# Model-Checking Infinite State-Space Systems with Fine-Grained Abstractions Using SPIN

Marsha Chechik, Benet Devereux, and Arie Gurfinkel

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

**Abstract.** In analyzing infinite-state systems, it is often useful to define multiple-valued predicates. Such predicates can determine the (finite) levels of desirability of the current system state and transitions between them. We can capture multiple-valued predicates as elements of a logic defined over finite total orders (FTOs). In this paper we extend automata-theoretic LTL model-checking to reasoning about a class of multiple-valued logics. We also show that model-checking over FTOs is reducible to classical model-checking, and thus can be implemented in SPIN.

## 1 Introduction

Currently, model-checking is essentially limited to reasoning about medium-sized finite-state models. Reasoning about large models, especially if these are not finite-state, is typically done using abstraction [CGL94]. Abstraction techniques, such as *abstract interpretation* [CC77], require the user to supply the mapping between concrete and abstract data types in their models. *Predicate abstraction*, introduced by Graf and Saidi [GS97], is a form of abstraction specified as a number of predicates over the concrete data. For example, if we are interested in checking whether $x$ is always positive, we can define predicates $x > 0$ and $x \leq 0$, and use them to compute the abstract system. A number of researchers, e.g., [CU98,VPP00,BDL96,DDP99,SS99], explored the use of predicate abstraction.

However, boolean predicates often do not give the desired precision. For example, consider reasoning about a leader-election protocol, parameterized by $N$ – the number of processes engaged in it. We can either set $N$ to be a (small) constant, and define predicates on the exact number of processes that have agreed on the elected leader; or leave $N$ as is, and define predicates such as "everyone agreed on the leader", "no one agreed on the leader", etc. In this situation we cannot ask questions about the likelihood of the agreement, whereas such questions may be desirable.

As an alternative, we propose modeling such systems using multiple-valued predicates, where their values form a linear order. In the above situation, we can assign different values to the level of agreement on the leader: "everyone agreed", "the agreement is likely", "no information is available", "the agreement is unlikely", "no one agreed", obtaining a linear order on the level of agreement. Furthermore, if we do not limit ourselves to classical logic, our model-checking procedure will distinguish between different values of agreement, e.g., between cases where no agreement has been reached and where complete agreement has not been reached, but the majority have agreed. Taking this reasoning one step further, we can assign values to transitions. Intuitively, a transition value is the *possibility* that it will be taken. Thus, we can potentially distinguish between paths that can always be taken, paths that can likely be taken, etc.

In fact, giving predicates values from a linear order can be useful in a variety of situations: (a) consensus-building, where the abstraction is over counting (e.g., the leader-election protocol mentioned above); (b) explicitly distinguishing between "regular" and "faulty" behaviors, where we may be interested in properties that hold always, and those that hold "most of the time", i.e., over "regular" behaviors; (c) rechecking a partial system after a change to it has been made, where we are interested in differentiating between possible effects of the change; (d) any situation where we want to assign "desirability" to a transition. This can happen in cases where we have varying tolerances, e.g., in analyzing families of SCR specifications [HJL96].

Note that using linear order-valued predicates does not increase the expressive power of our modeling language, since they can be encoded using a number of boolean predicates. However, such encoding results in cluttering the models with lots of auxiliary variables that bear no natural meaning, and, more importantly, greatly increases the sizes of the models, making model-checking less feasible [HK93].

Multiple-valued reasoning has been explored in a variety of domains. For example, a nine-valued logic is prescribed as a standard [IEE93] for VLSI design, where the interpretation of values is in terms of voltage thresholds. Other examples include databases [Gai79], knowledge representation [Gin87], and machine learning [Mic77]. However, most of the work concentrated on the 3-valued reasoning, with values "True", "Maybe" and "False". Melvin Fitting [Fit91,Fit92] has done seminal work in studying 3-valued modal logic, and our work on logic in this paper is somewhat similar to his. Three-valued logic has also been shown to be useful for analyzing programs using abstract interpretation [CD00,SRW99], and for analyzing partial models [BG99,BG00]. Bruns and Godefroid also proved that automata-theoretic model-checking on 3-valued predicates reduces to classical model-checking.

In this paper we give semantics to automata-theoretic model-checking over arbitrary finite linear orders. We define multiple-valued Büchi automata and multiple-valued LTL and show that such model-checking reduces to a classical problem, and thus can be implemented on top of SPIN. The rest of this paper is organized as follows: we review the definition of linear orders and define multiple-valued sets and relations over them in Section 2. $\chi$LTL, a multiple-valued extension of LTL, is defined in Section 3. Section 4 defines multiple-valued languages and Büchi automata. In Section 5 we show how to represent $\chi$LTL logic formulas as multiple-valued Büchi automata. Section 6 defines the model-checking problem on multiple-valued Büchi automata and shows that it reduces to a number of queries to a classical model-checker, such as SPIN. We conclude the paper in Section 7.

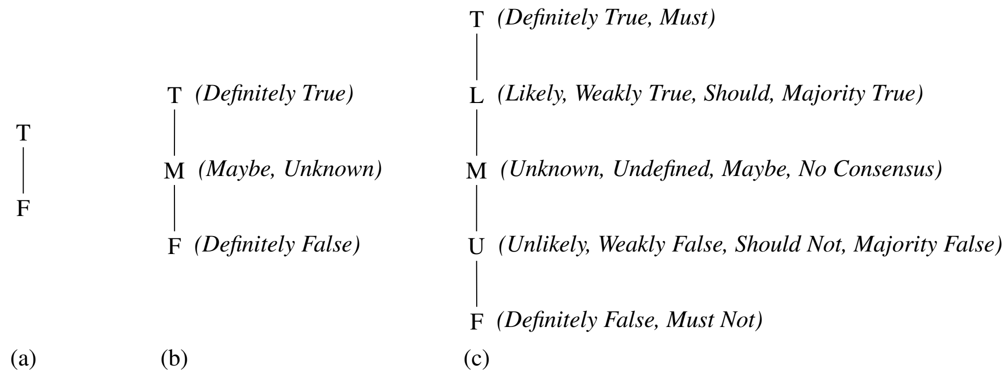


**Fig. 1.** (a) $\mathbf{2}$, the classical logic FTO; (b) $\mathbf{3}$, a three-valued logic FTO; (c) $\mathbf{5}$, a five-valued logic FTO and possible interpretations of its values.

## 2 Preliminaries

In this section we review the definition of logics based on total orders. We also define multiple-valued sets and relations over them.

### 2.1 Finite Total Orders

A *partial order* is a relation which is reflexive, symmetric, and transitive. A *partially ordered set*, usually abbreviated *poset*, is a pair $\mathcal{L} = (\mathcal{O}, \sqsubseteq)$ where $\mathcal{O}$ is a set and $\sqsubseteq$ a partial order defined on it. If, for all $a, b \in \mathcal{O}$, either $a \sqsubseteq b$ or $b \sqsubseteq a$, then $\sqsubseteq$ is a *total order* or *linear order*. We consider, for the purposes of this paper, only finite totally ordered sets, which we refer to as FTOs.

The operations of maximum and minimum are defined on FTOs as follows:

$$
\begin{array}{llll}
a \sqcap b = a \;\Leftrightarrow a \sqsubseteq b & \text{(minimum)} & a \sqcup b = b \;\Leftrightarrow a \sqsubseteq b & \text{(maximum)} \\
b \sqcap a = a \;\Leftrightarrow a \sqsubseteq b & \text{(minimum)} & b \sqcup a = b \;\Leftrightarrow a \sqsubseteq b & \text{(maximum)}
\end{array}
$$

**Lemma 1.** *Let $(\mathcal{O}, \sqsubseteq)$ be an FTO. Then for all $a, b, c \in \mathcal{O}$,*

$$
\begin{array}{ll}
c \sqsubseteq a \sqcap b \Leftrightarrow (c \sqsubseteq a) \wedge (c \sqsubseteq b) & (\texttt{min-}\wedge) \\
c \sqsubseteq a \sqcup b \Leftrightarrow (c \sqsubseteq a) \vee (c \sqsubseteq b) & (\texttt{max-}\vee)
\end{array}
$$

We further define $\bot = \bigsqcap \mathcal{O}$ and $\top = \bigsqcup \mathcal{O}$.

Any FTO of height $n$ is isomorphic to the integers from 0 to $(n-1)$ with the ordinary ordering. We call this isomorphism the *canonical isomorphism for the FTO* and denote it by $\zeta_{\mathcal{L}}$. The *difference* between two elements of an FTO is their absolute difference:

$$a \ominus b = |\zeta_{\mathcal{L}}(a) - \zeta_{\mathcal{L}}(b)|$$

and *negation* in the FTO can be defined in terms of difference:

$$\neg a \triangleq \top \ominus a \qquad \text{(def. of negation)}$$

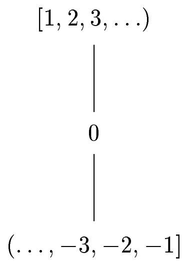


**Fig. 2.** An example of a multiple-valued set over **3**.

FTOs with this definition of negation satisfy the following properties:

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

In this paper we use multiple-valued logics whose truth values form an FTO. Conjunction and disjunction of the logic are defined as $\sqcap$ and $\sqcup$ (meet and join) operations of $(\mathcal{O}, \sqsubseteq)$, respectively, and negation is defined as the $\neg$ operator of $(\mathcal{O}, \sqsubseteq)$. In fact, we will not distinguish between an FTO and a logic it defines, referring to both as $\mathcal{L}$. We also note that most of the usual laws of logic are obtained in $\mathcal{L}$, with the exception of the laws of Universality ($a \sqcap \neg a = \bot$) and Excluded Middle ($a \sqcup \neg a = \top$).

Figure 1 presents several commonly-used FTOs: classical logic, a three-valued logic with uncertainty, and a five-valued logic with more degrees of uncertainty.

### 2.2 Multiple-Valued Sets and Relations

Let $\mathcal{L} = (\mathcal{O}, \sqsubseteq)$ be an FTO and $D$ be some (finite) domain. We consider $\mathcal{O}^D$, the set of all total functions from $D$ into $\mathcal{O}$, and refer to elements of $\mathcal{O}^D$ as *multiple-valued subsets of* $D$, and, when $D$ is clear from the context, just *multiple-valued sets* or MV-sets. We introduced this notion in [CDE01a], and briefly review it below.

**Definition 1.** *Given multiple-valued sets* $A, B \in \mathcal{O}^D$*,*

$$\begin{array}{rcll}
x \in_{\mathcal{L}} A & \triangleq & A(x) & \text{(MV-set membership)} \\
x \in_{\mathcal{L}} A \cup_{\mathcal{L}} B & \triangleq & A(x) \sqcup B(x) & \text{(MV-set union)} \\
x \in_{\mathcal{L}} A \cap_{\mathcal{L}} B & \triangleq & A(x) \sqcap B(x) & \text{(MV-set intersection)} \\
x \in_{\mathcal{L}} \overline{A} & \triangleq & \neg A(x) & \text{(MV-set complement)}
\end{array}$$

Consider the multiple-valued set of Figure 2. In this example, we use the three-valued FTO **3** to model ambiguity about whether 0 is a positive integer. The MV-set is $\mathbb{Z}^{+?} \in \mathbf{3}^{\mathbb{Z}}$, where for all $n \geq 1$, $(n \in_{\mathcal{L}} \mathbb{Z}^{+?}) = \top$; for all $n \leq -1$, $(n \in_{\mathcal{L}} \mathbb{Z}^{+?}) = \bot$; and $(0 \in_{\mathcal{L}} \mathbb{Z}^{+?})$ = M.

**Theorem 1.** *[Gol99] Let $D$ be a finite set, and $(\mathcal{O}, \sqsubseteq)$ be an FTO. Define $\sqsubseteq_{\mathcal{O}^D}$ as follows for any $f, g \in \mathcal{O}^D$:*

$$\begin{array}{|ll|}\hline \varphi\,\mathcal{U}\,\psi & = \psi \vee (\varphi \wedge \circ(\varphi\,\mathcal{U}\,\psi)) \\ \Diamond\psi & = \psi \vee \circ(\Diamond\psi) \\ \Box\psi & = \psi \wedge \circ(\Box\psi) \\ \varphi\,\mathcal{R}\,\psi & = \psi \wedge (\varphi \vee \circ(\varphi\,\mathcal{R}\,\psi)) \\ \hline \end{array}$$

**Fig. 3.** Properties of $\chi$LTL operators.

$$f \sqsubseteq_{\mathcal{O}^D} g \Leftrightarrow \forall d \in D \cdot f(d) \sqsubseteq g(d).$$

*Then* $(\mathcal{O}^D, \sqsubseteq_{\mathcal{O}^D})$ *is an FTO, with MV-union and MV-intersection defined as join and meet, respectively.*

The practical use of this result is that all of the properties defined for FTOs, such as the De Morgan rules and distributivity, carry over to MV-sets.

Given two sets $P, Q$, we can define a *multiple-valued relation* [CDE01a] on them as a multiple-valued subset of $P \times Q$, or an element of $\mathcal{O}^{P \times Q}$.

This work is, to our knowledge, the first use of valued subsets in formal verification; however, such theories are developed elsewhere [Eil78,Gol99].

## 3 $\chi$LTL

In this section we extend the semantics of LTL to allow reasoning over a given FTO $\mathcal{L} = (\mathcal{O}, \sqsubseteq)$, representing our multiple-valued logic. We refer to the resulting language as $\chi$LTL. Just like in classical propositional LTL, formulas in $\chi$LTL are built from a set *Prop* of values of atomic propositions and are closed under the application of propositional operators, the unary temporal connective $\circ$ ("next") and the binary temporal connective $\mathcal{U}$ ("until"). $\chi$LTL is interpreted over multiple-valued computations. A computation is a function $\pi : \mathbb{N} \to \mathcal{O}^{Prop}$ which assigns values from the logic $\mathcal{L}$ to the elements of *Prop* at each time instant (natural number). For a computation $\pi$ and a point $i \in \mathbb{N}$, we have:

$$\begin{array}{ll} \pi, i \models_{\mathcal{L}} p & \triangleq p \in_{\mathcal{L}} \pi(i) \\ \pi, i \models_{\mathcal{L}} \neg\varphi & \triangleq \neg(\pi, i \models_{\mathcal{L}} \varphi) \\ \pi, i \models_{\mathcal{L}} \varphi \wedge \psi & \triangleq \pi, i \models_{\mathcal{L}} \varphi \ \sqcap \ \pi, i \models_{\mathcal{L}} \psi \\ \pi, i \models_{\mathcal{L}} \varphi \vee \psi & \triangleq \pi, i \models_{\mathcal{L}} \varphi \ \sqcup \ \pi, i \models_{\mathcal{L}} \psi \end{array}$$

Now we define the temporal operators:

$$\begin{array}{ll} \pi, i \models_{\mathcal{L}} \circ\varphi & \triangleq \pi, i+1 \models_{\mathcal{L}} \varphi \\ \pi, i \models_{\mathcal{L}} \varphi\,\mathcal{U}\,\psi & \triangleq \bigsqcup\limits_{j \geq i} \big((\pi, j \models_{\mathcal{L}} \psi) \sqcap (\bigsqcap\limits_{i \leq k < j} \pi, k \models_{\mathcal{L}} \varphi)\big) \end{array}$$

The value of a property on a run is the value that it has in the 0th state of the run:

$$\pi \models_{\mathcal{L}} \varphi \ \triangleq \ \pi, 0 \models_{\mathcal{L}} \varphi$$

As usual, $\Diamond\varphi = \top\,\mathcal{U}\,\varphi$, $\Box\varphi = \neg\Diamond\neg\varphi$, and $\varphi\,\mathcal{R}\,\psi = \neg(\neg\varphi\,\mathcal{U}\,\neg\psi)$. $\chi$LTL operators satisfy the expected LTL properties, for example, the fixpoint properties in Figure 3.

Consider the example in Figure 4. This figure presents partial execution of the Leader Election protocol specified using the five-valued logic $\mathbf{5}$. Let $N$ be the number of processes (which we assume to be an even number), and $K$ be the number that

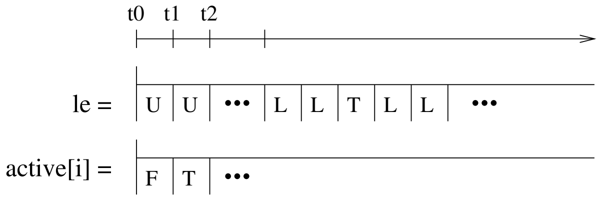


**Fig. 4.** A partial execution of the Leader Election protocol.

have agreed on the leader. We abstract $K$ using the 5-valued predicate $\texttt{le}$ ("leader elected") which is *true* when $K = N$, *weakly true* when $(N/2) < K < N$, *undecided* when $K = N/2$, *weakly false* when $0 < K < (N/2)$, and *false* when $K = 0$. Let $\texttt{active}[i]$ indicate that the $i$th process is currently active. In this system, $\pi, 0 \models_{\mathcal{L}} \texttt{le}$ is U, $\pi, 0 \models_{\mathcal{L}} \texttt{le} \wedge \texttt{active}[\texttt{i}]$ is F ($\perp$), and $\pi, 0 \models_{\mathcal{L}} \neg\texttt{active}[\texttt{i}]$ is T ($\top$). The value for $\texttt{le}$ indicates that originally there was no consensus on the leader (U), then consensus started forming (L) and was reached (T). However, in the next state one of the processes changed its mind, and thus the consensus went back to L. For this run, the value of $\Diamond\texttt{le}$ is T, but the value of $\Diamond\Box\texttt{le}$ is L. Note that we get this value without the need to re-annotate our model under a different level of abstraction and rerun the check.

## 4 Multiple-Valued Languages and Automata

In the task of using multiple-valued logic for system specification and verification, it is natural to consider *multiple-valued formal languages* and *multiple-valued automata*. We introduce them in this section.

### 4.1 Multiple-Valued Languages

Let $\Sigma$ be a finite alphabet, $\Sigma^*$ be the set of all *finite* words over $\Sigma$, $\Sigma^\omega$ be the set of all *infinite* words, and $\Sigma^{\leq\omega} = \Sigma^* \cup \Sigma^\omega$. We can catenate any two finite words, and consider the *empty string* $\lambda$ as the identity for catenation: $w\lambda = \lambda w = w$. The empty string is contained in $\Sigma^*$, but not in $\Sigma^\omega$.

**Definition 2.** *A* multiple-valued language *over an alphabet $\Sigma$ is a multiple-valued subset of $\Sigma^*$, or an element in $\mathcal{O}^{\Sigma^*}$; a* multiple-valued $\omega$-language *is an element in $\mathcal{O}^{\Sigma^\omega}$. A multiple-valued language $X$ is* proper *if* $(\lambda \in_{\mathcal{L}} X) = \top$.

We shall use the term "MV-language" to refer, indiscriminately, to any multiple-valued language or $\omega$-language, wherever the distinction is not important. An MV-language is an assignment of values to words. If $\mathcal{O} = \mathbf{2}$, then an MV-language is an ordinary formal language, where every word that is assigned value $\top$ is considered to be in the language. The properness criterion assures that $\lambda$ is contained in the language as the identity for catenation.

MV-languages are just MV-sets of words, so union, intersection, and complement are already defined on them. The standard language operation of *catenation* can be extended to the multiple-valued case, as given below.

**Definition 3.** *Given $X, Y \in \mathcal{O}^{\Sigma^*}$ and $w \in \Sigma^*$,*

$$w \in_{\mathcal{L}} XY \triangleq \bigsqcup_{\{u,v \mid w=uv\}} (u \in_{\mathcal{L}} X) \sqcap (v \in_{\mathcal{L}} Y) \text{ (MV-language catenation)}$$

Transitive closure (Kleene star) and infinite closure ($\omega$) can be defined in terms of multiple-valued catenation.

Consider the two multiple-valued languages $X = \{a \to T, ab \to L\}$ and $Y = \{bc \to M, c \to U\}$, defined on the logic $\mathbf{5}$. We are interested in the value that *abc* has in $XY$. It can be formed either by catenating $a$ and *bc*, with value $T \sqcap M = M$, or by catenating *ab* and $c$, with value $L \sqcap U = U$. By the definition, we take the maximum of those two values, making the value of $abc \in_{\mathcal{L}} XY$ to be $M$.

### 4.2 Multiple-Valued Automata

A multiple-valued finite automaton $A$ takes any word $w \in \Sigma^{\leq\omega}$ and computes its membership degree, a value in $\mathcal{O}$. Thus, an automaton corresponds to a multiple-valued language $L(A)$. Details about multiple-valued automata on *finite* words (in the more general case, of semiring-valued languages) can be found elsewhere [Eil78]; our treatment of multiple-valued infinite words and their automata is, so far as we know, new, but it is a natural extension.

A multiple-valued Büchi automaton has transitions between states that take on some value ranging between $\top$ or $\bot$ of an FTO. This value, intuitively, is a *possibility* that a transition will be taken. Thus, we can assign possibilities to individual transitions and to infinite strings that the automaton receives.

**Definition 4.** *A* multiple-valued Büchi automaton*, or χBüchi automaton, is a tuple $(\mathcal{L}, Q, q_0, \Sigma, \Delta, F)$ where:*

- *$\mathcal{L} = (\mathcal{O}, \sqsubseteq)$ is an FTO;*
- *$Q$ is a finite set of states;*
- *$q_0$ is the unique initial state;*
- *$\Sigma$ is a finite alphabet;*
- *$\Delta \in \mathcal{O}^{Q \times \Sigma \times Q}$ is the multiple-valued transition relation. $\Delta(q, \alpha, q')$ gives the value of the transition from $q$ to $q'$ on symbol $\alpha$;*
- *$F$ is a set of accepting states.*

The *runs* of the automaton are infinite sequences of states, always beginning with $q_0$. We define a projection of $Q$ onto $F$ as

$$\pi_F(q) = \begin{cases} q & \text{if } q \in F \\ \lambda & \text{otherwise} \end{cases}$$

which we extend to $Q^\omega$, and define the *accepting runs* $\mathcal{AR}$ of the automaton to be the elements of

$$\{\sigma \mid \pi_F(\sigma) \in F^\omega\}.$$

Intuitively, $\mathcal{AR}$ is the set of all runs in which some accepting state occurs infinitely often.

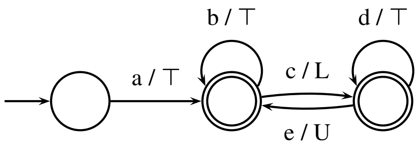


**Fig. 5.** An example $\chi$Büchi automaton.

For a $\chi$Büchi automaton $A$, $L(A) \in \mathcal{O}^{\Sigma^\omega}$ is the multiple-valued subset of $\Sigma^\omega$ defined by the automaton. The value assigned by the automaton to a word $w = w_0 w_1 w_2 \ldots$ in $\Sigma^\omega$ is given in terms of the accepting runs:

$$\big(w \in_{\mathcal{L}} L(A)\big) = \bigsqcup_{\sigma \in \mathcal{AR}} \bigsqcap_{i \in \mathbb{N}} \Delta(\sigma_i, w_i, \sigma_{i+1})$$

Consider the $\chi$Büchi automaton in Figure 5. This automaton assigns values from $\mathbf{5}$ to its inputs. In the input sequence $abbcd^\omega$, the prefix $abb$ takes the automaton only through $\top$-valued transitions. Then, $c$ follows an L-transition to an accepting state; after this occurs, the value of the whole sequence *cannot* exceed L. The automaton loops through the accepting state on the infinite sequence of $d$'s, so this word is accepted with value L.

$\chi$Büchi automata are similar in spirit to *Markov chains* [Fel68]. Markov chains also assign values, representing probabilities, to nonterminating finite-state computations, and have been used [VW86] to check probabilistic system specifications. Our approach is more *possibilistic*, motivated by the problem of requirements analysis. Given two independent events, the *probability* of the occurrence of at least one is the sum of their individual probabilities; but the *possibility* or *necessity* of at least one event occuring is the *maximum* of their individual possibilities.

### 4.3 Composition

Our definitions of parallel composition, synchronous and asynchronous, are extensions of the standard construction [Tho90].

We start by defining synchronous parallel composition, or MV-intersection of languages. Let $L_1$ and $L_2$ be multiple-valued $\omega$-languages and $A_1 = (\mathcal{L}, Q_1, q_0^1, \Sigma, \Delta_1, F_1)$ and $A_2 = (\mathcal{L}, Q_2, q_0^2, \Sigma, \Delta_2, F_2)$ be $\chi$Büchi automata for $L_1$ and $L_2$, respectively. We construct two classical automata, $\hat{A}_i$ (for $i = 1, 2$), where $\hat{\Delta}_i(q, \alpha, q')$ is true exactly if $\Delta_i(q, \alpha, q') \neq \bot$. Then we intersect the two classical automata, creating $\hat{A}_{12} = (Q_1 \times Q_2 \times \{0,1,2\}, (q_0^1, q_0^2, 0), \Sigma, \hat{\Delta}_{12}, F_1 \times F_2 \times \{2\})$. Finally, we create the *multiple-valued intersection* of the two $\chi$Büchi automata by transforming $\hat{A}_{12}$ into a $\chi$Büchi automaton $A_{12}$ with the new multiple-valued transition relation:

$$\Delta_{12}\big((q,r,j), \alpha, (q', r', j')\big) = \begin{cases} \Delta_1(q, \alpha, q') \sqcap \Delta_2(r, \alpha, r') & \text{if } \hat{\Delta}_{12}\big((q,r,j), \alpha, (q', r', j')\big) \\ \bot & \text{otherwise} \end{cases}$$

for all $j \in \{0, 1, 2\}$.

**Theorem 2.** *The value that $A_{12}$ gives to a word $w$ is the same as its value in $L_1 \cap_{\mathcal{L}} L_2$.*

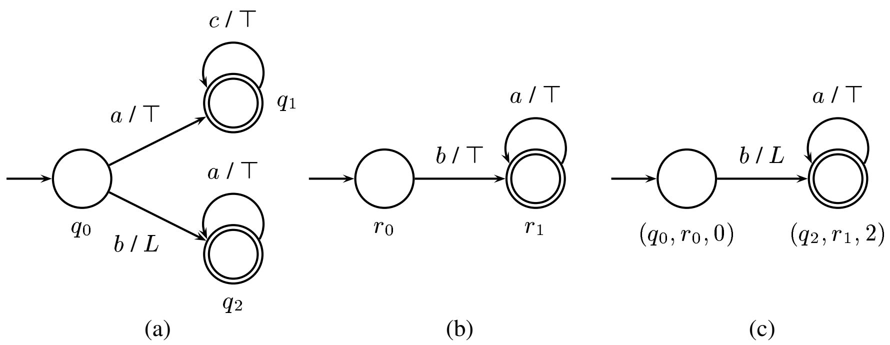


**Fig. 6.** Intersection of $\chi$Büchi automata. (c) shows the intersection of automata in (a) and (b).

Figure 6 illustrates the intersection construction. The first automaton gives the value $\top$ to $ac^\omega$ and L to $ba^\omega$; the second gives value $\top$ to $ba^\omega$. Every other word evaluates to $\perp$. In the intersection, $ac^\omega$ becomes $\perp$, and $ba^\omega$ evaluates to the minimum of L and $\top$, namely L. Note that $(q_2, r_1)$ is labelled with 2, making it an accepting state in the intersection automaton, because it is a final state in *both* $A_1$ and $A_2$.

We proceed to define asynchronous composition on two $\chi$Büchi automata with (possibly different) alphabets and the same logic.

**Definition 5.** *Let* $A_1 = (\mathcal{O}, Q_1, q_0^1, \Sigma_1, \Delta_1, F_1)$, $A_2 = (\mathcal{O}, Q_2, q_0^2, \Sigma_2, \Delta_2, F_2)$ *be two* $\chi$*Büchi automata. The* asynchronous composition $A_1 || A_2 = (\mathcal{O}, Q_1 \times Q_2, (q_0^1, q_0^2), \Sigma_1 \cup \Sigma_2, \Delta, F)$ *of the two automata has the following transition relation:*

$$\begin{array}{lll} \Delta\big((q_1, q_2), \alpha, (q_1', q_2)\big) = \Delta_1(q_1, \alpha, q_1') & \text{if } q_1 \neq q_1' \\ \Delta\big((q_1, q_2), \alpha, (q_1, q_2')\big) = \Delta_2(q_2, \alpha, q_2') & \text{if } q_2 \neq q_2' \\ \Delta\big((q_1, q_2), \alpha, (q_1, q_2)\big) = \Delta_1(q_1, \alpha, q_1) \sqcup \Delta_2(q_2, \alpha, q_2) & \text{otherwise} \end{array}$$

*A state* $(q_1, q_2)$ *of the asynchronous composition is considered final if either* $q_1$ *or* $q_2$ *are final, so*

$$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$$

## 5 Conversion between $\chi$LTL and $\chi$Büchi Automata

In this section we describe how to convert between $\chi$LTL formulas, defined in Section 3 and $\chi$Büchi automata. Our algorithm is based on the classical LTL to Büchi automata conversion algorithm presented in [GPVW95].

As in [GPVW95], we start by defining *Generalized* $\chi$*Büchi Automata* and *Labeled Generalized* $\chi$*Büchi Automata (LG*$\chi$*BA)*.

**Definition 6.** *A* Generalized $\chi$Büchi automaton *(G*$\chi$*BA) is a tuple* $(\mathcal{L}, Q, q_0, \Sigma, \Delta, \mathcal{F})$ *where* $\mathcal{L}, Q, q_0, \Sigma$ *and* $\Delta$ *are as in ordinary* $\chi$*Büchi automata, but* $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$ *is a set of k sets of accepting states. Each set* $F_i$ *has the projection* $\pi_{F_i}$ *defined for it, and the accepting runs are those where at least one element from* each $F_i$ *appears infinitely often:*

$$\mathcal{AR} = \{\sigma \mid \sigma \in q_0 Q^\omega \wedge \forall i \leq k \cdot \pi_{F_i}(\sigma) \in F_i^\omega\}$$

**Definition 7.** *A* Labeled Generalized $\chi$Büchi Automaton (LG$\chi$BA) *is a tuple* $(\mathcal{L}, Q, q_0, \Sigma, \Delta, \mathcal{F}, Lab)$ *where:*

- $\mathcal{L} = (\mathcal{O}, \sqsubseteq)$ *is an FTO;*
- $Q$ *is a finite set of states;*
- $q_0$ *is the unique initial state;*
- $\Sigma = \mathcal{O}^{Prop}$ *is an alphabet consisting of all multiple-valued sets over the set* $Prop$ *of propositional symbols;*
- $\Delta \in \mathcal{O}^{Q \times Q}$ *is a multiple-valued transition relation;*
- $\mathcal{F} = \{F_1, F_2, \ldots F_n\}$ *is a set of sets of accepting states;*
- $Lab : Q \to 2^{Prop \cup \neg Prop}$ *is a labeling function that assigns a subset of* $Prop \cup \neg Prop$ *to every state.*

The set of accepting runs ($\mathcal{AR}$) for a LG$\chi$BA is defined the same as for a Generalized $\chi$Büchi automaton given in Section 4.

Notice that each element $\alpha \in \Sigma$ is a total function from $Prop$ to $\mathcal{O}$. We extend this function to elements of $\neg Prop$ by defining $\alpha(\neg p) \triangleq \neg\alpha(p)$, $\forall p \in Prop$. Let $\hat{\alpha} : 2^{Prop \cup \neg Prop} \to \mathcal{O}$ be a set-wise extension of $\alpha$, defined as

$$\hat{\alpha}(D) \triangleq \bigsqcap_{d \in D} \alpha(d) \qquad \text{(set-wise extension)}$$

For a Labeled Generalized $\chi$Büchi automaton $A$, $L(A) \in \mathcal{O}^{\Sigma^\omega}$ is the multiple-valued subset of $\Sigma^\omega$ defined by the automaton. The value assigned by the automaton to a word $w = w_0 w_1 w_2 \ldots$ in $\Sigma^\omega$ is given in terms of the accepting runs:

$$w \in_{\mathcal{L}} L(A) = \bigsqcup_{\sigma \in \mathcal{AR}} \bigsqcap_{i \in \mathbb{N}} \Delta(\sigma_i, \sigma_{i+1}) \sqcap \hat{w}_i\big(Lab(\sigma_{i+1})\big)$$

where $\hat{w}_i$ is the set-wise extension of $w_i$.

Given an LTL property $\varphi$, the algorithm in [GPVW95] constructs a Labeled Generalized Büchi automaton in two major steps. In the first step, it uses the syntactic structure of the formula to construct a graph $G = (V, E)$ together with three labeling functions, *New*, *Old*, and *Next*, that assign a subset from a *closure* of $\varphi$ to each node of $G$. In the second step, the algorithm constructs an automaton, using $G$ to define its basic structure, and the labeling functions to define its accepting states and state labels. The resulting Generalized Labeled Büchi automaton accepts a word if and only if the word satisfies $\varphi$. This automaton can be easily converted into a Büchi automaton with a polynomial blowout in its size.

Since $\chi$LTL is syntactically equivalent to LTL, we reuse the graph construction part of the algorithm in [GPVW95]. Thus, given a $\chi$LTL property $\varphi$, our algorithm starts by constructing a graph $G = (V, E)$ and the node labeling functions *New*, *Old*, and *Next* using the procedure in [GPVW95]. However, we modify this procedure to ensure the correct handling of $p \wedge \neg p$ (not necessarily $\bot$) and $p \vee \neg p$ (not necessarily $\top$), where $p$ is any propositional formula. The algorithm then proceeds to construct a LG$\chi$BA $A = (\mathcal{L}, Q, q_0, \Sigma, \Delta, \mathcal{F}, Lab)$ by letting the set of states $Q$ of the automaton be the nodes of $G$, with the root node of $G$ being the initial state $q_0$. The accepting set $\mathcal{F}$ is constructed as in the original algorithm. The transition relation $\Delta$ is constructed from the edges of the graph $G$ such that $\Delta(q, q') = \top$ if the edge $(q, q')$ is in $G$, and $\Delta(q, q') =$

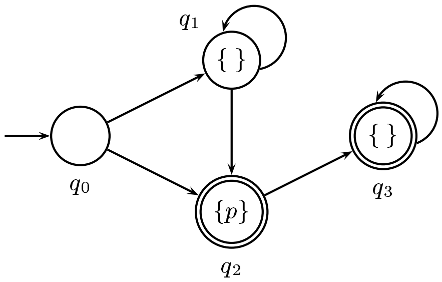


**Fig. 7.** A LG$\chi$BA corresponding to $\Diamond p$.

$\perp$ otherwise. Finally, the labeling function $Lab$ is constructed as a restriction of the labeling function *Old* to the set of all positive and negative propositional symbols of $\varphi$; that is, for a given state $q$, $Lab(q) = Old(q) \cap (Prop \cup \neg Prop)$. It is easy to show that the resulting LG$\chi$BA can be transformed into a $\chi$Büchi automaton via an extended version of the transformation used in the classical case.

For example, consider the automaton in Figure 7 which corresponds to a $\chi$LTL property $\Diamond p$. In this figure we show only $\top$ transitions. Every accepting run of this automaton must pass through the state $q_2$. Therefore, the value that the automaton assigns to a given word $w$ is

$$\bigsqcup_{i \in \mathbb{N}} (p \in_{\mathcal{L}} w_i)$$

which corresponds to the definition of $w \models_{\mathcal{L}} \Diamond p$ from Section 3.

**Theorem 3.** *The automaton $A$ constructed for a property $\varphi$ assigns a value $\ell$ to an infinite sequence $w$ over $\mathcal{O}^{Prop}$ if and only if $\ell = (w \models_{\mathcal{L}} \varphi)$.*

*Proof.* The proof is a straightforward extension of the proof of correctness of the algorithm in [GPVW95], and is omitted here. $\Box$

The immediate consequence of Theorem 3 is that if $\mathcal{L}$ is $\mathbf{2}$, the automaton constructed by our algorithm is equivalent to the Labeled Generalized Büchi automaton produced by the original algorithm in [GPVW95].

## 6 $\chi$LTL Model-Checking

In this section we define automata-theoretic multiple-valued model-checking and describe a decision procedure for it.

### 6.1 The Model-Checking Problem

Automata-theoretic model-checking procedure can be viewed as a function that receives a program $P$ and property $\varphi$ and returns a value from the logic $\mathcal{L}$ indicating the possibility that (the degree to which) $P$ satisfies $\varphi$. For example, in the classical case $MC(P, \varphi) = \top$ if and only if every computation of $P$ satisfies $\varphi$. In the remainder of the paper we use $MC$ to indicate the classical model-checking function. $MC$ is formally defined as

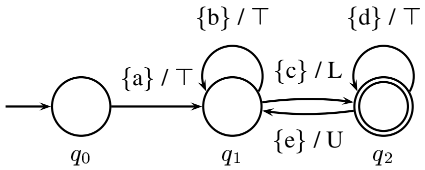


**Fig. 8.** An example Büchi automaton 2.

$$MC(P, \varphi) \triangleq \forall w \in \Sigma^\omega \cdot w \in L(A_P) \to w \in L(A_\varphi) \qquad (MC\text{-definition})$$

where $A_P$ and $A_\varphi$ are the Büchi automata corresponding to the program $P$ and property $\varphi$, respectively.

We extend this definition to the multiple-valued case and define a multiple-valued model-checking function $\chi MC$ as follows:

**Definition 8.** *Let $P$ be a multiple-valued program, $\varphi$ a $\chi$LTL property, and $A_P$, $A_\varphi$ the corresponding $\chi$Büchi automata. Then, the multiple-valued model-checking function $\chi MC$ is defined as*

$$\chi MC(P, \varphi) \triangleq \bigsqcap_{w \in \Sigma^\omega} \big(w \in_{\mathcal{L}} L(A_P) \to w \in_{\mathcal{L}} L(A_\varphi)\big) \qquad (\chi MC\text{-definition 1})$$

$$\triangleq \neg \bigsqcup_{w \in \Sigma^\omega} \big(w \in_{\mathcal{L}} L(A_P) \sqcap w \in_{\mathcal{L}} L(A_{\neg\varphi})\big) \qquad (\chi MC\text{-definition 2})$$

Intuitively, the possibility of a program satisfying a property is inversely proportional to the possibility that the program can produce a computation violating the property. For example, consider a $\chi$Büchi automaton in Figure 8, corresponding to some program $P$. The set of propositional symbols of this automaton is $\{a, b, c, d, e\}$, and as each transition is taken, exactly one of these symbols becomes $\top$ and the rest become $\bot$. Thus all transitions are labeled with singleton sets. For example, a transition between $q_1$ and $q_2$ is labeled with $\{c\}/\top$ to indicate that the transition is taken with possibility $L$ when $c$ becomes $\top$ (and $a$, $b$, $d$, $e$ become $\bot$). Any non-$\bot$ computation $w$ of this automaton contains a $w_i$ such that $(d \in_{\mathcal{L}} w_i) = \top$; therefore, the result of $\chi MC(P, \Diamond d)$ is $\top$. That is, the program satisfies the property $\Diamond d$ with the value $\top$. On the other hand, the value of $\chi MC(P, \Diamond\Box d) = L$ since there exists a computation $w = \{a\}(\{c\}\{d\}\{e\})^\omega$, s.t. $\big(w \in_{\mathcal{L}} L(A)\big) = U$ and $\big(w \in_{\mathcal{L}} L(\neg\Diamond\Box d)\big) = \top$.

To establish correctness of our definition we show that it is equivalent to the classical definition when the logic used is $\mathbf{2}$, and that it preserves the expected relationships between programs and $\chi$LTL properties.

**Theorem 4.** *Let $P$ be a program, $\varphi$ be a ($\chi$)LTL property, and $A_P$, $A_\varphi$ be the corresponding ($\chi$)Büchi automata. Then, if the logic $\mathcal{L}$ used to define the $\chi$Büchi automata is $\mathbf{2}$, then*

$$MC(P, \varphi) = \chi MC(P, \varphi)$$

*Proof.* Follows directly from the definitions of $MC$ and $\chi MC$. □

Intuitively, the degree to which a program $P$ satisfies a conjunction of two properties cannot exceed the degree to which it satisfies each of these properties individually. Similarly, the degree to which a program $P$ satisfies a disjunction of two properties is higher then the degree to which it satisfies each of the properties individually. Finally, in the classical case, if two programs satisfy a property, then so does their independent composition. This implies that in the multiple-valued case the degree to which a program $P_1 + P_2$ satisfies a given property $\varphi$ must equal the smallest degree to which each program satisfies the property individually.

**Theorem 5.** *Let $P_1$ and $P_2$ be programs, and $\varphi$ and $\psi$ be $\chi$LTL properties. Then,*

| | | |
|---|---|---|
| (1) | $\chi MC(P, \varphi \wedge \psi) = \chi MC(P, \varphi) \sqcap \chi MC(P, \psi)$ | (property intersection) |
| (2) | $\chi MC(P, \varphi) \sqcup \chi MC(P, \psi) \sqsubseteq \chi MC(P, \varphi \vee \psi)$ | (property union) |
| (3) | $\chi MC(P_1 + P_2, \varphi) = \chi MC(P_1, \varphi) \sqcap \chi MC(P_2, \varphi)$ | (program composition) |

*Proof.* The proof of (1) and (2) is based on the fact that the language of a $\chi$Büchi automaton $A_{\varphi\wedge\psi}$ ($A_{\varphi\vee\psi}$) is the multiple-valued intersection (union) of the languages $L(A_\varphi)$ and $L(A_\psi)$ corresponding to the properties $\varphi$ and $\psi$, respectively. The proof of (3) is based on the fact that the language of a $\chi$Büchi automaton $A_{P_1+P_2}$ is the multiple-valued union of the languages $L(A_{P_1})$ and $L(A_{P_2})$ corresponding to programs $P_1$ and $P_2$, respectively. $\square$

### 6.2 Decision Procedure for $\chi$LTL Model-Checking

In this section we show that a single $\chi$LTL model-checking problem, with an FTO of size $|\mathcal{O}|$, can be transformed into $(|\mathcal{O}| - 1)$ classical model-checking problems.

Recall the definition of $MC(P, \varphi)$. The formal definition is equivalent to the problem of *language containment*; we must check that $L(A_P) \subseteq L(A_\varphi)$. In practice, this is done via checking for emptiness of $L(A_P) \cap \overline{L(A_\varphi)}$, where $\overline{L(A_\varphi)} = L(A_{\neg\varphi})$ [VW86]. A classical $\omega$-language, viewed as an element $Z \in \mathbf{2}^{\Sigma^\omega}$, is *nonempty* if and only if there exists a $w \in \Sigma^\omega$ such that $(w \in_{\mathcal{L}} Z) = \top$; that is,

$$\textit{Nonempty}(Z) \triangleq \top \sqsubseteq \Big( \bigvee_{w \in \Sigma^\omega} (w \in Z) \Big) \qquad \text{(non-emptiness)}$$

We wish to restate $\chi MC\,(\mathcal{P}, \varphi)$ in terms of language intersection and emptiness as well, so we start by generalizing the above definition to MV-languages.

**Definition 9.** *Let $Z$ be an MV-language, $\mathcal{L} = (\mathcal{O}, \sqsubseteq)$ be an FTO, and $\ell \in (\mathcal{O} \setminus \{\bot\})$. Then*

$$\textit{Nonempty}(Z, \ell) \triangleq \ell \sqsubseteq \Big( \bigsqcup_{w \in \Sigma^\omega} (w \in_{\mathcal{L}} Z) \Big) \qquad (\ell\text{-non-emptiness})$$

If $\mathcal{O} = \mathbf{2}$ and $\ell = \top$, this definition reduces to the classical definition of emptiness. In the multiple-valued case, however, we can have degrees of emptiness, and this is captured by the generalized definition. For instance, if the maximal value of any word in an MV-language is M, then it is M-nonempty, but not L-nonempty or $\top$-nonempty.

We now define a reduction on MV-automata w.r.t. a logic value $\ell$, known as an $\ell$-cut [CDE[+]01b].

**Definition 10.** *Let $\mathcal{L} = (\mathcal{O}, \sqsubseteq)$ be an $FTO$. Then for any $\chi$Büchi automaton $A = (\mathcal{L}, Q, q_0, \Sigma, \Delta, F)$ and $\ell \in \mathcal{O}$, an* $\ell$-cut *of $A$, denoted $A^\ell$, is an automaton $(Q, q_0, \Sigma, \Delta^\ell, F)$ where:*

$$\Delta^\ell(q, \alpha, q') = \begin{cases} \top \text{ if } \ell \sqsubseteq \Delta(q, \alpha, q') \\ \bot \text{ otherwise} \end{cases} \quad \text{(definition of } \Delta^\ell\text{)}$$

The conversion from any $A$ to $A^\ell$ can be done in $O(|Q|^2)$ time. Now we establish a few properties of $\ell$-cuts.

**Theorem 6.** *Let $A_1$ and $A_2$ be arbitrary $\chi$Büchi automata. Then*

$$L\big((A_1 \cap_{\mathcal{L}} A_2)^\ell\big) = L(A_1^\ell) \cap L(A_2^\ell) \quad (\ell\text{-cut of language intersection})$$

*Proof.*

$$\begin{aligned}
& w \in L\big((A_1 \cap_{\mathcal{L}} A_2)^\ell\big) \\
\Leftrightarrow\ & \text{Definition of cut} \\
& \ell \sqsubseteq (w \in_{\mathcal{L}} L\big((A_1 \cap_{\mathcal{L}} A_2)\big) \\
\Leftrightarrow\ & \text{Theorem 2} \\
& \ell \sqsubseteq \big(w \in_{\mathcal{L}} (L(A_1) \cap_{\mathcal{L}} L(A_2))\big) \\
\Leftrightarrow\ & \text{MV-intersection} \\
& \ell \sqsubseteq \big((w \in_{\mathcal{L}} L(A_1)) \sqcap (w \in_{\mathcal{L}} L(A_2))\big) \\
\Leftrightarrow\ & \text{min-}\wedge \text{ rule} \\
& \big(\ell \sqsubseteq (w \in_{\mathcal{L}} L(A_1))\big) \wedge \big(\ell \sqsubseteq (w \in_{\mathcal{L}} L(A_2))\big) \\
\Leftrightarrow\ & \text{Definition of cut} \\
& \big(w \in L(A_1^\ell)\big) \wedge \big(w \in L(A_2^\ell)\big) \\
\Leftrightarrow\ & \text{Intersection} \\
& w \in L(A_1^\ell) \cap L(A_2^\ell)\big)
\end{aligned}$$

□

**Theorem 7.** *Let $A_1$ and $A_2$ be arbitrary $\chi$Büchi automata. Then*

$$L\big((A_1 \mid\mid A_2)^\ell\big) = L\big(A_1^\ell \mid\mid A_2^\ell\big) \quad (\ell\text{-cut of parallel composition})$$

*Proof.* It is obvious that all transitions which are *not* self-loops will be in the $\ell$-cut of the composition if and only if they are in the $\ell$-cut of the process which moves on the transition. Let $\Delta$ be the transition relation of $(A_1 \mid\mid A_2)^\ell$, and $\Delta'$ be the transition relation of $(A_1^\ell \mid\mid A_2^\ell)$. We show that the existence of a self-loop in $\Delta$ is equivalent to the existence of a self-loop in $\Delta'$:

$$\begin{aligned}
& \Delta\big((q_1, q_2), \alpha, (q_1, q_2)\big) \\
\Leftrightarrow\ & \text{cut of parallel composition} \\
& \ell \sqsubseteq \Delta_1(q_1, \alpha, q_1) \sqcup \Delta_2(q_2, \alpha, q_2) \\
\Leftrightarrow\ & \text{max-}\vee \\
& (\ell \sqsubseteq \Delta_1(q_1, \alpha, q_1)) \vee (\ell \sqsubseteq \Delta_2(q_2, \alpha, q_2)) \\
\Leftrightarrow\ & \text{definition of cut} \\
& \Delta_1^\ell(q_1, \alpha, q_1)) \vee \Delta_2^\ell(q_2, \alpha, q_2) \\
\Leftrightarrow\ & \text{classical parallel composition} \\
& \Delta'\big((q_1, q_2), \alpha, (q_1, q_2)\big)
\end{aligned}$$

□

Given a system $P$, and a $\chi$LTL property $\varphi$:

1. Convert $\neg\varphi$ to a $\chi$Büchi automaton $A_{\neg\varphi}$ using the method of Section 5.
2. Compute $C = P \cap_{\mathcal{L}} A_{\neg\varphi}$ according to the construction of Section 4.3.
3. For each $\ell \in \mathcal{O}$, construct the cut $C^{\ell}$ and check it for nonemptiness.
4. Let $\ell_{max}$ be the maximal $\ell$ for which $C^{\ell}$ is nonempty.
5. Return $\neg\ell_{max}$.

**Fig. 9.** Decision procedure for multi-valued model-checking.

Cuts can also be used to define the decision procedure for MV-language emptiness.

**Theorem 8.** *Let $\mathcal{L} = (\mathcal{O}, \sqsubseteq)$ be an $FTO$. Then for any $\chi$Büchi automaton $A = (\mathcal{L}, Q, q_0, \Sigma, \Delta, F)$ and $\ell \in \mathcal{O}$, the $\ell$-nonemptiness of $A$ is decidable.*

*Proof.* Construct $A^{\ell}$, the $\ell$-cut of $A$. $L(A^{\ell})$ is nonempty if and only if there is some word $w$, for which there is an accepting run $\sigma \in \mathcal{AR}$ with only $\top$-valued transitions. That is:

$$
\begin{aligned}
& w \in L(A^{\ell}) \\
\Leftrightarrow\ & \text{Büchi acceptance} \\
& \exists \sigma \in \mathcal{AR} \cdot \forall i \in \mathbb{N} \cdot \Delta^{\ell}(\sigma_i, w_i, \sigma_{i+1}) \\
\Leftrightarrow\ & \text{definition of } \Delta^{\ell} \\
& \exists \sigma \in \mathcal{AR} \cdot \forall i \in \mathbb{N} \cdot \ell \sqsubseteq \Delta(\sigma_i, w_i, \sigma_{i+1}) \\
\Leftrightarrow\ & \text{min-}\wedge \text{ rule} \\
& \exists \sigma \in \mathcal{AR} \cdot \ell \sqsubseteq \bigsqcap_{i \in \mathbb{N}} \Delta(\sigma_i, w_i, \sigma_{i+1}) \\
\Leftrightarrow\ & \text{max-}\vee \text{ rule} \\
& \ell \sqsubseteq \bigsqcup_{\sigma \in \mathcal{AR}} \bigsqcap_{i \in \mathbb{N}} \Delta(\sigma_i, w_i, \sigma_{i+1}) \\
\Leftrightarrow\ & \chi\text{Büchi acceptance} \\
& \ell \sqsubseteq \big(w \in L(A)\big)
\end{aligned}
$$

In other words, if $L(A^{\ell})$ is nonempty, then there is some word $w$ such that $\ell \sqsubseteq (w \in_{\mathcal{L}} L(A))$, and $L(A)$ is $\ell$-nonempty. Since $A^{\ell}$ is a classical Büchi automaton, its nonemptiness is decidable [Tho90]. $\square$

We now have an effective decision procedure for finding the $\ell$-nonemptiness of $L(A_P) \cap_{\mathcal{L}} L(A_{\neg\varphi})$ for any $\ell \in \mathcal{O}$. We can iterate this procedure to find the *maximal* $\ell$ for which this intersection is non-empty. The *complement* of this maximal $\ell$ can be returned as the value of property $\varphi$ in system $P$. Figure 9 describes the model-checking procedure in detail. In order to gain some intuition for this result, first consider the classical case, where we simply need to check that the intersection of the system with the negated property automaton is $\top$-nonempty: if it is $\top$-nonempty, there are $\bot$-valued counterexamples to the property.

### 6.3 $\chi$LTL Model-Checking in SPIN

In this section we show how to implement a multi-valued automata-theoretic model-checker, which we call MV-SPIN, using SPIN as a black box.

**procedure** MV-SPIN ($P$, $\varphi$)
    $A_\varphi$ = $\texttt{B2Prom}(\chi\texttt{2B}(\varphi))$
    **for** $\ell$ = $\top$ **downto** $\bot$
        $P'$ = $\texttt{Cut}(P, \ell)$
        $A'_\varphi$ = $\texttt{Cut}(A_\varphi, \ell)$
        $\texttt{ce}$ = SPIN ($P'$, $A'_\varphi$)
        **if** ($\texttt{ce} \neq \emptyset$)
            **return** $\neg\ell$ as answer and
                $\texttt{ce}$, if present, as the counter-example

**Fig. 10.** Algorithm for MV-SPIN.

In Section 6.2 we established that model-checking of a property $\varphi$ over a system $P$ reduces to computing a series of $\ell$-cuts over $P \cap_{\mathcal{L}} A_{\neg\varphi}$. By Theorem 6, we can perform $\ell$-cuts of the property and the system automaton individually. We also note that the system is usually not a monolithic Promela model, corresponding to one Büchi automaton, but a collection of processes which are run under asynchronous parallelism. Furthermore, SPIN does not compute the entire automaton of the model; instead, it performs model-checking on-the-fly [GPVW95]. Thus, our goal is to specify multiple-valued models in some Promela-like language, extended with MV-semantics and then generate Promela without building the complete Büchi automata.

Extending Promela with multiple-valued guard commands is not difficult, as indicated by the work on probabilistic GCL [HSM97]. Asynchronous parallel composition of $\chi$Büchi automata was given in Definition 5. By Theorem 7, $\ell$-cuts of the entire model are equal to $\ell$-cuts of each individual process. Assume that this operation is done by function `Cut` which takes a model in extended Promela and a logic value $\ell$ and converts it into "regular" Promela while performing the reduction $\ell$-cut.

The algorithm for MV-SPIN is given in Figure 10. Functions $\chi$`2B` and `B2Prom` are the modifications of existing LTL to Büchi automata and Büchi automata to Promela algorithms, respectively, enriched to handle $\chi$Büchi automata. The result of SPIN is stored in `ce`. If `ce` is empty, the classical model-checking procedure succeeded; else, `ce` is returned as the counter-example.

Note that the performance penalty of MV-SPIN w.r.t. SPIN manifests itself in a $O(|\mathcal{O}|)$ expansion in the size of the Büchi automaton constructed from the $\chi$LTL property, in executing SPIN up to $|\mathcal{O}|$ times and in executing up to $2 \times |\mathcal{O}|$ cuts. Cuts are performed on individual Promela processes and are proportional to the number of lines in respective text files. Thus, we get an overall $O(|\mathcal{O}|^2)$ performance penalty. However, the sizes of resulting models are smaller than they would have been if we replaced multiple-valued variables by a collection of boolean variables. In addition, FTOs allow to compactly represent incompleteness and uncertainty in the system; such situations can be modeled in classical logic by using additional variables and thus leading to the exponential growth in the size of the state space [CDE01a].

## 7 Conclusion

In this paper we extended classical automata-theoretic model-checking to reasoning over multiple-valued logics, whose values form total linear orders. We gave semantics to a multiple-valued extension of LTL, called $\chi$LTL, described notions of multiple-valued languages and automata, and defined a general model-checking problem. We also showed that the multiple-valued model-checking problem reduces to a set of queries to a classical model-checking procedure, and thus can be easily implemented on top of SPIN.

We further note that FTOs are a subclass of *quasi-boolean logics* – logics based on lattices with specially-defined negation. We used quasi-boolean logics in our previous work [CDE01a,CDE$^+$01b]. In fact, our definitions of $\chi$LTL, $\chi$Büchi automata and multiple-valued model-checking can be used verbatim if we replace FTOs by quasi-boolean logics. Furthermore, Theorem 8 also holds for all join-irreducible [CDE$^+$01b] elements of the lattices. However, we do not yet have an effective decision procedure for other elements of the logic.

## Acknowledgments

We thank members of the University of Toronto formal methods reading group, and in particular Steve Easterbrook and Albert Lai, for many useful discussions. This work was financially supported by NSERC and CITO.

## References

[BDL96] C. Barret, D. Dill, and K. Levitt. “Validity Checking for Combinations of Theories with Equality”. In *Formal Methods in Computer-Aided Design*, volume 1166 of *LNCS*, pages 187–201, November 1996.

[BG99] G. Bruns and P. Godefroid. “Model Checking Partial State Spaces with 3-Valued Temporal Logics”. In *Proceedings of CAV’99*, volume 1633 of *LNCS*, pages 274–287, 1999.

[BG00] G. Bruns and P. Godefroid. “Generalized Model Checking: Reasoning about Partial State Spaces”. In *Proceedings of CONCUR’00*, volume 877 of *LNCS*, pages 168–182, August 2000.

[CC77] P. Cousot and R. Cousot. “Static Determination of Dynamic Properties of Generalized Type Unions”. *SIGPLAN Notices*, 12(3), March 1977.

[CD00] M. Chechik and W. Ding. “Lightweight Reasoning about Program Correctness”. CSRG Technical Report 396, University of Toronto, March 2000.

[CDE01a] M. Chechik, B. Devereux, and S. Easterbrook. “Implementing a Multi-Valued Symbolic Model-Checker”. In *Proceedings of TACAS’01*, April 2001.

[CDE$^+$01b] M. Chechik, B. Devereux, S. Easterbrook, A. Lai, and V. Petrovykh. “Efficient Multiple-Valued Model-Checking Using Lattice Representations”. Submitted for publication, January 2001.

[CGL94] E.M. Clarke, O. Grumberg, and D.E. Long. “Model Checking and Abstraction”. *IEEE Transactions on Programming Languages and Systems*, 19(2), 1994.

[CU98] M. Colon and T. Uribe. “Generating Finite-State Abstractions of Reactive Systems using Decision Procedures”. In *Proceedings of the 10th Conference on Computer-Aided Verification*, volume 1427 of *LNCS*. Springer-Verlag, July 1998.

[DDP99] S. Das, D. Dill, and S. Park. “Experience with Predicate Abstraction”. In *Proceedings of the 11th International Conference on Computer-Aided Verification*, volume 1633 of *LNCS*, pages 160–171. Springer-Verlag, 1999.

[Eil78] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1978.

[Fel68] W. Feller. *An Introduction to Probability Theory and its Applications*, volume I. John Wiley and Sons, New York, 1968.

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

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

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

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

[Gol99] J. S. Golan. *Power Algebras over Semirings*. Kluwer Academic, 1999.

[GPVW95] R. Gerth, D. Peled, M. Vardi, and P. Wolper. "Simple On-the-fly Automatic Verification of Linear Temporal Logic". In *In Proceedings of 15th Workshop on Protocol Specification, Testing, and Verification*, Warsaw, North-Holland, June 1995.

[GS97] S. Graf and H. Saidi. "Construction of Abstract State Graphs with PVS". In *Proceedings of the 9th International Conference on Computer-Aided Verification*, volume 1254 of *LNCS*. Springer-Verlag, 1997.

[HJL96] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. "Automated Consistency Checking of Requirements Specifications". *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, July 1996.

[HK93] R. Hähnle and W. Kernig. Verification of switch-level designs with many-valued logic. In *International Conference LPAR '93*, volume 698. Springer-Verlag, 1993.

[HSM97] J. He, K. Seidel, and A. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2–3):171–192, April 1997.

[IEE93] IEEE Standard 1164–1993. 1993.

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

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

[SS99] H. Saidi and N. Shankar. "Abstract and Model Check while you Prove". In *Proceedings of the 11th Conference on Computer-Aided Verification*, volume 1633 of *LNCS*, pages 443–454, July 1999.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science Publishers B. V., 1990.

[VPP00] W. Visser, S. Park, and J. Penix. "Applying Predicate Abstraction to Model Check Object-Oriented Programs". In *Proceedings of 4th International Workshop on Formal Methods in Software Practice*, August 2000.

[VW86] M. Y. Vardi and P. Wolper. "An Automata-Theoretic Approach to Automatic Program Verification". In *Proceedings of 1st Symposium on Logic in Computer Science*, pages 322–331, Cambridge MA, 1986.