(1) Every atomic proposition $a \in A$ is a CTL formula.

(2) If φ and ψ are CTL formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $EX\varphi$, $AX\varphi$, $EF\varphi$, $AF\varphi$, $E[\varphi\ U\ \psi]$, $A[\varphi\ U\ \psi]$, $AG\varphi$, $EG\varphi$.

The logic connectives $\neg$, $\wedge$ and $\vee$ have their usual meanings. The existential and universal quantifiers E and A are used to quantify over paths. The operator X means "in the next state", F represents "sometime in the future", U is "until", and G is "globally". For example, $EX\varphi$ is TRUE in state s if φ holds in some immediate successor of s, while $AX\varphi$ is TRUE if φ holds in every immediate successor of s. $EF\varphi$ is TRUE in s if φ holds in the future along some path from s; $E[\varphi\ U\ \psi]$ is TRUE in s if along some path from s, φ continuously holds until ψ becomes TRUE. $EG\varphi$ hold in s if φ holds in every state along some path from s. $AF\varphi$, $A[\varphi\ U\ \psi]$ and $AG\varphi$ are defined similarly, replacing the quantification over some paths by the one over all paths. Formally,

$$\begin{array}{rcll} AG\varphi & = & \nu Z.\varphi \wedge AXZ & (AG \text{ fixpoint}) \\ EG\varphi & = & \nu Z.\varphi \wedge EXZ & (EG \text{ fixpoint}) \\ AF\varphi & = & \mu Z.\varphi \vee AXZ & (AF \text{ fixpoint}) \\ EF\varphi & = & \mu Z.\varphi \vee EXZ & (EF \text{ fixpoint}) \\ A[\varphi\ U\ \psi] & = & \mu Z.\psi \vee (\varphi \wedge AXZ) & (AU \text{ fixpoint}) \\ E[\varphi\ U\ \psi] & = & \mu Z.\psi \vee (\varphi \wedge EXZ) & (EU \text{ fixpoint}) \end{array}$$

Fig. 2. Fixpoint formulations of CTL operators.

$$\begin{array}{rll} M, s \models a & \text{iff} & a \in I(s) \\ M, s \models \neg\varphi & \text{iff} & M, s \not\models \varphi \\ M, s \models \varphi \wedge \psi & \text{iff} & M, s \models \varphi \ \wedge\ M, s \models \psi \\ M, s \models \varphi \vee \psi & \text{iff} & M, s \models \varphi \ \vee\ M, s \models \psi \\ M, s \models EX\varphi & \text{iff} & \exists t \in S,\ (s,t) \in R \ \wedge\ M, t \models \varphi \\ M, s_i \models EG\varphi & \text{iff} & \text{there exists some path } s_i, s_{i+1}, \ldots \text{ s.t. } \forall j \geq i \cdot M, s_j \models \varphi \\ M, s_i \models E[\varphi\ U\ \psi] & \text{iff} & \text{there exists some path } s_i, s_{i+1}, \ldots, \text{ s.t.} \\ & & \exists j \geq i \cdot M, s_j \models \psi \ \wedge\ \forall k \cdot i \leq k < j \Rightarrow M, s_k \models \varphi \end{array}$$

where the remaining operators are defined as follows:

$$\begin{array}{lll} A[\varphi\ U\ \psi] & \triangleq \neg E[\neg\psi\ U\ \neg\varphi \wedge \neg\psi] \ \wedge\ \neg EG\neg\psi & (\text{def. of } AU) \\ AX\varphi & \triangleq \neg EX\neg\varphi & (\text{def. of } AX) \\ AF\varphi & \triangleq A[\top\ U\ \varphi] & (\text{def. of } AF) \\ EF\varphi & \triangleq E[\top\ U\ \varphi] & (\text{def. of } EF) \\ AG\varphi & \triangleq \neg EF\neg\varphi & (\text{def. of } AG) \end{array}$$

Definitions of AF and EF indicate that we are using a "strong until", that is, $E[\varphi\ U\ \psi]$ and $A[\varphi\ U\ \psi]$ are TRUE only if ψ eventually occurs. Further, note that we are using EG, EX and EU as our adequate set (following [Huth and Ryan, 2000; Clarke et al., 1999]).

Alternatively, CTL operations can be described using their fixpoint formulations, as shown in Figure 2. This description is most useful for symbolic model-checking [McMillan, 1993].

Throughout the rest of the paper we will use a number of conventions: (1) our proofs follow the *calculational style* [Hehner, 1993; Back and von Wright, 1998]; (2) we refer to an unnamed function over the domain D as $\lambda x \in D \cdot$ *F-n Body*; (3) we use $\mu Z.f(Z)$ and $\nu Z.f(Z)$ to indicate the least and the greatest fixpoints of f, respectively [Back and von Wright, 1998]; (4) we use nat to refer to the set of natural numbers, and (5) we use $\exists!$ to mean "exists unique".

4. DEFINING THE LOGIC

This section introduces a family of quasi-boolean multi-valued logics. Section 4.1 provides the necessary lattice theory background, and can be skipped. The definition of quasi-boolean logics begins in Section 4.2.

Since our model-checker works for different multi-valued logics, we need a way to specify a particular logic we wish to use. We can specify a logic by giving its inference rules or by defining conjunction, disjunction and negation operations on the elements of the logic. Since our goal is model-checking as opposed to theorem proving, we chose the latter approach. Further, the logic should be as close to classical as possible; in particular, the defined operations should be idempotent, commutative, etc. Such properties can be easily guaranteed if we ensure that the values of the logic form a lattice. Indeed, lattices are a natural way to specify our logics, and our logics are just lattices with negation. In this section we give a brief introduction to lattice theory and describe the types of logics used by our model-checker.

4.1 Lattice Theory

DEFINITION 1. *A* lattice *is a partial order ($\mathcal{L}$, $\sqsubseteq$) for which a unique greatest lower bound and least upper bound exist for each finite subset of elements.*

Given lattice elements a and b, their greatest lower bound is referred to as *meet* and denoted by $a \sqcap b$, whereas their least upper bound is referred to as *join* and denoted by $a \sqcup b$. Meets and joins over an empty set are referred to as *top* and *bottom*, respectively:

$$\begin{array}{lll} \top \triangleq \sqcap\emptyset & & (\top \text{ definition}) \\ \bot \triangleq \sqcup\emptyset & & (\bot \text{ definition}) \end{array}$$

These represent the greatest and the least element of the lattice. Before giving their laws, we define *equality* over lattice elements:

$$a = b \triangleq a \sqsubseteq b \ \wedge \ b \sqsubseteq a \qquad \text{(equality)}$$

Then, the following laws hold for $\top$ and $\bot$:

$$\begin{array}{rcll} a \sqcup \top & = & \top & \quad (\text{base}) \\ a \sqcap \bot & = & \bot & \\ a \sqcap \top & = & a & \quad (\text{identity}) \\ a \sqcup \bot & = & a & \end{array}$$

Lattices have a number of additional properties, some of which are given below:

$$\begin{array}{rcll} a \sqcup a & = & a & \quad (\text{idempotence}) \\ a \sqcap a & = & a & \\ a \sqcup b & = & b \sqcup a & \quad (\text{commutativity}) \\ a \sqcap b & = & b \sqcap a & \\ a \sqcup (b \sqcup c) & = & (a \sqcup b) \sqcup c & \quad (\text{associativity}) \\ a \sqcap (b \sqcap c) & = & (a \sqcap b) \sqcap c & \\ a \sqcup (a \sqcap b) & = & a & \quad (\text{absorption}) \\ a \sqcap (a \sqcup b) & = & a & \\ a \sqsubseteq a' \ \wedge \ b \sqsubseteq b' & \Rightarrow & a \sqcap b \sqsubseteq a' \sqcap b' & \quad (\text{monotonicity}) \\ a \sqsubseteq a' \ \wedge \ b \sqsubseteq b' & \Rightarrow & a \sqcup b \sqsubseteq a' \sqcup b' & \end{array}$$

In addition, lattices have a number of rules that hold for them. We list several of them here, without proof:

$$\begin{array}{rcll} a \sqsubseteq b & \Leftrightarrow & \forall c,\ c \sqsubseteq a \Rightarrow c \sqsubseteq b & \quad (\sqsubseteq \text{ introduction}) \\ a \sqsubseteq b & \Leftrightarrow & \forall c,\ a \sqsubseteq c \Leftarrow b \sqsubseteq c & \quad (\sqsubseteq \text{ introduction}) \\ a \sqcap b \sqsubseteq b & \text{and} & a \sqcap b \sqsubseteq a & \quad (\sqcap \text{ elimination}) \\ a \sqsubseteq b \wedge a \sqsubseteq c & \Rightarrow & a \sqsubseteq b \sqcap c & \quad (\sqcap \text{ introduction}) \\ a \sqsubseteq a \sqcup b & \text{and} & b \sqsubseteq a \sqcup b & \quad (\sqcup \text{ introduction}) \\ a \sqsubseteq c \wedge b \sqsubseteq c & \Rightarrow & a \sqcup b \sqsubseteq c & \quad (\sqcup \text{ elimination}) \\ a \sqsubseteq b & \Leftrightarrow & a \sqcup b = b & \quad (\text{correspondence}) \\ a \sqsubseteq b & \Leftrightarrow & a \sqcap b = a & \quad (\text{correspondence}) \end{array}$$

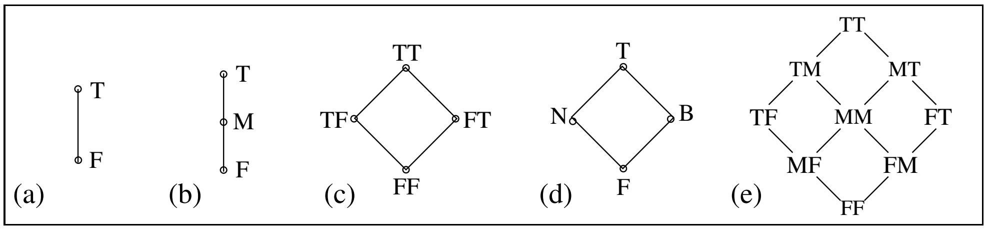

Fig. 3. Examples of quasi-boolean logics: (a) $\mathbf{2}$, the classical logic; (b) $\mathbf{3}$, a 3-valued logic representing uncertainty; (c) $\mathbf{2x2}$, a 4-valued logic representing disagreement; (d) Belnap's 4-valued logic representing inconsistent information; (e) a logic $\mathbf{3x3}$.

(a)

$\sqcap$	T	M	F
T	T	M	F
M	M	M	F
F	F	F	F

(b)

$\sqcup$	T	M	F
T	T	T	T
M	T	M	M
F	T	M	F

(c)

$\neg$	
T	F
M	M
F	T

Fig. 4. Tables of operations for $\mathbf{3}$: (a) meet table; (b) join table; (c) negation table.

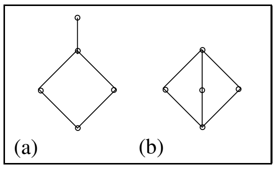

Fig. 5. Examples of lattices on which quasi-boolean logics cannot be defined: (a) a lattice without horizontal symmetry; (b) a non-distributive lattice.

DEFINITION 2. *A lattice is* distributive *iff*

$$\begin{aligned} a \sqcup (b \sqcap c) &= (a \sqcup b) \sqcap (a \sqcup c) \quad \text{(distributivity)} \\ a \sqcap (b \sqcup c) &= (a \sqcap b) \sqcup (a \sqcap c) \end{aligned}$$

In this paper we only use distributive lattices that have a finite number of elements and thus we can define logics on top of these lattices as follows:

DEFINITION 3. *A* logic *is a tuple* ($\mathcal{L}$, $\sqsubseteq$, $\neg$)*, where:*

—*($\mathcal{L}$, $\sqsubseteq$) is a finite distributive lattice, where $\top$ and $\bot$ correspond to values* TRUE *and* FALSE *of the logic, respectively;*

—*Conjunction and disjunction operators are $\sqcap$ and $\sqcup$ of ($\mathcal{L}$, $\sqsubseteq$);*

—*The partial order operation $a \sqsubseteq b$ means "b is more true than a";*

—*$\neg$ is a negation operator.*

Figure 3 gives examples of a few logics. Thermostat models in Figure 1(a-b),(d) used the logic $\mathbf{3}$, first defined in [Kleene, 1952], whereas the model in Figure 1(c) used the logic $\mathbf{3x3}$. Meet, join and negation tables for $\mathbf{3}$ are shown in Figure 4. Note that when the logic is classical, $\sqcap$ and $\sqcup$ operations are conventionally referred to as $\wedge$ and $\vee$, respectively. We will use these notations interchangeably when the interpretation is clear from the context. Further, in the logics in Figure 3, $\top$ is labeled 'T' (or 'TT') and $\bot$ is labeled 'F' (or 'FF') All underlying lattices in Figure 3 are distributive, whereas the lattice in Figure 5(b) is not.

DEFINITION 4. *A finite distributive lattice with negation is called a* finite Boolean logic *if every element* $a \in \mathcal{L}$ *has a unique* complement $\neg a \in \mathcal{L}$ *satisfying the following conditions:*

$$\begin{array}{rclll} \neg\neg a &=& a & (\neg \text{ involution}) & a \sqsubseteq b \Leftrightarrow \neg a \sqsupseteq \neg b \quad (\neg \text{ antimonotonic}) \\ \neg(a \sqcap b) &=& \neg a \sqcup \neg b & (\text{De Morgan}) & a \sqcap \neg a \;=\; \bot \quad (\neg \text{ contradiction}) \\ \neg(a \sqcup b) &=& \neg a \sqcap \neg b & (\text{De Morgan}) & a \sqcup \neg a \;=\; \top \quad (\neg \text{ exhaustiveness}) \end{array}$$

In fact, $\neg$ *involution, De Morgan and antimonotonic laws follow from* $\neg$ *contradiction and* $\neg$ *exhaustiveness.*

DEFINITION 5. *A* product *of two logics* $(\mathcal{L}_1, \sqsubseteq, \neg)$, $(\mathcal{L}_2, \sqsubseteq, \neg)$ *is a logic* $(\mathcal{L}_1 \times \mathcal{L}_2)$, *with the ordering* $\sqsubseteq$ *holding between two pairs iff it holds for each component separately, i.e.*

$$(a, b) \sqsubseteq (a', b') \;\Leftrightarrow\; a \sqsubseteq a' \;\wedge\; b \sqsubseteq b' \qquad (\sqsubseteq \text{ of pairs})$$

Bottom, top, complement, meet and join in the product logic are component-wise extensions of the corresponding operations of the component logics with the following properties:

$$\begin{array}{rcll} \bot_{\mathcal{L}_1\times\mathcal{L}_2} &=& (\bot_{\mathcal{L}_1}, \bot_{\mathcal{L}_2}) & (\bot \text{ of pairs}) \\ \top_{\mathcal{L}_1\times\mathcal{L}_2} &=& (\top_{\mathcal{L}_1}, \top_{\mathcal{L}_2}) & (\top \text{ of pairs}) \\ \neg(a, b) &=& (\neg a, \neg b) & (\neg \text{ of pairs}) \\ (a, b) \sqcap (a', b') &=& (a \sqcap a', b \sqcap b') & (\sqcap \text{ of pairs}) \\ (a, b) \sqcup (a', b') &=& (a \sqcup a', b \sqcup b') & (\sqcup \text{ of pairs}) \end{array}$$

Product of two logics preserves their distributivity, finiteness and boolean properties. For example, out of the five logics in Figure 3, only $\mathbf{2}$ and $\mathbf{2x2}$ are boolean. The former is boolean because ¬ T = F, ¬ F = T. The latter is a product of two $\mathbf{2}$ lattices and thus is finite, distributive and boolean. The logic $\mathbf{3}$ is not boolean because ¬M = M, and M $\sqcap$ ¬M $\neq \bot$, and thus neither is $\mathbf{3x3}$.

4.2 Quasi-Boolean Logics

In order to define suitable negation for our logics, we choose an operator mapping an element of the logic to another element of the logic (in lattice terms – an automorphism) that satisfies a number of laws that we typically associate with negation, such as involution and De Morgan laws.

DEFINITION 6. *A logic* ($\mathcal{L}$, $\sqsubseteq$, $\neg$) *is* quasi-boolean *[Bolc and Borowik, 1992] (also called* De Morgan *[Dunn, 1999]) iff* $\neg$ *is a lattice dual automorphism with period 2.*

THEOREM 1. *Let* ($\mathcal{L}$, $\sqsubseteq$, $\neg$) *be a quasi-boolean logic. Then:*

$$\begin{array}{rclll} \neg(a \sqcap b) &=& \neg a \sqcup \neg b & (\text{De Morgan}) & \neg\neg a \;=\; a \quad (\neg \text{ involution}) \\ \neg(a \sqcup b) &=& \neg a \sqcap \neg b & & a \sqsubseteq b \Leftrightarrow \neg a \sqsupseteq \neg b \quad (\neg \text{ antimonotonic}) \end{array}$$

where a, b *are elements of the logic. Thus,* $\neg a$ *is a* quasi-complement *of* a.

Proof:
De Morgan and antimonotonicity follow from the fact that ¬ is a lattice dual automorphism, and involution follows from the fact that ¬ has a period 2. □

Therefore, all boolean logics are quasi-boolean, whereas the converse is not true.

We are ready to state a few theorems about quasi-boolean logics.

THEOREM 2. *A product of two quasi-boolean logics is quasi-boolean, that is,*

$$\begin{array}{lrcl}
(1) & \neg\neg(a,b) & = & (a,b) \\
(2) & \neg((a_1,b_1)\sqcap(a_2,b_2)) & = & (\neg a_1,\neg b_1)\sqcup(\neg a_2,\neg b_2) \\
(3) & \neg((a_1,b_1)\sqcup(a_2,b_2)) & = & (\neg a_1,\neg b_1)\sqcap(\neg a_2,\neg b_2) \\
(4) & (a_1,b_1)\sqsubseteq(s_2,b_2) & \Leftrightarrow & \neg(a_1,b_1)\sqsupseteq\neg(a_2,b_2)
\end{array}$$

Proof:

(1) $\neg\neg(a,b)$	(2) $\neg((a_1,b_1)\sqcap(a_2,b_2))$	(4) $(a_1,b_1)\sqsubseteq(a_2,b_2)$
$\Leftrightarrow$ ($\neg$ of pairs)	$\Leftrightarrow$ ($\sqcap$ of pairs)	$\Leftrightarrow$ ($\sqsubseteq$ of pairs)
$\neg(\neg a,\neg b)$	$\neg((a_1\sqcap a_2),(b_1\sqcap b_2))$	$a_1\sqsubseteq a_2\ \wedge\ b_1\sqsubseteq b_2$
$\Leftrightarrow$ ($\neg$ of pairs)	$\Leftrightarrow$ ($\neg$ of pairs)	$\Leftrightarrow$ ($\neg$ antimonotonic)
$(\neg\neg a,\neg\neg b)$	$(\neg(a_1\sqcap a_2),\neg(b_1\sqcap b_2))$	$\neg a_1\sqsupseteq\neg a_2\ \wedge\ \neg b_1\sqsupseteq\neg b_2$
$\Leftrightarrow$ ($\neg$ involution)	$\Leftrightarrow$ (De Morgan)	$\Leftrightarrow$ ($\sqsupseteq$ of pairs)
(a,b)	$(\neg a_1\sqcup\neg a_2,\neg b_1\sqcup\neg b_2)$	$(\neg a_1,\neg b_1)\sqsupseteq(\neg a_2,\neg b_2)$
	$\Leftrightarrow$ ($\sqcup$ of pairs)	$\Leftrightarrow$ ($\neg$ of pairs)
	$(\neg a_1,\neg b_1)\sqcup(\neg a_2,\neg b_2)$	$\neg(a_1,b_1)\sqsupseteq\neg(a_2,b_2)$

The proof of (3) is similar to that of (2). □

For example, the logic $\mathbf{3}$ and all its products (e.g. $\mathbf{3x3}$ in Figure 3(e)) are quasi-boolean.

The identification of a suitable negation operator is greatly simplified by the observation that quasi-boolean logics have underlying lattices that are symmetric about their horizontal axes:

DEFINITION 7. *A lattice ($\mathcal{L}$, $\sqsubseteq$) is* horizontally-symmetric *iff there exists a bijective function H such that for every pair $a,b\in\mathcal{L}$,*

$$\begin{array}{rl}
a\sqsubseteq b\ \Leftrightarrow\ H(a)\sqsupseteq H(b) & \text{(order-embedding)} \\
H(H(a))=a & (H \text{ involution})
\end{array}$$

Notice that H is a lattice dual automorphism with period 2. Thus, horizontal symmetry is a necessary and sufficient condition for an ability to define a quasi-boolean logic over a distributive lattice, with a potential negation defined as $\neg a = H(a)$ for each element of the lattice. All lattices in Figure 3 exhibit horizontal symmetry and thus are quasi-boolean, whereas the lattice in Figure 5(a) is not. In the rest of the paper we assume that the negation operator on our logics is defined through horizontal symmetry, unless explicitly specified otherwise.

There are a number of other ways to define negation in horizontally-symmetric lattices. For example, in $\mathbf{2x2}$ shown in Figure 3(c), ¬TF = FT, ¬FT = TF. However, in Belnap's 4-valued logic in Figure 3(d), proposed for reasoning about inconsistent databases [Belnap, 1977; Anderson and Belnap, 1975], ¬N = N, ¬B = B. Thus, the two lattices are isomorphic, but the logics defined on them are not.

Finally, we define additional operators the usual way:

$$\begin{array}{rcll}
a\rightarrow b & \triangleq & \neg a\sqcup b & \text{(material implication)} \\
a\leftrightarrow b & \triangleq & (a\rightarrow b)\ \sqcap\ (b\rightarrow a) & \text{(equivalence)}
\end{array}$$

5. MULTI-VALUED SETS AND RELATIONS

In order to define multi-valued model-checking later in this paper, we begin by creating a data structure that allows definition and reasoning about operations on sets of states in

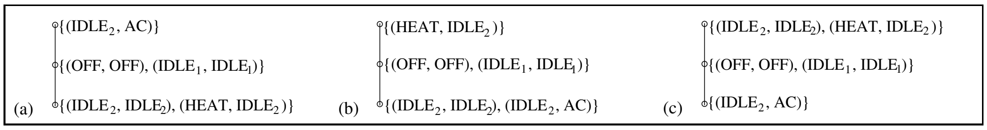

Fig. 6. (a) The 3-valued set of states reflecting values of `Above` in Thermo$_2$; (b) The 3-valued set of states reflecting values of `Below` in Thermo$_2$; (c) $\overline{[\![\mathtt{Above}]\!]}$ – a multi-valued complement of the mv-set in (a).

which a property holds. Such operations include union, intersection, complement, and backward image for computing predecessors. Given a multi-valued logic, we can treat these as operations over multi-valued sets: sets whose membership functions are multi-valued. We define the concept of multi-valued sets and relations over quasi-boolean logics in this section. This treatment is similar to the definition of L-fuzzy sets [Goguen, 1967].

5.1 Multi-Valued Sets

In classical set theory, a set is defined by a boolean predicate, also called a *membership function*. Typically, it is written using *set comprehension notation*: a predicate P defines the set $S = \{x \mid P(x)\}$. For instance, if $P = \lambda x \in nat \cdot 0 \leq x \leq 10$, then S is the set of all integers between 0 and 10 inclusive. If instead of using a boolean predicate, we allow the membership function to range over a multi-valued logic, we obtain a *multi-valued* set theory in which it is possible to make statements like "element x is more in set $\mathbb{S}$ than element y". We call the result *mv-sets* and denote them using a special font, e.g., $\mathbb{S}$.

Given a logic $L = (\mathcal{L}, \sqsubseteq, \neg)$, we define a multi-valued set theory relative to it. For an mv-set $\mathbb{S}$ and a candidate element x, we use $\mathbb{S}(x)$ to denote the membership degree of x in $\mathbb{S}$. In the classical case, this amounts to representing a set by its characteristic function.

For example, we can use the 3-valued logic representing uncertainty (**3** in Figure 3(b)) to encode information about values of variable `Above` in different states of Thermo$_2$ (Figure 1(d)). We refer to this mv-set as $[\![\mathtt{Above}]\!]$. Each value ℓ of **3** is associated with the set of states where `Above` has value ℓ. For example, `Above` is T in $\{(\mathtt{IDLE}_2, \mathtt{AC})\}$, M in $\{(\mathtt{OFF}, \mathtt{OFF}), (\mathtt{IDLE}_1, \mathtt{IDLE}_1)\}$, and F in $\{(\mathtt{IDLE}_2, \mathtt{IDLE}_2), (\mathtt{HEAT}, \mathtt{IDLE}_2)\}$. This mv-set can be graphically represented as shown in Figure 6(a), where the structure corresponds to that of the underlying logic.

We extend some standard set operations to the multi-valued case by lifting the lattice meet and join operations, as follows:

$$
\begin{array}{rcll}
(\mathbb{S} \cap_L \mathbb{S}')(x) & \triangleq & (\mathbb{S}(x) \sqcap \mathbb{S}'(x)) & \text{(Multi-valued intersection)} \\
(\mathbb{S} \cup_L \mathbb{S}')(x) & \triangleq & (\mathbb{S}(x) \sqcup \mathbb{S}'(x)) & \text{(Multi-valued union)} \\
\mathbb{S} \subseteq_L \mathbb{S}' & \triangleq & \forall x \cdot (\mathbb{S}(x) \sqsubseteq \mathbb{S}'(x)) & \text{(Set inclusion)} \\
\mathbb{S} = \mathbb{S}' & \triangleq & \forall x \cdot (\mathbb{S}(x) = \mathbb{S}'(x)) & \text{(Extensional equality)}
\end{array}
$$

For example, in computing intersection of mv-sets $[\![\mathtt{Above}]\!]$ and $[\![\mathtt{Below}]\!]$ given in Figure 6(a) and (b), respectively, we note that in state $(\mathtt{IDLE}_2, \mathtt{AC})$ `Above` is T and `Below` is F. Thus,

$$([\![\mathtt{Above}]\!] \cap_L [\![\mathtt{Below}]\!])\big((\mathtt{IDLE}_2, \mathtt{AC})\big) = \mathrm{T} \sqcap \mathrm{F} = \mathrm{F}$$

We can also extend the notion of *set complement* to the multi-valued case, by defining it in terms of the quasi-complement of L, and denoting it with a bar:

$$\overline{\mathbb{S}}(x) \triangleq \neg(\mathbb{S}(x)) \quad \text{(Multi-valued complement)}$$

Mv-set $\overline{[\![\texttt{Above}]\!]}$ is given in Figure 6(c).

We then obtain the expected properties:

$$\begin{array}{rcll} \overline{\mathbb{S} \cup_L \mathbb{S}'} & = & \overline{\mathbb{S}} \cap_L \overline{\mathbb{S}'} & \text{(De Morgan 1)} \\ \overline{\mathbb{S} \cap_L \mathbb{S}'} & = & \overline{\mathbb{S}} \cup_L \overline{\mathbb{S}'} & \text{(De Morgan 2)} \\ \mathbb{S} \subseteq_L \mathbb{S}' & = & \overline{\mathbb{S}'} \subseteq_L \overline{\mathbb{S}} & \text{(antimonotonicity)} \end{array}$$

THEOREM 3. *For a multi-valued set $\mathbb{S}$ defined over the 2-valued logic ($\{T, F\}$, $\sqsubseteq$, $\neg$), the following holds:*

(1) *The membership function $\mathbb{S}(x)$ is a boolean predicate*
(2) $(\mathbb{S} \cap_L \mathbb{S}') = (\mathbb{S} \cap \mathbb{S}')$
(3) $(\mathbb{S} \cup_L \mathbb{S}') = (\mathbb{S} \cup \mathbb{S}')$
(4) $\overline{\mathbb{S}} = \mathbb{S}^c$

where $\mathbb{S}^c$ is the classical set complement.

Proof:
The proof follows from the the fact that the set membership function $\mathbb{S}(x)$ is boolean. Then, each mv-set is boolean and thus union, intersection and complement reduce to classical. □

Note that if the underlying logic, L, is the interval $[0, 1]$, ordered by the usual $\leq$ relation on real numbers, then we obtain fuzzy set theory [Zadeh, 1987].

5.2 Multi-Valued Relations

Now we extend the concept of degrees of membership in an mv-set to degrees of relatedness of two entities. This concept, formalized by *multi-valued relations*, allows us to define multi-valued transitions in state machine models.

DEFINITION 8. *A multi-valued relation $\mathbb{R}$ on two sets S and T is an mv-set over $S \times T$.*

For example, the multi-valued relation representing values of transitions between pairs of states in Thermo_2 is given in Figure 7(a), where we encode states as follows:

$$\begin{array}{rclrcl} (\texttt{IDLE}_2, \texttt{AC}) & = & 1 & (\texttt{IDLE}_2, \texttt{IDLE}_2) & = & 4 \\ (\texttt{OFF}, \texttt{OFF}) & = & 2 & (\texttt{HEAT}, \texttt{IDLE}_2) & = & 5 \\ (\texttt{IDLE}_1, \texttt{IDLE}_1) & = & 3 & & & \end{array}$$

This relation, referred to as $\mathbb{T}$, is over $S \times S$, where S is the set of all states of Thermo_2.

DEFINITION 9. *The* forward image $\overrightarrow{\mathbb{R}}(\mathbb{Q})$ *of an mv-set $\mathbb{Q}$ over S under relation $\mathbb{R}$ is*

$$\overrightarrow{\mathbb{R}}(\mathbb{Q}) \triangleq \lambda t \cdot \bigsqcup_{s \in S} (\mathbb{Q}(s) \sqcap \mathbb{R}(s, t))$$

The backward image $\overleftarrow{\mathbb{R}}(\mathbb{Q})$ *of an mv-set $\mathbb{Q}$ over T under relation $\mathbb{R}$ is*

$$\overleftarrow{\mathbb{R}}(\mathbb{Q}) \triangleq \lambda s \cdot \bigsqcup_{t \in T} (\mathbb{Q}(t) \sqcap \mathbb{R}(s, t))$$

Intuitively, a forward image of an mv-set $\mathbb{Q}$ over the relation $\mathbb{R}$ represents all elements reachable from $\mathbb{Q}$ by $\mathbb{R}$, where multi-valued memberships of $\mathbb{R}$ and $\mathbb{Q}$ are taken into consideration. Similarly, a backward image of an mv-set $\mathbb{Q}$ over $\mathbb{R}$ represents all elements that can reach $\mathbb{Q}$ by $\mathbb{R}$. Given an mv-set over S, its forward image under the relation $\mathbb{R}$ is an mv-set over T; likewise, an mv-set over T has an mv-set over S as a backward image.

(a)	(b)	(c)
{(1,1), (1,4), (2,2), (2,3), (3,2), (3,3), (3,4), (3,5), (4,1), (4,4), (4,5), (5,5)}	{(IDLE$_2$, AC), (IDLE$_2$, IDLE$_2$)}	{(IDLE$_2$, AC), (IDLE$_2$, IDLE$_2$ (IDLE$_1$, IDLE$_1$)}
{(1,2), (4,2), (4,3), (5,2), (5,3)}	{(OFF, OFF), (IDLE$_1$, IDLE$_1$), (HEAT, IDLE$_2$)}	{(OFF, OFF), (HEAT, IDLE$_2$)}
{(1,3), (1,5), (2,1), (2,4), (2,5), (5,1)}		

Fig. 7. (a) The multi-valued relation between pairs of states of Thermo$_2$; (b) Forward image of $[\![\texttt{Above}]\!]$ over the relation in (a); (c) Backward image of $[\![\texttt{Above}]\!]$ over the relation in (a).

The forward and the backward images of $[\![\texttt{Above}]\!]$ (see Figure 6(a)) under the multi-valued relation $\mathbb{T}$ between states of Thermo$_2$ (Figure 7(a)) are shown in Figure 7(b) and (c), respectively. For example, when we compute backward image of $(\texttt{IDLE}_1, \texttt{IDLE}_1)$, we get

$$\bigsqcup_{t \in S} [\![\texttt{Above}]\!](t) \sqcap \mathbb{T}((\texttt{IDLE}_1, \texttt{IDLE}_1), t) = (\mathrm{T} \sqcap \mathrm{T}) \sqcup (\mathrm{M} \sqcap \mathrm{T}) \sqcup (\mathrm{M} \sqcap \mathrm{T}) \sqcup (\mathrm{F} \sqcap \mathrm{T}) \sqcup (\mathrm{F} \sqcap \mathrm{T}) = \mathrm{T}$$

which indicates that there exists a TRUE transition from $(\texttt{IDLE}_1, \texttt{IDLE}_1)$ to a state where `Above` is TRUE.

If $\mathbb{R}$ is a boolean relation and $\mathbb{Q}$ is a classical set, then forward and backward images, referred to as $\overrightarrow{\mathbb{R}}^B$ and $\overleftarrow{\mathbb{R}}^B$, respectively, become:

$$\begin{aligned} \overrightarrow{\mathbb{R}}^B(\mathbb{Q}) &\triangleq \lambda t \cdot \bigvee\nolimits_{\{s \in S \mid \mathbb{R}(s,t)\}} \mathbb{Q}(s) \\ \overleftarrow{\mathbb{R}}^B(\mathbb{Q}) &\triangleq \lambda s \cdot \bigvee\nolimits_{\{t \in S \mid \mathbb{R}(s,t)\}} \mathbb{Q}(t) \end{aligned}$$

This gives rise to the following theorem:

THEOREM 4. *Let L be $(\{T, F\}, \sqsubseteq, \neg)$. Then $\mathbb{R}$ becomes a boolean relation, and $\mathbb{Q}$ becomes a set (an mv-set where for each element s $\mathbb{Q}(s)$ is either $\top$ or $\bot$). In this case,*

$$\begin{aligned} (1)\ \overrightarrow{\mathbb{R}}(\mathbb{Q}) &= \overrightarrow{\mathbb{R}}^B(\mathbb{Q}) \\ (2)\ \overleftarrow{\mathbb{R}}(\mathbb{Q}) &= \overleftarrow{\mathbb{R}}^B(\mathbb{Q}) \end{aligned}$$

Proof:

$$\begin{aligned} & \overrightarrow{\mathbb{R}}(\mathbb{Q}) \\ = & \ \text{(Definition of } \overrightarrow{\mathbb{R}}) \\ & \lambda t \cdot \textstyle\bigsqcup_{s \in S} (\mathbb{Q}(s) \sqcap \mathbb{R}(s,t)) \\ = & \ (\mathbb{R} \text{ is a boolean relation}) \\ & \lambda t \cdot \textstyle\bigsqcup_{\{s \in S \mid \mathbb{R}(s,t)\}} \mathbb{Q}(s) \\ = & \ \text{(Theorem 3)} \\ & \lambda t \cdot \textstyle\bigvee_{\{s \in S \mid \mathbb{R}(s,t)\}} \mathbb{Q}(s) \\ = & \ \text{(Definition of } \overrightarrow{\mathbb{R}}^B) \\ & \overrightarrow{\mathbb{R}}^B(\mathbb{Q}) \end{aligned}$$

The proof of (2) is similar. ◻

6. MULTI-VALUED CTL MODEL-CHECKING

In this section we extend the notion of boolean model-checking described in Section 3 by defining multi-valued Kripke structures, which we call χKripke structures, and multi-valued CTL (χCTL).

6.1 Semantics

M is a *χKripke structure* if $M = (S, s_0, \mathbb{R}, I, A, L)$, where:

—$L = (\mathcal{L}, \sqsubseteq, \neg)$ is a quasi-boolean logic. This is the logic for all mv-sets in the model.

—A is a (finite) set of atomic propositions, otherwise referred to as variables. We assume that all variables are of the same type, with values ranging over the values of the logic L.

—S is a (finite) set of states; each state is identified by a unique (within M) label s.

—$s_0 \in S$ is the initial state.

—$\mathbb{R} : S \times S \to \mathcal{L}$ is the multi-valued transition relation.

—$I : S \to \mathcal{L}^A$ is a (total) labeling function that maps states in S to mv-sets over A.

Intuitively, for any atomic proposition $a \in A$, $(I(s))(a) = \ell$ means that the variable a has value ℓ in state s. Given an atomic proposition $a \in A$, $I'_a : S \to \mathcal{L}$ is a (total) multi-valued characteristic function for an mv-set of S. I'_a is defined as follows:

$$I'_a \triangleq \lambda s \cdot (I(s))(a)$$

Thus, for each proposition a, I'_a *partitions* the state-space with respect to it, i.e. for each state s, $\exists! \ell \cdot I'_a(s) = \ell$.

Note that a χKripke structure is a completely connected graph. As with classical model-checking, we ensure that all traces have infinite length by requiring that there is at least one non-$\perp$ transition out of each state (if necessary, by adding a non-$\perp$ self-loop to terminal states). Formally,

$$\forall s \in S \cdot \exists t \in S \cdot \mathbb{R}(s,t) \neq \perp$$

To avoid clutter, when we present finite-state machines graphically, we follow the convention of not showing $\perp$ transitions. Our models of the thermostat given in Figure 1 are examples of χKripke structures.

6.2 Multi-Valued CTL

Here we give semantics of CTL operators on a χKripke structure M over a quasi-boolean logic L. We refer to this language as *multi-valued CTL*, or χCTL.

In extending the CTL operators, we want to ensure that the expected properties of EX, EG and EU, which form the adequate set for CTL, are still preserved. Other operators are defined as described in Section 3. χCTL operators are defined on *traces* which are sets of computations.

DEFINITION 10. A computation *of a χKripke structure M from a (reachable) state s is an infinite sequence of states $s_0, s_1, \ldots$ s.t. $s = s_0$, and $\mathbb{R}(s_i, s_{i+1}) \neq \perp$. This sequence of states is also referred to as a* path. *A* trace *from s is a (possibly empty) set of computations emanating from s. Traces can be viewed as* trees *of computations.*

We also note that evaluating a formula φ in a state s is the same as evaluating φ on a tree of all computations emanating from s.

We start defining χCTL by giving the semantics of propositional operators. We use the double-brace notation, adopted from denotational semantics, and write $[\![\varphi]\!]$ to denote the mv-set of states representing a degree to which φ holds. Note that we have already used this notation when illustrating mv-sets in Section 5.

The semantics is as follows:

$$\begin{array}{rcl}
[\![a]\!] & \triangleq & I'_a \\
[\![\neg\varphi]\!] & \triangleq & \overline{[\![\varphi]\!]} \\
[\![\varphi \wedge \psi]\!] & \triangleq & [\![\varphi]\!] \cap_L [\![\psi]\!] \\
[\![\varphi \vee \psi]\!] & \triangleq & [\![\varphi]\!] \cup_L [\![\psi]\!]
\end{array}$$

We proceed by defining the EX operator. Recall from Section 3 that in classical CTL, this operator is defined using existential quantification over next states. We extend the notion of existential quantification for multi-valued reasoning through the use of disjunction. This treatment of quantification is standard [Belnap, 1977; Rasiowa, 1978]. The semantics of EX is:

$$[\![EX\varphi]\!] \triangleq \overleftarrow{\mathbb{R}}\,([\![\varphi]\!]) \quad \text{(def. of } EX\text{)}$$

Note that we use our definition of backward image (Definition 8), i.e. for a state s,

$$[\![EX\varphi]\!](s) = \bigsqcup_{t \in S} ([\![\varphi]\!](s) \sqcap \mathbb{R}(s,t))$$

If we are reasoning about a model which was produced by merging two (classical) models, we can think of $EX\varphi$ as representing a question "does there exist a next state in each individual model where φ is TRUE, even if the two individual models do not agree on what this state is". For example, consider evaluating a property $EXEX\neg\texttt{Running}$ in different models of the thermostat (see Figure 1). This property is T in state HEAT of the Heater Aspect, satisfied by the path HEAT, OFF, OFF. This property is also T in state $\texttt{IDLE}_2$ of the AC Aspect, satisfied by the path $\texttt{IDLE}_2$, $\texttt{IDLE}_1$, OFF. In Thermo_1 the property $EXEX\neg\texttt{Running}$ evaluates to TF on path $(\texttt{HEAT}, \texttt{IDLE}_2)$, $(\texttt{OFF}, \texttt{OFF})$, $(\texttt{OFF}, \texttt{OFF})$ and to FT on path $(\texttt{HEAT}, \texttt{IDLE}_2)$, $(\texttt{IDLE}_1, \texttt{IDLE}_1)$, $(\texttt{OFF}, \texttt{OFF})$. Their disjunction, and therefore the value of $EXEX\neg\texttt{Running}$ at $(\texttt{HEAT}, \texttt{IDLE}_2)$, is TT.

AX is then defined, following the AX property in Section 3, as

$$[\![AX\varphi]\!] \triangleq \overline{[\![EX\neg\varphi]\!]} \quad \text{(def. of } AX\text{)}$$

Expanding this definition, $[\![AX\varphi]\!] = \overline{\overleftarrow{\mathbb{R}}\,(\overline{[\![\varphi]\!]})} = \lambda s \cdot \bigsqcap_{t \in S}([\![\varphi]\!](t) \sqcup \neg\mathbb{R}(s,t))$, we see that universal quantification in the AX operator is replaced by conjunction.

Note that our definitions of EX and AX enjoy some familiar properties of their CTL counter-parts. In particular,

$$\begin{array}{rcll}
[\![EX(\varphi \vee \psi)]\!] & = & [\![EX\varphi]\!] \cup_L [\![EX\psi]\!] & (EX \text{ of disjunction}) \\
[\![AX(\varphi \wedge \psi)]\!] & = & [\![AX\varphi]\!] \cap_L [\![AX\psi]\!] & (AX \text{ of conjunction})
\end{array}$$

We further define EG and EU using the EG and EU fixpoint properties in Figure 2:

$$\begin{array}{rcll}
[\![EG\varphi]\!] & \triangleq & \nu\mathbb{Z}.[\![\varphi]\!] \cap_L [\![EX\mathbb{Z}]\!] & \text{(def. of } EG\text{)} \\
[\![E[\varphi\ U\ \psi]]\!] & \triangleq & \mu\mathbb{Z}.[\![\psi]\!] \cup_L ([\![\varphi]\!] \cap_L [\![EX\mathbb{Z}]\!]) & \text{(def. of } EU\text{)}
\end{array}$$

Then $A[\varphi\ U\ \psi]$ becomes

$$[\![A[\varphi\ U\ \psi]]\!] \triangleq \overline{[\![E[\neg\psi\ U\ \neg\varphi \wedge \neg\psi]]\!]} \cap_L \overline{[\![EG\neg\psi]\!]} \quad \text{(def. of } AU)$$

and the remaining χCTL operators are defined as their classical counter-parts:

$$\begin{array}{lcll} [\![AF\varphi]\!] & \triangleq & [\![A[\top\ U\ \varphi]]\!] & \text{(def. of } AF) \\ [\![EF\varphi]\!] & \triangleq & [\![E[\top\ U\ \varphi]]\!] & \text{(def. of } EF) \\ [\![AG\varphi]\!] & \triangleq & \overline{[\![EF\neg\varphi]\!]} & \text{(def. of } AG) \end{array}$$

Note that our definition of EU also preserves the familiar property of its CTL counter-part:

$$[\![E[\varphi\ U\ \psi]]\!] = [\![E\,[\varphi\ U\ E[\varphi\ U\ \psi]]]\!] \quad (EU \text{ expansion})$$

Property	χCTL formulation	Heater Model	AC Model	Thermo$_1$	Thermo$_2$
Prop. 1.	$AG\ EX\,\texttt{IDLE}_1$	F	T	FT	M
Prop. 2	$E\ [\neg\texttt{Below}\ U\ (\neg\texttt{Below} \wedge \texttt{Heat})]$	F	–	FF	F
Prop. 3.	$AG\ EF\neg\texttt{Running}$	T	T	TT	M
Prop. 4.	$AG\ (\texttt{Heat} \rightarrow \neg\texttt{Air})$	–	–	TT	T
Prop. 5.	$AG\ (\texttt{Above} \rightarrow \neg\texttt{Heat})$	–	–	TT	T

Table I. Results of verifying properties of the thermostat system.

The properties of the thermostat system that we identified in Section 2 can be translated into χCTL as described in Table I. A state name in a property, such as $\texttt{IDLE}_1$ in Prop. 1, is a short-cut for specifying a conjunction of values of variables in that state. For example, in the Heater Aspect,

$$\texttt{IDLE}_1 \triangleq \texttt{Running} = \mathrm{T} \wedge \texttt{Below} = \mathrm{M} \wedge \texttt{Heat} = \mathrm{F}$$

The table also lists the values of these properties in each of the models given in Figure 1. We use "–" to indicate that the result cannot be obtained from this model. The two models disagreed on the value of Prop. 1, so in Thermo$_1$ this property evaluated to FT, whereas in Thermo$_2$ we lost information about the exact cause of disagreement. Prop. 2 cannot be evaluated in the AC Aspect, but is F in the Heater Aspect and in both combined models. The two models agreed on Prop. 3 but disagreed on the *reasons* why this property is T: in the Heater Aspect, the path satisfying this property from state $\texttt{IDLE}_2$ went directly to state $\texttt{OFF}$, whereas in the AC Aspect, the path went through state $\texttt{IDLE}_1$ first. Thermo$_1$ preserved the agreement. In Thermo$_2$, where transitions from $(\texttt{IDLE}_2, \texttt{IDLE}_2)$ to $(\texttt{IDLE}_1, \texttt{IDLE}_1)$ and $(\texttt{OFF}, \texttt{OFF})$ are abstracted to M, the property was no longer T.

6.3 Correctness and Termination

We begin by proving a few correctness theorems about χCTL.

THEOREM 5. *χCTL reduces to CTL when the logic is $\mathbf{2}$. That is,*

$$\begin{array}{lrcl} (1) & [\![EX\varphi]\!] & = & [\![EX^B\varphi]\!] \\ (2) & [\![EG\varphi]\!] & = & [\![EG^B\varphi]\!] \\ (3) & [\![E[\varphi\ U\ \psi]]\!] & = & [\![E[\varphi\ U^B\ \psi]]\!] \end{array}$$

where EG^B, EU^B, and EX^B are classical CTL operators defined in Section 3.

Proof:
(1) holds by Theorem 4. The proof for (2) is based on the fact that $[\![EG\varphi]\!]$ can be represented by $\nu\mathbb{Z}.F(\mathbb{Z})$, where $F(\mathbb{Z}) = [\![\varphi]\!] \cap_L [\![EX\mathbb{Z}]\!]$. Since $F(\mathbb{Z})$ reduces to its boolean version syntactically, $[\![EG\varphi]\!]$ reduces to $[\![EG^B\varphi]\!]$. The proof for (3) is similar to that of (2).

□

THEOREM 6. *χCTL operators AX and EX are monotone.*

Proof:
We begin by reciting some properties of monotone functions and fixpoints. Assume that F and G are monotone functions. Then, $F(\mathbb{Z}) \cup G(\mathbb{Z})$, $F(\mathbb{Z}) \cap G(\mathbb{Z})$, and $\overline{F(\overline{\mathbb{Z}})}$ are monotone.

We want to show that EX is monotone:

$$
\begin{array}{ll}
 & \mathbb{Z} \subseteq_L \mathbb{Y} \\
\Rightarrow & \text{(def. of } \subseteq_L\text{)} \\
 & \forall t \in S \cdot \mathbb{Z}(t) \sqsubseteq \mathbb{Y}(t) \\
\Rightarrow & (\sqcap \text{ is monotone}) \\
 & \forall s \in S\ \forall t \in S \cdot \mathbb{R}(s,t) \sqcap \mathbb{Z}(t) \sqsubseteq \mathbb{R}(s,t) \sqcap \mathbb{Y}(t) \\
\Rightarrow & (\bigsqcup \text{ is monotone}) \\
 & \forall s \in S \cdot \bigsqcup_{t \in S} \mathbb{R}(s,t) \sqcap \mathbb{Z}(t) \sqsubseteq \mathbb{R}(s,t) \sqcap \mathbb{Y}(t) \\
\Rightarrow & \text{(def. of } EX\text{)} \\
 & \forall s \in S \cdot [\![EX\mathbb{Z}]\!](s) \sqsubseteq [\![EX\mathbb{Y}]\!](s) \\
\Rightarrow & \text{(def. of } \subseteq_L\text{)} \\
 & [\![EX\mathbb{Z}]\!] \subseteq_L [\![EX\mathbb{Y}]\!]
\end{array}
$$

AX is monotone because $[\![AX]\!] = \overline{[\![EX\neg\mathbb{Z}]\!]}$ and EX is monotone. □

We proceed by proving that χCTL is well defined, i.e., that each property φ will have exactly one value in each state of the system.

THEOREM 7. *The definition of χCTL ensures that for each φ, $[\![\varphi]\!]$ forms a partition.*

Proof:
To prove this theorem, it suffices to prove that the computation of $EX\varphi$ forms a partition. All other operators are computed using fixpoints and applications of $\cup_L$, $\cap_L$, and $\neg$ operators which, given mv-sets where the characteristic function is total, produce mv-sets with a total characteristic function, thereby forming a partition. We now turn to showing that the computation of $EX\varphi$ forms a partition.

$$
\begin{array}{ll}
 & \forall s \in S\ \exists! \ell \cdot [\![EX\varphi]\!](s) = \ell \\
= & \text{(def. of } EX\text{)} \\
 & \forall s \in S\ \exists! \ell \cdot \bigsqcup_{t \in S}([\![\varphi]\!](t) \sqcap \mathbb{R}(s,t)) = \ell \\
\Leftarrow & ([\![\varphi]\!] \text{ is a partition}), (\mathbb{R}(s,t) \text{ is an mv-relation s.t.} \forall (s,t)\ \exists! \ell' \in \mathcal{L} \cdot \mathbb{R}(s,t) = \ell'), \text{(Definition 1)} \\
 & \forall s \in S\ \exists! \ell\ \exists! \ell_t \cdot \bigsqcup_{t \in S} \ell_t = \ell \\
\Leftarrow & (\bigsqcup \text{ preserves partition property}) \\
 & \top
\end{array}
$$

□

THEOREM 8. *Let M be an arbitrary χKripke structure and φ be an arbitrary χCTL formula. Computation of φ on M is guaranteed to terminate.*

Proof:
By Theorem 6 and monotonicity of $\cap_L$ and $\cup_L$ we conclude that all χCTL operators involve computation of fixpoints over monotone functions. Since monotone functions over finite lattices are continuous, by the Knaster-Tarski theorem, these fixpoints always exist. □

THEOREM 9. *Fixpoint properties of (derived) χCTL operators are the same as for CTL operators. That is,*

$$\begin{array}{rcll}
\llbracket AG\varphi \rrbracket & = & \nu\mathbb{Z}.\llbracket \varphi \rrbracket \cap_L \llbracket AX\mathbb{Z} \rrbracket & (AG \text{ fixpoint}) \\
\llbracket AF\varphi \rrbracket & = & \mu\mathbb{Z}.\llbracket \varphi \rrbracket \cup_L \llbracket AX\mathbb{Z} \rrbracket & (AF \text{ fixpoint}) \\
\llbracket EF\varphi \rrbracket & = & \mu\mathbb{Z}.\llbracket \varphi \rrbracket \cup_L \llbracket EX\mathbb{Z} \rrbracket & (EF \text{ fixpoint}) \\
\llbracket A[\varphi \; U \; \psi] \rrbracket & = & \mu\mathbb{Z}.\llbracket \psi \rrbracket \cup_L (\llbracket \varphi \rrbracket \cap_L \llbracket AX\mathbb{Z} \rrbracket) & (AU \text{ fixpoint})
\end{array}$$

Proof:
We structure the proof by showing that definitions for the temporal operators AG, AF, EF and AU are the same as their fixpoints. First we recall the property relating least and greatest fixpoints [Kozen, 1983]

$$\mu\mathbb{Z}.F(\mathbb{Z}) \; = \; \overline{\nu\mathbb{Z}.\overline{F(\overline{\mathbb{Z}})}} \qquad \text{(negation fixpoint)}$$

$$\begin{array}{rl}
 & \llbracket EF\varphi \rrbracket \\
= & \text{(def. of } EF\text{)} \\
 & \llbracket E[\top \; U \; \varphi] \rrbracket \\
= & (EU \text{ fixpoint}) \\
 & \mu\mathbb{Z}.\llbracket \varphi \rrbracket \cup_L (\llbracket \top \rrbracket \cap_L \llbracket EX\mathbb{Z} \rrbracket) \\
= & \text{(identity of } \top\text{)} \\
 & \mu\mathbb{Z}.\llbracket \varphi \rrbracket \cup_L \llbracket EX\mathbb{Z} \rrbracket
\end{array}$$

$$\begin{array}{rl}
 & \llbracket AF\varphi \rrbracket \\
= & \text{(def. of } AF\text{)} \\
 & \llbracket A[\top \; U \; \varphi] \rrbracket \\
= & (AU \text{ fixpoint}) \\
 & \mu\mathbb{Z}.\llbracket \varphi \rrbracket \cup_L (\llbracket \top \rrbracket \cap_L \llbracket AX\mathbb{Z} \rrbracket) \\
= & \text{(identity of } \top\text{)} \\
 & \mu\mathbb{Z}.\llbracket \varphi \rrbracket \cup_L \llbracket AX\mathbb{Z} \rrbracket
\end{array}$$

$$\begin{array}{rl}
 & \llbracket AG\varphi \rrbracket \\
= & \text{(def. of } AG\text{)} \\
 & \overline{\llbracket EF\neg\varphi \rrbracket} \\
= & (EF \text{ fixpoint}) \\
 & \overline{\mu\mathbb{Z}.\llbracket \neg\varphi \rrbracket \cup_L \llbracket EX\mathbb{Z} \rrbracket} \\
= & \text{(negation fixpoint)} \\
 & \nu\mathbb{Z}.\overline{\llbracket \neg\varphi \rrbracket \cup_L \llbracket EX\neg\mathbb{Z} \rrbracket} \\
= & \text{(De Morgan laws)} \\
 & \nu\mathbb{Z}.\llbracket \varphi \rrbracket \cap_L \overline{\llbracket EX\neg\mathbb{Z} \rrbracket} \\
= & \text{(def. of } AX\text{)} \\
 & \nu\mathbb{Z}.\llbracket \varphi \rrbracket \cap_L \llbracket AX\mathbb{Z} \rrbracket
\end{array}$$

We now show that the definition and the fixpoint formulation of AU are equivalent. We start by reformulating the definition of AU:

$$\begin{array}{rl}
 & \llbracket A[\varphi \; U \; \psi] \rrbracket \\
= & \text{(def. of } AU\text{)} \\
 & \overline{\llbracket E[\neg\psi \; U \; \neg\varphi \wedge \neg\psi] \rrbracket} \; \cap_L \; \overline{\llbracket EG\neg\psi \rrbracket} \\
= & \text{(expanding } EU \text{ and } EG \text{ using their definitions)} \\
 & \overline{\mu\mathbb{Z}.\llbracket \neg\varphi \; \wedge \; \neg\psi \rrbracket \cup_L (\llbracket \neg\psi \rrbracket \cap_L \llbracket EX\mathbb{Z} \rrbracket)} \cap_L \overline{\nu\mathbb{Z}.\llbracket \neg\psi \rrbracket \cap_L \llbracket EX\mathbb{Z} \rrbracket} \\
= & \text{(negation fixpoint), (def. of } AX\text{)} \\
 & (\nu\mathbb{Z}.\llbracket \varphi \vee \psi \rrbracket \cap_L (\llbracket \psi \rrbracket \cup_L \llbracket AX\mathbb{Z} \rrbracket)) \cap_L (\mu\mathbb{Z}.\llbracket \psi \rrbracket \cup_L \llbracket AX\mathbb{Z} \rrbracket) \\
= & \text{(distributivity and absorption)} \\
 & (\nu\mathbb{Z}.\llbracket \psi \rrbracket \cup_L ((\llbracket \varphi \rrbracket \cup_L \llbracket \psi \rrbracket) \cap_L \llbracket AX\mathbb{Z} \rrbracket)) \cap_L (\mu\mathbb{Z}.\llbracket \psi \rrbracket \cup_L \llbracket AX\mathbb{Z} \rrbracket) \\
= & \text{(distributivity and absorption)} \\
 & (\nu\mathbb{Z}.\llbracket \psi \rrbracket \cup_L (\llbracket \varphi \rrbracket \cap_L \llbracket AX\mathbb{Z} \rrbracket)) \cap_L (\mu\mathbb{Z}.\llbracket \psi \rrbracket \cup_L \llbracket AX\mathbb{Z} \rrbracket)
\end{array}$$

Now, let $F(\mathbb{Z}) = \llbracket \psi \rrbracket \cup_L (\llbracket \varphi \rrbracket \cap_L \llbracket AX\mathbb{Z} \rrbracket)$ and $G(\mathbb{Z}) = \llbracket \psi \rrbracket \cup_L \llbracket AX\mathbb{Z} \rrbracket$. Then $\overline{\llbracket E[\neg\psi \; U \; \neg\varphi \wedge \neg\psi] \rrbracket} = \mathbb{K} = \nu\mathbb{Z}.F(\mathbb{Z})$, $\overline{\llbracket EG\neg\psi \rrbracket} = \mathbb{Y} = \mu\mathbb{Z}.G(\mathbb{Z})$, and $\llbracket A[\varphi \; U \; \psi] \rrbracket = \mathbb{W} = \mu\mathbb{Z}.F(\mathbb{Z})$. We need to show that $\mathbb{W} = \mathbb{K} \cap_L \mathbb{Y}$.

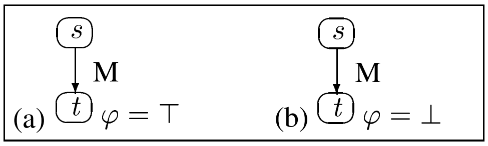

Fig. 8. Two examples where $AX\varphi$ does not give an intuitive answer.

Recall that if a function F is monotone and continuous, then there exists $n \in nat$ s.t. $\mu\mathbb{Z}.F(\mathbb{Z}) = F^n([\![\bot]\!])$ and $\nu\mathbb{Z}.F(\mathbb{Z}) = F^n([\![\top]\!])$. We also name ith iterations of F and G: $\mathbb{Z}_i = F^i([\![\top]\!])$, $\mathbb{W}_i = F^i([\![\bot]\!])$ and $\mathbb{Y}_i = G^i([\![\bot]\!])$.

We first show that $\forall i \in nat \cdot \mathbb{K}_i \cap_L \mathbb{Y}_i = \mathbb{W}_i$. The proof is by induction.

$$
\begin{array}{rl}
\text{Base Case:} & \mathbb{K}_0 \cap_L \mathbb{Y}_0 \\
= & \text{(def. of } \mathbb{K}_0, \mathbb{Y}_0\text{)} \\
 & [\![\top]\!] \cap_L [\![\bot]\!] \\
= & \text{(expanding mv-sets)} \\
 & [\![\bot]\!] \\
= & \text{(def. of } \mathbb{W}_0\text{)} \\
 & \mathbb{W}_0 \\
\text{IH:} & \text{Assume } \mathbb{K}_i \cap_L \mathbb{Y}_i = \mathbb{W}_i \text{ for } i = k \\
\text{Inductive Case:} & \text{Proof for } i = k+1 \\
 & \mathbb{K}_{k+1} \cap_L \mathbb{Y}_{k+1} \\
= & \text{(expanding each function once)} \\
 & ([\![\psi]\!] \cup_L ([\![\varphi]\!] \cap_L [\![AX\mathbb{K}_k]\!])) \cap_L ([\![\psi]\!] \cup_L [\![AX\mathbb{Y}_k]\!]) \\
= & \text{(distributivity and absorption)} \\
 & [\![\psi]\!] \cup_L ([\![\psi]\!] \cap_L [\![AX\mathbb{K}_k]\!] \cap_L [\![AX\mathbb{Y}_k]\!]) \\
= & (AX \text{ of conjunction)} \\
 & [\![\psi]\!] \cup_L ([\![\psi]\!] \cap_L [\![AX(\mathbb{K}_k \cap_L \mathbb{Y}_k)]\!]) \\
= & \text{(inductive hypothesis)} \\
 & [\![\psi]\!] \cup_L ([\![\psi]\!] \cap_L [\![AX(\mathbb{W}_k)]\!]) \\
= & \text{(def. of } \mathbb{W}_{k+1}\text{)} \\
 & \mathbb{W}_{k+1}
\end{array}
$$

Therefore, $\forall i \in nat \cdot \mathbb{W}_i = \mathbb{K}_i \cap_L \mathbb{Y}_i$ which implies that $\mathbb{W} = \mathbb{K} \cap_L \mathbb{Y}$ and thus the definition and the fixpoint formulation of AU are identical. □

We further note that χCTL's AX operator was defined via its relationship to EX. However, in some cases $AX\varphi \not\sqsubseteq EX\varphi$. For example, consider the two models in Figure 8. In part (a), $EX\varphi$ evaluates to M in s; therefore, we expect $AX\varphi$ to evaluate to M whereas it yields $\top$. In part (b), $EX\varphi$ evaluates to $\bot$ in s; therefore, we expect $AX\varphi$ to evaluate to $\bot$, whereas it yields M. The reason is that state s in both models in Figure 8 does not have any outgoing $\top$ transitions. A different formulation of AX (we will refer to it as AX_2) is possible:

$$[\![AX_2\varphi]\!] \triangleq [\![AX\varphi]\!] \cap_L [\![EX\varphi]\!]$$

This operator gives the expected values M and $\bot$, respectively, in the two examples in Figure 8. This operator reduces to AX if $\forall \varphi \cdot AX\varphi \sqsubseteq EX\varphi$. In particular, it reduces to AX on all models where every state has at least one outgoing $\top$ transition. Definition of AX_2 may be used instead of AX in computing AG, AF and AU fixpoints. In fact, this definition of AU has been proposed by other researchers [Bultan et al., 2000] for reasoning about non-Kripke structures. We plan to explore the tradeoffs between these two operators in our future work. Further, defining operators through AX will likely lead to defining all temporal operators directly through (multi-valued) μ-calculus rather than through χCTL's adequate set.

6.4 Running Time

To determine the running time of our algorithm, we note that this time is dominated by the fixpoint computations of EG and EU. The fixpoint computation convergence can take up to $|S| \times h$ iterations, where h is the height of the lattice $\mathcal{L}$. To understand the running time of each iteration, we need to analyze the running time of individual operations: mv-set union, intersection, complement, and backward image. The first three operations can be done in the time linear in the size of the representation of the corresponding mv-sets ($O(|S|)$). Backward image includes a disjunction ($O(|S|)$) of conjunctions between the formula and the transition relation (each taking $O(|S|)$, for the total time of $O(|S|^2)$). Thus, each fixpoint requires $O(|S| \times h \times |S|^2) = O(|S|^3 \times h)$. Since a given χCTL formula φ contains at most $|\varphi|$ different subformulas, the running time of our model-checking algorithm is $O(|S|^3 \times h \times |\varphi|)$. If instead of $|S|$ we use n to represent the size of the model, which equals $|S| + |R|$, then the overall running time is $O(h \times n^2 \times |\varphi|)$, which is the expected result for CTL model-checking [Clarke et al., 1999]. Finally, $|S| \leq |\mathcal{L}|^{|A|}$, so the running time of our model-checker is bounded above by $O(h \times |\mathcal{L}|^{3\times|A|} \times |\varphi|)$.

Note that our estimate of $|S|$ is too pessimistic. In multi-valued model-checking we can often compactly encode the state-space using the richer logic. For example, consider the following case: we have a classical model with n states and wish to differentiate between m of those states ($m << n$) by introducing an extra variable a. In classical model-checking this uncertainty can only be handled by duplicating each of $n-m$ states (one for each value of a). In fact, most of these states are likely to be reachable; thus, the size of the state space nearly doubles. In the multi-valued case, the reachable state-space will increase at most by m states. This computation did not take into an account the presence of uncertainty in transitions; these could also be encoded into the binary representation, but would lead to a further state-space increase. Thus, we conjecture that $|S| << |\mathcal{L}|^{|A|}$ because of the compact encoding of models using multiple values of the logic, and because many variable/value combinations are unreachable. Further, encoding mv-set operations using MDDs (multi-valued decision diagrams [Srinivasan et al., 1990]), allows us to achieve more practical running times. We address these issues in more detail in Section 8 and in the companion paper [Chechik et al., 2001b].

7. FAIRNESS

In this section we address the problem of multi-valued model-checking with fairness.

7.1 Intuition

In classical model-checking, it is often easier to specify all behaviors of the system being modeled, plus some additional "unwanted" behaviors, and then restrict the analysis to just the "wanted" behaviors of the system. This approach is often taken in practice, because it allows complicated systems to be specified more compactly. Since computations considered in classical model-checking are infinite, a natural way to partition behaviors into "wanted" and "unwanted" is by specifying progress that should be made on a fair computation (path). Thus, we define a computation as fair if and only if a certain progress state, or a sequence of states, occurs in it infinitely often [Clarke et al., 1986]. Formally, DEFINITION 11. *A path is fair w.r.t. a set of fairness conditions* $C = \{c_1, c_2, ..., c_n\}$ *iff every* c_i *is* TRUE *on it infinitely often.*

THEOREM 10. *The following statements are equivalent for a path* π *and fairness conditions* $C = \{c_1, ..., c_k\}$*:*

(1) *Each fairness condition* c_i *occurs infinitely often in* π*;*
(2) *A sequence* $c_1, c_2, ..., c_k$ *occurs infinitely often in* π.

Proof:
(1) ⇒ (2): Starting at the beginning of π, find the first occurrence of c_1 (this can always be done because c_1 occurs infinitely often in π). After that point, find the first occurrence of c_2, etc. This process can be repeated forever, and thus a sequence $c_1, c_2, ..., c_k$ occurs infinitely often in π.
(2) ⇒ (1): if a sequence $c_1, c_2, ..., c_k$ occurs infinitely often in π, then each element of it occurs infinitely often in π, so π is fair. □

7.2 Fairness in Multi-Valued Model-Checking

In the multi-valued case we want to preserve the ability to specify a larger set of computations than necessary and then restrict our attention to the "wanted", or fair ones. Multi-valued models already have a notion of "possible" computation: it is a computation where the conjunction of values of transitions between states is non-$\bot$, and "impossible" otherwise. Thus, if the goal of fairness in the multi-valued model-checking is to enable specification of a system in a concise form, fairness must be able to effectively turn some "possible" computations into "impossible" ones. Therefore, a "wanted" path in the fair system is a "possible" path conjoined with the appropriate fairness condition. Further, fair paths should preserve the "possibility" values of their underlying models, whereas unfair paths should have value $\bot$. One easy way to guarantee that is by ensuring that the fairness condition is 2-valued, because $\top$ and $\bot$ give us the desired base and identity laws ($x \sqcap \top = x$ and $x \sqcap \bot = \bot$). Thus, we assume that fairness constraints are given by a set of χCTL formulas $C = \{c_1, c_2, ..., c_n\}$ such that each formula always evaluates to either $\top$ or $\bot$. Intuitively, such expressions consist of boolean predicates ($\sqsubseteq$, $\sqsupseteq$, $=$, $\neq$) on χCTL formulas. Fairness conditions are formally defined by the following grammar:

BoolAtom	:=	χCTL expression Op χCTL expression
Op	$\in$	$\{\sqsubseteq, \sqsubset, \sqsupseteq, \sqsupset, =, \neq\}$
FairnessCond	:=	BoolAtom $\vee$ BoolAtom | BoolAtom $\wedge$ BoolAtom
		| $\neg$ BoolAtom | BoolAtom

For example, $AX\varphi$ may evaluate to M when the logic is **3** and thus cannot be used to specify a fairness condition. On the other hand, $\varphi \sqsubseteq AX\psi$ always evaluates to $\top$ or $\bot$ and thus can be used to specify fairness. Fairness conditions partition the sets of states into mv-sets. For notational convenience we assume that these mv-sets are over the same logic $L = (\mathcal{L}, \sqsubseteq, \neg)$ as the model, even though for each fairness condition c_i, $\forall s \in S \cdot [\![c_i]\!](s) \in \{\top, \bot\}$.

DEFINITION 12. *A trace in a* χ*Kripke structure* M *is* fair *w.r.t. a set of fairness conditions* $C = \{c_1, c_2, ..., c_n\}$ *iff each computation comprising it is fair w.r.t. C.*

Following Huth and Ryan [Huth and Ryan, 2000], we write A_C and E_C for the operators A and E restricted to fair paths. For example, $[\![A_C G\varphi]\!](s) = \top$ means that φ is TRUE in every fair trace.

7.3 Fair EG

As in CTL, χCTL operators $E_C G$, $E_C[\varphi\ U\ \psi]$ and $E_C X$ form an adequate set. We start by giving a formulation for $[\![E_C G\varphi]\!](s) = \ell$. This formula means that there exists a trace beginning with state s on which $EG\varphi$ holds with value ℓ, and each formula in C is $\top$ infinitely often along each path. Alternatively, if $C = \{c_1, c_2\}$, it is the repetition of the following sequence: φ holds until c_1, and from that point on, φ holds until c_2. Formally, we can define this using the following fixpoint formulation:

$$[\![E_C G\varphi]\!] \triangleq \nu\mathbb{Z}.[\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_1 \wedge EXE[\varphi\ U\ \varphi \wedge c_2 \wedge \mathbb{Z}]]]\!] \qquad \text{(def. 1 of } E_C G)$$

When $C = \{c_1, c_2, ..., c_k\}$, the above definition can be extended appropriately. The problem with this definition, however, is that it is dependent on the size of C. We thus seek an alternative definition:

$$[\![E_C G\varphi]\!] \triangleq \nu\mathbb{Z}.[\![\varphi]\!] \cap_L \bigcap\nolimits_{L\,k=1}^{n} [\![EXE[\varphi\ U\ \varphi \wedge \mathbb{Z} \wedge c_k]]\!] \qquad \text{(def. 2 of } E_C G)$$

We are now ready to study properties of the two definitions of $E_C G$. We begin by showing that Definition 2 reduces to that of EG when there are no fairness conditions present, and then proceed to show that the two definitions of $E_C G$ are equivalent.

THEOREM 11. *When $C = \{\top\}$ (no fairness), Definition 2 of $E_C G$ reduces to*

$$[\![E_C G\varphi]\!] = \nu\mathbb{Z}.[\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge \mathbb{Z}]]\!] = \nu\mathbb{Z}.[\![\varphi]\!] \cap_L [\![EX\mathbb{Z}]\!] = [\![EG\varphi]\!]$$

Proof:
Let $F(\mathbb{Z}) = [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge \mathbb{Z}]]\!]$ and $G(\mathbb{Z}) = [\![\varphi]\!] \cap_L [\![EX\mathbb{Z}]\!]$. Then, by Definition 2 of $E_C G$, $[\![E_C G\varphi]\!] = \mathbb{W} = \nu\mathbb{Z}.F(\mathbb{Z})$ and $[\![EG\varphi]\!] = \mathbb{Y} = \nu\mathbb{Z}.G(\mathbb{Z})$. We start by proving an intermediate result indicating that $[\![E[\varphi\ U\ \varphi \wedge \mathbb{W}]]\!]$ is the same as $\mathbb{W}$:

$$\begin{aligned}
& [\![E[\varphi\ U\ \varphi \wedge \mathbb{W}]]\!] \\
= \ & (EU \text{ fixpoint}) \\
& ([\![\varphi]\!] \cap_L \mathbb{W}) \cup_L ([\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge \mathbb{W}]]\!]) \\
= \ & \text{(absorption), (def. of } F) \\
& \mathbb{W} \cup_L F(\mathbb{W}) \\
= \ & \text{(since } \mathbb{W} \text{ is a fixpoint of } F) \\
& \mathbb{W}
\end{aligned}$$

To show that $[\![E_C G\varphi]\!] = [\![EG\varphi]\!]$ we need to show that $\mathbb{Y} = \mathbb{W}$. The proof consists of two parts: (1) showing that $\mathbb{W}$ is a fixpoint of G and thus $\mathbb{W} \subseteq_L \mathbb{Y}$ (because $\mathbb{Y}$ is the greatest fixpoint of G); and (2) showing that $\mathbb{W} \supseteq_L \mathbb{Y}$.

$$\begin{aligned}
(1)\ & G(\mathbb{W}) \\
= \ & \text{(definition of } G) \\
& [\![\varphi]\!] \cap_L [\![EX\mathbb{W}]\!] \\
= \ & \text{(since } \mathbb{W} = [\![E[\varphi\ U\ \varphi \wedge \mathbb{W}]]\!]) \\
& [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge \mathbb{W}]]\!] \\
= \ & \text{(definition of } F) \\
& F(\mathbb{W}) \\
= \ & (\mathbb{W} \text{ is the fixpoint of } F) \\
& \mathbb{W}
\end{aligned}$$

To prove (2) we start by defining $\mathbb{W}_i = F^i([\![\top]\!])$ and $\mathbb{Y}_i = G^i([\![\top]\!])$. Since F and G are monotone and continuous, there exists $n \in nat$ s.t. $\mathbb{W} = \mathbb{W}_n \ \wedge\ \mathbb{Y} = \mathbb{Y}_n$. We now show that $\forall i \in nat \cdot \mathbb{W}_i \supseteq_L \mathbb{Y}_i$.

Base Case: $\mathbb{W}_0 = [\![\top]\!] = \mathbb{Y}_0$
IH: Assume $\mathbb{W}_i \supseteq_L \mathbb{Y}_i$ for $i = k$
Inductive Case: Proof for $i = k + 1$
Note that $\mathbb{W}_{k+1} = [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge \mathbb{W}_k]]\!] = G([\![E[\varphi\ U\ \varphi \wedge \mathbb{W}_k]]\!])$
and $\mathbb{Y}_{k+1} = [\![\varphi]\!] \cap_L [\![EX\mathbb{Y}_k]\!] = G(\mathbb{Y}_k)$
$G([\![E[\varphi\ U\ \varphi \wedge \mathbb{W}_k]]\!]) \supseteq_L G(\mathbb{Y}_k)$
$\Leftarrow$ (G is monotone)
$[\![E[\varphi\ U\ \varphi \wedge \mathbb{W}_k]]\!] \supseteq_L \mathbb{Y}_k$
$\Leftarrow$ (EU fixpoint)
$(\mathbb{W}_k \cup_L ([\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge \mathbb{W}_k]]\!])) \supseteq_L \mathbb{Y}_k$
$\Leftarrow$ (monotonicity of $\cup_L$), (absorption)
$\mathbb{W}_k \supseteq_L \mathbb{Y}_k$
$\Leftarrow$ (inductive hypothesis)
$\top$

Thus, by induction, $\forall n \in nat \cdot \mathbb{W}_n \supseteq_L \mathbb{Y}_n$, so $\mathbb{W} \supseteq_L \mathbb{Y}$. Combining this with results of part (1), we get that $\mathbb{W} = \mathbb{Y}$, so, using Definition 2 of $E_C G$, $[\![E_C G\varphi]\!] = [\![EG\varphi]\!]$. □

Now we set out to show that Definition 1 and Definition 2 of $E_C G$ are equivalent. We assume that $C = \{c_1, c_2\}$ for brevity. The reasoning can be expanded for an arbitrary $C = \{c_1, ..., c_k\}$. Let $F(\mathbb{Z}) = [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_1 \wedge EXE[\varphi\ U\ \varphi \wedge c_2 \wedge \mathbb{Z}]]]\!]$. Then, by Definition 1, $E_C G\varphi = \nu\mathbb{Z}.F(\mathbb{Z})$. Also, let $G(\mathbb{Z}) = [\![\varphi]\!] \cap_L \bigcap_{k=\{1,2\}}^{L} [\![EXE[\varphi\ U\ \varphi \wedge c_k \cap_L \mathbb{Z}]]\!]$. Then, by Definition 2, $E_C G\varphi = \nu\mathbb{Z}.G(\mathbb{Z})$.

We define $\mathbb{K}_i$ and $\mathbb{M}_i$ to represent all states that are beginning of the path on which the sequence c_1, c_2 (respectively, c_2, c_1) holds i times. $\mathbb{K}_i$ and $\mathbb{M}_i$ are defined recursively as

$$\begin{array}{ll} \mathbb{K}_0 = [\![\top]\!] & \mathbb{M}_0 = [\![\top]\!] \\ \mathbb{K}_n = [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_1 \wedge \mathbb{M}_{n-1}]]\!] & \\ \mathbb{M}_n = [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_2 \wedge \mathbb{K}_{n-1}]]\!] & \end{array}$$

We also define the nth iteration of $\nu\mathbb{Z}.G(\mathbb{Z})$ explicitly:

$$\begin{array}{rcl} G^0([\![\top]\!]) & = & [\![\top]\!] \\ G^{n+1}([\![\top]\!]) & = & G_1(G^n([\![\top]\!])) \cap_L G_2(G^n([\![\top]\!])) \end{array}$$

Note that $\mathbb{K}_{2n} = F^n([\![\top]\!])$. We are therefore interested in the degree to which $\mathbb{K}_n$ and $\mathbb{M}_n$ approximate $G^n([\![\top]\!])$. We characterize this formally in the following lemma:

LEMMA 1. $\forall n \in nat$,

$$\begin{array}{ll} (1)\ \mathbb{K}_n \supseteq_L G^n([\![\top]\!]) & (2)\ \mathbb{M}_n \supseteq_L G^n([\![\top]\!]) \\ (3)\ \mathbb{K}_{2n} \subseteq_L G^n([\![\top]\!]) & (4)\ \mathbb{M}_{2n} \subseteq_L G^n([\![\top]\!]) \end{array}$$

Proof:
In the proof we will use the following results, proofs of which are omitted for brevity:

$\mathbb{K}_n \supseteq_L \mathbb{K}_{n+1}$ and $\mathbb{M}_n \supseteq_L \mathbb{M}_{n+1}$ (monotonicity of $\mathbb{K}_n$, $\mathbb{M}_n$)
$[\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ c_2 \wedge \varphi \wedge \mathbb{K}_n]]\!] \subseteq_L \mathbb{M}_n$ (relation between $\mathbb{K}_n$ and $\mathbb{M}_n$)
(holds because $\mathbb{M}_n = [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_2 \wedge \mathbb{K}_{n-1}]]\!]$)
$[\![E[\varphi\ U\ \psi]]\!] \supseteq_L [\![E[\varphi\ U\ \varphi \wedge EXE[\varphi\ U\ \psi]]]\!]$ (monotonicity 1 of EU)
(holds because of EU expansion)
$[\![\varphi]\!] \supseteq_L [\![\psi]\!] \Rightarrow [\![E[p\ U\ \varphi]]\!] \supseteq_L [\![E[p\ U\ \psi]]\!]$ (monotonicity 2 of EU)

We are now ready to prove (1)-(4), which we do by induction on n.

Base Case: $G^0([\![\top]\!]) = \top$ (def. of $G^0([\![\top]\!])$)
IH: Assume (1)-(4) hold for $n = k$
Inductive Case: Proof for $n = k + 1$
(1) Enough to show $\mathbb{K}_{n+1} \supseteq_L G_1(G^n([\![\top]\!]))$
$\top$
$\Rightarrow$ (IH)
$\mathbb{M}_n \supseteq_L G^n([\![\top]\!])$
$\Rightarrow$ (monotonicity 2 of EU)
$[\![\varphi]\!] \cap_L [\![EXE[\varphi\, U\, \varphi \wedge c_1 \wedge \mathbb{M}_n]]\!]$
$\supseteq_L [\![\varphi]\!] \cap_L [\![EXE[\varphi\, U\, \varphi \wedge c_1 \wedge G^n([\![\top]\!])]]\!]$
$\Rightarrow$ (def. of $\mathbb{K}_{n+1}$, G_1, G^{n+1})
$\mathbb{K}_{n+1} \supseteq_L G_1(G^n([\![\top]\!])) \supseteq_L G^{n+1}([\![\top]\!])$
(2) Proof is similar to that of (1).
(3) Need to show $\mathbb{K}_{2n+2} \subseteq_L G^{n+1}([\![\top]\!])$
We will show $(3a)$ $\mathbb{K}_{2n+2} \subseteq_L G_1(G^n([\![\top]\!]))$
$(3b)$ $\mathbb{K}_{2n+2} \subseteq_L G_2(G^n([\![\top]\!]))$
Then by $\cap_L$ elimination, we will have the desired property.
$(3a)$ $\mathbb{K}_{2n+2}$
$=$ (def. of $\mathbb{K}_{2n+2}$)
$[\![\varphi]\!] \cap_L [\![EXE[\varphi\, U\, \varphi \wedge c_1 \wedge EXE[\varphi\, U\, \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]]\!]$
$\subseteq_L$ (relation between $\mathbb{K}_n$ and $\mathbb{M}_n$))
$[\![\varphi]\!] \cap_L [\![EXE[\varphi\, U\, \varphi \wedge c_1 \wedge \mathbb{M}_{2n}]]\!]$
$\subseteq_L$ (IH and monotonicity)
$[\![\varphi]\!] \cap_L [\![EXE[\varphi\, U\, \varphi \wedge c_1 \wedge G^n([\![\top]\!])]]\!]$
$=$ (def. of G_1)
$G_1(G^n([\![\top]\!]))$
$(3b)$ $\mathbb{K}_{2n+2}$
$=$ (def. of $\mathbb{K}_{2n+2}$)
$[\![\varphi]\!] \cap_L [\![EXE[\varphi\, U\, \varphi \wedge c_1 \wedge EXE[\varphi\, U\, \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]]\!]$
$\subseteq_L$ (monotonicity)
$[\![\varphi]\!] \cap_L [\![EXE[\varphi\, U\, \varphi \wedge EXE[\varphi\, U\, \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]]\!]$
$\subseteq_L$ (monotonicity 1 of EU)
$[\![\varphi]\!] \cap_L [\![EXE[\varphi\, U\, \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]\!]$
$\subseteq_L$ (IH and monotonicity)
$[\![\varphi]\!] \cap_L [\![EXE[\varphi\, U\, \varphi \wedge c_2 \wedge G^n([\![\top]\!])]]\!]$
$=$ (def. of G_2)
$G_2(G^n([\![\top]\!]))$
(4) Proof is similar to that of (3).

□

THEOREM 12. *Definition 1 and Definition 2 of $E_C G\varphi$ are equivalent.*

Proof:
Recall that $\mathbb{K}_{2n} = F^n([\![\top]\!])$, defined above. Since F^n converges, $\exists N_1 \in nat \cdot \forall n \geq N_1 \cdot \nu\mathbb{Z}.F(\mathbb{Z}) = \mathbb{K}_{2n}$. Since G^n converges, $\exists N_2 \in nat \cdot \forall n \geq N_2 \cdot \nu\mathbb{Z}.G(\mathbb{Z}) = G^n([\![\top]\!])$. Further, by Lemma 1, $\forall n \geq N_2$

$$\begin{array}{lclcl} \mathbb{K}_{2n} & \subseteq_L & G^n([\![\top]\!]) & = & \nu\mathbb{Z}.G(\mathbb{Z}) \\ \mathbb{K}_{2n} & \supseteq_L & G^{2n}([\![\top]\!]) & = & \nu\mathbb{Z}.G(\mathbb{Z}) \end{array}$$

So, $\forall n \geq N_2 \cdot \mathbb{K}_{2n} = \nu\mathbb{Z}.G(\mathbb{Z})$.

Let $m = \max\{N_1, N_2\}$. Then, $\nu\mathbb{Z}.F(\mathbb{Z}) = \mathbb{K}_{2m} = \nu\mathbb{Z}.G(\mathbb{Z})$. So, the two definitions for $E_C G$ are equivalent. □

7.4 Fairness in Other χCTL Operators

Computing $E_C X\varphi$ in s amounts finding successors of s which are at the start of some fair computation path, and computing $EX\varphi$ using only these successors. In such states $E_C G\top$ has a value other than $\perp$. Thus, the formulation for $E_C X\varphi$ is

$$[\![E_C X\varphi]\!] \triangleq [\![EX(\varphi \wedge (E_C G\top \sqsupset \perp))]\!]$$

For a similar reason, the formulation for $E_C[\varphi\ U\ \psi]$ is

$$[\![E_C[\varphi\ U\ \psi]]\!] = [\![E[\varphi\ U\ (\psi \wedge (E_C G\top \sqsupset \perp))]]\!]$$

Note that both formulations are similar to those of classical CTL.

7.5 Running Time

Running time of the model-checker under fairness conditions C is dominated by the computation of $E_C G$ which includes a nested fixpoint. The inner fixpoint is used to compute $E[\varphi\ U\ (\mathbb{Z} \wedge c_k)]$ and takes, using a regular model-checking algorithm, $O(h \times |S|^3)$. The outer fixpoint converges after at most $h \times |S|$ iterations, and each iteration involves computing $|C|$ inner fixpoints. Thus, $E_C G\varphi$ can be computed in $O(|C| \times h^2 \times |S|^4)$ time. Since a given formula φ contains at most $|\varphi|$ different subformulas, the running time of the model-checker under fairness conditions C is $O(|\varphi| \times |C| \times h^2 \times |S|^4)$. Note that running time for χCTL with fairness reduces to that of the classical CTL model-checker when the underlying logic is classical.

8. IMPLEMENTATION AND APPLICATIONS

In this section we briefly discuss implementation choices for χChek and describe some potential applications for it.

8.1 Implementation

As a proof of concept, we have developed a prototype implementation (in Java) of a multi-valued model-checker called χChek. The checking engine takes as input a list of χCTL formulas to verify, a model of the system represented as a χKripke structure, and a specification of the underlying quasi-boolean logic. For each χCTL formula, χChek calculates its logic value in the initial state, returning a counter-example or a witness, if appropriate. The counter-example generator for χCTL is discussed in detail elsewhere [Chechik and Gurfinkel, 2001].

Practical symbolic model-checking in a given domain (probabilistic, multi-valued, timed, etc.) depends on efficient algorithms for storing and manipulating sets of states in which a property holds. In our case, we need efficient implementations of operations (union, intersection, complement and backward image) on the mv-set datatype. χChek uses a general mv-set interface, and we are experimenting with alternative implementations of mv-sets. As in classical symbolic model-checking, we represent mv-sets using decision diagrams. Several varieties of decision diagrams are suitable. For example, if the mv-set membership function is kept multi-valued, then mv-sets can be easily implemented using Multi-Valued Decision Diagrams (MDDs) [Srinivasan et al., 1990]. Alternatively, each mv-set can be thought of as a collection of classical sets. Symbolically, this approach can be implemented using Multi-Valued Binary-Terminal Decision Diagrams (MBTDDs) [Sasao and Butler, 1996]. The tradeoffs between these encodings depend on the number of factors, including the types of questions asked, the size and the shape of the logic, and the variable ordering. We describe the implementation and evaluate the performance characteristics of different implementations of the mv-set datatype in a companion paper [Chechik et al., 2001b].

8.2 Applications

Multi-valued model-checking has a number of potential applications in software engineering. We can characterize different applications by considering what the intermediate values in the logic could be used to represent, and how a model-checker could be used to support different kinds of analysis. For example:

—The intermediate values of the logic could represent incomplete information (or uncertainty). Such applications typically use a 3-valued logic. A 3-valued model can be interpreted as a compact representation for a set of *completions* [Bruns and Godefroid, 2000]. For example, a completion of a partial model can be generated by replacing each MAYBE value by either $\top$ or $\bot$. If a property is $\top$ (respectively, $\bot$) in a partial model, then it is $\top$ ($\bot$) in all completions. Thus, we can use a 3-valued model-checker to determine if particular properties hold, even though the model is incomplete. We can also use this approach to reduce the size of classical model-checking problems by creating abstractions of models with large state-spaces. We describe one such case study below.

—The intermediate values of the logic could represent disagreement. Such applications typically use logics defined over product lattices. For example, in Section 2, we used a 9-valued logic to explicitly capture disagreements when we merged two 3-valued models. If a property is $\top$ (respectively, $\bot$) in each individual model, then it will be $\top$ ($\bot$) in the merged model. However, we can use a multi-valued model-checker to check properties that are not expressible in the individual models, because they combine the vocabulary of different models or because they refer to interactions between the individual models. We are exploring this approach for the feature interaction problem in telephony [Easterbrook and Chechik, 2001], and as a tool to support stakeholder negotiations in requirements engineering by tracing from specific disagreements to the properties they affect.

—The intermediate values of the logic could represent relative desirability (or priority). Such applications typically use chain lattices (total orders) with an odd number of values. The middle value of the lattice is interpreted as ”don’t care”, and the remaining values represent greater desirability (respectively, undesirability) as we move towards $\top$ ($\bot$). We interpret an n-valued model as representing $(n-1)/2$ layers, where each layer adds behaviors to the previous ones. We are exploring this approach for reasoning about requirements prioritization and survivable systems. In each of these applications, the multi-valued model checker allows us to check which properties are supported by which layer.

We now demonstrate the use of χChek for reasoning about state-space abstraction.

The use of abstraction has long been proposed as a way to overcome the state-space explosion problem in classical model-checking. Abstraction collapses sets of concrete states into a single abstract state, thus indicating that any differences between the concrete states within a single abstract state are ignored [Dams et al., 1997; Dams, 1996]. One

(a)
```
MODULE Button (reset)
VAR
  pressed : boolean;
ASSIGN
  init (pressed)  := 0;
  next (pressed) :=
    case
      reset    : 0;
      pressed : 1;
      1          : {0, 1};
    esac
```

(b)
```
MODULE Button (reset)
VAR
  pressed : boolean;
  button   : boolean;
ASSIGN
  init (button)  := 0;
  next (button) := {0, 1};

  init (pressed)  := 0;
  next (pressed) :=
    case
      button : 1;
      reset   : 0;
      1          : pressed;
    esac
```

(c)
```
MODULE Button (reset)
VAR
  pressed : boolean;
  button   : {T, M, F};
ASSIGN
  button :=
    case
      reset | pressed : M
      1                        : {0, 1}
    esac
  init (pressed)  := 0;
  next (pressed) :=
    case
      button = T : 1;
      reset         : 0;
      1                : pressed;
    esac
```

Fig. 9. Three models of the elevator button: (a) the original module `Button` of Plath & Ryan (in SMV); (b) a modified module `Button`; (c) an abstracted module `Button`.

way to ensure property preservation during abstraction is to guarantee that a set of states in the concrete model is composed into a single abstract state only if these states have an equivalent transition relation (one can refer to them as *symmetric*). For example, states s_1 and s_2 can become part of the same abstract state if

$$\forall t \cdot (R(s_1,t) \Leftrightarrow R(s_2,t)) \ \wedge \ (R(t,s_1) \Leftrightarrow R(t,s_2))$$

If states earmarked for abstraction into the same abstract state disagree on a value of a variable, we cannot assign this variable any of the values of classical logic in the abstract state. Instead, a 3-valued logic can be used, since disagreement is naturally modeled with value MAYBE (M). This abstraction guarantees *state-wise preservation* [Dams et al., 1997]: if a formula evaluates to $\top$ ($\bot$) in an abstract state, it evaluates to $\top$ ($\bot$) in the corresponding concrete state.

For the case study, we have used the SMV elevator model of Plath & Ryan [Plath and Ryan, 1999]. This model consists of a single elevator which accepts requests made by users pressing buttons on the floor landings and from inside the elevator. The elevator moves up and down between floors and opens and closes its doors in response to these requests, according to Single Button Collective Control (SBCC) [Berney and dos Santos, 1985].

The model is implemented as several SMV modules. The `Main` module declares several instances of the module `Button` (one per floor, called $\texttt{landingBut}_i$), parameterized by the condition under which the request is considered fulfilled (`reset`), and one instance of the module `Lift`, called `lift`. The `Lift` module declares the variables `floor`, `door` and `direction` as well as further instances of `Button` to indicate requests from within the elevator (also one per floor, called $\texttt{liftBut}_i$).

The SMV module `Button` is shown in Figure 9(a). Once a button is pressed, it latches and remains pressed until the elevator fulfills the request. We modified the module by modeling the latching explicitly: each variable `pressed` in the module `Button` is decomposed into two variables, with `button` representing the actual button that users can press, and `pressed` representing the latching. The modified button is shown in Figure 9(b), and its

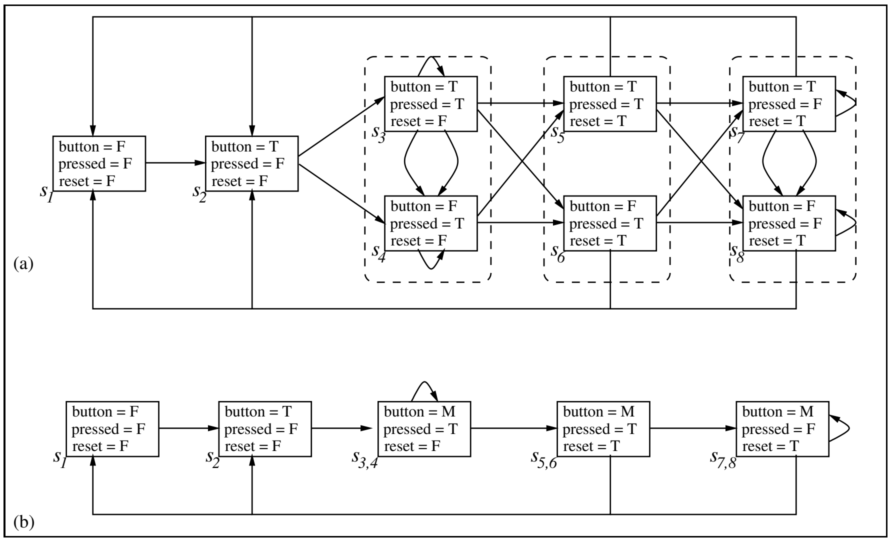

Fig. 10. State machines: (a) of the modified module `Button`; (b) of the abstracted module `Button`..

state machine – in Figure 10(a). The model has eight possible states, corresponding to the evaluation of the tuple (`button`, `pressed`, `reset`). For simplicity of the presentation, in the state-machine description we assumed that a button cannot be reset until it has been latched, i.e., `reset` cannot become $\top$ if `pressed` is $\bot$.

We also noted that pairs of states s_3 and s_4, s_5 and s_6 and s_7 and s_8 (indicated by dashed lines in Figure 10(a)) have symmetric transition relations and thus can be abstracted. This corresponds to the value of `button` being irrelevant when `pressed` or `reset` are $\top$. Thus, we can model `button` by a 3-valued variable, as shown in Figure 9(c). The state machine model of the abstract system is shown in Figure 10(b). When this module is composed with the rest of the elevator model, we get a 3-valued model which cannot be directly verified using a classical model-checker. We proceed with the verification using two techniques (a) a method of Bruns & Godefroid, and (b) using χChek.

The resulting model can be verified using the method of Bruns & Godefroid [Bruns and Godefroid, 1999; Bruns and Godefroid, 2000] via two queries to a classical model-checker. The method is only applicable to formulas that do not contain negations. Hence, the first step is to compute the *complement closure* [Bruns and Godefroid, 2000] of the model by adding an extra variable $\bar{a}$ for each variable a, such that in each state of the model, $\bar{a}$ is equal to $\neg a$. The second step is to build two versions of the model: the *pessimistic* version replaces each M value with $\bot$, while the *optimistic* version replaces each M with $\top$. The property to be checked must also be converted into the positive normal form, by pushing all negations to the level of atomic propositions and replacing each negated variable a with the corresponding variable $\bar{a}$. The model-checker is called on the pessimistic and the optimistic models, and the results are combined as follows: if the pessimistic model yields $\top$, return $\top$; else if the optimistic model yields $\bot$, return $\bot$; otherwise, return M.

The properties of the elevator system that we verified appear in Figure 11. These are taken directly from [Plath and Ryan, 1999], with the following exceptions: (a) we replaced variable `pressed` by `button` in all terms involving landing or elevator buttons because of our change to the module `Button`; (b) we added property 5; (c) we selected only a subset of

1. "If the door closes, it will eventually open":
   $AG(\texttt{lift.door} = \texttt{closed} \rightarrow AF\ \texttt{lift.door} = \texttt{open})$
2. "Pressing a landing button guarantees that the lift will arrive at that landing and open its doors":
   $AG(\texttt{landingBut2.button} \rightarrow AF(\texttt{lift.floor} = 2 \wedge \texttt{lift.door} = \texttt{open}))$
3. "If a button inside the lift is pressed, the lift will eventually arrive at the corresponding floor":
   $AG(\texttt{lift.liftBut2.button} \rightarrow AF(\texttt{lift.floor} = 2 \wedge \texttt{lift.door} = \texttt{open}))$
4. "The lift may stop at floor 2 for landing calls when traveling downwards":
   $\neg AG((\neg\texttt{lift.floor} = 2\ \wedge\ \neg\texttt{lift.liftBut2.button}\ \wedge\ \texttt{lift.direction} = \texttt{down})$
   $\rightarrow \texttt{lift.door} = \texttt{closed})$
5. "Whenever a button indicator is on, a button is being pressed"
   $AG(\texttt{lift.liftBut2.pressed} \rightarrow \texttt{lift.liftBut2.button})$

Fig. 11. Properties of the elevator system.

properties from [Plath and Ryan, 1999]. Similar properties can be formulated for all other floors and all other landing and elevator buttons. In a correct elevator system, we expect property 1 to evaluate to $\bot$, properties 2-4 to evaluate to $\top$, and property 5 to evaluate to $\bot$.

Given a 2-valued logic as input, χChek acts as a classical model-checker. Thus, we can compare the two approaches using the same model-checking engine, and hence factor out implementation issues in the experiments. We parameterized the model by the number of floors, and ran our experiments using models with 3 and 4 floors. In the classical and in the multi-valued approach, we ran χChek with mv-sets implemented using MDDs on a Pentium III with 850 MHz processor and 256 MB RAM, running Sun JDK 1.3 under Linux 2.2.19. Table II summarizes the results. For example, verification of property 1 on the 4-floor elevator can be done in $4.638 + 4.594 = 9.232$ seconds using two queries to "classical" χChek, and in 2.29 seconds when χChek uses 3-valued logic directly. We add the running times of the two queries because in general both queries need to be performed to obtain the answer. Further, note that property 5 evaluates to M in both models, as opposed to the expected $\bot$. This was caused by the abstraction we made to the module `Button`: it is possible that `pressed` is $\top$ while `button` is M.

The size of the transition relation does not change from property to property, so in Table II we show it only once for each elevator model. The process of constructing the complement closure in the Bruns & Godefroid approach substantially increases the size of the models and thus significantly slows down the analysis, as confirmed by the experiments. An additional problem with this approach is that it does not directly support negative properties; thus, the model-checker needs to explicitly implement the algorithm for each CTL operator, that is, the adequate set for CTL should include all CTL operators.

Finally, note that in our comparisons we did not optimize either method. Potentially, by determining which atomic propositions are negated in the formulas to be verified and only including these in the complement closure, the size of the transition relation and the running time in the Bruns & Godefroid method can be improved. However, in the case of the elevator model, most of the atomic propositions can appear on the left-hand side of the implication, and thus need to be negated. Multi-valued model-checking can also be improved: currently we represent all variables in multi-valued logic, e.g., `pressed` and `reset` are 3-valued, even though they do not need to be. Representing boolean variables explicitly can lead to faster verification times.

Model	CTL Property Number	Result	Bruns & Godefroid			χChek
			Pessimistic	Optimistic	Total	
3-floor	1.	$\bot$	0.756 s	0.774 s	1.53 s	0.306 s
	2.	$\top$	0.273 s	0.182 s	0.455 s	0.114 s
	3.	$\top$	0.249 s	0.173 s	0.422 s	0.119 s
	4.	$\top$	0.608 s	0.634 s	1.242 s	0.299 s
	5.	M	0.087 s	0.155 s	0.242 s	0.105 s
Size of trans. relation			2130	2153		954
4-floor	1.	$\bot$	4.638 s	4.594 s	9.232 s	2.29 s
	2.	$\top$	0.936 s	0.942 s	1.878 s	0.463 s
	3.	$\top$	0.869 s	1.044 s	1.913 s	0.494 s
	4.	$\top$	3.767 s	3.698 s	7.465 s	2.122 s
	5.	M	0.047 s	0.502 s	0.549	0.298 s
Size of trans. relation			5249	5307		2367

Table II. Elevator abstraction: comparison between two runs of the classical model-checker and a run of χChek..

9. CONCLUSION

In this section we summarize the paper, compare the work presented here with that of other researchers, and outline directions for future work.

9.1 Summary

Multi-valued logics can be useful for describing models that contain partial information or inconsistencies. In particular, they apply well for reasoning about information gathered from multiple sources. Not only can multi-valued logics allow formal reasoning when inconsistencies are present, they help determine which inconsistencies matter, and what the sources of these inconsistencies are.

In this paper we presented an extension of classical CTL model-checking to reasoning about a class of multi-valued logics called *quasi-boolean* logics. We gave semantics to and categorized properties of a multi-valued extension of CTL, called χCTL, described a notion of multi-valued Kripke structures, and showed how model-checking can be extended to dealing with systems containing fairness. We have also implemented a multi-valued symbolic model-checker χChek, and illustrated its behaviour and performance.

The methodology for using χChek for understanding the impact of inconsistencies and for negotiation is yet to be validated on case studies. However, preliminary results show that the ability to change the logic, and thus the level of abstraction of treatment of inconsistencies, is indispensable for understanding the source and impact of inconsistencies. This can be done naturally and effectively using χChek.

9.2 Related Work

Multi-valued logics have been explored for a variety of applications in databases [Gaines, 1979], knowledge representation [Ginsberg, 1987], machine learning [Michalski, 1977], and circuit design [Hazelhurst, 1996]. A number of specific propositional multi-valued logics have been proposed and studied. For example, Łukasiewicz [Łukasiewicz, 1970] first introduced a 3-valued logic to allow for propositions whose truth values are 'unknown'. and Kleene [Kleene, 1952] introduced several alternative 3-valued logics. Belnap [Belnap, 1977] proposed a 4-valued logic that also introduces the value "both" (i.e. TRUE *and* FALSE), to handle inconsistent assertions in database systems. Each of these logics can be generalized to allow for additional levels of uncertainty or disagreement. The class of quasi-boolean logics defined in this paper includes many existing multi-valued propositional logics, including those of Kleene and Belnap. Work has also been done on deciding a more general class of multi-valued logics. In particular, the work of Hähnle and others [Hähnle, 1994; Sofronie-Stokkermans, 2000] has led to the development of several theorem-provers for first-order multi-valued logics. Multi-valued extensions of models logics have been explored by Fitting [Fitting, 1991; Fitting, 1992] who suggests two different approaches for doing this: the first extends the interpretation of atomic formulas in each world to be multi-valued; the second also allows multi-valued accessibility relations between worlds. The latter approach is the one we adopted for χCTL, as it is more general.

Several authors [Bruns and Godefroid, 1999; Bruns and Godefroid, 2000; Huth et al., 2001; Godefroid et al., 2001] discuss model-checking over 3-valued logic. Bruns and Godefroid [Bruns and Godefroid, 1999; Bruns and Godefroid, 2000] have introduced Kripke structures with 3-valued state variables and boolean transitions representing *partial state-spaces*, referring to them as *partial Kripke structures*. They extended branching-time temporal logic to this case, proposing 3-valued logic for expressing properties of partial models. Model-checking of positive properties (properties that do not contain negation) in this logic reduces to two questions to a classical model-checker. This approach can be also applied to full μ-calculus by computing *complement closure* [Bruns and Godefroid, 2000] of the model at the expense of increasing the size of the model and the verification time. To make the analysis more precise, the authors describe *thorough semantics* of 3-valued model-checking under which a property evaluates to MAYBE if and only if there are two refinements of the partial model that disagree on the value of this property.

Huth et al. [Huth et al., 2001; Godefroid et al., 2001] apply the *modal transition systems* (MTS) of Larsen and Thomsen [Larsen and Thomsen, 1988] to problems which have been treated with 3-valued logic, such as the partial specifications of Bruns and Godefroid, above, and the 3-valued program analysis of Sagiv, Reps, and Wilhelm [Sagiv et al., 1999]. These systems have "must", "may", and "must not" type transitions. The authors define a 3-valued extension of the modal μ-calculus for MTS and describe an algorithm for model-checking queries in a fragment of this language using classical model-checking.

When 3-valued logics are applied to reasoning about inconsistencies, all inconsistencies are represented using the value M. When model-checking returns $\top$ or $\bot$, this indicates that inconsistencies do not matter. However, when model-checking returns M, there is insufficient support for discovering sources of inconsistencies or for negotiation. If model-checking on a larger class of logics is possible, such as with χChek, we can refine the logic when model-checking returns M, e.g., keeping track of exact sources of all disagreements, and thus provide direct support for understanding which inconsistencies matter.

Work of Huth and Pradhan [Huth and Pradhan, 2001] attempts to aid in discovering sources of inconsistencies between multiple viewpoints. Each viewpoint is a possibly incomplete but consistent description over the same global vocabulary. Systems are on different levels of abstraction. The methodology assumes presence of a dominance preorder of experts, with inconsistencies appearing if for a given expert, there are two experts dominating it, which disagree on the value of a property. All model-checking is done classically, since each model is assumed to be consistent. Our work is complementary to the above: Huth and Pradhan propose to handle inconsistencies between refinements of the same system, whereas we handle inconsistencies between different descriptions on the same level of abstraction.

Symbolic probabilistic model-checking has been implemented as part of the tool PROB-

VERUS [Baier et al., 1997]. The models used in this work are Kripke structures where edges are labeled with probabilities assigned to the corresponding transitions, and state variables are classically-valued. Thus, the data structures used by PROBVERUS are *Multi-Terminal Binary Decision Diagrams* (MTBDDs) [Baier and Clarke, 1998], which are equivalent to the *Algebraic Decision Diagrams* (ADDs) of Somenzi et al. [Bahar et al., 1993]. Each non-terminal node of MTBDDs has two children, and the number of terminal nodes depends on the range of the function being represented.

9.3 Future Work

We plan to extend the work presented in this paper in a number of directions. First of all, success of multi-valued model-checking depends in part on our ability to engineer the model-checker to handle non-trivial problems. To this effect, we are currently working on optimizations of symbolic representations of mv-sets [Chechik et al., 2001b] and empirically characterizing the tradeoffs between different symbolic representations.

Preliminary work on multi-valued LTL (χLTL) model-checking has been reported in [Chechik et al., 2001a]. We are now interested in extending χChek to handling χLTL properties symbolically. In addition, in the current paper we talked about the different avenues one may want to take in defining "more intuitive" χCTL operations. To this end, we plan to extend our model-checking approach to reasoning about multi-valued μ-calculus properties.

We are also interested in conducting a number of case studies that use the multi-valued model-checking approach described in this paper. The two particular domains we plan to concentrate on are requirements engineering and feature interaction. In particular, we hope to apply and refine our methodology for the use of model-checking for negotiations between stakeholders.

The ability to merge individual system descriptions will have a big impact the effectiveness of the above approach. Preliminary work on merging information has been outlined in [Easterbrook and Chechik, 2001], and a report on a more complete formal treatment is forthcoming [Easterbrook et al., 2001].

Acknowledgments

We would like to thank Wendy MacCaull, Albert Lai, Christopher Thompson-Walsh and Victor Petrovykh for many interesting discussions and for their help implementing the model-checker. We are also indebted to Albert Lai for his help with several aspects of lattice theory presented in this paper. Finally, we are grateful to members of the University of Toronto formal methods reading group for helping us refine ideas presented in this paper and improve the clarity of the presentation. This work was financially supported by NSERC and CITO.

REFERENCES

Anderson, A. and Belnap, N. (1975). *Entailment. Vol. 1.* Princeton University Press.

Atlee, J. and Gannon, J. (1993). "State-Based Model Checking of Event-Driven System Requirements". *IEEE Transactions on Software Engineering*, pages 22–40.

Back, R.-J. and von Wright, J. (1998). *Refinement Calculus: A Systematic Approach.* Springer-Verlag.

Bahar, R., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A., and Somenzi, F. (1993). "Algebraic Decision Diagrams and Their Applications". In *IEEE /ACM International Conference on CAD*, pages 188–191, Santa Clara, California. IEEE Computer Society Press.

Baier, C. and Clarke, E. M. (1998). "The Algebraic Mu-Calculus and MTBDDs". In *Proceedings of the 5th Workshop on Logic, Language, Information and Computation, (WoLLIC'98)*.

Baier, C., Clarke, E. M., Hartonas-Garmhausen, V., Kwiatkowska, M. Z., and Ryan, M. (1997). "Symbolic Model Checking for Probabilistic Processes". In Degano, P., Gorrieri, R., and Marchetti-Spaccamela, A., editors, *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *LNCS*, pages 430–440, Bologna, Italy. Springer-Verlag.

Belnap, N. (1977). "A Useful Four-Valued Logic". In Dunn and Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 30–56. Reidel.

Berney, G. and dos Santos, S. (1985). *Elevator Analysis, Design and Control*. IEE Control Engineering Series 2. Peter Peregrinus Ltd.

Bolc, L. and Borowik, P. (1992). *Many-Valued Logics*. Springer-Verlag.

Bruns, G. and Godefroid, P. (1999). "Model Checking Partial State Spaces with 3-Valued Temporal Logics". In *Proceedings of CAV'99*, volume 1633 of *LNCS*, pages 274–287.

Bruns, G. and Godefroid, P. (2000). "Generalized Model Checking: Reasoning about Partial State Spaces". In *Proceedings of CONCUR'00*, volume 877 of *LNCS*, pages 168–182.

Bultan, T., Gerber, R., and League, C. (2000). "Composite Model Checking: Verification with Type-Specific Symbolic Representations". *ACM Transactions on Software Engineering and Methodology*, 9(1):3–50.

Chechik, M., Devereux, B., and Gurfinkel, A. (2001a). "Model-Checking Infinite State-Space Systems with Fine-Grained Abstractions Using SPIN". In *Proceedings of the 8th SPIN Workshop on Model Checking Software*, volume 2057 of *LNCS*, pages 16–36. Springer.

Chechik, M. and Ding, W. (2001). "Lightweight Reasoning about Program Correctness". In *Proceedings of CASCON*. To appear.

Chechik, M. and Gurfinkel, A. (2001). "Counter-Example Generation for Multi-Valued Model-Checking". In preparation.

Chechik, M., Gurfinkel, A., Devereux, B., Lai, A., and Easterbrook, S. (2001b). "Symbolic Data Structures for Multi-Valued Model-Checking". Submitted for publication.

Clarke, E., Emerson, E., and Sistla, A. (1986). "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications". *ACM Transactions on Programming Languages and Systems*, 8(2):244–263.

Clarke, E., Grumberg, O., Hiraishi, H., Jha, S., Long, D., McMillan, K., and Ness, L. (1993). "Verification of the Futurebus+ Cache Coherence Protocol". In *Proceedings of CHDL*.

Clarke, E., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press.

Cousot, P. and Cousot, R. (1999). "Refining Model Checking by Abstract Interpretation". *Authomated Software Engineering, special issue on Automated Software Analysis*, 6:69–95.

Dams, D. (1996). *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, The Netherlands.

Dams, D., Gerth, R., and Grumberg, O. (1997). "Abstract Interpretation of Reactive Systems". *ACM Transactions on Programming Languages and Systems*, 2(19):253–291.

Dill, D. (1996). "The Murϕ Verification System". In Alur, R. and Henzinger, T., editors, *Computer-Aided Verification Computer*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393, New York, N.Y. Springer-Verlag.

Dunn, J. (1999). "A Comparative Study of Various Model-Theoretic Treatments of Negation: A History of Formal Negation". In Gabbay, D. and Wansing, H., editors, *What is Negation*. Kluwer Academic Publishers.

Easterbrook, S. and Chechik, M. (2001). "A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints". In *Proceedings of International Conference on Software Engineering (ICSE'01)*, pages 411–420.

Easterbrook, S., Chechik, M., and Gurfinkel, A. (2001). "Merging Multi-Valued Models". In preparation.

Fitting, M. (1991). "Many-Valued Modal Logics". *Fundamenta Informaticae*, 15(3-4):335–350.

Fitting, M. (1992). "Many-Valued Modal Logics II". *Fundamenta Informaticae*, 17:55–73.

Gaines, B. R. (1979). "Logical Foundations for Database Systems". *International Journal of Man-Machine Studies*, 11(4):481–500.

Ginsberg, M. (1987). "Multi-valued logic". In Ginsberg, M., editor, *Readings in Nonmonotonic Reasoning*, pages 251–255. Morgan-Kaufmann Pub.

Godefroid, P., Huth, M., and Jagadeesan, R. (2001). "Abstraction-based Model Checking using Modal Transition Systems". In Larsen, K. and Nielsen, M., editors, *Proceedings of 12th International Conference on Concurency Theory (CONCUR 2001)*, volume 2154 of *LNCS*, pages 426–440. Springer.

Goguen, J. (1967). L-fuzzy sets. *Journal of Mathematical Analysis and Applications*, 18(1):145–174.

Hähnle, R. (1994). *Automated Deduction in Multiple-Valued Logics*, volume 10 of *International Series of Monographs on Computer Science*. Oxford University Press.

Hazelhurst, S. (1996). *Compositional Model Checking of Partially Ordered State Spaces*. PhD thesis, Department of Computer Science, University of British Columbia.

Hazelhurst, S. (1999). "Generating and Model Checking a Hierarchy of Abstract Models". Technical Report TR-Wits-CS-1999-0, Department of Computer Science University of the Witwatersrand, Johannesburg, South Africa.

Hehner, E. (1993). *A Practical Theory of Programming*. Texts and Monographs in Computer Science. Springer-Verlag, New York.

Holzmann, G. (1991). *Design and Validation of Computer Protocols*. Prentice Hall.

Holzmann, G. (1997). "The Model Checker SPIN". *IEEE Transactions on Software Engineering*, 23(5):279–295.

Huth, M., Jagadeesan, R., and Schmidt, D. A. (2001). "Modal Transition Systems: A Foundation for Three-Valued Program Analysis". In *Proceedings of 10th European Symposium on Programming (ESOP)*, volume 2028 of *LNCS*, pages 155–169. Springer.

Huth, M. and Pradhan, S. (2001). "Aspect-driven Property Verification of Inconsistent System Descriptions". Submitted for publication.

Huth, M. and Ryan, M. (2000). *Logic in Computer Science: Modeling and Reasoning About Systems*. Cambridge University Press.

Kleene, S. C. (1952). *Introduction to Metamathematics*. New York: Van Nostrand.

Kozen, D. (1983). "Results on the Propositional μ-calculus". *Theoretical Computer Science*, 27:334–354.

Larsen, K. and Thomsen, B. (1988). "A Model Process Logic". In *Third Annual Symposium on Logic in Computer Sciences*, pages 203–210. IEEE Computer Society Press.

Łukasiewicz, J. (1970). *Selected Works*. North-Holland, Amsterdam, Holland.

McMillan, K. (1993). *Symbolic Model Checking*. Kluwer Academic.

Michalski, R. S. (1977). "Variable-Valued Logic and its Applications to Pattern Recognition and Machine Learning". In Rine, D. C., editor, *Computer Science and Multiple-Valued Logic: Theory and Applications*, pages 506–534. North-Holland, Amsterdam.

Plath, M. and Ryan, M. (1999). "SFI: A Feature Integration Tool". In Berghammer, R. and Lakhnech, Y., editors, *Tool Support for System Specification, Development and Verification*, Advances in Computer Science, pages 201–216. Springer.

Rasiowa, H. (1978). *An Algebraic Approach to Non-Classical Logics. Studies in Logic and the Foundations of Mathematics*. Amsterdam: North-Holland.

Sagiv, M., Reps, T., and Wilhelm, R. (1999). "Parametric Shape Analysis via 3-Valued Logic". In *Proceedings of 26th Annual ACM Symposium on Principles of Programming Languages*.

Sasao, T. and Butler, J. (1996). "A Method to Represent Multiple-Output Switching Functions Using Multi-Valued Decision Diagrams". In *Proceedings of IEEE International Symposium on Multiple-Valued Logic*, pages 248–254, Santiago de Compostela, Spain.

Sofronie-Stokkermans, V. (2000). Automated theorem proving by resolution for finitely-valued logics based on distributive lattices with operators. *Multiple-Valued Logic*.

Srinivasan, A., Kam, T., Malik, S., and Brayton, R. (1990). "Algorithms for Discrete Function Manipulation". In *IEEE International Conference on Computer-Aided Design*, pages 92–95.

Visser, W., Park, S., and Penix, J. (2000). "Applying Predicate Abstraction to Model Check Object-Oriented Programs". In *Proceedings of 4th International Workshop on Formal Methods in Software Practice*.

Zadeh, L. (1987). "Fuzzy Sets". In Yager, R. R., Ovchinnikov, S., Tong, R. M., and Nguyen, H. T., editors, *Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh*, pages 29–44, New York. John Wiley & Sons, Inc.