Model-Checking Over Multi-Valued Logics

Marsha Chechik, Steve Easterbrook, and Victor Petrovykh

{chechik,sme,victor}@cs.toronto.edu
Department of Computer Science, University of Toronto
Toronto ON M5S 3G4, Canada

Abstract. Classical logic cannot be used to effectively reason about systems with *uncertainty* (lack of essential information) or *inconsistency* (contradictory information often occurring when information is gathered from multiple sources). In this paper we propose the use of *quasi-boolean* multi-valued logics for reasoning about such systems. We also give semantics to a multi-valued extension of CTL, describe an implementation of a symbolic multi-valued CTL model-checker called χchek, and analyze its correctness and running time.

1 Introduction

In the last few years, *model checking* [9] has become established as one of the most effective automated techniques for analyzing correctness of software artifacts. Given a system and a property, a model checker builds the reachability graph (explicitly or symbolically) by exhaustively exploring the state-space of the system. Model-checking has been effectively applied to reasoning about correctness of hardware, communication protocols, software requirements and code, etc. A number of industrial model checkers have been developed, including SPIN [18], SMV [22], and Murϕ [12].

Despite their variety, existing model-checkers are typically limited to reasoning in classical logic. However, there are a number of problems in software engineering for which classical logic is insufficient. One of these is reasoning under *uncertainty*, or when essential information is not available. This can occur either when complete information is not known or cannot be obtained (e.g., during requirements analysis), or when this information has been removed (abstraction). Classical model-checkers typically deal with uncertainty by creating extra states, one for each value of the unknown variable and each feasible combination of values of known variables. However, this approach adds significant extra complexity to the analysis.

Classical reasoning is also insufficient for models that contain *inconsistency*. Inconsistency arises frequently in software engineering [15]. In requirements engineering, models are frequently inconsistent because they combine conflicting points of view. During design and implementation, inconsistency arises when integrating components developed by different people. Conventional reasoning systems cannot cope with inconsistency; the presence of a single contradiction results in trivialization — anything follows from $A \wedge \neg A$. Hence, faced with an inconsistent description and the need to perform automated reasoning, we must either discard information until consistency is achieved again, or adopt a non-classical logic. The problem with the former approach is that we may be forced to make premature decisions about which information to discard [19].

Although inconsistency in software engineering occurs very frequently, there have been relatively few attempts to develop automated reasoning tools for inconsistent models. Two notable exceptions are Hunter and Nuseibeh [20], who use a Quasi-Classical (QC) logic to reason about evolving specifications, and Menzies et al. [23], who use a paraconsistent form of abductive inference to reason about information from multiple points of view.

Paraconsistent logics are a promising alternative to classical reasoning — they permit some contradictions to be true, without the resulting trivialization of classical logic. The development of paraconsistent logics has been driven largely by the need for automated reasoning systems that do not give spurious answers if their databases become inconsistent. They are also of interest to mathematicians as a way of addressing the paradoxes in semantics and set theory. A number of different types of paraconsistent logic have been studied [24]. For example, relevance logics use an alternative form of entailment that requires a "relevant" connection between the antecedents and the consequents. Non-truth functional logics use a weaker form of negation so that proof rules such as disjunctive syllogism (i.e., $(A \vee B, \neg B) \vdash A$) fail. Multi-valued logics use additional truth values to represent different types of contradiction.

Multi-valued logics provide a solution to both reasoning under uncertainty and under inconsistency. For example, we can use "no information available" and "no agreement" as logic values. In fact, model-checkers based on three-valued and four-valued logics have already been studied. For example, [8] used a three-valued logic for interpreting results of model-checking with abstract interpretation, whereas [16, 17] used four-valued logics for reasoning about abstractions of detailed gate or switch-level designs of circuits.

Different multi-valued logics are useful for different purposes. For example, we may wish to have several levels of uncertainty. We may wish to use different multi-valued logics to support different ways of merging information from multiple sources: keeping track of the origin of each piece of information, doing a majority vote, giving priority to one information source, etc. Thus, rather than restricting ourselves to any particular multi-valued logic, we are interested in extending the classical symbolic model-checking procedure to enable reasoning about arbitrary multi-valued logics, as long as conjunction, disjunction and negation of the logical values are specified.

This work is part of the χbel[1] (the Multi-Valued **B**elief **E**xploration **L**ogics) project, outlined in [14]. The description of the system together with the description of the desired multi-valued logic and the set of correctness criteria expressed in CTL become input to our model-checker, called χchek, which returns a value of the logic best characterizing the validity of the formula in each initial state.

The rest of this paper is organized as follows: Section 2 describes a simple thermostat system which is used as a running example throughout the paper. Section 3 gives background on CTL model-checking. Section 4 describes the types of logics that we can analyze and the ways to represent them. Section 5 describes the multi-valued transition structures and extends CTL to reasoning over them. Section 6 discusses the implementation of χchek, whereas Section 7 contains the analysis of its correctness and running

[1] pronounced "Chibel"

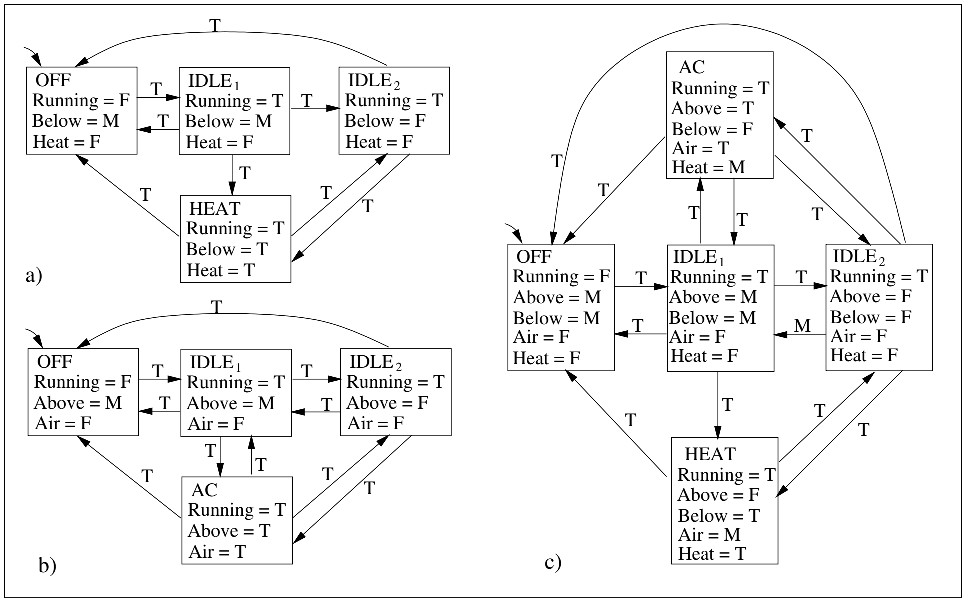

Fig. 1. Models of the thermostat. (a) Heat only; (b) AC only; (c) combined model.

time. We conclude the paper with a summary of results and outline of future work in Section 8.

2 Example

Consider three models of the thermostat given in Figure 1. Figure 1(a) describes a very simple thermostat that can run a heater if the temperature falls below desired. The system has one indicator (`Below`), a switch to turn it off and on (`Running`) and a variable indicating whether the heater is running (`Heat`). The system starts in state `OFF`[2] and transitions into $\mathtt{IDLE}_1$ when it is turned on, where it awaits the reading of the temperature indicator. Once the temperature is determined, the system transitions either into $\mathtt{IDLE}_2$ or into `HEAT`. The value of the temperature indicator is unknown in states `OFF` and $\mathtt{IDLE}_1$. To model this, we could duplicate the states, assigning `Below` the value T in one copy and F in the other — the route typically taken by conventional model-checkers. Alternatively, we can model the system using the three-valued logic: T, F and M (Maybe), assigning `Below` the value M, as depicted in Figure 1(a)[3].

We can ask this thermostat model a number of questions:

Prop. 1. Can the system transition into $\mathtt{IDLE}_1$ from everywhere?
Prop. 2. Can the heater be turned on when the temperature becomes below desired?
Prop. 3. Can the system be turned off in every computation?

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

[3] Each state in this and the other two systems in Figure 1 contains a self-loop with the value T which we omitted to avoid clutter.

Figure 1(b) describes another aspect of the thermostat system – running the air conditioner. The behavior of this system is similar to that of the heater, with one difference: this system handles the failure of the temperature indicator. If the temperature reading cannot be obtained in states `AC` or $\texttt{IDLE}_2$, the system transitions into state $\texttt{IDLE}_1$.

Finally, Figure 1(c) contains a merged model, describing the behavior of the thermostat that can run both the heater and the air conditioner. In this merge, we used the same three-valued logic, for simplicity. When the individual descriptions agree that the value of a variable or transition is T (F), it is mapped into T (F) in the combined model; all other values are mapped into M. During the merge, we used the simple invariants describing the behavior of the environment ($\texttt{Below} \rightarrow \neg\texttt{Above}$, $\texttt{Above} \rightarrow \neg\texttt{Below}$). Thus, the value of `Below` in state `AC` is inferred to be F. Note that the individual descriptions disagree on some states and transitions. For example, they disagree on a transition between $\texttt{IDLE}_2$ and $\texttt{IDLE}_1$; thus it receives the value M. Also, it is possible that the heater is on while the air conditioner is running.

Further details on the merge procedure are outside the scope of this paper, except to note that we could have chosen any of a number of different multi-valued logics to handle different combinations of values in the individual models. For example, we could have used a 9-valued logic where each value is a tuple formed from the values of the two individual models.

We can ask the combined model a number of questions that cannot be answered by either individual model, e.g.

Prop. 4. Is heat on only if air conditioning is off?
Prop. 5. Can heat be on when the temperature is above desired?

3 CTL Model-Checking

CTL model-checking is an automatic technique for verifying properties expressed in a propositional branching-time temporal logic called *Computational Tree Logic* (CTL) [9]. The system is defined by a Kripke structure, and properties are evaluated on a tree of infinite computations produced by the model of the system. The standard notation $M, s \models P$ indicates that a formula P 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 $L : S \rightarrow \mathcal{P}(A)$. R must be total, i.e, $\forall s \in S,\ \exists t \in S$, s.t, $(s, t) \in R$. If a state s_n has no successors, we add a self-loop to it, so that $(s_n, s_n) \in R$. For each $s \in S$, the labeling function provides a list of atomic propositions which are *True* in this state.

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 $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $EX\varphi$, $AX\varphi$, $EF\varphi$, $AF\varphi$, $E[\varphi U \psi]$, $A[\varphi U \psi]$.

The logic connectives $\neg$, $\wedge$ and $\vee$ have their usual meanings. The existential (universal) quantifier E (A) is used to quantify over paths. The operator X means "at the next step",

F represents "sometime in the future", and U is "until". Therefore, $EX\varphi$ ($AX\varphi$) means that φ holds in some (every) immediate successor of the current program state; $EF\varphi$ ($AF\varphi$) means that φ holds in the future along some (every) path emanating from the current state; $E[\varphi U\psi]$ ($A[\varphi U\psi]$) means that for some (every) computation path starting from the current state, φ continuously holds until ψ becomes true. Finally, we use $EG(\varphi)$ and $AG(\varphi)$ to represent the property that φ holds at every state for some (every) path emanating from s_0. Formally,

$$
\begin{aligned}
M, s_0 \models a &\text{ iff } a \in L(s_0)\\
M, s_0 \models \neg\varphi &\text{ iff } M, s_0 \not\models \varphi\\
M, s_0 \models \varphi \wedge \psi &\text{ iff } M, s_0 \models \varphi \ \wedge \ M, s_0 \models \psi\\
M, s_0 \models \varphi \vee \psi &\text{ iff } M, s_0 \models \varphi \ \vee \ M, s_0 \models \psi\\
M, s_0 \models EX\varphi &\text{ iff } \exists t \in S,\ (s_0, t) \in R \ \wedge \ M, t \models \varphi\\
M, s_0 \models AX\varphi &\text{ iff } \forall t \in S,\ (s_0, t) \in R \rightarrow M, t \models \varphi
\end{aligned}
$$

$$
\begin{aligned}
M, s_0 \models E[\varphi U\psi] \text{ iff } & \text{there exists some path } s_0, s_1, \ldots, \text{ s.t.}\\
& \exists i,\ i \geq 0 \ \wedge \ M, s_i \models \psi \ \wedge\\
& \forall j,\ 0 \leq j < i \rightarrow M, s_j \models \varphi\\
M, s_0 \models A[\varphi U\psi] \text{ iff } & \text{for every path } s_0, s_1, \ldots,\\
& \exists i,\ i \geq 0 \ \wedge \ M, s_i \models \psi \ \wedge\\
& \forall j,\ 0 \leq j < i \rightarrow M, s_j \models \varphi.
\end{aligned}
$$

where the remaining operators are defined as follows:

$$
\begin{aligned}
AF(\varphi) &\equiv A[\top U\varphi] && (\text{def. of } AF)\\
EF(\varphi) &\equiv E[\top U\varphi] && (\text{def. of } EF)\\
AG(\varphi) &\equiv \neg EF(\neg\varphi) && (\text{def. of } AG)\\
EG(\varphi) &\equiv \neg AF(\neg\varphi) && (\text{def. of } EG)
\end{aligned}
$$

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.

4 Specifying the Logic

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

4.1 Lattice Theory

We introduce lattice theory here following the presentation in [2].

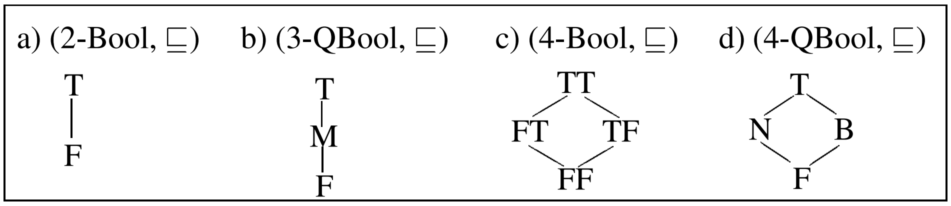

Fig. 2. Examples of logic lattices: (a) a two-valued lattice representing classical logic; (b) a three-valued lattice reflecting uncertainty; (c) a four-valued boolean lattice, a product of two (2-Bool, $\sqsubseteq$) lattices; (d) a four-valued quasi-boolean lattice.

Definition 1 Lattice *is a partial order ($\mathcal{L}$, $\sqsubseteq$) for which a unique greatest lower bound and least upper bound, denoted* $a \sqcap b$ *and* $a \sqcup b$ *exist for each pair of elements* (a, b).

The following are the properties of lattices:

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

$a \sqcap b$ and $a \sqcup b$ are referred to as *meet* and *join*, representing for us conjunction and disjunction operations, respectively. Figure 2 gives examples of a few logic lattices. Our partial order operation $a \sqsubseteq b$ means that "b is more true than a".

Definition 2 *A lattice is* distributive *if*

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

All lattices in Figure 2 are distributive.

Definition 3 *A lattice is* complete *if the least upper bound and the greatest lower bound for each subset of elements of the lattice is an element of the lattice. Every complete lattice has a top and bottom.*

$$
\begin{array}{ll}
\bot = \sqcap \mathcal{L} & (\bot \text{ characterization}) \\
\top = \sqcup \mathcal{L} & (\top \text{ characterization})
\end{array}
$$

In this paper we use T to indicate $\top$ of the lattice, and F to indicate its $\bot$, although in principle $\top$ and $\bot$ might be labelled differently.

Finite lattices are complete by definition. Thus, all lattices representing finite-valued logics are complete.

Definition 4 *A complete distributive lattice is called a* complete Boolean lattice *if every element* $a \in \mathcal{L}$ *has a unique* complement $\neg a \in \mathcal{L}$ *satisfying the following conditions:*

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

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

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

$$(a, b) \sqsubseteq (a', b') \Leftrightarrow a \sqsubseteq a' \;\wedge\; b \sqsubseteq b'$$

Bottom, top, complement, meet and join in the product lattice are component-wise extensions of the corresponding operations of the component lattices. Product of two lattices preserves their distributivity, completeness and boolean properties. For example, out of the four lattices in Figure 2, only (2-Bool, $\sqsubseteq$) and (4-Bool, $\sqsubseteq$) are boolean. The former is boolean because $\neg$ T = F, $\neg$ F = T. The latter is a product of two (2-Bool, $\sqsubseteq$) lattices and thus is complete, distributive and boolean. The lattice (3-QBool, $\sqsubseteq$) is not boolean because $\neg$ M = M, and M $\sqcap \neg$ M $\neq \bot$.

4.2 Quasi-Boolean Lattices

Definition 6 *A distributive lattice* $(\mathcal{L}, \sqsubseteq)$ *is* quasi-boolean *[4] (also called* de Morgan *[13]) if there exists a unary operator* $\neg$ *defined for it, with the following properties* (a, b *are elements of* $(\mathcal{L}, \sqsubseteq)$*):*

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

Thus, $\neg a$ *is a* quasi-complement *of* a.

Therefore, all boolean lattices are also quasi-boolean, whereas the converse is not true. Logics represented by quasi-boolean lattices will be referred to as *quasi-boolean logics*.

Theorem 1 *A product of two quasi-boolean lattices is quasi-boolean*

Proof:
Refer to the Appendix for proof of this and other theorems of this paper. □

For example, the lattice (3-QBool, $\sqsubseteq$), first defined in [21], and all its products are quasi-boolean. We refer to n-valued boolean lattices as (n-Bool, $\sqsubseteq$) and to quasi-boolean lattices as (n-QBool, $\sqsubseteq$). (4-QBool, $\sqsubseteq$) is a lattice for a logic proposed by Belnap for reasoning about inconsistent databases [3, 1]. This lattice is quasi-boolean ($\neg$N = N; $\neg$B = B) and thus not isomorphic to (4-Bool, $\sqsubseteq$).

In the rest of this paper we assume that the negation operator given for our logic makes the lattice quasi-boolean. What do quasi-boolean lattices look like? Below we define lattices which are (geometrically) horizontally-symmetric and show that, with negation defined by the horizontal symmetry, this is a sufficient condition for quasi-booleanness. We define:

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

$$\begin{aligned} a \sqsubseteq b \;\Leftrightarrow\; H(a) &\sqsupseteq H(b) \quad (\mathrm{order-embedding}) \\ H(H(a)) &= a \quad (\mathrm{H\ involution}) \end{aligned}$$

Theorem 2 *Let $(\mathcal{L}, \sqsubseteq)$ be a horizontally-symmetric lattice. Then the following hold for any two elements a, $b \in \mathcal{L}$:*

$$\begin{aligned} H(a \sqcap b) &= H(a) \sqcup H(b) \\ H(a \sqcup b) &= H(a) \sqcap H(b) \end{aligned}$$

Thus, horizontal symmetry is a sufficient condition for the corresponding lattice to be quasi-boolean, with $\neg a = H(a)$ for each element of the lattice, since it guarantees antimonotonicity and involution by definition, and de Morgan laws via Theorem 2.

5 Multi-Valued CTL Model-Checking

In this section we extend the notion of boolean model-checking described in Section 3 by defining multi-valued Kripke structures and multi-valued CTL.

5.1 Defining the Model

A state machine M is a *multi-valued Kripke (χKripke) structure* if $M = (S, S_0, R, I, A, L)$, where

- L is a quasi-boolean logic represented by a lattice ($\mathcal{L}$, $\sqsubseteq$).
- A is a (finite) set of atomic propositions, otherwise referred to as variables (e.g. `Running`, `Below`, `Heat` in Figure 1(a)). For simplicity, we assume that all variables are of the same type.
- S is a (finite) set of states; each state is identified by a unique (within M) label s. $S_0 \subseteq S$ is the non-empty set of initial states.
- Each transition (s, t) in M has a logical value in $\mathcal{L}$, referred to as $\langle s, t\rangle^M$, or, when M is clear from the context, simply as $\langle s, t\rangle$. Then, $R : \mathcal{L} \to \mathcal{P}(S \times S)$ is the labeling function mapping logical value $v \in \mathcal{L}$ into a set of transitions $\{(s, t)\}$ where the value of each (s, t) is v, i.e.

  $$\forall v \in \mathcal{L}, R(v) = \{(s, t) \mid s \in S \wedge t \in S \wedge \langle s, t\rangle = v\}$$

  We also ensure that there is at least one non-false transition out of each state, extending the classical notion of Kripke structures. Formally,

  $$\forall s \in S, \exists t \in S, \exists v \in \mathcal{L} \text{ s.t. } v \neq \bot \wedge (s, t) \in R(v)$$

$\neg AX\varphi = EX(\neg\varphi)$	(negation of "`next`")
$A[\bot U\varphi] = E[\bot U\varphi] = \varphi$	($\bot$ "`until`")
$A[\varphi U\psi] = \psi \vee (\varphi \wedge AXA[\varphi U\psi] \wedge EXA[\varphi U\psi])$	(AU fixpoint)
$E[\varphi U\psi] = \psi \vee (\varphi \wedge EXE[\varphi U\psi])$	(EU fixpoint)

Fig. 3. Properties of CTL operators.

To avoid clutter, we follow the convention of finite-state machines of not drawing F transitions. Thus, in Figure 1(a), transition between $\mathtt{IDLE}_2$ and $\mathtt{IDLE}_1$ is F, whereas in Figure 1(c) this transition is M. We refer to a value that a variable (an atomic proposition) a takes in state s as $\langle a\rangle_s^M$, or, when M is clear from context, simply as $\langle a\rangle_s$.

– $I : A \times \mathcal{L} \to \mathcal{P}(S)$ is a labeling function that maps each atomic proposition (variable) a and each logical value v to a set of states where the value of a is v, i.e.,

$$\forall v \in \mathcal{L}, \forall a \in A,\ I(a, v) = \{s \mid s \in S \ \wedge\ \langle a\rangle_s^M = v\}$$

Thus, the expression $(\langle\varphi\rangle_s = \mathrm{T}) \vee (\langle s, t\rangle = \mathrm{F})$ means that we check whether the value of φ in s is T, or whether the value of the transition (s, t) is F. Alternatively, and equivalently, we can think of a state s as a vector of variables A and their values in $\mathcal{L}$ together with a unique label s.

5.2 Multi-Valued CTL

Here we give semantics of CTL operators on a χKripke structure M over a quasi-boolean logic L. We will refer to this language as *multi-valued CTL, or χCTL*. L is described by a finite, quasi-boolean lattice $(\mathcal{L}, \sqsubseteq)$, and thus the conjunction $\wedge$ ($\sqcap$ operation of the lattice), disjunction $\vee$ ($\vee$ operation of the lattice) and negation $\neg$ operations are available. We also define implication $\to$ as follows:

$$a \to b \ \equiv\ \neg a \vee b \quad \text{(definition of } \to)$$

In extending the CTL operators, we want to ensure that the expected CTL properties, given in Figure 3, are still preserved. Note that the AU fixpoint is somewhat unusual because it includes an additional conjunct, $EXA[fUg]$. The reason for this term is to preserve a "strong until" semantics for states that have no outgoing T transitions. This term was introduced by [6] for reasoning about non-Kripke structures.

We start defining χCTL by giving the semantics of propositional operators. Here, s is a state and $v \in \mathcal{L}$ is a logic value:

$$\begin{aligned}
\langle a\rangle_s &= v \text{ iff } s \in I(a, v)\\
\langle \neg\varphi\rangle_s &= v \text{ iff } \langle\varphi\rangle_s = \neg v\\
\langle \varphi \wedge \psi\rangle_s &= v \text{ iff } (\langle\varphi\rangle_s \wedge \langle\psi\rangle_s) = v\\
\langle \varphi \vee \psi\rangle_s &= v \text{ iff } (\langle\varphi\rangle_s \vee \langle\psi\rangle_s) = v
\end{aligned}$$

We proceed by defining EX and AX operators. Recall from Section 3 that these operators were defined using existential and universal quantification over next states. We extend the notion of quantification for multi-valued reasoning by using conjunction and

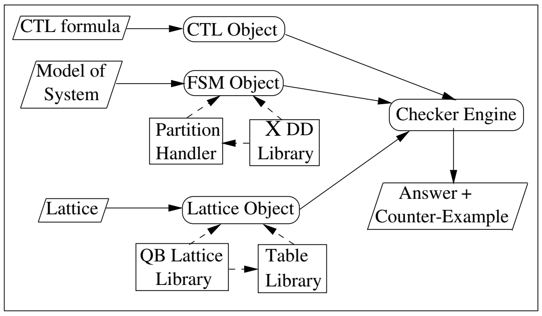

Fig. 4. Architecture of χchek.

disjunction operators. This treatment of quantification is standard [3, 25]. The semantics of EX and AX operators is given below:

$$\langle EX\varphi\rangle_s = v \text{ iff } (\bigvee_{t\in \mathrm{succ}(s)}(\langle s,t\rangle \wedge \langle\varphi\rangle_t)) = v$$
$$\langle AX\varphi\rangle_s = v \text{ iff } (\bigwedge_{t\in \mathrm{succ}(s)}(\langle s,t\rangle \rightarrow \langle\varphi\rangle_t)) = v$$

Theorem 3 *Definitions of $\langle EX\varphi\rangle_s$ and $\langle AX\varphi\rangle_s$ preserve the negation of "next" property, i.e.*

$$\forall s \in S,\ \neg\langle AX\varphi\rangle_s = \langle EX\neg\varphi\rangle_s$$

Finally, we define AU and EU operators using the AU and EU fixpoint properties and the above definition of AX and EX:

$$\langle E[\varphi U\psi]\rangle_s = v \text{ iff } (\langle\psi\rangle_s \vee (\langle\varphi\rangle_s\ \wedge\ \bigvee_{t\in \mathrm{succ}(s)}(\langle s,t\rangle \wedge \langle E[\varphi U\psi]\rangle_t))) = v$$
$$\langle A[\varphi U\psi]\rangle_s = v \text{ iff } (\langle\psi\rangle_s \vee (\langle\varphi\rangle_s\ \wedge\ \bigwedge_{t\in \mathrm{succ}(s)}(\langle s,t\rangle \rightarrow \langle A[\varphi U\psi]\rangle_t)$$
$$\wedge\ \bigvee_{t\in \mathrm{succ}(s)}(\langle s,t\rangle \wedge \langle A[\varphi U\psi]\rangle_t))) = v$$

The remaining CTL operators, $AF(\varphi)$, $EF(\varphi)$, $AG(\varphi)$, $EG(\varphi)$ are the abbreviations for $A[\top U\varphi]$, $E[\top U\varphi]$, $\neg EF(\neg\varphi)$, $\neg AF(\neg\varphi)$, respectively.

6 χchek: A Multi-Valued Model-Checker

In this section we describe our implementation of a multi-valued CTL model-checker. This symbolic model-checker, called χchek, is written in Java, and its architecture is depicted in Figure 4. The checking engine receives the χCTL formulas to verify, the model of the system represented as an χKripke structure, and a lattice of logic values, and checks whether the specified property holds, returning an answer (one of the values of the passed lattice) and a counter-example, if appropriate. χchek uses four supplementary libraries: χDDs (a multi-valued extension of binary decision diagrams [5], described in [10]), a library for handling quasi-boolean lattices, a partition handler and a table inverter. The functionality of the latter two libraries is described below.

6.1 Table Library

The Table library contains several tables, indexed by the elements of the lattice, that give quick access to a variety of operations on lattice elements. In order to enable this indexing, we define $\mathtt{Ord} : \mathcal{L} \rightarrow \mathbb{N}$ — a total order on the elements of our lattice $(\mathcal{L}, \sqsubseteq)$. $\mathtt{Ord}()$

is a bijection, mapping each element $v \in \mathcal{L}$ onto the set $\{1...|\mathcal{L}|\}$. For example, we can order the elements of the lattice (3-QBool, $\sqsubseteq$) as follows:

$\texttt{Ord}$(T) = 1 $\qquad$ $\texttt{Ord}$(M) = 2 $\qquad$ $\texttt{Ord}$(F) = 3

This ordering is referred to as T < M < F.

Using $\texttt{Ord}()$ and the primitive lattice operations, we compute *inverted tables*: given a value, these tables give pairs of elements yielding this value when the corresponding operation is performed on them. Three inverted tables, $\texttt{InvTable}_\wedge$, $\texttt{InvTable}_\rightarrow$ and $\texttt{InvTable}_\vee$ are computed, one for each operator. For a table T and a value v, we use notation T_v to indicate an element associated with value v. $\texttt{InvTable}_\vee$ is defined as

$$\forall v \in \mathcal{L},\ \texttt{InvTable}_{\vee,v} = \{(v_1, v_2) \mid v_1, v_2 \in \mathcal{L} \vee v_1 \wedge v_2 = v\}$$

For example, for the lattice (3-QBool, $\sqsubseteq$), $\texttt{InvTable}_{\vee,\mathrm{M}} = \{$(M, M), (F, M), (M, F)$\}$. $\texttt{InvTable}_\wedge$ and $\texttt{InvTable}_\rightarrow$ are defined similarly.

Afterwards, we build generalized versions of the inverted tables for conjunction and disjunction over more than two operands. We call them $\texttt{BigOPTable}_\wedge$ and $\texttt{BigOPTable}_\vee$. Given a logic value v, $\texttt{BigOPTable}_{op,v}$ gives sets of logic values, where the corresponding operation op over the elements of the set yields v. For example, for the lattice (3-QBool, $\sqsubseteq$), $\texttt{BigOPTable}_{\vee,\mathrm{T}}$ is $\{\{$T$\}, \{$T, M$\}, \{$T, F$\}, \{$T, M, F$\}\}$. $\texttt{BigOPTable}_\vee$ is defined as

$$\forall v \in \mathcal{L},\ \texttt{BigOPTable}_{\vee,v} = \{V \mid V \in \mathcal{P}(\mathcal{L}) \ \wedge\ \bigvee V = v\}$$

$\texttt{BigOPTable}_\wedge$ is defined similarly.

6.2 The Partition Handler

Central to the design of χchek is the notion of a *partition*. A partition separates the states of the model into subsets corresponding to the different values of the logic for a proposition φ. More formally, a partition $[\varphi]^M$ for a property φ and a machine M is a tuple of sets such that the ith element of the tuple is a set of states where φ has the value $\texttt{Ord}^{-1}(i)$ in M. When the choice of M is clear, we omit it from notation, referring to a partition simply as $[\varphi]$. The sets of states in a partition are mutually disjoint; thus, each proposition partitions the state space. For a value $v \in \mathcal{L}$, we write $[\varphi]_v$ to indicate the set of states associated with v. For an example in Figure 1(a) and ordering T < M < F, $[\texttt{Below}] = (\{\texttt{HEAT}\}, \{\texttt{OFF}, \texttt{IDLE}_1\}, \{\texttt{IDLE}_2\})$. Thus,

$$\forall s \in S, \forall v \in \mathcal{L},\ (s \in [\varphi]_v \ \Leftrightarrow\ \langle\varphi\rangle_s = v)$$

In fact, operations $\langle\rangle$ and $[]$ are Galois-connected. We also refer to $\texttt{PartitionType}$ as a type $(s_i | 1 \leq i \leq |\mathcal{L}| \rightarrow s_i \in \mathcal{P}(S))$. Note that $[\varphi]$ is of type PartitionType; however, an element of PartitionType does not require its sets to be mutually-disjoint, although the union of the sets should still cover the entire state-space.

Further, we define a *predecessor function* $\texttt{pred}()$ which receives a partition $[\varphi]$ and an operator $op \in \{\wedge, \rightarrow\}$ and returns a PartitionType. A state s is associated with value v_1 in $\texttt{pred}([\varphi], op)$ iff s has a successor state t where φ has value v_2, and $\langle s, t\rangle\ op\ v_2 =$

function pred $([\varphi], op)\{$
    foreach $v \in \mathcal{L}$
        $[\text{pred}]_v := \{s \mid \exists t \in S, \exists (v_1, v_2) \in \mathtt{InvTable}_{op,v}, \text{s.t.}$
                $((s,t) \in R(v_1)) \ \wedge \ (\langle\varphi\rangle_t = v_2)\}$
    return pred
$\}$
function doOP$([\varphi], op, [\psi])$ $\{$
    foreach $v \in \mathcal{L}$
        $[\text{result}]_v := \{[\varphi]_a \cap [\psi]_b \mid (a,b) \in \mathtt{InvTable}_{op,v}\}$
    return result
$\}$
function doBigOP$(op, [\varphi])$ $\{$
    foreach $v \in \mathcal{L}$
        $[\text{result}]_v := \emptyset$
        foreach $V \in \mathtt{BigOPTable}_{op,v}$
            $[\text{result}]_v := \{\bigcap_{v_i \in V} [\varphi]_{v_i} - \bigcup_{v_i \in (\mathcal{L} - V)} [\varphi]_{v_i}\} \cup [\text{result}]_v$
    return result
$\}$
function QUntil$(\text{quantifier}, [\varphi], [\psi])$ $\{$
    $QU_0 = [\psi]$
    repeat
        $\text{EXTerm}_{i+1} := \mathtt{doBigOP}(\vee, \mathtt{pred}(QU_i, \wedge))$
        if (quantifier is A)
            $\text{AXTerm}_{i+1} := \mathtt{doBigOP}(\wedge, \mathtt{pred}(QU_i, \rightarrow))$
        else
            $\text{AXTerm}_{i+1} := [\varphi]$
        foreach $v_1, v_2, v_3, v_4 \in \mathcal{L}$
            toMove := $[\varphi]_{v_1} \cap [\psi]_{v_2} \cap [\text{AXTerm}_{i+1}]_{v_3} \cap [\text{EXTerm}_{i+1}]_{v_4}$
            dest := $(v_1 \wedge v_3 \wedge v_4) \vee v_2$
            move all the states in toMove to $[QU_{i+1}]_{\mathrm{dest}}$
    until $QU_{i+1} = QU_i$
    return QU_n
$\}$
procedure Check$(p)\{$
    Case
        $p \in A$: **return** [p] **where** $\forall v \in \mathcal{L}, [p]_v := I(p,v)$
        $p = \neg\varphi$: **return** [p] **where** $\forall v \in \mathcal{L}, [p]_v := [\varphi]_{\neg v}$
        $p = \varphi \wedge \psi$: **return** doOp$([\varphi], \wedge, [\psi])$
        $p = \varphi \vee \psi$: **return** doOp$([\varphi], \vee, [\psi])$
        $p = EX\varphi$: **return** $\mathtt{doBigOP}(\vee, \mathtt{pred}([\varphi], \wedge))$
        $p = AX\varphi$: **return** $\mathtt{doBigOP}(\wedge, \mathtt{pred}([\varphi], \rightarrow))$
        $p = E[\varphi U \psi]$: **return** $\mathtt{QUntil}(\mathtt{E}, [\varphi], [\psi])$
        $p = A[\varphi U \psi]$: **return** $\mathtt{QUntil}(\mathtt{A}, [\varphi], [\psi])$
$\}$

Fig. 5. Algorithm for χchek.

v_1. The function is given in Figure 5. Note that the result of pred is not necessarily a partition: a state can be in more than one set. For the lattice (3-QBool, $\sqsubseteq$), its ordering T < M < F and the model in Figure 1(c), $\texttt{pred}([\texttt{Running}], \wedge)$ returns $(\{\texttt{IDLE}_1, \texttt{IDLE}_2, \texttt{AC}, \texttt{HEAT}\}$, $\{\texttt{IDLE}_2\}$, $\{\texttt{IDLE}_1, \texttt{IDLE}_2, \texttt{AC}, \texttt{HEAT}, \texttt{OFF}\}$.

We further define functions $\texttt{doOp}()$ and $\texttt{doBigOp}()$, described in Figure 5. These functions evaluate the expression using the appropriate table ($\texttt{InvTable}_{op}$ or $\texttt{BigOPTable}_{op}$). Given partitions $[\varphi]$ and $[\psi]$, doOP returns a partition for $\varphi \;\; op \;\; \psi$. For the lattice (3-QBool, $\sqsubseteq$) and the model in Figure 1(c), $[\texttt{doOP}([\texttt{Above}], \vee, [\texttt{Below}])]_{\mathrm{T}}$ returns a set of states in which $\texttt{Above} \vee \texttt{Below}$ is T, namely, $\{\texttt{AC}, \texttt{HEAT}\}$.

$\texttt{doBigOP}(op, [\varphi])$ computes BigOP over a collection of states. Here, $[\varphi]$ is required to be of PartitionType but does not need to be a partition, that is, the union of sets of states associated with each value of $[\varphi]$ includes the entire state space, but these sets are not necessarily pairwise disjoint. $\texttt{BigOPTable}_{op,v}$ includes sets of logic values such that the operation op performed on them yields v. Thus, for each V in $\texttt{BigOPTable}_{op,v}$, we compute the intersection of states for which φ has a value in V and subtract the union of states in which φ does not have a value in V. $[\mathrm{result}]_v$ is the union of all states computed via the above process for all V in $\texttt{BigOPTable}_{op,v}$. For the model in Figure 1(c), $[\texttt{doBigOP}(\vee, [\texttt{Heat}]_{)}]_{\mathrm{T}}$ returns $\{\texttt{HEAT}\}$. Note that if $[\varphi]$ represents a partition, $\texttt{doBigOP}(op, [\varphi])$ simply returns a partition $[\psi]$ where $[\psi]_v = [\varphi]_v$ for each $v \in \mathcal{L}$.

6.3 Algorithm of χchek

The high-level algorithm, inspired by Bultan's symbolic model checker for infinite-state systems [6, 7] and an abstract model-checker of [8], is given in procedure $\texttt{Check}()$ in Figure 5. In order to formulate the algorithm, we extend our notation. If $[\varphi]$ is a partition, let $\langle[\varphi]\rangle_s$ refer to the value that state s has in this partition.

The algorithm recursively goes through the structure of the property under the analysis, associating each subproperty φ with a partition $[\varphi]$. In fact, Check always returns partitions on the state-space (see Theorem 5). For the example in Figure 1(c) and the lattice ordering T < M < F,

$$
\begin{aligned}
\texttt{Check}(\neg\texttt{Running}) &= (\{\texttt{OFF}\}, \{\}, \{\texttt{IDLE}_1, \texttt{IDLE}_2, \texttt{AC}, \texttt{HEAT}\}) \\
\texttt{Check}(\texttt{Above} \vee \texttt{Below}) &= (\{\texttt{AC}, \texttt{HEAT}\}, \{\texttt{OFF}, \texttt{IDLE}_1\}, \{\texttt{IDLE}_2\}) \\
\texttt{Check}(AX\ \neg\texttt{Heat}) &= (\{\texttt{OFF}, \texttt{AC}\}, \{\}, \{\texttt{IDLE}_1, \texttt{IDLE}_2, \texttt{HEAT}\})
\end{aligned}
$$

Function $\texttt{QUntil}()$ determines the value of EU and AU using a fixpoint algorithm given in Figure 5. The lowest ("most false") value that $A[\varphi U \psi]/E[\varphi U \psi]$ can have in each state s is $\langle\psi\rangle_s$. Thus, QU_0 is equal to $[\psi]$. At each iteration, the algorithm computes EXTerm_{i+1}, equal to $EXQU_i$. If the function is called with the universal quantifier, then it also computes AXTerm_{i+1}, equal to $AXQU_i$. Otherwise, AXTerm_{i+1} is not necessary, and thus we let AXTerm_{i+1} be $[\varphi]$. $AXQU_i$ and $EXQU_i$ are computed by invoking the function $\texttt{doBigOP}()$ and passing it the result of the appropriate pred call. Then, for each state s, the algorithm determines where this state should be by computing dest := $\langle\psi\rangle_s \vee (\langle\varphi\rangle_s \wedge \langle\mathrm{AXTerm}_{i+1}\rangle_s \wedge \langle\mathrm{EXTerm}_{i+1}\rangle_s)$. If dest value is different from the one s had in QU_i, then it has to be moved to the appropriate place in QU_{i+1}. The algorithm proceeds until no further changes to QU_i can be made.

For example, suppose we are computing $E[\neg\texttt{Below}\ U\ \texttt{Heat}]$ for our model in Figure 1(c) under the ordering T < M < F. QU_0 is initialized to $(\{\texttt{HEAT}\}, \{\texttt{AC}\}, \{\texttt{OFF}, \texttt{IDLE}_1, \texttt{IDLE}_2\})$. $\texttt{IDLE}_2$ has HEAT among its successors, so $\langle \text{EXTerm}_1 \rangle_{\texttt{IDLE}_2}$ is T. Thus,

$$\langle \texttt{Heat} \rangle_{\texttt{IDLE}_2} \vee (\langle \neg\texttt{Below} \rangle_{\texttt{IDLE}_2} \wedge \langle \text{EXTerm}_1 \rangle_{\texttt{IDLE}_2} = \text{F} \vee (\text{T} \wedge \text{T}) = \text{T}$$

and so $\texttt{IDLE}_2$ should move to T. Using a similar process, we decide that dest for $\texttt{IDLE}_1$ in QU_1 is M, and that dest for AC and OFF in QU_2 are T and M, respectively. The next iteration does not change QU_2, and thus the algorithm terminates returning $(\{\texttt{HEAT}, \texttt{AC}, \texttt{IDLE}_2\}, \{\texttt{OFF}, \texttt{IDLE}_1\}, \{\})$.

Property	χCTL formulation	Heater Model	AC Model	Combined Model
Prop. 1.	$AG\ EX\, \texttt{IDLE}_1$	F	T	F
Prop. 2.	$A\ [\neg\texttt{Heat}\ U\ \texttt{Below}]$	T	–	T
Prop. 3.	$AG\ AF \neg\texttt{Running}$	F	F	F
Prop. 4.	$AG\ (\texttt{Heat} \leftrightarrow \texttt{Air})$	–	–	F
Prop. 5.	$AG\ (\texttt{Above} \rightarrow \neg\texttt{Heat})$	–	–	M

Table 1. Results of verifying properties of the thermostat system.

The properties of the thermostat system that we identified in Section 2 can be translated into χCTL as described in Table 1. The table also lists the values of these properties in each of the models given in Figure 1. We use "–" to indicate that the result cannot be obtained from this model. For example, the two individual models disagree on the question of reachability of state $\texttt{IDLE}_1$ from every state in the model, whereas the combined model concludes that it is F.

7 Correctness and Termination of χchek

In this section, we analyze running time of χchek and prove its correctness and termination.

7.1 Complexity

Theorem 4 *Procedure* $\texttt{Check}(p)$ *terminates on every* χ*CTL formula* p.

Computation of Until takes the longest time. Each state can change its position in QU_i at most h times, where h is the height of the lattice $(\mathcal{L}, \sqsubseteq)$. Thus, the maximum number of iterations of the loop in QUntil is $|S| \times h$. Each iteration takes the time to compute doBigOP on pred: $O(|\mathcal{L}| \times 2^{|\mathcal{L}|} \times |S| + |\mathcal{L}|^2 \times |S|^2)$, plus the time to compute toMove and dest sets: $|\mathcal{L}|^4 \times O(|S|)$. Therefore, the running time of QUntil is

$$O(2^{|\mathcal{L}|} \times |S|^2 \times |S| \times h) = O(2^{|\mathcal{L}|} \times |S|^3)$$

and the running time of the entire model-checking algorithm on a property p is

$$O(2^{|\mathcal{L}|} \times |S|^3 \times |p|)$$

Note that in reality the running time is smaller, because BigOPTable can be optimized and because set operations are BDD-based.

Procedure `BooleanCheck`(p)
Case
$p \in A$: **return** $I(p, \top)$
$p = \neg\varphi$: **return** $(S - \varphi)$
$p = \varphi \wedge \psi$: **return** $(\varphi \cap \psi)$
$p = \varphi \vee \psi$: **return** $(\varphi \cup \psi)$
$p = EX\varphi$: **return** $\mathtt{pre}(\varphi)$
$p = AX\varphi$: **return** $(S - \mathtt{pre}(S - \varphi))$
$p = E[\varphi U \psi]$: $Q_0 = \emptyset$
$Q_{i+1} = Q_i \cup (\psi \vee (\varphi \wedge EXQ_i))$
return Q_n when $Q_n = Q_{n+1}$
$p = A[\varphi U \psi]$: $Q_0 = \emptyset$
$Q_{i+1} = Q_i \cup (\psi \vee (\varphi \wedge EXQ_i \wedge AXQ_i))$
return Q_n when $Q_n = Q_{n+1}$

Fig. 6. Boolean Model-Checking Algorithm(adapted from [6]).

7.2 Correctness

In this section we prove correctness of χchek by showing that it always returns exactly one answer (well-foundness) and that this answer is correct, i.e., it preserves the properties of χCTL. We also show that multi-valued model-checking reduces to well-known boolean model-checking [22] if $(\mathcal{L}, \sqsubseteq)$ is the two-valued lattice representing classical logic.

We start by determining that procedure `Check`() associates each state s with exactly one logical value for each χCTL property p.

Theorem 5 *The answer returned by procedure* `Check`() *is always well-founded, i.e.*

$$(a)\ \forall p \in P, \forall s \in S, \exists v_i \in \mathcal{L}, \text{s.t. } s \in [\mathtt{Check}(p)]_{v_i} \qquad \text{(Each state in one set)}$$
$$(b)\ \forall p \in P, \forall s \in S, \exists v_i, v_j \in \mathcal{L}, \text{s.t.}$$
$$(s \in [\mathtt{Check}(p)]_{v_i} \ \wedge\ s \in [\mathtt{Check}(p)]_{v_j}) \rightarrow v_i = v_j \ \text{(Each state only in one set)}$$

Now we show that our algorithm preserves the expected properties of χCTL formulas given in Figure 3.

Theorem 6 *χchek preserves the negation of "next" property, i.e.*

$$\forall s \in S, s \in [\mathtt{Check}(AX\varphi)]_v \ \Leftrightarrow\ s \in [\mathtt{Check}(EX\neg\varphi)]_{\neg v}$$

Theorem 7 *χchek preserves fixpoint properties of AU and EU, i.e.*

$$(1)\ \forall s \in S, \langle \mathtt{Check}(A[\varphi U \psi]) \rangle_s = \langle \mathtt{Check}(\psi) \rangle_s \vee (\langle \mathtt{Check}(\varphi) \rangle_s \wedge \langle \mathtt{Check}(AXA[\varphi U \psi]) \rangle_s \wedge \langle \mathtt{Check}(EXA[\varphi U \psi]) \rangle_s)$$
$$(2)\ \forall s \in S, \langle \mathtt{Check}(E[\varphi U \psi]) \rangle_s = \langle \mathtt{Check}(\psi) \rangle_s \vee (\langle \mathtt{Check}(\varphi) \rangle_s \wedge \langle \mathtt{Check}(EXE[\varphi U \psi]) \rangle_s)$$

($\perp$ "until") follows easily from AU and EU fixpoints.

Our last correctness criterium is that the answers given by χchek on (2-Bool, $\sqsubseteq$), a two-valued boolean lattice representing classical logic, are the same as given by a regular symbolic model-checker. We start by defining a "boolean symbolic model-checker" on Kripke structures, following [6] and changing some notation to make it closer to the one used in this paper. In particular, labeling functions used in boolean model-checking typically map a formula into a set of states where it is true, with the assumption that it is false in all other states. Thus, φ maps into $[\varphi]_\top$ in our notation. The algorithm is given in Figure 6, with pre defined as follows:

$$\texttt{pre}(Q) \equiv \{s \mid t \in Q \;\wedge\; (s,t) \in R\}$$

That is, $\texttt{pre}(Q)$ computes all the states that can reach elements in Q in one step.

Theorem 8 *χchek, called on (2-Bool, $\sqsubseteq$), returns the same answers as a boolean model-checker. More precisely, $\forall s \in S$, $\forall p \in P$,*

$$(1)\ s \in [\texttt{Check}(p)]_\top \Rightarrow s \in \texttt{BooleanCheck}(p)$$
$$(2)\ s \in [\texttt{Check}(p)]_\bot \Rightarrow s \notin \texttt{BooleanCheck}(p)$$

8 Conclusion and Future Work

Multi-valued logics are a useful tool for describing models that contain incomplete information or inconsistency. In this paper we presented an extension of classical CTL model-checking to reasoning about arbitrary quasi-boolean logics. We also described an implementation of a symbolic multi-valued model-checker χchek and proved its termination and correctness.

We plan to extend the work presented here in a number of directions to ensure that χchek can effectively reason about non-trivial systems. We will start by addressing some of the limitations of our χKripke structures. In particular, so far we have assumed that our variables are of the same type, with elements described by values of the lattice associated with that machine. We need to generalize this approach to variables of different types.

Further, in this work we have only addressed single-processor models. We believe that synchronous systems can be easily handled by our framework, and it is essential to extend our model-checking engine to reasoning about synchronous as well as asynchronous systems.

We are also in the process of defining and studying a number of optimizations for storage and retrieval of logic tables. These optimizations and the use of the χDD library do not change the worst-case running-time of χchek, computed in Section 7. However, they significantly affect average-case running time. Once the implementation of the model-checker is complete, we intend to conduct a series of case studies to ensure that it scales up to reasoning about non-trivial systems.

Finally, we are interested in studying the properties of χchek in the overall framework of χbel. This framework involves reasoning about multiple inconsistent descriptions of a system. We are interested in characterizing the relationship between the types of merge of individual descriptions and the interpretation of answers given by χchek on the merged model.

Acknowledgments

We thank members of University of Toronto formal methods reading group, and in particular Ric Hehner, Albert Lai, Benet Devereux and Christopher Thompson-Walsh for numerous interesting and useful discussions and for careful readings of earlier drafts of this paper. We are also indebted to Albert and Benet for the proof of Theorem 2.

We gratefully acknowledge the financial support provided by NSERC and CITO.

References

1. A.R. Anderson and N.D. Belnap. *Entailment. Vol. 1*. Princeton University Press, 1975.
2. R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Approach*. Springer-Verlag, 1998.
3. N.D. Belnap. "A Useful Four-Valued Logic". In Dunn and Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 30–56. Reidel, 1977.
4. L. Bolc and P. Borowik. *Many-Valued Logics*. Springer-Verlag, 1992.
5. R. E. Bryant. "Symbolic Boolean manipulation with ordered binary-decision diagrams". *Computing Surveys*, 24(3):293–318, September 1992.
6. T. Bultan, R. Gerber, and C. League. "Composite Model Checking: Verification with Type-Specific Symbolic Representations". *ACM Transactions on Software Engineering and Methodology*, 9(1):3–50, January 2000.
7. T. Bultan, R. Gerber, and W. Pugh. "Symbolic Model Checking of Infinite State Programs Using Presburger Arithmetic". In *Proceedings of International Conference on Computer-Aided Verification*, Haifa, Israel, 1997.
8. M. Chechik. "On Interpreting Results of Model-Checking with Abstraction". (submitted for publication), May 2000.
9. E.M. Clarke, E.A. Emerson, and A.P. Sistla. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications". *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.
10. B. Devereux. Multi-valued decision diagrams (χdds). Technical report, University of Toronto, Department of Computer Science, August 2000.
11. E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer, 1990.
12. David L. Dill. "The Murϕ Verification System". In R. Alur and T.A. Henzinger, editors, *Computer-Aided Verification Computer*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393, New York, N.Y., 1996. Springer-Verlag.
13. J.M. Dunn. "A Comparative Study of Various Model-Theoretic Treatments of Negation: A History of Formal Negation". In Dov Gabbay and Heinrich Wansing, editors, *What is Negation*. Kluwer Academic Publishers, 1999.
14. S. Easterbrook and M. Chechik. "A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints". (submitted for publication), August 2000.
15. C. Ghezzi and B. A. Nuseibeh. "Introduction to the Special Issue on Managing Inconsistency in Software Development". *IEEE Transactions on Software Engineering*, 24(11):906–1001, November 1998.
16. S. Hazelhurst. *Compositional Model Checking of Partially Ordered State Spaces*. PhD thesis, Department of Computer Science, University of British Columbia, 1996.
17. S. Hazelhurst. "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, March 1999.

18. G.J. Holzmann. “The Model Checker SPIN”. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.
19. A. Hunter. “Paraconsistent Logics”. In D. Gabbay and Ph. Smets, editors, *Handbook of Defeasible Reasoning and Uncertain Information*, volume 2. Kluwer, 1998.
20. A. Hunter and B. Nuseibeh. “Managing Inconsistent Specifications: Reasoning, Analysis and Action”. *ACM Transactions on Software Engineering and Methodology*, 7(4):335–367, October 1998.
21. S. C. Kleene. *Introduction to Metamathematics*. New York: Van Nostrand, 1952.
22. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic, 1993.
23. T.J. Menzies, S. Easterbrook, B. Nuseibeh, and S. Waugh. “An Empirical Investigation of Multiple Viewpoint Reasoning in Requirements Engineering”. In *Proceedings of the Fourth International Symposium on Requirements Engineering (RE'99)*, Limerick, Ireland, June 7-11 1999. IEEE Computer Society Press.
24. Graham Priest and Koji Tanaka. “Paraconsistent Logic”. In *The Stanford Encyclopedia of Philosophy*. Stanford University, 1996.
25. H. Rasiowa. *An Algebraic Approach to Non-Classical Logics. Studies in Logic and the Foundations of Mathematics*. Amsterdam: North-Holland, 1978.

A Appendix

In this appendix we give proofs for the theorems appearing in the main body of the paper. The proofs follow the calculational style of [11]. Section A.1 presents proofs of theorems of lattice theory; Section A.2 gives proofs of correctness of the definition of χCTL operators; Section A.3 lists properties of logic tables computed for χchek; finally, Section A.4 uses the properties given in Section A.3 to prove correctness and termination of the implementation of χchek.

A.1 Lattice Theory

Lattices have a number of properties that hold for them. We list several of them here, without proof.

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

The following are the properties of the product of two lattices $(\mathcal{L}_1, \sqsubseteq)$ and $(\mathcal{L}_2, \sqsubseteq)$:

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

Theorem 1. *A product of two quasi-boolean lattices is quasi-boolean, that is,*

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

Proof:

(1) $\neg\neg(a,b)$
$\Leftrightarrow$ ($\neg$ of pairs)
$\neg(\neg a,\neg b)$
$\Leftrightarrow$ ($\neg$ of pairs), ($\neg$ involution)
(a,b)

(2) $\neg((a_1,b_1)\sqcap(a_2,b_2))$
$\Leftrightarrow$ ($\sqcap$ of pairs)
$\neg((a_1\sqcap a_2),(b_1\sqcap b_2))$
$\Leftrightarrow$ ($\neg$ of pairs)
$(\neg(a_1\sqcap a_2),\neg(b_1\sqcap b_2))$
$\Leftrightarrow$ (de Morgan)
$(\neg a_1\sqcup\neg a_2,\neg b_1\sqcup\neg b_2)$
$\Leftrightarrow$ ($\sqcup$ of pairs)
$(\neg a_1,\neg b_1)\sqcup(\neg a_2,\neg b_2)$

The proof for (3) is similar. □

Theorem 2. *Let $(\mathcal{L},\sqsubseteq)$ be a horizontally-symmetric lattice. Then the following hold for any two elements a, $b\in\mathcal{L}$:*

$$H(a\sqcap b) = H(a)\sqcup H(b)$$
$$H(a\sqcup b) = H(a)\sqcap H(b)$$

Proof:
We will prove the first of these equations here. The second one is a dual. We show

$$(1)\ H(a\sqcap b)\sqsubseteq H(a)\sqcup H(b)\ (1)$$
$$(2)\ H(a\sqcap b)\sqsupseteq H(a)\sqcup H(b)\ (2)$$

(1) $H(a\sqcap b)\sqsubseteq H(a)\sqcup H(b)$
$\Leftarrow$ ($\sqsubseteq$ introduction)
$\forall z,\ (H(a\sqcap b)\sqsubseteq H(z))\Leftarrow(H(a)\sqcup H(b)\sqsubseteq H(z))$
$\Leftarrow$ (H is order-embedding)
$\forall z,(z\sqsubseteq a\sqcap b)\Leftarrow(H(a)\sqcup H(b)\sqsubseteq H(z))$
$\Leftarrow$ ($\sqcap$ elimination)
$\forall z,\ (z\sqsubseteq a\ \wedge\ z\sqsubseteq b)\Leftarrow(H(a)\sqcup H(b)\sqsubseteq H(z))$
$\Leftarrow$ (H is order-embedding)
$\forall z,\ (H(a)\sqsubseteq H(z)\ \wedge\ H(b)\sqsubseteq H(z))\Leftarrow(H(a)\sqcup H(b)\sqsubseteq H(z))$
$\Leftarrow$ (rewriting $\Leftarrow$), ($\sqcup$ introduction)
$\forall x,\ ((H(a)\sqcup H(b)\sqsubseteq H(z))\ \wedge\ (H(a)\sqsubseteq H(a)\sqcup H(b))\ \wedge\ (H(b)\sqsubseteq H(a)\sqcup H(b))$
$\Leftarrow$ (transitivity)
$\top$

(2) $H(a)\sqcup H(b)\sqsubseteq H(a\sqcap b)$
$\Leftarrow$ ($\sqcup$ elimination)
$H(a)\sqsubseteq H(a\sqcap b)\ \wedge\ H(b)\sqsubseteq H(a\sqcap b)$
$\Leftarrow$ (H is order-embedding)
$a\sqcap b\sqsubseteq a\ \wedge\ a\sqcap b\sqsubseteq b$
$\Leftarrow$ ($\sqcap$ elimination)
$\top$

□

A.2 χCTL

Theorem 3. *Definitions of* $\langle EX\varphi\rangle_s$ *and* $\langle AX\varphi\rangle_s$ *preserve the negation of "next" property, i.e.*

$$\forall s \in S,\ \neg\langle AX\varphi\rangle_s = \langle EX\neg\varphi\rangle_s$$

Proof:
Let $s \in S$ be an arbitrary state. Then,

$$
\begin{array}{cl}
 & \neg\langle AX\varphi\rangle_s \\
= & \text{(definition of } AX) \\
 & \neg(\bigwedge_{t\in \mathrm{succ}(s)}(\langle s,t\rangle \to \langle\varphi\rangle_t)) \\
= & \text{(de Morgan), because } \neg \text{ is a quasi-boolean operator} \\
 & \bigvee_{t\in \mathrm{succ}(s)} \neg(\langle s,t\rangle \to \langle\varphi\rangle_t) \\
= & \text{(definition of } \to\text{), (de Morgan)} \\
 & \bigvee_{t\in \mathrm{succ}(s)}(\neg\neg\langle s,t\rangle \wedge \neg\langle\varphi\rangle_t) \\
= & (\neg \text{ involution), because } \neg \text{ is a quasi-boolean operator} \\
 & \bigvee_{t\in \mathrm{succ}(s)}(\langle s,t\rangle \wedge \neg\langle\varphi\rangle_t) \\
= & \text{(definition of } \langle\neg\varphi\rangle_s) \\
 & \bigvee_{t\in \mathrm{succ}(s)}(\langle s,t\rangle \wedge \langle\neg\varphi\rangle_t) \\
= & \text{(definition of } EX) \\
 & \langle EX\neg\varphi\rangle_s
\end{array}
$$

□

A.3 Table Library

Here we give properties of inverse and BigOP tables defined in Section 6.1.

Lemma 1. *The following are properties of inverse tables, with* $op \in \{\wedge, \vee, \to\}$*:*

$$
\begin{array}{ll}
\forall v \in \mathcal{L},\ (a,b) \in \mathtt{InvTable}_{\to,v} \Leftrightarrow (a,\neg b) \in \mathtt{InvTable}_{\wedge,\neg v} & (\to \text{ of } \mathtt{InvTable}) \\
\forall v \in \mathcal{L},\ (a,b) \in \mathtt{InvTable}_{\wedge,v} \Leftrightarrow (\neg a,\neg b) \in \mathtt{InvTable}_{\vee,\neg v} & (\text{de Morgan of } \mathtt{InvTable}) \\
\forall v \in \mathcal{L}, \mathtt{InvTable}_{op,v} \neq \emptyset & (\text{non} - \text{emptiness of } \mathtt{InvTable}) \\
\forall v_1, v_2 \in \mathcal{L}, \exists v_3 \in \mathcal{L}, \text{s.t. } (v_1,v_2) \in \mathtt{InvTable}_{op,v_3} & (\text{completeness of } \mathtt{InvTable}) \\
\forall v_1, v_2, v_3, v_4 \in \mathcal{L},\ ((v_1,v_2) \in \mathtt{InvTable}_{op,v_3} \wedge & \\
\quad (v_1,v_2) \in \mathtt{InvTable}_{op,v_4}) \to (v_3 = v_4) & (\text{uniqueness of } \mathtt{InvTable})
\end{array}
$$

Proof:
From the definition of inverse tables, negation properties, the definition of $\to$ and lattice properties. □

Lemma 2. *The following are the properties of BigOP tables, with* $op \in \{\wedge, \vee\}$*:*

$$
\begin{array}{lll}
\forall v \in \mathcal{L}, & (\emptyset \in \mathtt{BigOPTable}_{\vee,v} \Leftrightarrow \mathcal{L} \in \mathtt{BigOPTable}_{\wedge,\neg v}) \wedge & \\
 & (\forall V \in \mathcal{P}(\mathcal{L}) - \emptyset,\ V \in \mathtt{BigOPTable}_{\vee,v} \Leftrightarrow & \\
 & \qquad \{\neg v \mid v \in V\} \in \mathtt{BigOPTable}_{\wedge,\neg v}) & (\text{negation of } \mathtt{BigOPTable}) \\
\forall v \in \mathcal{L}, & (\emptyset \in \mathtt{BigOPTable}_{\wedge,v} \Leftrightarrow \mathcal{L} \in \mathtt{BigOPTable}_{\vee,\neg v}) \wedge & \\
 & (\forall V \in \mathcal{P}(\mathcal{L}) - \emptyset,\ V \in \mathtt{BigOPTable}_{\wedge,v} \Leftrightarrow & \\
 & \qquad \{\neg v \mid v \in V\} \in \mathtt{BigOPTable}_{\vee,\neg v}) & (\text{negation of } \mathtt{BigOPTable})
\end{array}
$$

$$\forall v \in \mathcal{L}, \quad \texttt{BigOPTable}_{op,v} \neq \emptyset \qquad \text{(non-emptiness of BigOPTable)}$$
$$\forall V \in \mathcal{P}(\mathcal{L}), \exists v \in \mathcal{L}, \text{s.t. } V \in \texttt{BigOPTable}_{op,v} \qquad \text{(completeness of BigOPTable)}$$
$$\forall V \in \mathcal{P}(\mathcal{L}), \forall v_1, v_2 \in \mathcal{L},\ (V \in \texttt{BigOPTable}_{op,v_1} \wedge V \in \texttt{BigOPTable}_{op,v_2}) \rightarrow (v_1 = v_2) \qquad \text{(uniqueness of BigOPTable)}$$

Proof:
By construction of BigOP tables and by the idempotency property of lattices. □

Lemma 3. *The following are the properties of predecessor relations:*

$$\forall \varphi, \forall s \in S, \exists v \in \mathcal{L}, \text{s.t. } s \in [\texttt{pred}([\varphi], op)]_v \ \text{(completeness of pred)}$$
$$[\texttt{pred}([\varphi], \wedge)]_v = [\texttt{pred}([\neg\varphi], \rightarrow)]_{\neg v} \qquad \text{(implication of pred)}$$

Proof:
(completeness of $\texttt{pred}$)
Pick φ, pick a state s, pick $t \in \mathrm{succ}(s)$. Then, let $v_1 = \langle s, t \rangle$ and $v_2 = \langle \varphi \rangle_t$. Further, let $v = v_1\ op\ v_2$. Then, by the completeness of $\texttt{InvTable}$ property, $s \in [\texttt{pred}([\varphi], op)]_v$.

(implication of $\texttt{pred}$)
Let $s \in S$ be an arbitrary state. Then,

$s \in [\texttt{pred}([\varphi], \wedge]_v)$
$\Leftrightarrow$ (definition of $\texttt{pred}$)
$\exists t \in S, \exists (v_1, v_2) \in \texttt{InvTable}_{\wedge,v}, \text{s.t. } (s,t) \in R(v_1) \wedge (\langle \varphi \rangle_t = v_2)$
$\Leftrightarrow$ ($\rightarrow$ of $\texttt{InvTable}$)
$\exists t \in S, \exists (v_1, \neg v_2) \in \texttt{InvTable}_{\rightarrow,\neg v}, \text{s.t. } (s,t) \in R(v_1) \wedge (\langle \varphi \rangle_t = \neg v_2)$
$\Leftrightarrow$ (definition of $\texttt{pred}$)
$s \in [\texttt{pred}([\neg\varphi], \rightarrow]_{\neg v})$

□

A.4 Correctness and Termination

Theorem 4. *Procedure* $\texttt{Check}(p)$ *terminates on every* χ*CTL formula* p.

Proof:
Proof is on the structure of property p. Obviously, for all operators except "until", $\texttt{Check}(p)$ terminates. We give proof for computing AU here. To prove that the execution of $\texttt{QUntil}$ terminates, it suffices to show that $\forall s \in S, \forall i, \langle QU_i \rangle_s \sqsubseteq \langle QU_{i+1} \rangle_s$. Then, QU_i can change value at most h times, where h is the height of lattice $(\mathcal{L}, \sqsubseteq)$.

The proof goes by induction on i. Pick $s \in S$. Then,

Base case: $\langle QU_0 \rangle_s$
$=$ (definition of $\texttt{QUntil}$)
$\langle \psi \rangle_s$
$\sqsubseteq$ (monotonicity of $\wedge$ (since it is $\sqcap$) and $\vee$ (since it is $\sqcup$))
$\langle \psi \rangle_s \vee (\langle \varphi \rangle_s \wedge \langle \mathrm{AXTerm}_1 \rangle_s \wedge \langle \mathrm{EXTerm}_1 \rangle_s)$
$=$ (definition of $\texttt{QUntil}$)
$\langle QU_1 \rangle_s$

IH: $\langle QU_i \rangle_s \sqsubseteq \langle QU_{i+1} \rangle_s$

Prove: $\langle QU_{i+1}\rangle_s \sqsubseteq \langle QU_{i+2}\rangle_s$

Proof: $\langle QU_{i+1}\rangle_s$

$=$ (definition of `QUntil`)

$\langle\psi\rangle_s \vee (\langle\varphi\rangle_s \wedge \langle \mathrm{AXTerm}_{i+1}\rangle_s \wedge \langle \mathrm{EXTerm}_{i+1}\rangle_s)$

$=$ (definition of `QUntil`)

$\langle\psi\rangle_s \vee (\langle\varphi\rangle_s \wedge \bigwedge_{t\in\mathtt{succ}(s)}(\neg\langle s,t\rangle \vee \langle QU_i\rangle_s) \wedge \bigvee_{t\in\mathtt{succ}(s)}(\langle s,t\rangle \wedge \langle QU_i\rangle_s))$

$\sqsubseteq$ (IH), (monotonicity)

$\langle\psi\rangle_s \vee (\langle\varphi\rangle_s \wedge \bigwedge_{t\in\mathtt{succ}(s)}(\neg\langle s,t\rangle \vee \langle QU_{i+1}\rangle_s) \wedge \bigvee_{t\in\mathtt{succ}(s)}(\langle s,t\rangle \wedge \langle QU_{i+1}\rangle_s))$

$=$ (definition of `QUntil`)

$\langle\psi\rangle_s \vee (\langle\varphi\rangle_s \wedge \langle \mathrm{AXTerm}_{i+2}\rangle_s \wedge \langle \mathrm{EXTerm}_{i+2}\rangle_s)$

$=$ (definition of `QUntil`)

$\langle QU_{i+2}\rangle_s$

□

Theorem 5. *The answer returned by function* `Check` *is always well-founded, i.e.*

$$(a)\ \forall p \in P, \forall s \in S, \exists v_i \in \mathcal{L}, \text{s.t. } s \in [\mathtt{Check}(p)]_{v_i} \qquad \text{(Each state in one set)}$$

$$(b)\ \forall p \in P, \forall s \in S, \exists v_i, v_j \in \mathcal{L}, \text{s.t.}$$
$$(s \in [\mathtt{Check}(p)]_{v_i}\ \wedge\ s \in [\mathtt{Check}(p)]_{v_j}) \rightarrow v_i = v_j \quad \text{(Each state only in one set)}$$

Proof:
The proof is by induction on the length of p.

Base case: $p \in A$. $\mathtt{Check}(p)$ uses I which is guaranteed to return a partition by definition.
IH: Assume $\mathtt{Check}(p)$ returns a partition when $|p| \leq n$.
Prove: $\mathtt{Check}(p)$ returns a partition when $|p| = n+1$.
Proof:

- $p = \neg\varphi$ Then, $[\varphi]$ is a partition by IH, and $\neg v$ is onto by $\neg$ involution.
- $p = \varphi \wedge \psi$ Pick state $s \in S$. Since $[\varphi]$ and $[\psi]$ are partitions, $\exists v_1, v_2 \in \mathcal{L}$ s.t. $s \in [\varphi]_{v_1}$ and $s \in [\psi]_{v_2}$.
  (a) By completeness of `InvTable`, $\exists v_3, \text{s.t. } (v_1, v_2) \in \mathtt{InvTable}_{\wedge, v_3}$, so $s \in [p]_{v_3}$.
  (b) By uniqueness of `InvTable`.
- $p = \varphi \vee \psi$ The proof is similar to the one above.
- $p = EX\varphi$. Pick a state $s \in S$.
  Create a set $V = \{v \mid s \in [\mathrm{pred}([\varphi], \wedge)]_v\}$
  (a) By completeness of `BigOPTable`, $\exists v_i \in \mathcal{L}, \text{s.t. } s \in [\mathrm{doBigOP}(\vee, \mathtt{pred}([\varphi], \wedge))]_{v_i}$.
  (b) By uniqueness of `BigOPTable`, the above-found v_i is unique.
- $p = AX\varphi$. The proof is similar to the one above.
- $p = E[\varphi U \psi]$ Partitionness is maintained as an invariant of `QUntil`:
  `QUntil` starts of with a partition, and `move` preserves partition.
- $p = A[\varphi U \psi]$ Same as above.

□

Theorem 6. *χchek preserves the negation of "next" property, i.e.*

$$\forall s \in S, s \in [\mathtt{Check}(AX\varphi)]_v \ \Leftrightarrow\ s \in [\mathtt{Check}(EX\neg\varphi)]_{\neg v}$$

Proof:
We prove

$$(1)\ s \in [\texttt{Check}(AX\varphi)]_v \quad \Rightarrow s \in [\texttt{Check}(EX\neg\varphi)]_{\neg v}$$
$$(2)\ s \in [\texttt{Check}(EX\neg\varphi)]_{\neg v} \Rightarrow s \in [\texttt{Check}(AX\varphi)]_v$$

(1) $s \in [\texttt{Check}(AX\varphi)]_v$
$\Rightarrow$ (definition of $\texttt{Check}$)
$s \in [\texttt{doBigOP}(\wedge, \texttt{pred}([\varphi], \rightarrow))]_v$
$\Rightarrow$ (definition of $\texttt{doBigOP}$)
$\exists V \in \texttt{BigOPTable}_{\wedge,v}, \mathrm{s.t.}\ (s \in \bigcap_{v_i \in V}[\texttt{pred}([\varphi], \rightarrow)]_{v_i}) \wedge (s \notin \bigcup_{v_i \in (\mathcal{L}-V)}[\texttt{pred}([\varphi], \rightarrow)]_{v_i})$
$\Rightarrow$ (negation of $\texttt{BigOPTable}$)
$\exists V_1 \in \texttt{BigOPTable}_{\vee,\neg v}, \mathrm{s.t.}\ (s \in \bigcap_{\neg v_i \in V_1}[\texttt{pred}([\varphi], \rightarrow)]_{v_i}) \wedge (s \notin \bigcup_{\neg v_i \in (\mathcal{L}-V_1)}[\texttt{pred}([\varphi], \rightarrow)]_{v_i})$
$\Rightarrow$ (implication of $\texttt{pred}$)
$\exists V_1 \in \texttt{BigOPTable}_{\vee,\neg v}, \mathrm{s.t.}\ (s \in \bigcap_{\neg v_i \in V_1}[\texttt{pred}([\neg\varphi], \wedge)]_{\neg v_i}) \wedge (s \notin \bigcup_{\neg v_i \in (\mathcal{L}-V_1)}[\texttt{pred}([\neg\varphi], \wedge)]_{\neg v_i})$
$\Rightarrow$ (definition of $\texttt{doBigOP}$)
$s \in [\texttt{doBigOP}(\vee, \texttt{pred}([\neg\varphi], \wedge))]_{\neg v_i}$
$\Rightarrow$ (definition of $\texttt{Check}$)
$s \in [\texttt{Check}(EX[\neg\varphi])]_{\neg v}$

Proof of (2) is similar, and is based on implication of $\texttt{pred}$ and negation of $\texttt{BigOPTable}$. □

Theorem 7. *Xchek preserves fixpoint properties of AU and EU, i.e.*

$$(1)\ \forall s \in S, \langle \texttt{Check}(A[\varphi U \psi]) \rangle_s = \langle \texttt{Check}(\psi) \rangle_s \vee (\langle \texttt{Check}(\varphi) \rangle_s \wedge \langle \texttt{Check}(AXA[\varphi U \psi]) \rangle_s \wedge \langle \texttt{Check}(EXA[\varphi U \psi]) \rangle_s)$$
$$(2)\ \forall s \in S, \langle \texttt{Check}(E[\varphi U \psi]) \rangle_s = \langle \texttt{Check}(\psi) \rangle_s \vee (\langle \texttt{Check}(\varphi) \rangle_s \wedge \langle \texttt{Check}(EXE[\varphi U \psi]) \rangle_s)$$

Proof:
Pick a state s.

(1) $s \in [\texttt{Check}(A[\varphi U \psi])]_v$
$\Leftrightarrow$ (definition of $\texttt{Check}$)
$s \in [\texttt{QUntil}(A, [\varphi], [\psi])]_v$
$\Leftrightarrow$ (definition of $\texttt{QUntil}$)
$\exists n > 0$, s.t., $QU_{n+1} = QU_n$
$\wedge\ s \in [QU_{n+1}]_v \Leftrightarrow (v = \langle \texttt{Check}(\psi) \rangle_s \vee (\langle \texttt{Check}(\varphi) \rangle_s \wedge \langle \mathrm{AXTerm}_{n+1} \rangle_s \wedge \langle \mathrm{EXTerm}_{n+1} \rangle_s))$
$\Leftrightarrow$ (definition of AXTerm), (definition of EXTerm), (definition of AX in $\texttt{Check}$)
$\exists n > 0$, s.t., $QU_{n+1} = QU_n$
$\wedge\ s \in [QU_{n+1}]_v \Leftrightarrow (v = \langle \texttt{Check}(\psi) \rangle_s \vee (\langle \texttt{Check}(\varphi) \rangle_s \wedge \langle \texttt{Check}(AXQU_n) \rangle_s$
$\wedge \langle \texttt{Check}(EXQU_n) \rangle_s))$
$\Leftrightarrow$ (combining the two conjuncts)
$s \in [QU_n]_v \Leftrightarrow (v = \langle \texttt{Check}(\psi) \rangle_s \vee (\langle \texttt{Check}(\varphi) \rangle_s \wedge \langle \texttt{Check}(AXQU_n) \rangle_s \wedge \langle \texttt{Check}(EXQU_n) \rangle_s))$
$\Leftrightarrow$ $(A[\varphi U \psi] = QU_n)$
$\langle \texttt{Check}(A[\varphi U \psi]) \rangle_s = \langle \texttt{Check}(\psi) \rangle_s \vee$
$(\langle \texttt{Check}(\varphi) \rangle_s \wedge \langle \texttt{Check}(AXA[\varphi U \psi]) \rangle_s \wedge \langle \texttt{Check}(EXE[\varphi U \psi]) \rangle_s)$

Proof of (2) is similar. □

In our last theorem we want to prove that the result of calling χchek with (2-Bool, $\sqsubseteq$), a lattice representing classical logic, is the same as the result of the boolean CTL model-checker.

We start by defining inverse and BigOP tables for a boolean lattice:

$$\begin{array}{ll}
\mathtt{InvTable}_{\wedge,\top} = \{(\top,\top)\} & \mathtt{BigOPTable}_{\wedge,\top} = \{\{\top\},\emptyset\} \\
\mathtt{InvTable}_{\wedge,\bot} = \{(\top,\bot),(\bot,\bot),(\bot,\top)\} & \mathtt{BigOPTable}_{\wedge,\bot} = \{\{\bot\},\{\bot,\top\}\} \\
\mathtt{InvTable}_{\vee,\top} = \{(\top,\bot),(\top,\top),(\bot,\top)\} & \mathtt{BigOPTable}_{\vee,\top} = \{\{\top\},\{\top,\bot\}\} \\
\mathtt{InvTable}_{\vee,\bot} = \{(\bot,\bot)\} & \mathtt{BigOPTable}_{\vee,\bot} = \{\{\bot\},\emptyset\}
\end{array}$$

Lemma 4. *The following relations hold for each $s \in S$ when multi-valued model-checking is called on (2-Bool, $\sqsubseteq$):*

$$\begin{array}{ll}
\forall s \in S, s \in [\mathtt{pred}([\varphi],\wedge)]_\top \Rightarrow s \in \mathtt{pre}(\varphi) & (\text{property of } [\mathtt{pred}([\varphi],\wedge)]_\top) \\
\forall s \in S, s \in [\mathtt{pred}([\varphi],\rightarrow)]_\top \Rightarrow s \in \neg\mathtt{pre}(\neg\varphi) & (\text{property of } [\mathtt{pred}([\varphi],\rightarrow)]_\top) \\
\forall s \in S, s \in [\mathtt{pred}([\varphi],\wedge)]_\bot \Rightarrow s \in \neg\mathtt{pre}(\varphi) & (\text{property of } [\mathtt{pred}([\varphi],\wedge)]_\bot) \\
\forall s \in S, s \in [\mathtt{pred}([\varphi],\rightarrow)]_\bot \Rightarrow s \in \mathtt{pre}(\neg\varphi) & (\text{property of } [\mathtt{pred}([\varphi],\rightarrow)]_\bot)
\end{array}$$

Proof:
We prove properties of $[\mathtt{pred}([\varphi],\wedge)]_\top$ and $[\mathtt{pred}([\varphi],\rightarrow)]_\top$. The others follow from (implication of $\mathtt{pred}$). For an arbitrary state $s \in S$,

property of $[\mathtt{pred}([\varphi],\wedge)]_\top$:
$s \in [\mathtt{pred}([\varphi],\wedge)]_\top$
$\Rightarrow$ (definition of $\mathtt{pred}$)
$\exists t \in S, \exists (v_1,v_2) \in \mathtt{InvTable}_{\wedge,\top}, \text{s.t. } (s,t) \in R(v_1) \wedge \langle\varphi\rangle_t = v_2$
$\Rightarrow$ (value of $\mathtt{InvTable}_{\wedge,\top}$)
$\exists t \in S, \text{ s.t. } (s,t) \in R \wedge \langle\varphi\rangle_t = \top$
$\Rightarrow$ (definition of $\mathtt{pre}$)
$s \in \mathtt{pre}(\varphi)$
property of $[\mathtt{pred}([\varphi],\rightarrow)]_\top$:
$s \in [\mathtt{pred}([\varphi],\rightarrow)]_\top$
$\Rightarrow$ (definition of $\mathtt{pred}$)
$\exists t \in S, \exists (v_1,v_2) \in \mathtt{InvTable}_{\rightarrow,\top}, \text{s.t. } (s,t) \in R(v_1) \wedge \langle\varphi\rangle_t = v_2$
$\Rightarrow$ (value of $\mathtt{InvTable}_{\vee,\top}$)
$\exists t \in S, \neg(((s,t) \notin R(\top)) \wedge (\langle\neg\varphi\rangle_t = \top))$
$\Rightarrow$ (definition of $\mathtt{pre}$)
$s \notin \mathtt{pre}(\neg\varphi)$

$\square$

Theorem 8. *χchek, called on (2-Bool, $\sqsubseteq$), returns the same answers as a boolean model-checker. More precisely, $\forall s \in S$, $\forall p \in P$,*

$$\begin{array}{l}
(1)\ s \in [\mathtt{Check}(p)]_\top \Rightarrow s \in \mathtt{BooleanCheck}(p) \\
(2)\ s \in [\mathtt{Check}(p)]_\bot \Rightarrow s \notin \mathtt{BooleanCheck}(p)
\end{array}$$

Proof:
The proof is by induction on the structure of property p.

Base Case:

$p \in A$: $\texttt{Check}(p)$ and $\texttt{BooleanCheck}(p)$ give the same answers by definition.

IH: Assume (1) and (2) hold for properties of length $\leq n$.

Prove: (1) and (2) hold for properties of length $n+1$.

Proof:

$p = \neg\varphi$:

(1): $s \in [\texttt{Check}(p)]_\top$
$\Rightarrow$ (definition of $\texttt{Check}$)
$s \in [\varphi]_\bot$
$\Rightarrow$ (definition of $\texttt{Check}$)
$s \in [\texttt{Check}(\varphi)]_\bot$
$\Rightarrow$ (IH)
$s \notin \texttt{BooleanCheck}(\varphi)$
$\Rightarrow$ (definition of $\texttt{BooleanCheck}$)
$s \in \texttt{BooleanCheck}(p)$

Proof for (2) is similar

$p = \varphi \wedge \psi$:

(1): $s \in [\texttt{Check}(p)]_\top$
$\Rightarrow$ (definition of $\texttt{doOp}$)
$s \in [\varphi]_a \ \wedge \ s \in [\psi]_a \ \wedge \ (a,b) \in \texttt{InvTable}_{\wedge,\top}$
$\Rightarrow$ (value of $\texttt{InvTable}_{\wedge,\top}$)
$s \in [\varphi]_\top \ \wedge \ s \in [\psi]_\top$
$\Rightarrow$ (changing notation)
$s \in (\varphi \cap \psi)$
$\Rightarrow$ (definition of $\texttt{BooleanCheck}$)
$s \in \texttt{BooleanCheck}(p)$

Proof for (2) is similar. Because of the value of $\texttt{InvTable}_{\wedge,\bot}$, $s \in [\varphi]_a \ \wedge \ s \in [\psi]_a \ \wedge \ (a,b) \in \texttt{InvTable}_{\wedge,\bot}$ implies that $s \notin [\varphi]_\top \vee s \notin [\psi]_\top$.

$p = \varphi \vee \psi$: Proofs are similar to the ones above and are based on values of $\texttt{InvTable}_{\vee,\top}$ and $\texttt{InvTable}_{\vee,\bot}$.

$p = EX\varphi$:

(1): $s \in [\texttt{Check}(p)]_\top$
$\Rightarrow$ (definition of $\texttt{Check}$)
$s \in [\texttt{doBigOP}(\vee, \texttt{pred}([\varphi], \wedge))]_\top$
$\Rightarrow$ (definition of $\texttt{doBigOP}$), (value of $\texttt{BigOPTable}_{\vee,\top}$)
$$[\text{result}]_\top = \emptyset \cup ([\text{pred}(\varphi,\wedge)]_\top - [\text{pred}(\varphi,\wedge)]_\bot) \cup ([\text{pred}(\varphi,\wedge)]_\top \cap [\text{pred}(\varphi,\wedge)]_\bot)$$
$\Rightarrow$ (properties of $[\texttt{pred}([\varphi],\rightarrow)]_\top$ and $[\texttt{pred}([\varphi],\rightarrow)]_\bot$)
$$[\text{result}]_\top = (\texttt{pre}(\varphi) - (S - \texttt{pre}(\varphi))) \cup (\texttt{pre}(\varphi) \cap (S - \texttt{pre}(\varphi)))$$
= (set theory)
$s \in \texttt{pre}(\varphi)$
$\Rightarrow$ (definition of $\texttt{BooleanCheck}$)
$s \in \texttt{BooleanCheck}(p)$

(2): $s \in [\texttt{Check}(p)]_\bot$
$\Rightarrow$ (definition of $\texttt{Check}$), (definition of $\texttt{doBigOP}$), (value of $\texttt{BigOPTable}_{\vee,\bot}$)
$$[\text{result}]_\bot = \emptyset \cup ([\text{pred}(\varphi,\wedge)]_\bot - [\text{pred}(\varphi,\wedge)]_\top)$$
$\Rightarrow$ (properties of $[\texttt{pred}([\varphi],\rightarrow)]_\top$ and $[\texttt{pred}([\varphi],\rightarrow)]_\bot$), (logic)
$s \notin \texttt{pre}(\varphi)$
$\Rightarrow$ (definition of $\texttt{BooleanCheck}$)
$s \in \texttt{BooleanCheck}(p)$

$p = AX\varphi$: The proof of (1) and (2) is similar to the one above and is based on properties of $[\mathtt{pred}([\varphi], \wedge)]_\top$ and $[\mathtt{pred}([\varphi], \wedge)]_\bot$, values of $\mathtt{BigOPTable}_{\wedge,\top}$ and $\mathtt{BigOPTable}_{\wedge,\bot}$.

$p = A[\varphi U \psi]$ Since $\mathtt{Check}(p)$ expands into computing QU_n in $\mathtt{QUntil}(A, [\varphi], [\psi])$, the proof for (1) goes by induction on n — the length of the path from s to a state where ψ holds.

Base case: $n = 0$.

$QU_1 = QU_0 \;\wedge\; s \in [Q_0]_\top$

$s \in [\psi]_\top$

$\Rightarrow$ (definition of $\mathtt{BooleanCheck}$)

$s \in Q_1$

$\Rightarrow$ (definition of $\mathtt{BooleanCheck}$)

$s \in \mathtt{BooleanCheck}(p)$

IH: Assume (1) holds for all $n \leq k$.

Prove: (1) holds for $n = k + 1$.

Proof: $s \in [\mathtt{Check}(p)]_\bot$

$\Rightarrow$ (definition of $\mathtt{QUntil}$), (definition of EXTerm), (definition of AXTerm)

$(QU_{k+2} = QU_{k+1}) \;\wedge\; (\langle\psi\rangle_s \vee (\langle\varphi\rangle_s \wedge \langle AXQU_{k+1}\rangle_s \wedge \langle EXQU_{k+1}\rangle_s) = \top)$

$\Rightarrow$ (boolean lattice rules)

$(QU_{k+2} = QU_{k+1}) \wedge$
$\quad (\langle\psi\rangle_s = \top \vee (\langle\varphi\rangle_s = \top \;\wedge\; \langle AXQU_{k+1}\rangle_s = \top \;\wedge\; \langle EXQU_{k+1}\rangle_s = \top))$

$\Rightarrow$ (Theorem 8 for $p = AX\varphi$), (Theorem 8 for $p = EX\varphi$), (IH)

$s \in \mathtt{BooleanCheck}(\psi) \vee (s \in \mathtt{BooleanCheck}(\varphi) \wedge$
$\quad s \in \mathtt{BooleanCheck}(AXQ_k) \;\wedge\; s \in \mathtt{BooleanCheck}(EXQ_k))$

$\Rightarrow$ (definition of $\mathtt{BooleanCheck}$)

$s \in \mathtt{BooleanCheck}(p)$

(2)$s \in [\mathtt{Check}(p)]_\bot$

$\Rightarrow$ (definition of $\mathtt{Check}$), (definition of AXTerm), (definition of EXTerm)

$\forall i, \langle\psi\rangle_s \vee (\langle\varphi\rangle_s \wedge \langle AXQU_i\rangle_s \wedge \langle AXQU_i\rangle_s) = \bot$

$\Rightarrow$ (boolean lattice laws)

$\forall i, \langle\psi\rangle_s = \bot \;\wedge\; (\langle\varphi\rangle_s = \bot \vee \langle AXQU_i\rangle_s = \bot \vee \langle EXQU_i\rangle_s = \bot)$

$\Rightarrow$ (Theorem 8 for $p = AX\varphi$), (Theorem 8 for $p = EX\varphi$), (Base Case)

$\forall i,\; s \notin \mathtt{BooleanCheck}(\psi) \;\wedge\; (s \notin \mathtt{BooleanCheck}(\varphi) \vee$
$\quad s \notin \mathtt{BooleanCheck}(AXQ_i) \vee s \notin \mathtt{BooleanCheck}(EXQ_i))$

$\Rightarrow$ (definition of $\mathtt{BooleanCheck}$)

$\forall i, s \notin Q_i \Leftrightarrow s \notin \mathtt{BooleanCheck}$

$p = E[\varphi U \psi]$: The proof of (1) and (2) is similar to the one above.

$\square$