MULTI-VALUED SYMBOLIC MODEL-CHECKING: FAIRNESS, COUNTER-EXAMPLES, RUNNING TIME

by

Arie Gurfinkel

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Multi-Valued Symbolic Model-Checking: Fairness, Counter-Examples, Running Time

Arie Gurfinkel

Master of Science

Graduate Department of Computer Science

University of Toronto

2003

Multi-valued model-checking is an effective technique for reasoning about systems with incomplete or inconsistent information. In particular, it is well suited for reasoning about abstract, partial, and feature-based system descriptions. The technique is based on extending the classical model-checking algorithm over two-valued logic to arbitrary finite logics whose truth values form a distributive De Morgan lattice.

In this thesis we address several issues surrounding the usability of multi-valued model-checking. Firstly, we provide an improved analysis of the worst-case complexity of the symbolic multi-valued model-checking algorithm, and show that it is independent of the height of the lattice. Secondly, we extend the notion of fairness to a multi-valued models, thus enabling application of multi-valued model-checking to asynchronous concurrent systems. Thirdly, we introduce multi-valued witnesses and counter-examples that aid in interpreting the results of the model-checker. Finally, we describe the design and implementation of a multi-valued model-checker χChek.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

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 [CES86], i.e., it returns the value of the predicate $M \models \varphi$. Model-checking has been effectively applied to reasoning about correctness of hardware [CGH$^+$93], communication protocols [Hol91], and software requirements [AG93]. A number of classical model-checkers are currently used for industrial applications, including SPIN [Hol97], SMV [McM93], and Murϕ [Dil96].

Multi-valued model-checking, i.e., reasoning with values other than TRUE and FALSE, is a generalization of classical model-checking [CEP01, CDE$^+$01, CDEG02, CDG02]. 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 [KNPS00].

The domain of logical values used in multi-valued model-checking is given by finite distributive lattices. 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* [Ras78]. 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.

Multi-valued logics are particularly well suited for representing imprecise models, such as abstract, partial or inconsistent models. Their main advantage is that they allow one to separate the task of developing the right representation for the information, which is done by choosing an appropriate lattice, from the task of actual modeling. As such, they simplify both the modeling and the reasoning tasks.

For example, a partial model can be represented using two classical models. One, called the optimistic model, treats all unknown information as TRUE, and one, called the pessimistic model, treats all unknown information as FALSE. In addition, each model contains auxiliary variables, one for each original model variable, to handle negation. This significantly complicates the task of modeling since the user must keep track of two models, and the dependencies between the auxiliary variables. Moreover, to model-checking a partial model represented in such a form requires two queries to a model-checker, one for optimistic model and one for the pessimistic one.

Alternatively, the same partial model can be represented by a single multi-valued model using a 3-valued logic with values TRUE, FALSE, and MAYBE, where the value MAYBE is used to represent the partial (or unknown) part of the system. This simplifies the modeling task, since the fact that the model contains unknown information is represented directly by the logic. Furthermore, this also simplifies the analysis since only a single model is being checked.

The major limitation of the current work on multi-valued model-checking is that it does not address the key usability issues, such as fairness and counter-examples. The goal of this work is to remedy this problem by introducing a multi-valued counterpart of fairness, that is vital for verification of concurrent systems, and a multi-valued counterpart for counter-examples, that is indispensable for interpreting the results of the model-checker.

1.2 Related Work

Multi-valued algebras (often called ”logics”) have been explored for a variety of applications in databases [Gai79], knowledge representation [Gin87], machine learning [Mic77], and circuit design [Haz96]. A number of specific propositional multi-valued algebras have been proposed and studied. For example, Łukasiewicz [Łuk70] first introduced a 3-valued logic to allow for propositions whose truth values are ‘unknown’. and Kleene [Kle52] introduced several alternative 3-valued algebras. Belnap [Bel77] 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 considered in this thesis 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 [Häh94, SS00] has led to the development of several theorem-provers for first-order multi-valued logics. Multi-valued extensions of modal logics have been explored by Fitting [Fit91, Fit92] 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.

Several authors [BG99, BG00, HJS01, GHJ01, KP02] discuss model-checking over 3-valued logics. Bruns and Godefroid [BG99, BG00] 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* [BG00] 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. [HJS01, GHJ01] apply the *modal transition systems* (MTS) of Larsen and Thomsen [LT88] 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 [SRW99]. 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 [HP01] 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 [BCHG+97]. 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) [BC98], which are equivalent to the *Algebraic Decision Diagrams* (ADDs) of Somenzi et al. [BFG+93]. Each non-terminal node of MTBDDs has two children, and the number of terminal nodes depends on the range of the function being represented.

1.3 Contributions of this Thesis

In this work we make several independent contributions to the study of multi-valued model-checking.

Firstly, we improve on the previously known worst-case complexity bounds for symbolic multi-valued model-checking algorithms. We show that in the worst-case, the number of iterations required by the algorithm is independent of the height of the logic used. Thus, it follows that the difference in complexity between multi-valued and classical model-checking is completely determined by the difference in the decision diagrams used by them. This result allows for a more precise analysis of the applicability of multi-valued model-checking techniques in practice. In particular, it gives a promise for the applicability of *temporal logic queries* [GDC02], where the height of the model-checking logic is extremely large.

Secondly, we extend a notion of fairness to multi-valued model-checking, which is crucial for the application of multi-valued model-checking techniques to concurrent systems. We present a formal definition of the multi-valued fairness, and argue for its correctness by establishing that it satisfies several key intuitive properties. Part of this work has previously appeared in [CDEG02].

Thirdly, we introduce the notion of multi-valued witnesses and counter-examples, that are imperative for understanding the reasons behind the result of the model-checker. Our major contribution in this area is the realization that counter-examples and witnesses can be treated uniformly as proofs of correctness of temporal logic statements. Using this fact we take an unconventional approach to witness and counter-example generation by first developing a proof system for our temporal logic χCTL, introducing an automated techniques for generating proofs, and finally showing how witnesses and counter-examples can be extracted from the proofs. The immediate benefit of this approach is that the witness extraction phase can be done interactively via a proof-browser tool, allowing the user to prefer one witness over another. We also develop a prototype implementation of such a proof-browser tool, KegVis, whose application in the case of classical model-checking has been reported in [GC02].

Finally, we present the design and construction of a symbolic multi-valued model-checker χChek that served as the basis for the implementation of the ideas described in this work.

1.4 Organization of this Thesis

In this thesis we address four different facets of multi-valued model-checking. Chapter 2 introduces classical and multi-valued model-checking providing the necessary background for the rest of this work. In Chapter 3, we improve on the previously known worst-case complexity bounds for symbolic multi-valued model-checking. We then extend the multi-valued temporal logic χCTL to handle fairness condition in Chapter 4. In Chapter 5, we present a novel approach for counter-example generation via proof of satisfaction of temporal formulas and apply it to multi-valued setting. Chapter 6 provides a birds-eye view of the design of the symbolic multi-valued model-checker χChek which is used as the basis for the implementation of both fairness and counter-example generation algorithms described in this thesis. Finally, Chapter 7 concludes the thesis, outlines current limitations, and describes our plans for future work.

Chapter 2

Background

2.1 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) [CES86]. 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 $I : S \rightarrow 2^A$. 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 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,

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

where the remaining operators are defined as follows:

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

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 [HR00, CGP99]).

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

Figure 2.1: Fixpoint formulations of CTL operators.

Alternatively, CTL operations can be described using their fixpoint formulations, as shown in Figure 2.1. This description is most useful for symbolic model-checking [McM93].

Throughout the rest of this thesis we use a number of conventions: (1) our proofs follow the *calculational style* [Heh93, BvW98]; (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 [Koz83]; (4) we use nat to refer to the set of natural numbers, and (5) we use $\exists!$ to mean "exists unique".

2.2 Lattice Theory

We give a brief overview of lattice theory, for a more complete presentation see [Bir67].

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:

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

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

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

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

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

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

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

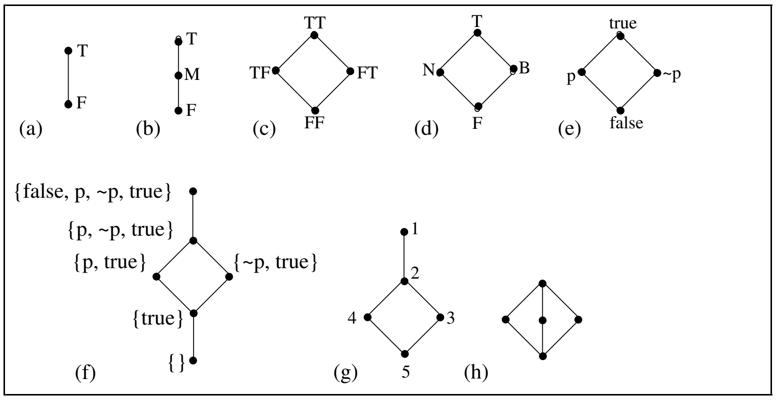

Figure 2.2: Examples of a few lattices.

Definition 2 *A lattice is* distributive *iff*

$$a \sqcup (b \sqcap c) \;=\; (a \sqcup b) \sqcap (a \sqcup c) \quad \text{(distributivity)}$$
$$a \sqcap (b \sqcup c) \;=\; (a \sqcap b) \sqcup (a \sqcap c) \quad \text{(distributivity)}$$

Figure 2.2 gives examples of a few lattices. The lattice in Figure 2.2(h) is non-distributive, whereas all other lattices are distributive.

2.3 Quasi-Boolean Algebras

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

Definition 3 *[Ras78] 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 $\mathcal{L} \to \mathcal{L}$ s.t. every element $a \in \mathcal{L}$ corresponds to a unique element $\neg a \in \mathcal{L}$ satisfying the following conditions:*

$$\neg(a \sqcap b) = \neg a \sqcup \neg b \quad \text{(De Morgan)} \qquad \neg\neg a = a \quad (\neg \text{ involution})$$
$$\neg(a \sqcup b) = \neg a \sqcap \neg b \qquad a \sqsubseteq b \Leftrightarrow \neg a \sqsupseteq \neg b \quad (\neg \text{ antimonotonic})$$

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 [BB92, Dun99]. Note that the negation operator satisfying the above properties is a *lattice dual isomorphism* with period 2 [Bir67].

We now define several quasi-boolean algebras using the lattices in Figure 2.2. The domain of logical values of the classical logic, referred to as $\mathbf{2}$, is the lattice in Figure 2.2(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 2.2(b), where ¬T = F, ¬F = T, ¬M = M. This logic was first defined in [Kle52]. The four-valued logic $\mathbf{2x2}$ is defined on the lattice in Figure 2.2(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 2.2(d) is the Belnap's 4-valued logic which has been used for reasoning about inconsistent databases [Bel77, AB75]. Here, ¬N = N and ¬B = B. The underlying lattice is isomorphic to the one in Figure 2.2(c), but the resulting quasi-boolean algebras are not. The lattice in Figure 2.2(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 2.2(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 [GDC02, BG01].

[1] Given the ordered set $(\mathcal{L}, \sqsubseteq)$ and a subset $B \subseteq \mathcal{L}$, $\uparrow B$ 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 lattice in Figure 2.2(h) cannot be used as a basis for a quasi-boolean algebra because it is non-distributive. The lattice in Figure 2.2(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* $a \in \mathcal{L}$,

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

For example, the algebra **2** is boolean, whereas **3** is not (M $\sqcap \neg$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*

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

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 now introduce a few theorems about quasi-boolean algebras.

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 2.2(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.

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 $a, b \in \mathcal{L}$,*

$$a \sqsubseteq b \;\Leftrightarrow\; H(a) \sqsupseteq H(b) \quad \textit{(order-embedding)}$$
$$H(H(a)) = a \qquad \textit{(H involution)}$$

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 2.2(a)-(f) exhibit horizontal symmetry and thus are quasi-boolean, whereas the lattice in Figure 2.2(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 2.2(d), negation is not defined via horizontal symmetry: ¬N = N, ¬B = B.

Finally, we define implication and equivalence as follows:

$$a \rightarrow b \;\triangleq\; \neg a \sqcup b \qquad \text{(material implication)}$$
$$a \leftrightarrow b \;\triangleq\; (a \rightarrow b) \;\sqcap\; (b \rightarrow a) \qquad \text{(equivalence)}$$

2.4 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

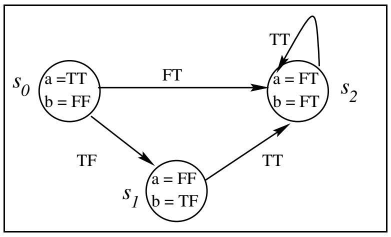

Figure 2.3: Ex1: a simple χKripke structure.

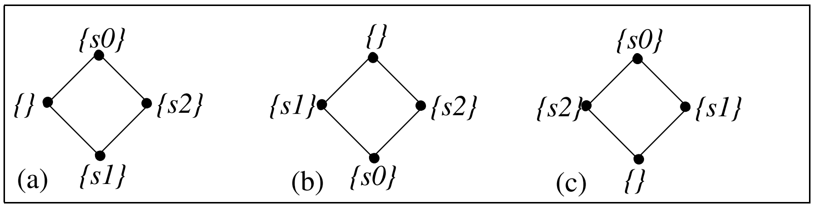

Figure 2.4: Several mv-sets for the example in Figure 2.3: (a) corresponding to variable a; (b) corresponding to variable b; (c) $\overline{[\![b]\!]}$ – a multi-valued complement of the mv-set in (b).

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 $S \to \mathcal{L}$.*

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 2.3. This machine uses the quasi-boolean algebra **2x2** where the logical values form the lattice in Figure 2.2(c), and exemplifies χKripke structures — multi-valued generalizations of Kripke structures, defined formally in Section 2.6. 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 2.3: 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 2.4(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 2.4(a) and (b), respectively, we note that in state s_1 a is FF and b is TF. Thus,

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

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:

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

Mv-set $\overline{[\![b]\!]}$ is given in Figure 2.4(c).

We then obtain the desired properties:

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

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 [Zad87].

2.5 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 2.3, 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 2.5. This relation is referred to as $\mathbb{T}$. For example, the value of the transition (s_0, s_1) is TF, so $\mathbb{T}((s_0, s_1)) = \text{TF}$.

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 backward and the forward images of $[\![a]\!]$ (see Figure 2.4(a)) under the multi-valued relation $\mathbb{T}$ between the pairs of states of the χKripke structure Ex1. These are shown in Figures 2.5(b) and (c), respectively. For example, when we compute backward image of $[\![a]\!]$ at s_0, we get

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

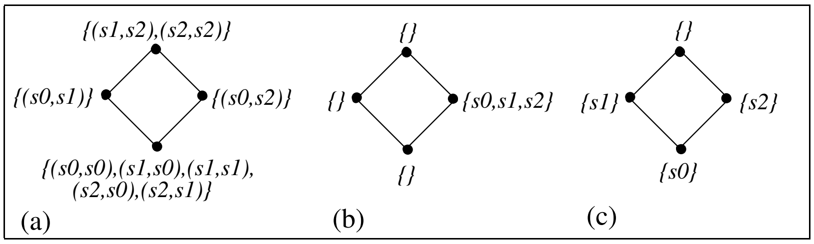

Figure 2.5: (a) The multi-valued relation between pairs of states of Ex1; (b) Backward image of $[\![a]\!]$ over the relation in (a); (c) Forward image of $[\![a]\!]$ over the relation in (a).

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

2.6 Multi-Valued CTL

In this section we extend the notion of boolean model-checking described in Section 2.1 by defining multi-valued Kripke structures, which we call χKripke structures, and multi-valued CTL (χCTL).

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, and not labeling $\top$ transitions. One χKripke structure, shown in Figure 2.3, was introduced in Section 2.4.

Next, 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 2.1.

Definition 10 *A* computation *of a χKripke structure M from a (reachable) state s is an infinite sequence of states $s_0, s_1, ...$ 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.

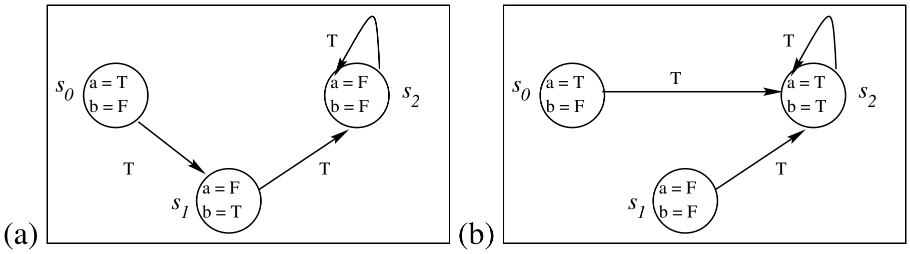

Figure 2.6: Two classical Kripke structures: (a) Ex$_l$; (b) Ex$_r$.

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 2.4. For simplicity of presentation we also extend the notation to mv-sets, and let $[\![\mathbb{Z}]\!] = \mathbb{Z}$.

The semantics is as follows:

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

We proceed by defining the EX operator. Recall from Section 2.1 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 [Bel77, Ras78]. The semantics of EX is:

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

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

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

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 2.6. χKripke structure Ex1, shown in Figure 2.3, 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 2.1, as

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

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

Note that our definitions of EX and AX enjoy some familiar properties of their CTL counterparts. In particular, both EX and AX are monotone, and

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

We further define EG and EU using the EG and EU fixpoint properties in Figure 2.1:

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

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

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

and the remaining χCTL operators are defined as their classical counterparts:

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

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

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

Note that the χCTL operators defined above can also be defined directly, through their fixpoint characterization.

Theorem 2 *Fixpoint properties of (derived) χCTL operators are the same as for CTL operators. That is,*

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

We also introduce a bounded version of EU and EG operators.

$$E[\varphi\ U_i\ \psi] \quad \triangleq \quad \begin{cases} \psi & \text{if } i = 0 \\ \psi \vee \varphi \wedge EXE[\varphi\ U_{i-1}\ \psi] & \text{if } i > 0 \end{cases}$$

$$EG^i\varphi \quad \triangleq \quad \begin{cases} \varphi & \text{if } i = 0 \\ \varphi \wedge EXEG^{i-1}\varphi & \text{if } i > 0 \end{cases}$$

Intuitively, the bound i corresponds to the restriction of the operators to all finite paths of length at most i. Formally, they correspond to the ith approximation in their respective fixpoint computation. As such, the χCTL operators EU, and EG are the limits of EU_i and EG^i, respectively.

$$\begin{array}{lcl} E[\varphi U \psi] & = & E[\varphi U_\infty \psi] \\ EG\varphi & = & EG^\infty\varphi \end{array}$$

It is also convenient to refer to various subsets of χCTL. In the rest of this thesis we use χACTL to refer to the universal fragment of χCTL, and χECTL to refer to its existential fragment.

Chapter 3

Complexity of Symbolic Multi-Valued Model-Checking

3.1 Introduction

It was shown in[CDEG02] that the worst-case complexity of model-checking of a χCTL property φ on a χKripke structure M is in $O(h \times |S| \times |\varphi| \times D)$, where S is the state space of M, h is the height of the lattice $\mathcal{L}$ used by the model-checker, and D is the complexity of performing EX operation. The intuition behind this result is as follows: (a) multi-valued model checking is reducible to a computation of least and greatest fixpoint of a monotone function over the lattice $L_{mv} = S \rightarrow \mathcal{L}$ of mv-sets over $|S|$; (b) for a monotone function $f : L \rightarrow L$, where L is a lattice of height h, complexity of computing the least and greatest fixpoints is in $O(h)$; (c) the height of L_{mv} is bounded above by $h \times |S|$.

However, it is easy to see that this is not a tight bound. In particular, we claim that the worst-case complexity is linear in $|S|$, but is *independent* of h. For example, consider the χKripke structure in Figure 2.6(a). We can compute the value of $[\![EFa]\!](s_0)$ on it in exactly 3 steps, independently of the height of the lattice used to specify it. In the worst case, the computation of $[\![EFa]\!](s)$ requires examination of all acyclic paths from s, and is, therefore, bounded by the length of the longest such path.

Let us examine how the computation of EFa is performed by the model-checker. Recall from Section 2.6 that $EFa = E[\top Ua]$, and let $EF^i a \triangleq E[\top U_i a]$. At the 0th iteration, the model checking algorithm computes $[\![EF^0 a]\!]$, which is equivalent to $[\![p]\!]$; at the 1st iteration it computes $[\![EF^1 a]\!]$; and at the ith it computes $[\![EF^i a]\!]$. The algorithm terminates when $[\![EF^{i-1} a]\!] = [\![EF^i a]\!]$, and returns $[\![EF^i a]\!]$ as the result.

Computation of $[\![EF^i a]\!](s)$ is equivalent to computing $[\![EFa]\!](s)$ restricted to paths of length at most i. Thus, in this case the model-checking algorithm reduces to a breadth-first search of the computational tree of the χKripke structure. As such, it must terminate as soon as the longest acyclic path from s has been explored. Finally, since the length of the longest acyclic path is bounded above by $|S|$, the worst-case complexity of computing the EF operator must be in $O(|S| \times D)$.

The rest of the chapter is organized as follows. Section 3.2 establishes the technical machinery used to prove the new complexity bounds. Section 3.3 formally proves our conjecture by establishing that the worst-case complexity of model-checking a property φ is in $O(|S| \times |\varphi| \times D)$. Finally, we compare our results with the complexity bounds established by Konikowska and Penzcek [KP02] in Section 3.4.

3.2 Extending the Transition Relation

Intuitively, a multi-valued model-checker computes supremum and infimum over paths of a χKripke structure. Let $\mathbb{R}$ be a transition relation of a χKripke structure M, and $s, t \in S$ be two states of M. Then, $\mathbb{R}(s,t)$ is the value of the (s,t) transition in M, and, since every transition is unique, it is also the supremum over all (s,t) transitions in M. Similarly, $\bigvee_{u \in S} \mathbb{R}(s,u) \wedge \mathbb{R}(u,t)$ is the supremum over all (s,t) paths of length 2 in M. Let us define $\mathbb{R}^n(s,t)$ to be the supremum over all (s,t)-paths of length n.

Definition 11 *Let $\mathbb{R}$ be the transition relation of a χKripke structure M, and $u_0, u_n \in S$ be* *any two states of M. Then*

$$\mathbb{R}^n(u_0, u_n) \triangleq \bigvee_{u_1,\ldots,u_{n-1}\in S} \bigwedge_{i=0}^{n-1} \mathbb{R}(u_i, u_{i+1}) \quad \textit{(}\mathbb{R}^n(s,t)\textit{ defn.)}$$

The following theorem establishes that R^n can also be defined recursively.

Theorem 3 *Let $s, t \in S$ be any two states of M, then*

$$\begin{aligned}\mathbb{R}^n(s,t) &= \bigvee_{u\in S} \mathbb{R}(s,u) \wedge \mathbb{R}^{n-1}(u,t) \\ &= \bigvee_{u\in S} \mathbb{R}^{n-1}(s,u) \wedge \mathbb{R}(u,t)\end{aligned}$$

Note that $R^i(s,t)$ is not monotone in i. That is,

$$i \geq j \not\Rightarrow \mathbb{R}^i(s,t) \sqsupseteq \mathbb{R}^j(s,t)$$

This corresponds to our intuition, since existence of an (s,t)-path of length n does not imply existence of any other path. However, if we consider all paths of length at most n, then the resulting function should be monotone, and in fact non-decreasing.

Theorem 4 *Let $\mathbb{R}$ be the transition relation of a χKripke structure M, and $s, t \in S$ be two states of M. Then, $\mathbb{R}^{\leq i}(s,t) \triangleq \bigvee_i \mathbb{R}^i(s,t)$ is monotone in i:*

$$\forall i, j \in nat \cdot i \geq j \Rightarrow \mathbb{R}^{\leq i}(s,t) \sqsupseteq \mathbb{R}^{\leq j}(s,t)$$

We denote the limit of $\mathbb{R}^{\leq i}$ as i approaches infinity by $\mathbb{R}^{\leq\infty}$, and define it formally as:

$$\mathbb{R}^{\leq\infty}(s,t) \triangleq \bigvee_{i=1}^{\infty} \mathbb{R}^i(s,t) \quad (\mathbb{R}^{\leq\infty}\text{ defn.})$$

Intuitively, $\mathbb{R}^{\leq\infty}$ is the transitive closure of $\mathbb{R}$, that is $\mathbb{R}^{\leq\infty}(s,t)$ is the supremum over *all* (s,t)-paths. Note that we still must show that $\mathbb{R}^{\leq\infty}$ is well defined. We do this by showing that there exists $n \in nat$ such that $\mathbb{R}^{\leq\infty}(s,t) = \mathbb{R}^{\leq n}(s,t)$.

Lemma 1 *Let $\mathbb{R}$ be the transition relation of a χKripke structure M, and S be the state space of M. Then,*

$$\forall n \in nat \cdot n \geq |S| \Rightarrow \mathbb{R}^{\leq|S|}(s,t) \sqsupseteq \mathbb{R}^n(s,t)$$

Proof:

We show by induction on n, that for all $n \geq |S|$,

$$\forall u_0, \ldots, u_n \cdot \exists r \leq |S| \cdot \exists w_0, \ldots, w_r \cdot \bigwedge_{i=0}^{n-1} \mathbb{R}(u_i, u_{i+1}) \sqsubseteq \bigwedge_{j=0}^{r-1} \mathbb{R}(w_j, w_{j+1})$$

which is sufficient to establish the lemma. Intuitively, we show that for any path of length greater than $|S|$, there exists an equivalent path of length at most $|S|$.

The base case is $n = |S|$, which holds trivially. Assume for $|S| \leq n \leq k$, and show for $n = k + 1$. Let $i, j \in nat$ be such that $j > i \wedge u_i = u_j$. The existence of such i, j is guaranteed by the fact that $k + 1 > |S|$:

$$k + 1 > |S| \Rightarrow \exists i, j \cdot j > i \wedge u_i = u_j$$

Let $m = j - i$, and define $w_0, \ldots, w_{k+1-m}$ as follows, where $0 \leq p \leq k + 1 - m$:

$$w_p = \begin{cases} u_p & 0 \leq p \leq i \\ u_{p+m} & i < p \end{cases}$$

Note that this definition ensures that the first and last states of the sequences u_i and w_i are the same. That is, $u_0 = w_0$ and $u_{k+1} = w_{k+1-m}$. Clearly, $k + 1 - m \leq k$, and

$$\bigwedge_{a=0}^{k+1} \mathbb{R}(u_a, u_{a+1}) \sqsubseteq \bigwedge_{b=0}^{k+1-m} \mathbb{R}(w_b, w_{b+1})$$

Thus, we can apply the inductive hypothesis to complete the proof. □

From Lemma 1 it follows that $\mathbb{R}^{\leq\infty}$ has an upper bound.

Corollary 1 *Let $\mathbb{R}$ be the transition relation of a χKripke structure M, and S be its state space. Then,*

$$\mathbb{R}^{\leq|S|}(s, t) \sqsupseteq \mathbb{R}^{\leq\infty}(s, t)$$

Finally, combining the results of Theorem 4 with Corollary 1 we show that $\mathbb{R}^{\leq\infty}$ is well-defined.

Theorem 5 *Let $\mathbb{R}$ be the transition relation of a χKripke structure M, and S be its state space. Then,*

$$\forall k \in nat \cdot k \geq |S| \Rightarrow \mathbb{R}^{\leq k}(s, t) = \mathbb{R}^{\leq\infty}(s, t)$$

The definition of $\mathbb{R}^{\leq\infty}(s,t)$ lets us talk about arbitrary (s,t)-paths. However, its limitation is that we must always specify the destination state. To remedy this problem, we define $\mathbb{R}^i(s)$ to be the supremum over all paths of length i, starting at s, and denote its limit by $\mathbb{R}^\infty(s)$. Formally,

$$\begin{array}{lll} \mathbb{R}^i(s) & \triangleq & \bigvee_{t\in S}\mathbb{R}^i(s,t) \qquad (\mathbb{R}^i(s)\text{ defn.}) \\ \mathbb{R}^\infty(s) & \triangleq & \bigvee_{t\in S}\mathbb{R}^\infty(s,t) \quad (\mathbb{R}^\infty(s)\text{ defn.}) \end{array}$$

Since existence of a path of length n implies existence of a shorter path, $\mathbb{R}^i(s)$ is monotone, and in fact is non-increasing. Next, we formally establish the monotonicity of $\mathbb{R}^i(s)$, and show that $\mathbb{R}^\infty(s)$ is well-defined.

Theorem 6 *Let $\mathbb{R}$ be the transition relation of a χKripke structure M, and s be any state of M. Then,*

$$\forall i,j\in nat\cdot i\geq j\Rightarrow \mathbb{R}^i(s)\sqsubseteq\mathbb{R}^j(s)$$

Proof:

With out loss of generality, let $i=j+1$.

$$\begin{array}{lll} & \mathbb{R}^j(s)\sqsupseteq\mathbb{R}^{j+1}(s) & \text{(def. of }\mathbb{R}^i(s)) \\ = & \bigvee_{u\in S}\mathbb{R}^j(s,u)\sqsupseteq\bigvee_{t\in S}\mathbb{R}^{j+1}(s,t) & \text{(by Theorem 3)} \\ = & \bigvee_{u\in S}\mathbb{R}^j(s,u)\sqsupseteq\bigvee_{t\in S}\bigvee_{u\in S}\mathbb{R}^j(s,u)\wedge\mathbb{R}(u,t) & \text{(generalization)} \\ \Leftarrow & \bigvee_{u\in S}\mathbb{R}^j(s,u)\sqsupseteq\bigvee_{u\in S}\mathbb{R}^j(s,u)\wedge\mathbb{R}(u,t) & \text{(monotonicity)} \\ \Leftarrow & \mathbb{R}^j(s,u)\sqsupseteq\mathbb{R}^j(s,u)\wedge\mathbb{R}(u,t) & \text{(specialization)} \\ \Leftarrow & \mathbb{R}^j(s,u)\sqsupseteq\mathbb{R}^j(s,u) & \end{array}$$

$\square$

Next, we show that $\mathbb{R}^i(s)$ is bounded from below.

Lemma 2 *Let $\mathbb{R}$ be the transition relation of a χKripke structure M, S be its state space, and $s\in S$. Then,*

$$\forall n\in nat\cdot n>|S|\Rightarrow\mathbb{R}^n(s)\sqsupseteq\mathbb{R}^{|S|+1}(s)$$

Proof:

Here, we prove the lemma for $n = |S| + 2$. The proof for arbitrary n is similar. Notice, that to establish the lemma it is sufficient to show that for every path of length $|S| + 1$ there exists an equivalent path of length $|S| + 2$. That is, we want to establish that

$$\forall u_0, \ldots, u_{|S|+1} \cdot \exists w_0, \ldots, w_{|S|+2} \cdot \bigwedge_{i=0}^{|S|} \mathbb{R}(u_i, u_{i+1}) \sqsubseteq \bigwedge_{j=0}^{|S+1|} \mathbb{R}(w_j, w_{j+1})$$

As in Lemma 1, since we have more elements $\{u_i\}$ than the underlying state space S, we let $i, j \in nat$ be such that $i < j \wedge u_i = u_j$, and let $m = j - i$. Next, we define w_p, where p ranges between 0 and $|S| + 2$, as follows:

$$w_p = \begin{cases} u_p & 0 \leq p \leq j \\ u_{i+(p-j \mod m)} & p > j \end{cases}$$

Finally, we show that $\{w_p\}$ satisfies our condition.

$$\begin{array}{lll} & \bigwedge_{p=0}^{|S|+1} \mathbb{R}(w_p, w_{p+1}) & \text{(distributivity)} \\ = & (\bigwedge_{p=0}^{j-1} \mathbb{R}(w_p, w_{p+1})) \wedge (\bigwedge_{p=j}^{|S|+1} \mathbb{R}(w_p, w_{p+1})) & \text{(def. of } w_p\text{, modular arithmetic)} \\ = & \bigwedge_{p=0}^{j-1} \mathbb{R}(w_p, w_{p+1}) & \text{(def. of } w_p\text{)} \\ = & \bigwedge_{p=0}^{j-1} \mathbb{R}(u_p, u_{p+1}) & \text{(since } j \leq |S| + 1\text{)} \\ \sqsupseteq & \bigwedge_{p=0}^{|S|} \mathbb{R}(u_p, u_{p+1}) & \end{array}$$

□

Combining the results of Theorem 6 with Lemma 2, we show that $\mathbb{R}^\infty(s)$ is well-defined.

Theorem 7 *Let $\mathbb{R}$ be the transition relation of a χKripke structure M. Then,*

$$\forall n \in nat \cdot n > |S| \Rightarrow \mathbb{R}^n(s) = \mathbb{R}^\infty(s)$$

Intuitively, Theorem 7 states that it is sufficient to explore only the paths of length $|S| + 1$.

3.3 Deriving the Worst-Case Complexity

To simplify our presentation, we introduce a notation for a nested EX operator EX^n.

$$EX^n\varphi = \begin{cases} \varphi & n = 0 \\ EXEX^{n-1}\varphi & n > 0 \end{cases}$$

Intuitively, $[\![EX\varphi]\!](s)$ computes the value of φ over all immediate successors of s, whereas $[\![EX^n\varphi]\!](s)$ computes the value of φ over all states reachable from s in exactly n steps. This intuition is formalized below.

Theorem 8 *Let φ be a* propositional *formula, M be a χKripke structure, and $\mathbb{R}$ its transition relation. Furthermore, let M_n be a χKripke structure obtained from M by replacing its transition relation with $\mathbb{R}^n$. Then,*

$$[\![EX^n\varphi]\!]^M = [\![EX\varphi]\!]^{M_n}$$

Proof:

The proof proceeds by induction on n. The base case, when $n = 1$ is trivial. Assume that the equality holds for $n = k$, and show for $n = k + 1$.

$$\begin{array}{rll}
 & [\![EX^{k+1}\varphi]\!]^M(s) & \text{(defn. of } EX^n\text{)} \\
= & [\![EX(EX^k\varphi)]\!]^M(s) & \text{(defn. of } EX\text{)} \\
= & \bigvee_{u\in S}(\mathbb{R}(s,u) \wedge [\![EX^k\varphi]\!]^M(u)) & \text{(inductive hyp.)} \\
= & \bigvee_{u\in S}(\mathbb{R}(s,u) \wedge [\![EX\varphi]\!]^{M_k}(u)) & \text{(defn. of } EX\text{)} \\
= & \bigvee_{u\in S}(\mathbb{R}(s,u) \wedge \bigvee_{t\in S}\mathbb{R}^k(u,t) \wedge [\![\varphi]\!]^{M_k}(t)) & \text{(distributivity)} \\
= & \bigvee_{u\in S}\bigvee_{t\in S}(\mathbb{R}(s,u) \wedge \mathbb{R}^k(u,t) \wedge [\![\varphi]\!]^{M_k}(t)) & \text{(commutativity)} \\
= & \bigvee_{t\in S}((\bigvee_{u\in S}\mathbb{R}(s,u) \wedge \mathbb{R}^k(u,t)) \wedge [\![\varphi]\!]^{M_k}(t)) & \text{(defn. of } \mathbb{R}^n\text{)} \\
= & \bigvee_{t\in S}(\mathbb{R}^{k+1}(s,t) \wedge [\![\varphi]\!]^{M_k}(t)) & (\varphi \text{ is propositional)} \\
= & \bigvee_{t\in S}(\mathbb{R}^{k+1}(s,t) \wedge [\![\varphi]\!]^{M_{k+1}}(t)) & \text{(defn. of } EX\text{)} \\
= & [\![EX\varphi]\!]^{M_{k+1}}(s) &
\end{array}$$

$\square$

We are now in the position to derive upper bounds for the complexity of multi-valued model-checking. We start with the EF operator. Notice that we can express $EF\varphi$ using nested EX operator as follows:

$$\begin{array}{lcll} EF^n\varphi & = & \bigvee_{i=0}^{n} EX^i\varphi & (EF^n \text{ expansion}) \\ EF\varphi & = & \bigvee_{i=0}^{\infty} EX^i\varphi & (EF \text{ expansion}) \end{array}$$

The proof of the expansion is trivial, and is omitted here.

In [CDEG02] it was shown that given a χKripke structure, there exists a bound $n = h \times |S|$ such that $[\![EF\varphi]\!] = [\![EF^n\varphi]\!]$, where h is the height of the lattice, and S is the state space. This bound is exactly the number of iterations required to compute EF in the model-checking algorithm. We show that $n = |S|$ is sufficient.

Theorem 9 *Let φ be a propositional formula, M be a χKripke structure, and $s \in S$ be an arbitrary state of M. Then,*

$$[\![EF\varphi]\!](s) = [\![EF^{|S|}\varphi]\!](s)$$

Proof:

$$\begin{array}{cll} & [\![EF\varphi]\!]^M(s) & (EF \text{ expansion}) \\ = & \bigvee_{i=0}^{\infty}[\![EX^i\varphi]\!]^M(s) & (\text{by Theorem 8}) \\ = & \bigvee_{i=0}^{\infty}[\![EX\varphi]\!]_i^M(s) & (\text{def. of } EX) \\ = & \bigvee_{i=0}^{\infty}(\bigvee_{t\in S}\mathbb{R}^i(s,t) \wedge [\![\varphi]\!]^M(t)) & (\text{commutativity}) \\ = & \bigvee_{t\in S}(\bigvee_{i=0}^{\infty}\mathbb{R}^i(s,t) \wedge [\![\varphi]\!]^M(t)) & (\text{distributivity}) \\ = & \bigvee_{t\in S}\big((\bigvee_{i=0}^{\infty}\mathbb{R}^i(s,t)) \wedge [\![\varphi]\!]^M(t)\big) & (\text{def of } R^{\leq\infty}(s,t)) \\ = & \bigvee_{t\in S}(\mathbb{R}^{\leq\infty}(s,t) \wedge [\![\varphi]\!]^M(t)) & (\text{by Theorem 5}) \\ = & \bigvee_{t\in S}(\mathbb{R}^{\leq|S|}(s,t) \wedge [\![\varphi]\!]^M(t)) & (\text{by Theorem 8}) \\ = & \bigvee_{i=0}^{|S|}[\![EX^i\varphi]\!]^M(s) & (EF^n \text{ expansion}) \\ = & [\![EF^{|S|}\varphi]\!]^M(s) & \end{array}$$

□

The new complexity bounds for EF and EU operators, follow from the theorem.

Corollary 2 *The complexity of computing the EF operator is $O(|S| \times D)$.*

Corollary 3 *The complexity of computing EU operator is $O(|S| \times D)$.*

Proof:

Computing $E[\varphi U \psi]$ on a χKripke structure M is equivalent to computing $EF\psi$ restricted to paths on which φ holds. Thus, it is equivalent to computing $EF\psi$ on a χKripke structure M', obtained from M by replacing its transition relation with $\mathbb{R}'(s,t) = \mathbb{R}(s,t) \wedge [\![\varphi]\!](s)$. Since, the state space of M' is exactly the same as of M, the result follows. □

The proof of complexity for EG operator is quite similar. First we express $EG\top$ using nested EX:

$$\begin{array}{rcll} EG^i\top & = & EX^i\top & (EG^i \text{ expansion}) \\ EG\top & = & EX^\infty\top & (EG \text{ expansion}) \end{array}$$

Next, we show that for a given χKripke structure M, $[\![EG\top]\!] = [\![EG^{|S|+1}\top]\!]$.

Theorem 10 *Let φ be a propositional formula, M be a χKripke structure, and $s \in S$ be an arbitrary state of M. Then,*

$$[\![EG\top]\!](s) = [\![EG^{|S|+1}\top]\!](s)$$

Proof:

$$\begin{array}{rll} & [\![EG\varphi]\!]^M(s) & (EG \text{ expansion}) \\ = & [\![EX^\infty\top]\!]^M(s) & \text{(by Theorem 8)} \\ = & [\![EX\top]\!]^{M_\infty}(s) & \text{(def. of } EX) \\ = & \bigvee_{t\in S} \mathbb{R}^\infty(s,t) & \text{(def. of } \mathbb{R}^\infty(s)) \\ = & \mathbb{R}^\infty(s) & \text{(by Theorem 7)} \\ = & \mathbb{R}^{|S|+1}(s) & \text{(def. of } \mathbb{R}^n(s)) \\ = & \bigvee_{t\in S} \mathbb{R}^{|S|+1}(s,t) & \text{(def. of } EX) \\ = & [\![EX\top]\!]^{M_{|S|+1}}(s) & \text{(by Theorem 8)} \\ = & [\![EX^{|S|+1}\top]\!]^M(s) & (EG^n \text{ expansion}) \\ = & [\![EG^{|S|+1}\top]\!]^M(s) & \end{array}$$

□

Corollary 4 *The complexity of computing $EG\top$ on a χKripke structure M is in $O(|S| \times D)$.*

Finally, we extend the result of Corollary 4 to an arbitrary EG operator.

Corollary 5 *The complexity of computing $EG\varphi$ on a χKripke structure M is in $O(|S| \times D)$.*

Proof:

The proof is similar to the proof of Corollary 3. Computing $EG\varphi$ on a χKripke structure M is equivalent to computing $EG\top$ on a χKripke structure M', obtained from M by replacing its transition relation with $\mathbb{R}'(s,t) = [\![\varphi]\!](s) \wedge \mathbb{R}(s,t)$. □

Finally, combining all of the results, we obtain our main theorem.

Theorem 11 *The complexity of computing $[\![\varphi]\!]$ for a χCTL formula φ, on a χKripke structure M is in $O(|S| \times |\varphi| \times D)$.*

Note that the bound $|S|$ is used to ensure that every path of that length contains a cycle. Alternatively, an even better bound can be obtained using the diameter of the graph induced by the χKripke structure, where the diameter of the graph is given by the length of the largest acyclic path between any two states.

3.4 Discussion and Future Work

In this chapter we have shown that the complexity of χCTL model-checking is linear in the size of the state space and the size of the formula. This is exactly the complexity of CTL model-checking. Therefore, one may be led to conclude that χCTL model-checking is no more expensive than its classical counterpart. However, what is hidden in our analysis is the difference between the cost D of performing the EX operation in the multi-valued case, and the cost D_c of performing the same operation in the classical case. In our presentation, we assume that EX operation is implemented by decision diagrams: MDDs [CGD+02] in multi-valued case, and BDDs [Bry86] in the classical case. Thus, to analyze the difference between D and D_c, we must examine the difference in complexity between these decision diagrams. This topic has been already covered in depth by [Weg00], and is beyond the scope of this thesis.

An alternative approach to χCTL model-checking is to reduce the problem to several classical problems. The reduction was first formally established by Bruns and Godefroid [BG99] in the case of 3-valued logic, and extended to the general case in [KP02], but is also present implicitly in the work of Chechik et al. [CGD+02]. If for simplicity, we restrict our attention to χKripke structures with boolean variables and multi-valued transition relation, then the general idea is: (a) reduce a χKripke structure M to k Kripke structures $M_1, \ldots, M_k$ by transforming the transition relation of M; (b) use classical model-checking algorithm to find the value of a given formula φ on each model M_i; and (c) combine the results of step (b) into a single multi-valued answer. The value k depends on the lattice L used to specify M and is bounded above by number of join-irreducible elements of L [CGD+02]. Note that the state space of each M_i is the same as of M, although its reachable part may differ.

With this reduction at hand, the complexity of model-checking a χCTL formula φ is in $O(k \times |S| \times |\varphi| \times D_c)$, where D_c is as defined above. It then follows that performing multi-valued model-checking directly is more efficient if $D \leq k \times D_c$. However, once again, since we assume that the basic operations are performed by decision diagrams, this comparison reduces to comparing the differences between the diagrams. A more detailed analysis between the two approaches, including some empirical evidence, can be found in [CGD+02].

Chapter 4

Fairness

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

4.1 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 [CES86]. Formally,

Definition 12 *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 12 *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. $\square$

4.2 Fairness in Multi-Valued Model-Checking

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

χCTL formulas. Fairness conditions are formally defined by the following grammar:

$$\begin{array}{lcl} \chi\texttt{CTLExpr} & := & \text{defined in Section 2.6} \\ \texttt{BoolAtom} & := & \chi\texttt{CTLExpr Op } \chi\texttt{CTLExpr} \\ \texttt{Op} & \in & \{\sqsubseteq, \sqsubset, \sqsupseteq, \sqsupset, =, \neq\} \\ \texttt{FairnessCond} & := & \texttt{FairnessCond} \vee \texttt{BoolAtom} \mid \texttt{FairnessCond} \wedge \texttt{BoolAtom} \\ & & \mid \neg\texttt{FairnessCond} \mid \texttt{BoolAtom} \end{array}$$

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

Definition 13 *A 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 [HR00], we write A_C and E_C for the operators A and E restricted to paths satisfying the fairness condition C. For example, $[\![A_{\{c_1,c_2\}}G\varphi]\!](s) = \top$ means that φ is TRUE in every trace on which c_1 and c_2 occur infinitely often.

4.3 Fair EG

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

$$[\![E_cG\varphi]\!] \triangleq \nu Z \cdot [\![\varphi]\!] \cap_L (F_1 \circ \cdots \circ F_k)(\mathbb{Z}) \quad \text{(def. 1 of } E_CG)$$

Where F_i is defined as $F_i(\mathbb{Z}) \triangleq [\![EXE[\varphi U\varphi \wedge c_i \wedge \mathbb{Z}]]\!]$. For example, if $C = \{c_1, c_2\}$ this definition simplifies to:

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

The problem with this definition, however, is that it is dependent on the size of C, and a particular order at which elements of C occur on the path. We thus seek an alternative definition:

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

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

Theorem 13 *When $C = \{\top\}$ (no fairness), Definition 2 of $E_C G$ reduces to*

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

Proof:

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

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

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

G); and (2) showing that $\mathbb{W} \supseteq_L \mathbb{Y}$.

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

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

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

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, using Definition 2 of $E_C G$, $[\![E_C G\varphi]\!] = [\![EG\varphi]\!]$. □

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

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

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

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

$$\begin{aligned} G_i(\mathbb{Z}) &= [\![\varphi]\!] \cap_L [\![EXE[\varphi U \varphi \wedge c_i \wedge \mathbb{Z}]]\!] \\ G^0([\![\top]\!]) &= [\![\top]\!] \\ G^{n+1}([\![\top]\!]) &= G_1(G^n([\![\top]\!])) \cap_L G_2(G^n([\![\top]\!])) \end{aligned}$$

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 3 $\forall n \in nat$,

$$(1)\quad \mathbb{K}_n \supseteq_L G^n([\![\top]\!]) \qquad (2)\quad \mathbb{M}_n \supseteq_L G^n([\![\top]\!])$$
$$(3)\quad \mathbb{K}_{2n} \subseteq_L G^n([\![\top]\!]) \qquad (4)\quad \mathbb{M}_{2n} \subseteq_L G^n([\![\top]\!])$$

Proof:

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

$\mathbb{K}_n \supseteq_L \mathbb{K}_{n+1}$ and $\mathbb{M}_n \supseteq_L \mathbb{M}_{n+1}$ (monotonicity of $\mathbb{K}_n$, $\mathbb{M}_n$)

$[\![\varphi]\!] \ \cap_L \ [\![EXE[\varphi\ U\ c_2 \wedge \varphi \wedge \mathbb{K}_n]]\!] \subseteq_L \mathbb{M}_n$ (relation between $\mathbb{K}_n$ and $\mathbb{M}_n$)

(holds because $\mathbb{M}_n = [\![\varphi]\!] \ \cap_L \ [\![EXE[\varphi\ U\ \varphi \wedge c_2 \wedge \mathbb{K}_{n-1}]]\!]$)

$[\![E[\varphi\ U\ \psi]]\!] \supseteq_L [\![E[\varphi\ U\ \varphi \wedge EXE[\varphi\ U\ \psi]]]\!]$ (monotonicity 1 of EU)

(holds because of EU expansion)

$[\![\varphi]\!] \supseteq_L [\![\psi]\!] \Rightarrow [\![E[p\ U\ \varphi]]\!] \supseteq_L [\![E[p\ U\ \psi]]\!]$ (monotonicity 2 of EU)

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

Base Case: $G^0([\![\top]\!]) = \top$ (def. of $G^0([\![\top]\!])$)

IH: Assume (1)-(4) hold for $n = k$

Inductive Case: Proof for $n = k + 1$

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

$\top$

$\Rightarrow$ (IH)

$\mathbb{M}_n \supseteq_L G^n([\![\top]\!])$

$\Rightarrow$ (monotonicity 2 of EU)

$[\![\varphi]\!] \ \cap_L \ [\![EXE\,[\varphi\; U\; \varphi \wedge c_1 \wedge \mathbb{M}_n]]\!]$

$\supseteq_L [\![\varphi]\!] \ \cap_L \ [\![EXE\,[\varphi\; U\; \varphi \wedge c_1 \wedge G^n([\![\top]\!])]]\!]$

$\Rightarrow$ (def. of $\mathbb{K}_{n+1}$, G_1, G^{n+1})

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

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

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

We will show $(3a)$ $\mathbb{K}_{2n+2} \subseteq_L G_1(G^n([\![\top]\!]))$

$(3b)$ $\mathbb{K}_{2n+2} \subseteq_L G_2(G^n([\![\top]\!]))$

Then by $\cap_L$ elimination, we will have the desired property.

$(3a)$ $\mathbb{K}_{2n+2}$

$=$ (def. of $\mathbb{K}_{2n+2}$)

$[\![\varphi]\!] \ \cap_L \ [\![EXE\,[\varphi\; U\; \varphi \wedge c_1 \wedge EXE\,[\varphi\; U\; \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]]\!]$

$\subseteq_L$ (relation between $\mathbb{K}_n$ and $\mathbb{M}_n$))

$[\![\varphi]\!] \ \cap_L \ [\![EXE\,[\varphi\; U\; \varphi \wedge c_1 \wedge \mathbb{M}_{2n}]]\!]$

$\subseteq_L$ (IH and monotonicity)

$[\![\varphi]\!] \ \cap_L \ [\![EXE\,[\varphi\; U\; \varphi \wedge c_1 \wedge G^n([\![\top]\!])]]\!]$

$=$ (def. of G_1)

$G_1(G^n([\![\top]\!]))$

$(3b)$ $\mathbb{K}_{2n+2}$

$=$ (def. of $\mathbb{K}_{2n+2}$)

$[\![\varphi]\!] \cap_L [\![EXE\,[\varphi\; U\; \varphi \wedge c_1 \wedge EXE\,[\varphi\; U\; \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]]\!]$

$\subseteq_L$ (monotonicity)

$[\![\varphi]\!] \cap_L [\![EXE\,[\varphi\; U\; \varphi \wedge EXE\,[\varphi\; U\; \varphi \wedge c_2 \wedge \mathbb{K}_{2n}]]]\!]$

□

Theorem 14 *Definition 1 and Definition 2 of $E_C G\varphi$ are equivalent.*

Proof:

Recall that $\mathbb{K}_{2n} = F^n([\![\top]\!])$, defined above. Since F^n converges, $\exists N_1 \in nat \cdot \forall n \geq N_1 \cdot \nu\mathbb{Z}.F(\mathbb{Z}) = \mathbb{K}_{2n}$. Since G^n converges, $\exists N_2 \in nat \cdot \forall n \geq N_2 \cdot \nu\mathbb{Z}.G(\mathbb{Z}) = G^n([\![\top]\!])$. Further, by Lemma 3, $\forall n \geq N_2$

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

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

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

As a corollary to Theorem 14 we get that the order in which the fairness conditions occur on a path is of no importance.

Corollary 6 *Let $C = \{c_1, \ldots, c_n\}$ be a set of fairness conditions, and let $\sigma : [1 \ldots n] \rightarrow [1 \ldots n]$ be a permutation, then*

$$\nu Z \cdot [\![\varphi]\!] \cap_L (F_1 \circ \cdots \circ F_k)(\mathbb{Z}) \quad = \quad \nu Z \cdot [\![\varphi]\!] \cap_L (F_{\sigma(1)} \circ \cdots \circ F_{\sigma(k)})(\mathbb{Z})$$

We now show that our definition of fairness satisfies the intuitive condition that any finite path that terminates in a state from which there exists a fair path, can itself be extended to a fair path.

Theorem 15 *Let $C = \{c_1, \ldots, c_n\}$ be a set of fairness conditions, and φ be any χCTL formula, then the following holds:*

$$\begin{aligned} &(1) \quad \varphi \wedge EXE[\varphi U c_i \wedge \varphi \wedge E_C G\varphi] &= \quad & E_C G\varphi \\ &(2) \quad E[\varphi U E_C G\varphi] &= \quad & E_C G\varphi \\ &(3) \quad \varphi \wedge EXE_C G\varphi &= \quad & E_C G\varphi \end{aligned}$$

Proof:

(1): For brevity we assume that $C = \{c_1, c_2\}$. Let $\mathbb{K}_n$ and $\mathbb{M}_n$ be defined as in Lemma 3. Since for any given model the fixpoint computation of $E_C G\varphi$ is guaranteed to converge there exists an n such that:

$$\begin{array}{llllll} (i) & \mathbb{K}_{2n} & = & \mathbb{K}_{2n+2} & = & E_C G\varphi \\ (ii) & \mathbb{M}_{2n} & = & E_C G\varphi & & \end{array}$$

and by monotonicity of $\mathbb{K}_n$ we get

$$\begin{array}{ll} & [\![E_C G\varphi]\!] \\ & \text{(choice of } n\text{)} \\ = & \mathbb{K}_{2n} \\ & \text{(monotonicity of } \mathbb{K}_n\text{)} \\ = & \mathbb{K}_{2n+1} \\ & \text{(def. of } \mathbb{K}_n\text{)} \\ = & [\![\varphi \wedge EXE[\varphi U c_1 \wedge \varphi \wedge M_{2n}]]\!] \\ & \text{(choice of } n\text{)} \\ = & [\![\varphi \wedge EXE[\varphi U c_1 \wedge \varphi \wedge E_C G\varphi]]\!] \end{array}$$

Thus, we obtain the desired result:

$$\varphi \wedge EXE[\varphi U c_1 \wedge \varphi \wedge E_C G\varphi] \quad = \quad E_C G\varphi$$

The proof for (2) is a simple consequence of (1); and (3) follows from (1) and (2). □

Notice that as a consequence of (3) we obtain the inequality $EG\varphi \sqsupseteq E_C G\varphi$, since $EG\varphi$ is the greatest fixpoint of $F(\mathbb{Z}) = [\![\varphi \wedge EX\mathbb{Z}]\!]$.

Finally we show the $E_C G\varphi$ is monotone with respect to the set of fairness conditions C.

Theorem 16 *Let $C_1 = \{a_1, \ldots, a_n\}$ and $C_2 = C_1 \cup \{b_1, \ldots, b_m\}$ be two sets of fairness conditions, then $E_{C_1} G\varphi \sqsupseteq E_{C_2} G\varphi$.*

Proof:

Let

$$\begin{array}{lll} F(\mathbb{Z}) & = & \varphi \wedge \bigwedge_{k=1}^{n} EXE[\varphi U \varphi \wedge a_k \wedge \mathbb{Z}] \\ G(\mathbb{Z}) & = & F(\mathbb{Z}) \wedge \bigwedge_{k=1}^{m} EXE[\varphi U \varphi \wedge b_k \wedge \mathbb{Z}] \end{array}$$

Then, $E_{C_1}G\varphi = \nu\mathbb{Z}.F(\mathbb{Z})$ and $E_{C_2}\varphi = \nu\mathbb{Z}.G(\mathbb{Z})$. By monotonicity of $\wedge$ it follows that $F(\mathbb{Z}) \sqsupseteq G(\mathbb{Z})$, and therefore $E_{C_1}G\varphi \sqsupseteq E_{C_2}G\varphi$. □

4.4 Fairness in Other χCTL Operators

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

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

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 equivalent to those of classical CTL.

4.5 Running Time

Running time of the model-checker under fairness conditions C is dominated by the computation of $E_C G$ whose μ-calculus expression is of alternation depth 2 [BS01]. That is, each iteration of the outer fixpoint requires the computation of the inner fixpoint. The inner fixpoint corresponds to a χCTL formula $E[\varphi\ U\ (c_k \wedge \mathbb{Z})]$, and is computed using the regular model-checking algorithm. Thus, the complexity of computing the inner fixpoint is in $O(MC \times D)$, where MC is the number of iterations until convergence of the fixpoint, and D is the complexity of each iteration. If we treat the inner fixpoint of $E_C G$ as a constant, then it is equivalent to a regular EG without fairness and can be computed in MC iterations as well. Finally, computation of $E_C G$ requires computing $|C|$ inner fixpoints, and thus its complexity is $O(|C| \times MC^2 \times D)$. Similarly, the complexity of model-checking an arbitrary χCTL formula φ, under the fairness condition C, is in $O(|\varphi| \times |C| \times MC^2 \times D)$.

4.6 Discussion and Future Work

In this chapter we have shown how fairness conditions can be handled in the context of multi-valued model-checking. Our extension of χCTL to fairness is very similar to one of CTL [CGP99]. In both cases, the EG operator is redefined to account for fairness conditions, with the rest of the operators defined through it. If the logic $\mathbf{2}$ is used for the analysis, then our definitions of fair χCTL become equivalent to one of fair CTL.

An alternative interpretation of fairness was proposed by Huth [Hut02] in the context of Kripke MTS. A Kripke Modal Transition System (MTS) is an extension of a χKripke structure over the logic $\mathbf{3}$, where transitions are labeled with actions in addition to logic values. To our knowledge, this is the only other attempt to address the issue of fairness in multi-valued model-checking. As in our case, fairness is introduced by extending the EG operator using a μ-calculus formula with the alternating depth of 2, but fairness conditions are not required to be boolean. In fact, the definition of fair EG presented in [Hut02] is syntactically equivalent to our definition in Section 4.3, with the exception that the requirement for the fairness condition to be boolean is dropped. As such, our definition of fair EG is a strict subset of the one defined by Huth.

Although it may seem strange that in extending classical CTL to χCTL we keep fairness conditions boolean, we believe that this exactly captures our intuitive notion of fairness. This is especially evident in logics whose truth values do not form a finite total order. In these cases, the interplay between Huth's definition of fairness and the interpretation of quantification over paths completely obscures the result of the model-checker.

In this chapter we have only explored fairness in isolation as it applies to a single multi-valued model. However, we believe that in most practical application of multi-valued modeling, each individual model is only a part of the bigger picture. For example, a multi-valued model can correspond to a partial result of negotiation between multiple stakeholders, or alternatively it can represent the result of a partial refinement [BG00]. In the future we plan to investigate the interactions between fairness as defined in this chapter and various applications of multi-valued modeling.

Chapter 5

Witnesses and Counter-examples

5.1 Introduction

A classical model-checker can tell the user not only whether a desired temporal property is satisfied (or violated), but also generate a witness (or a counter-example), explaining the reasons behind the answer. Typically, witnesses and counter-examples are fairly small and are given in terms of states and transitions of the model; thus, they are readily understood by engineers and can be effectively used for debugging the model. The witness and counter-example generation ability has been one of the major advantages of model-checking in comparison with other verification methods.

The goal of this chapter is to extend the notion of a witness and a counter-example to the multi-valued case. First, let us examine what are the requirements of a witness. Intuitively, a witness is a part of the model that is sufficient to prove the result of the model-checker. That is, the witness is sufficient to prove $[\![\varphi]\!](s) = \ell$, where φ is a temporal property, s is a state of the model, and ℓ is some logic value.

In order to prove that $[\![\varphi]\!](s) = \ell$, we must show that both $[\![\varphi]\!](s) \sqsupseteq \ell$ and $[\![\varphi]\!](s) \sqsubseteq \ell$ are valid. However, in the two special cases where ℓ is either $\top$ or $\bot$ we only need to prove the first or the second statement, respectively. The other one follows from the axioms of $\top$ and

$\bot$. Thus, in the classical case, a witness to $[\![\varphi]\!](s) = \top$ is a subset of the computational tree of the model, such that the value of φ restricted to it is above $\top$. Dually, a counter-example for $[\![\varphi]\!] = \bot$ is a subset of the computational tree such that the value of φ is below $\bot$ when restricted to it.

Extending this to the multi-valued case we get that a witness to $[\![\varphi]\!](s) = \ell$ is a subset of the computational tree of the model on which φ is above ℓ, and a counter-example is a subset of the computational tree on which φ is below ℓ. The consequence of this definition is that a witness to a $\chi ACTL$ property, or a counter-example to a $\chi ECTL$ property, necessarily contain all (or most) paths through the model. Thus, we restrict our attention to witnesses of $\chi ECTL$ properties, and counter-examples of $\chi ACTL$ properties, just as the case in the classical model-checking.

Furthermore, the fact that in $\chi ECTL$ existential quantification is equivalent to supremum (or join), and universal quantification to infimum (or meet), results in the following consequences: (a) a witness (or a counter-example) for any operator is not necessarily linear, and (b) a witness (or a counter-example) is only sufficient to prove that $[\![\varphi]\!](s) \sqsupseteq \ell$ (or $[\![\varphi]\!](s) \sqsubseteq \ell$).

In the rest of this chapter, we show how to automatically generate witnesses and counter-examples for χCTL. Note, that our approach is quite different from the one used by Clarke et al. in [CGMZ95] for classical model-checking. Instead of developing an algorithm to construct witnesses and counter-examples from the model, we first develop a proof system for χECTL and χACTL, show how to use it to automatically generate proofs, and finally how to extract witnesses and counter-examples from the proofs. Although, as was shown in [GC02], it is possible to extend this technique to full χCTL, here we only consider the χACTL and χECTL subsets.

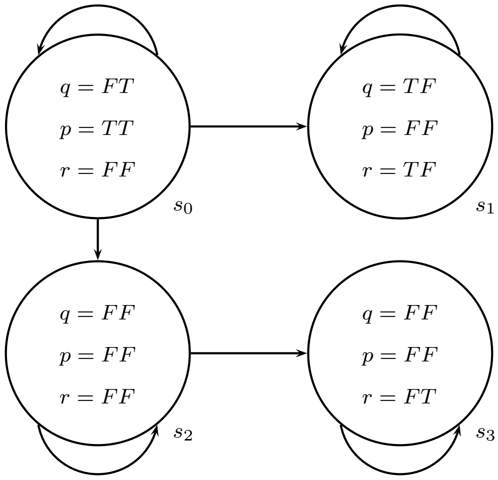

Figure 5.1: A χKripke structure.

5.2 Proof Rules for χECTL

In this section we develop a proof system for sentences of the form $[\![\varphi]\!](s) \sqsupseteq \ell$, where φ is an χECTL formula, ℓ is a lattice value, and s is a state of a given χKripke structure M. We show that the proof system is sound and complete.

Since we are interested in proving statements about a particular χKripke structure we assume that our proof system incorporates the proof systems for quasi-boolean lattices and χKripke structures. Here, we only develop the additional rules for the temporal logic.

We start by identifying the axioms of our proof system. The set of axioms is comprised of all of the axioms of the theory of quasi-boolean lattices, the axioms of the theory of χKripke structures, and the axiomatization of the particular lattice and the χKripke structure we are dealing with. The axiomatization of a quasi-boolean lattice is simply the axioms defining the relations $\sqsupseteq$ and $\neg$, and the axiomatization of a χKripke structure is the axioms defining the transition relation $\mathbb{R}$ and the state labeling function I.

For example, consider the χKripke structure in Figure 5.1, specified using logic 4. Some of the axioms describing the lattice are:

$$\text{TT} \sqsupseteq \text{TF} \quad \text{TF} \sqsupseteq \text{FF}$$

$$\neg\text{TT} = \text{FF} \quad \neg\text{TF} = \text{FT}$$

$$\frac{\ell_1 \sqsupseteq \ell}{[\![\ell_1]\!](s) \sqsupseteq \ell}\ \text{value-rule} \qquad \frac{\neg\ell_1 \sqsupseteq \ell}{[\![\neg\ell_1]\!](s) \sqsupseteq \ell}\ \text{neg-value-rule}$$

$$\frac{\exists \ell_1 \in \mathcal{L} \cdot I(s,p) = \ell_1 \wedge \ell_1 \sqsupseteq \ell}{[\![p]\!](s) \sqsupseteq \ell}\ \text{atomic-rule} \quad \frac{\exists \ell_1 \in \mathcal{L} \cdot \neg I(s,p) = \ell_1 \wedge \ell_1 \sqsupseteq \ell}{[\![\neg p]\!](s) \sqsupseteq \ell}\ \text{neg-atomic-rule}$$

$$\frac{\begin{array}{c}\exists \ell_1, \ell_2 \in \mathcal{L} \cdot [\![\varphi]\!](s) \sqsupseteq \ell_1 \wedge \\ [\![\psi]\!](s) \sqsupseteq \ell_2 \wedge (\ell_1 \sqcap \ell_2 \sqsupseteq \ell)\end{array}}{[\![\varphi \wedge \psi]\!](s) \sqsupseteq \ell}\ \wedge\text{-rule} \qquad \frac{\begin{array}{c}\exists \ell_1, \ell_2 \in \mathcal{L} \cdot [\![\varphi]\!](s) \sqsupseteq \ell_1 \wedge \\ [\![\psi]\!](s) \sqsupseteq \ell_2 \wedge (\ell_1 \sqcup \ell_2 \sqsupseteq \ell)\end{array}}{[\![\varphi \vee \psi]\!](s) \sqsupseteq \ell}\ \vee\text{-rule}$$

$$\frac{\begin{array}{c}\exists t_1, \ldots, t_n \in S \cdot \exists \ell_1, \ldots, \ell_n \in \mathcal{L} \cdot [\![\mathbb{R}(s, t_1) \wedge \varphi]\!](t_1) \sqsupseteq \ell_1 \\ \wedge \cdots \wedge [\![\mathbb{R}(s, t_n) \wedge \varphi]\!](t_n) \sqsupseteq \ell_n \wedge (\bigsqcup_{i=1}^{n} \ell_i) \sqsupseteq \ell\end{array}}{[\![EX\varphi]\!](s) \sqsupseteq \ell}\ EX$$

Figure 5.2: Proof rules for existential propositional temporal logic (χEPTL).

And some of the axioms describing the χKripke structure are:

$$\mathbb{R}(s_0, s_1) = \text{TT} \quad \mathbb{R}(s_0, s_3) = \text{FF}$$
$$I(s_0, p) = \text{TT} \quad I(s_0, q) = \text{FT}$$

We are now at the position to define the rules for the temporal logic. In what follows, we use p and q to represent any atomic propositions, s and t to represent states, φ and ψ to represent temporal formulas, and ℓ and ℓ_i to represent lattice elements. We also use the notation $\{s\}$ to stand for the smallest formula which is $\top$ at state s, and $\bot$ otherwise (i.e $[\![\{s\}]\!](t) \triangleq (s = t)$). We use $\overline{\{s\}}$ for negation of $\{s\}$.

The proof rules for χEPTL, the logic consisting of non-temporal operators and EX, are summarized in Figure 5.2. They follow directly from the definitions of χECTL, which guarantees their soundness.

Notice that the EX-rule introduces existential quantifiers, to eliminate which we introduce the one-point rule:

$$\frac{f(d)}{\exists x \in D \cdot f(x)}\ \text{one-point rule}$$

$$\frac{\llbracket \psi \rrbracket(s) \sqsupseteq \ell}{\llbracket E[\varphi U_0 \psi] \rrbracket(s) \sqsupseteq \ell}\ EU_0 \quad \frac{\llbracket \psi \vee \varphi \wedge EXE[\varphi U_{n-1} \psi] \rrbracket(s) \sqsupseteq \ell}{\llbracket E[\varphi U_n \psi] \rrbracket(s) \sqsupseteq \ell}\ EU_i$$

$$\frac{\exists n \in nat \cdot \llbracket E[\varphi U_n \psi] \rrbracket(s) \sqsupseteq \ell}{\llbracket E[\varphi U \psi] \rrbracket(s) \sqsupseteq \ell}\ EU$$

Figure 5.3: Proof rules for bounded and unbounded EU.

The proof rules for bounded and unbounded version of EU are given in Figure 5.3. As in the previous case, the rules for bounded EU follow directly from its definition. The rule for the bounded EU is the consequence of the monotonicity of EU_i:

$$\forall i, j \in nat \cdot i \geq j \quad \Rightarrow \quad E[\varphi U_i \psi] \sqsupseteq E[\varphi U_j \psi]$$

Notice that since we assume that the state space is finite the rule is actually bi-directional. That is, for a given χKripke structure M there always exists a natural number n such that $E[\varphi U \psi] = E[\varphi U_n \psi]$, given by the diameter of the directed graph induced by M.

To complete our proof system, we still need to find a proof rule for EG. Unfortunately, we cannot proceed as in the previous cases and use the χECTL equivalence $EG\varphi = \varphi \wedge EXEG\varphi$ to define the proof rule. Doing so results in a proof system which is not complete, since the rule can introduce infinite cycles into our proofs.

Let us instead explore several properties of EG. Intuitively, $\llbracket EG\varphi \rrbracket(s)$ is the result of evaluating $G\varphi$ on all infinite paths emanating from the state s. Moreover, since we are dealing with finite state systems, every infinite path can be decomposed into a finite (and potentially empty) prefix, and a finite repeating suffix. Thus, we should be able to decompose $\llbracket EG\varphi \rrbracket(s)$ into the join of the EG restricted to all non-trivial cycles around s, and EG restricted to all infinite paths that do not contain s in the future.

Let us first consider the restriction of $\llbracket EG\varphi \rrbracket(s)$ to all non-trivial cycles around s. Essentially, this is simply a fair-EG, where the fairness condition is given by a single formula $\{s\}$. Furthermore, since we are interested in the value of this fair-EG starting from the state s, it is sufficient to restricting our attention to all non-trivial finite paths from s to itself. We now formalize our intuition, but first let us prove a small technical result.

Lemma 4 *The following are equivalent:*

1. $E[\varphi U \psi_1 \wedge \psi_2 \wedge \{s\}]$
2. $[\![\psi_1]\!](s) \wedge E[\varphi U \psi_2 \wedge \{s\}]$

Proof:

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

$$\begin{array}{rcll} \forall s \in S \cdot \{s\} \wedge \psi & = & \{s\} \wedge [\![\psi]\!](s) & \text{(state restriction)} \\ \forall \ell \in \mathcal{L} \cdot EX(\ell \wedge \psi) & = & \ell \wedge EX\psi & \text{(constant removal)} \end{array}$$

The proof proceeds by induction on the fixpoint characterization of EU.

Base Case:

$$\begin{array}{rll} & E[\varphi U_0 \psi_1 \wedge \psi_2 \wedge \{s\}] & \text{(def. of } EU_0) \\ = & \{s\} \wedge \psi_1 \wedge \psi_2 & \text{(state restriction)} \\ = & \{s\} \wedge [\![\psi_1]\!](s) \wedge \psi_2 & \text{(def. of } EU_0) \\ = & [\![\psi_1]\!](s) \wedge E[\varphi U_0 \{s\} \wedge \psi_2] & \end{array}$$

Induction Hypothesis:

$$E[\varphi U_k \psi_1 \wedge \psi_2 \wedge \{s\}] \quad = \quad [\![\psi_1]\!](s) \wedge E[\varphi U_k \psi_2 \wedge \{s\}]$$

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

$$\begin{array}{rlr} & E[\varphi U_{k+1}\ \psi_1 \wedge \psi_2 \wedge \{s\}] & \text{(def. of } EU_i) \\ = & \big(\{s\} \wedge \psi_1 \wedge \psi_2\big) \vee \big(\varphi \wedge EXE[\varphi U_k\ \psi_1 \wedge \psi_2 \wedge \{s\}]\big) & \text{(induction hyp., state restriction)} \\ = & \big([\![\psi_1]\!](s) \wedge \{s\} \wedge \psi_2\big) \vee \big(\varphi \wedge EX([\![\psi_1]\!](s) \wedge E[\varphi U_k\ \{s\} \wedge \psi_2])\big) & \text{(constant removal)} \\ = & \big([\![\psi_1]\!](s) \wedge \{s\} \wedge \psi_2\big) \vee \big(\varphi \wedge [\![\psi_1]\!](s) \wedge EXE[\varphi U_k\ \{s\} \wedge \psi_2]\big) & \text{(idempotence)} \\ = & [\![\psi_1]\!](s) \wedge (\{s\} \wedge \psi_2 \vee \varphi \wedge EXE[\varphi U_k\ \{s\} \wedge \psi_2]) & \text{(def. of } EU_i) \\ = & [\![\psi_1]\!](s) \wedge E[\varphi U_{k+1}\ \{s\} \wedge \psi_2] & \end{array}$$

□

Now, using the previous Lemma, we show that $[\![EG\varphi]\!](s)$ restricted to all paths that contain the state s infinitely often is equivalent to evaluating φ on all finite paths from s to itself.

Lemma 5 *The following are equivalent:*

1. $[\![E_{\{s\}}G\varphi]\!](s)$

2. $[\![\varphi \wedge EXE[\varphi U \varphi \wedge \{s\}]]\!](s)$

Proof:

Let $F(\mathbb{Z}) = \varphi \wedge EXE[\varphi U \varphi \wedge \{s\} \wedge \mathbb{Z}]$, then from the definition of fair EG we know that

$$\nu\mathbb{Z}.F(\mathbb{Z}) = E_{\{s\}}G\varphi$$

We show that (1) is equivalent to (2) by showing that $[\![F^i(\top)]\!](s)$ is equivalent to (2) for all i. The proof proceeds by induction on i.

Base Case:

$$\begin{array}{rll} & F(\top) & \text{(def. of } F) \\ = & \varphi \wedge EXE[\varphi U \varphi \wedge \{s\}] & \end{array}$$

Induction Hypothesis:

$$[\![F^k(\mathbb{Z})]\!](s) \quad = \quad [\![\varphi \wedge EXE[\varphi U \varphi \wedge \{s\}]]\!](s)$$

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

$$\begin{array}{rll} & [\![F^{k+1}(\top)]\!](s) & \text{(def. of } F) \\ = & [\![\varphi \wedge EXE[\varphi U \varphi \wedge \{s\} \wedge F^k(\top)]]\!](s) & \text{(state restriction)} \\ = & [\![[\![F^k(\top)]\!](s) \wedge \varphi \wedge EXE[\varphi U \varphi \wedge \{s\}]]\!](s) & \text{(def. of } F) \\ = & [\![F^k(\top)]\!](s) \cap_L [\![F(\top)]\!](s) & \text{(monotonicity)} \\ = & [\![F(\top)]\!](s) & \text{(def. of } F) \\ = & [\![\varphi \wedge EXE[\varphi U \varphi \wedge \{s\}]]\!](s) & \end{array}$$

□

The above Lemma is sufficient to derive our proof rule for EG; however, we need a stronger result to prove completeness of our proof system. We start by proving yet another equivalence.

Lemma 6 *The following are equivalent:*

1. $EG(\varphi \wedge \overline{\{s\}}) \vee E[\varphi U \varphi \wedge \{s\}]$

2. $EG(\varphi) \vee E[\varphi U \varphi \wedge \{s\}]$

Proof:

The proof proceeds by induction on the fixpoint characterization of EU and EG.

Base Case:

$$\begin{aligned}
& EG^0(\varphi \wedge \overline{\{s\}}) \vee E[\varphi U_0 \varphi \wedge \{s\}] && \text{(def. of } EG \text{ and } EU\text{)} \\
= \;& (\varphi \wedge \overline{\{s\}} \wedge EX\top) \vee (\varphi \wedge \{s\}) && \text{(distributivity)} \\
= \;& \varphi \wedge (\overline{\{s\}} \wedge EX\top \vee \{s\}) && \text{(distributivity)} \\
= \;& \varphi \wedge \big((\overline{\{s\}} \vee \{s\}) \wedge (EX\top \vee \{s\})\big) && \text{(using } \overline{\{s\}} \vee \{s\} = \top\text{)} \\
= \;& \varphi \wedge (EX\top \vee \{s\}) && \text{(distributivity)} \\
= \;& (\varphi \wedge \{s\}) \vee (\varphi \wedge EX\top) && \text{(def. of } EG \text{ and } EU\text{)} \\
= \;& E[\varphi U_0 \varphi \wedge \{s\}] \vee EG^0\varphi &&
\end{aligned}$$

Inductive Hypothesis:

$$EG^n(\varphi \wedge \overline{\{s\}}) \vee E[\varphi U_n \varphi \wedge \{s\}] \quad = \quad E[\varphi U_n \varphi \wedge \{s\}] \vee EG^n\varphi$$

Ind. Case:

$$\begin{aligned}
& EG^{n+1}(\varphi \wedge \overline{\{s\}}) \vee E[\varphi U_{n+1} \varphi \wedge \{s\}] && \text{(def. of } EG \text{ and } EU\text{)} \\
= \;& (\varphi \wedge \overline{\{s\}} \wedge EXEG^n(\varphi \wedge \overline{\{s\}})) \vee (\varphi \wedge \{s\}) \vee (\varphi \wedge EXE[\varphi U_n \varphi \wedge \{s\}]) && \text{(distributivity)} \\
= \;& \varphi \wedge \big(EX(EG^n(\varphi \wedge \overline{\{s\}}) \vee E[\varphi U_n \varphi \wedge \{s\}]) \vee \{s\}\big) && \text{(induction hypothesis)} \\
= \;& \varphi \wedge \big(EX(EG^n\varphi \vee E[\varphi U_n \varphi \wedge \{s\}]) \vee \{s\}\big) && \text{(distributivity)} \\
= \;& (\varphi \wedge \{s\}) \vee (\varphi \wedge EXEG^n\varphi) \vee (\varphi \wedge EXE[\varphi U_n \varphi \wedge \{s\}]) && \text{(def. of } EU \text{ and } EG\text{)} \\
= \;& EG^{n+1}\varphi \vee E[\varphi U_{n+1} \varphi \wedge \{s\}] &&
\end{aligned}$$

□

Finally, we obtain an equivalence which serves as the basis for our EG proof rule.

Theorem 17 *The following are equivalent:*

1. $[\![EG\varphi]\!](s)$

2. $[\![(\varphi \wedge EXE[\varphi U \varphi \wedge \{s\}]) \vee (\varphi \wedge EXEG(\varphi \wedge \overline{\{s\}}))]\!](s)$

Proof:

We first show that (2) $\sqsupseteq$ (1).

$$\begin{array}{rll} & \varphi \wedge EXE[\varphi U \varphi \wedge \{s\}] \vee \varphi \wedge EXEG(\varphi \wedge \overline{\{s\}}) & \text{(distributivity)} \\ = & \varphi \wedge (EX(E[\varphi U \varphi \wedge \{s\}] \vee EG(\varphi \wedge \overline{\{s\}}))) & \text{(by Lemma 6)} \\ = & \varphi \wedge (EX(E[\varphi U \varphi \wedge \{s\}] \vee EG\varphi)) & \text{(distributivity)} \\ = & (\varphi \wedge EXE[\varphi U \varphi \wedge \{s\}]) \vee (\varphi \wedge EXEG\varphi) & \text{(def. of } EG\text{)} \\ = & (\varphi \wedge EXE[\varphi U \varphi \wedge \{s\}]) \vee EG\varphi & \text{(monotonicity)} \\ \sqsupseteq & EG\varphi & \end{array}$$

To show that (1) $\sqsupseteq$ (2) note that $EG\varphi \sqsupseteq \varphi \wedge EXEG(\varphi \wedge \overline{\{s\}})$ by monotonicity. Therefore, it is sufficient to show that

$$[\![EG\varphi]\!](s) \quad \sqsupseteq \quad [\![\varphi \wedge EXE[\varphi U \varphi \wedge \{s\}]]\!](s)$$

$$\begin{array}{rll} & [\![EG\varphi]\!](s) & \text{(by Theorem 16)} \\ \sqsupseteq & [\![E_{\{s\}}G\varphi]\!](s) & \text{(by Lemma 5)} \\ = & [\![\varphi \wedge EXE[\varphi U \varphi \wedge \{s\}]]\!](s) & \end{array}$$

□

Using Theorem 17 we obtain the EG-rule:

$$\frac{[\![\varphi \wedge EXE[\varphi U \varphi \wedge \{s\}] \vee \varphi \wedge EXEG(\varphi \wedge \overline{\{s\}})]\!](s) \sqsupseteq \ell}{[\![EG\varphi]\!](s) \sqsupseteq \ell} \; EG$$

Finally, we show that the proof system developed so far is sound and complete.

Theorem 18 *The proof system for* χ*ECTL is sound and complete.*

Proof:

The soundness follows from each proof rule being sound. Since each rule was derived as a consequence of some equivalence, the proof of soundness is trivial, and is omitted here.

To prove completeness we show that using our proof system it is possible to prove any valid statement of the form $[\![\varphi]\!](s) \sqsupseteq \ell$. The proof proceeds on the structure of the formula φ, where φ is:

1. a propositional temporal formula (i.e. it does not contain EU and EG).

2. (1) with addition of bounded EU.

3. (2) with addition of unbounded EU.

4. (3) with addition of EG (i.e. $\varphi \in \chi ECTL$).

We only present a sketch of the proof here.

To prove (1) notice that each rule for a propositional temporal formula φ reduces φ to its sub-formulas. Therefore, the proof proceeds by induction on the number of sub-formulas of φ.

The proof of (2) is obtained from the fact that for each formula with a bounded EU there exists an equivalent formula without the EU. (i.e. simply expand the EU_i using its definition).

If φ contains an unbounded EU, than for a given χKripke structure there exists an equivalent formula in which all unbounded EU operators are replaced with their bounded version. Thus, the prove of (3) is trivial.

The prove of (4) is based on the fact that the EG-rule reduces an $EG\varphi$ formula to a formula from (3) (the EU part), and a formula containing EG. However, the new EG operator is restricted to a subset of the state space that does not contain the current state s. Therefore, this rule can only be applied $|S|$ times, ensuring that there are no infinite loops in the proof. $\square$

5.3 Automatic Proof Generation

Given a statement $[\![\varphi]\!](s) \sqsupseteq \ell$, we are interested in automatically generating a proof of its validity. One way to accomplish this is to embed the proof system of Section 5.2 into an automated theorem prover, such as PVS [OSR93], and use its facilities to generate the proof. However, this approach is viable only if there exist an efficient algorithm for the application of the one-point rule. The goal of this section is to develop such an algorithm.

In the rest of the section, we assume that there are two decision procedures available to us: `modelCheck`, and `qblat`. The decision procedure $\texttt{modelCheck}(\varphi, s)$ computes the value $[\![\varphi]\!](s)$, and $\texttt{qblat}(\varphi)$ decides the validity of a quasi-boolean lattice equation (or inequality) φ.

We start with the simple statement $[\![p]\!] \sqsupseteq \ell$, where p is an atomic proposition, and apply the atomic-rule:

$$\frac{\exists \ell_1 \in \mathcal{L} \cdot I(s,p) = \ell_1 \wedge \ell_1 \sqsupseteq \ell}{[\![p]\!](s) \sqsupseteq \ell} \text{ atomic-rule}$$

Next we need to decide if it is possible to instantiate ℓ_1 to apply the one-point rule. Since result of model-checking $[\![p]\!](s)$ is just the value of $I(s,p)$, we get the following proposition:

Proposition 1 *Let p be an atomic proposition, and s be a state of a χKripke structure M. Then,*

$$I(s,p) = \ell \;\;\Leftrightarrow\;\; \texttt{modelCheck}(p,s) = \ell$$

Therefore, the one-point rule is applicable if and only if instantiating ℓ_1 with the result of $\texttt{modelCheck}(p,s)$ does not result in an invalid statement. Since the statement $I(s,p) = \texttt{modelCheck}(p,s)$ is always valid, this is equivalent to requiring $\texttt{modelCheck}(p,s) \sqsupseteq \ell$ to be valid. Putting all of the pieces together, we obtain the algorithm in Figure 5.4(a) that uses to `qblat` to test the validity of the lattice inequality. The case of $[\![\neg p]\!](s) \sqsupseteq \ell$ is handled similarly.

Next we consider the boolean connectives $\wedge$ and $\vee$. Given a statement of the form $[\![\varphi \vee \psi]\!](s) \sqsupseteq \ell$, we apply the $\vee$-rule:

$$\frac{\begin{array}{c}\exists \ell_1, \ell_2 \in \mathcal{L} \cdot [\![\varphi]\!](s) \sqsupseteq \ell_1 \wedge \\ [\![\psi]\!](s) \sqsupseteq \ell_2 \wedge \ell_1 \sqcup \ell_2 \sqsupseteq \ell\end{array}}{[\![\varphi \vee \psi]\!](s) \sqsupseteq \ell} \;\vee\text{-rule}$$

Using the monotonicity of the $\sqcup$ operator, we get the following proposition.

Proposition 2 *Let φ, and ψ be χECTL formulas, let s be a state of a χKripke structure, and ℓ a lattice element, then:*

$$\begin{array}{ll} & \exists \ell_1, \ell_2 \in \mathcal{L} \cdot [\![\varphi]\!](s) \sqsupseteq \ell_1 \wedge [\![\psi]\!](s) \sqsupseteq \ell_2 \wedge \ell_1 \sqcup \ell_2 \sqsupseteq \ell \\ \Leftrightarrow & \exists \ell_1, \ell_2 \in \mathcal{L} \cdot [\![\varphi]\!](s) = \ell_1 \wedge [\![\psi]\!](s) = \ell_2 \wedge \ell_1 \sqcup \ell_2 \sqsupseteq \ell \end{array}$$

Using the fact that $[\![\varphi]\!](s) \triangleq \texttt{modelCheck}(\varphi, s)$, we get that the one-point rule is applicable if and only if instantiating ℓ_1 to $\texttt{modelCheck}(\varphi, s)$ and ℓ_2 to $\texttt{modelCheck}(\psi, s)$ does not result

(a)

proc `atomicOnePoint`(p, s, ℓ)
  $k :=$ `modelCheck`(p, s)
  if `qblat`$(k \sqsupseteq \ell)$ **then**
    apply one-point rule substituting k for ℓ_1
  else
    terminate with **invalid**
  end if
end proc

(b)

proc `orOnePoint`(φ, ψ, s, ℓ)
  $k_\varphi :=$ `modelCheck`(φ, s)
  $k_\psi :=$ `modelCheck`(ψ, s)
  if `qblat`$(k_\varphi \sqcup k_\psi \sqsupseteq \ell)$ **then**
    apply one-point rule substituting k_φ for ℓ_1, k_ψ for ℓ_2
  else
    terminate with **invalid**
  end if
end proc

(c)

proc `euOnePoint`(φ, ψ, s, ℓ)
  $i := 0$
  $eu =$ `modelCheck`$(E[\varphi U \psi], s)$
  $eui = \bot$
  while $eui \neq eu$ **and not** `qblat`$(eui \sqsupseteq \ell)$ **do**
    $eui :=$ `modelCheck`$(E[\varphi U_i \psi], s)$
    $i := i + 1$
  end while
  if `qblat`$(eui \sqsupseteq \ell)$ **then**
    apply one-point rule substituting i for n
  else
    terminate with **invalid**
  end if
end proc

(d)

proc `exOnePoint`(φ, s, ℓ)
  $k :=$ `modelCheck`$(EX\varphi, s)$
  if not `qblat`$(k \sqsupseteq \ell)$ **then**
    terminate with **invalid**
  end if
  $(r_1, p_1) \ldots, (r_n, p_n) :=$ `exWitness`(φ, s)
  apply one-point rule substituting (r_i, p_i) for (t_i, ℓ_i)
end proc

Figure 5.4: Algorithms for automatic proof generation.

in invalid statements. As in the previous case, this simplifies to requiring that $\ell_1 \sqcup \ell_2 \sqsupseteq \ell$ is valid for the instantiated values. Putting this together, we get the algorithm in Figure 5.4(b). The $\wedge$ operator is handled similarly.

We now examine the case of the unbounded EU operator. Given the statement $[\![E[\varphi U \psi]]\!](s) \sqsupseteq \ell$ we first apply the EU-rule:

$$\frac{\exists n \in nat \cdot [\![E[\varphi U_n \psi]]\!](s) \sqsupseteq \ell}{[\![E[\varphi U \psi]]\!](s) \sqsupseteq \ell}\ EU$$

Recall that the bounded until (EU_i), when viewed as a function on the bound, is monotone. Moreover, it is bounded above by the unbounded until (EU). Therefore, we can find the instantiation of n by a linear search. The algorithm for the application of one-point rule is given in Figure 5.4(c).

For example, consider the χKripke structure in the Figure 5.1, and assume that we want to prove that $[\![E[pUq]]\!](s_0) \sqsupseteq$ TT. After the application of the EU-rule, we get

$$\exists n \in nat \cdot [\![E[pU_nq]]\!](s_0) \ \sqsupseteq \ \text{TT}$$

To apply the one-point rule, we first try $[\![E[pU_0q]]\!](s_0) = [\![q]\!](s_0) =$ FT $\not\sqsupseteq$ TT. Then, increasing the bound, we get $[\![E[pU_1q]]\!](s_0) =$ TT, and therefore we can apply the one-point rule by instantiating n to 1.

Finally, let us examine the EX operator. Given the statement $[\![EX\varphi]\!](s) \sqsupseteq \ell$, first the EX-rule is applied:

$$\frac{\begin{array}{l}\exists \ell_1, \ldots, \ell_n \in \mathcal{L} \cdot \exists t_1, \ldots, t_n \in S \cdot [\![\mathbb{R}(s,t_1) \wedge \varphi]\!](t_1) \sqsupseteq \ell_1 \\ \wedge \cdots \wedge [\![\mathbb{R}(s,t_n) \wedge \varphi]\!](t_n) \wedge (\bigvee_{1 \leq i \leq n} \ell_i) \sqsupseteq \ell\end{array}}{[\![EX\varphi]\!](s) \sqsupseteq \ell} \ EX$$

Then we need to eliminate the existential quantifiers by applying the one-point rule, but first we establish the following proposition.

Proposition 3 *Let φ be a χECTL formula, s a state of a χKripke structure M, and $\mathbb{R}(s,t)$ its transition relation, and let* $\text{img} : S \to \mathcal{L}$ *be a function:* $\text{img}(t) \triangleq [\![\mathbb{R}(s,t) \wedge \varphi]\!](t)$. *Then,*

$$\begin{array}{ll} & \exists \ell_1, \ldots, \ell_n \in \mathcal{L} \cdot \exists t_1, \ldots, t_n \in S \cdot [\![\mathbb{R}(s,t_1) \wedge \varphi]\!](t_1) \sqsupseteq \ell_1 \\ & \wedge \cdots \wedge [\![\mathbb{R}(s,t_n) \wedge \varphi]\!](t_n) \wedge (\bigvee_{1 \leq i \leq n} \ell_i) \sqsupseteq \ell \\ \Leftrightarrow & \bigvee_{t \in S} \text{img}(t) \sqsupseteq \ell \end{array}$$

From this proposition, it follows that to apply the one-point rule, we simply need to find a subset $T = \{t_1, \ldots, t_n\}$ of S, such that $\bigvee \mathit{img}(T) \sqsupseteq \ell$, where $\mathit{img}(T) = \{f(t) \mid t \in T\}$, and let $\ell_i = \mathit{img}(t_i)$. A trivial solution is to let T equal S; however, such a solution is not practical due to the large size of S.

Instead, let us take a closer look at $\mathit{img}(S) = \{\ell_1, \ldots, \ell_n\}$. The size of this set, $|\mathit{img}(S)|$, is bounded above by the size of the lattice, which in most applications is reasonably small. Thus, we can construct the set T of size $|\mathit{img}(S)|$ by letting $t_i \in S$ be such that $\mathit{img}(t_i) = \ell_i$. Namely, we let t_i be an element of $\mathit{img}^{-1}(\ell_i)$.

Notice that unlike the previous cases, we no longer use the model-checker as a black-box. In order to apply the one-point rule for EX we require the model-checker either to produce the set T, or to provide enough information for us to compute it efficiently. Moreover, the set $T \subseteq S$, together with the original state s constitutes a witness for $[\![EX\varphi]\!](s) \sqsupseteq \ell$; therefore, essentially we require that the model-checker is able to produce a witness for EX.

We show that this requirement can be met efficiently in the case of a symbolic model-checker implemented using Multi-Valued Decision Diagrams (MDDs), such as χChek [CDG02]. In the framework of symbolic model-checking, the function *img* is represented by an MDD u_{img}, and *img*(S) is just the set of all terminal nodes reachable from the root node u_{img}. To improve efficiency we store the terminal nodes reachable from each node. This can be done at the expense of $|\mathcal{L}|$ bits per node. In this case, we can find *img*(S) in constant time. Finding $t \in$ *img*$^{-1}(\ell)$ for some $\ell \in$ *img*(S) can be done in time linear in the height of the diagram by traversing the diagram toward the terminal node ℓ.

The algorithm for the application of the one-point rule is given in Figure 5.4(d). The algorithm makes use of the function $\texttt{exWitness}(\varphi, s)$ that computes the witness for $[\![EX\varphi]\!](s) = \texttt{modelCheck}(EX\varphi, s)$. The witness is returned as a set of pairs (t_i, ℓ_i), such that *img*$(t_i) = \ell_i$, and $\bigvee_{1 \leq i \leq n} \ell_i = \texttt{modelCheck}(EX\varphi, s)$.

We conclude this section with an example for the EX operator. Assume we want to prove that $[\![EXq]\!](s_0) \sqsupseteq$ TT in the χKripke structure in Figure 5.1. First we apply the EX-rule and obtain:

$$\frac{\begin{array}{c}\exists \ell_1, \ldots, \ell_n \in \mathcal{L} \cdot \exists t_1, \ldots, t_n \in S \cdot [\![\mathbb{R}(s_0, t_1) \wedge q]\!](t_1) \sqsupseteq \ell_1 \\ \wedge \cdots \wedge [\![\mathbb{R}(s_0, t_n) \wedge q]\!](t_n) \sqsupseteq \ell_n \wedge (\bigvee_{1 \leq i \leq n} \ell_i \sqsupseteq \mathrm{TT}\end{array}}{[\![EXq]\!](s_0) \sqsupseteq \mathrm{TT}}\ EX$$

Next, we compute *img*$(S) = \{\mathrm{TF}, \mathrm{FT}\}$, and representatives from its inverse: $s_0 \in$ *img*$^{-1}$(FT), and $s_1 \in$ *img*$^{-1}$(TF). Finally, we apply the one point rule, and obtain:

$$[\![\mathbb{R}(s_0, s_0) \wedge q]\!](s_0) \sqsupseteq \mathrm{FT} \wedge [\![\mathbb{R}(s_0, s_1) \wedge q]\!](s_1) \sqsupseteq \mathrm{TF} \wedge \mathrm{FT} \sqcup \mathrm{TF} \sqsupseteq \mathrm{TT}$$

(a)

$$\cfrac{\cfrac{[\![\mathbb{R}(s,t'_1) \wedge \varphi]\!](t'_1) \sqsupseteq \ell_1 \qquad \cdots \qquad [\![\mathbb{R}(s,t'_n) \wedge \varphi]\!](t'_n) \sqsupseteq \ell_n \qquad \bigsqcup_{1 \le i \le n} \ell'_i \sqsupseteq \ell}{\begin{array}{l}\exists \ell_1, \ldots, \ell_n \in \mathcal{L} \cdot \exists t_1, \ldots, t_n \in S \cdot [\![\mathbb{R}(s,t_1) \wedge \varphi]\!](t_1) \sqsupseteq \ell_1 \wedge \cdots \wedge \\ [\![\mathbb{R}(s,t_n) \wedge \varphi]\!](t_n) \sqsupseteq \ell_n \wedge \bigsqcup_{1 \le i \le n} \ell_i \sqsupseteq \ell\end{array}}\ \text{one-point rule}}{[\![EX\varphi]\!](s) \sqsupseteq \ell}\ EX$$

(b)

$$\frac{[\![\mathbb{R}(s,t'_1) \wedge \varphi]\!](t'_1) \sqsupseteq \ell_1 \qquad \cdots \qquad [\![\mathbb{R}(s,t'_n) \wedge \varphi]\!](t'_n) \sqsupseteq \ell_n}{[\![EX\varphi]\!](s) \sqsupseteq \ell}$$

(c)

$$\begin{tikzcd}
{[\![\mathbb{R}(s,t'_1) \wedge \varphi]\!](t'_1) \sqsupseteq \ell_1} & \cdots & {[\![\mathbb{R}(s,t'_n) \wedge \varphi]\!](t'_n) \sqsupseteq \ell_n} \\
 & {[\![EX\varphi]\!](s) \sqsupseteq \ell} \arrow[ul, "{\mathbb{R}(s,t'_1)}"] \arrow[u] \arrow[ur, "{\mathbb{R}(s,t'_n)}"'] &
\end{tikzcd}$$

(d)

$$\begin{tikzcd}
t'_1 & \cdots & t'_n \\
 & s \arrow[ul, "{\mathbb{R}(s,t'_1)}"] \arrow[u] \arrow[ur, "{\mathbb{R}(s,t'_n)}"'] &
\end{tikzcd}$$

Figure 5.5: From proofs to witnesses.

5.4 Witness Generation

In this section we explore the connection between proofs for χECTL and witnesses for model-checking. We show that a witness can be extracted from a proof of $[\![\varphi]\!](s) \sqsupseteq \ell$, where $\ell \in \mathcal{L}$ is such that $[\![\varphi]\!](s) = \ell$. We also suggest a presentation for the proofs, centered around the structure of the witness.

Before considering witness extraction for general χECTL formulas, we examine the case of the EX operator only. Recall from Section 5.3 that application of a one-point rule to EX is essentially equivalent to computing a witness for it. The general form of the proof of validity of $[\![EX\varphi]\!](s) \sqsupseteq \ell$ is shown in Figure 5.5(a). This proof corresponds to a witness for EX, namely, a tree rooted at s, with children $t_1, \ldots, t_n$, where an edge from s to t_i is labeled by the value of $\mathbb{R}(s, t_i)$. This correspondence between the proof and the witness suggests a simple procedure for extracting the witness from the proof:

- remove all nodes from the proof tree except for the root node, and nodes that are result of the application of the one-point rule (see Figure 5.5(b)).
- replace horizontal bars by directed edges, and label each edge incoming into a node $[\![\mathbb{R}(s, t_i) \wedge \varphi]\!](t_i)$ by the value of $\mathbb{R}(s, t_i)$ (see Figure 5.5(c)).
- relabel the top node as s, and each node of the form $[\![\mathbb{R}(s, t_i) \wedge \varphi]\!](t_i)$ by t_i (see Figure 5.5(d)).

After applying the above procedure to the proof tree of EX we obtain the witness for EX.

In general, the proof tree for a statement $[\![\varphi]\!](s) \sqsupseteq \ell$ can be partitioned into the proof nodes that are a direct result of the application of the one-point rule to EX, and the rest. The application of the EX one-point rule directly corresponds to a step in a χKripke structure, whereas the rest of the nodes can be seen as the "glue" that binds all of the steps into a complete witness. The witness can be extracted from the proof tree by the same procedure used in the EX case. Given a proof tree, remove all of the nodes except for the root node, and the ones resulted from the application of the EX one-point rule, and then convert the remaining nodes into states.

As shown by the algorithm presented above, extracting witnesses from proofs amounts to hiding certain proof steps. Thus, proofs and witnesses are just two extremes in the presentation of the reasons behind the result of the model-checker to the user. The trade-off here is between size and ease of use. Proofs exhibit all of the reasoning steps explicitly, and therefore, make

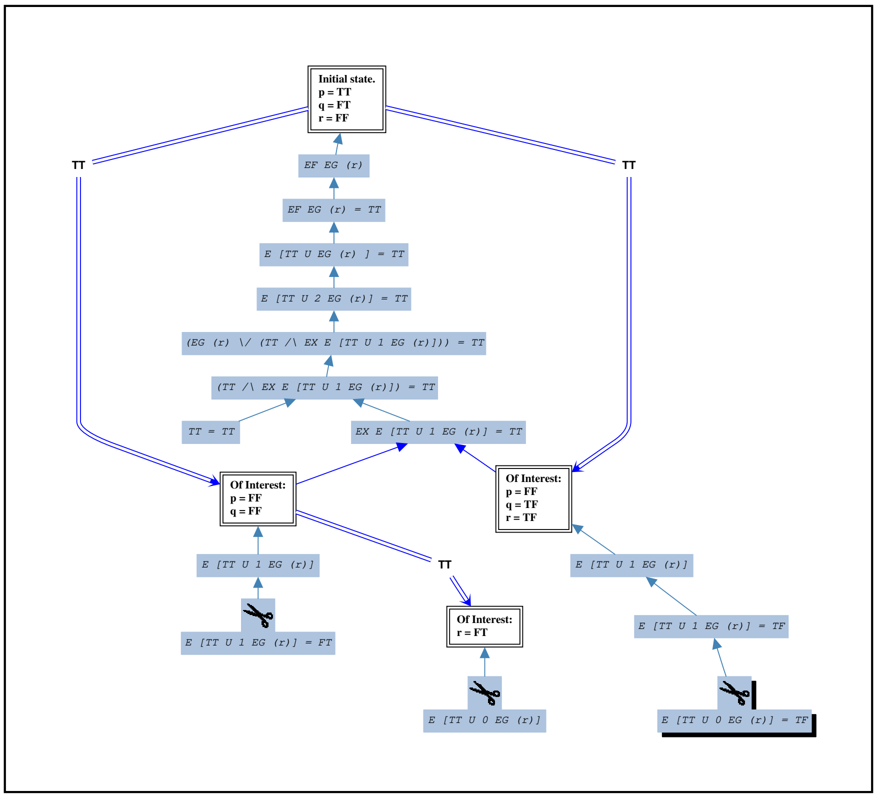

Figure 5.6: Snapshot of KegVis.

it easy for the user to follow each step. However, this excessive verbosity makes a proof much larger than a corresponding witness. On the other hand, witnesses only contains the axioms used by the model-checker, and therefore, require an intimate knowledge of the model-checking algorithm to be understood.

To leverage the advantages of both presentations we have developed and interactive witness browser tool — KegVis. Internally, the tool uses the proof view of the witness, while providing the witness view to the user. The user can than navigate the witness as usual, or expand various proof steps to get a better understanding of the witness. Alternatively, the information available in the proof can be used to navigate to different "points of interest".

A snapshot of KegVis, showing the proof for $P = EFEGr \sqsupseteq$ TT on the χKripke structure from Figure 5.1, is shown in Figure 5.6. Initially, the user is presented with the witness view of the proof, indicated by double-line nodes and arrows in the graph. Each state of the χKripke structure is represented by its evaluation of the atomic propositions, and for conciseness only the propositions that change from one state to the next are shown. For example, the root state is s_0 which evaluates r to FF, its left successor is the state s_2, and since the value of r does not change between s_0 and s_2 it is not shown.

Additionally, each state node is labeled with the χCTL formula whose proof depends on that state. The scissors symbol next to a graph node indicates that a subgraph from this node is hidden. Thus, the root node is required to prove $EFEGr$, and its left and right successors are required to prove $E[\top\ U_1\ EGr]$, etc.

The user of the tool can navigate the witness, or explore different proof parts by expanding them. For example, in the current snapshot the user has decided to fully expand the proof attached to the root node. The information gathered from the proof can be used to gain better understanding of the witness, and to support navigation through the witness. For example, examining the proof attached to the root node we see that our original formula is simplified to $E[\top\ U_2\ EGr]$, which means that each path in the witness must be no longer than three states long. For a more comprehensive description of different browsing strategies made possible by KegVis the reader is referred to [GC02].

5.5 Optimality, Minimality, and Finding the "Best" Witness

In previous sections we have shown that the proofs for χECTL formulas can be efficiently generated automatically, and that there is a one-to-one correspondence between proofs and witnesses. However, the downside of the technique presented so far, is that it treats all proofs (and witnesses) as equal. In practice, there is usually more than one witnesses for a given χECTL formula, with some being preferable to others.

For example, if the witness is generated as a refutation for the negation of the formula (i.e. it is a counter-example), then it is reasonable to assume that a smaller witness is preferred to a larger one. On the other hand, if the witness is generated as part of model-exploration, such as the case in query-checking [GDC02], a larger witness that explores an "interesting" part of

$$\dfrac{[\![p]\!](s_0) \sqsupseteq \text{TT} \qquad [\![q]\!](s_0) \sqsupseteq \text{FT} \qquad \text{TT} \sqcup \text{FT} \sqsupseteq \text{TT}}{[\![p \vee q]\!](s_0) \sqsupseteq \text{TT}} \; \vee\text{-rule}$$

Figure 5.7: A proof of $[\![p \vee q]\!](s_0)$.

the model is preferred to a smaller one. For example, we may prefer a witness that visits a designated state s, over a witness that does not.

Notice that the preference for a witness can be either global or local. A global preference refers to a complete witness, and the local preference specifies what is preferred at each decision point during the proof generation. For example, preferring a smaller witness over a larger one is a global preference. Its local counterpart is requiring that for each operator the part of the witness corresponding to it is the smallest possible one. Essentially, a local preference specifies a greedy optimization technique to be used during proof generation.

In this section, we show how the framework developed so far can be extended to allow one to explicitly specify local preferences. Intuitively, this can be achieved by strengthening the proof rules by embedding the preference in them. For the rest of the section, we assume that a locally smaller witness is preferred.

In general, a witness is a subtree of the computational tree of a χKripke structure. Thus, there are two dimensions along which it can be minimized: (a) the length of each path which is bounded above by the height of the witness, and (b) the breadth factor of each state on the witness. Unfortunately, even in the classical case, computing locally shortest witness for a fair EG operator was shown to be NP-hard [CGMZ95], thus we only concentrate on the EX, EU and non-temporal operators.

We begin by identifying the points in the proof where a local preference can be applied. These points must correspond to a place in the proof where it can branch along different paths. Thus, for χECTL these are identified by a disjunction in the formula. In what follows, we consider two cases separately: the $\vee$ operator itself, and its occurrence as part of the EX operator.

$$
\dfrac{
\dfrac{
\dfrac{[\![r]\!](s_1) \sqsupseteq \text{TF} \qquad \dfrac{\dfrac{[\![r]\!](s_1) \sqsupseteq \text{TF}}{[\![E[\top\ U_0\ r]]\!](s_1) \sqsupseteq \text{TF}}\ EU_0}{[\![EXE[\top\ U_0\ r]]\!](s_1) \sqsupseteq \text{TF}}\ EX}{[\![r \vee EXE[\top\ U_0\ r]]\!](s_1) \sqsupseteq \text{TF}}\ \vee\text{-rule}
}{[\![E[\top\ U_1\ r]]\!](s_1) \sqsupseteq \text{TF}}\ EU_i
\qquad
\begin{array}{c}[\![r]\!](s_3) \sqsupseteq \text{FT} \\ \vdots \\ [\![E[\top\ U_1\ r]]\!](s_2) \sqsupseteq \text{FT}\end{array}
}{
\dfrac{[\![E[\top\ U_2\ r]]\!](s_0) \sqsupseteq \text{TT}}{[\![E[\top\ U\ r]]\!](s_0) \sqsupseteq \text{TT}}\ EU
}\ EX
$$

Figure 5.8: Partial proof of $[\![E[\top\ U\ r]]\!](s_0)$.

Consider the proof of $P = [\![p \vee q]\!](s_0) \sqsupseteq$ TT for χKripke structure in Figure 5.1, shown in Figure 5.7. Clearly, since $[\![p]\!](s_0) =$ TT it is sufficient to justify P; however, our current $\vee$-rule requires us to exhibit that $[\![q]\!](s) \sqsupseteq$ FT as well. We solve this problem by strengthening the $\vee$-rule as follows:

$$
\dfrac{\exists \ell_1 \cdot [\![\varphi]\!](s) \sqsupseteq \ell_1 \wedge \ell_1 \sqsupseteq \ell}{[\![\varphi \vee \psi]\!](s) \sqsupseteq \ell}\ \vee\text{-rule}
$$

This ensures that our proof does not contain any unnecessary branches.

Next, consider the proof for $[\![E[\top\ U\ r]]\!](s_0) \sqsupseteq$ TT on the χKripke structure in Figure 5.1, a partial version of which is illustrated in Figure 5.8. The problematic leaf is the one labeled with

$$
[\![r \vee EXE[\top\ U_0\ r]]\!](s_1) \quad \sqsupseteq \quad \text{TF}
$$

Since both $[\![r]\!](s_1) =$ TF and $[\![EXE[\top\ U_0\ r]]\!](s) =$ TF, the proof generator makes a non-deterministic choice between what formula to expand. Thus, it potentially decides to expand the EU formula, leading to a longer witness. This particular problem is solved by introducing an addition heuristic, telling the proof generator to always resolve non-deterministic choice by picking the smallest formula.

However, in general, it is not possible to predict the size of the witness based solely on the form of the formula. A choice of a good heuristic typically depends on the additional domain and model knowledge. We leave the exploration and evaluation of various heuristics possible in this case for future work.

Finally, we consider the EX operator. Clearly, the height of the witness to $[\![EX\varphi]\!](s) \sqsupseteq \ell$ is 1. Moreover, the length of every branch of this witness is exactly 1. The breadth of the node s is determined by the value to which we instantiate n in the application of the EX one-point rule. Recall, that in Section 5.3, we have shown that n is bounded above by $|\mathit{img}(S)|$, where $\mathit{img}(x) \triangleq [\![\mathbb{R}(s,x) \wedge \varphi]\!](x)$; however, we can do better. For example, let us assume that $\mathit{img}(S) = \{\mathrm{TT}, \mathrm{TF}, \mathrm{FT}\}$, then $\vee \mathit{img}(S) = \mathrm{TT}$. Following the algorithm presented in Section 5.3, the witness is obtained by picking $t_1, t_2, t_3 \in S$, such that $\mathit{img}(t_1) = \mathrm{TT}$, $\mathit{img}(t_2) = \mathrm{TF}$, and $\mathit{img}(t_3) = \mathrm{FT}$. However, clearly the path s, t_1 is sufficient since $\mathit{img}(t_1) = \mathrm{TT} = \bigvee \mathit{img}(S)$. In general, the witness is given by $A \subseteq S$, such that $\bigvee \mathit{img}(A) = \bigvee \mathit{img}(S)$, and therefore, to minimize the breadth factor of the witness we must minimize $|A|$.

Definition 14 *Let B be a subset of a lattice $\mathcal{L}$, and $A \subseteq B$. Then, A is join-irredundant in B if and only if*

- $\bigvee A = \bigvee B$*, and*
- $\forall C \subset A \cdot \bigvee C \neq \bigvee B$

Thus, we can minimize $|A|$ as follows: let $K = \{k_1, \ldots, k_m\}$ be the minimal join-irredundant subset of $\mathit{img}(S)$, then $A = \{a_1, \ldots, a_m\}$, where $a_i \in \mathit{img}^{-1}(k_i)$.

5.6 Counter-Examples for χACTL

In this section we extend the proof system to handle counter-examples for χACTL. The following proposition identifies the connection between χACTL and χECTL.

Proposition 4 *Let φ be a χACTL formula. Then, there exists a χECTL formula ψ, such that* $[\![\neg\varphi]\!](s) = [\![\psi]\!](s)$

$$\frac{[\![EX\neg\varphi]\!](s_0) \sqsupseteq \ell}{[\![AX\varphi]\!](s_0) \sqsubseteq \ell}\ AX \qquad \frac{[\![E[\top U \neg\varphi]]\!](s) \sqsupseteq \ell}{[\![AG\varphi]\!](s) \sqsubseteq \ell}\ AG$$

$$\frac{[\![EX\neg\varphi]\!](s) \sqsupseteq \ell}{[\![\neg AX\varphi]\!](s) \sqsupseteq \ell}\ \neg AX \qquad \frac{[\![E[\top U \neg\varphi]]\!](s) \sqsupseteq \ell}{[\![\neg AG\varphi]\!](s) \sqsupseteq \ell}\ \neg AG$$

$$\frac{[\![EG\neg\psi \vee E[\neg\varphi U \neg\varphi \wedge \neg\psi]]\!](s) \sqsupseteq \ell}{[\![A[\varphi U \psi]]\!](s_0) \sqsubseteq \ell}\ AU$$

$$\frac{[\![EG\neg\psi \vee E[\neg\varphi U \neg\varphi \wedge \neg\psi]]\!](s) \sqsupseteq \ell}{[\![\neg A[\varphi U \psi]]\!](s_0) \sqsupseteq \ell}\ \neg AU$$

Figure 5.9: Proof rules for χACTL.

From Proposition 4 it follows that to prove $[\![\varphi]\!](s) \sqsubseteq \ell$ for a χACTL formula φ, it is sufficient to prove $[\![\psi]\!](s) \sqsupseteq \ell$ for a χECTL formula ψ corresponding to φ. The proof rules for χACTL based on this observation are shown in Figure 5.9.

Notice, that the rules for AX, EU, and AG operators introduce negation in front of their operands. Thus, it is necessary to introduce additional rules to handle negated versions of AX, EX, and EU.

The χACTL proof rules introduced here reduce proofs about χACTL to proofs about χECTL. Therefore, automated proof generation, as well as counter-example extraction from the proofs is exactly the same as in the case of χECTL.

5.7 Extension to Fair χECTL

In this section, we show the final extension of the proof system to fair version of χECTL.

Recall that the fair version of χECTL is obtained by defining a fair EG operator, and redefining the EX and EU operator through it. The proof rules for E_CX, and E_CU, shown in Figure 5.10, simply expand their definitions in terms of unfair version of the operators, and a

$$\frac{[\![EX(\varphi \wedge (E_C G\top \neq \bot))]\!](s) \sqsupseteq \ell}{[\![E_C X\varphi]\!](s) \sqsupseteq \ell}\ E_C X \qquad \frac{[\![E[\varphi U \psi \wedge (E_C G\top \neq \bot)]]\!](s) \sqsupseteq \ell}{[\![E_C[\varphi U \psi]]\!](s) \sqsupseteq \ell}\ E_C U$$

$$\frac{[\![(\varphi \wedge (F_1 \circ \cdots \circ F_k)(EXE[\varphi U \varphi \wedge \{s\}])) \vee (\varphi \wedge EXE_C G(\varphi \wedge \overline{\{s\}}))]\!](s) \sqsupseteq \ell}{[\![E_C G\varphi]\!](s) \sqsupseteq \ell}\ E_C G$$

Figure 5.10: Proof rules for fair χECTL.

fair EG.

To obtain the proof rule for $E_C G$ we need an equivalent of Theorem 17 from Section 5.2. Using the same intuition, we decompose $[\![E_C G\varphi]\!](s)$ into $E_C G$ restricted to all *fair* s to s paths, and fair paths without further occurrence of s. The result is summarized in Theorem 19, that is state here without proof.

Theorem 19 *Let φ be a χFECTL formula, s be a state of a χKripke structure, $C = \{c_1, \ldots, c_k\}$ be a set of fairness conditions, and F_i be defined as in Section 4.3. Then,*

$$[\![E_C G\varphi]\!](s) = [\![(\varphi \wedge (F_1 \circ \cdots \circ F_n)(EXE[\varphi U \varphi \wedge \{s\}])) \vee (\varphi \wedge EXE_C G(\varphi \wedge \overline{\{s\}}))]\!](s)$$

Finally, the proof rule obtained from Theorem 19 is given in Figure 5.10. The extension of the proof system to counter-examples for fair χACTL is trivial, and is omitted here.

5.8 Discussion and Future Work

In this chapter we have presented a technique for witness and counter-example generation for multi-valued model-checking. Unlike the mulit-valued model-checking algorithm, this technique is not a direct extension of a classical algorithm. Instead, it is based on the combination of the tableaux-based local model-checking algorithm [SS98] with the counter-example generation algorithm of Clarke et al. [CGMZ95, CLJV02]. In fact, the automated proof-generation of Section 5.3 can be seen as simulating a run of a local model-checker, where the information collected from the run of a global model-checker is used to guide the construction of the proof.

However, unlike Stevens et al. [SS98], we restrict our attention to χCTL, and use the insights provided by the counter-example generation algorithm of Clarke et al. [CGMZ95, CLJV02] to derive a specialized EG-rule.

The automated proof-generation algorithm presented in Section 5.3 makes use of a model-checker as a decision procedure. Alternatively, the same information can be extracted from the support sets of Tan and Cleaveland [TC02], or deductive proofs of Namjoshi [Nam01]. This makes it possible to use the technique presented here for interactive unrolling of deductive proofs (and support sets) into witnesses and counter-examples.

In this chapter we have concentrated on the technical issues surrounding counter-example and witness generation for multi-valued model-checking. We have only briefly discussed the potential of introducing an ordering on witnesses and identifying the best or most interesting witness. One interesting future direction for our work is to explore the interaction between the heuristics for picking the best witness and property patterns [DAC98]. Property patterns simplify users interaction with the model-checker by allowing the user to specify a property of interest without requiring an intimate knowledge of a temporal logic. Another advantage of property patterns is their close correspondence to user intentions. As such, it may be possible to construct heuristics to automatically produce a witness that the user considers to be most interesting, based on a particular property pattern used.

Chapter 6

Design of χChek

6.1 High-level Overview

χChek [CDG02] is a multi-valued symbolic decision diagram based model-checker. It is a generalization of an existing symbolic model-checking algorithm to χCTL. Both the multi-valued model-checking with fairness, described in Chapter 4, and the counter-example generator, described in Chapter 5, have been implemented as part of χChek. In this section we describe the design and current implementation of χChek.

Verification using χChek proceeds similarly to verification using a classical model-checker. Given a χCTL formula φ, and a multi-valued model M, χChek computes the result of $[\![\varphi]\!]$ on the initial state of M. The internal structure of χChek is given in Figure 6.1.

The model-checking of a property φ on a model M proceeds in several phases. First, the model is compiled into an mv-set based representation by a model compiler. The result of this phase is the symbol table describing the atomic propositions and lattice elements, an mv-set representation of the transition relation, and an mv-set encoding the initial state of M, respectively. Then, the symbol table along with the property φ are passed to the χCTL parser that builds a parse tree for φ. Once the parsing phase has been completed, the χChek engine applies the model-checking algorithm to the parse tree of the property, resulting in an mv-set

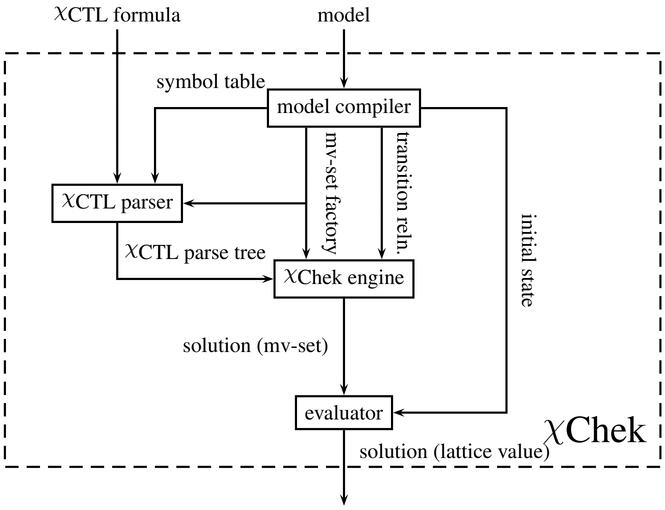

Figure 6.1: The internal structure of χChek.

representation of $[\![\varphi]\!]$. Finally, the evaluator restricts the result to the initial state, and returns the lattice element $\ell = [\![\varphi]\!](s_0)$.

The design of χChek is highly modularized to allow for easy extension and integration with other tools. For example, we have extended χChek to temporal logic queries (TLQ) [GDC02] by adding an implementation for a TLQ parser. Currently, we are in the process of integrating χChek with the SAL framework [BGL$^+$00] developed by SRI.

In the rest of this chapter we describe the design and the implementation of the χChek engine and the model compiler, as well as the data types used by them.

6.2 χChek Engine

The χChek engine implements the model-checking algorithm, illustrated in Figure 6.2. The entry point to the algorithm is the function `checkXCTL` that takes a parse tree of the formula p, and computes an mv-set representation of $[\![p]\!]$. `checkXCTL` uses the mv-set representation of the transition relation $\mathbb{R}$, and the parse tree for the formula obtained from the model compiler, and χCTL parser, respectively. Note that the transition relation $\mathbb{R}$ is used in the `backwardImage` computation. `checkXCTL` also makes use of two helper functions `checkEG`, and `checkEU` that

```
Global MvSet ℝ

function checkXCTL(XCTL p) : MvSet
    case p ∈ A: return projection(p)
    case p = ¬φ: return ptwiseApply(¬, checkXCTL(φ))
    case p = φ ∧ ψ: return ptwiseApply(∧, checkXCTL(φ), checkXCTL(ψ))
    case p = φ ∨ ψ: return ptwiseApply(∨, checkXCTL(φ), checkXCTL(ψ))
    case p = EX φ: return backwardImg(checkXCTL(φ))
    case p = E[φ U ψ]: return checkEU(checkXCTL(φ), checkXCTL(ψ))
    case p = EG φ: return checkEG(checkXCTL(φ))
end function

function checkEU(MvSet φ, MvSet ψ) : MvSet
    result_0 = constant(⊥)
    do
        result_{i+1} := ptwiseApply(∨, ψ, ptwiseApply(∧, φ, backwardImg(result_i, ℝ)))
    while result_i ≠ result_{i+1}
    return result_i
end function

function checkEG(MvSet φ) : MvSet
    result_0 = constant(⊤)
    do
        result_{i+1} := ptwiseApply(∧, φ, backwardImg(result_i, ℝ))
    while result_i ≠ result_{i+1}
    return result_i
end function
```

Figure 6.2: The model-checking algorithm.

carry out the fix-point computations of the corresponding χCTL operators, as defined in Section 2.6.

The model-checking algorithm implemented by χChek engine relies heavily on an mv-set library. The implementation details of the mv-set library are described below.

In what follows, the type `Oper` represents binary or unary logic operator, and `XCTL` represents a parse tree for a χCTL expression. `Var` represents an atomic proposition (a variable), where we adopt the following conventions: (a) for every variable v in the source state there is a corresponding variable v' in the destination state, and (b) there exists a variable ordering, such that each variable has a unique index in it. For the purposes of this presentation, we do not distinguish between a variable name and its index in the variable ordering. Type `Val` represents logic values, with a special value *null* used to denote a "don't care" value. `Vector` is a vector of elements of type `Val`, such that for any `Vector` $\vec{s}$ and any atomic proposition v, both $\vec{s}[v]$ and $\vec{s}[v']$ are defined. Finally, `MvSet` is a type representing an mv-set, i.e., a mapping between vectors and their values.

The mv-set library provides two interfaces: `MvSetFactory` to create new mv-sets, and `MvSet` to manipulate existing mv-sets.

Functions provided by the `MvSetFactory` interface are summarized below.

- Function `constant(Val` ℓ`)` returns an mv-set $\mathbb{S}$ where membership of each element is ℓ: $\forall \vec{x} \cdot \mathbb{S}(\vec{x}) = \ell$. For example, `constant`(TF) returns an mv-set where $\forall \vec{x} \cdot \mathbb{S}(x) = \text{TF}$.

- Function `projection(Var` v`)` creates an mv-set $\mathbb{S}$ where membership of each element is determined by the value of variable v.

- Function `point(Vector` $\vec{x}$, `Val` ℓ`)` returns an mv-set $\mathbb{S}$ in which $\vec{x}$ has membership ℓ, and all other elements have membership $\bot$: $\forall \vec{y} \cdot$ **if** $\mathbb{S}(\vec{y}) = \vec{x}$ **then** ℓ **else** $\bot$. For example, `point`((TT,TF,TF), FT), returns an mv-set where (TT,TF,TF) has value TF, while the remaining $4^3 - 1$ vectors have value FF.

Next, we summarize the functions provided by the `MvSet` interface.

- Function `ptwiseApply` (`Oper` *op*, `MvSet` $\mathbb{S}$) returns an mv-set $\mathbb{T}$ s.t. $\forall \vec{x} \cdot \mathbb{T}(\vec{x}) = op\ \mathbb{S}(\vec{x})$. Membership of each element in the mv-set returned by `ptwiseApply` (`Oper` *op*, `MvSet` $\mathbb{S}$, `MvSet` $\mathbb{T}$) is determined by a pairwise application of *op* to each element of $\mathbb{S}$ and $\mathbb{T}$.

- Function `cofactor(MvSet` $\mathbb{S}$, `Vector` $\vec{x}$`)` returns an mv-set $\mathbb{T}$ in which an element $\vec{y}$ has membership $\mathbb{S}(\vec{y})$ if it matches $\vec{x}$ and $\bot$ otherwise: $\forall \vec{y} \cdot$ **if** $\vec{y} = \vec{x}$ **then** $\mathbb{T}(\vec{y}) = \mathbb{S}(\vec{y})$ **else** $\mathbb{T}(\vec{y}) = \bot$.

- Function `backwardImg` (`MvSet` $\mathbb{S}$, `MvRelation` $\mathbb{R}$) returns an mv-set corresponding to $\overleftarrow{\mathbb{R}}(\mathbb{S})$ in Definition 8.

	Boolean-terminal	**Multi-terminal**
Boolean branching factor	BDD-vector	ADD
Multi-valued branching factor	MBTDD-vector	MDD

Table 6.1: Choices of decision diagram packages for implementing mv-sets..

The implementation of the mv-set library is built on top of several decision diagram packages. The choices we face with these is whether each node has two successors (boolean) or many successors (multi-valued), and whether the mv-set membership function is represented by a single multi-valued functions or by a collection of boolean functions. The first choice is referred to as *the branching factor*, and the second – deciding on the number of terminal nodes: *boolean-terminal* or *multi-terminal*. These choices are summarized in Table 6.1. The entries of the tables are the names of decision diagram packages supporting this implementation. For example, MDDs are multi-valued multi-terminal decision diagrams. When *boolean-terminal* diagrams are used, the mv-set membership function must be encoded by a collection of diagrams. We refer to this collection as a *decision diagram vector*. For example, BDD-vector refers to a representation where a collection of *binary decision diagrams* is used to encode a single mv-set membership function. The reader is referred to [CGD$^+$02] for an empirical evaluation of the different choices.

All four varieties of decision diagrams listed in Table 6.1 have been proposed in the literature: MDDs were first described by Srinivasan et al [SKMB90]. They included MBTDDs as a special case, but these are discussed in more detail by Sasao and Butler [SB96]. ADDs were proposed by Bahar et al [BFG$^+$93] (these are also known under the name MTBDDs [FMY97]), and BDDs were introduced by Akers [Ake78] and later by Bryant [Bry86], who suggested the added properties of reducedness and orderedness to guarantee canonicity. Technology for the diagrams with a branching factor of 2, i.e., ADDs and BDDs, is very mature and is supported by standard libraries such as CUDD [Som01].

The current implementation of χChek includes two distinct mv-set libraries. One, that supports all of the choices described above, and is based on a custom Java-based decision diagram package developed at the University of Toronto. In addition, χChek also includes an mv-set library implemented on top of the state-of-the art BDD and ADD implementations that are part of the CUDD [Som01] package.

6.3 Model Compiler

Unlike most model-checkers, χChek does not specify its own modeling language. Instead, it defines an abstract model compiler module that is responsible for transforming a model in a given modeling language into a form suitable for model-checking. This allows for greater flexibility: to add support for a new modeling language one simply needs to implement an appropriate model compiler for it.

Each model compiler must satisfy the following requirements: (a) it must be a Java bean [Mic97], (b) it must implement the compiler interface described below. The fact that each model compiler must be a Java bean allows adding new compilers to χChek at runtime.

A lifetime of a typical model compiler proceeds as follows. First, the user identifies the compiler by providing the name of the Java class implementing it. This class is then initialized and the Java Bean API is used to extract its properties. The user can then examine and modify the properties through a property editor dialog. Finally, the model-compiler is initialized with the new set of properties, executed, and the compiler interface is used to obtain information from it.

The compiler interface provides the following methods:

- Function `compile` () compiles the model and returns an `MvSet` representation of the transition relation.

- Function `getInitStates` () — returns an `MvSet` representation of the initial states of the model.

- Function `getSymbolTable` () — returns the symbol table describing the atomic propositions occurring in the model. The symbol table is represented by the standard Java `Map` class, mapping names of atomic propositions to their type. It is used to validate a χCTL formula during parsing, and is not described here any further.

The only mechanism available to a model compiler to communicate with the user is by advertising its properties through the `BeanInfo` interface. For example, a compiler that requires a name of the file containing the model advertises a `fileName` property. The model compiler is also responsible for picking the decision diagrams used to implement an mv-set. This can be done either by applying certain heuristics to find the type of decision diagrams best suited for the model, or by simply advertising an `mvset` property and letting the user make this decision. Since Java Bean Framework allows each bean to provide its own property editor, the compiler has full control over the interface used to elicit the values of the properties from the user.

It is also possible to chain compilers together to allow for various model transformations prior to model-checking. For example, NUSMV [CCGR99] provides a "forward search" option that given an initial state and a transition relation generates a bisimilar transition relation in which only reachable states have outgoing transitions. This transformation is known to significantly reduce model-checking time for some problems [McM93]. This can be achieved with χChek by defining a `ForwardSearchCompiler` that takes an arbitrary model compiler as an argument and applies the *forward search* algorithm on the transition relation generated by it. Other kind of model compositions can be accomplished by a similar technique. For example, it is possible to define a `ParallelCompositionCompiler` that performs synchronous or asynchronous composition of finitely many models, each compiled by an appropriate compiler.

The current implementation of χChek provides the following model compilers: `XMLModelCompiler`, `SMVModelCompiler`, and `TwoWayMergeCompiler`. The `XMLModelCompiler` is used to compile a χKripke structure specified as a directed graph in Graph eXchange Language (GXL) [HWS00]. This is the simplest method for providing a model to χChek, however, it is best suited for fairly small models, or for models generated automatically by other tools. The `SMVCompiler` is used to compile models specified in the input language of NuSMV [CCGR99]. It is primarily used for comparisons between χChek and NuSMV. Finally, the `TwoWayMergeCompiler` implements a multi-valued merge of two models that is useful for discovering feature interactions; the actual algorithm is fully described in [CDEG02].

Another interesting use of the model compiler framework is to use it to specify parametric models. For example, consider a model Lift of a simple elevator controller. This model can be easily specified abstractly, where the number of floors served by an elevator is specified by an external parameter. However, most modeling languages used by current model-checkers, such as the input language of NuSMV do not have enough expressive power to express this. Alternatively, it is possible to write a program that given the number of floors generates the actual concrete model in the input language of the model-checker that is than verified. The model compiler framework provides a much more elegant solution. In the case of the elevator controller, the model is represented by a model compiler Lift that contains a single user-controlled property `numberOfFloors`. When this compiler is initialized it elicits from the user the value of the `numberOfFloors` parameter, generates the `MvSet` representation of the transition relation of the elevator controller servicing the specified number of floors, and passes it directly to the χChek engine. The actual parametrized model can be either specified as a template in some modeling language, or can be coded directly in Java. As such, this approach allows to use the full power of the Java programming language for model generation.

6.4 Discussion and Related Work

In this chapter we have presented the design and the implementation of symbolic multi-valued model-checker χChek. The model checking algorithm used by χChek is based on symbolic fixpoint evaluation of monotone functions over mv-sets. An alternative approach to multi-valued model-checking was pioneered by Bruns and Godefroid [BG99], and later extended by Konikowska and Penczek [KP02]. The idea behind this approach is to reduce a multi-valued model-checking problem to several classical problems. However, to our knowledge, currently no tool support for this approach is available. At the same time, χChek provides an implementation that is very similar to the approach taken by Bruns and Godefroid. Namely, when the BDD-vector data structure is used to represent mv-sets, the algorithm used by χChek is equivalent to solving several classical model-checking problems in parallel.

The current implementation of χChek is applicable only to moderately sized problems. The major bottleneck is in our implementation of the decision diagram library used to support the `MvSet` data type. The current version of the decision diagram library is mostly a prototype and is lacking a lot of optimizations present in the state of the art decision diagrams libraries such as CUDD [Som01]. In the future, we plan to continue developing χChek to make it applicable to realistically sized problems.

Chapter 7

Conclusions and Future Work

7.1 Summary

This work addresses various aspects surrounding usability of multi-valued model-checking. We have significantly improved the previously known worst-case complexity bounds for multi-valued model-checking by showing that it is independent of the height of the lattice. The new complexity bounds are comparable to those of classical model-checking algorithms which serves as an indication of potential scalability of the multi-valued model-checking algorithm.

The multi-valued counterpart of fairness developed here is the first step in applying multi-valued model-checking to asynchronous concurrent systems. We have also developed a proof system for χCTL and shown how it can be used to express multi-valued witnesses and counter-examples. This is particularly important from the usability perspective since it provides a mechanism to express the reasons behind the result produced by the model-checker.

Finally, we have described the design and the implementation of a symbolic multi-valued model-checker χChek that was used to implement and validate our ideas. We belief that this work is the first step toward making multi-valued model-checking usable in practice.

7.2 Future Work

The work presented in this thesis as well as our experience with χChek show that multi-valued model-checking is in fact feasible. However, the technology is still far from being applicable to real-life systems. In the future, we plan to continue to refine and evaluate our approach.

The key issue to usability of multi-valued model-checking is identifying the types of problems best suitable for multi-valued modeling and analysis. One promising direction, pioneered by Bruns and Godefroid [BG99], is to look at a multi-valued model as an abstraction over a collection of classical models. This allows one to related the result of model-checking a property φ on a multi-valued model to classical models. For example, if A is a multi-valued model that is refined by A', then if φ evaluates to $\top$ on A it is guaranteed to evaluate to $\top$ on A'. The notion of refinement can also be used to approximate model compositions. For example, if A and B are two models, then C is their possible composition if and only if C refines both A and B. However, Bruns and Godefroid consider the idea of refinement only in the case when the logic is $\mathbf{3}$. In the future we plan to explore the consequences of extending this definition to the larger class of quasi-boolean logics.

Bibliography

[AB75] A.R. Anderson and N.D. Belnap. *Entailment. Vol. 1.* Princeton University Press, 1975.

[AG93] J.M. Atlee and J. Gannon. “State-Based Model Checking of Event-Driven System Requirements”. *IEEE Transactions on Software Engineering*, 19(1):22–40, January 1993.

[Ake78] S. Akers. “Binary Decision Diagrams”. *IEEE Trans. Computers*, C-27:509–516, 1978.

[BB92] L. Bolc and P. Borowik. *Many-Valued Logics*. Springer-Verlag, 1992.

[BC98] C. Baier and E. M. Clarke. “The Algebraic Mu-Calculus and MTBDDs”. In *Proceedings of the 5th Workshop on Logic, Language, Information and Computation, (WoLLIC’98)*, pages 27–38, 1998.

[BCHG+97] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan. “Symbolic Model Checking for Probabilistic Processes”. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440, Bologna, Italy, July 1997. Springer.

[Bel77] N.D. Belnap. “A Useful Four-Valued Logic”. In Dunn and Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 30–56. Reidel, 1977.

[BFG$^+$93] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. "Algebraic Decision Diagrams and Their Applications". In *IEEE /ACM International Conference on Computer-Aided Disign (ICCAD'93)*, pages 188–191, Santa Clara, California, 1993. IEEE Computer Society Press.

[BG99] G. Bruns and P. Godefroid. "Model Checking Partial State Spaces with 3-Valued Temporal Logics". In *Proceedings of Proceedings of 11th International Conference on Computer-Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287, Trento, Italy, 1999. Springer.

[BG00] G. Bruns and P. Godefroid. "Generalized Model Checking: Reasoning about Partial State Spaces". In C. Palamidessi, editor, *Proceedings of 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 168–182, University Park, PA, USA, August 2000. Springer.

[BG01] G. Bruns and P. Godefroid. "Temporal Logic Query-Checking". In *Proceedings of 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 409–417, Boston, MA, USA, June 2001. IEEE Computer Society.

[BGL$^+$00] S. Bensalem, V. Ganesh, Y. Lakhech, C. Munox, S. Owre, H. Rueb, J. Rushby, V. Rusu, H. Saidi, N. Shankar, E. Singerman, and A. Tiwari. "An Overview of SAL". In *Proceedings of LMF2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, June 2000.

[Bir67] G. Birkhoff. *Lattice Theory*. Americal Mathematical Society, Providence, RI, 3 edition, 1967.

[Bry86] R. E. Bryant. "Graph-based algorithms for boolean function manipulation.". *Transactions on Computers*, 8(C-35):677–691, 1986.

[BS01] J.C. Bradfield and C. Stirling. *Handbook of Process Algebra*, chapter "Modal Logics and μ-calculus: An Introduction ", pages 293–330. Elsevier, 2001.

[BvW98] R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Approach*. Springer-Verlag, 1998.

[CCGR99] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. "NUSMV: a new Symbolic Model Verifier". In N. Halbwachs and D. Peled, editors, *Proceedings of 11th Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 495–499, Trento, Italy, July 1999. Springer.

[CDE$^+$01] M. Chechik, B. Devereux, S. Easterbrook, A. Lai, and V. Petrovykh. "Efficient Multiple-Valued Model-Checking Using Lattice Representations". In K.G. Larsen and M. Nielsen, editors, *Proceedings of 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 451–465, Aalborg, Denmark, August 2001. Springer.

[CDEG02] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. "Multi-Valued Symbolic Model-Checking". CSRG Tech Report 448, University of Toronto, August 2002.

[CDG02] M. Chechik, B. Devereux, and A. Gurfinkel. "χChek: A Multi-Valued Model-Checker". In *Proceedings of 14th International Conference on Computer-Aided Verification (CAV'02)*, Lecture Notes in Computer Science, pages 505–509, Copenhagen, Denmark, July 2002. Springer.

[CEP01] M. Chechik, S. Easterbrook, and V. Petrovykh. "Model-Checking Over Multi-Valued Logics". In *Proceedings of Formal Methods Europe (FME'01)*, volume 2021 of *Lecture Notes in Computer Science*, pages 72–98. Springer, March 2001.

[CES86] 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.

[CGD+02] M. Chechik, A. Gurfinkel, B. Devereux, A. Lai, and S. Easterbrook. "Symbolic Data Structures for Multi-Valued Model-Checking". CSRG Tech Report 446, University of Toronto, January 2002.

[CGH+93] E.M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. Long, K.L. McMillan, and L.A. Ness. "Verification of the Futurebus+ Cache Coherence Protocol". In L. Claesen, editor, *Proceedings of 11th International Symposium on Computer Hardware Description Languages and their Applications*, North-Holland, April 1993.

[CGMZ95] E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. "Efficient Generation of Counterexamples and Witnesses in Symbolic Model Checking". In *Proceedings of 32nd Design Automation Conference (DAC 95)*, pages 427–432, San Francisco, CA, USA, 1995.

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[CLJV02] E.M. Clarke, Y. Lu, S. Jha, and H. Veith. Tree-Like Counterexamples in Model Checking. In *Proceedings of the Seventeenth Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 19–29, Copenhagen, Denmark, July 2002. IEEE Computer Society.

[DAC98] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. "A System of Specification Patterns". Patterns catalog is available at `http://www.cis.ksu.edu/\verb1~1dwyer/spec-patterns.html`. A collection of over 500 temporal properties is available at `http://www.cis.ksu.edu/\verb1~1dwyer/SPAT/SURVEY/ALL.raw`, 1998.

[Dil96] D.L. Dill. "The Murϕ Verification System". In R. Alur and T.A. Henzinger, editors, *Proceedings of 8th International Conference on Computer-Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393, New Brunswick, NJ, USA, 1996. Springer.

[Dun99] 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.

[Fit91] Melvin Fitting. "Many-Valued Modal Logics". *Fundamenta Informaticae*, 15(3-4):335–350, 1991.

[Fit92] Melvin Fitting. "Many-Valued Modal Logics II". *Fundamenta Informaticae*, 17:55–73, 1992.

[FMY97] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. "Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation". *Formal Methods in System Design: An International Journal*, 10(2/3):149–169, April 1997.

[Gai79] B. R. Gaines. "Logical Foundations for Database Systems". *International Journal of Man-Machine Studies*, 11(4):481–500, 1979.

[GC02] A. Gurfinkel and M. Chechik. "Proof-like Counter-Examples". Submitted for publication, October 2002.

[GDC02] A. Gurfinkel, B. Devereux, and M. Chechik. "Model Exploration with Temporal Logic Query Checking". In *Proceedings of SIGSOFT Conference on Foundations of Software Engineering (FSE'02)*, Charleston, South Carolina, November 2002. ACM Press. (to appear).

[GHJ01] P. Godefroid, M. Huth, and R. Jagadeesan. "Abstraction-based Model Checking using Modal Transition Systems". In K.G. Larsen and M. Nielsen, editors, *Proceedings of 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440, Aalborg, Denmark, 2001. Springer.

[Gin87] M. Ginsberg. "Multi-valued logic". In M. Ginsberg, editor, *Readings in Non-monotonic Reasoning*, pages 251–255. Morgan-Kaufmann Pub., 1987.

[Häh94] Reiner Hähnle. *Automated Deduction in Multiple-Valued Logics*, volume 10 of *International Series of Monographs on Computer Science*. Oxford University Press, 1994.

[Haz96] S. Hazelhurst. *Compositional Model Checking of Partially Ordered State Spaces*. PhD thesis, Department of Computer Science, University of British Columbia, 1996.

[Heh93] E.C.R. Hehner. *A Practical Theory of Programming*. Texts and Monographs in Computer Science. Springer-Verlag, New York, 1993.

[HJS01] Michael Huth, Radha Jagadeesan, and David A. Schmidt. "Modal Transition Systems: A Foundation for Three-Valued Program Analysis". In *Proceedings of 10th European Symposium on Programming (ESOP'01)*, volume 2028 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2001.

[Hol91] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[Hol97] G.J. Holzmann. "The Model Checker SPIN". *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[HP01] Michael Huth and Shekhar Pradhan. "Model-Checking View-Based Partial Specifications". *Electronic Notes in Theoretical Computer Science*, 45, November 2001.

[HR00] M. Huth and M. Ryan. *Logic in Computer Science: Modeling and Reasoning About Systems*. Cambridge University Press, 2000.

[Hut02] M. Huth. "Model-Checking Modal Transition Systems Using Kripke Structures". In *Proceedings of Third International Workshop on Verification, Model-Checking, and Abstract Interpretation*, Venice, Italy, January 2002.

[HWS00] Richard C. Holt, Andreas Winter, and Andy Schürr. "GXL: Towards a Standard Exchange Format". Technical Report 1–2000, Universität Koblenz-Landau, Institut für Informatik, Rheinau 1, D-56075 Koblenz, 2000.

[Kle52] S. C. Kleene. *Introduction to Metamathematics*. New York: Van Nostrand, 1952.

[KNPS00] M.Z. Kwiatkowska, G. Norman, D.A. Parker, and R. Segala. "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, 2000.

[Koz83] D Kozen. "Results on the Propositional μ-calculus". *Theoretical Computer Science*, 27:334–354, 1983.

[KP02] B. Konikowska and W. Penczek. "Reducing Model Checking from Multi-Valued CTL* to CTL*". In *Proceedings of 13 International Conference on Concurrency Theory (CONCUR'02)*, Lecture Notes in Computer Science, Brno, Czech Republic, August 2002. Springer.

[LT88] K.G. Larsen and B. Thomsen. "A Modal Process Logic". In *Proceedings of 3rd Annual Symposium on Logic in Computer Science (LICS'88)*, pages 203–210. IEEE Computer Society Press, 1988.

[Łuk70] J. Łukasiewicz. *Selected Works*. North-Holland, Amsterdam, Holland, 1970.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic, 1993.

[Mic77] R. S. Michalski. "Variable-Valued Logic and its Applications to Pattern Recognition and Machine Learning". In D. C. Rine, editor, *Computer Science and Multiple-Valued Logic: Theory and Applications*, pages 506–534. North-Holland, Amsterdam, 1977.

[Mic97] Sun Microsystems. "Java Bean API Specification". Available at `http://java.sun.com/beans`, 1997.

[Nam01] K. Namjoshi. Certifying Model Checkers. In *Proceedings of 13th International Conference on Computer-Aided Verification (CAV'01)*, volume 2102 of *LNCS*. Springer-Verlag, 2001.

[OSR93] S. Owre, N. Shankar, and J. Rushby. "User Guide for the PVS Specification and Verification System (Draft)". Technical report, Computer Science Lab, SRI International, Menlo Park, CA, 1993.

[Ras78] H. Rasiowa. *An Algebraic Approach to Non-Classical Logics. Studies in Logic and the Foundations of Mathematics*. Amsterdam: North-Holland, 1978.

[SB96] T. Sasao and J.T. Butler. "A Method to Represent Multiple-Output Switching Functions Using Multi-Valued Decision Diagrams". In *Proceedings of IEEE International Symposium on Multiple-Valued Logic (ISMVL'96)*, pages 248–254, Santiago de Compostela, Spain, 1996.

[SKMB90] A. Srinivasan, T. Kam, S. Malik, and R.E. Brayton. "Algorithms for Discrete Function Manipulation". In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'90)*, pages 92–95, Santa Clara, CA, USA, November 1990. IEEE Computer Society.

[Som01] F. Somenzi. "CUDD: CU Decision Diagram Package Release", 2001.

[SRW99] M. Sagiv, T. Reps, and R. Wilhelm. "Parametric Shape Analysis via 3-Valued Logic". In *Proceedings of 26th Annual ACM Symposium on Principles of Programming Languages*, pages 105–118, New York, NY, 1999. ACM.

[SS98] P. Stevens and C. Stirling. "Practical Model-Checking using Games". In B. Steffen, editor, *Proceedings of the 4th International Conference on Tools and algorithms for the construction and analysis of systems (TACAS '98)*, volume 1384, pages 85–101, New York, NY, USA, 1998. Springer-Verlag.

[SS00] Viorica Sofronie-Stokkermans. Automated theorem proving by resolution for finitely-valued logics based on distributive lattices with operators. *Multiple-Valued Logic*, 2000.

[TC02] L. Tan and R. Cleaveland. Evidence-Based Model Checking. In *Proceedings of 14th Conference on Computer-Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 455–470, Copenhagen, Denmark, July 2002. Springer-Verlag.

[Weg00] I. Wegener. *"Branching Programs and Binary Decision Diagrams: Theory and Applications"*. Monographs on Discrete Mathematics and Applications. SIAM, 2000.

[Zad87] L.A. Zadeh. "Fuzzy Sets". In R. R. Yager, S. Ovchinnikov, R. M. Tong, and H. T. Nguyen, editors, *Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh*, pages 29–44, New York, 1987. John Wiley & Sons, Inc.