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 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., 2002] 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 show that χCTL preserves the properties we expect it to have, 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 and the general theory of multi-valued model checking, and describes a multi-valued symbolic model-checker χChek. χChek [Chechik et al., 2002a] is a generalization of an existing symbolic model-checking algorithm [McMillan, 1993] for a multi-valued extension of the temporal logic CTL.

Classical model-checking takes a model, M, of a system (expressed as a finite state machine), and a temporal correctness property, φ, (expressed as a formula in a suitable temporal logic), and determines whether or not the model satisfies the property [Clarke et al., 1986], i.e., it returns the value of the predicate $M \models \varphi$. Model-checking has been effectively applied to reasoning about correctness of hardware [Clarke et al., 1993], com-

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

munication 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 2.

Multi-valued model-checking, i.e., reasoning with values other than TRUE and FALSE, is a generalization of classical model-checking. The following can be generalized:

- Variables in the finite state machine can be multi-valued or boolean.
- Transitions between states in the finite state machine can be multi-valued or boolean.
- The satisfaction relation can be multi-valued or boolean.

For example, in probabilistic modeling, transitions are labeled with values representing the probability that they are taken. The satisfaction relation then becomes the probability that a desired property holds [Kwiatkowska et al., 2000].

The advantages of automated reasoning with multiple values have been recognized by a number of researchers. For example, 3-valued logic has been used for interpreting results of static analysis with abstraction [Chechik and Ding, 2001; Sagiv et al., 1999], and for analyzing partial models [Bruns and Godefroid, 1999; Bruns and Godefroid, 2000]. In the latter case, intermediate value of the logic is used to annotate partial information, and checking the models allows the analyst to determine whether the desired properties are preserved in all refinements of this system. 4-valued logic has been used for reasoning about abstractions of detailed gate or switch-level designs of circuits [Hazelhurst, 1996; Hazelhurst, 1999] and for reasoning about systems containing disagreements [Easterbrook and Chechik, 2001] which get introduced when we compose two models drawn from different sources. By checking a model in which disagreements are explicitly represented, the analyst can reason about how such disagreements affect various temporal properties and thus support potential negotiation.

We are interested in generalizing these approaches, by defining model-checking algorithms over a large class of logics. In particular, we pose the following requirements to this class: (a) many of the desired properties of classical logic, e.g., associativity, commutativity, idempotance, etc. are preserved; (b) the logics can be used for representing a large class of systems; (c) model-checking using these logics remains tractable. We intentionally leave probabilistic systems outside the scope of this paper, concentrating instead on finite logics.

We choose finite distributive lattices to be the domain of logical values used in multi-valued model-checking. These lattices ensure commutativity, idempotance, and a variety of other classical properties. If, in addition, we can define a negation operator that preserves De Morgan laws and involution ($\neg\neg a = a$), such lattices are called *quasi-boolean*, and the resulting structures are called *quasi-boolean algebras* [Rasiowa, 1978]. Classical boolean logic as well as 3- and 4-valued logics described in the literature are some examples of quasi-boolean algebras. In what follows, we often use terms "algebras" and "logics" interchangeably, to indicate a set of elements closed under logical operations. We define quasi-boolean algebras formally and discuss their properties in Section 3.

In addition to identifying a useful class of well-behaved algebras, this paper makes several other contributions to the study of multi-valued model-checking. First, we apply a theory of multi-valued sets and relations to create the core structure for our symbolic model-checking algorithm. Multi-valued sets are sets whose membership functions are multi-valued. We use multi-valued sets 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 set 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 4.

Second, we define a multi-valued semantics for CTL, and demonstrate that this semantics preserves the desired 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 given multi-valued logic. We present χCTL and χKripke structures in Section 5. We also show that χCTL is decidable and analyze fixpoint properties of its operators.

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, i.e., occur under reasonable assumptions on process scheduling. 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 6.

We describe the implementation details and some potential applications of multi-valued model-checking in Section 7. We further note that the multi-valued model-checking decision procedure can be either implemented directly or reduced to classical [Konikowska and Penczek, 2002]. The tradeoffs between these choices are studied in the companion paper [Chechik et al., 2002b].

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 8).

2. 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 set 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

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

Fig. 1. Fixpoint formulations of CTL operators.

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,

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

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 1. This description is most useful for symbolic model-checking [McMillan, 1993].

Throughout the rest of the paper we 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 [Kozen, 1983]; (4) we use nat to refer to the set of natural numbers, and (5) we use $\exists!$ to mean "exists unique".

3. DEFINING THE LOGIC

As mentioned in the introduction to this paper, we are interested in choosing a class of logics that are natural for describing realistic problems, have many properties of classical logic, and still enable tractable model-checking algorithms. We found that quasi-boolean algebras satisfy the above criteria and define them below. We start with some lattice theory background in Section 3.1 which can be skipped. The definition of quasi-boolean algebras is given in Section 3.2.

3.1 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 $a \sqcup b$. Meet and join of an empty set are defined to be *top* and *bottom* of the lattice, respectively:

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

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

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

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

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

Lattices have a number of additional properties, some of which are given below. Note that this set of properties is redundant.

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

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

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

Figure 2 gives examples of a few lattices. The lattice in Figure 2(h) is non-distributive, whereas all other lattices are distributive.

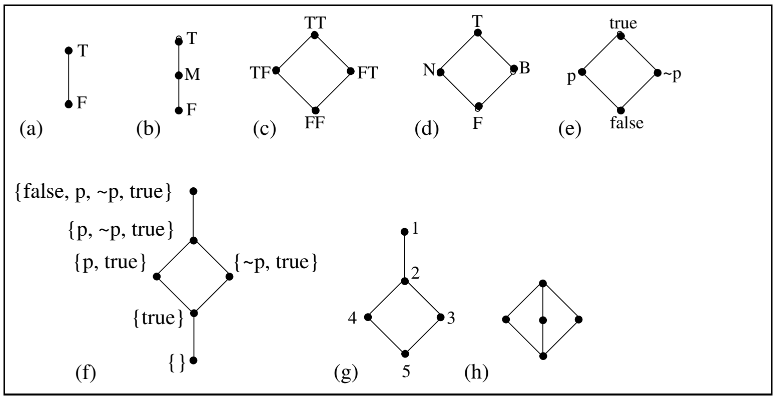

Fig. 2. Examples of a few lattices.

3.2 Quasi-Boolean Algebras

In this section we define a class of quasi-boolean algebras and study their properties.

DEFINITION 3. *[Rasiowa, 1978] A quasi-boolean algebra is a tuple* $(\mathcal{L}, \sqcap, \sqcup, \neg)$*, where:*

- *(*$\mathcal{L}$*,* $\sqsubseteq$*) is a finite distributive lattice;*
- *Conjunction (*$\sqcap$*) and disjunction (*$\sqcup$*) are meet and join operators of (*$\mathcal{L}$*,* $\sqsubseteq$*), respectively;*
- *Negation* $\neg$ *is a function* $\mathcal{L} \rightarrow \mathcal{L}$ *s.t. every element* $a \in \mathcal{L}$ *corresponds to a unique element* $\neg a \in \mathcal{L}$ *satisfying the following conditions:*

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

where $b \in \mathcal{L}$*.* $\neg a$ *is called* a quasi-complement *of* a*.*

We do not explicitly include $\sqsubseteq$ in our definition of quasi-boolean algebras. It is present implicitly as an ordering over the elements of $\mathcal{L}$.

Quasi-boolean algebras, also known as De Morgan algebras, are a familiar concept in logic [Bolc and Borowik, 1992; Dunn, 1999]. Note that the negation operator satisfying the above properties is a *lattice dual isomorphism* with period 2 [Birkhoff, 1967].

We now define several quasi-boolean algebras using the lattices in Figure 2. The domain of logical values of the classical logic, referred to as $\mathbf{2}$, is the lattice in Figure 2(a). Note that in this case, $\sqcup$ and $\sqcap$ are conventionally referred to as $\vee$ and $\wedge$, respectively. We also use these notations interchangeably when the interpretation is clear from the context. The three-valued logic $\mathbf{3}$ is defined on the lattice in Figure 2(b), where ¬T = F, ¬F = T, ¬M = M. This logic was first defined in [Kleene, 1952]. The four-valued logic $\mathbf{2x2}$ is defined on the lattice in Figure 2(c), where ¬ TF = FT and ¬ FT = TF. This logic can be used for reasoning about inconsistency. Note that in logics $\mathbf{2}$, $\mathbf{3}$ and $\mathbf{2x2}$, the partial order operation of the underlying lattice is interpreted as logical implication, e.g. $a \sqsubseteq b$ means "b is more true than a". Further, $\top$ and $\bot$ of the lattice are interpreted as values TRUE and FALSE of the logic, respectively.

A quasi-boolean algebra defined over the lattice in Figure 2(d) is the Belnap's 4-valued logic which has been used for reasoning about inconsistent databases [Belnap, 1977; Anderson and Belnap, 1975]. Here, ¬N = N and ¬B = B. The underlying lattice is isomorphic to the one in Figure 2(c), but the resulting quasi-boolean algebras are not. The lattice in Figure 2(e) represents the set of propositional formulas over $\{p\}$, referred to as $PF(\{p\})$. The logic defined over this lattice is isomorphic to $\mathbf{2x2}$, even though the interpretation of the individual values is different. The lattice in Figure 2(f) represents the upsets[1] of $PF(\{p\})$, ordered by set inclusion. Such lattices are called *upset lattices*, and quasi-boolean algebras defined on them are used for query-checking [Gurfinkel et al., 2002; Bruns and Godefroid, 2001].

The lattice in Figure 2(h) cannot be used as a basis for a quasi-boolean algebra because it is non-distributive. The lattice in Figure 2(g) cannot be used either because no suitable quasi-complement can be found for element 2.

DEFINITION 4. *A tuple* $L = (\mathcal{L}, \sqcap, \sqcup, \neg)$ *is a* finite Boolean algebra *if* L *is a quasi-boolean algebra and additionally, for every element* $a \in \mathcal{L}$*,*

$$\begin{array}{rcll} a \sqcap \neg a & = & \bot & \neg \text{ contradiction} \\ a \sqcup \neg a & = & \top & \neg \text{ exhaustiveness} \end{array}$$

For example, the algebra $\mathbf{2}$ is boolean, whereas $\mathbf{3}$ is not (M ⊓ ¬M $\neq \bot$).

DEFINITION 5. *A* product *of two logics* $L_1 = (\mathcal{L}_1, \sqcap_1, \sqcup_1, \neg_1)$ *and* $L_2 = (\mathcal{L}_2, \sqcap_2, \sqcup_2, \neg_2)$ *is a logic* $L_1 \times L_2 = (\mathcal{L}_1 \times \mathcal{L}_2, \sqcap, \sqcup, \neg)$ *where*

$$\begin{array}{rcll} \neg(a,b) & = & (\neg_1 a, \neg_2 b) & \neg \text{ of pairs} \\ (a,b) \sqcap (a',b') & = & (a \sqcap_1 a', b \sqcap_2 b') & \sqcap \text{ of pairs} \\ (a,b) \sqcup (a',b') & = & (a \sqcup_1 a', b \sqcup_2 b') & \sqcup \text{ of pairs} \end{array}$$

Thus, the operations on the product logic are the component-wise extensions of their individual counterparts. Similar properties hold for $\top$, $\bot$, and the ordering:

$$\begin{array}{rcll} \bot_{\mathcal{L}_1 \times \mathcal{L}_2} & = & (\bot_{\mathcal{L}_1}, \bot_{\mathcal{L}_2}) & \bot \text{ of pairs} \\ \top_{\mathcal{L}_1 \times \mathcal{L}_2} & = & (\top_{\mathcal{L}_1}, \top_{\mathcal{L}_2)} & \top \text{ of pairs} \\ (a,b) \sqsubseteq (a',b') & \Leftrightarrow & a \sqsubseteq_1 a' \ \wedge \ b \sqsubseteq_2 b' & \sqsubseteq \text{ of pairs} \end{array}$$

We are ready to state a few theorems about quasi-boolean algebras. Proofs of several theorems of this paper are given in the Appendix.

THEOREM 1. *A product of two quasi-boolean algebras is quasi-boolean, that is,*

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

Thus, a product of two algebras preserves their distributivity, finiteness and (quasi-)boolean properties. Consider the algebra $\mathbf{2x2}$ defined as a product of two $\mathbf{2}$ algebras, over the lattice in Figure 2(c). This algebra is boolean because $\mathbf{2}$ is boolean. The algebra $\mathbf{3}$ is quasi-boolean, and all its products, e.g., $\mathbf{3x3}$, are quasi-boolean.

[1] Given the ordered set $(\mathcal{L}, \sqsubseteq)$ and a subset $B \subseteq L$, $\uparrow B$ is the set $\{\ell \in \mathcal{L} \mid \exists b \in B \cdot b \sqsubseteq \mathcal{L}\}$. A subset B of $\mathcal{L}$ is an *upset* if $\uparrow B = B$.

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

DEFINITION 6. *A lattice* ($\mathcal{L}$, $\sqsubseteq$) *is* horizontally-symmetric *iff there exists a bijective function* H *such that for every pair* $a, b \in \mathcal{L}$,

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

Notice that H is a lattice dual automorphism with period 2. Thus, horizontal symmetry is a necessary and sufficient condition for defining a quasi-boolean algebra over a distributive lattice, with a potential negation defined as $\neg a = H(a)$ for each element of the lattice. Lattices in Figure 2(a)-(f) exhibit horizontal symmetry and thus are quasi-boolean, whereas the lattice in Figure 2(g) is not. In the rest of the paper we assume that the negation operator on our algebras is defined via horizontal symmetry, unless explicitly specified otherwise. Note that in Belnap's 4-valued logic, defined on top of the lattice in Figure 2(d), negation is not defined via horizontal symmetry: ¬N = N, ¬B = B.

Finally, we define implication and equivalence as follows:

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

4. 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 which a property holds. Such operations include union, intersection, complement, and backward image for computing predecessors. Given a quasi-boolean algebra, 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 algebras in this section. This treatment is similar to the definition of L-fuzzy sets [Goguen, 1967].

4.1 Multi-Valued Sets

In classical set theory, a set is defined by a boolean predicate, also called a *membership* or a *characteristic* function. Typically, it is written using a *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 elements of a given algebra, 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*.

DEFINITION 7. *Given an algebra* $L = (\mathcal{L}, \sqcap, \sqcup, \neg)$, *and a classical set* S, *a multi-valued subset of* S, *referred to as* $\mathbb{S}$, *is a total function* $S \rightarrow \mathcal{L}$.

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.

We illustrate mv-sets using a simple state machine shown in Figure 3. This machine uses the quasi-boolean algebra $\mathbf{2x2}$ where the logical values form the lattice in Figure 2(c),

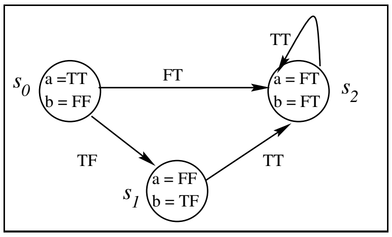

Fig. 3. Ex1: a simple χKripke structure.

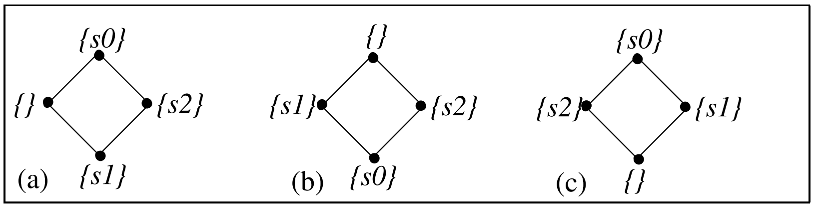

Fig. 4. Several mv-sets for the example in Figure 3: (a) corresponding to variable a; (b) corresponding to variable b; (c) $\overline{[\![b]\!]}$ – a multi-valued complement of the mv-set in (b).

and exemplifies χKripke structures — multi-valued generalizations of Kripke structures, defined formally in Section 5.1. In classical symbolic model-checking, expressions such as "x" are used to indicate the set of states where x is TRUE. Similarly, we use multi-valued expressions to partition the state space of the system. For example, a variable a represents different states of the χKripke structure in Figure 3: each value ℓ of $\mathbf{2} \times \mathbf{2}$ is associated with the set of states where a has value ℓ. In particular, a has value TT in $\{s_0\}$, FT in $\{s_2\}$, FF in $\{s_1\}$ and TF in $\{\}$. This mv-set, referred to as $[\![a]\!]$, can be graphically represented as shown in Figure 4(a), where the structure corresponds to that of the underlying algebra.

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 $[\![a]\!]$ and $[\![b]\!]$ given in Figure 4(a) and (b), respectively, we note that in state s_1 a is FF and b is TF. Thus,

$$([\![a]\!] \cap_L [\![b]\!])(s_1) = \text{FF} \sqcap \text{TF} = \text{FF}$$

We 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 $\overline{[\![b]\!]}$ is given in Figure 4(c).

We then obtain the desired 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 2. *For a multi-valued set $\mathbb{S}$ defined* **2***, 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 elements of the underlying algebra range in the interval $[0, 1]$, ordered by the usual $\leq$ relation on real numbers, then we obtain fuzzy set theory [Zadeh, 1987].

4.2 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 $S \times T$.*

Let S be the set of states of the χKripke structure in Figure 3, referred to as Ex1. The multi-valued relation over $S \times S$ represents values of transitions between pairs of states of Ex1 and is shown in Figure 5. This relation is referred to as $\mathbb{T}$. For example, the value of the transition (s_0, s_1) is TF, so $\mathbb{T}((s_0, s_1)) =$ TF.

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.

We now consider computing the forward and the backward images of $[\![a]\!]$ (see Figure 4(a)) under the multi-valued relation $\mathbb{T}$ between the pairs of states of the χKripke structure Ex1. These are shown in Figures 5(b) and (c), respectively. For example, when

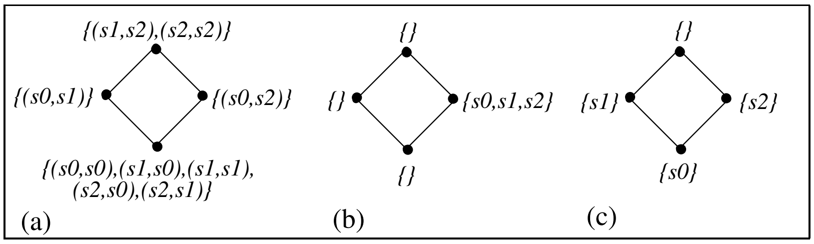

Fig. 5. (a) The multi-valued relation between pairs of states of Ex1; (b) Forward image of $[\![a]\!]$ over the relation in (a); (c) Backward image of $[\![a]\!]$ over the relation in (a).

we compute backward image of s_0, we get

$$\bigsqcup_{t\in S}([\![a]\!](t) \sqcap \mathbb{T}(s_0, t)) = (\text{TT} \sqcap \text{FF}) \sqcup (\text{FF} \sqcap \text{TF}) \sqcup (\text{FT} \sqcap \text{FT}) = \text{FT}$$

which indicates that there exists an FT transition from state s_0 to another state (actually, s_2), where a is FT.

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:

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

This gives rise to the following theorem:

THEOREM 3. *Let L be **2**. 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,*

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

Proof:

$$\begin{array}{rlr} & \overrightarrow{\mathbb{R}}(\mathbb{Q}) & \text{Definition of } \overrightarrow{\mathbb{R}} \\ = & \lambda t \cdot \bigsqcup_{s\in S}(\mathbb{Q}(s) \sqcap \mathbb{R}(s,t)) & \mathbb{R} \text{ is a boolean relation} \\ = & \lambda t \cdot \bigsqcup_{\{s\in S \mid \mathbb{R}(s,t)\}} \mathbb{Q}(s) & \text{Theorem 2} \\ = & \lambda t \cdot \bigvee_{\{s\in S \mid \mathbb{R}(s,t)\}} \mathbb{Q}(s) & \text{Definition of } \overrightarrow{\mathbb{R}}^B \\ = & \overrightarrow{\mathbb{R}}^B(\mathbb{Q}) & \end{array}$$

The proof of (2) is similar. □

5. MULTI-VALUED CTL MODEL-CHECKING

In this section we extend the notion of boolean model-checking described in Section 2 by defining multi-valued Kripke structures, which we call χKripke structures, and multi-valued CTL (χCTL).

5.1 Semantics

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

- $L = (\mathcal{L}, \sqcap, \sqcup, \neg)$ is a quasi-boolean algebra. This is the algebra 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 algebra 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. One χKripke structure, shown in Figure 3, was introduced in Section 4.

5.2 Multi-Valued CTL

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

In extending the CTL operators, we want to ensure that the desired properties of EX, EG and EU, which form the adequate set for CTL, are still preserved. Definitions of other operators are given in Section 2.

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.

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 4.

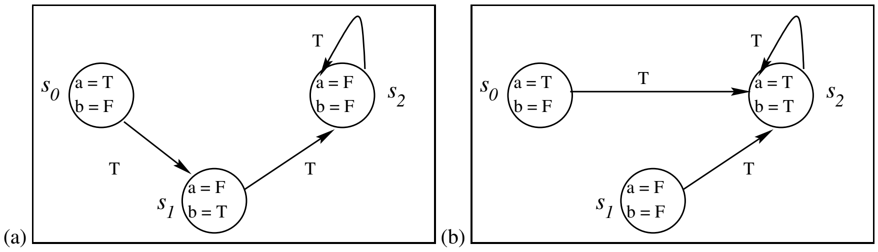

Fig. 6. Two classical Kripke structures: (a) Ex_l; (b) Ex_r.

The semantics is as follows:

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

We proceed by defining the EX operator. Recall from Section 2 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:

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

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

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

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 the two classical Kripke structures, Ex_l and Ex_r, shown in Figure 6. χKripke structure Ex1, shown in Figure 3, constitutes one possible merge of Ex_l and Ex_r. In this case, states with the same name are merged. For example, a variable b has values T and F in state s_1 of Ex_l and Ex_r, respectively; therefore, in Ex1, this variable has value TF. Similarly, a transition (s_0, s_1) is present in Ex_l and absent in Ex_r; therefore, it has value TF in Ex1. Consider evaluating a property EXb in state s_0 of these three models. This property is T in Ex_l, because b is T in s_1, and T in Ex_r, because b is T in s_2. In Ex1, this property evaluates to TF on path (s_0, s_1) and to FT on path (s_0, s_2). Their disjunction, and therefore the value of EXb in state s_0, is TT.

AX is then defined, following the AX duality in Section 2, as

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

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 counterparts. 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 1:

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

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 counterparts:

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

Note that our definition of EU also preserves the familiar property of its CTL counterpart:

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

5.3 Properties of χCTL

We begin with some sanity checks on χCTL.

THEOREM 4. *χCTL reduces to CTL when the algebra is* **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 2.

Proof:
(1) holds by Theorem 3. 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).

□

We now formulate two theorems the proofs of which are given in the Appendix.

THEOREM 5. *χCTL operators AX and EX are monotone.*

THEOREM 6. *The definition of χCTL ensures that for each φ, $[\![\varphi]\!]$ forms a partition.*

Thus, χCTL is well defined, i.e., each property φ has exactly one value in each state of the system.

THEOREM 7. *Multi-valued model-checking is decidable.*

Proof:
From Theorem 5 and monotonicity of $\cap_L$ and $\cup_L$, we conclude that all χCTL operators involve computation of fixpoints over monotone functions, thus resulting in a natural algorithm. The termination of this algorithm is guaranteed by the usual application of the Knaster-Tarski theorem. □

THEOREM 8. *Fixpoint properties of (derived) χCTL operators are the same as for CTL operators. That is,*

$$\begin{array}{llcll} (1) & [\![AG\varphi]\!] & = & \nu\mathbb{Z}.[\![\varphi]\!] \cap_L [\![AX\mathbb{Z}]\!] & AG \text{ fixpoint} \\ (2) & [\![AF\varphi]\!] & = & \mu\mathbb{Z}.[\![\varphi]\!] \cup_L [\![AX\mathbb{Z}]\!] & AF \text{ fixpoint} \\ (3) & [\![EF\varphi]\!] & = & \mu\mathbb{Z}.[\![\varphi]\!] \cup_L [\![EX\mathbb{Z}]\!] & EF \text{ fixpoint} \\ (4) & [\![A[\varphi\ U\ \psi]]\!] & = & \mu\mathbb{Z}.[\![\psi]\!] \cup_L ([\![\varphi]\!] \cap_L [\![AX\mathbb{Z}]\!]) & AU \text{ fixpoint} \end{array}$$

Note that our ability to prove the above properties of χCTL operators was dependent on the fact that our model-checking is defined over quasi-boolean algebras. For a treatment of properties of χCTL when the model-checking is defined over a more general class of algebras, please see [Devereux, 2002; Chechik and MacCaull, 2002].

5.4 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 algebra. 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 increases 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 algebra, 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 7 and in the companion paper [Chechik et al., 2002b].

6. FAIRNESS

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

6.1 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 9. *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) $\Rightarrow$ (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) $\Rightarrow$ (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. □

6.2 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. 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 algebra $L = (\mathcal{L}, \sqcap, \sqcup, \neg)$ as the model, even though for each fairness condition c_i, $\forall s \in S \cdot [\![c_i]\!](s) \in \{\top, \bot\}$.

We begin defining multi-valued fairness by introducing the concept of *mv-trace*. In classical model-checking, a *trace* from s is a single computation, emanating from s, of the corresponding Kripke structure. On the other hand, a trace in multi-valued model-checking may correspond to several computations. For example, consider the χKripke structures Ex$_l$, Ex$_r$ and Ex1, shown in Figures 6 and 3, respectively. As indicated earlier, Ex1 is the merge of the other two models. Next-state computations from s_0 in models Ex$_l$ and Ex$_r$ are s_0, s_1 and s_0, s_2, respectively. Yet, as shown in Section 5.2, both are necessary to evaluate EXb. This leads us to a conclusion that χCTL operators, and their fair counterparts, are defined on *sets of computations*, which we refer to as *mv-traces*.

DEFINITION 12. *An mv-trace in a χKripke 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 mv-trace.

6.3 Fair EG

As in CTL, χCTL operators $E_C G$, $E_C[\varphi\ U\ \psi]$ and $E_C X$ form an adequate set. Given a formulation for $E_C G$, we can define other fair χCTL operators, as shown in Section 6.4.

Thus, we start with noting that $[\![E_C G\varphi]\!](s) = \ell$ means that there exists an mv-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:

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

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, calling the new operator $E_C G'$.

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

We are now ready to study properties of $E_C G$ and $E_C G'$. We begin by showing that $E_C G'$ becomes EG when there are no fairness conditions present, and then proceed to show that the operators $E_C G$ and $E_C G'$ are equivalent. Proofs of both theorems are given in the Appendix.

THEOREM 10. *When $C = \{\top\}$ (no fairness), $E_C G'$ becomes*

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

THEOREM 11. *Operators $E_C G$ and $E_C G'$ are equivalent.*

6.4 Fairness in Other χCTL Operators

Computing $E_C X\varphi$ in s amounts to 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

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

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

6.5 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.

7. IMPLEMENTATION AND APPLICATIONS

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

7.1 Implementation

As a proof of concept, we have developed a prototype implementation (in Java) of a multi-valued model-checker called χChek [Chechik et al., 2002a]. 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 algebra. For each χCTL formula, χChek calculates its 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, 2002].

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]. Of course, the multi-valued model-checking problem also reduces to several queries to the classical model-checker, run on top of Binary Decision Diagrams (BDDs) [Konikowska and Penczek, 2002]. The tradeoffs between these encodings depend on a number of factors, including the types of questions asked, the size and the shape of the elements of the underlying lattice, 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., 2002b].

7.2 Applications

Multi-valued model-checking has a number of potential applications in software engineering, in modeling of uncertainty, disagreement, relative priority, and general model exploration. 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 quasi-boolean algebras defined over product lattices. 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.
- Elements of our quasi-boolean algebras do not have to be interpreted as logical values. Consider the *query-checking* problem [Chan, 2000], the inputs to which are a (classical) model and a temporal logic query (TLQ). A TLQ is a temporal logic formula with placeholders, e.g., $AG?$. Query-checking results in finding the strongest set of assignments of propositional formulas to placeholders, such that replacing the placeholders with any of these assignments results in a temporal logic formula that holds in the model. Thus, query-checking is a form of model exploration. This problem can be formulated as a multi-valued model checking problem on *upset* lattices, where the elements are sets of propositional formulas ordered by set inclusion. One example of upset lattices was given in Figure 2(f). The reduction of the query-checking problem to multi-valued model-checking problem is described in [Gurfinkel et al., 2002].

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

(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. 7. 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`.

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 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 identified for abstraction into the same abstract state disagree on a value of a variable, we cannot assign either TRUE or FALSE to this variable 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 $\top$ ($\perp$) in an abstract state, it evaluates to $\top$ ($\perp$) 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 by 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 $\texttt{liftBut}_i$).

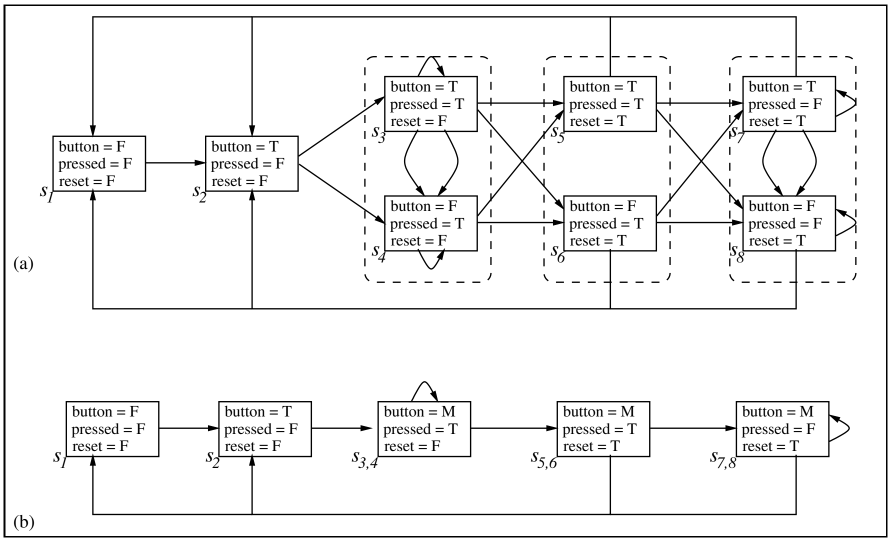

Fig. 8. State machines: (a) of the modified module `Button`; (b) of the abstracted module `Button`..

The SMV module `Button` is shown in Figure 7(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 7(b), and its state machine – in Figure 8(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 8(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 7(c). The state machine model of the abstract system is shown in Figure 8(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) reduction to classical model-checking, proposed by Bruns & Godefroid, and (b) directly, using χChek.

The first technique consists of two queries to a classical model-checker and is applicable to formulas where negation is applied only to atomic propositions. The first step is the computation of 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 building 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

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. 9. Properties of the elevator system.

$\top$, return $\top$; else if the optimistic model yields $\bot$, return $\bot$; otherwise, return M.

The properties of the elevator system that we verified are given in Figure 9. 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 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 classical 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 I 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 I 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 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.

Also, 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 im-

Model	CTL Property Number	Result	Bruns & Godefroid			χChek
			Pessimistic	Optimistic	Total	
3-floor	1.	$\perp$	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.	$\perp$	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 I. Elevator abstraction: comparison between two runs of the classical model-checker and a run of χChek..

proved: currently we represent all variables as if they can range over all values of the 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.

Finally, note that this example is fairly small. However, we expect the relationship between the performance of the two approaches to stay the same for models of any size.

8. CONCLUSION

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

8.1 Summary

Multi-valued algebras can be useful in a variety of verification tasks. For example, they can help reason about partial systems, solve feature interaction problems, and support general model exploration.

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

8.2 Related Work

Multi-valued algebras (often called "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 algebras 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 algebras. 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 algebras 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 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 logics. 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 a 3-valued logic for expressing properties of partial models. Model-checking of positive properties (properties that do not contain negation) in this algebra 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 a *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 a 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 algebras 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 algebras is possible, such as with χChek, we can refine the algebra when model-checking returns M, e.g., keeping track of exact sources of all disagreements, and thus allow the users to determine which inconsistencies matter and help focus potential negotiations.

Work of Huth and Pradhan [Huth and Pradhan, 2001] helps 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 multi-valued models encode inconsistencies between the different descriptions on the same level of abstraction.

Symbolic probabilistic model-checking has been implemented as part of the tool PROBVERUS [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.

8.3 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., 2002b] 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., 2001]. We are now interested in extending χChek to handling χLTL properties symbolically. We are also interested in conducting a number of case studies that use the multi-valued model-checking approach described in this paper.

Acknowledgments

We would like 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 and the anonymous referees for helping us refine the 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.

Birkhoff, G. (1967). *Lattice Theory.* Americal Mathematical Society, Providence, RI, 3 edition.

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.

Bruns, G. and Godefroid, P. (2001). "Temporal Logic Query-Checking". In *Proceedings of 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 409–417.

Chan, W. (2000). "Temporal-Logic Queries". In *Proceedings of the 12th Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 450–463. Springer-Verlag.

Chechik, M., Devereux, B., and Gurfinkel, A. (2001). "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., Devereux, B., and Gurfinkel, A. (2002a). "χChek: A Multi-Valued Model-Checker". In *Proceedings of International Conference on Computer-Aided Verification– CAV'02*, pages 505–509.

Chechik, M. and Ding, W. (2001). "Lightweight Reasoning about Program Correctness". In *Proceedings of CASCON*, pages 127–140.

Chechik, M. and Gurfinkel, A. (2002). "Counter-Example Generation for Multi-Valued Model-Checking". In preparation.

Chechik, M., Gurfinkel, A., Devereux, B., Lai, A., and Easterbrook, S. (2002b). "Symbolic Data Structures for Multi-Valued Model-Checking". CSRG Tech Report 446, University of Toronto.

Chechik, M. and MacCaull, W. (2002). "CTL Model-Checking Over Logics with Non-Classical Negations". CSRG Tech Report 449, University of Toronto.

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.

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.

Devereux, B. (2002). "Strong Next-time Operators for Multiple-Valued μ-calculus". In *Proceedings of FLOC'02 Workshop on Fixpoints in Computer Science (FICS*.

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.

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.

Gurfinkel, A., Devereux, B., and Chechik, M. (2002). "Model Exploration with Temporal Logic Query Checking". In *Proceedings of SIGSOFT Conference on Foundations of Software Engineering (FSE'02)*. (to appear).

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.

Konikowska, B. and Penczek, W. (2002). "Reducing Model Checking from Multi-Valued CTL* to CTL*". In *Proceedings of CONCUR'02*.

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

Kwiatkowska, M., Norman, G., Parker, D., and Segala, R. (2000). "Symbolic Model Checking of Probabilistic Processes Using MTBDDs and the Kronecker Representation". In *Proceedings of TACAS 2000*, number 1587 in Lecture Notes in Computer Science. Springer-Verlag.

Larsen, K. and Thomsen, B. (1988). "A Modal 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.

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.

A. PROOFS

We give proofs of selected theorems here.

THEOREM 1. *A product of two quasi-boolean algebras is quasi-boolean, that is,*

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

Proof:

$$\begin{array}{lcr}
(1) & \neg\neg(a,b) & \neg \text{ of pairs} \\
\Leftrightarrow & \neg(\neg a,\neg b) & \neg \text{ of pairs} \\
\Leftrightarrow & (\neg\neg a,\neg\neg b) & \neg \text{ involution} \\
\Leftrightarrow & (a,b) & \\
\\
(2) & \neg((a_1,b_1)\sqcap(a_2,b_2)) & \sqcap \text{ of pairs} \\
\Leftrightarrow & \neg((a_1\sqcap a_2),(b_1\sqcap b_2)) & \neg \text{ of pairs} \\
\Leftrightarrow & (\neg(a_1\sqcap a_2),\neg(b_1\sqcap b_2)) & \text{De Morgan} \\
\Leftrightarrow & (\neg a_1\sqcup\neg a_2,\neg b_1\sqcup\neg b_2) & \sqcup \text{ of pairs} \\
\Leftrightarrow & (\neg a_1,\neg b_1)\sqcup(\neg a_2,\neg b_2) & \\
\\
(4) & (a_1,b_1)\sqsubseteq(a_2,b_2) & \sqsubseteq \text{ of pairs} \\
\Leftrightarrow & a_1\sqsubseteq a_2 \ \wedge\ b_1\sqsubseteq b_2 & \neg \text{ antimonotonic} \\
\Leftrightarrow & \neg a_1\sqsupseteq\neg a_2 \ \wedge\ \neg b_1\sqsupseteq\neg b_2 & \sqsupseteq \text{ of pairs} \\
\Leftrightarrow & (\neg a_1,\neg b_1)\sqsupseteq(\neg a_2,\neg b_2) & \neg \text{ of pairs} \\
\Leftrightarrow & \neg(a_1,b_1)\sqsupseteq\neg(a_2,b_2) &
\end{array}$$

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

THEOREM 5. *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:

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

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

THEOREM 6. *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.

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

□

THEOREM 8. *Fixpoint properties of (derived) χCTL operators are the same as for CTL operators. That is,*

$$\begin{array}{lrcll}
(1) & [\![AG\varphi]\!] & = & \nu\mathbb{Z}.[\![\varphi]\!] \cap_L [\![AX\mathbb{Z}]\!] & AG \text{ fixpoint} \\
(2) & [\![AF\varphi]\!] & = & \mu\mathbb{Z}.[\![\varphi]\!] \cup_L [\![AX\mathbb{Z}]\!] & AF \text{ fixpoint} \\
(3) & [\![EF\varphi]\!] & = & \mu\mathbb{Z}.[\![\varphi]\!] \cup_L [\![EX\mathbb{Z}]\!] & EF \text{ fixpoint} \\
(4) & [\![A[\varphi\ U\ \psi]]\!] & = & \mu\mathbb{Z}.[\![\psi]\!] \cup_L ([\![\varphi]\!] \cap_L [\![AX\mathbb{Z}]\!]) & AU \text{ fixpoint}
\end{array}$$

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]

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

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

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

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

We now show (4). We start by reformulating the definition of AU:

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

$$
\begin{aligned}
& (\nu\mathbb{Z}.[\![\varphi \vee \psi]\!] \cap_L ([\![\psi]\!] \cup_L [\![AX\mathbb{Z}]\!])) \cap_L (\mu\mathbb{Z}.[\![\psi]\!] \cup_L [\![AX\mathbb{Z}]\!]) \\
= \; & \text{by distributivity and absorption} \\
& (\nu\mathbb{Z}.[\![\psi]\!] \cup_L (([\![\varphi]\!] \cup_L [\![\psi]\!]) \cap_L [\![AX\mathbb{Z}]\!])) \cap_L (\mu\mathbb{Z}.[\![\psi]\!] \cup_L [\![AX\mathbb{Z}]\!]) \\
= \; & \text{by distributivity and absorption} \\
& (\nu\mathbb{Z}.[\![\psi]\!] \cup_L ([\![\varphi]\!] \cap_L [\![AX\mathbb{Z}]\!])) \cap_L (\mu\mathbb{Z}.[\![\psi]\!] \cup_L [\![AX\mathbb{Z}]\!])
\end{aligned}
$$

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

Recall that if a function F is monotone and continuous, then there exists $n \in nat$ s.t. $\mu\mathbb{Z}.F(\mathbb{Z}) = F^n([\![\bot]\!])$[2] and $\nu\mathbb{Z}.F(\mathbb{Z}) = F^n([\![\top]\!])$. We also name ith iterations of F and G: $\mathbb{K}_i = F^i([\![\top]\!])$, $\mathbb{W}_i = F^i([\![\bot]\!])$ 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.

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

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. □

THEOREM 10. *When $C = \{\top\}$ (no fairness), $E_C G'$ becomes*

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

Proof:
Let $F(\mathbb{Z}) = [\![\varphi]\!] \cap_L [\![EXE[\varphi \; U \; \varphi \wedge \mathbb{Z}]]\!]$ and $G(\mathbb{Z}) = [\![\varphi]\!] \cap_L [\![EX\mathbb{Z}]\!]$. By definition, $[\![E_C G'\varphi]\!] = \mathbb{W} = \nu\mathbb{Z}.F(\mathbb{Z})$ 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 $\mathbb{W}$:

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

[2] F^n stands for applying F n times.

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 $\mathbb{W} \supseteq_L \mathbb{Y}$.

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

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 $\forall i \in nat \cdot \mathbb{W}_i \supseteq_L \mathbb{Y}_i$.

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

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, by definition, $[\![E_C G'\varphi]\!] = [\![EG\varphi]\!]$. ☐

Now we set out to show that $E_C G$ and $E_C G'$ 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, $E_C G\varphi = \nu\mathbb{Z}.F(\mathbb{Z})$. Also, let $G(\mathbb{Z}) = [\![\varphi]\!] \ \cap_L \bigcap_{L\ k=\{1,2\}} [\![EXE[\varphi\ U\ \varphi \wedge c_k \cap_L \mathbb{Z}]]\!]$. Then, by definition, $E_C G'\varphi = \nu\mathbb{Z}.G(\mathbb{Z})$.

We are interested in representing paths on which the sequence c_1, c_2 (respectively, c_2, c_1) holds i times. We do so by encoding the states from which these paths emanate, using mv-sets $\mathbb{K}_i$ and $\mathbb{M}_i$, respectively. These are defined recursively as follows:

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

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 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 $\mathbb{M}_n$

The above 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

The above 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 $n = k$

Ind. Case: Proof for $n = k + 1$

(1) Enough to show $\mathbb{K}_{n+1} \supseteq_L G_1(G^n([\![\top]\!]))$

$$
\begin{array}{lll}
 & \top & \text{IH} \\
\Rightarrow & \mathbb{M}_n \supseteq_L G^n([\![\top]\!]) & \text{monotonicity 2 of } EU \\
\Rightarrow & \quad [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_1 \wedge \mathbb{M}_n]]\!] & \\
 & \supseteq_L [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_1 \wedge G^n([\![\top]\!])]]\!] & \text{def. of } \mathbb{K}_{n+1}, G_1, G^{n+1} \\
\Rightarrow & \mathbb{K}_{n+1} \supseteq_L G_1(G^n([\![\top]\!])) \supseteq_L G^{n+1}([\![\top]\!]) &
\end{array}
$$

(2) Proof is similar to that of (1).

(3) Need to show $\mathbb{K}_{2n+2} \subseteq_L G^{n+1}([\![\top]\!])$

We 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 have the desired property.

$$
\begin{array}{lll}
(3a) & \mathbb{K}_{2n+2} & \text{def. of } \mathbb{K}_{2n+2} \\
= & [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_1 \wedge EXE[\varphi\ U\ \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]]\!] & \text{relation between } \mathbb{K}_n \text{ and } \mathbb{M}_n \\
\subseteq_L & [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_1 \wedge \mathbb{M}_{2n}]]\!] & \text{IH and monotonicity} \\
\subseteq_L & [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_1 \wedge G^n([\![\top]\!])]]\!] & \text{def. of } G_1 \\
= & G_1(G^n([\![\top]\!])) & \\
(3b) & \mathbb{K}_{2n+2} & \text{def. of } \mathbb{K}_{2n+2} \\
= & [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_1 \wedge EXE[\varphi\ U\ \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]]\!] & \text{monotonicity} \\
\subseteq_L & [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge EXE[\varphi\ U\ \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]]\!] & \text{monotonicity 1 of } EU \\
\subseteq_L & [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]\!] & \text{IH and monotonicity} \\
\subseteq_L & [\![\varphi]\!] \cap_L [\![EXE[\varphi\ U\ \varphi \wedge c_2 \wedge G^n([\![\top]\!])]]\!] & \text{def. of } G_2 \\
= & G_2(G^n([\![\top]\!])) &
\end{array}
$$

(4) Proof is similar to that of (3).

□

THEOREM 11. *Operators $E_C G$ and $E_C G'$ 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, $\forall n \geq N_2$

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

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, operators $E_C G$ and $E_C G'$ are equivalent. □