August 2005

An Introduction to Multi-Valued Model Checking

Georgios E. Fainekos
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-05-16.

An Introduction to Multi-Valued Model Checking

Abstract

Nowadays computer systems have become ubiquitous. Most of the resources in the development of such systems, and especially in the fail-safe ones, are allocated into the simulation and verification of their behavior. One such automated method of verification is model checking. Given a mathematical description of the real system and a specification usually in the form of temporal logics, a model checker verifies whether the specification is satisfied on the model of the system. Recently, a multi-valued extension to the classical model checking has been proposed. In this approach both the model of the system and the specification take truth values over lattices with more then just two values. Such an extension enhances the expressive power of temporal logics and allows reasoning under uncertainty. Some of the applications that can take advantage of the multi-valued model checking are abstraction techniques, reasoning about conflicting viewpoints and temporal logic query checking. In this paper, we present three different approaches to the multi-valued model checking problem. The first is a reduction from multi-valued CTL* to CTL*, the second a multi-valued CTL symbolic model checking algorithm and, finally, a reduction technique from multi-valued μ-calculus to the classical one.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-05-16.

An Introduction to Multi–Valued Model Checking

Georgios E. Fainekos

Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19104, USA
E-mail: fainekos (at) grasp.cis.upenn.edu

Technical Report MS-CIS-05-16

August 15, 2005

Abstract

Nowadays computer systems have become ubiquitous. Most of the resources in the development of such systems, and especially in the fail-safe ones, are allocated into the simulation and verification of their behavior. One such automated method of verification is model checking. Given a mathematical description of the real system and a specification usually in the form of temporal logics, a model checker verifies whether the specification is satisfied on the model of the system. Recently, a multi-valued extension to the classical model checking has been proposed. In this approach both the model of the system and the specification take truth values over lattices with more then just two values. Such an extension enhances the expressive power of temporal logics and allows reasoning under uncertainty. Some of the applications that can take advantage of the multi-valued model checking are abstraction techniques, reasoning about conflicting viewpoints and temporal logic query checking. In this paper, we present three different approaches to the multi-valued model checking problem. The first is a reduction from multi-valued CTL* to CTL*, the second a multi-valued CTL symbolic model checking algorithm and, finally, a reduction technique from multi-valued μ-calculus to the classical one.

Index Terms

Survey, multi-valued, temporal logic, model checking.

I. Introduction

A. A Short Historical Review of (Multi-Valued) Logics

Logic can be a powerful tool for concretely describing formal statements and, more importantly, for reasoning over them. The development of the foundations of logic in philosophy and mathematics is attributed to Aristotle from Stagira (384-322 B.C.). In his book, the *Metaphysics 4* he declares and defends the following principles [58]:

1) An assertive sentence is true if and only if what it asserts is the case.
2) Every assertive sentence is either true or false.
3) Two contradictory assertive sentences cannot both be true.
4) Two contradictory assertive sentences cannot both be false.

Note that in the modern propositional logic with the introduction of negation as an operation, the last three statements become equivalent. The last principle is known as the *law of the excluded middle*, while the second one is the famous *principle of bivalence* which is the basis of all modern formal reasoning. One may claim that theoretical computer science is based on the principle of bivalence and the classical propositional logic and that on the other hand propositional logic has been influenced by the developments in the theoretical computer science (computability issues, finite model theory etc). Nowadays that computing is ubiquitous, the tremendous importance of the world of bivalence, that is the world of 0's and 1's, becomes more apparent as both computer hardware and software design are based on this principle.

Even though Aristotle's theory seems to be well defined and articulated on solid grounds, it comes with its own problems. Aristotle himself in *On Interpretation 9* developed a sequence of arguments that made the occurrence of every event in the world necessary. The argument is as follows [58]:

1) Let us claim that a state of affairs p is the case at the present time t_n.

2) Then at each moment in the past, it was true to affirm that p will be true at t_n.
3) Therefore at each moment in the past, it was not the case that p does not occur at t_n.
4) Therefore at each moment in the past, it was not possible that p will not be the case at t_n.
5) Hence, at each moment in the past, it was necessary that p will be the case at time t_n

This theory is called *Logical Determinism* and it states that every event in the real world is predetermined from eternity and that there is no moment in history that the occurrence of the event is undecided. Of course Aristotle and many others were not satisfied with such a possibility as this makes the judgement of the people obsolete and their actions predetermined. Thus, Aristotle revisited his principles and decided to restrict their scope to past and present events as well as to future events whose occurrence was certain (for example a moon eclipse). Other researchers, and especially Lukasiewicz, believe that Aristotle had already suggested the existence of a third logical value that was neither true nor false. For example, the infamous sentence "*There is going to be a sea-battle tomorrow*" takes this third logical value as we are not sure today whether there is going to be a battle tomorrow.

Even though this issue raised a fierce debate among philosophers, an appropriate framework for many valued logics was not developed until the 1920's when Lukasiewicz established the principles of a three-valued propositional calculus and Post defined the finite many-valued logical algebras. But there were still problems with the interpretation of the third logical value of Lukasiewicz, until in 1988 Nowak proved the formal correctness of the interpretation over De Morgan lattices instead of Boolean algebras. The reader who is interested in the subject can find an excellent introduction in [50].

Multi-valued logics have a wide range of applications from control (in terms of fuzzy logic [52]) to robotics [56] and from philosophy [50] (by providing tools for proving the independence of axioms and by formalising the intensional functions) to computer hardware and software design. To that end not many people know that there was an attempt to design computers based on multi-valued logics by both the former Soviet Union (SETUN in 1958) and the United States (TERNAC in 1973), but they were abandoned as they did not demonstrate any clear advantages compared to the binary computers. A look at the proceedings of the IEEE Computer Society on multi-valued logics [1] will convince the reader of the active scientific interest in the field of computer hardware and software design using multi-valued logics. The most recent application of multi-valued logics to the latter is the multi-valued model checking [10], [12], [40].

B. (Multi-Valued) Model Checking

In a nutshell, *model checking* [19] is the algorithmic procedure for testing whether a specification formula holds over some semantic model. The model of the system is usually given in the form of a discrete transition system (Kripke structure - See Section II-D), timed automaton [3] or hybrid system [2]. The specification is usually a simple reachability or safety query or a formula in the form of temporal or modal logics (for example LTL or CTL [23] or μ-calculus [43]). Temporal and modal logics can capture specifications that depend on the evolution of time (both past and future). The interested reader can find a very succinct and informative introduction to the model checking problem in [4].

Model checking is of tremendous importance to the software engineering community as well as to the designers of protocols and specifications. The latter might sound too general, but it actually is as broad as the title implies. In this category, we can include hardware designs, communication protocols, medical devices, avionics, mobile robots etc. As the scientific field of formal verification is by now well established, we do not give references to individual applications, but we would like to point the reader to the textbooks [19], [4], [36]. To motivate, though, the interest of the reader in the methods of formal verification and especially to model checking, we would like to point out several applications of these techniques to aerospace problems [47], [35], [53] that we believe designate the importance of the methods.

In multi-valued model checking both the atomic propositions of the model and the specification are not interpreted any more using the usual 1 (true) and 0 (false) values, but instead they are evaluated over a lattice. Informally, a lattice is a set of objects equipped with some ordering relation which does not have to be total. For example, the classical two valued logic can be interpreted over the elements of the lattice $\mathfrak{L}_2$ (see Fig. 1). Also, consider the lattice $\mathfrak{L}_3$ with values $\{0, 1/2, 1\}$, then we can give the following interpretation to the values: 1 is "true", $1/2$ is "maybe" or "possible" or "undetermined" (not enough information available) and 0 is "false".

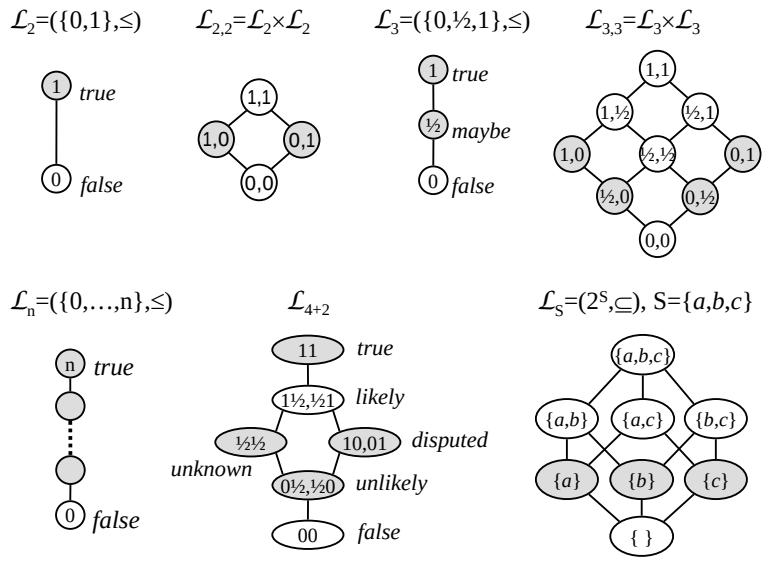

Fig. 1. Some lattices of practical interest. The gray nodes denote the join irreducible elements of each lattice (See Section II-A).

Another interesting lattice is the $\mathfrak{L}_{2,2}$ which can be used to model two opposing parties. In this case, a result like $(1, 0)$ would mean that the first party believes that the specification is correct while the second that it is false. Thus, there exists a preference or a ranking on the elements of the lattice which can be interpreted according to the application at hand. The interpretation of the elements of the lattice is one of the drawbacks of multi-valued model checking. One has to be very well acquainted with multi-valued logics in order to "tame" their additional expressive power.

On the other hand, the main advantages of multi-valued model checking over the traditional two valued model checking can be summarised as follows. Multi-valued model checking can be useful for reasoning about very large [6], [13] (or even infinite) state spaces using various abstractions. For example, the unexplored part of the state space can be mapped to a dummy state in which the atomic propositions take a third logical value which is interpreted as "undetermined". Then a multi-valued model checker is employed which returns "true" or "false" if the specification is satisfied or not and "undetermined" if it is necessary to explore a larger part of the state space (for further details see Section VI-D). Also, multi-valued model checking can handle uncertainty in both the model of the system and the specification. The uncertainty in the model can occur due to incomplete information about the real system. Furthermore, the user might need to know in what degree the specification is satisfied. For example consider the case where N processes must reach a consensus. If the logic has five values: "*everyone agreed*", "*agreement is likely*", "*no information*", "*agreement unlikely*" and "*nobody agreed*", then we can figure out whether the majority of the processes has agreed.

One of the first applications of 3-valued model checking appears in [5] where it is utilised in the qualitative simulation of continuous dynamical systems [59]. In this framework, temporal logics are employed in order to constrain the state space of the simulations, define the behavior of input variables and refine the trajectories generated by the qualitative differential equations. The model checking is done on the fly and it is interleaved with the qualitative simulation. Hence, as it is the case with the on-the-fly model checking, a temporal logic formula evaluates to the third logical value $1/2$ (undetermined) when there is not enough information to derive its actual value. Another source of ambiguity comes from the evaluation of the atomic propositions. For example, an atomic proposition might state that $x \leq 3$, but the qualitative simulation indicates that x is within the interval $[2, 4]$. In this case, the atomic proposition would evaluate to $1/2$ (undetermined), whereas if x were in the interval $[0, 1]$, it would evaluate to 1 (true).

Another interesting application of multi-valued model checking is the study of conflicting viewpoints or multiple viewpoint checking [22], [14], [38]. In this framework, models that describe inconsistent or conflicting viewpoints are merged together and a multi-valued model checker is employed to check whether the partial disagreements affect the required global specifications expressed, again, in temporal logics. Finally, one last application is the temporal logic query checking [8], [11], [34], [15]. Here, the elements of the lattice are sets of propositional

formulas and the goal of the method is to discover properties of the system. For example, if we want to find out the invariants of the model, we perform a query on the system of the form $AG?_\phi$ (meaning: what is the property that holds on all paths and at all times starting from a state s?). If the query is successful, then we get back a formula ϕ which satisfies the CTL formula $AG\phi$ at state s.

The multi-valued model checking algorithms are mainly divided into two categories. The first category includes all the algorithms that reduce the multi-valued problem to a set of classical two-valued model checking problems [7], [9], [32], [13], [39], [38]. For example, the model checking problem over the lattice $\mathfrak{L}_3$ can be reduced to two classical two-valued model checking problems. The other category includes the algorithms that directly attack the problem. To the author's best knowledge, there exist only two approaches in this category. The first is an extension of the CTL symbolic model checking algorithm [19] to the multi-valued case[1] [31], [12], while the second is based on an extension of alternating automata on trees [44], again, to the multi-valued case [10], [9].

The obvious advantage of the reduction methods is that they can readily use the existing model checking algorithms. Hence, any improvement in the performance of the classical model checkers implies a similar improvement to the multi-valued ones. Even though the reduction techniques appear at first to not be able to handle infinite lattices[2] as they would require the execution of an infinite number of classical model checking problems, this is not entirely true. Every finite Kripke structure has a finite number of atomic propositions which can take a finite number of values over the lattice. If we restrict the infinite lattice to these values, then we can apply the reduction methods. On the other hand, direct methods can be optimized for the multi-valued model checking problem.

Another classification of the multi-valued model checking algorithms can be obtained when we consider the logic under investigation, i.e. CTL*, LTL, CTL and μ-calculus. For example the following papers target the logic μ-calculus [10], [9], [41], [32]. To the authors best knowledge, the only multi-valued model checker for LTL is the work of Chechik et al [13]. This multi-valued model checker has fairly limited scope as it allows only two-valued transitions in the Kripke structure and requires totally ordered sets for the atomic propositions. The papers [42], [39], [40] discuss the problem of multi-valued CTL* model checking. Finally, a direct method for CTL model checking and the accompanying software platform are presented in the papers [12], [16], [31].

In this paper, we focus our attention mainly on [42], [12], [9] out of the growing literature on multi-valued model checking. These three papers actually span most of the range of existing methods. The presentation of the material starts in Section II with an introduction to order theory, lattices and algebras as these are the basis of both multi-valued model checking and μ-calculus and CTL symbolic model checking. Also, an introduction to Kripke structures and fix-point functions is necessary. In Section III, we present the CTL* temporal logic and its fragments as well as the μ-calculus. The next section, deals with the CTL symbolic model checking and automata theoretic model checking as they are both necessary for model checking CTL*. Another more elegant method to model check CTL* is to use the μ-calculus model checking algorithm.

In Section V, we introduce multi-valued sets, relations and Kripke structures. The next section deals with the multi-valued model checking problem. Starting from [42], we present a method for reducing the multi-valued CTL* model checking problem to the two valued case. [12] develops an extension of the classical two-valued CTL symbolic model checking algorithm to the multi-valued CTL case. Even though [9] introduces a reduction and a direct algorithm, we present only the reduction method due to space limitations. Finally, the paper concludes with a discussion and directions for future research.

II. Background

In this section, we present several mathematical notions that we will be using in the following. Even though most of these formalisms are well known to the mathematically versed reader, we chose to present them mostly for defining a uniform notation throughout this document and, also, as a brief introduction for some readers. This section is structured as follows. We start by presenting ordered sets and lattices which play a very important role in the multi-valued logics as well as in μ-calculus and, then, we continue with the classical notion of Kripke structures (by classical we mean not multi-valued). Several useful lemmas and theorems are presented in Appendix A.

[1]Actually, the first 3-valued CTL model checking algorithm was based on explicit state manipulation and it is presented in [6].

[2]This could be the case for example when the atomic propositions take values over the dense interval $[0, 1]$.

A. An Introduction to Lattices

Informally, an ordering imposes a comparison among objects of a set according to some property. It is a very fundamental notion in mathematics and it is omnipresent in our daily life. For example, we say that "George is older then John" or that "Two first cousins have a common grandfather". An ordering is also antisymmetric and transitive. Hence, by saying that "John is older then Mike" we know that "Mike is not older then John" and that "George is older then Mike" as well. Also notice that according to the property under question, it is not necessary that two objects can be compared.

Definition 2.1 (Quasi-order): Let S be a set. A *quasi-order* is a binary relation $\sqsubseteq$ such that for all $x, y, z \in S$ the following properties hold:

$$x \sqsubseteq x \qquad \textit{(reflexivity)}$$
$$x \sqsubseteq y \text{ and } y \sqsubseteq z \text{ imply } x \sqsubseteq z \qquad \textit{(transitivity)}$$

Definition 2.2 (Partial order): A *partial order* $\sqsubseteq$ on S is a quasi-order such that for all $x, y, z \in S$ the following additional property holds:

$$x \sqsubseteq y \text{ and } y \sqsubseteq x \text{ imply } x = y \qquad \textit{(antisymmetry)}$$

A *partially ordered set* (sometimes also called *ordered set* or *poset*) will be denoted by the pair $\mathfrak{S} = (S, \sqsubseteq)$. Whenever the set S is the set of reals ($real$) or the set of natural numbers (nat), then we will denote the order relation by $\leq$.

Definition 2.3 (Linear order): Let $\mathfrak{S}$ be a poset. If the following condition holds, then the pair $(S, \sqsubseteq)$ is called *linear ordered set* (or *totally ordered set* or *chain*) and the binary relation $\sqsubseteq$ is called *total order*:

$$\text{for all } x, y \in S \text{ it is either } x \sqsubseteq y \text{ or } y \sqsubseteq x$$

that is, there do not exist incomparable elements in the set S. Two elements x, y of a set S are incomparable $x \parallel y$ iff $x \not\sqsubseteq y$ and $y \not\sqsubseteq x$. An ordered set $\mathfrak{S}$ is an *anti-chain* if all the elements are pairwise incomparable, i.e. $(\forall x, y \in S).(x \parallel y)$.

If there exists an element $\bot \in S$ such that $(\forall x \in S).(\bot \sqsubseteq x)$, then we will call that element *bottom*. Similarly, if there exists $\top \in S$ such that $(\forall x \in S).(x \sqsubseteq \top)$, then $\top$ is called *top*. A finite chain always has a top and a bottom element. As an example, consider the powerset $\mathcal{P}(X)$ (i.e. the set of all subsets of X or $\mathcal{P}(X) = 2^X$) of a finite set X, then the set inclusion relation $\subseteq$ induces a partial order on $\mathcal{P}(X)$. Let $A, B \in \mathcal{P}(X)$ and define $A \sqsubseteq B$ iff $A \subseteq B$. The top and bottom elements of the poset $(\mathcal{P}(X), \sqsubseteq)$ are $\bot = \emptyset$ and $\top = X$. The same is not true for infinite chains: the set of natural numbers nat has as bottom element the zero $\bot = 0$, but it has no top element.

Let $(S, \sqsubseteq)$ be a poset and $X \subseteq S$. If there exists some $u \in S$ such that $x \sqsubseteq u$ for all $x \in X$, then u is called an *upper bound* of X. The set of all upper bounds will be denoted by X^u, i.e.

$$X^u = \{u \in S \,|\, (\forall x \in X).(x \sqsubseteq u)\}$$

If the set X^u has a least element u, then this is called the *least upper bound* or *supremum* of X and it is denoted by $sup(X)$. If the least upper bound exists, then it is unique (by the antisymmetry of the ordering relation $\sqsubseteq$). The least upper bound exists iff there exists $u \in S$ such that

$$(\forall s \in S).((\forall x \in X).(x \sqsubseteq s) \leftrightarrow u \sqsubseteq s)$$

Dually, we can define the *lower bound* l, i.e. $(\forall x \in X).(l \sqsubseteq x)$, the set X^l of all lower bounds

$$X^l = \{l \in S \,|\, (\forall x \in X).(l \sqsubseteq x)\}$$

and the *greatest lower bound* or *infimum* of X (denoted by $inf(X)$). The greatest lower bound exists iff there exists $l \in S$ such that

$$(\forall s \in S).((\forall x \in X).(s \sqsubseteq x) \leftrightarrow s \sqsubseteq l)$$

Definition 2.4 (Order-preserving, -embedding, -isomorphism): Let $(X, \sqsubseteq_X)$ and $(Y, \sqsubseteq_Y)$ be posets and let $f : X \to Y$ be a map, then f is called:

- *order-preserving* (or *monotone*) when: $(\forall x_1, x_2 \in X).(x_1 \sqsubseteq_X x_2 \to f(x_1) \sqsubseteq_Y f(x_2))$
- *order-embedding* when: $(\forall x_1, x_2 \in X).(x_1 \sqsubseteq_X x_2 \leftrightarrow f(x_1) \sqsubseteq_Y f(x_2))$
- *order-isomorphism* when f is an order-embedding that maps X onto Y

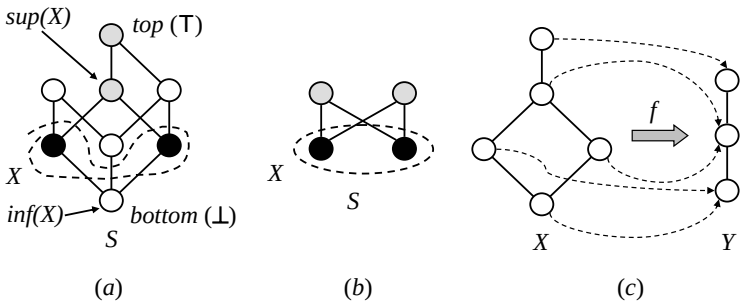

Fig. 2. (a) A poset S. The black nodes denote the set X and the gray the set X^u. (b) Note that the set X in this case does not have a supremum. (c) An example of an order preserving map f from X to Y.

Here, we give the definitions of two important families of sets which we will be using later.

Definition 2.5 (Down-set and Up-Set): Let $(S, \sqsubseteq)$ be an ordered set and $X \subseteq S$, then:

- X is called a *down-set* (or *order ideal*) if $x \in X$, $s \in S$ and $s \sqsubseteq x$ implies $s \in X$
- X is called an *up-set* (or *order filter*) if $x \in X$, $s \in S$ and $s \sqsupseteq x$ implies $s \in X$

Let us define the following notation for $X \subseteq S$ and $x \in S$:

$$X^{\downarrow} := \{s \in S \mid (\exists x \in X).(s \sqsubseteq x)\} \quad \text{and} \quad X^{\uparrow} := \{s \in S \mid (\exists x \in X).(s \sqsupseteq x)\}$$
$$x^{\downarrow} := \{s \in S \mid s \sqsubseteq x\} \quad \text{and} \quad x^{\uparrow} := \{s \in S \mid s \sqsupseteq x\}$$

Note that X is a down-set iff $X = X^{\downarrow}$. The sets $x^{\downarrow}$ and $x^{\uparrow}$ are called principal.

In the following, we state several definitions about lattices. The interested reader can find a detailed introduction to this subject in [21]. Lattices are of fundamental importance to both fixpoint calculi and many-valued logics. In order to formally introduce lattices, we need to define the functions *join* $\sqcup : S \times S \to S$ and *meet* $\sqcap : S \times S \to S$ as:

$$x \sqcup y := sup(\{x, y\}) \text{ and } x \sqcap y := inf(\{x, y\})$$

A very important fact for the following analysis is that the join and meet functions are order preserving (see Lemma A.2). Also for this and the following sections, we adopt the following notation. For a poset S:

$$\bigsqcup S := sup(S) \quad \text{which is read as join of } S$$
$$\bigsqcap S := inf(S) \quad \text{which is read as meet of } S$$

It is straightforward to prove that if $(S, \sqsubseteq)$ is a poset, $x, y \in S$ and $x \sqsubseteq y$, then $x \sqcup y = y$ and $x \sqcap y = x$. Also due to the reflexivity property of $\sqsubseteq$ we have that $x \sqcup x = x$ and $x \sqcap x = x$ (*idempotency laws*).

Definition 2.6 (Directed Sets, Lattices and Complete Lattices): A poset $\mathfrak{L} = (L, \sqsubseteq)$ is called:

- *directed set*, if for all $\{x, y\} \subseteq L$ it is $\{x, y\}^u \neq \emptyset$ and $\{x, y\}^l \neq \emptyset$
- *lattice*, if for all $x, y \in L$, there exist both $x \sqcup y$ and $x \sqcap y$
- *complete lattice*, if for all $X \subseteq L$, there exist both $\bigsqcup X$ and $\bigsqcap X$
- *c-complete lattice*, if $\mathfrak{L}$ is a complete lattice with an unary operator $\sim$, called complement, satisfying $\sim \bot = \top$ and $\sim \top = \bot$.

The above definition of lattices $(L, \sqsubseteq)$ is based on partial orders, but equivalently we can define lattices as algebraic structures $(L, \sqcup, \sqcap)$ that satisfy the *commutative*, *associative*, *absorption* and *idempotency* laws (see Theorem A.5). Thus, we will be using the above two definitions of lattices interchangeably. Note here that the *distributive* and *neutral element laws* need not hold on a lattice. Also from now on whenever we say that the *dual* holds for a statement, we mean the statement that is derived by interchanging the $\sqcap$ and $\sqcup$ functions in the original statement. This is also known as the *duality principle of lattices*.

Definition 2.7: Let us define the following set of axioms for x, y, z in some set S:

$$\begin{aligned} x \sqcup y &= y \sqcup x && \textit{(commutative laws)} \\ x \sqcap y &= y \sqcap x \\ (x \sqcup y) \sqcup z &= x \sqcup (y \sqcup z) && \textit{(associative laws)} \\ (x \sqcap y) \sqcap z &= x \sqcap (y \sqcap z) \\ x \sqcup (x \sqcap y) &= x && \textit{(absorption laws)} \\ x \sqcap (x \sqcup y) &= x \\ x \sqcup x &= x && \textit{(idempotency laws)} \\ x \sqcap x &= x \end{aligned}$$

Definition 2.8 (Distributive Lattice): A lattice $\mathfrak{L}$ is said to be *distributive* iff it satisfies the following *distributive law*

$$(\forall x, y, z \in L).(x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z))$$

or its dual (see Lemma A.8).

We conclude this section by giving the definitions of the continuous function and the join-irreducible elements. In computation theory, function monotonicity is a property that intuitively describes how good a function approximation is by imposing an ordering on its range. The property of function continuity is stronger than the property of monotonicity and thus it is preferred in many applications.

Definition 2.9 (Continuous Function): Let $(X, \sqsubseteq_X)$ and $(Y, \sqsubseteq_Y)$ be posets and $f : X \to Y$ be a map, then f is called *continuous* iff $\bigsqcup f(Z) = f(\bigsqcup Z)$ and $\bigsqcap f(Z) = f(\bigsqcap Z)$ for all directed non-empty subsets $Z \subseteq X$.

The Fundamental Theorem of Arithmetic, which says that every natural number is a product of prime numbers, generalises also for lattices. To that account every element of a finite lattice is a join of *join-irreducible* elements.

Definition 2.10 (Join-irreducible elements): Let $\mathfrak{L}$ be a lattice, then $x \in L$ is called *join-irreducible* if:

- $x \neq \bot$ (in case L has a bottom)
- $x = y \sqcup z$ implies $x = y$ or $x = z$ for all $y, z \in L$.

The second condition can be replaced by the more intuitive condition $y < x$ and $z < x$ imply $y \sqcup z < x$ for all $y, z \in L$. The set of all the join-irreducible elements of the lattice $\mathfrak{L}$ is denoted by $\mathcal{J}(\mathfrak{L})$. In a graph representation of a finite lattice the join-irreducible elements can be found by visual inspection: they are the elements which have exactly one lower cover, i.e. they are connected with only one element from below (see Fig. 3). In a finite lattice, every element can be written as a join of join-irreducible elements (see Proposition A.12).

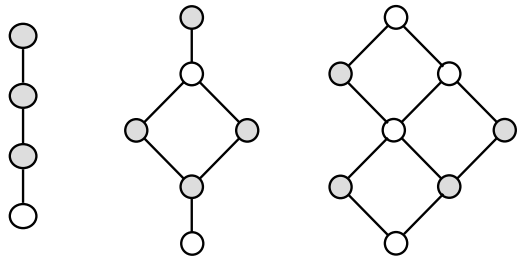

Fig. 3. Some lattices. The gray nodes denote the join irreducible elements of each lattice. Notice that every white node (besides the bottom element) is the join of a set of join irreducible elements.

Example 2.1: Some examples of join-irreducible elements are: (i) Every element in a chain besides the bottom are join-irreducible (see lattice L_n Fig. 1). (ii) In the lattice $(2^S, \subseteq)$ where S is a finite set of objects the only join-irreducible elements are the singleton sets $\{s\}$ for $s \in S$.

B. Quasi-Boolean Algebras and Boolean Algebras

The quasi-Boolean algebras are actually very similar to the "de Morgan lattices" with the only difference that the later do not require the existence of a least and a greatest element. The "de Morgan lattices" have been extensively discussed in the literature of constructive logic and relevance logic. First, we will introduce quasi-Boolean algebras [55] and then derive Boolean algebras as a special case.

Definition 2.11 (Quasi-Boolean Algebras): A *quasi-Boolean algebra* is a structure $\mathfrak{B} = (B, \sqcap, \sqcup, \sim, \bot, \top)$ where $(B, \sqcap, \sqcup)$ is a distributive lattice, $\bot$ and $\top$ are the least and the greatest elements and $\sim$ is an unary operation on B of period two such that for every $x \in B$ there exists a unique element $\sim x \in B$ satisfying the following axioms for all $x, y \in B$:

$$\begin{aligned} \sim (x \sqcap y) &= \sim x \sqcup \sim y \quad && (\textit{De Morgan}) \\ \sim (x \sqcup y) &= \sim x \sqcap \sim y \\ \sim\sim x &= x && (\textit{involution}) \\ x \sqsubseteq y \quad \text{iff} \quad \sim y &\sqsubseteq \sim x && (\textit{antimonotonic}) \end{aligned}$$

Hence on a quasi-Boolean algebra $\mathfrak{B}$, we can define the multi-valued *implication* $x \Rightarrow y := \sim x \sqcup y$ and *equivalence* $x \Leftrightarrow y := (x \Rightarrow y) \sqcap (y \Rightarrow x)$ for all $x, y \in B$. Note that on quasi-Boolean algebras equivalence is not the same as *equality* $x = y := (x \sqsubseteq y) \wedge (y \sqsubseteq x)$. For notational simplification, we will denote the quasi-Boolean algebra on a distributive lattice $\mathfrak{L}_x$ by $\mathfrak{B}_x$.

Remark 2.1: The product of two quasi-Boolean algebras is also a quasi-Boolean algebra. For the definition and proofs see [12]. For the definition of product on lattices see Definition 2.15 in [21].

Remark 2.2: The definition of the negation operator for quasi-Boolean algebras is simplified by the fact that the distributive lattices are symmetrical about their horizontal axis (see [12]).

Definition 2.12 (Boolean Algebra): A *Boolean algebra* is a quasi-Boolean algebra $\mathfrak{B}$ with the additional condition that for every element $x \in B$, it is:

$$\begin{aligned} x \sqcap \sim x &= \bot \quad \text{Law of Non-Contradiction} \\ x \sqcup \sim x &= \top \quad \text{Law of Excluded Middle} \end{aligned}$$

Example 2.2: Two Boolean algebras that are mentioned in this paper:

- Let the underlying lattice be the $\mathfrak{L}_2$, then we get the usual Boolean algebra denoted as $\mathfrak{B}_2 = (\{0, 1\}, \wedge, \vee, \neg, 0, 1)$ with the usual operators of implication $x \rightarrow y := \neg x \vee y$ and equivalence $x \leftrightarrow y := (x \rightarrow y) \wedge (y \rightarrow x)$ for all $x, y \in \{0, 1\}$.
- Let S be a finite set, then the powerset 2^S with the ordering induced by the set inclusion operation $\subseteq$ defines a lattice $(2^S, \subseteq)$. The structure $\mathfrak{B}_S = (2^S, \cap, \cup, \sim, \emptyset, S)$ with the complementation operation defined as $\sim T := S \backslash T$ for all $T \subseteq S$ is a Boolean algebra called the *powerset algebra* on S.

C. Fixpoint Functions

Fixpoint functions are of high importance for computer science. In our case, fixpoint functions are the basis of μ-calculus, multi-valued and CTL symbolic model checking. Let $(L, \sqsubseteq)$ be a lattice and $x \in L$, then x is called a *fixpoint* of a function f if $f(x) = x$ holds. Fixpoints do not exist for all functions. The following theorem by Tarski and Knaster gives the conditions for a function to have fixpoints as well as the means to compute them.

Theorem 2.1 (Tarksi/Knaster Fixpoint Theorem [62]): Let $(L, \sqsubseteq)$ be a complete lattice and $f : L \rightarrow L$ be an order-preserving function, then f has fixpoints. The least and greatest fixpoints are characterised as follows:

$$\mu x.f(x) = \bigsqcap \{x \in L \mid f(x) = x\} = \bigsqcap \{x \in L \mid f(x) \sqsubseteq x\}$$

$$\nu x.f(x) = \bigsqcup \{x \in L \mid f(x) = x\} = \bigsqcup \{x \in L \mid x \sqsubseteq f(x)\}$$

Let $y, z \in L$ such that $y \sqsubseteq f(y)$, $y \sqsubseteq \mu x.f(x)$, $f(z) \sqsubseteq z$, $\nu x.f(x) \sqsubseteq z$ and, let f be in addition continuous, then the iteration:

- y_i defined as $y_0 := y$ and $y_{i+1} := f(y_i)$ converges to $\mu x.f(x)$
- z_i defined as $z_0 := z$ and $z_{i+1} := f(z_i)$ converges to $\nu x.f(x)$

Algorithm 1 Computing the least or the greatest fixpoint

```
1: procedure FxPoint(x, f)
2:     x' ← f(x)
3:     while x ≠ x' do
4:         x ← x'
5:         x' ← f(x)
6:     end while
7:     return x
8: end procedure
```

Proof: See 3.6 in [57] ∎

For $x \in L$, we define $f^i(x)$ to be the application of f i times to x, or more formally, $f^1(x) = f(x)$ and $f^{i+1}(x) = f(f^i(x))$ for all $i \geq 2$. Sometimes the initialization of the recursion is done by setting $y = \bot$ and $z = \top$. Note that the recursion might not always converge to the fixpoint. This is the case for infinite lattices. On the other hand, on finite lattices the recursive procedure always converges and, furthermore, it takes at maximum $|S|$ iterations to converge. Algorithm 1 presents the recursive procedure for determining $\mu x.f(x)$ and $\nu x.f(x)$ by setting $x = y$ and $x = z$ as defined in Theorem 2.1. Finally, let us mention that if f is defined over a quasi-Boolean algebra $\mathfrak{B}$, that is $f : B \to B$, then the greatest and least fixpoint operators have the following properties (for a proof see Lemma 3.13 in [57]):

$$\nu x.f(x) =\sim \mu x. \sim f(\sim x) \text{ and } \mu x.f(x) =\sim \nu x. \sim f(\sim x)$$

D. Kripke structures

In this section, we will give a brief introduction to Kripke structures which one could claim that constitute the foundations of computer science. For a very detailed discussion and analysis of these models the reader is referred to [57].

The first step of any kind of analysis of real world phenomena is to capture them with an appropriate mathematical model. This comprises the basis of any science and hence finding the right modeling framework is of tremendous importance. One way to model the computations of a reactive finite system (and maybe the most dominant one) is Kripke structures. Kripke structures abstract away the inputs and outputs of a program and capture only the states and computations of the system. In this paper, we do not address how we create such a model, but we defer the reader to [4], [36], [19] for a short introduction.

Definition 2.13 (Kripke Structure): A *Kripke structure* is a tuple $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ where:

- S is a finite set of states
- S_0 is the set of the possible initial states
- $R \subseteq S \times S$ is the transition relation
- AP is a finite set of atomic propositions
- $\mathcal{O} : S \to 2^{AP}$ is an observation map that maps a state s to the set of atomic propositions that are true in the state s

Kripke structures model infinite computations (there are no accepting states as in automata). This implies that we need to focus our interest only on infinite sequences of states (i.e. $s_0, s_1, s_2, \ldots$ such that $s_0 \in S_0$ and $(s_i, s_{i+1}) \in R$) which we will call *paths* or *executions* of the Kripke structure. One way to model this is to require the transition relation to be total, i.e. $(\forall s \in S).(\exists t \in S).((s,t) \in R)$. But this approach requires some additional bookkeeping as sometimes we might get Kripke structures that have deadlocks (for example after taking the product of two Kripke structures with no deadlock states). Another approach to this problem is to make the semantics of our logics such that we only consider infinite paths. We follow the former approach as this seems to be the most common in the literature, but we derive it as a special case of the later.

In order to describe an infinite sequence of states, we will use the function $\pi : nat \to S$ defined as: $\pi(i)$ is the i-th state in the sequence $s_0, s_1, s_2, \ldots$. In the following (slightly abusing the notation), π will denote a path of the Kripke structure and $\pi[i]$ will denote the actual sequence of states, that is $\pi[i] = \pi(i), \pi(i+1), \pi(i+2), \ldots$.

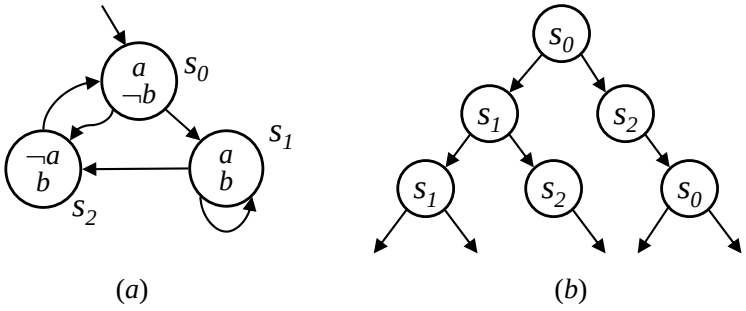

Fig. 4. (a) An example of a Kripke structure. (b) The evolution of the computation paths of the Kripke structure.

Definition 2.14 (Paths on a Kripke Structure): Let $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ be a Kripke structure, then for each $s \in S$:

$$Paths_{\mathcal{K}}(s) := \{\pi : nat \to S \mid (\pi(0) = s) \wedge ((\forall i \in nat).((\pi(i), \pi(i+1)) \in R))\}$$

which is the set of all infinite paths (functions) starting at state s
Also, we define $S_{inf} := \{s \in S \mid Paths_{\mathcal{K}}(s) \neq \emptyset\}$ and for all $T \subseteq S$, $Paths_{\mathcal{K}}(T) := \bigcup_{s \in T} Paths_{\mathcal{K}}(s)$. Note that we have assumed that the transition relation R is total, hence $S_{inf} = S$.

Definition 2.15 (Traces and Languages): A *trace* is the sequence of observations $\mathcal{O}(\pi(0)), \mathcal{O}(\pi(1)), \mathcal{O}(\pi(2)), \ldots$ which will be again denoted by $\mathcal{O}(\pi[0])$. The definition of the trace as a function will be the composition of the maps $\mathcal{O}$ and π, i.e. the map $\mathcal{O} \circ \pi : nat \to 2^{AP}$. The language of a state $s \in S$ is the (infinite) set of infinite sequences $Lang_{\mathcal{K}}(s) := \{\mathcal{O} \circ \pi[0] \mid \pi \in Paths_{\mathcal{K}}(s)\}$. The language of the structure $\mathcal{K}$ is defined as $Lang(\mathcal{K}) := \bigcup_{s \in S_0} Lang_{\mathcal{K}}(s)$.

In temporal logics we need to reason about the predecessor states of a state or a set of states. For this reason we define the following sets.

Definition 2.16 (Predecessor Sets): Let $R \subseteq S_1 \times S_2$ be a binary relation and let $Q \subseteq S_2$, then:

$$\begin{aligned} pre^R_{\exists}(Q) &:= \{s_1 \in S_1 \mid (\exists s_2).((s_1, s_2) \in R \wedge s_2 \in Q)\} \\ pre^R_{\forall}(Q) &:= \{s_1 \in S_1 \mid (\forall s_2).((s_1, s_2) \in R \to s_2 \in Q)\} \end{aligned}$$

Intuitively, $pre^R_{\exists}(Q)$ is the set of states in S_1 that have at least one successor in Q and $pre^R_{\forall}(Q)$ is the set of states in S_1 that have all their successors in set Q. Note though that if $Q = \emptyset$ and there is no transition from a state in S_1 to a state in S_2, i.e. $(\forall s_1).(\forall s_2).((s_1, s_2) \notin R)$, then the condition $(\forall s_2).((s_1, s_2) \in R \to s_2 \in Q)$ is vacuously true and $pre^R_{\forall}(Q) = S_1$. Finally, let us mention that the predecessor sets have several interesting properties (the proofs are straightforward from the definitions).

Lemma 2.2: Let $R \subseteq S_1 \times S_2$ be a binary relation and let $Q \subseteq S_2$, then:

1) *Duality*:

$$pre^R_{\exists}(Q) = S_1 \backslash pre^R_{\forall}(S_2 \backslash Q) \text{ and } pre^R_{\forall}(Q) = S_1 \backslash pre^R_{\exists}(S_2 \backslash Q)$$

2) *Monotonicity*:

$$Q \subseteq Q' \text{ implies } pre^R_{\exists}(Q) \subseteq pre^R_{\exists}(Q') \text{ and } pre^R_{\forall}(Q) \subseteq pre^R_{\forall}(Q')$$

3) *Distributivity*:

$$\begin{aligned} pre^R_{\exists}(\textstyle\bigcup_i Q_i) &= \textstyle\bigcup_i pre^R_{\exists}(Q_i) \quad \textstyle\bigcup_i pre^R_{\forall}(Q_i) \subseteq pre^R_{\forall}(\bigcup_i Q_i) \\ pre^R_{\exists}(\textstyle\bigcap_i Q_i) &\subseteq \textstyle\bigcap_i pre^R_{\exists}(Q_i) \quad pre^R_{\forall}(\bigcap_i Q_i) = \textstyle\bigcap_i pre^R_{\forall}(Q_i) \end{aligned}$$

III. Temporal logics and the μ-calculus

In the following sections, we formally describe the temporal logic named CTL* (CTL stands for Computational Tree Logic) [23] and the μ-calculus [43]. Even though μ-calculus is more expressive then CTL* and every CTL* formula can be converted to en equivalent μ-calculus formula, their different model checking algorithms and underlying theoretical basis merit the separate and explicit presentation of these two logics. Also in this section we are going to describe two common fragments of CTL* namely the linear temporal logic (LTL) and the computational tree logic (CTL).

A. Temporal Logics

The need to utter statements whose truth value depends on the current, future and past time has led to the development of the temporal logics which are a branch of modal logics. They were first introduced in the middle of the previous century by philosophers who wanted to argue about the passage of time. With this formalism, we can express properties like *always*, *until*, *before* and *sometimes*. For example, temporal logics can formally capture statements like "Whenever there are clouds in the sky, it rains the next day" (if we consider the time to be discrete and representing days). The truth of this sentence depends of course on the model of our world. It could be the case that in some other world this statement is always true, but in our world this is not the case. On the other hand, the statement "Whenever there are clouds in the sky, it sometimes rains the next day" is true as this statement captures every possible future.

In the context of computer science, temporal logics were first (very insightfully) employed by Pnueli [54] in the seventies for the analysis of distributed systems. Since then temporal logics in combination with model checking techniques have provided a valuable tool for the analysis and design of software [36], [19], control [61], real time monitoring [49], planning [29] and mobile robot motion planning [24]. In addition, they can provide a formal framework for wrapping the above scientific results with a higher layer of human-machine interaction through natural languages [45].

B. The Propositional Branching Temporal Logic CTL*

We first give some informal description of CTL* formulas. The propositional logic is the traditional logic whose semantics are interpreted using the Boolean algebra over the lattice $\mathfrak{L}_2$ (see Section II-B). The set of well formed formulas $form$ of propositional logic are build upon a set AP of atomic propositions and some binary and unary operators like *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), *implication* ($\rightarrow$), and *equivalence* ($\leftrightarrow$). It was proved that the propositional logic can have as a basis for its axiomatisation just the two operators ($\neg$,$\vee$) while the rest of them can be derived. CTL* is obtained from the standard propositional logic by adding temporal operators such as *eventually* (F or in the *future*), *always* (G or *globally*), *next* (X), *until* ($\mathcal{U}$) and *before* ($\mathcal{B}$) in combination with the path quantifiers *for all* paths (A) and *for some* path (E). The usual basis for the temporal operators is the pair (X,$\mathcal{U}$).

The intuition behind these operators is as follows. $F\phi$ means that at some point in the future the formula ϕ will hold, whereas $G\phi$ means that ϕ should hold at every moment in the future. $X\phi$ states that ϕ will hold at the next time moment (note that we regard time as a discrete sequence and not as continuous quantity). $\phi_1 \mathcal{U} \phi_2$ means that ϕ_1 should be true at every time in the future until ϕ_2 becomes true. Finally, $\phi_1 \mathcal{B} \phi_2$ is true when ϕ_1 holds at some time before ϕ_2 becomes true or when neither ϕ_1 or ϕ_2 ever become true. Let's consider again the two examples mentioned above (Section III-A). Let the set of atomic propositions be $AP = \{clouds, rain\}$, then formally we can write "$A[G(clouds \rightarrow Xrain)]$" and "$A[G(clouds \rightarrow E[Xrain])]$" respectively.

1) CTL Syntax:* CTL* is a temporal logic whose syntax contains both *state* formulas ϕ_s and *path* formulas ϕ_p. Path formulas are interpreted over paths of the Kripke structure, whereas state formulas are interpreted on states of the Kripke structure. Every state formula is also a path formula in the sense that it should hold on the first state $\pi(0)$ of a path π. Let AP be the set of atomic propositions that take values on $\{0, 1\}$, then the set $form$ of well formed CTL* formulas are generated according to the following grammar:

$$\begin{aligned} \phi_s &::= a \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid E[\phi_p] \\ \phi_p &::= \phi_s \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid X\phi_p \mid [\phi_p \mathcal{U} \phi_p] \end{aligned}$$

where $a \in AP$. As usual, we denote the boolean constants by 1 (*true*) and 0 (*false*). Given the basis ($\neg$,$\vee$) and for all $\phi_1, \phi_2 \in form$, we can define conjunction ($\wedge$) by $\phi_1 \wedge \phi_2 := \neg(\neg\phi_1 \vee \neg\phi_2)$, implication ($\rightarrow$) by $\phi_1 \rightarrow \phi_2 := \neg\phi_1 \vee \phi_2$ and equivalence ($\leftrightarrow$) as $\phi_1 \leftrightarrow \phi_2 := \phi_1 \rightarrow \phi_2 \wedge \phi_2 \rightarrow \phi_1$. Also, the universal path quantifier can be derived from the existential one: $A[\phi] = \neg E[\neg\phi]$ for $\phi \in form$. Furthermore, we can also derive additional temporal operators such as:

- *Eventually*: $F\phi = [1 \mathcal{U} \phi]$
- *Before (weak)*: $[\phi_1 \mathcal{B} \phi_2] = \neg[\neg\phi_1 \mathcal{U} \phi_2]$
- *Release*: $[\phi_1 \mathcal{R} \phi_2] = \neg[\neg\phi_1 \mathcal{U} \neg\phi_2]$

- *Always*: $G\phi = [0\,\mathcal{B}\neg\phi]$ or $G\phi = [0\mathcal{R}\phi]$

The temporal operator (weak) before ($\mathcal{B}$) is the dual of the (strong) until ($\mathcal{U}$). The operator release ($\mathcal{R}$) is also dual of the until, but it has less intuitive semantics. In cases where we cannot use the negation operator (as it is sometimes the case in multi-valued logics), we will replace before with release in the syntax of CTL*. There exist various versions (weak and strong) of the above operators as well as some additional temporal operators that we will not mention here. The interested reader is pointed to [57] for a detailed exposition.

2) CTL* Semantics: Let $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ be a Kripke structure and π be a path of $\mathcal{K}$. We define the semantics of CTL* formulas using the Kripke structure $\mathcal{K}$. Let $a \in AP$. When a state formula ϕ_s is true on a state $s \in S$, we write $(\mathcal{K}, s) \models_s \phi_s$ which is read as "s satisfies ϕ_s". The semantics of any state formula can be defined as:

$$\begin{array}{ll} (\mathcal{K}, s) \models_s a & \text{iff} \quad a \in \mathcal{O}(s) \\ (\mathcal{K}, s) \models_s \neg\phi_s & \text{iff} \quad (\mathcal{K}, s) \not\models_s \phi_s \\ (\mathcal{K}, s) \models_s \phi_{s1} \vee \phi_{s2} & \text{iff} \quad (\mathcal{K}, s) \models_s \phi_{s1} \vee (\mathcal{K}, s) \models_s \phi_{s2} \\ (\mathcal{K}, s) \models_s E\phi_p & \text{iff} \quad (\exists \pi \in Paths_{\mathcal{K}}(s)).((\mathcal{K}, \pi[0]) \models_p \phi_p) \end{array}$$

And additionally:

$$(\mathcal{K}, s) \models_s A\phi_p \quad \text{iff} \quad (\forall \pi \in Paths_{\mathcal{K}}(s)).((\mathcal{K}, \pi[0]) \models_p \phi_p)$$

Path formulas ϕ_p are interpreted over a path π of the Kripke structure. When the path satisfies the formula we write $(\mathcal{K}, \pi[0]) \models_p \phi_p$. Let $i, j, k \in nat$. The semantics of any path formula can be recursively defined as:

$$\begin{array}{rll} (\mathcal{K}, \pi[i]) \models_p \phi_s & \text{iff} & (\mathcal{K}, \pi(i)) \models_s \phi_s \\ (\mathcal{K}, \pi[i]) \models_p \neg\phi_p & \text{iff} & (\mathcal{K}, \pi[i]) \not\models_p \phi_p \\ (\mathcal{K}, \pi[i]) \models_p \phi_{p1} \vee \phi_{p2} & \text{iff} & (\mathcal{K}, \pi[i]) \models_p \phi_{p1} \vee (\mathcal{K}, \pi[i]) \models_p \phi_{p2} \\ (\mathcal{K}, \pi[i]) \models_p X\phi_p & \text{iff} & (\mathcal{K}, \pi[i+1]) \models_p \phi_p \\ (\mathcal{K}, \pi[i]) \models_p [\phi_{p1}\,\mathcal{U}\phi_{p2}] & \text{iff} & (\exists j \geq i).[(\forall k \in [i, j)).((\mathcal{K}, \pi[k]) \models_p \phi_{p1}) \wedge ((\mathcal{K}, \pi[j]) \models_p \phi_{p2})] \end{array}$$

And additionally:

$$\begin{array}{rll} (\mathcal{K}, \pi[i]) \models_p F\phi_p & \text{iff} & (\exists j \geq i).((\mathcal{K}, \pi[j]) \models_p \phi_p) \\ (\mathcal{K}, \pi[i]) \models_p G\phi_p & \text{iff} & (\forall j \geq i).((\mathcal{K}, \pi[j]) \models_p \phi_p) \\ (\mathcal{K}, \pi[i]) \models_p [\phi_{p1}\,\mathcal{B}\phi_{p2}] & \text{iff} & (\forall j \geq i).[(\forall k \in [i, j)).((\mathcal{K}, \pi[k]) \not\models_p \phi_{p1}) \rightarrow ((\mathcal{K}, \pi[j]) \not\models_p \phi_{p2})] \\ (\mathcal{K}, \pi[i]) \models_p [\phi_{p1}\mathcal{R}\phi_{p2}] & \text{iff} & (\forall j \geq i).[(\exists k \in [i, j)).((\mathcal{K}, \pi[k]) \models_p \phi_{p1}) \vee ((\mathcal{K}, \pi[j]) \models_p \phi_{p2})] \end{array}$$

The reason that we also give the formal semantics of the universal path quantifier and the derived temporal operators is that they are going to be necessary in the various fragments of CTL* and in the Negation Normal Form[3] (NNF) of the various temporal logics. In the following, we will drop the subscripts s and p (unless we introduce some new grammar) as it will always be clear from the context whether a subformula is interpreted over states or paths. In CTL* the user has to be cautious with the usage of the existential quantifiers. The following example indicates why.

Example 3.1: Consider the CTL* formulas $E[Fb] \wedge E[Fc]$ and $E[Fb \wedge Fc]$ (Figures 5 and 6). The first formula expresses the property that at the current state, where we observe a, there exist two paths, one that leads to b and one (not necessarily the same) that leads to c (Figure 5), whereas the second formula describes the existence of a path that leads eventually to b and then to c or vice versa (Figure 6).

[3]The Negation Normal Form (NNF) is a form for logic formulas where the negation operator appears only in front of atomic propositions. Any temporal logic formula interpreted over Boolean truth values can be converted to NNF. For further details see [57].

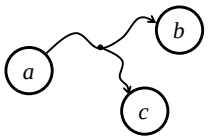

Fig. 5. The possible bifurcation of a path that leads to two separate paths.

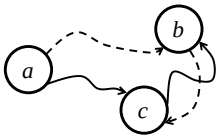

Fig. 6. Two possible cases: *Solid line* a path that leads to c and then to b. *Dashed line* a path that leads to b and then to c.

3) Set of states that satisfy formula ϕ: In order to implement several model checking algorithms, we need to know the set of states of a structure $\mathcal{K}$ that satisfy a state formula ϕ. Let us define $[\![\phi]\!]_{\mathcal{K}} := \{s \in S \mid (\mathcal{K}, s) \models_s \phi\}$, then the following lemma gives us the set of rules to calculate these sets (for the proof see Appendix B).

Lemma 3.1: Let $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ be a Kripke structure and ϕ be a state formula. The set of the following rules can determine the set of states $[\![\phi]\!]_{\mathcal{K}}$ that satisfy ϕ on $\mathcal{K}$. Let ϕ and ψ be CTL* state formulas and a be an atomic proposition i.e. $a \in AP$, then:

$$\begin{aligned}
[\![a]\!]_{\mathcal{K}} &= \{s \in S \mid a \in \mathcal{O}(s)\} \\
[\![\neg\phi]\!]_{\mathcal{K}} &= S \backslash [\![\phi]\!]_{\mathcal{K}} = \overline{[\![\phi]\!]}_{\mathcal{K}} \\
[\![\phi \vee \psi]\!]_{\mathcal{K}} &= [\![\phi]\!]_{\mathcal{K}} \cup [\![\psi]\!]_{\mathcal{K}} \\
[\![\phi \wedge \psi]\!]_{\mathcal{K}} &= [\![\phi]\!]_{\mathcal{K}} \cap [\![\psi]\!]_{\mathcal{K}} \\
[\![EX\phi]\!]_{\mathcal{K}} &= pre^R_{\exists}(S_{inf} \cap [\![\phi]\!]_{\mathcal{K}}) \\
[\![AX\phi]\!]_{\mathcal{K}} &= pre^R_{\forall}((S \backslash S_{inf}) \cup [\![\phi]\!]_{\mathcal{K}})
\end{aligned}$$

As R is total and, hence, $S_{inf} = S$, the two last equations of Lemma 3.1 reduce to $[\![EX\phi]\!]_{\mathcal{K}} = pre^R_{\exists}([\![\phi]\!]_{\mathcal{K}})$ and $[\![AX\phi]\!]_{\mathcal{K}} = pre^R_{\forall}([\![\phi]\!]_{\mathcal{K}})$. Note that here when $[\![\phi]\!]_{\mathcal{K}} = \emptyset$, then there is no problem with the universal predecessor set $pre^R_{\forall}([\![\phi]\!]_{\mathcal{K}})$ as we quantify over the whole set S and the transition relation R is total. Furthermore, due to Lemma 2.2 both operators EX and AX are order preserving. Before closing this section, let us define the characteristic function of the set $[\![\phi]\!]_{\mathcal{K}}$ to be a boolean function $\|\phi\|_{\mathcal{K}} : S \to \{0, 1\}$ with definition:

$$\|\phi\|_{\mathcal{K}}(s) = 1 \quad \text{iff} \quad s \in [\![\phi]\!]_{\mathcal{K}}$$

Sometimes we drop the subscript $\mathcal{K}$ to make the text more readable. Even though the norm notation $\|.\|$ is not preferred in this context, we believe that it makes the text more clear as it differentiates between the actual set and its membership function.

C. The Linear Temporal Logic and the Computational Tree Logic

Unfortunately, the power and expressiveness of CTL* makes the model checking problem hard. This is not entirely true, as the CTL* model checking problem has the same complexity as the LTL model checking problem, but it does require additional bookkeeping as it is a combination of CTL and LTL model checking. Hence, we have to consider fragments of CTL* for which efficient model checking tools exist.

One such fragment is the propositional linear temporal logic (LTL) [23]. LTL avoids quantification over paths, and is generated by the following grammar:

$$\begin{aligned}
\phi_s &::= A[\phi_p] \\
\phi_p &::= a \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid X\phi_p \mid [\phi_p \mathcal{U} \phi_p]
\end{aligned}$$

The semantics of LTL are naturally inherited from the semantics of CTL*. Therefore, $(\mathcal{K}, s) \models_s A\phi_p$ if all paths π originating at s satisfy the path formula ϕ_p. Note that it is preferred in LTL to drop the universal quantifier A and to reason using only the path formulas ϕ_p.

Another fragment of CTL* is the computational tree logic (CTL) [23] which is generated by the following restricted syntax:

$$\begin{aligned} \phi_s &::= a \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid E[\phi_p] \mid A[\phi_p] \\ \phi_p &::= X\phi_s \mid [\phi_s \mathcal{U} \phi_s] \end{aligned}$$

Note that the difference between CTL* and CTL is that in CTL, path formulas may no longer be nested. They require the use of a path quantifier (either E or A) to convert a path formula into a state formula. Therefore the grammar described above for generating CTL formulas can be equivalently expressed as [19]:

$$\phi_s ::= a \mid \neg\phi_s \mid \phi_s \vee \phi_s \mid EX\phi_s \mid EG\phi_s \mid E[\phi_s \mathcal{U} \phi_s]$$

The semantics of CTL are naturally inherited from the semantics of CTL*. Below, we will also present the derived operators of CTL as they are mandatory for the CTL model checking algorithms [19]:

$$\begin{aligned} AX\phi &= \neg EX\neg\phi \\ EF\phi &= E[1\,\mathcal{U}\phi] \\ AG\phi &= \neg EF\neg\phi \\ AF\phi &= \neg EG\neg\phi \\ A[\phi_1 \mathcal{U} \phi_2] &= \neg E[\neg\phi_2 \mathcal{U} (\neg\phi_1 \wedge \neg\phi_2)] \wedge \neg EG\neg\phi_2 \\ A[\phi_1 \mathcal{B} \phi_2] &= \neg E[\neg\phi_1 \mathcal{U} \phi_2] \\ E[\phi_1 \mathcal{B} \phi_2] &= \neg A[\neg\phi_1 \mathcal{U} \phi_2] \end{aligned}$$

LTL and CTL are incomparable, but they do have a common fragment [48]. The main difference between the semantics of LTL and CTL is that LTL formulas are interpreted over all paths starting from $\pi(0)$, whereas CTL formulas are interpreted over possible paths generated from a given state. In CTL one can express potential reachability $AG\,EF\phi$, but one cannot define properties along a path. On the other hand, LTL cannot distinguish between different paths that can be generated from the same state, but it can state properties that take place frequently often in the future, for example $A[GF\phi]$. The choice of which logic to use depends on two factors. Theoretically, one must choose the logic that can expresses the property of interest. Practically, one must also consider the model checking tools and the temporal logics that they focus on. For example, SPIN [36] focuses on LTL model checking, whereas NUSMV [17] can be applied to both CTL and LTL model checking.

D. Nondeterministic Büchi Automata

Büchi automata [63] extend the classical notion of automata [37] by considering as inputs infinite words instead of finite ones. We prefer to treat them under the logics section as Büchi automata can be used to capture LTL specifications [64]. The conversion from LTL formulas to Büchi automata is a well studied problem that has given rise to many efficient algorithms [27], [28].

Definition 3.1 (Büchi automaton): A Büchi automaton is a tuple $B = (S, S_0, \Sigma, \rightarrow_B, F)$ where:

- S is a finite set of states
- S_0 is the set of the possible initial states
- Σ is the input alphabet of the automaton
- $\rightarrow_B \subseteq S \times \Sigma \times 2^S$ is a nondeterministic transition relation
- $F \subseteq S$ is the set of accepting states

An *infinite word* w is a member of Σ^ω, which informally means that we concatenate an infinite number of symbols from Σ (i.e. $w = w_0, w_1, w_2, \ldots$ with $w_i \in \Sigma$). A *run* r of B is the sequence of states $r = r_0, r_1, r_2, \ldots$ with $r_i \in S$ that occurs under the input word w. Let $lim(\cdot)$ be the function that returns the set of states that are encountered infinitely often in the run r of B. The language of B, i.e. $Lang(B)$, consists of all the input words that have a run that is accepted by B.

Definition 3.2 (Büchi acceptance): A Büchi automaton B accepts an infinite input word w iff the run $r = r_0, r_1, r_2, \ldots$ such that $r_i \xrightarrow{w_i}_B r_{i+1}$ with $r_0 \in S_0$, $r_i \in S$ and $w_i \in \Sigma$, satisfies the relationship $lim(r) \cap F \neq \emptyset$.

Finding the existence of accepting runs is an easy problem. First, we convert the Büchi automaton to a directed graph and, then, we find the strongly connected components (SCC) in the graph [36]. If at least one SCC that contains a final state ($s \in F$) is reachable from some state in the set of initial states S_0, then the language $L(B)$ of B is non-empty. The reasoning behind the above procedure is that there exists a finite prefix of an infinite run that takes you to a final state and then the suffix of the run encounters the final state an infinite number of times. The aforementioned algorithm (for more details see [36]) can be performed in time linear in the size of the input graph, but the conversion from LTL to Büchi automaton generates an automaton which has in the worst case size exponential in the size of the LTL formula. Finally, Büchi automata are closed under complementation and intersection.

E. The μ-calculus

The μ-calculus [43] is not actually a calculus as the name implies (for that we need also a set of axioms and rules to derive theorems), but a formal language. As a language, though, is very expressive and it can capture many other temporal (like CTL*) and program logics. The fact that there also exist efficient model checking algorithms for μ-calculus has made this language a valuable tool and a target for theoretical research.

Every closed μ-calculus formula is interpreted over a Kripke structure $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ and evaluates to the set of states that satisfy it. The μ-calculus is based on the computation of fixpoints of functions $f : 2^S \to 2^S$ that map a set of states of the Kripke structure $\mathcal{K}$ to a set of states. If these functions f are order-preserving, we know that they are also continuous as long as the Kripke structure is finite. Note that the powerset of the finite set of states of $\mathcal{K}$ along with the ordering induced by the set inclusion relation form a Boolean algebra $(2^S, \cap, \cup, \sim, \emptyset, S)$[4]. Thus, we know that the fixpoints of the functions exist and that we can calculate them using the Tarski/Knaster Fixpoint Theorem (See Section 2.1). These fixpoint functions are usually called *state transformers*.

Let us again denote the set of states where a μ-calculus formula ϕ holds by $[\![\phi]\!]_{\mathcal{K}}$. We need to discover the relationship between the fixpoint function f and the $[\![\phi]\!]_{\mathcal{K}}$. Assume that the variable X appears in ϕ (note that X ranges over sets of states). Then $[\![\phi]\!]_{\mathcal{K}}$ depends on the set of states that X holds. Hence, we can make $[\![\phi]\!]_{\mathcal{K}}$ a function in X and write $[\![\phi(X)]\!]_{\mathcal{K}}$. Let VAR be the set of variables that range over subsets T of S and $e : \text{VAR} \to 2^S$ be the environment. The role of the environment is to replace the variable X by the set of states Q and to retain everything else the same (we write $e[X \leftarrow Q]$ and it is $e[X \leftarrow Q](X) = Q$). We define the state transformer $f_{\mathcal{K},\phi} : 2^S \to 2^S$ to be the function $f_{\mathcal{K},\phi}(Q) := [\![\phi(X)]\!]_{\mathcal{K}} e[X \leftarrow Q]$.

Now, we proceed to define the syntax of the μ-calculus. We need to impose certain structural restrictions on the formulas in order to guarantee the monotonicity of the state transformers.

Definition 3.3 (The Syntax of μ-calculus formulas): Let $X \in \text{VAR}$ and $a \in AP$, then the set of μ-calculus pre-formulas can be derived according to the following grammar:

$$\phi ::= a \mid X \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \Diamond\phi \mid \Box\phi \mid \mu X.\phi \mid \nu X.\phi$$

The set of μ-calculus formulas is formed as a subset of the pre-formulas by imposing the additional condition that in the subformulas $\mu X.\phi(X)$ and $\nu X.\phi(X)$ the occurrences of X in ϕ should be under an even number of negation symbols.

This last condition is imposed in order to guarantee the monotonicity of the state transformers. If a variable X appears under a least μX or greatest νX fixpoint operator, then it is called *bound* variable otherwise it is called *free*. A μ-calculus formula without free variables is called *closed*. If a μ-calculus formula ϕ has the free variables $X_1, X_2, \ldots, X_n$, then we denote it by $\phi(X_1, X_2, \ldots, X_n)$.

Definition 3.4 (The Semantics of μ-calculus formulas): Let $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ be a Kripke structure and ϕ be a μ-calculus formula. The set of the following rules can determine the set of states $[\![\phi]\!]_{\mathcal{K}} e$ that satisfy ϕ on $\mathcal{K}$. Let ϕ and ψ be μ-calculus formulas, a be an atomic proposition i.e. $a \in AP$, and X a variable, i.e.

[4]We will sometimes denote $\emptyset$ by $\bot$ and S by $\top$ employing the usual lattice notation

$X \in$ VAR, then:

$$\begin{aligned}
[\![a]\!]_{\mathcal{K}}e &= \{s \in S \mid a \in \mathcal{O}(s)\} \\
[\![X]\!]_{\mathcal{K}}e &= e(X) \\
[\![\neg\phi]\!]_{\mathcal{K}}e &= S \backslash [\![\phi]\!]_{\mathcal{K}}e = \overline{[\![\phi]\!]}_{\mathcal{K}}e \\
[\![\phi \vee \psi]\!]_{\mathcal{K}}e &= [\![\phi]\!]_{\mathcal{K}}e \cup [\![\psi]\!]_{\mathcal{K}}e \\
[\![\phi \wedge \psi]\!]_{\mathcal{K}}e &= [\![\phi]\!]_{\mathcal{K}}e \cap [\![\psi]\!]_{\mathcal{K}}e \\
[\![\Diamond\phi]\!]_{\mathcal{K}}e &= pre^R_{\exists}([\![\phi]\!]_{\mathcal{K}}e) \\
[\![\Box\phi]\!]_{\mathcal{K}}e &= pre^R_{\forall}([\![\phi]\!]_{\mathcal{K}}e) \\
[\![\mu X.\phi(X)]\!]_{\mathcal{K}}e &= \bigcap\{Q \in 2^S \mid [\![\phi(X)]\!]_{\mathcal{K}}e[X \leftarrow Q] \subseteq Q\} \\
[\![\nu X.\phi(X)]\!]_{\mathcal{K}}e &= \bigcup\{Q \in 2^S \mid Q \subseteq [\![\phi(X)]\!]_{\mathcal{K}}e[X \leftarrow Q]\}
\end{aligned}$$

An alternative way to define the semantics of the last two rules is: $[\![\mu X.\phi(X)]\!]_{\mathcal{K}}e$ and $[\![\nu X.\phi(X)]\!]_{\mathcal{K}}e$ are the least and greatest fixpoints respectively of the state transformer $f_{\mathcal{K},\phi} : 2^S \to 2^S$ defined by:

$$f_{\mathcal{K},\phi}(Q) := [\![\phi(X)]\!]_{\mathcal{K}}e[X \leftarrow Q]$$

Informally, the operator $\Diamond\phi$ means that there exists a state reachable in one step where ϕ holds, while the operator $\Box\phi$ means that ϕ holds on all states reachable in the next step. Notice the difference with the EX and AX operators. The operators $\Diamond$, $\Box$ do not require the existence of an infinite path as EX and AX do. When $S_{inf} = S$, then the semantics of these operators are the same (respectively).

We have already proven the the operations $\cup$ and $\cap$ (as join and meet operations over the powerset algebra on S) and the functions $pre^R_{\exists}$ and $pre^R_{\forall}$ are order-preserving (Lemma A.2 and Lemma 2.2 respectively). For a proof about the monotonicity of the fixpoint operators $[\![\mu X.\phi(X)]\!]_{\mathcal{K}}e$ and $[\![\nu X.\phi(X)]\!]_{\mathcal{K}}e$ see Lemma 3.16 in [57]. Using the above lemmas we can now prove the monotonicity of any state transformer of a μ-calculus formula (Theorem 3.17 in [57]). Hence the conclusion is that we can use the Tarski-Knaster theorem to compute the fixpoints of the state transformers. Note though that the formulas $\mu X.\phi(X)$ and $\nu X.\phi(X)$ where X does not occur in ϕ under an even number of negation symbols could also have fixpoints, but this is not guaranteed.

Example 3.2: Here, we present some examples of μ-calculus formulas (for proofs of the deductions see [57]).

- $[\![\nu X.X]\!]_{\mathcal{K}} = S$ and $[\![\mu X.X]\!]_{\mathcal{K}} = \emptyset$
- $[\![\nu X.X \vee \phi]\!]_{\mathcal{K}} = S$ and $[\![\mu X.X \vee \phi]\!]_{\mathcal{K}} = [\![\phi]\!]_{\mathcal{K}}$
- $[\![\nu X.\Diamond X]\!]_{\mathcal{K}} = S_{inf}$ and $[\![\mu X.\Diamond X]\!]_{\mathcal{K}} = \emptyset$
- $[\![\nu X.\phi \wedge \Diamond X]\!]_{\mathcal{K}} = [\![EG\phi]\!]_{\mathcal{K}}$ and $[\![\mu X.\phi \wedge \Diamond X]\!]_{\mathcal{K}} = \emptyset$
- $[\![\nu X.\phi \wedge (\nu Y.\Diamond Y) \vee \Diamond X]\!]_{\mathcal{K}} = S_{inf}$ and $[\![\mu X.\phi \wedge (\nu Y.\Diamond Y) \vee \Diamond X]\!]_{\mathcal{K}} = [\![EF\phi]\!]_{\mathcal{K}}$
- $[\![\nu X.\phi \wedge \Diamond\Diamond X]\!]_{\mathcal{K}} = \{s \in S \mid (\exists \pi \in Paths_{\mathcal{K}}(s)).(\forall t \in nat).((\mathcal{K}, \pi(2t)) \models_s \phi)\}$ (note that this is not expressible with temporal logics) and $[\![\mu X.\phi \wedge \Diamond\Diamond X]\!]_{\mathcal{K}} = \emptyset$

IV. Classical Model Checking

This section briefly introduces the most popular model checking techniques and algorithms. As this section serves as a bridge to the multi-valued model checking, we do not give a detailed exposition of the algorithms. The model checking algorithms are categorised according to the logics that they can handle. First, we present the CTL symbolic model checking and then the automata theoretic LTL model checking. We continue with a short description of the CTL* model checking. Finally, we describe the main points of μ-calculus model checking procedure and we conclude with an introduction to the two most popular model checking software packages.

A. Symbolic Model Checking

One of the breakthroughs in model checking that enabled the verification of large scale systems is the development of the symbolic model checking and the introduction of the Ordered Binary Decision Diagrams (OBDDs). In this paper, we do not discuss or present OBDDs; the reader is referred to [60] for an introduction.

Let us mention though that OBDDs are a normal form (like the Disjunctive Normal Form - DNF) that can efficiently (in terms of size) capture Boolean formulas and easily operate on them (negation, conjunction, etc). Symbolic model checking is based on the manipulation of boolean formulas (made possible by the OBDD representation) instead of explicit states and transitions. In other words, it offers a solution to the state explosion problem as it avoids the explicit representation in the memory of all the states of the system. Symbolic model checking was mainly developed for CTL, but later LTL was incorporated by adding fairness constraints. The basis of CTL symbolic model checking is the fixpoint characterisation of the temporal operators.

Lemma 4.1: Let $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ be a Kripke structure and ϕ, ϕ_1 and ϕ_2 be CTL (state) formulas. Each of the basic CTL temporal operators can be characterised as a least or greatest fixpoint of an appropriate transformer for $Z \subseteq S$:

$$\begin{aligned}
[\![AF\phi]\!]_{\mathcal{K}} &= \mu Z.[\![\phi]\!]_{\mathcal{K}} \cup [\![AX\ Z]\!]_{\mathcal{K}} \\
[\![EF\phi]\!]_{\mathcal{K}} &= \mu Z.[\![\phi]\!]_{\mathcal{K}} \cup [\![EX\ Z]\!]_{\mathcal{K}} \\
[\![AG\phi]\!]_{\mathcal{K}} &= \nu Z.[\![\phi]\!]_{\mathcal{K}} \cap [\![AX\ Z]\!]_{\mathcal{K}} \\
[\![EG\phi]\!]_{\mathcal{K}} &= \nu Z.[\![\phi]\!]_{\mathcal{K}} \cap [\![EX\ Z]\!]_{\mathcal{K}} \\
[\![A[\phi_1\,\mathcal{U}\phi_2]]\!]_{\mathcal{K}} &= \mu Z.[\![\phi_2]\!]_{\mathcal{K}} \cup ([\![\phi_1]\!]_{\mathcal{K}} \cap [\![AX\ Z]\!]_{\mathcal{K}} \\
[\![E[\phi_1\,\mathcal{U}\phi_2]]\!]_{\mathcal{K}} &= \mu Z.[\![\phi_2]\!]_{\mathcal{K}} \cup ([\![\phi_1]\!]_{\mathcal{K}} \cap [\![EX\ Z]\!]_{\mathcal{K}}) \\
[\![A[\phi_1\,\mathcal{B}\phi_2]]\!]_{\mathcal{K}} &= \nu Z.\overline{[\![\phi_2]\!]}_{\mathcal{K}} \cap ([\![\phi_1]\!]_{\mathcal{K}} \cup [\![AX\ Z]\!]_{\mathcal{K}}) \\
[\![E[\phi_1\,\mathcal{B}\phi_2]]\!]_{\mathcal{K}} &= \nu Z.\overline{[\![\phi_2]\!]}_{\mathcal{K}} \cap ([\![\phi_1]\!]_{\mathcal{K}} \cup [\![EX\ Z]\!]_{\mathcal{K}})
\end{aligned}$$

where by slight abuse of notation we define $[\![EX\ Z]\!]_{\mathcal{K}} = pre^R_{\exists}(Z)$ and $[\![AX\ Z]\!]_{\mathcal{K}} = pre^R_{\forall}(Z)$.

Proof: See Lemmas 9 to 12 in Chapter 6 in [19] for the proof of the EG and EU temporal operators. Here, we will just show how to derive the operator AB from EU.

$$\begin{aligned}
[\![A[\phi_1\,\mathcal{B}\phi_2]]\!]_{\mathcal{K}} &= \overline{[\![E[\neg\phi_1\,\mathcal{U}\phi_2]]\!]}_{\mathcal{K}} \\
&= \overline{\mu Z.[\![\phi_2]\!]_{\mathcal{K}} \cup (\overline{[\![\phi_1]\!]}_{\mathcal{K}} \cap pre^R_{\exists}(Z))} \\
&= \nu Z.\overline{([\![\phi_2]\!]_{\mathcal{K}} \cup (\overline{[\![\phi_1]\!]}_{\mathcal{K}} \cap pre^R_{\exists}(\overline{Z})))} \\
&= \nu Z.\overline{[\![\phi_2]\!]}_{\mathcal{K}} \cap \overline{(\overline{[\![\phi_1]\!]}_{\mathcal{K}} \cap \overline{pre^R_{\forall}(Z)})} \\
&= \nu Z.\overline{[\![\phi_2]\!]}_{\mathcal{K}} \cap ([\![\phi_1]\!]_{\mathcal{K}} \cup pre^R_{\forall}(Z))
\end{aligned}$$

We have already mentioned in Section II-C that the greatest fixpoint operator has the property $\nu x.f(x) = \sim \mu x. \sim f(\sim x)$ over an algebra $\mathfrak{B}$ and that the powerset of S, i.e. 2^S, employed with the set inclusion operation ($\subseteq$) forms a Boolean algebra $(2^S, \cap, \cup, \sim, \emptyset, S)$. ■

Notice that the least fixpoints correspond to eventualities while the greatest fixpoints to safety (i.e. a property that should always be satisfied). One other thing to note is the abuse of notation in the fixpoint transformers given in Lemma 4.1. The variable Z represents sets of states whereas the temporal operators operate on temporal logic formulas. The way to think about these fixpoint equations is that the states in the set Z are identified with the temporal logic formula they satisfy. If we were to use, though, the membership functions that were introduced in Section III-B.3, then we could write the fixpoint equations as they are actually used in the symbolic model checking algorithm.

Let us now explain why actually this approach is called symbolic. Assume that we have a set of state symbols SS (or better state variables) such that $SS \cap AP = \emptyset$ and that $|S| = 2^m$ (without loss of generality), then we can encode all the states of the Kripke structure using m symbols from SS. Let $v \subseteq SS$ with $|v| = m$, then each state $s \in S$ can be represented by a characteristic function (that is there exists a map $g : \{0,1\}^m \to \{0,1\}$) which is build on symbols from the set v using the Boolean operators. Actually, we denote each state $s \in S$ by the subset of symbols of v that are true on s or equivalently by the corresponding binary vectors. Note that most of the authors prefer to use an encoding where a state s is denoted by the set $v = u \cup u'$ where $u \subseteq SS$ with $|u| \leq m$ and $u' = \{\neg a \mid a \in v \backslash u\}$. The latter approach helps the OBDD representation of a set of states. For example, assume that two states s_1 and s_2 are encoded as $\{a, b\}$ and $\{a, \neg b\}$ and that $\mathbb{S}_1 : \{0,1\}^2 \to \{0,1\}$ and $\mathbb{S}_2 : \{0,1\}^2 \to \{0,1\}$ represent the Boolean functions that encode the two states. Then for the state s_1 we

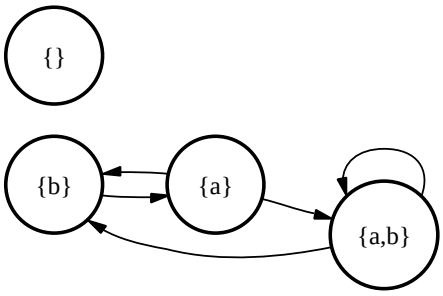

Fig. 7. Encoding the Kripke structure with state variables

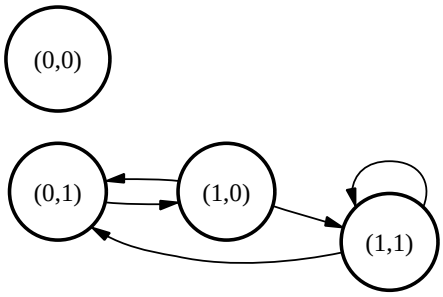

Fig. 8. Encoding the Kripke structure with Boolean vectors

have $\mathbb{S}_1(a,b) = a \wedge b$, for the state s_2, $\mathbb{S}_2(a,b) = a \wedge \neg b$ and, finally, the set of states $S_{12} = \{s_1, s_2\}$ can be represented symbolically as $\mathbb{S}_{12}(a,b) = (a \wedge b) \vee (a \wedge \neg b) = a$. We can also use the symbols from the set of atomic propositions (AP) instead of some new set SS, but we have no guarantees that for all $s_1, s_2 \in S$ it is the case that $\mathcal{O}(s_1) \neq \mathcal{O}(s_2)$.

In a similar way we can encode the transition relation R. In this case, we need two sets of boolean variables, one to encode the starting state and the other the final. Let the two boolean sets be v and v' (again with $|v| = |v'| = m$), then the transition relation will be denoted by $\mathbb{R}(v, v')$. Each Boolean transition function is built on the state variables that encode the starting state in conjunction with the set of state variables that encode the final state. An example will make the notion more concrete. Assume that we have a transition from state s_1 to state s_2 (as introduced above), then the transition will be encoded by $\mathbb{R}(a, b, a', b') = (a \wedge b) \wedge (a' \wedge \neg b')$. Propositional formulas have an exponential number of satisfying assignments and thus they can encode an exponential number of states or transitions.

Example 4.1 (Encoding a Kripke Structure): We will now present an example from [57]. Assume that we have a Kripke structure with 3 states $\{s_0, s_1, s_2\}$ as in Figure 4. Note that due to the way we encode the states, we are forced to add to the structure an extra state that does not alter in any way the behaviour of the system (see Figures 7 and 8). The transition relation is encoded as follows: $\mathbb{R}(a, b, a', b') = [(a \wedge b \wedge a' \wedge b') \vee (a \wedge b \wedge \neg a' \wedge b') \vee (\neg a \wedge b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge b') \vee (a \wedge \neg b \wedge \neg a' \wedge b')]$. There is one disjunct for each transition in the system. Even though this might not seem an efficient way to encode the transition relation, the encoding actually reduces to $\mathbb{R}(a, b, a', b') = (a \wedge b') \vee (\neg a \wedge b \wedge a' \wedge \neg b')$. Verify that $\mathbb{R}((1,1),(0,1)) = 1$ and that $\mathbb{R}((0,1),(0,1)) = 0$.

Hence, we can encode the membership function of a set of states that satisfy a formula ϕ by a Boolean function $\|\phi\| : \{0,1\}^m \rightarrow \{0,1\}$ built on a set of state variables $v \subseteq SS$ with $|v| = m$. If we also define $Z : \{0,1\}^m \rightarrow \{0,1\}$, $\|EXZ\|(v) = (\exists v').(\mathbb{R}(v, v') \wedge Z(v'))$ and $\|AXZ\|(v) = (\forall v').(\mathbb{R}(v, v') \rightarrow Z(v'))$, then we can rewrite the fixpoint equations of the basic temporal operators as:

$$\begin{aligned} \|EG\phi\| &= \nu Z.\|\phi\| \wedge \|EX\, Z\| \\ \|E[\phi_1 \mathcal{U} \phi_2]\| &= \mu Z.\|\phi_2\| \vee (\|\phi_1\| \wedge \|EX\, Z\|) \end{aligned}$$

Now that we have introduced the CTL temporal operators as fixpoints, we can give a high level description of the model checking algorithm. The algorithmic procedure for symbolic CTL model checking is tightly intergraded with the OBDD representation of the Kripke structure and the specification. Here, we ignore the OBDD representation issues and we assume that the membership function $\|\phi\|$ is a Boolean function. We will denote the recursive procedure for determining the satisfiability of formula ϕ on a structure $\mathcal{K}$ by $Check_{\mathcal{K}}()$. $Check_{\mathcal{K}}$ takes as argument a specification in CTL and returns an OBDD (not unique) that describes the set of states where ϕ is true. We define $Check_{\mathcal{K}}$ inductively on the structure of the CTL specification (see Algorithm 2). For example, the cases $\neg\phi$ and $\phi \vee \psi$ can be handled by the standard algorithms for computing negation and disjunction with OBDD structures. Also in the case of an atomic proposition $a \in AP$, $Check_{\mathcal{K}}(a)$ returns an OBDD that represents the set of states where a holds (actually it returns the function $\|a\|$). Here, we will discuss only the temporal operators EX, EG and EU and their algorithmic procedures.

Algorithm 2 The CTL symbolic model checking algorithm

procedure $Check_{\mathcal{K}}(\phi)$ ▷ $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ is the Kripke Structure
  Case ϕ
    $a \in AP$ **return** $\|a\|_{\mathcal{K}}$ ▷ $\|a\|_{\mathcal{K}}$ is in OBDD form
    $\neg\phi_1$ **return** $\neg Check_{\mathcal{K}}(\phi_1)$ ▷ OBDD negation operator
    $\phi_1 \vee \phi_2$ **return** $Check_{\mathcal{K}}(\phi_1) \vee Check_{\mathcal{K}}(\phi_2)$ ▷ OBDD disjunction operator
    $EX\phi_1$ **return** $CheckEX(Check_{\mathcal{K}}(\phi_1))$ ▷ $CheckEX(\|\phi\|(v)) = (\exists v').(\mathbb{R}(v, v') \wedge \|\phi\|(v'))$
    $EG\phi_1$ **return** $FxPoint(1, f_{Check_{\mathcal{K}}(\phi_1)}(Z))$ ▷ $f_{\|\phi_1\|}(Z) = \|\phi_1\| \wedge \|EX\ Z\|$
    $E[\phi_1 \mathcal{U} \phi_2]$ **return** $FxPoint(0, f_{Check_{\mathcal{K}}(\phi_1), Check_{\mathcal{K}}(\phi_2)}(Z))$ ▷ $f_{\|\phi_1\|,\|\phi_2\|}(Z) = \|\phi_2\| \vee (\|\phi_1\| \wedge \|EX\ Z\|)$
  End Case
end procedure

Let these procedures be denoted by $CheckEX()$, $CheckEG()$ and $CheckEU()$ respectively. They take as inputs the states where the subformulae hold and they return a new set of states. The temporal operator $EX\phi$ has straightforward implementation using Quantified Boolean Formulas (QBF). We are looking for a state that has a successor state where ϕ is true:

$$CheckEX(\|\phi\|(v)) = (\exists v').(\mathbb{R}(v, v') \wedge \|\phi\|(v'))$$

where for $x \in SS$ we have $(\exists x).(f(x)) = f(0) \vee f(1)$ and $(\forall x).(f(x)) = f(0) \wedge f(1)$. The $CheckEG()$ and $CheckEU()$ procedures are easily implemented using the fixpoint algorithm (Algorithm 1).

Definition 4.1 (CTL Model Checking): Let $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ be a Kripke structure and ϕ be a CTL formula. We say that the structure $\mathcal{K}$ satisfies the CTL specification ϕ and we write $\mathcal{K} \models \phi$ if and only if $S_0 \subseteq [\![\phi]\!]_{\mathcal{K}}$.

Since a formula ϕ has at most $|\phi|$ subformulae and Algorithm 2 starts from the most deeply nested subformula, the procedure $Check_{\mathcal{K}}()$ is called at most $O(\phi)$ times. Each call of $Check_{\mathcal{K}}()$ is dominated by the fixpoint Algorithm 1 which converges within S iterations and/or the quantification within the EX temporal operator which can take at most $O(|\mathcal{K}|) = O(|S| + |R|)$ time. Hence, the worst case running time is $O(|S| \times |K| \times |\phi|)$. But actually, it can be proven that the upper bound of the running time for the CTL model checking algorithm is bilinear in the size of the formula and the Kripke structure and, furthermore, that the problem is P-*hard* [20].

Theorem 4.2 (CTL Model Checking Running Time [19]): There exists an algorithm that can decide the satisfiability of a CTL formula ϕ with respect to a Kripke structure $\mathcal{K}$, i.e. $\mathcal{K} \models \phi$, in $O(|\mathcal{K}| \times |\phi|)$ time.

B. LTL Automata Theoretic Model Checking

The presentation of the LTL automata theoretic model checking is going to be fairly brief as this section is only needed for presenting later on the CTL* model checking algorithm. Also, it seems that LTL is not the preferred approach for the multi-valued model checking.

The LTL model checking is based on language inclusion. First, we convert the Kripke structure $\mathcal{K}$ that describes the system to an automaton $\mathcal{K}_A$. We want to check whether the language $Lang(\mathcal{K}_A)$ of the automaton $\mathcal{K}_A$ is a subset of the language $Lang(B)$ of the Büchi automaton B, i.e. $Lang(\mathcal{K}_A) \subseteq Lang(B)$. If this is the case then we know that all the traces generated by the automaton $\mathcal{K}_A$ (Kripke structure) are allowed by the specification formula. In order to check the language inclusion, we check whether the intersection of $Lang(\mathcal{K}_A)$ with the complement of the language $Lang(B)$ is empty, that is whether $Lang(\mathcal{K}) \cap \overline{Lang(B)} = \emptyset$ (where $\overline{Lang(B)} = \Sigma^\omega - Lang(B)$). If this is true, then we know that the automaton (Kripke structure) and the negation of the specification do not have any common traces and hence the Kripke structure satisfies the initial specification.

Now, consider the automaton A that derives from the product of $\mathcal{K}_A$ and the Büchi automaton $B^\neg$ that represents the negation of the temporal logic specification formula ϕ. The LTL model checking problem reduces to the problem of finding the accepting executions of the automaton A. Informally, the Büchi automaton $B^\neg$ restricts the behaviour of $\mathcal{K}_A$ by permitting only certain non-acceptable transitions. Our goal is to find the finite or infinite paths of the automaton A that belong to the language $Lang(A)$. If $Lang(A) = \emptyset$, then we know that the model satisfies the specification. The non-emptiness problem of the language $L(A)$ can be solved as described in Section III-D. For further details on the LTL model checking see [36], [19]. The complexity of the

Algorithm 3 High Level Description of LTL Model Checking Procedure

```
1: procedure LTLCHECK(K, φ)
2:     K_A ← Kripke2Automaton(K)                    ▷ Convert the Kripke structure to automaton
3:     B^¬ ← LTL2Buechi(¬φ)                         ▷ Convert the negation of the specification to Büchi automaton
4:     A ← Product(K, B)
5:     SCC_A ← StronglyConnectedComponents(A)
6:     for ∀a_0 ∈ A_0 do
7:         Tree ← DepthFirstSearch(A, a_0)
8:         if (∃path ∈ Tree).(∃t ∈ path).(∃scc ∈ SCC_A).(∃s ∈ scc).(s = t) then
9:             return Path and scc                  ▷ Return counter-example, K ⊭ φ
10:        end if
11:    end for
12:    print(Ok)                                    ▷ K ⊨ φ
13: end procedure
```

algorithm is linear in the size of the Kripke structure and exponential in the size of the formula (see Section III-D).

Theorem 4.3 (LTL Model Checking Running Time [19]): There exists an algorithm that can decide the satisfiability of an LTL formula ϕ with respect to a Kripke structure $\mathcal{K}$, i.e. $\mathcal{K} \models \phi$, in $|\mathcal{K}| \cdot 2^{O(\phi)}$ time.

C. CTL* Model Checking

In a nutshell, the CTL* model checking algorithm is a combination of the LTL and CTL model checking algorithms [19]. The basic algorithm is a recursive procedure that works bottom-up on the structure of the formula. First, we preprocess the CTL* formula so as it contains only existential path quantifiers E using the conversion $E\phi = \neg A \neg \phi$. At each recursion we determine the state subformulas, if they are in CTL form we apply the CTL model checker and if they are in LTL form, i.e. $E\phi$, we apply the LTL model checker. After all the state subformulas have been verified, we replace them with new atomic propositions both in the CTL* formula and the labelling of the model and repeat the procedure at the next level.

The running time of CTL* model checking algorithm depends on the running times of the LTL and CTL model checking algorithms. As the CTL running time is linear in both the size of the model and the size of the formula, the CTL* model checking is dominated by the running time of the LTL model checker.

Theorem 4.4 (CTL* Model Checking Running Time [19]): There exists an algorithm that can decide the satisfiability of an CTL* formula ϕ with respect to a Kripke structure $\mathcal{K}$, i.e. $\mathcal{K} \models \phi$, in $|\mathcal{K}| \cdot 2^{O(\phi)}$ time.

D. A Naive Model Checking Algorithm for the μ-calculus

When we model check a μ-calculus formula ϕ with respect to a Kripke structure $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$, we are trying to determine whether $S_0 \subseteq [\![\phi]\!]_{\mathcal{K}}$. If the answer is positive then we know that the model satisfies the specification. The basic μ-calculus model checking algorithm is a direct implementation of the semantics of the μ-calculus operators presented in Definition 3.4. The pseudocode for the algorithmic procedure $Check_{\mathcal{K}}(\phi, e)$ appears in Algorithm 4. $Check_{\mathcal{K}}(\phi, e)$ takes as inputs the μ-calculus formula ϕ, the environment e and, implicitly, the Kripke structure $\mathcal{K}$ and returns the set of states that satisfy it.

If we assume that the sets of states are represented by Boolean arrays of size $|S|$, (for example $A[s] = 1$ iff s is in set A), then the operations of union, intersection and complementation take $O(|S|)$ time. Similarly, if we encode the transition relation R by the list of pairs of states that belong to R, then the predecessor set can be computed in $O(|R|)$ time as we have to go through the whole list due to the quantifiers. In practical implementations, though, for the computation of the predecessor set, we also have a $O(|S|)$ overhead. Hence, the total running time for calculating the predecessor set is $O(|S| + |R|) = O(|\mathcal{K}|)$. Notice that in the naive algorithm, the running time is dominated by the nesting of the fixpoint operators. Each fixpoint operator, for example $\phi = \mu X.\psi$, takes at maximum $|S|$ iterations before it converges. Imagine now that there exists a nested fixpoint operator within the subformula ψ, then the nested fixpoint operator is going to be called $O(|S|)$ times,

Algorithm 4 The Naive μ-Calculus Model Checking Algorithm

procedure $Check_{\mathcal{K}}(\phi, e)$
  Case ϕ
    $a \in AP$ **return** $[\![a]\!]_{\mathcal{K}} e$
    $\neg\phi_1$ **return** $S \backslash Check_{\mathcal{K}}(\phi_1, e)$
    $\phi_1 \vee \phi_2$ **return** $Check_{\mathcal{K}}(\phi_1, e) \cup Check_{\mathcal{K}}(\phi_2, e)$
    $\phi_1 \wedge \phi_2$ **return** $Check_{\mathcal{K}}(\phi_1, e) \cap Check_{\mathcal{K}}(\phi_2, e)$
    $\Diamond\phi_1$ **return** $pre^{R}_{\exists}(Check_{\mathcal{K}}(\phi_1, e))$
    $\Box\phi_1$ **return** $pre^{R}_{\forall}(Check_{\mathcal{K}}(\phi_1, e))$
    $\mu X.\phi_1(X)$
      $P \leftarrow \{\}$
      repeat
        $Q \leftarrow P$
        $P \leftarrow Check_{\mathcal{K}}(\phi_1, e[X \leftarrow Q])$
      until $Q = P$
      return P
    $\nu X.\phi_1(X)$ **return** $Check_{\mathcal{K}}(\neg\mu X.\neg\phi_1(\neg X))$
  End Case
end procedure

each time with different initial conditions, and, thus, leading to $O(|S|^2)$ iterations. Hence, if there exist n nested fixpoint operators the total number of iterations is going to be $O(|S|^n)$. Let us denote the nesting of fixpoint operators in formula ϕ by $nest(\phi)$. $nest(\phi)$ is recursively defined for μ-calculus formulas ϕ_1, ϕ_2 and $a \in AP$ as:

$$nest(a) := 0 \qquad nest(\neg\phi_1) := nest(\phi_1)$$
$$nest(\phi_1 \wedge \phi_2) := max(nest(\phi_1), nest(\phi_2)) \qquad nest(\phi_1 \vee \phi_2) := max(nest(\phi_1), nest(\phi_2))$$
$$nest(\Diamond\phi_1) := nest(\phi_1) \qquad nest(\Box\phi_1) := nest(\phi_1)$$
$$nest(\mu X.\phi_1) := nest(\phi_1) + 1 \qquad nest(\nu X.\phi_1) := nest(\phi_1) + 1$$

As there can exist only $O(|\phi|)$ fixpoint operators and each iteration takes $O(|\mathcal{K}|)$ time, we can derive the following theorem.

Theorem 4.5 (μ-Calculus Model Checking Running Time [57]): There exists an algorithm that can decide the satisfiability of a μ-calculus formula ϕ with respect to a Kripke structure $\mathcal{K}$ in $O(|\mathcal{K}| \times |\phi| \times |S|^{nest(\phi)})$ time.

Remark 4.1: There do exist, though, better algorithms that try to minimise the fixpoint nestings. For an overview see [57]. Also note that the μ-calculus model checking problem is in $NP \cap coNP$, this implies that a non-deterministic machine can solve the model checking problem in polynomial time.

E. Model Checking in Practice

Out of the variety of model checking tools that have been developed over the years, the most dominant ones are:

- NuSMV[5] [17] which is based on symbolic model checking techniques and is mainly targeted for CTL model checking problems.
- Spin[6] [36] which uses an automaton approach to the model checking problem and accepts only LTL formulas.

Both toolboxes support hierarchy and composition, multiple agents, generation of counterexamples in case the temporal formula is invalidated, nondeterministic environments and so on. Of course, there are also several differences between the two toolboxes mainly concerning the way they deal with the state explosion problem, the user interface and the fact that Spin only supports asynchronous communication among the agents. Furthermore, Spin gives us the option of using breadth first search (BFS) or depth first search (DFS) in the generation of

[5]http://nusmv.irst.itc.it/

[6]http://spinroot.com/spin/whatispin.html

counter examples. This allows for the generation of traces that are optimal in the sense of minimum number of transitions (trace length).

On the other hand, NUSMV can handle both CTL and LTL model checking. NUSMV was initially based on SMV [51] but since then has been extended and offers some additional functionalities, like: modular, open, model-checked and well-documented software architecture, LTL model checking extended with past operators (by a reduction to CTL model checking with fairness constraints), and bounded model checking based on SAT solvers.

At the end of the day, the choice of model checking method is problem dependent. Besides the explicit difference in the expressive power of the two temporal logics (CTL and LTL), the user has also to take into account the following facts. Exhaustive CTL model checking is more efficient then LTL as CTL model checking has complexity that is linear in the size of the formula whereas LTL has complexity that is exponential (both techniques have complexity linear in the size of the model). On the other hand, LTL model checking is better suited for exploring errors on the fly.

V. MULTI-VALUED SETS, RELATIONS AND KRIPKE STRUCTURES

Having introduced the necessary theoretical background, we proceed to define multi-valued sets, relations and Kripke structures which are going to be the foundations of the multi-valued model checking. We define the operations of complementation, intersection and backward image over multi valued sets, i.e. sets that the membership function takes values over a lattice. In this section, the presentation of the material combines results from various sources [12], [10], [42] as common grounds have not yet been established.

A. Multi-Valued Sets and Multi-Valued Relations

In the classical notion of sets, the membership of an object to a set is determined by the set's *membership* or *characteristic* function. Assume that we have a set of objects S and that we want to define a subset T of S such that every object in T satisfies a property H. Let the membership function of T be $h : S \to \{0, 1\}$ with definition: $h(s) = 1$ $(true)$ if s satisfies property H and $h(s) = 0$ $(false)$ otherwise. Then the collection of objects of S that constitute the subset T is denoted by $\{s \in S \mid h(s)\}$, which implies that for all the objects $t \in T$ it is the case that $h(t) = 1$. For example, consider S to be nat and the membership function to be $h(s) = (s \leq 5)$, then the set $T = \{s \in S \mid s \leq 5\}$ is $T = \{0, 1, 2, 3, 4, 5\}$.

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

Definition 5.1 (Multi-Valued Sets): Let $\mathfrak{L} = (L, \sqcap, \sqcup)$ be a lattice and S a set of objects, then a *multi-valued set*, denoted by $\mathbb{S}$, is a total function $\mathbb{S} : S \to L$.

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

Definition 5.2 (mv-Union, mv-Intersection, mv-Set Inclusion, mv-Equality): Let $\mathfrak{L} = (L, \sqcap, \sqcup)$ be a lattice, then we define:

$$
\begin{aligned}
(\mathbb{S} \cap_L \mathbb{S}')(x) &:= \mathbb{S}(x) \sqcap \mathbb{S}'(x) && \textit{(mv-intersection)} \\
(\mathbb{S} \cup_L \mathbb{S}')(x) &:= \mathbb{S}(x) \sqcup \mathbb{S}'(x) && \textit{(mv-union)} \\
\mathbb{S} \subseteq_L \mathbb{S}' &:= (\forall x).(\mathbb{S}(x) \sqsubseteq \mathbb{S}'(x)) && \textit{(mv set inclusion)} \\
\mathbb{S} =_L \mathbb{S}' &:= (\forall x).(\mathbb{S}(x) = \mathbb{S}'(x)) && \textit{(mv-equality)}
\end{aligned}
$$

Definition 5.3 (mv-Complementation, De Morgan, mv-Antimonotonicity): Let $\mathfrak{B} = (B, \sqcap, \sqcup, \sim, \bot, \top)$ be an algebra, then the multi-valued set will be the total function $\mathbb{S} : S \to B$. When the values of the set are over an algebra B, then we denote the mv-operations of Definition 5.2 using the subscript B. Now, we can define the

mv-set complement operation using the algebra's complementation operation and, also, derive the De Morgan laws:

$$\begin{aligned}
\overline{\mathbb{S}}(x) &:= \sim(\mathbb{S}(x)) && \textit{(mv-complementation)}\\
\overline{\mathbb{S} \cap_B \mathbb{S}'} &= \overline{\mathbb{S}} \cup_B \overline{\mathbb{S}'} && \textit{(De-Morgan)}\\
\overline{\mathbb{S} \cup_B \mathbb{S}'} &= \overline{\mathbb{S}} \cap_B \overline{\mathbb{S}'} &&\\
\mathbb{S} \subseteq_B \mathbb{S}' &= \overline{\mathbb{S}'} \subseteq_B \overline{\mathbb{S}} && \textit{(antimonotonicity)}
\end{aligned}$$

Note that all the above definitions actually follow the definitions for the algebraization of the classical two-valued logic. Hence, in the special case where the algebra is over the lattice $\mathfrak{L}_2$ we get the classical two valued set theory (see Theorem 2 in [12]). Now that we have established the notion of mv-sets, we proceed to define multi-valued relations (or mv-relations). Defining mv-relations is important as they are necessary for defining Kripke structures with multi-valued transition relations.

Definition 5.4 (Multi-Valued Relations): A *multi-valued relation* $\mathbb{R}$ on sets S and T over a lattice $\mathfrak{L}$ is a function $\mathbb{R} : S \times T \to L$.

B. Multi-Valued Kripke Structures

The extension of the classical notion of Kripke structures to the multi-valued ones (mv-Kripke structures) is straightforward. Note that some authors perform multi-valued model checking on mv-Kripke structures where the predicates take values from an mv-algebra [13], [9], but they keep the transition relation defined over $\mathfrak{B}_2$, while others consider also mv-transition relations [12], [10], [42]. In the following we will denote the multi-valued Kripke structures by $\mathcal{M}$ ($\mathcal{M}$ for Model).

Definition 5.5 (Multi-Valued Kripke Structure): A *multi-valued Kripke structure* (mv-Kripke structure) is a tuple $\mathcal{M} = (S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathfrak{L}, D)$ where:

- S is a finite set of states
- S_0 is the set of initial states
- $\mathbb{R} : S \times S \to L$ is a partial function called the *mv-transition relation*
- AP is a finite set of atomic propositions that take values over the lattice $\mathfrak{L}$
- $\mathbb{O} : S \times AP \to L$ is a total labelling function that maps a pair $(s, a) \in S \times AP$ to some $l \in L$
- $\mathfrak{L}$ is a lattice $(L, \sqcap, \sqcup)$ or an algebra $(L, \sqcap, \sqcup, \sim, \bot, \top)$
- $D \subset L$ is the set of *designated values*

Let $Dom(\mathbb{R})$ denote the domain of the mv-transition relation. Note that $\mathbb{R}$ can be total (as it is usually the case), i.e. $Dom(\mathbb{R}) = S \times S$, and it is defined even for the cases where for $s, t \in \mathbb{R}$ we have $\mathbb{R}(s, t) = \bot$. In order to keep the semantics of the multi-valued logics as close as possible to the two-valued logics, we define the notion of designated values [42]. Let $D \subset L$ be the set of *designated values* and $N := L \backslash D$ the set of *non-designated values*, then we imply that the values in D are the acceptable ones. For example, if $\mathbb{T} : S \to L$ is the characteristic function for a set of states $T \subseteq S$ that satisfy a property ϕ, then for all $t \in T$ it is $\mathbb{T}(t) \in D$. In most of the literature, the authors let $N = \{\bot\}$ and, hence, $D = L \backslash N$. The sets D and N have the following properties:

- The set of designated values D is non-empty and it is upwards closed i.e. $D = D^{\uparrow}$
- The set of non-designated values N is non-empty and it is downwards closed i.e. $N = N^{\downarrow}$

Using the notion of designated values, we define the transition relation $\mathbb{R}$ such that it has at least one transition with value in D for each state $s \in S$, i.e.

$$(\forall s \in S).(\exists s' \in S).((s, s') \in Dom(\mathbb{R}) \wedge \mathbb{R}(s, s') \in D)$$

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

Definition 5.6 (Paths of mv-Kripke structures): Let $\mathcal{M} = (S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathfrak{L}, D)$ be an mv-Kripke structure, then for each $s \in S$:

- $Paths^{all}_{\mathcal{M}}(s) := \{\pi : nat \to S \mid (\pi(0) = s) \wedge (\forall i \in nat).((\pi(i), \pi(i+1)) \in Dom(\mathbb{R}))\}$
- $Paths_{\mathcal{M}}(s) := \{\pi \in Paths^{all}_{\mathcal{M}}(s) \mid (\forall i \in nat).(\mathbb{R}(\pi(i), \pi(i+1)) \in D)\}$

Also, we need to revisit the notion of predecessor states. We need to extend the definition in order to handle multi-valued sets of states. The extension seems to be straightforward, if one follows the standard definition of

existential and universal quantification as introduced for the First Order predicate calculi of non-classical logics [55]. Thus, we replace the existential quantifiers with join operations over the objects of the set and the universal quantifiers with meet operations. Note that the definition of the mv-predecessor membership function reduces to the classical one (Definition 2.16) when the algebra is $\mathfrak{B}_2$ (for a proof see Theorem 3 in [12]).

Definition 5.7 (Predecessor mv-Sets): Let $\mathbb{R} : S_1 \times S_2 \to B$ be an mv-relation defined over an algebra $(B, \sqcap, \sqcup, \sim, \bot, \top)$ and let $\mathbb{Q} : S_2 \to B$ be an mv-set, then for all $s_1 \in S_1$ we define $pre^{\mathbb{R}}_{\exists}(\mathbb{Q}) : S_1 \to B$ and $pre^{\mathbb{R}}_{\forall}(\mathbb{Q}) : S_1 \to B$ as:

$$\begin{aligned} pre^{\mathbb{R}}_{\exists}(\mathbb{Q})(s_1) &:= \bigsqcup_{s_2 \in S_2} (\mathbb{R}(s_1, s_2) \sqcap \mathbb{Q}(s_2)) \\ pre^{\mathbb{R}}_{\forall}(\mathbb{Q})(s_1) &:= \bigsqcap_{s_2 \in S_2} (\mathbb{R}(s_1, s_2) \Rightarrow \mathbb{Q}(s_2)) \end{aligned}$$

The universal and existential predecessor sets are dual, that is, for some $\mathbb{Q} : S_2 \to B$ and for all $s_1 \in S_1$ we have:

$$\begin{aligned} pre^{\mathbb{R}}_{\forall}(\mathbb{Q})(s_1) &= \textstyle\bigsqcap_{s_2 \in S_2} (\mathbb{R}(s_1, s_2) \Rightarrow \mathbb{Q}(s_2)) \\ &= \textstyle\bigsqcap_{s_2 \in S_2} (\sim \mathbb{R}(s_1, s_2) \sqcup \mathbb{Q}(s_2)) \\ &= \textstyle\bigsqcap_{s_2 \in S_2} \sim (\mathbb{R}(s_1, s_2) \sqcap \sim \mathbb{Q}(s_2)) \\ &= \sim \textstyle\bigsqcup_{s_2 \in S_2} (\mathbb{R}(s_1, s_2) \sqcap \sim \mathbb{Q}(s_2)) \\ &= \overline{\textstyle\bigsqcup_{s_2 \in S_2} (\mathbb{R}(s_1, s_2) \sqcap \overline{\mathbb{Q}}(s_2))} \\ &= \overline{pre^{\mathbb{R}}_{\exists}(\overline{\mathbb{Q}})(s_1)} \end{aligned}$$

and vise versa.

Lemma 5.1: The predecessor mv-sets $pre^{\mathbb{R}}_{\exists}$ and $pre^{\mathbb{R}}_{\forall}$ are order preserving.

Proof: Let $\mathbb{R} : S_1 \times S_2 \to B$, $\mathbb{Q} : S_2 \to B$, $\mathbb{P} : S_2 \to B$ and assume $\mathbb{Q} \subseteq_B \mathbb{P}$, then:

$$\begin{array}{cl} \mathbb{Q} \subseteq_B \mathbb{P} & \text{iff (by definition of } \subseteq_B) \\ (\forall s_2 \in S_2).(\mathbb{Q}(s_2) \sqsubseteq \mathbb{P}(s_2)) & \text{implies (by monotonicity of } \sqcap) \\ (\forall s_1 \in S_1).(\forall s_2 \in S_2).(\mathbb{R}(s_1, s_2) \sqcap \mathbb{Q}(s_2) \sqsubseteq \mathbb{R}(s_1, s_2) \sqcap \mathbb{P}(s_2)) & \text{implies (by monotonicity of } \sqcup) \\ (\forall s_1 \in S_1).(\bigsqcup_{s_2 \in S_2} (\mathbb{R}(s_1, s_2) \sqcap \mathbb{Q}(s_2)) \sqsubseteq \bigsqcup_{s_2 \in S_2} (\mathbb{R}(s_1, s_2) \sqcap \mathbb{P}(s_2))) & \text{implies (by definition of } pre^{\mathbb{R}}_{\exists}) \\ (\forall s_1 \in S_1).(pre^{\mathbb{R}}_{\exists}(\mathbb{Q})(s_1) \sqsubseteq pre^{\mathbb{R}}_{\exists}(\mathbb{P})(s_2)) & \text{iff (by definition of } \subseteq_B) \\ pre^{\mathbb{R}}_{\exists}(\mathbb{Q}) \subseteq_B pre^{\mathbb{R}}_{\exists}(\mathbb{P}) & \end{array}$$

The case of $pre^{\mathbb{R}}_{\forall}$ follows easily from the duality property. ■

Example 5.1 (Using the $pre^{\mathbb{R}}_{\exists}$ and $pre^{\mathbb{R}}_{\forall}$ functions): In order to make clear the notion of multi-valued sets and the usage of the predecessor functions, we present an example from [12]. Assume that we have two different (classical) Kripke structures $\mathcal{K}_1$ and $\mathcal{K}_2$ that model a real system. The differences in the two structures can be attributed to the attempts of two independent experts to model the same system. We can merge the two models to get a multi-valued model $\mathcal{M}$ (Figure 9). One can think of the resulting structure $\mathcal{M}$ as a model where we can verify properties that hold despite the possible local inconsistencies in the initial models $\mathcal{K}_1$ and $\mathcal{K}_2$. As an example, consider the case where we want to verify that the property $EX\, b$ is true despite the differences in the two models (see Fig. 12).

In the model $\mathcal{M}$, both the atomic propositions and the transition relation take truth values over the lattice $\mathfrak{L}_{2,2}$ (Fig. 1). It can be easily proven that the lattice $\mathfrak{L}_{2,2}$ extended with a negation operator $\sim$ such that $\sim (1,1) = (0,0)$, $\sim (0,0) = (1,1)$, $\sim (1,0) = (0,1)$ and $\sim (0,1) = (1,0)$ is a Boolean algebra (as a product of two Boolean algebras). In Figure 10, we present some multi-valued sets. Each diagram represents an mv-set that specifies the degree that each state of $\mathcal{M}$ belongs to it. Similarly in Figure 11, we present the mv-transition relation. Finally, Figure 12 presents the results of the computations for the predecessor mv-sets using Definition 5.7.

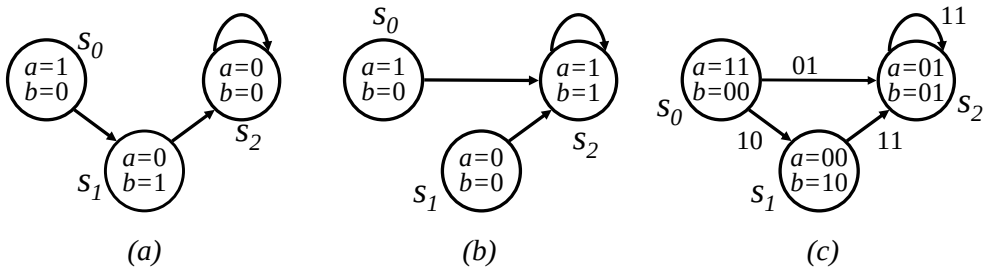

Fig. 9. Two independent experts model a real system as the Kripke structures $\mathcal{K}_1$ and $\mathcal{K}_2$. In this example, the two models are combined by simply merging the states with the same name. Also assume that the initial state of $\mathcal{M}$ is s_0. (a) The structure $\mathcal{K}_1$. (b) The structure $\mathcal{K}_2$. (c) The merged multi-valued model $\mathcal{M}$.

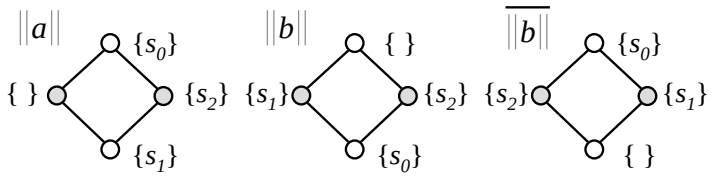

Fig. 10. The multi-valued sets $\|a\|$, $\|b\|$ and $\overline{\|b\|}$. Here, the notation $\|a\|$ with $a \in AP$ represents the multi-valued set that indicates the degree that a state s of $\mathcal{M}$ satisfies the property a.

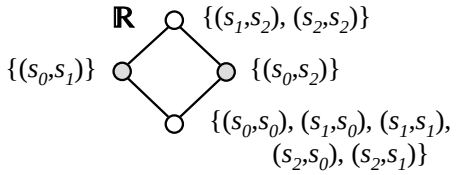

Fig. 11. The mv-transition relation $\mathbb{R}$ of $\mathcal{M}$.

Remark 5.1: When we operate over a lattice $\mathfrak{L}$ with a negation operation different from the one defined for the quasi-Boolean algebras, then we cannot use anymore the implication semantics for $pre^{\mathbb{R}}_{\forall}(\mathbb{Q})$. In this case, we might use the definition:

$$pre^{\mathbb{R}}_{\forall}(\mathbb{Q})'(s_1) := \bigsqcap_{\{s_2 \in S_2 \mid \mathbb{R}(s_1,s_2) \in D\}} (\mathbb{R}(s_1, s_2) \sqcap \mathbb{Q}(s_2))$$

as proposed by Konikowska and Pencek (see [42]). In this case, though, we loose the duality property of the existential and universal predecessor mv-sets. Also note that when the assumptions of Theorem 6.2 hold, then the semantics of $pre^{\mathbb{R}}_{\forall}(\mathbb{Q})$ and $pre^{\mathbb{R}}_{\forall}(\mathbb{Q})'$ coincide, that is, for every $s \in S$ it is $pre^{\mathbb{R}}_{\forall}(\mathbb{Q})(s) \in D$ iff $pre^{\mathbb{R}}_{\forall}(\mathbb{Q})'(s) \in D$ (see Proposition 6.3). Finally, it is important to mention that depending on the application the semantics of $pre^{\mathbb{R}}_{\forall}(\mathbb{Q})$ or $pre^{\mathbb{R}}_{\forall}(\mathbb{Q})'$ could be counter-intuitive (even though in the first case they are mathematically correct). The following example explains why.

Example 5.2 (Comparison of the mv-predecessor semantics): Consider the multi-valued Kripke structure $\mathcal{M}$ in Figure 13. We would like to verify the property $AX\, a$. In this case, the semantics of the formula are given either by $\|AX\, a\| = pre^{\mathbb{R}}_{\forall}(\|a\|)$ or $\|AX\, a\| = pre^{\mathbb{R}}_{\forall}(\|a\|)'$ (see Section VI). Let us compute the truth value of $\|AX\, a\|(s_0)$ assuming that $D = \{1/2, 1\}$:

$$\begin{aligned}
pre^{\mathbb{R}}_{\forall}(\|a\|)(s_0) &= \textstyle\bigsqcap_{s \in S}(\mathbb{R}(s_0, s) \Rightarrow \|a\|(s)) \\
&= (\sim \mathbb{R}(s_0, s_0) \sqcup \|a\|(s_0)) \sqcap (\sim \mathbb{R}(s_0, s_1) \sqcup \|a\|(s_1)) \sqcap (\sim \mathbb{R}(s_0, s_2) \sqcup \|a\|(s_2)) \\
&= (\sim 0 \sqcup 0) \sqcap (\sim 1/2 \sqcup 0) \sqcap (\sim 1/2 \sqcup 0) = 1 \sqcap 1/2 \sqcap 1/2 = 1/2 \\
pre^{\mathbb{R}}_{\forall}(\|a\|)'(s_0) &= \textstyle\bigsqcap_{\{s \in S \mid \mathbb{R}(s_0,s) \in D\}}(\mathbb{R}(s_0, s) \sqcap \|a\|(s)) \\
&= (\mathbb{R}(s_0, s_1) \sqcap \|a\|(s_1)) \sqcap (\mathbb{R}(s_0, s_2) \sqcap \|a\|(s_2)) = (1/2 \sqcap 0) \sqcap (1/2 \sqcap 0) = 0
\end{aligned}$$

Note that in the first case we get $\|AX\, a\|(s_0) = 1/2$ which seems to be counter-intuitive as in all the states it is the case that $\|a\| = 0$. Thus, we would not expect the answer to be $1/2$ (maybe). On the other hand, let us consider again the Example 5.1. In this case, if we compute the truth value of $\|AX\, a\|(s_0)$ while assuming that

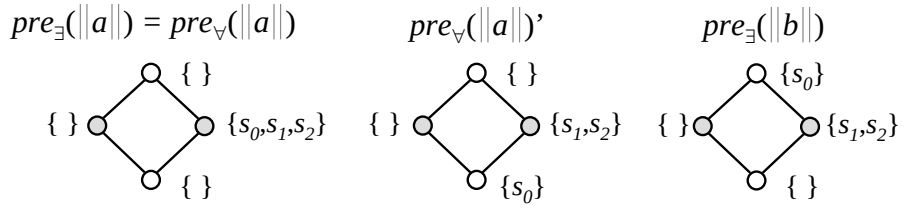

Fig. 12. Some multi-valued predecessor sets. As an example, consider the following computation: $pre^{\mathbb{R}}_{\exists}(\|a\|)(s_1) = (\mathbb{R}(s_1,s_0) \sqcap \|a\|(s_0)) \sqcup (\mathbb{R}(s_1,s_1) \sqcap \|a\|(s_1)) \sqcup (\mathbb{R}(s_1,s_2) \sqcap \|a\|(s_2)) = ((0,0) \sqcap (1,1)) \sqcup ((0,0) \sqcap (0,0)) \sqcup ((1,1) \sqcap (0,1)) = (0,0) \sqcup (0,0) \sqcup (0,1) = (0,1)$. Also, note that $\|EX\, b\|(s_0) = pre^{\mathbb{R}}_{\exists}(\|b\|)(s_0) = (1,1)$.

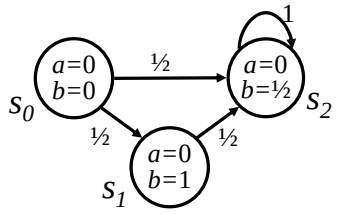

Fig. 13. A multi-valued Kripke structure $\mathcal{M}$ where the atomic propositions and the transition relation take truth values over the quasi-Boolean algebra $\mathfrak{B}_3$.

$D = \{(1,1),(1,0),(0,1)\}$, then we get that:

$$\begin{aligned} pre^{\mathbb{R}}_{\forall}(\|a\|)(s_0) &= (\sim \mathbb{R}(s_0,s_0) \sqcup \|a\|(s_0)) \sqcap (\sim \mathbb{R}(s_0,s_1) \sqcup \|a\|(s_1)) \sqcap (\sim \mathbb{R}(s_0,s_2) \sqcup \|a\|(s_2)) \\ &= (\sim (0,0) \sqcup (1,1)) \sqcap (\sim (1,0) \sqcup (0,0)) \sqcap (\sim (0,1) \sqcup (0,1)) \\ &= (1,1) \sqcap (0,1) \sqcap (1,1) = (0,1) \\ pre^{\mathbb{R}}_{\forall}(\|a\|)'(s_0) &= (\mathbb{R}(s_0,s_1) \sqcap \|a\|(s_1)) \sqcap (\mathbb{R}(s_0,s_2) \sqcap \|a\|(s_2)) \\ &= ((1,0) \sqcap (0,0)) \sqcap ((0,1) \sqcap (0,1)) = (0,0) \end{aligned}$$

The complete mv-sets are presented in Figure 12. Notice that in the first case the result is what we have expected it to be. The fact that $pre^{\mathbb{R}}_{\forall}(\|a\|)(s_0) = (0,1)$ implies that the property holds on $\mathcal{K}_2$ while it fails to hold on $\mathcal{K}_1$.

In order to simplify the notation, we define for each atomic proposition $a \in AP$ a characteristic function that maps each state $s \in S$ to a value in the lattice $\mathfrak{L}$. If we denote this function (mv-set) by $\mathbb{O}_a : S \to L$ with definition

$$\mathbb{O}_a(s) := \mathbb{O}(s,a)$$

then $\mathbb{O}_a$ defines a partition of the state space with respect to the atomic proposition a. In a sense, the characteristic function $\mathbb{O}_a$ maps each state $s \in S$ to the degree that s belongs to mv-set $\mathbb{O}_a$. This is in accordance with the classical two-valued set membership $[\![a]\!]_{\mathcal{K}} = \{s \in S \mid a \in \mathcal{O}(s)\}$. Consider the mv-set $\mathbb{O}_a$ over the $\mathfrak{L}_2$ lattice, then we get that for all s in S either $\mathbb{O}_a(s) = 1$ or $\mathbb{O}_a(s) = 0$ which is actually the predicate $a \in \mathcal{O}(s)$.

VI. Multi-Valued Model Checking

In this section, we discuss some of the approaches to the multi-valued model checking problem. The definition of the problem is a straightforward extension of the classical model checking problem. For the following, we will be using the notation $\|\phi\|_{\mathcal{M}}$ to denote the function (or mv-set) $\|\phi\|_{\mathcal{M}} : S \to L$ that maps a state s of the mv-Kripke structure $\mathcal{M}$ to a truth value from the lattice $\mathfrak{L}$.

Definition 6.1 (The multi-valued Model Checking Problem): Given a multi-valued Kripke structure $\mathcal{M} = (S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathfrak{L}, D)$, a state $s_0 \in S_0$ and a temporal or modal logic formula ϕ determine the degree that the state s_0 satisfies the specification ϕ, that is compute $\|\phi\|_{\mathcal{M}}(s)$.
Alternatively, we could define the mv-model checking problem as: $(\forall s_0 \in S_0).(\|\phi\|_{\mathcal{M}}(s_0) \in D)$.

As mentioned earlier in the introduction, the papers that are going to be presented in this section are [42], [12] and [9]. Even though the purpose of the section is to introduce the aforementioned papers, we try to maintain

a uniform notation not only throughout this section but also throughout the whole paper. Thus, some comments have been added and some minor details have been altered. Furthermore, we give the proofs to all the theorems and lemmas that are not included in the original papers.

A. Reducing Multi-Valued CTL* to CTL* using designated values

The first approach is a reduction method for the multi-valued CTL* which is based on the concept of designated values [42] (see Section V-B). The reduction is quite simple and the intuition behind it is that designated values D are the equivalent of the truth values in the classical model checking. That is, when a formula ϕ evaluates to some $l \in D$, then this is regarded as equivalent to *true* in the two-valued logic, whereas if $l \in N$ it is regarded as *false*. Note that in this reduction method, the only restriction that the authors place on the lattice is that it has to be a c-complete lattice (to that end it could even be infinite). Let us first present the semantics of the Negation Normal Form (NNF) mv-CTL*.

Definition 6.2 (The Semantics of mv-CTL* in NNF): Let the mv-membership function $\|\phi_s\|_{\mathcal{M}} : S \to L$ to denote the degree that a state $s \in S$ satisfies the state specification ϕ_s and, also, $\|\phi_p\|_{\mathcal{M}} : Paths_{\mathcal{M}}(S) \to L$ to denote the degree that a path $\pi \in Paths_{\mathcal{M}}(s)$ starting from state $s \in S$ satisfies the path specification ϕ_p. Let $s \in S$, $\pi \in Paths_{\mathcal{M}}(S)$ and $a \in AP^+$ with $AP^+ = AP \cup \overline{AP}$ and $\overline{AP} = \{\neg a \mid a \in AP\}$, then the multi-valued semantics of state mv-CTL* formulas in NNF with respect to $\mathcal{M} = (S, S_0, \mathbb{R}, AP^+, \mathbb{O}, \mathfrak{L}, D)$ are defined as follows (we sometimes drop the subscript $\mathcal{M}$):

$$
\begin{aligned}
\|a\| &:= \mathbb{O}_a \\
\|\phi_s \vee \psi_s\| &:= \|\phi_s\| \cup_L \|\psi_s\| \\
\|\phi_s \wedge \psi_s\| &:= \|\phi_s\| \cap_L \|\psi_s\| \\
\|E\phi_p\|(s) &:= \bigsqcup_{\pi \in Paths_{\mathcal{M}}(s)} \|\phi_p\|(\pi[0]) \\
\|A\phi_p\|(s) &:= \bigsqcap_{\pi \in Paths_{\mathcal{M}}(s)} \|\phi_p\|(\pi[0])
\end{aligned}
$$

and for $i, j, k \in nat$ the semantics of the path formulas in NNF are :

$$
\begin{aligned}
\|\phi_s\|(\pi[i]) &:= \|\phi_s\|(\pi(i)) \\
\|\phi_p \vee \psi_p\|(\pi[i]) &:= \|\phi_p\|(\pi[i]) \sqcup \|\psi_p\|(\pi[i]) \\
\|\phi_p \wedge \psi_p\|(\pi[i]) &:= \|\phi_p\|(\pi[i]) \sqcap \|\psi_p\|(\pi[i]) \\
\|X\phi_p\|(\pi[i]) &:= \mathbb{R}(\pi(i), \pi(i+1)) \sqcap \|\phi_p\|(\pi[i+1]) \\
\|[\phi_p \mathcal{U} \psi_p]\|(\pi[i]) &:= \|\psi_p\|(\pi[i]) \sqcup \bigsqcup_{j>i} \big(\|\phi_p\|(\pi[i]) \sqcap \bigsqcap_{i<k<j} \big(\mathbb{R}(\pi(k-1), \pi(k)) \sqcap \|\phi_p\|(\pi[k]) \big) \sqcap \\
&\quad \sqcap \big(\mathbb{R}(\pi(j-1), \pi(j)) \sqcap \|\psi_p\|(\pi[j]) \big) \big) \\
\|[\phi_p \mathcal{R} \psi_p]\|(\pi[i]) &:= \|\psi_p\|(\pi[i]) \sqcap \bigsqcap_{j>i} \big(\|\phi_p\|(\pi[i]) \sqcup \bigsqcup_{i<k<j} \big(\mathbb{R}(\pi(k-1), \pi(k)) \sqcap \|\phi_p\|(\pi[k]) \big) \sqcup \\
&\quad \sqcup \big(\mathbb{R}(\pi(j-1), \pi(j)) \sqcap \|\psi_p\|(\pi[j]) \big) \big)
\end{aligned}
$$

Note that here we have used the release operator instead of the more intuitive before operator as the structure of the NNF mv-CTL* does not allow the use of the negation operator. For the definition of the semantics of the until and release operators we have used the usual equivalence between the existential quantifier and the join operation and between the universal quantifier and the meet operation. Note, though, that the quantification in the path operator A takes place over the paths that have designated values and not all the possible states as is in the Definition 5.7. Also, the characteristic function for the operator AX has the definition $\|AX\phi\|(s) = \bigsqcap_{\pi \in Paths_{\mathcal{M}}(s)} (\mathbb{R}(\pi(0), \pi(1)) \sqcap \|\phi\|(\pi(1)))$ instead of the usual semantics based on implication $\|AX\phi\|(s) = \bigsqcap_{t \in S} (\mathbb{R}(s,t) \sqcap \|\phi\|(t)))$ (See Section VI-B).

Definition 6.3 (CTL* Model Checking with Designated Values [42]): Given a multi-valued Kripke structure $\mathcal{M} = (S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathfrak{L}, D)$ over a c-complete lattice $\mathfrak{L}$ and an mv-CTL* state formula ϕ, the model checking problem is defined as checking whether $(\forall s_0 \in S_0).(\|\phi\|_{\mathcal{M}}(s_0) \in D)$.

The method presented here takes advantage of the dichotomy induced from the designated and non-designated values to reduce the mv-CTL* model checking problem to the classical CTL* model checking problem. The first theorem presents a reduction method from the normal negation form (NNF) of the mv-CTL*.

Theorem 6.1 (Reduction from NNF mv-CTL* to CTL* using Designated Values [42]): Assume that $\mathfrak{L}$ is a c-complete lattice. Let the designated values D and non-designated values N be closed under arbitrary bounds, that is, the lower and upper bounds of arbitrary subsets of their elements. Define a translation τ from $\mathcal{M} = (S, S_0, \mathbb{R}, AP^+, \mathbb{O}, \mathfrak{L}, D)$ to $\mathcal{K} = (S, S_0, R, AP^+, \mathcal{O})$ such that:

1) $R = \{(s, s') \in S \times S \mid \mathbb{R}(s, s') \in D\}$
2) for any $a \in AP^+$ it is $\|a\|_{\mathcal{K}}(s) = 1$ iff $\|a\|_{\mathcal{M}}(s) \in D$

Then for any state formula ϕ_s and any path formula ϕ_p of NNF mv-CTL* over the lattice L and any state $s \in S$ and path $\pi \in Paths_{\mathcal{M}}(s)$ of $\mathcal{M}$, we have:

$$\begin{aligned} \|\phi_s\|_{\mathcal{M}}(s) \in D \quad &\text{iff} \quad (\mathcal{K}, s) \models_s \phi_s \\ \|\phi_p\|_{\mathcal{M}}(\pi[0]) \in D \quad &\text{iff} \quad (\mathcal{K}, \pi[0]) \models_p \phi_p \end{aligned}$$

Proof: As the whole proof appears in [42], we will only present some comments and a few cases from the induction. First note that due to the definition of the transition relation we have $Paths_{\mathcal{M}}(s) = Paths_{\mathcal{K}}(s)$ for all $s \in S$. Second, for any subset L_s of L the following properties hold (because D and N are closed under arbitrary bounds):

1) $\bigsqcup L_s \in D$ iff $(\exists l \in L_s).(l \in D)$
2) $\bigsqcap L_s \in D$ iff $(\forall l \in L_s).(l \in D)$

The proof of the theorem is an easy induction on the structure of the formula ϕ. Some of the cases are as follows for some $s \in S$ and $\pi \in Paths_{\mathcal{M}}(s)$:

- $\phi = a$ with $a \in AP^+$, then it holds by definition.
- $\phi = \phi_1 \vee \phi_2$, then $\|\phi\|_{\mathcal{M}}(s) = (\|\phi_1\|_{\mathcal{M}}(s) \sqcup \|\phi_2\|_{\mathcal{M}}(s)) \in D$ iff (by property 1) $(\exists i).(\|\phi_i\|_{\mathcal{M}}(s) \in D)$ iff (by induction hypothesis (IH)) $(\exists i).((\mathcal{K}, s) \models_s \phi_i$ iff $(\mathcal{K}, s) \models_s \phi_1 \vee \phi_2 = \phi$.
- $\phi = [\phi_1 \,\mathcal{U} \phi_2]$, then we have $\|\phi\|_{\mathcal{M}}(\pi[i]) \in D$. If $j = i$, then $\|\phi_2\|(\pi[i]) \in D)$ iff (by IH) $(\mathcal{K}, \pi[i]) \models_p \phi_2$ iff $(\mathcal{K}, \pi[i]) \models_p [\phi_1 \,\mathcal{U} \phi_2]$. The case where $j = i + 1$ is similar to the previous one, so we omit it. Now assume that $j > i$, then by property 1 we have that: $(\exists j).((R(\pi(j-1), \pi(j)) \sqcap \|\phi_2\|_{\mathcal{M}}(\pi[j])) \in D$ iff $\|\phi_2\|_{\mathcal{M}}(\pi[j]) \in D$ because $R(\pi(j-1), \pi(j)) \in D$ and D is closed under arbitrary bounds; and, also, by property 2 we have that: $(\forall k \in (0, j)).((R(\pi(k-1), \pi(k)) \sqcap \|\phi_1\|_{\mathcal{M}}(\pi[k])) \in D)$ iff $\|\phi_1\|_{\mathcal{M}}(\pi[k]) \in D$. Now using the induction hypothesis, we get that on the same path π it is $(\mathcal{K}, \pi[j]) \models_p \phi_2$ and for all $0 < k < j$ we have $(\mathcal{K}, \pi[k]) \models_p \phi_1$ which by definition is $(\mathcal{K}, \pi[i]) \models_p [\phi_1 \,\mathcal{U} \phi_2] = \phi$.
- $\phi = E\,\psi$, then $\|E\,\psi\|_{\mathcal{M}}(s) \in D$ iff (by property 1) $(\exists \pi \in Paths_{\mathcal{M}}(s)).(\|\psi\|_{\mathcal{M}}(\pi[0])\| \in D)$ iff (by IH) $(\exists \pi \in Paths_{\mathcal{K}}(s)).((\mathcal{K}, \pi[0]) \models_p \psi$ iff (by definition) $(\mathcal{K}, s) \models_s E\,\psi = \phi$.

■

Note that by using the NNF of mv-CTL*, we might lose in expressive power due to the unspecified negation operation over the lattice $\mathfrak{L}$. The authors in [42] consider the case, though, where the complement operator is such that it converts designated values to non-designated and vise versa. If we replace $a \in AP^+$ with $a \in AP$, add the rule $\|\neg\phi_x\| :=\sim \|\phi_x\|$ and remove the dual operators, then we get the semantics of the mv-CTL* where the negation operator can appear anywhere in the formula. The following theorem deals with this case.

Theorem 6.2 (Reduction from mv-CTL* to CTL* using Designated Values [42]): Assume that $\mathfrak{L}$ is a c-complete lattice, then:

1) Let the designated values D and non-designated values N be closed under arbitrary bounds.
2) $l \in D$ implies $\sim l \in N$ and $l \in N$ implies $\sim l \in D$

Define a translation τ from $\mathcal{M} = (S, S_0, \mathbb{R}, AP, \mathbb{O}, L, D)$ to $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ such that:

1) $R = \{(s, s') \in S \times S \mid \mathbb{R}(s, s') \in D\}$

2) for any $a \in AP$ it is $\|a\|_{\mathcal{K}}(s) = 1 \quad iff \quad \mathbb{O}_a(s) \in D$

Then for any state formula ϕ_s and any path formula ϕ_p of mv-CTL* over the lattice L and any state $s \in S$ and path $\pi \in Paths_{\mathcal{M}}(s)$ of $\mathcal{M}$, we have:

$$\|\phi_s\|_{\mathcal{M}}(s) \in D \quad \text{iff} \quad (\mathcal{K}, s) \models_s \phi_s$$
$$\|\phi_p\|_{\mathcal{M}}(\pi[0]) \in D \quad \text{iff} \quad (\mathcal{K}, \pi[0]) \models_p \phi_p$$

Proof: Here, we discuss the case $\phi_s = \neg\psi_s$. Let $s \in S$, then we have $\|\neg\psi_s\|_{\mathcal{M}}(s) = \sim \|\psi_s\|_{\mathcal{M}}(s) \in D$ or that $\|\psi_s\|_{\mathcal{M}}(s) \in N$. Using the induction hypothesis we get that $(\mathcal{K}, s) \not\models_s \psi_s$ or that $(\mathcal{K}, s) \models_s \neg\psi_s = \phi_s$. ■

Remark 6.1: Due to the above reduction procedures the complexity of the multi-valued CTL* model checking is the same as for the classical two-valued CTL* model checking.

Remark 6.2: A counter example or witness in the two valued model checking problem is also a counter example or witness in the multi-valued case. That is, as long as we discard the exact value of the formula in the multi-valued case and we are only concerned whether $\|\phi\|_{\mathcal{M}}(s) \in D$ or not.

Example 6.1: Some logics that satisfy the conditions of Theorem 6.1 are those that take values over finite linear orders. Assume that we have a lattice $(L, \leq)$ such that $L = \{0, 1, 2, \ldots, k\}$, $\bot = 0$ and $\top = k$, then if we let $D = d^{\uparrow}$ and $N = L \backslash D$ for some $1 \leq d \leq k$, both D and N will be closed under arbitrary bounds as they are finite. Some logics that are based on finite linear orders are the following: the three-valued Kleene logic, the many-valued Lukasiewicz logic, the finite-valued Gödel logics and the many-valued Rosser-Turquette logics. For a detailed exposition of the above logics the reader is referred to [30].

Example 6.2: Some logics that satisfy the conditions of Theorem 6.2 are the following.

- Logics over finite linear orders:
  - *Many-valued Rosser-Turquette logics* with the complement operator defined for $l \in L$ as: $\sim l = \top$ if $l < d$ and $\bot$ otherwise. In this case, $\sim l = \top$ iff $l \in N$ and $\sim l = \bot$ iff $l \in D$.
  - *Gödel logic* with the complement operator defined for $l \in L$ as: $\sim l = \top$ iff $l = \bot$ and $\sim l = \bot$ iff $l \neq \bot$. Then the second assumption in Theorem 6.2 is satisfied iff $d = 1$ (i.e. all the non-zero numbers are designated values).
  - *Many-valued Lukasiewicz logic* with the complement operator defined for $l \in L$ as: $\sim l = k - l$. Then the second assumption in Theorem 6.2 is satisfied iff $k = 2d - 1$ (i.e. the set L must have an even number of elements).
- The multi-valued logic over the lattice $\mathfrak{L}_{2,2}$ of Fig. 1 with the complement operator defined as: $\sim (1,1) = (0,0)$, $\sim (0,0) = (1,1)$, $\sim (1,0) = (0,1)$ and $\sim (0,1) = (1,0)$. Note that in this case $(1,0)$ and $(0,1)$ cannot be both in the designated D or the non-designated N values as, then the set D or N would not be closed under arbitrary bounds. For example, if $D = \{(1,1), (1,0), (0,1)\}$, then $(1,0) \sqcap (0,1) = (0,0) \notin D$. Also, in this case the second assumption of the theorem is violated as $(1,0) \in D$ and $\sim (1,0) = (0,1) \in D$. Hence, the only two possible partitions of $\mathfrak{L}$ are: (i) $D = \{(1,1), (1,0)\}$ and $N = \{(0,1), (0,0)\}$ and (ii) $D = \{(1,1), (1,0)\}$ and $N = \{(0,1), (0,0)\}$.

Example 6.3: Let us consider again the mv-Kripke structure $\mathcal{M}$ of Example 5.1 (Fig. 9.(c)). We are asked to verify the property $EG\, a$ when the truth values of interest (designated values) are in the set $D = \{(1,1), (0,1)\}$. Using the translation τ of Theorem 6.2 for the set D, we get the classical Kripke structure $\mathcal{K}_2$ in Figure 9.(b). Hence, $\|EG\, a\|_{\mathcal{M}}(s_0) \in D$ iff $(\mathcal{K}_2, s_0) \models EG\, a$. It is easily verifiable that $\|EG\, a\|_{\mathcal{K}_2}(s_0) = 1$ and, thus, the property holds either on both models ($\mathcal{K}_1$ and $\mathcal{K}_2$) or only on the second one ($\mathcal{K}_2$).

Proposition 6.3: Let $\mathcal{M} = (S, S_0, \mathbb{R}, AP^+, \mathbb{O}, \mathfrak{L}, D)$ be an mv-Kripke structure and ϕ be an mv-CTL* formula. If the assumptions of Theorem 6.2 hold, then for every state $s \in S$ it is:

$$pre^{\mathbb{R}}_{\forall}(\|\phi\|)(s) \in D \text{ iff } pre^{\mathbb{R}}_{\forall}(\|\phi\|)'(s) \in D$$

Proof: Let $s, t \in S$ with $\mathbb{R}(s,t) \in D$, then:

($\Rightarrow$) $(\mathbb{R}(s,t) \sqcap \|\phi\|(t)) \in D$ implies that $\|\phi\|(t) \in D$. It is $\sim \mathbb{R}(s,t) \in N$, hence $(\sim \mathbb{R}(s,t) \sqcup \|\phi\|(t)) \in D$ because D is upwards closed.

($\Leftarrow$) $(\sim \mathbb{R}(s,t) \sqcup \|\phi\|(t)) \in D$ implies that $\|\phi\|(t) \in D$ by property 1 in the proof of Theorem 6.1. Hence, $(\mathbb{R}(s,t) \sqcap \|\phi\|(t)) \in D$ as D is closed under arbitrary bounds.

■

Remark 6.3: The authors in [42] also extend the bisimulation relation of two-valued models to the many-valued case and prove that mv-CTL* does not distinguish between two bisimilar many-valued models.

B. Direct Multi-Valued CTL Symbolic Model Checking

The symbolic model checking methods (See Section IV-A) have been the flagship of the formal verification techniques. Here, we give a brief presentation of the extension of the classic two-valued symbolic model checking [51] to the many-valued case over quasi-Boolean algebras [31]. The section begins by defining the semantics of the multi-valued CTL (mv-CTL) and concludes by giving some comments on the running time of the basic algorithm.

Let $\mathcal{M} = (S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathfrak{B}, D)$ be the multi-valued model of the system. In [12], the authors set $Dom(\mathbb{R}) = S \times S$ and $N = \{\bot\}$, hence there exists at least one non-bottom transition out of every state. Also, they prefer the double-brace notation $[\![.]\!]$ for denoting the membership functions of mv-sets $[\![\phi]\!] : S \to B$ for the states that satisfy the property ϕ. In the following, we borrow the norm notation $\|.\|_{\mathcal{M}}$ to define the membership functions of the mv-sets in order to differentiate them from the notation of the actual two-valued sets. Also, we will drop the subscript $\mathcal{M}$ so as to make the text more readable.

Definition 6.4 (mv-CTL Semantics): Let the mv-membership function $\|\phi\| : S \to B$ denote the degree that a state $s \in S$ satisfies a specification ϕ, then the multi-valued semantics of the core operators of mv-CTL with respect to $\mathcal{M}$ are defined as follows:

$$\begin{aligned}
\|a\| &:= \mathbb{O}_a \text{ for } a \in AP \\
\|\neg\phi\| &:= \overline{\|\phi\|} \\
\|\phi \vee \psi\| &:= \|\phi\| \cup_B \|\psi\| \\
\|EX\phi\| &:= pre^{\mathbb{R}}_{\exists}(\|\phi\|) \\
\|EG\phi\| &:= \nu\mathbb{Z}.\|\phi\| \cap_B \|EX\mathbb{Z}\| \\
\|E[\phi\,\mathcal{U}\psi]\| &:= \mu\mathbb{Z}.\|\psi\| \cup_B (\|\phi\| \cap_B \|EX\mathbb{Z}\|)
\end{aligned}$$

where $\|EX\mathbb{Z}\| = pre^{\mathbb{R}}_{\exists}(\mathbb{Z})$.
Next, we present the definitions of the derived operators.

- Conjunction:
$$\|\phi \wedge \psi\| := \overline{\overline{\|\phi\|} \cup_B \overline{\|\psi\|}} = \|\phi\| \cap_B \|\psi\|$$
- The temporal operator next over all paths starting from state s:
$$\|AX\phi\| := pre^{\mathbb{R}}_{\forall}(\|\phi\|) = \overline{\|EX\neg\phi\|}$$
Note that for $s \in S$ the definition $\|AX\phi\|(s) = \bigsqcap_{t\in S}(\mathbb{R}(s,t) \Rightarrow \|\phi\|(t))$ involves a quantification over all states t in the state space S, hence it is possible to run into the problems outlined in Section V-B. The familiar properties of the EX and AX operators are maintained: $\|EX(\phi \vee \psi)\| = \|EX\phi\| \cup_B \|EX\psi\|$ and $\|AX(\phi \wedge \psi)\| = \|AX\phi\| \cap_B \|AX\psi\|$.
- The rest of the temporal operators:
$$\begin{aligned}
\|AF\phi\| &:= \|A[\top\,\mathcal{U}\phi]\| \\
\|EF\phi\| &:= \|E[\top\,\mathcal{U}\phi]\| \\
\|AG\phi\| &:= \overline{\|EF\neg\phi\|} \\
\|A[\phi\,\mathcal{U}\psi]\| &:= \overline{\|E[\neg\psi\,\mathcal{U}\neg\phi \wedge \neg\psi]\|} \cap_B \overline{\|EG\neg\psi\|} \\
\|E[\phi\,\mathcal{B}\psi]\| &:= \overline{\|A[\neg\phi\,\mathcal{U}\psi]\|} \\
\|A[\phi\,\mathcal{B}\psi]\| &:= \overline{\|E[\neg\phi\,\mathcal{U}\psi]\|}
\end{aligned}$$

Algorithm 5 The mv-CTL symbolic model checking algorithm

procedure $Check_{\mathcal{M}}(\phi)$ ▷ $\mathcal{M} = (S, S_0, \mathbb{R}, AP, \mathbb{O}, \mathfrak{B}, D)$ is the mv-Kripke Structure
  Case ϕ
    $a \in AP$ **return** $\|a\|_{\mathcal{M}}$ ▷ $\|a\|_{\mathcal{M}}$ is in symbolic representation
    $\neg\phi_1$ **return** $\sim Check_{\mathcal{M}}(\phi_1)$ ▷ symbolic complementation
    $\phi_1 \vee \phi_2$ **return** $Check_{\mathcal{M}}(\phi_1) \sqcup Check_{\mathcal{M}}(\phi_2)$ ▷ symbolic join operation
    $EX\phi_1$ **return** $CheckEX(Check_{\mathcal{M}}(\phi_1))$ ▷ $CheckEX(\|\phi\|) = pre^{\mathbb{R}}_{\exists}(\|\phi\|)$
    $EG\phi_1$ **return** $FxPoint(\top, f_{Check_{\mathcal{M}}(\phi_1)}(Z))$ ▷ $f_{\|\phi_1\|}(Z) = \|\phi_1\| \sqcap \|EX\, Z\|$
    $E[\phi_1 \mathcal{U} \phi_2]$ **return** $FxPoint(\bot, f_{Check_{\mathcal{M}}(\phi_1), Check_{\mathcal{M}}(\phi_2)}(Z))$ ▷ $f_{\|\phi_1\|,\|\phi_2\|}(Z) = \|\phi_2\| \sqcup (\|\phi_1\| \sqcap \|EX\, Z\|)$
  End Case
end procedure

Finally, note that mv-CTL reduces to classical CTL when the algebra is on $\mathfrak{L}_2$ (see Theorem 4 in [12]). Now, we state a series of important theorems and give a sketch of the proofs for the sake of completeness (all the proofs appear in detail in [12]).

Theorem 6.4: The mv-CTL temporal operators EX and AX are order preserving.

Proof: See Lemma 5.1. ■

Theorem 6.5: The definition of mv-CTL ensures that for all ϕ, $\|\phi\|_{\mathcal{M}}$ forms a partition.

Proof: The proof can be done by structural induction on the formula ϕ. Here, we give only a sketch of the proof. The base case where $\phi = a \in AP$ is valid by definition. The mv-set operations as defined in Section V-A also maintain the partition due to the uniqueness of the supremum and infimum. Furthermore, the temporal operator EX also defines a partition, because the transition relation is a function mapping a pair of states to a unique value in the lattice. Thus, the temporal operators that are defined as fixpoint operators also preserve the partition. ■

Theorem 6.6: The mv-CTL model checking problem is decidable.

Proof: The finiteness of the multi-valued Kripke structure and the monotonicity of the predecessor functions as well as the monotonicity of the join and meet operations guarantee the termination of the fixpoint algorithm (see Section II-C). ■

Theorem 6.7: The fixpoint properties of the derived mv-CTL operators are the same as for the CTL operators:

$$\begin{aligned}
\|AF\phi\| &= \mu\mathbb{Z}.\|\phi\| \cup_B \|AX\, \mathbb{Z}\| \\
\|EF\phi\| &= \mu\mathbb{Z}.\|\phi\| \cup_B \|EX\, \mathbb{Z}\| \\
\|AG\phi\| &= \nu\mathbb{Z}.\|\phi\| \cap_B \|AX\, \mathbb{Z}\| \\
\|A[\phi_1 \mathcal{U} \phi_2]\| &= \mu\mathbb{Z}.\|\phi_2\| \cup_B (\|\phi_1\| \cap_B \|AX\, \mathbb{Z}\|) \\
\|A[\phi_1 \mathcal{B} \phi_2]\| &= \nu\mathbb{Z}.\overline{\|\phi_2\|} \cap_B (\|\phi_1\| \cup_B \|AX\, \mathbb{Z}\|) \\
\|E[\phi_1 \mathcal{B} \phi_2]\| &= \nu\mathbb{Z}.\overline{\|\phi_2\|} \cap_B (\|\phi_1\| \cup_B \|EX\, \mathbb{Z}\|)
\end{aligned}$$

Proof: The cases of AG, AF, EF and AU are proven in [12] using the definition of the EG and EU temporal operators. Here, we will just show how to derive the operator AB from EU.

$$\begin{aligned}
\|A[\phi_1 \mathcal{B} \phi_2]\| &= \overline{\|E[\neg\phi_1 \mathcal{U} \phi_2]\|} \\
&= \overline{\mu Z.\|\phi_2\| \cup_B (\|\neg\phi_1\| \cap_B \|EX\, Z\|)} \\
&= \nu Z.\overline{(\|\phi_2\| \cup_B (\overline{\|\phi_1\|} \cap_B \|EX \sim Z\|))} \\
&= \nu Z.\overline{\|\phi_2\|} \cap_B \overline{(\overline{\|\phi_1\|} \cap_B \overline{\|AX\, Z\|})} \\
&= \nu Z.\overline{\|\phi_2\|} \cap_B (\|\phi_1\| \cup_B \|AX\, Z\|)
\end{aligned}$$

We have already mentioned in Section II-C that the greatest fixpoint operator has the property $\nu x.f(x) = \sim \mu x. \sim f(\sim x)$ over an algebra $\mathfrak{B}$. ■

As one might have expected, the algorithm for mv-CTL symbolic model checking has the same structure as the classical two-valued one (see Algorithm 5). The multi-valued version of the algorithm takes again as input a CTL formula, but now it returns mv-sets (i.e. mappings from states to elements of a quasi-Boolean algebra) encoded

by sets of Binary Decision Diagrams (BDD) or Multi-valued Decision Diagrams (MDD). For an exhaustive discussion on the possible symbolic encodings for the mv-sets and the mv-transition relations see [16].

Example 6.4: Once more, we consider the mv-Kripke structure $\mathcal{M}$ of Example 5.1 (Fig. 9.(c)). We want to compute the value of the specification $EG\,a$. For this reason we use the fixpoint function $\|EG\,a\| = \nu\mathbb{Z}.\|a\| \cap_B pre_\exists^{\mathbb{R}}(\mathbb{Z})$. The algorithm is initialized by setting $(\forall s \in S).(\mathbb{Z}_0(s) = (1,1))$. The sequence of computations appears in Figure 14. Notice that after 2 iterations the fixpoint algorithm has converged. Hence, $\|EG\,a\|_{\mathcal{M}}(s_0) = (0,1)$ and we can conclude that the property holds only in the model that the second expert has provided ($\mathcal{K}_2$).

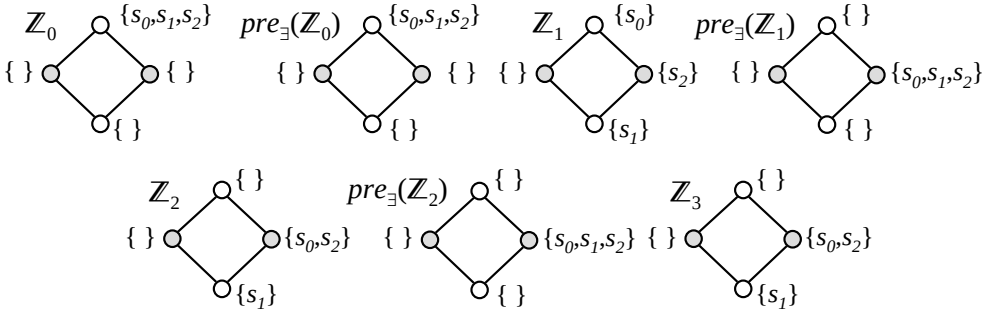

Fig. 14. Computing the fixpoint function $\|EG\,a\| = \nu\mathbb{Z}.\|a\| \cap_B pre_\exists^{\mathbb{R}}(\mathbb{Z})$ on the multi-valued Kripke structure $\mathcal{M}$ from Figure 9.(c).

The running time of the algorithm is again dominated by the fixpoint computations. The computation of the fixpoints converges in $O(|S|)$ iterations. Here, we will give the reasoning behind the computation of the EU temporal operator. For a thorough discussion on the algorithm's running time we refer the reader to [31].

Let us consider the formula $E[\psi\,\mathcal{U}\,\phi]$. Informally, each iteration of the fixpoint operator considers one more step along the paths (or better considers the transitions to the the next states) of the mv-Kripke structure. Hence, after n iterations, we will be looking at the values of ϕ at the states reachable after n transitions. The value of the formula at that point will be the join of the values of ϕ at these states "weighted" by the path that leads to each state. The weight of a path $s_0, s_1, \ldots, s_n$ can be expressed as $\prod_{i=0}^{n}(\mathbb{R}(s_i, s_{i+1}) \sqcap \|\psi\|(s_i))$. On a Kripke structure (which is essentially a directed graph) every state s is reachable by a path of length less then or equal to $|S|$. Every state reachable by a path of length greater then $|S|$ necessarily contains a cycle which can be removed resulting in a path of length less then $|S+1|$. This result in conjunction with the fact that as the length of the path increases, its weight decreases, leads to the conclusion that the fixpoint computation converges in at most $|S|+1$ iterations.

Assume now that the join and meet operations over a lattice $\mathfrak{L}$ take t_L time. The operations of mv-union, mv-intersection and mv-complementation can be done in time linear in the size of the representation of the operants, i.e. the mv-sets, and, hence, their running time is $O(|S| \times t_L)$. The predecessor function involves a join or meet quantification over all possible states and, thus, its running time is $O(|\mathcal{M}| \times t_L)$. So the running time of each iteration is dominated by the computation of the predecessor mv-set. Finally, each mv-CTL formula ϕ can contain at most $|\phi|$ different subformulas. Hence, in the worst case we have at most $|\phi|$ fix-point operators, each taking at most $O(|S|)$ iterations, which in turn take at most $O(|\mathcal{M}| \times t_L)$ steps and we can safely conclude that the running time of the mv-CTL symbolic model checker is $O(|\phi| \times |S| \times |\mathcal{M}| \times t_L)$.

Theorem 6.8 (Running Time of mv-CTL Symbolic Model Checking [31]): There exists an algorithm that can decide the satisfiability of an mv-CTL formula ϕ with respect to a Kripke structure $\mathcal{M}$ in $O(|\phi| \times |S| \times |\mathcal{M}| \times t_L)$ time.

Remark 6.4: In [31], [12], the authors also introduce the notion fairness for the mv-CTL symbolic model checking and in [31], [33] they show how to generate witnesses and counter-examples.

C. Model Checking Multi-Valued μ-Calculus by Reduction

The reduction method that we present here [10], [9] is actually inspired by the previous work of Fitting on multi-valued modal logics [25], [26]. There the author assumes an ordering relation among the experts and that each expert has her own mind about what is possible (her own Kripke model). The problem under investigation is how do the experts assign truth values on a common modal logic and on which sentences do they agree?

As in the previous sections, we will start the presentation of the method by giving first the semantics of the multi-valued μ-calculus. Let ϕ be a μ-calculus formula whose propositions take values on a quasi-Boolean algebra $\mathfrak{B} = (B, \sqcap, \sqcup, \sim, \bot, \top)$ and let us again denote the characteristic function of the mv-set by $\|\phi\|_{\mathcal{M}} : S \to B$, where $\mathcal{M} = (S, S_0, \mathbb{R}, AP^+, \mathbb{O}, \mathfrak{B}, D)$ is the multi-valued model of the system ($AP^+ = AP \cup \overline{AP}$ with $\overline{AP} = \{\neg a \mid a \in AP\}$). Also, the authors let $Dom(\mathbb{R}) = S \times S$ and $N = \{\bot\}$. The environment ε in this case maps states to elements of B. We write $\varepsilon[X \leftarrow \mathbb{S}]$ with $X \in VAR$ and $\mathbb{S} : S \to B$ for the environment that maps $\mathbb{S}$ to X and leaves the rest as is. The state transformers in mv-μ-calculus are mappings from mv-sets (functions) to mv-sets (functions), that is $f_{\mathcal{M},\phi}(\mathbb{Q}) : (S \to B) \to (S \to B)$ with definition $f_{\mathcal{M},\phi}(\mathbb{Q}) := \|\phi(X)\|_{\mathcal{M}}\varepsilon[X \leftarrow \mathbb{Q}]$.

Definition 6.5 (The Semantics of mv-μ-Calculus in NNF): Let $s \in S$, $X \in VAR$ and $a \in AP^+$, then the multi-valued semantics of μ-calculus formulas in NNF with respect to $\mathcal{M}$ are defined as follows:

$$\begin{aligned}
\|a\|_{\mathcal{M}}\varepsilon &:= \mathbb{O}_a \\
\|X\|_{\mathcal{M}}\varepsilon &:= \varepsilon(X) \\
\|\phi \vee \psi\|_{\mathcal{M}}\varepsilon &:= \|\phi\|_{\mathcal{M}}\varepsilon \cup_B \|\psi\|_{\mathcal{M}}\varepsilon \\
\|\phi \wedge \psi\|_{\mathcal{M}}\varepsilon &:= \|\phi\|_{\mathcal{M}}\varepsilon \cap_B \|\psi\|_{\mathcal{M}}\varepsilon \\
\|\Diamond\phi\|_{\mathcal{M}}\varepsilon &:= pre_{\exists}^{\mathbb{R}}(\|\phi\|_{\mathcal{M}}\varepsilon) \\
\|\Box\phi\|_{\mathcal{M}}\varepsilon &:= pre_{\forall}^{\mathbb{R}}(\|\phi\|_{\mathcal{M}}\varepsilon) \\
\|\mu X.\phi(X)\|_{\mathcal{M}}\varepsilon &:= \bigcap\nolimits_B \{\mathbb{Q} \in B^S \mid \|\phi(X)\|_{\mathcal{M}}\varepsilon[X \leftarrow \mathbb{Q}] \subseteq_B \mathbb{Q}\} \\
\|\nu X.\phi(X)\|_{\mathcal{M}}\varepsilon &:= \bigcup\nolimits_B \{\mathbb{Q} \in B^S \mid \mathbb{Q} \subseteq_B \|\phi(X)\|_{\mathcal{M}}\varepsilon[X \leftarrow \mathbb{Q}]\}
\end{aligned}$$

Here, the notation B^S implies the multi-valued equivalent of the powerset of S. In other words, the set of all functions from S to B. We already know the the operations $\sqcup$ and $\sqcap$ and the functions $pre_{\exists}^{\mathbb{R}}$ and $pre_{\forall}^{\mathbb{R}}$ are order-preserving (Lemma A.2 and Lemma 5.1 respectively). The proof of the monotonicity of the fixpoint operators $\|\mu X.\phi(X)\|_{\mathcal{M}}\varepsilon$ and $\|\nu X.\phi(X)\|_{\mathcal{M}}\varepsilon$ is an easy modification of Lemma 3.16 in [57] (we just replace the sets with mv-sets). Using the above lemmas we can prove the monotonicity of any state transformer of the multi-valued μ-calculus (straightforward modification of Theorem 3.17 in [57]). Hence the conclusion is that we can use the Tarski-Knaster theorem to compute the fixpoints of the state transformers.

As we have already mentioned, the mv-μ-calculus model checking problem in this section is defined over quasi-Boolean algebras. Hence, the expressive power of the μ-calculus in normal negation form is the same as the expressive power of the full μ-calculus due to the properties of the negation operator (complement) of the quasi-Boolean algebras. The reason that the authors prefer to use μ-calculus in NNF is to ease the technical aspects of some proofs. Note also that when the formulas are in NNF, then we can relax the constraint of quasi-Boolean algebras and just talk about distributive lattices.

Proposition 6.9: The semantics of mv-μ-calculus in NNF over a distributive lattice $\mathfrak{L}$ collapse to the classical two-valued semantics of μ-calculus in NNF when $\mathfrak{L} = \mathfrak{L}_2$.

Let us assume that the transition relation $\mathbb{R}$ is the membership function of the classical two-valued binary relation $R \subseteq S^2$ (R needs to be total). In order to keep the notation consistent with the last two sections we still assume that $\mathbb{R} : S \to B$, but we impose the restriction that the range of $\mathbb{R}$ is $\{\bot, \top\}$ and hence $(s, s') \in R$ iff $\mathbb{R}(s, s') = \top$. We define a classical Kripke structure $\mathcal{K}_x = (S, S_0, R, AP^+, \mathcal{O}_x)$ which has the same state space S, initial states S_0 and transition relation R as the multi-valued Kripke structure $\mathcal{M}$. Their only difference is with the observation function, which we define for all $a \in AP^+$, for all $s \in S$ and for some $x \in B$ as:

$$a \in \mathcal{O}_x(s) \quad \text{iff} \quad \mathbb{O}_a(s) \sqsupseteq x$$

As we might expect if $x \sqsupseteq x'$, then a formula that holds on $\mathcal{K}_x$ also holds on $\mathcal{K}_{x'}$.

Proposition 6.10: Let $\mathcal{M}$ be an mv-Kripke structure over a finite distributive lattice $\mathfrak{L} = (L, \sqcap, \sqcup)$, ϕ an mv-μ-calculus formula in NNF, $s \in S$ and $x, x' \in L$, then $\|\phi\|_{\mathcal{K}_x}e(s) = 1$ and $x \sqsupseteq x'$ imply $\|\phi\|_{\mathcal{K}_{x'}}e(s) = 1$.

Proof: The proof is done by induction on the alternation depth[7] of the formula ϕ. As in the base case there are no fixpoint operators, we proceed by induction on the structure of the formula ϕ. We do not consider all the

[7]Informally, the alternation depth of a μ-calculus formula ϕ is the maximum number of alternations between nested fixpoint operators μ and ν. Different authors give different definitions of the alternation depth. For a discussion see [57].

cases as several of them have dual formulation. Note that in the base case there are no free variables and that there is no negation case as the μ-calculus formula is in NNF.

- case $\phi = a$, $a \in AP^+$, then $\|a\|_{\mathcal{K}_x} e(s) = 1$ iff $a \in \mathcal{O}_x(s)$ iff $\mathbb{O}_a(s) \sqsupseteq x$ which implies that $\mathbb{O}_a(s) \sqsupseteq x'$ iff $\|a\|_{\mathcal{K}_{x'}} e(s) = 1$.
- case $\phi = \phi_1 \vee \phi_2$, then $\|\phi_1 \vee \phi_2\|_{\mathcal{K}_x} e(s) = 1$ iff ($\|\phi_1\|_{\mathcal{K}_x} e(s) = 1$ or $\|\phi_2\|_{\mathcal{K}_x} e(s) = 1$) which implies by the induction hypothesis that ($\|\phi_1\|_{\mathcal{K}_{x'}} e(s) = 1$ or $\|\phi_2\|_{\mathcal{K}_{x'}} e(s) = 1$) iff $\|\phi_1 \vee \phi_2\|_{\mathcal{K}_{x'}} e(s) = 1$.
- case $\phi = \Diamond\psi$, then $\|\Diamond\psi\|_{\mathcal{K}_x} e(s) = 1$ iff $s \in pre^R_\exists([\![\psi]\!]_{\mathcal{K}_x} e)$. Hence, there exists some $s' \in S$ with $(s, s') \in R$ such that $\|\psi\|_{\mathcal{K}_x} e(s') = 1$ which implies by the induction hypothesis that $\|\psi\|_{\mathcal{K}_{x'}} e(s') = 1$ iff $\|\Diamond\psi\|_{\mathcal{K}_{x'}} e(s) = 1$.

This concludes the base case, i.e. when there are no fixpoint operators. We assume that the hypothesis holds for alternation depth n and we want to prove that it also holds for alternation depth $n + 1$. We proceed again by structural induction on the formula ϕ. We will only consider the case for the fixpoint operator $\mu X.f(X)$ as the proof for the operator $\nu X.f(X)$ is similar. The proofs presented in the base case, i.e. at alternation depth 0, also hold at alternation depth $n + 1$.

- case $\phi = \mu X.\psi(X)$ where ψ is a formula of alternation depth n, then $\|\mu X.\psi(X)\|_{\mathcal{K}_x} e(s) = 1$ iff $s \in f^{|S|+1}_{\mathcal{K}_x,\psi}(\emptyset)$. Similarly, we get $\|\mu X.\psi(X)\|_{\mathcal{K}_{x'}} e(s) = 1$ iff $s \in f^{|S|+1}_{\mathcal{K}_{x'},\psi}(\emptyset)$. As both $f^{|S|+1}_{\mathcal{K}_x,\psi}(\emptyset)$ and $f^{|S|+1}_{\mathcal{K}_{x'},\psi}(\emptyset)$ are of alternation depth less or equal to n the induction hypothesis applies and we get that $s \in f^{|S|+1}_{\mathcal{K}_x,\psi}(\emptyset)$ implies $s \in f^{|S|+1}_{\mathcal{K}_{x'},\psi}(\emptyset)$.

which concludes the proof. ■

The main theorem in [10] for the reduction method for mv-μ-calculus states that the value of an mv-μ-calculus formula in NNF over a distributive lattice $\mathfrak{L}$ can be determined by model checking the structures $\mathcal{K}_x$ with respect to the join-irreducible elements x of $\mathfrak{L}$. Before we state and prove the theorem we need the following lemma.

Lemma 6.11: Let $\mathcal{M}$ be an mv-Kripke structure over a finite distributive lattice $\mathfrak{L}$, ϕ an mv-μ-calculus formula in NNF, $s \in S$ and $x \in \mathcal{J}(\mathfrak{L})$, then $\|\phi\|_{\mathcal{K}_x} e(s) = 1$ iff $\|\phi\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$.

Proof: The proof is done by induction on the alternation depth of the formula ϕ. As in the base case there are no fixpoint operators, we proceed by induction on the structure of the formula ϕ. Note that in the base case there are no free variables and that there is no negation case as the μ-calculus formula is in NNF.

- case $\phi = a$, $a \in AP^+$, then $\|a\|_{\mathcal{K}_x} e(s) = 1$ iff (by definition) $\mathbb{O}_a(s) \sqsupseteq x$ iff $\|a\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$
- case $\phi = \phi_1 \vee \phi_2$, then $\|\phi_1 \vee \phi_2\|_{\mathcal{K}_x} e(s) = 1$ iff ($\|\phi_1\|_{\mathcal{K}_x} e(s) = 1$ or $\|\phi_2\|_{\mathcal{K}_x} e(s) = 1$) iff by induction hypothesis ($\|\phi_1\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$ or $\|\phi_2\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$) iff ($(\Rightarrow)$ by definition of $\sqcup$ and $(\Leftarrow)$ Lemma A.13) $\|\phi_1\|_{\mathcal{M}} \varepsilon(s) \sqcup \|\phi_2\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$ iff $\|\phi_1 \vee \phi_2\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$
- case $\phi = \phi_1 \wedge \phi_2$, then $\|\phi_1 \wedge \phi_2\|_{\mathcal{K}_x} e(s) = 1$ iff ($\|\phi_1\|_{\mathcal{K}_x} e(s) = 1$ and $\|\phi_2\|_{\mathcal{K}_x} e(s) = 1$) iff by induction hypothesis ($\|\phi_1\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$ and $\|\phi_2\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$) iff ($(\Rightarrow)$ Lemma A.2 and $(\Leftarrow)$ by definition of $\sqcap$) $\|\phi_1\|_{\mathcal{M}} \varepsilon(s) \sqcap \|\phi_2\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$ iff $\|\phi_1 \wedge \phi_2\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$
- case $\phi = \Diamond\psi$, then $\|\Diamond\psi\|_{\mathcal{K}_x} e(s) = 1$ iff $s \in pre^R_\exists([\![\psi]\!]_{\mathcal{K}_x} e)$. Hence, there exists some $s' \in S$ with $(s, s') \in R$ such that $\|\psi\|_{\mathcal{K}_x} e(s') = 1$ iff (by induction hypothesis) $\|\psi\|_{\mathcal{M}} \varepsilon(s') \sqsupseteq x$. It is $\mathbb{R}(s, s') = \top$ so $\mathbb{R}(s, s') \sqcap \|\psi\|_{\mathcal{M}} \varepsilon(s') \sqsupseteq \mathbb{R}(s, s') \sqcap x = x$ iff ($(\Rightarrow)$ implies and $(\Leftarrow)$ $\exists s' \in S$ such that) $pre^{\mathbb{R}}_\exists(\|\psi\|_{\mathcal{M}} \varepsilon)(s) \sqsupseteq x$ iff $\|\Diamond\psi\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$.
- case $\phi = \Box\psi$, omitted as it is the dual of the above case.

This concludes the base case, i.e. when there are no fixpoint operators. We assume that the hypothesis holds for alternation depth n and we want to prove that it also holds for alternation depth $n + 1$. We proceed again by structural induction on the formula ϕ. We will only consider the case for the fixpoint operator $\mu X.f(X)$ as the proof for the operator $\nu X.f(X)$ is similar. The proofs presented in the base case, i.e. at alternation depth 0, also hold at alternation depth $n + 1$.

- case $\phi = \mu X.\psi(X)$ where ψ is a formula of alternation depth n, then $\|\mu X.\psi(X)\|_{\mathcal{K}_x} e(s) = 1$ iff $s \in f^{|S|+1}_{\mathcal{K}_x,\psi}(\emptyset)$. Similarly, we get $\|\mu X.\psi(X)\|_{\mathcal{M}} \varepsilon(s) \sqsupseteq x$ iff $f^{|S|+1}_{\mathcal{M},\psi}(\bot)(s) \sqsupseteq x$. As both $f^{|S|+1}_{\mathcal{K}_x,\psi}(\emptyset)$ and $f^{|S|+1}_{\mathcal{M},\psi}(\bot)$ are of alternation depth less or equal to n the induction hypothesis applies and we get that $s \in f^{|S|+1}_{\mathcal{K}_x,\psi}(\emptyset)$ iff $f^{|S|+1}_{\mathcal{M},\psi}(\bot)(s) \sqsupseteq x$.

which concludes the proof. ■

Algorithm 6 Reduction algorithm for the mv-μ-calculus

```
1: procedure REDUCEMUCALC(M, φ)
2:     A ← J(L)
3:     B ← ∅
4:     while A ≠ ∅ do
5:         x ← maximal element of A
6:         if s ∈ [[φ]]_{K_x} e then
7:             C ← {x' ∈ J(L) | x' ⊑ x}
8:             B ← B ∪ C
9:             A ← A\C
10:        else
11:            A ← A\{x}
12:        end if
13:    end while
14:    return B
15: end procedure
```

Theorem 6.12: Let $\mathcal{M}$ be a mv-Kripke structure over a finite distributive lattice $\mathfrak{L}$, ϕ an mv-μ-calculus formula in NNF and $s \in S$, then $\|\phi\|_{\mathcal{M}}\varepsilon(s) = \bigsqcup\{x \in \mathcal{J}(\mathfrak{L}) \mid \|\phi\|_{\mathcal{K}_x} e(s) = 1\}$.

Proof: For $s \in S$, it is $\|\phi\|_{\mathcal{M}}\varepsilon(s) \in L$. Hence, by Lemma A.12 we get that $\|\phi\|_{\mathcal{M}}\varepsilon(s) = \bigsqcup\{x \in \mathcal{J}(\mathfrak{L}) \mid x \sqsubseteq \|\phi\|_{\mathcal{M}}\varepsilon(s)\}$ and, hence, using Lemma 6.11 we derive the result $\|\phi\|_{\mathcal{M}}\varepsilon(s) = \bigsqcup\{x \in \mathcal{J}(\mathfrak{L}) \mid \|\phi\|_{\mathcal{K}_x} e(s) = 1\}$. ■

The algorithm that implements the above reduction procedure is presented in Algorithm 6. Let $x_1, x_2 \in L$ and $x_1 \sqsubseteq x_2$, then due to Proposition 6.10 if $\mathcal{K}_{x_1}$ satisfies the formula ϕ we do not have to check whether $s \in [\![\phi]\!]_{\mathcal{K}_{x_2}}$. The running time of the algorithm depends on the running time of the classical two-valued μ-calculus model checker. The classical model checker is called at most $|\mathcal{J}(\mathfrak{L})|$ times. It might seem that the worst running time occurs when the lattice $\mathfrak{L}$ is a finite linear order, but even in that case we can optimize by performing binary search. That is, we first model check the element in the middle of the lattice and then we recurse on the upper or lower half according to the result. In this case, the algorithm will call the classical model checker $O(log(|\mathcal{J}(\mathfrak{L})|))$ times.

Example 6.5: Assume that we have the quasi-Boolean algebra $\mathfrak{B}_3$ (see Fig. 1), an mv-Kripke structure $\mathcal{M}$ and an mv-μ-calculus formula ϕ. The set of the join-irreducible elements in this case is $\mathcal{J}(\mathfrak{B}_3) = \{1/2, 1\}$. The Kripke structure $\mathcal{K}_1$ expresses the pessimistic viewpoint that $1/2$ is false, while $\mathcal{K}_{1/2}$ expresses the optimistic viewpoint that both the values 1 and $1/2$ are true. If $\mathcal{K}_1$ satisfies the formula ϕ at state $s \in S$, then $\|\phi\|_{\mathcal{M}}(s) = \bigsqcup\{1/2, 1\} = 1$. If on the other hand $\mathcal{K}_{1/2}$ satisfies the formula ϕ at state $s \in S$, then $\|\phi\|_{\mathcal{M}}(s) = \bigsqcup\{1/2\} = 1/2$, otherwise the result is 0.

In the above reduction procedure, we have assumed that the range of the transition relation is binary. The authors in [10] also present a two-step reduction method for the case where the transition relation is multi-valued. Here we just give the reduction method without the proofs due to space limitations. As before, for each join irreducible element x of the lattice $\mathfrak{L}$ we create the Kripke structure $\mathcal{K}_x = (S, S_0, R, AP^+, \mathcal{O}_x)$, but now we define an extended structure $\mathcal{K}_x^e = (S, S_0, R_x^+, R_x^-, AP^+, \mathcal{O}_x)$ with two new transition relations:

$$(s, s') \in R_x^+ \quad \text{iff} \quad \mathbb{R}(s, s') \sqsupseteq x$$
$$(s, s') \in R_x^- \quad \text{iff} \quad \neg(\sim \mathbb{R}(s, s') \sqsupseteq x)$$

The semantics of a μ-calculus formula over such a structure are the same as in the classical case (see Section III-E) with the exception that the modal operators $\Box$ and $\Diamond$ need to be modified:

$$[\![\Diamond\phi]\!]_{\mathcal{K}_x^e} e := \{s \in S \mid (\exists s').((s, s') \in R_x^+ \wedge s' \in [\![\phi]\!]_{\mathcal{K}_x^e} e)\}$$
$$[\![\Box\phi]\!]_{\mathcal{K}_x^e} e := \{s \in S \mid (\forall s').((s, s') \in R_x^- \rightarrow s' \in [\![\phi]\!]_{\mathcal{K}_x^e} e)\}$$

Lemma 6.13: Let $\mathcal{M} = (S, S_0, \mathbb{R}, AP^+, \mathbb{O}, \mathfrak{L}, D)$ be an mv-Kripke structure, ϕ be a μ-calculus formula, $s \in S$ and $x \in \mathcal{J}(\mathfrak{L})$, then $(\mathcal{K}_x^e, s) \models \phi$ iff $\|\phi\|_{\mathcal{M}}\varepsilon(s) \sqsupseteq x$.

The second step of the reduction proceeds by constructing a classical Kripke structure $\mathcal{K}'_x = (S', S'_0, R'_x, AP^+_e, \mathcal{O}'_x)$ as follows:

$$\begin{array}{rcl} AP^+_e & := & AP^+ \cup \{\xi\} \\ S' & := & \{(s,\star) \mid s \in S, \star \in \{+,-\}\} \\ S'_0 & := & \{(s,+) \mid s \in S_0\} \\ a \in \mathcal{O}'_x(s,\star) & \text{iff} & (a \in \mathcal{O}_x(s)) \vee (a = \xi \wedge \star = +) \\ ((s,\star),(s',\star')) \in R'_x & \text{iff} & (s,s') \in R^{\star'}_x \end{array}$$

The states $(s,+)$ and $(s,-)$ are strongly bisimilar, hence $(s,+)$ satisfies ϕ iff $(s,-)$ does. The authors also define a mapping Γ from μ-calculus formulas (in NNF) to μ-calculus formulas as follows:

$$\begin{array}{rcll} \Gamma(a) & := & a & \text{with } a \in AP^+ \text{ or } a \in \text{VAR} \\ \Gamma(\phi_1 \oplus \phi_2) & := & \Gamma(\phi_1) \oplus \Gamma(\phi_2) & \text{with } \oplus \in \{\vee, \wedge\} \\ \Gamma(\Diamond\phi) & := & \Diamond(\xi \wedge \Gamma(\phi)) & \\ \Gamma(\Box\phi) & := & \Box(\xi \vee \Gamma(\phi)) & \\ \Gamma(\sigma X.\phi(X)) & := & \sigma X.\Gamma(\phi(X)) & \text{with } \sigma \in \{\nu, \mu\} \end{array}$$

Now we can state the following proposition.

Proposition 6.14: Let $\mathcal{K}^e_x = (S, S_0, R^+_x, R^-_x, AP^+, \mathcal{O}_x)$, $s \in S$, $x \in \mathcal{J}(\mathfrak{L})$ and ϕ be a μ-calculus formula, then $(\mathcal{K}^e_x, s) \models \phi$ iff $(\mathcal{K}'_x, (s,+)) \models \Gamma(\phi)$.

In the following example, we apply the reduction method of this section to the multi-valued Kripke structure $\mathcal{M}$ of example 5.1. The reader is encouraged to compare the results of this approach with the Examples 6.3 and 6.4.

Example 6.6: As before, we are given an mv-Kripke structure $\mathcal{M}$ (see Figure 9.c) and we want to model check the property $\phi = \nu X.a \wedge \Diamond X$ which is equivalent to $EG\,a$. The lattice $\mathfrak{L}_{2,2}$ has two join irreducible elements: $\mathcal{J}(\mathfrak{L}) = \{(1,0),(0,1)\}$. Hence, using the above reduction method we construct two classical Kripke structures $\mathcal{K}'_{(1,0)}$ and $\mathcal{K}'_{(0,1)}$ which are presented in Figure 15. The translated formula is $\Gamma(\phi) = \nu X.a \wedge \Diamond(\xi \wedge X)$. Let us consider the first model checking problem on the structure $\mathcal{K}'_{(1,0)}$. It is easy to see that $[\![a]\!]_{\mathcal{K}'_{(1,0)}} e = \{(s_0,+),(s_0,-)\}$ and that $[\![\xi]\!]_{\mathcal{K}'_{(1,0)}} e = \{(s_0,+),(s_1,+),(s_2,+)\}$. By initializing the fixpoint algorithm with $X_0 = S'$, we get the following computations:

- **1st iteration:** $[\![\xi]\!]_{\mathcal{K}'_{(1,0)}} e \cap X_0 = [\![\xi]\!]_{\mathcal{K}'_{(1,0)}} e$, $pre_\exists^{R'_{(1,0)}}([\![\xi]\!]_{\mathcal{K}'_{(1,0)}} e) = S'$, $X_1 = [\![a]\!]_{\mathcal{K}'_{(1,0)}} e \cap S' = [\![a]\!]_{\mathcal{K}'_{(1,0)}} e$
- **2nd iteration:** $[\![\xi]\!]_{\mathcal{K}'_{(1,0)}} e \cap X_1 = \{(s_0,+)\}$, $pre_\exists^{R'_{(1,0)}}(\{(s_0,+)\}) = \emptyset$, $X_2 = \emptyset$

Hence, we can safely conclude that $(\mathcal{K}'_{(1,0)}, (s_0,+)) \not\models \Gamma(\phi)$. Similarly, if we model check the second structure we get that $(\mathcal{K}'_{(0,1)}, (s_0,+)) \models \Gamma(\phi)$ and, therefore, $\|\phi\|_{\mathcal{M}} \varepsilon(s_0) = (0,1)$.

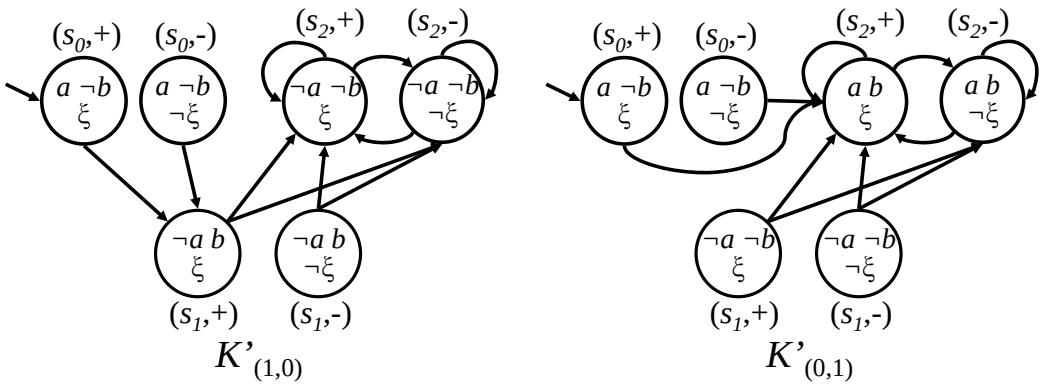

Fig. 15. The two Kripke structures resulting from the reduction procedure - one for each join irreducible element of the lattice.

D. Applications of Multi-Valued Model Checking

We have already mentioned in the introduction that there exist three main application areas for multi-valued model checking. The first is to check the degree of agreement of inconsistent specifications. Toy examples of such an application were presented in Sections V and VI of this document. As these examples do not capture real-life applications, the reader is referred to [14] and [22] for a more detailed discussion. The second application area is the temporal logic quarry checking ([34], [8], [11], [15]), which we do not discuss here as such an application is out of the scope of this introductory paper. The application that we do present here is model checking partial state spaces, which may result from abstraction methods or due to incomplete information about the model [6], [7].

Assume that we have a Kripke structure $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ and that the size of the state space S is so large (or even infinite) that the model checking problem becomes time consuming or even intractable. In order to remedy the state explosion problem, many authors have developed various abstraction techniques [18], [46] so as to reduce the size of the state space of the model. The resulting abstract structure can be efficiently model checked and the outcome of the verification procedure may be also valid in the original model under certain conditions. The problem that may arise is that the abstraction methods usually work for some very restricted fragments of temporal logics and can potentially generate spurious counter-examples. The multi-valued model checking can provide an extension to the abstraction techniques which allows the verification of specifications in the full version of a temporal logic and alleviates the problem of spurious counter-examples.

Before we explain in detail how multi-valued model checking can be useful in these problems, we need to present some machinery. We define a *partial Kripke structure* to be an mv-Kripke structure over the algebra $\mathfrak{B}_3$ where some atomic propositions take the truth value "unknown" when there is no information available about them in certain states of the system. For the following, we consider the quasi-Boolean algebra $\mathfrak{B}_3 = (\{0, 1/2, 1\}, \leq, \sim)$ with the negation operation defined as $\sim 1 = 0$, $\sim 0 = 1$ and $\sim 1/2 = 1/2$. Here, the value $1/2$ is interpreted as "unknown whether true or false". Also, we let $D = \{1, 1/2\}$ and we drop $\mathfrak{B}_3$ and D from the mv-Kripke structure notation for clarity of the presentation. In addition, we set the range of $\mathbb{R}$ to be $\{0, 1\}$, i.e. there are no multi-valued transitions. Let us consider the poset $\mathfrak{S}_3 = (\{0, 1/2, 1\}, \sqsubseteq)$ with the ordering relation $\sqsubseteq$ defined as: $1/2 \sqsubseteq 1$, $1/2 \sqsubseteq 0$, $0 \parallel 1$ and $x \sqsubseteq x$ for any $x \in \{0, 1/2, 1\}$. It is easy to see that the poset $\mathfrak{S}_3$ can be used to define a preorder on the partial Kripke structures according to the information that they contain. The following definition provides us with a bisimulation relation that formalises this ordering relation between partial Kripke structures.

Definition 6.6 (Completeness Preorder): Let $\mathcal{M}_1 = (S_1, S_{01}, \mathbb{R}_1, AP, \mathbb{O}_1)$ and $\mathcal{M}_2 = (S_2, S_{02}, \mathbb{R}_2, AP, \mathbb{O}_2)$ be partial Kripke structures, then the *completeness preorder* is the greatest relation $\preceq \subseteq S_1 \times S_2$ such that for $s_1 \in S_1$ and $s_2 \in S_2$, $s_1 \preceq s_2$ implies that:

- $(\forall a \in AP).(\mathbb{O}_1(s_1, a) \sqsubseteq \mathbb{O}_2(s_2, a))$
- If $\mathbb{R}_1(s_1, s_1') = 1$, then there is some $s_2' \in S_2$ such that $\mathbb{R}_2(s_2, s_2') = 1$ and $s_1' \preceq s_2'$
- If $\mathbb{R}_2(s_2, s_2') = 1$, then there is some $s_1' \in S_1$ such that $\mathbb{R}_1(s_1, s_1') = 1$ and $s_1' \preceq s_2'$

The reader who is familiar with the definition of the bisimulation relation (otherwise see [57]) may notice that according to the above definition the states s_1 and s_2 are bisimilar and, furthermore, the state s_2 contains more information then state s_1. The following theorem provides us with the tools to order the Kripke structures according to how many of their properties are completely defined, i.e. they are either "true" or "false".

Theorem 6.15 (Adapted from [6], [7]): Let $\mathcal{M}_1 = (S_1, S_{01}, \mathbb{R}_1, AP, \mathbb{O}_1)$ and $\mathcal{M}_2 = (S_2, S_{02}, \mathbb{R}_2, AP, \mathbb{O}_2)$ be partial Kripke structures, $s_1 \in S_1$, $s_2 \in S_2$ and Φ_μ be the set of all well formed mv-μ-calculus formulas taking values over $\mathfrak{B}_3$, then:

$$s_1 \preceq s_2 \text{ iff } (\forall \phi \in \Phi_\mu).(\|\phi\|_{\mathcal{M}_1}\varepsilon(s_1) \sqsubseteq \|\phi\|_{\mathcal{M}_2}\varepsilon(s_2))$$

Proof: As the proof of this theorem is long, we only give some pointers. First of all, note that the negation operation $\sim$, the meet $\sqcap$ and the join $\sqcup$ as defined for the quasi-Boolean algebra $\mathfrak{B}_3$ above are all monotonic with respect to the relation $\sqsubseteq$. The $(\Rightarrow)$ direction of the proof is similar to the ones presented in Section VI-C. The other direction is more involved and consists of the construction of a μ-calculus formula ϕ such that if $s_1 \not\preceq s_2$ then $\|\phi\|_{\mathcal{M}_1}\varepsilon(s_1) \sqsupset \|\phi\|_{\mathcal{M}_2}\varepsilon(s_2))$. ■

When an mv-μ-calculus formula ϕ evaluates to "true" (or "false") over a partial Kripke structure, then the same formula also evaluates to "true" (or "false") on every more complete partial Kripke structure. On the other hand, when the formula evaluates to "unknown", then we must check the formula on a more complete Kripke structure. This implies an iterative procedure that would terminate when the (finite) Kripke structure has complete information over all its states. Note though that a formula ϕ evaluates to $1/2$ even if all the more complete models evaluate to either "true" or "false". This is an undesirable situation as it would be more meaningful to have ϕ evaluate to $1/2$ if and only if there exists a more complete partial Kripke structure where ϕ evaluates to "true" and another one where ϕ evaluates to "false". For example, consider the case where $\phi = (a \vee \neg a) \wedge b$ with $\mathbb{O}(s, a) = 1/2$ and $\mathbb{O}(s, b) = 1$, then $\|\phi\|_{\varepsilon}(s) = 1/2$ even though we know that ϕ should evaluate to true (as it does in all the more complete partial Kripke structures where a is defined). This problem is discussed in [7] where the authors propose the framework of *generalised model checking* which informally can be described as a combination of satisfiability checking and model checking.

Let us now describe the procedure for model checking partially explored state spaces. We create the following partial Kripke structure $\mathcal{M} = (S', S'_0, R', AP, \mathbb{O})$ from the complete Kripke structure $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$. Let $S_E \subseteq S$ be the state space of $\mathcal{K}$ that has been explored (or that we are interested in) and, similarly, $R_E \subseteq R$ denotes the set of explored transitions. All the unexplored states $S \backslash S_E$ are mapped to a dummy state $s_{1/2}$ and their atomic propositions evaluate to $1/2$. In the same fashion, the outgoing transitions from S_E that go to some state in $S \backslash S_E$ are now pointing to $s_{1/2}$. Also, a self-loop is added to the new state $s_{1/2}$ in order to keep the transition relation total. In summary,

$$
\begin{array}{lcl}
S' & = & S_E \cup \{s_{1/2}\} \\
\mathbb{O}_a(s) & = & \begin{cases} a \in \mathcal{O}(s) & \text{if } s \in S_E \\ 1/2 & \text{otherwise} \end{cases} \\
R' & = & R_E \cup \{(s, s_{1/2}) \mid s \in S_E \wedge (\exists t \in S \backslash S_E).((s,t) \in R \backslash R_E)\} \cup \{(s_{1/2}, s_{1/2})\}
\end{array}
$$

If there exists some state $s'_0 \in S'_0$ such that $s'_0 = s_0 \in S_0$, then the proof of the following theorem is straightforward.

Theorem 6.16 (Adapted from [6], [7]): Consider the complete Kripke structure $\mathcal{K} = (S, S_0, R, AP, \mathcal{O})$ and let $\mathcal{M} = (S', S'_0, R', AP, \mathbb{O})$ be a partial Kripke structure derived from $\mathcal{K}$. Then

$$s'_0 \preceq s_0$$

Using Theorems 6.15 and 6.16, it is easy to see that when a 3-valued μ-calculus formula ϕ evaluates to "true" (or "false") over the partial Kripke structure $\mathcal{M}$, then it also evaluates to "true" (or "false") over the complete Kripke structure $\mathcal{K}$. On the other hand, if ϕ evaluates to $1/2$, then further exploration of the state space is required. Note that the results presented here also hold for CTL and CTL^* formulas. Hence, any of the multi-valued model checking methods presented in the previous sections can be employed for the model checking of partial state spaces.

VII. Conclusions, Discussion and Future Directions

In this paper, we have given an introduction to the multi-valued model checking problem. The somewhat extended introduction to the lattice theory was deemed necessary in order to present the multi-valued temporal logics and model checking, as well as the fix-point computations in μ-calculus and CTL symbolic model checking. Furthermore, presenting several definitions and lemmas concerning lattice theory was mandatory for the proofs of the theorems that followed. An extensive introduction to the classical two-valued temporal logics and the μ-calculus was given as a bridge to the multi-valued cases. We believe that the notion of temporal logics is hard to grasp for the non-insider, let alone their multi-valued version, where the truth values range over some lattice. Finally, the presentation of the classical model checking algorithms was also regarded as necessary for mainly two reasons. First, because some of the multi-valued model checkers presented here reduce to them and, thus, the reader should be able to refer to their details (i.e. algorithm, complexity issues, etc). Second, because the mv-CTL model checking is very similar to the classical CTL model checking.

The value and importance of the model checking algorithms is indisputable by now. Their applications on fail safe systems has been a tremendous assistance to engineers and system designers. The multi-valued model

checking comes as an extension to the classical one and provides us with tools that have additional capabilities. Within the last 6 years, this additional power has already been put to work on applications such as query temporal checking, model checking inconsistent specifications and in abstraction techniques. The author firmly believes that such techniques can also have potential applications in the field of robotics, where uncertainty and inconsistent view-points among robots are part of the real-world implementations.

Of course, many skeptics can claim that multi-valued model checking does not offer us anything more then the classical two-valued model checking. The same views were expressed about multi-valued logics and, furthermore, the fact that multi-valued model checking can be reduced to the classical one offers grounds for such a criticism. One more issue that may constrain the popularity of multi-valued model checking is that the user must be well versed in many-valued logics, otherwise he or she might not be able to express the required properties or even worse he or she might get wrong results. On the other hand, even if multi-valued model checking can just offer one higher level of expressive power then the two-valued version, this might open avenues for new discoveries and applications.

The authors of the presented papers believe that future work should move towards the following directions. Along the lines of mv-CTL*model checking using designated values, the next step is to develop proof systems for logics taking values over c-complete lattices. As the research in the direction of mv-CTL symbolic model checking is more thorough, the authors suggest several extensions. Some of them are the following (see [31]): extending the fairness constraints over one multi-valued model to applications that involve several models, exploring the interaction between heuristics for selecting the best witness and property patterns (the latter can be used to help users express properties of interest without an extensive knowledge of temporal logics) and, finally, extending the mv-CTL model checker in order to handle mv-LTL formulas.

ACKNOWLEDGEMENTS

This report was compiled for the requirements of the Written Preliminary Examination II. I would like to thank the WPE II committee, professors Rajeev Alur, Insup Lee and George J. Pappas, for their comments. I am grateful to professor Jean Gallier for his continuous availability to answer any possible question. Also, I would like to thank Dr. Patrice Godefroid for the private communication. Many thanks to Hadas Kress-Gazit for helping me improve the presentation of this paper and to Dimitris Vytiniotis for (patiently) listening to my mathematical inquiries.

REFERENCES

[1] *34th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2004), 19-22 May 2004, Toronto, Canada*. IEEE Computer Society, 2004.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3–34, 1995.

[3] R. Alur, "Timed automata," in *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 1999, pp. 8–22.

[4] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen, *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[5] G. Brajnik and D. J. Clancy, "Focusing qualitative simulation using temporal logic: Theoretical foundations," *Annals of Mathematics and Artificial Intelligence*, vol. 22, no. 1-2, pp. 59–86, 1998.

[6] G. Bruns and P. Godefroid, "Model checking partial state spaces with 3-valued temporal logics," in *Proceedings of the 11th International Conference on Computer Aided Verification (CAV)*. London, UK: Springer-Verlag, 1999, pp. 274–287.

[7] ——, "Generalized model checking: Reasoning about partial state spaces," in *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR)*. London, UK: Springer-Verlag, 2000, pp. 168–182.

[8] ——, "Temporal logic query checking," in *LICS '01: Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*. Washington, DC, USA: IEEE Computer Society, 2001, p. 409.

[9] ——, "Model checking with multi-valued logics." Bell Labs, Lucent Technologies, Tech. Rep. ITD-03-44535H, May 2003.

[10] ——, "Model checking with multi-valued logics." in *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, ser. Lecture Notes in Computer Science, vol. 3142. Springer-Verlag, 2004, pp. 281–293.

[11] W. Chan, "Temporal-logic queries," in *Proceedings of the 12th International Conference on Computer Aided Verification (CAV)*, vol. 1855. London, UK: Springer-Verlag, 2000, pp. 450–463.

[12] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel, "Multi-valued symbolic model-checking," *ACM Trans. Softw. Eng. Methodol.*, vol. 12, no. 4, pp. 1–38, Oct. 2004.

[13] M. Chechik, B. Devereux, and A. Gurfinkel, "Model-checking infinite state-space systems with fine-grained abstractions using spin," in *Proceedings of the 8th international SPIN workshop on Model checking of software (SPIN)*. New York, NY, USA: Springer-Verlag New York, Inc., 2001, pp. 16–36.

[14] M. Chechik and S. Easterbrook, "Reasoning about compositions of concerns," in *Proceedings of Workshop on Advanced Separation of Concerns in Software Engineering, at the 23rd International Conference on Software Engineering (ICSE-01)*, May 2001.

[15] M. Chechik and A. Gurfinkel, "Tlqsolver: A temporal logic query checker." in *Proceedings of the 15th International Conference on Computer Aided Verification (CAV)*, vol. 2725. Springer-Verlag, 2003, pp. 210–214.

[16] M. Chechik, A. Gurfinkel, B. Devereux, A. Lai, and S. Easterbrook, "Data structures for symbolic multi-valued model-checking," Department of Computer Science, University of Toronto, Tech. Rep. CSRG, Jan. 2002.

[17] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *Proceedings of International Conference on Computer-Aided Verification*, July 2002.

[18] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 5, pp. 1512–1542, September 1994.

[19] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: MIT Press, 1999.

[20] E. Clarke and H. Schlingloff, "Model checking," in *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Elsevier Science, 2000, ch. 21, pp. 1367–1522.

[21] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, 2nd ed. Cambridge, United Kingdom: Cambridge University Press, 2002.

[22] S. Easterbrook and M. Chechik, "A framework for multi-valued reasoning over inconsistent viewpoints," in *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, 2001, pp. 411–420.

[23] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science: Formal Models and Semantics*, J. van Leeuwen, Ed., vol. B. North-Holland Pub. Co./MIT Press, 1990, pp. 995–1072.

[24] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005.

[25] M. C. Fitting, "Many-valued modal logics," *Fundam. Inf.*, vol. 15, no. 3-4, pp. 235–254, 1991.

[26] ——, "Many-valued modal logics ii," *Fundam. Inf.*, vol. 17, no. 1-2, pp. 55–73, 1992.

[27] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Proceedings of the 13th Conference on Computer Aided Verification (CAV'01)*, ser. Lecture Notes in Computer Science, G. Berry, H. Comon, and A. Finkel, Eds., no. 2102. Springer Verlag, 2001, pp. 53–65.

[28] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*. Chapman & Hall, Ltd., 1996, pp. 3–18.

[29] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning*. Elsevier, 2004.

[30] S. Gottwald, *A Treatise on Many-Valued Logics*, ser. Studies in Logic and Computation, D. M. Gabbay, Ed. Baldock, Hertfordshire, England: Research Studies Press LTD., 2001.

[31] A. Gurfinkel, "Multi-valued symbolic model-checking: Fairness, counter-examples, running time," Master's thesis, Department of Computer Science, University of Toronto, October 2002.

[32] A. Gurfinkel and M. Chechik, "Multi-valued model checking via classical model checking," in *Proceedings of CONCUR 2003, LNCS 2761*, R. Amadio and D. Lugiez, Eds. Springer-Verlag Berlin Heidelberg, 2002, pp. 266–280.

[33] ——, "Proof-like counter-examples." in *TACAS*, 2003, pp. 160–175.

[34] A. Gurfinkel, B. Devereux, and M. Chechik, "Model exploration with temporal logic query checking," *SIGSOFT Softw. Eng. Notes*, vol. 27, no. 6, pp. 139–148, 2002.

[35] K. Havelund, M. Lowry, and J. Penix, "Formal analysis of a space-craft controller using spin," *IEEE Trans. Software Eng.*, vol. 27, pp. 749 – 765, 2001.

[36] G. Holzmann, *The Spin Model Checker, Primer and Reference Manual*. Reading, Massachusetts: Addison-Wesley, 2004.

[37] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 2nd ed. Reading, Mass.: Addison-Wesley, 2001.

[38] M. Huth and S. Pradhan, "Consistent partial model checking," *Electronic Notes in Theoretical Computer Science*, vol. 73, 2003.

[39] B. Konikowska and W. Penczek, "Reducing model checking from multi-valued ctl* to ctl*," in *Proceedings of CONCUR 2002, LNCS 2421*, 2002.

[40] ——, "Model checking for multi-valued ctl*," in *Beyond two: theory and applications of multiple-valued logic*. Heidelberg, Germany: Physica-Verlag GmbH, 2003, pp. 193–210.

[41] ——, "Model checking multi-valued modal μ-calculus: Revisited," in *Proceedings of Concurrency, Specification and Programming - CS&P 2004*, 2004.

[42] ——, "On designated values in multi-valued ctl model checking," *Fundamenta Informaticae*, vol. 57, pp. 1–14, 2004.

[43] D. Kozen, "Results on the propositional μ-calculus," in *Proceedings of the 9th Colloquium on Automata, Languages and Programming*. London, UK: Springer-Verlag, 1982, pp. 348–359.

[44] O. Kupferman, M. Y. Vardi, and P. Wolper, "An automata-theoretic approach to branching-time model checking," *J. ACM*, vol. 47, no. 2, pp. 312–360, 2000.

[45] T. Laureys, "From event-based semantics to linear temporal logic: The logical and computational aspects of a natural language interface for hardware verification," Master's thesis, University of Edinburgh, Nov. 1999.

[46] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem, "Property preserving abstractions for the verification of concurrent systems," *Formal Methods in System Design*, vol. 6, no. 1, pp. 11–44, 1995.

[47] R. R. Lutz, "Targeting safety-related errors during software requirements analysis," in *SIGSOFT '93: Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering*. New York, NY, USA: ACM Press, 1993, pp. 99–106.

[48] M. Maidl, "The common fragment of CTL and LTL," in *Proc. 41th Annual Symposium on Foundations of Computer Science*, 2000, pp. 643–652.

[49] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals." in *FORMATS/FTRTFT*, 2004, pp. 152–166.

[50] G. Malinowski, *Many-Valued Logics*, ser. Oxford Logic Guides. Oxford Science Publications, 1993.
[51] K. L. McMillan, "Symbolic model checking: an approach to the state explosion problem," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.
[52] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Menlo Park, CA: Addison Wesley Longman, 1998. [Online]. Available: http://www.ece.osu.edu/ passino/FCbook.pdf
[53] P. Pingree, E. Mikk, G. Holzmann, M. Smith, and D. Dams, "Validation of mission critical software design and implementation using model checking," in *Proceedings of the 21st Digital Avionics Systems Conference*, vol. 1, October 2002, pp. 6A4–1 – 6A4–12.
[54] A. Pnueli, "The temporal logic of programs," in *Proceedings of the 18th IEEE Symposium Foundations of Computer Science*, 1977, pp. 46–57.
[55] H. Rasiowa, *An Algebraic Approach to Non-Classical Logics*. Amsterdam: North-Holland, 1974.
[56] A. Saffiotti, K. Konolige, and E. H. Ruspini, "A multivalued-logic approach to integrating planning and control," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 481–526, 1995.
[57] K. Schneider, *Verification of Reactive Systems – Formal Methods and Algorithms*, ser. Texts in Theoretical Computer Science (EATCS Series). Springer, 2004.
[58] G. Seel, *Ammonius and the Seabattle: Texts, Commentary, and Essays*. Berlin, Germany: Walter de Gruyter GmbH & Co., 2001.
[59] B. Shults and B. Kuipers, "Proving properties of continuous systems: Qualitative simulation and temporal logic." *Artif. Intell.*, vol. 92, no. 1-2, pp. 91–129, 1997.
[60] F. Somenzi, *Binary Decision Diagrams*, ser. Calculational System Design, volume 173 of NATO Science Series F: Computer and Systems Sciences. IOS Press, 1999, pp. 303–366.
[61] P. Tabuada and G. J. Pappas, "From discrete specifications to hybrid control," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, December 2003.
[62] A. Tarski, "A lattice-theoretical fixpoint theorem and its applications," *Pacific Journal of Mathematics*, vol. 5, pp. 285–309, 1955.
[63] W. Thomas, "Automata on infinite objects," in *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*, J. van Leeuwen, Ed. Amsterdam: Elsevier, 1990, pp. 133–191.
[64] P. Wolper, "Constructing automata from temporal logic formulas: a tutorial," in *Lectures on formal methods and performance analysis: first EEF/Euro summer school on trends in computer science*. Springer-Verlag New York, Inc., 2002, pp. 261–277.

APPENDIX

A. Elements of Ordering Relations and Lattices

Lemma A.1 (Connecting ordering relation and down-sets): Let $(S, \sqsubseteq)$ be an ordered set and $x, y \in S$, then the following are equivalent:

1) $x \sqsubseteq y$
2) $x^{\downarrow} \subseteq y^{\downarrow}$

Proof: Straightforward by the definitions. ■

Lemma A.2 (Monotonicity of Join and Meet): Let $(L, \sqsubseteq)$ be a lattice and $x, y, z, w \in L$, then:

1) $x \sqsubseteq y$ implies $x \sqcup z \sqsubseteq y \sqcup z$ and $x \sqcap z \sqsubseteq y \sqcap z$.
2) $x \sqsubseteq y$ and $z \sqsubseteq w$ imply $x \sqcup z \sqsubseteq y \sqcup w$ and $x \sqcap z \sqsubseteq y \sqcap w$.

Proof: We will only prove the second claim. We know that $\{x, z\}^u = \{s \in L \mid (x \sqsubseteq s) \wedge (z \sqsubseteq s)\} \supseteq \{s \in L \mid (y \sqsubseteq s) \wedge (w \sqsubseteq s)\} = \{y, w\}^u$. By definition of $y \sqcup w$ we know that $y \sqcup w \in \{y, w\}^u$ and that for all $s \in \{y, w\}^u$ it is $y \sqcup w \sqsubseteq s$. Hence, we also have $y \sqcup w \in \{x, z\}^u$ and $x \sqcup z \sqsubseteq y \sqcup w$. The dual case is proven in the same way. ■

Lemma A.3 (The Connecting Lemma): Let $\mathfrak{L}$ be a lattice and $x, y \in L$, then

$$x \sqsubseteq y \text{ iff } x \sqcup y = y \text{ iff } x \sqcap y = x$$

Proof: See 2.8 in [21]. ■

Theorem A.4: Let $(L, \sqsubseteq)$ be a lattice, then the binary operations $\sqcup$ and $\sqcap$ satisfy for all $x, y, z \in L$ the axioms of Definition 2.7.

Proof: See 2.9 in [21]. ■

Theorem A.5: Let $(L, \sqcup, \sqcap)$ be a lattice where the binary operations $\sqcup$ and $\sqcap$ satisfy the axioms of Definition 2.7, then:

1) if we define $\sqsubseteq$ on L by $a \sqsubseteq b \leftrightarrow a \sqcup b = b$, then $\sqsubseteq$ is an order relation.
2) if $\sqsubseteq$ is defined as above, then $(L, \sqsubseteq)$ is a lattice where for all $x, y \in L$ the following hold:

$$x \sqcup y = sup(\{x, y\}) \text{ and } x \sqcap y = inf(\{x, y\})$$

Proof: See 2.10 in [21]. ■

Note that due to the associativity law of the lattices, we can prove by induction that for all x_i of a lattice $\mathfrak{L}$ we have:

$$\bigsqcup\{x_1, \ldots, x_n\} = x_1 \sqcup x_2 \sqcup \ldots \sqcup x_n \tag{1}$$

as well as the dual of that. Hence, we can state the following lemma.

Lemma A.6: Let $\mathfrak{L}$ be a lattice, then $\bigsqcup X$ and $\bigsqcap X$ exist for any finite, non-empty subset X of L. Hence, every finite lattice is complete.

The first of the following lemmas gives us insights on the structure of a lattice, while the second one is fundamental in the context of distributive lattices.

Lemma A.7: Let $\mathfrak{L}$ be a lattice and $x, y, z \in L$, then $x \sqcap (y \sqcup z) \sqsupseteq (x \sqcap y) \sqcup (x \sqcap z)$ and the dual.

Proof: Notice that $y \sqsubseteq y \sqcup z$ and $z \sqsubseteq y \sqcup z$. We know that $\sqcap$ is a monotonic function (Lemma A.2), hence $x \sqcap y \sqsubseteq x \sqcap (y \sqcup z)$. In the same way, we get $x \sqcap z \sqsubseteq x \sqcap (y \sqcup z)$. Now consider

$$\begin{aligned}((x \sqcap y) \sqcup (x \sqcap z)) \sqcup (x \sqcap (y \sqcup z)) &= (x \sqcap y) \sqcup (x \sqcap z) \sqcup (x \sqcap (y \sqcup z)) \\ &= \bigsqcup\{x \sqcap y, x \sqcap z, x \sqcap (y \sqcup z)\} \\ &= x \sqcap (y \sqcup z)\end{aligned}$$

Thus by the connecting lemma (see Lemma A.3 above) we conclude that $(x \sqcap y) \sqcup (x \sqcap z) \sqsubseteq x \sqcap (y \sqcup z))$. ■

Lemma A.8: Let $\mathfrak{L}$ be a lattice, then the statement $(\forall x, y, z \in L).(x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z))$ is equivalent with its dual, i.e. $(\forall x, y, z \in L).(x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z))$.

Proof: See 4.3 in [21]. ■

Lemma A.9: Let $(X, \sqsubseteq_X)$, $(Y, \sqsubseteq_Y)$ be posets, $Z \subseteq X$ and $f : X \to Y$ be an order preserving map, then if $\bigsqcup f(Z)$ exists in Y and $\bigsqcup Z$ exists in X, we have $\bigsqcup f(Z) \sqsubseteq_Y f(\bigsqcup Z)$. Also, if $\bigsqcap f(Z)$ and $\bigsqcap Z$ exist, then $\bigsqcap f(Z) \sqsupseteq_Y f(\bigsqcap Z)$.

Proof: The proof is by induction on the size of the subset Z. For $|Z| = 1$ the relationship is trivially satisfied. Let $|Z| = k$ and $\bigsqcup f(Z) \sqsubseteq_Y f(\bigsqcup Z)$. Consider $Z' = Z \cup \{z_{k+1}\}$. It is $z_{k+1} \sqsubseteq_X (\bigsqcup Z) \sqcup z_{k+1}$ and $\bigsqcup Z \sqsubseteq_X (\bigsqcup Z) \sqcup z_{k+1}$, which implies that $f(z_{k+1}) \sqsubseteq_Y f(\bigsqcup Z')$ and $f(\bigsqcup Z) \sqsubseteq_Y f(\bigsqcup Z')$. Hence using the Lemma A.2 above, we get $f(z_{k+1}) \sqcup f(\bigsqcup Z) \sqsubseteq_Y f(\bigsqcup Z')$. Now using the induction hypothesis we get $f(z_{k+1}) \sqcup \bigsqcup f(Z) = \bigsqcup f(Z') \sqsubseteq_Y f(\bigsqcup Z')$. ■

Lemma A.10: Every continuous function $f : X \to Y$ is also order-preserving.

Proof: Let $x_1, x_2 \in X$ and $x_1 \sqsubseteq_X x_2$. Thus, $x_1 \sqcup x_2$ exists and $x_1 \sqcup x_2 = x_2$. Hence, $f(x_1 \sqcup x_2) = f(x_2)$ and as f is continuous $f(x_1 \sqcup x_2) = f(x_1) \sqcup f(x_2)$. Thus, we conclude that $f(x_2) = f(x_1) \sqcup f(x_2)$ which implies $f(x_1) \sqsubseteq_Y f(x_2)$ by the connecting lemma (see Lemma A.3 above). ■

Lemma A.11: Let $(X, \sqsubseteq_X)$ and $(Y, \sqsubseteq_Y)$ be finite posets and $f : X \to Y$ be an order-preserving function, then f is also continuous.

Proof: See Lemma 5 in Chapter 6 in [19] for a proof when $X = Y$ and $\cap$ or $\cup$ continuity. ■

Proposition A.12: Let $\mathfrak{L}$ be a finite lattice, then:

- Let $y, z \in L$ and $y \not\sqsubseteq z$, then there exists $x \in \mathcal{J}(\mathfrak{L})$ such that $x \sqsubseteq y$ and $x \not\sqsubseteq z$.
- For all $x \in L$ it is $x = \bigsqcup\{y \in \mathcal{J}(\mathfrak{L}) \mid y \sqsubseteq x\}$

Proof: See Proposition 2.45 in [21]. ■

Lemma A.13: Let $\mathfrak{L}$ be a distributive lattice and let $x \in L$ with $x \neq \bot$ in case L has a bottom, then the following are equivalent:

1) x is join-irreducible
2) if $y, z \in L$ and $x \sqsubseteq y \sqcup z$, then $x \sqsubseteq y$ or $x \sqsubseteq z$
3) for any $k \in nat$, if $y_1, \ldots, y_k \in L$ and $x \sqsubseteq y_1 \sqcup \ldots \sqcup y_k$, then $x \sqsubseteq y_i$ for some $1 \leq i \leq k$.

Proof: See Lemma 5.11 in [21]. ■

B. Notes on Temporal Logics and Model Checking

Proof: **[Lemma 3.1]** The proof of the lemma is straightforward. For the proof of the $[\![EX\phi]\!]_{\mathcal{K}}$ case see Lemma 2.43 in [57]. Here, we will just prove the $[\![AX\phi]\!]_{\mathcal{K}}$ case (there exists a shorter proof derived from the

EX operator):

$$
\begin{aligned}
[\![AX\phi]\!]_{\mathcal{K}} &= \{s \in S \mid (\mathcal{K}, s) \models_s AX\phi\} \\
&= \{s \in S \mid (\forall \pi \in Paths_{\mathcal{K}}(s)).((\mathcal{K}, \pi[0]) \models_p X\phi)\} \\
&= \{s \in S \mid (\forall \pi \in Paths_{\mathcal{K}}(s)).((\mathcal{K}, \pi[1]) \models_p \phi)\} \\
&= \{s \in S \mid (\forall s' \in S).((s, s') \in R \rightarrow (\forall \pi \in Paths_{\mathcal{K}}(s')).((\mathcal{K}, \pi[0]) \models_p \phi)\} \\
&= \{s \in S \mid (\forall s' \in S).((s, s') \in R \rightarrow (\forall \pi \in Paths_{\mathcal{K}}(s')).((\mathcal{K}, s') \models_s \phi)\} \\
&= \{s \in S \mid (\forall s' \in S).((s, s') \in R \rightarrow (\forall \pi \in Paths_{\mathcal{K}}(s')).(s' \in [\![\phi]\!]_{\mathcal{K}})\} \\
&= \{s \in S \mid (\forall s' \in S).((s, s') \in R \rightarrow \neg(\exists \pi \in Paths_{\mathcal{K}}(s')).(s' \notin [\![\phi]\!]_{\mathcal{K}})\} \\
&= \{s \in S \mid (\forall s' \in S).((s, s') \in R \rightarrow \neg(s' \in S_{inf} \wedge s' \notin [\![\phi]\!]_{\mathcal{K}})\} \\
&= \{s \in S \mid (\forall s' \in S).((s, s') \in R \rightarrow (s' \notin S_{inf} \vee s' \in [\![\phi]\!]_{\mathcal{K}})\} \\
&= \{s \in S \mid (\forall s' \in S).((s, s') \in R \rightarrow (s' \in S \backslash S_{inf} \vee s' \in [\![\phi]\!]_{\mathcal{K}})\} \\
&= \{s \in S \mid (\forall s' \in S).((s, s') \in R \rightarrow s' \in ((S \backslash S_{inf}) \cup [\![\phi]\!]_{\mathcal{K}})\} \\
&= pre^R_\forall((S \backslash S_{inf}) \cup [\![\phi]\!]_{\mathcal{K}})
\end{aligned}
$$

■