Creating Büchi Automata for Multi-valued Model Checking

Stefan Vijzelaar, Wan Fokkink

Creating Büchi Automata for Multi-valued Model Checking

Stefan Vijzelaar and Wan Fokkink

VU University Amsterdam, The Netherlands
{s.j.j.vijzelaar,w.j.fokkink}@vu.nl

Abstract In explicit state model checking of linear temporal logic properties, a Büchi automaton encodes a temporal property. It interleaves with a Kripke model to form a state space, which is searched for counter-examples. Multi-valued model checking considers additional truth values beyond the Boolean *true* and *false*; these values add extra information to the model, e.g. for the purpose of abstraction or execution steering. This paper presents a method to create Büchi automata for multi-valued model checking using quasi-Boolean logics. It allows for multi-valued propositions as well as multi-valued transitions. A logic for the purpose of execution steering and abstraction is presented as an application.

1 Introduction

Model checking is a technique used to automatically verify whether a system adheres to a given specification; or more specifically for this paper, that a property is never violated during the execution of a system. This can be implemented as a search through a product state space of two interleaved automata: the Kripke model that describes the system under verification; and the Büchi automaton that describes the property under verification. The Kripke model is generally an abstraction of a concrete system; it can be created by hand or is derived with the help of automation from a more detailed model. The Büchi automaton encodes the negation of the property being verified, which is usually expressed in linear temporal logic (LTL). The resulting product state space can then be searched for executions of the system that violate the property.

Algorithms to generate Büchi automata generally assume that the Kripke model and LTL property are based on Boolean logic. We are interested in verification based on multi-valued logics. These logics extend the set of Boolean truth values *true* and *false* with new truth values. Thus additional information can be encoded, such as uncertainty caused by a loss of information during abstraction, or the ability of certain transitions to be enabled or disabled at will during execution. We need multi-valued versions of Kripke models, Büchi automata and LTL to support these applications.

Multi-valued definitions of Kripke models and Büchi automata follow naturally from their Boolean definitions. Creating a multi-valued Büchi automaton, that correctly encodes a temporal property for multi-valued model checking,

however, requires more care. When model checking LTL properties using Boolean logic, it is customary to assume all executions of the Kripke model are infinite.

This assumption can be guaranteed using stutter extension: the ability to extend any finite execution to an infinite one without influencing the validity of certain LTL properties. In the multi-valued setting this is not possible.

In this paper we show how to create multi-valued Büchi automata for LTL properties. We present definitions of the LTL operators that are compatible with multi-valued logics and do not require stutter extension. To ensure correct results for the weak next operator, we introduce the notion of maximality: a progress condition that considers to what degree of truth executions can halt when competing executions can to some degree continue. Based on these revised LTL definitions we describe an algorithm for constructing Büchi automata for multi-valued Kripke models, supporting both multi-valued atomic propositions and transitions.

To give an example of multi-valued LTL model checking we look at the application of execution steering. Our nine-valued steering logic indicates which transitions in a Kripke model can be enabled or disabled during execution. This shows the necessity of maximality to get correct multi-valued results.

Although there is a considerable amount of work on model checking with multi-valued logics (see e.g. [4,9]), to our knowledge there are no algorithms for explicit state multi-valued LTL model checking supporting multi-valued propositions and transitions. Chechik et al. use Büchi automata for verification of multi-valued computations, but with Boolean transitions [5]; Andrade et al. use a SAT solver for multi-valued LTL model checking over quasi-Boolean logics [1].

2 Preliminaries

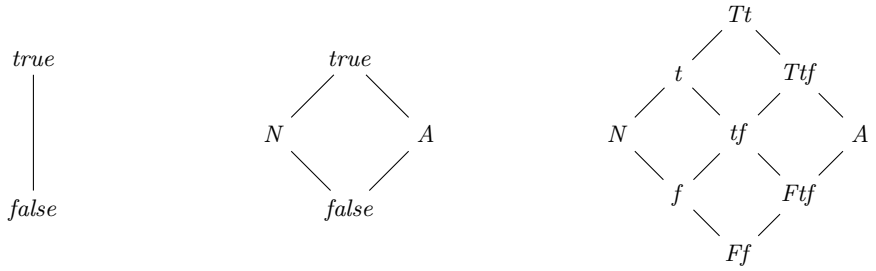
Models and temporal logics typically use Boolean logic: transitions between states either exist or do not exist; atomic propositions either hold for a state or do not hold; and by extension temporal properties over a model can be verified or falsified. They are either *true* or *false*. It is customary to only draw *true* transitions in a graph: missing transitions are assumed to be *false*.

Additional truth values in the logic can increase its expressiveness and lead to more informative answers when verifying a property. Such multi-valued logics, which are logics with more than two truth values, can be defined using lattices.

2.1 Lattices

A lattice $\mathcal{L} = \langle L, \sqsubseteq \rangle$ is a partially ordered ($\sqsubseteq$) set of elements L, in which any two elements have a least upper bound (join or $\sqcup$) and a greatest lower bound (meet or $\sqcap$). A lattice has a join and meet for each non-empty finite subset of elements. Therefore, a non-empty finite lattice is bounded, and has a least element (bottom or $\bot$) and greatest element (top or $\top$). In a distributive lattice meet and join distribute over each other.

A Boolean logic can be described as a lattice consisting of only two elements, with *false* being the bottom and *true* being the top; see Fig. 1a. The Boolean conjunction ($\wedge$) and disjunction ($\vee$) operations map respectively to the meet ($\sqcap$) and join ($\sqcup$) of the lattice. To create a multi-valued logic we can use lattices that have additional elements beyond *true* and *false*.



(a) Boolean (b) Belnap (c) Steering

Figure 1: Distributive lattices

2.2 Quasi-Boolean logics

The multi-valued logics we are interested in are quasi-Boolean logics, also called De Morgan logics. Without the requirements of excluded middle ($x \vee \neg x = \mathit{true}$) and noncontradiction ($x \wedge \neg x = \mathit{false}$), they generalise Boolean logics.

The lattice of a quasi-Boolean logic $\mathcal{L} = \langle L, \leq, \neg \rangle$ is bounded and distributive: the bottom element is *false*, the top element is *true*, meet is used as a conjunction ($\wedge$), and join is used as a disjunction ($\vee$). Negation ($\neg$) requires an appropriate involution which, in addition to being its own inverse, should adhere to De Morgan's laws. It follows by definition that disjunction and conjunction are distributive, and the law of double negation applies.

A typical example of a distributive lattice is the one used for Belnap logic, as depicted in Fig. 1b. This logic can be used to encode may and must transitions resulting from abstraction, using respectively N or *true* for may transitions, and A or *true* for must transitions. The steering logic shown in Fig. 1c can encode steering information and will be explained in more detail later. Per definition its element *Tt* is equal *true* and its element *Ff* is equal to *false*; and one could define the elements N and A as respectively the empty string and *TFtf* for reasons of consistency, but this is deemed impractical.

Note that the lattices in Fig. 1 are depicted as Hasse diagrams in which only the transitive reduction of the partial ordering is represented by lines between elements: an element is smaller in $\leq$ than any directly connected element that is further up. (The transitive closure relates any indirectly connected elements.)

2.3 Multi-valued Kripke models

Multi-valued Kripke models are a generalisation of Kripke models and can use values of any quasi-Boolean logic for transitions and atomic propositions, instead of being limited to the usual Boolean values *true* and *false*. Similarly temporal properties are evaluated over the Kripke model by using the operators as defined by the quasi-Boolean logic. We follow the definition presented in [10].

Definition 1. *A multi-valued Kripke model is a tuple* $M = \langle \mathcal{L}, AP, S, s_0, R, \Theta \rangle$*, where* $\mathcal{L} = \langle L, \leq, \neg \rangle$ *is a quasi-Boolean logic,* AP *a set of atomic propositions,* S *a finite set of states,* s_0 *the initial state,* $R : S \times S \to L$ *a transition relation mapping to truth values of* $\mathcal{L}$*, and* $\Theta : AP \to (S \to L)$ *a labelling function assigning truth values to states for each atomic proposition.*

Definition 2. *A path* $\pi = s_1, s_2, ...$ *is an infinite sequence of states in a multi-valued Kripke model* $M = \langle \mathcal{L}, AP, S, s_0, R, \Theta \rangle$ *with* $s_n \in S$ *for all* $n \geq 1$*. The path is called finite iff* $R(s_k, s_{k+1}) =$ *false for some* $k \geq 1$*, and infinite otherwise.*

2.4 Linear temporal logic

Linear temporal logic (LTL) is used to describe properties of paths through a Kripke model. We use LTL in release positive normal form to aid in our construction of Büchi automata. This is without loss of generality, since any LTL formula can be written in release positive normal form [3]. We also distinguish between a weak and a strong next operator to allow for transitions with truth values different than *true*, for example, when considering finite paths.

Definition 3. *An LTL formula* φ *over a set of atomic propositions* AP *is in release positive normal form if:*

$$\varphi = l \mid p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathsf{X}_s\, \varphi \mid \mathsf{X}_w\, \varphi \mid \varphi_1\, \mathsf{U}\, \varphi_2 \mid \varphi_1\, \mathsf{R}\, \varphi_2$$

With $l \in L$ *a truth value;* $p \in AP$ *a proposition;* X_s *and* X_w *the strong and weak next operators;* U *and* R *the until and release operators; and* $\neg$*,* $\wedge$*,* $\vee$ *the Boolean connectives.*

The strong next operator $\mathsf{X}_s\, \varphi$ requires that the next state on a path is reachable and that φ holds in this next state. The weak next operator $\mathsf{X}_w\, \varphi$ requires that φ holds in the next state on a path or that this next state is unreachable. Note that these definitions coincide when the next state is reachable. The until operator $\varphi\, \mathsf{U}\, \psi$ verifies whether φ holds in all states up to, but not necessarily including, a state where ψ holds. The release operator $\varphi\, \mathsf{R}\, \psi$ verifies whether ψ holds in all states up to, and including, a state where φ holds. For the until operator to hold it is required for ψ to hold eventually; for the release operator it is sufficient when ψ holds indefinitely. Precise semantics of these operators will be presented in the next section when we look at multi-valued LTL.

LTL formulas apply to paths in the Kripke model using its labelling function Θ; but the formulas do not state whether they apply to all paths or a single path.

An LTL property can either be universally quantified, when we want to verify the property, or existentially quantified, when we want to find a counterexample. A property that needs to be verified for all paths can be put in its negated form to look for counterexamples; a step often taken by model checkers.

2.5 Multi-valued Büchi automata

Multi-valued Büchi automata as used in this paper are a generalisation of Boolean non-deterministic Büchi automata by using a multi-valued transition relation.

Definition 4. *A multi-valued non-deterministic Büchi automaton is a tuple* $\mathcal{A} = \langle \mathcal{L}, \Sigma, Q, q_0, \delta, F \rangle$*, where* $\mathcal{L} = \langle L, \leq, \neg \rangle$ *is a quasi-Boolean logic,* Σ *an alphabet,* Q *a finite set of states,* q_0 *the initial state,* $\delta : Q \times \Sigma \times Q \to L$ *a transition relation to truth values of* $\mathcal{L}$*, and* $F \subseteq Q$ *a set of accepting states.*

The quasi-Boolean logic of the Büchi automata is chosen to match the logic of the Kripke model. The alphabet Σ is defined as $\Sigma = L^{AP}$ with AP the set of atomic propositions of the Kripke model; the transition relation δ can then be defined using the operators of the quasi-Boolean logic.

2.6 Bilattices

Bilattices [7] contain two orderings over the same set of elements. A bilattice is distributive if the meet and join operators of both its orderings are distributive with respect to each other, resulting in twelve distributive laws.

Definition 5. *A bilattice is a tuple* $\mathcal{B} = \langle L, \leq_1, \leq_2 \rangle$*, with* L *a set of elements, and* $\leq_1$*,* $\leq_2$ *partial orderings on* L*. Both* $\langle L, \leq_1 \rangle$ *and* $\langle L, \leq_2 \rangle$ *form a lattice.*

In the context of logics and abstractions, one ordering is generally called the truth ordering $\leq_t$ and the other the information ordering $\leq_i$ (see e.g. [10]). The truth ordering, with a suitable definition for negation, defines a quasi-Boolean logic; the information ordering models information loss due to abstraction.

The lattices in Fig. 1b and Fig. 1c are also bilattices: an element is smaller in $\leq_t$ than any directly connected element that is further up, and an element is smaller in $\leq_i$ than any directly connected element that is more to the right. To distinguish between lattice operations of the two orderings, we use $\wedge$ or $\vee$ to indicate a meet or join over $\leq_t$, and $\otimes$ or $\oplus$ to indicate a meet or join over $\leq_i$. For more details on using the information order for abstraction of a Kripke model see [11].

3 Multi-valued LTL

The Boolean definitions of LTL operators can be carried over to a multi-valued logic by using the multi-valued definitions of the Boolean connectives. Some simplifications made to the definitions in Boolean logics, however, do not apply to the multi-valued setting, and can cause problems if not correctly dealt with.

3.1 Stutter extension of Kripke models

In Boolean LTL model checking it is customary to assume that all transitions on paths through the Kripke model are *true*, and that all paths are infinite [3]. This can be ensured by using the stutter invariance of LTL properties without a next operator: the truth of such properties does not change if a state already on a path is finitely repeated. For example the path $\pi_1 = s_1, s_2, s_3, ...$ cannot be distinguished from $\pi_2 = s_1, s_2, s_2, s_2, s_3, ...$ by stutter invariant properties.

The requirement that stuttering is limited to a finite number of repetitions prevents paths from diverging; a path that diverges gets stuck in the repeated state, and never continues on the original path. This however does not apply to deadlock states, since there is no path to continue on; therefore, in Boolean LTL model checking, self loops can be placed on deadlock states, such that finite paths ending in a deadlock state change into infinite paths diverging on the deadlock state. This is called a stutter extension and when applied to a Boolean Kripke model ensures that all its paths are infinite.

3.2 Strong and weak next operators

In Boolean model checking with stutter extensions there is no difference between strong and weak next, since all transitions in a path are *true* and there is always a next state. In multi-valued model checking this is no longer the case, since besides *true* and *false* there can be truth values for which stutter extension is not a solution. Adding a self loop in those cases would cause unwanted divergence, and it would suggest that an execution can simultaneously halt and continue.

Without stutter extensions, even Boolean model checking needs to make a distinction between a strong and weak next operator, but at least the requirement of stutter invariance can be safely dropped.

Definition 6. *Given a single path $\pi = s_1, s_2, ...$ in a multi-valued Kripke model $M = \langle \mathcal{L}, AP, S, s_0, R, \Theta \rangle$. The strong and weak next operators of LTL have the following definitions respectively:*

$$[\mathsf{X}_s\, \varphi]_1 = R(s_1, s_2) \wedge [\varphi]_2 \qquad\qquad [\mathsf{X}_w\, \varphi]_1 = \neg R(s_1, s_2) \vee [\varphi]_2$$

Evaluation of a property ψ over the path $\pi' = s_n, s_{n+1}, ...$ is indicated by $[\psi]_n$.

These definitions only consider a single path in isolation, ignoring all other transitions in the Kripke model that are not a part of it. We will remove this restriction in the following sections when we introduce the notion of maximality; and we will see that it is necessary to consider the truth values of all outgoing transitions for each state in a path.

For Boolean logic, due to the law of excluded middle, an alternative definition of X_w is $\neg R(s_1, s_2) \vee (R(s_1, s_2) \wedge [\varphi]_2)$. This can be rewritten as $(\neg R(s_1, s_2) \vee [\varphi]_2) \wedge (R(s_1, s_2) \vee \neg R(s_1, s_2))$, in which the second disjunct is *true*. In quasi-Boolean logics we lack the law of excluded middle, but the linear-time semantics of LTL still require that transitions are either taken or not: it makes no sense

to evaluate a property over an execution that neither halts nor continues. We assume this requirement holds for each transitions of the Kripke model by taking $R(s_1, s_2) \vee \neg R(s_1, s_2) = \mathit{true}$, resulting in the definition for X_w as given above. This does not introduce any requirements on the Kripke model or make any assumptions on the value of $R(s_1, s_2)$.

3.3 Until and release operators

Using the definitions for strong and weak next, we can define the until and release operators for a single multi-valued path. The next operators, which would otherwise break stutter invariance, will preserve this invariance when used in the context of the until and release operators. Note that the duality $\neg(\varphi \,\mathsf{U}\, \psi) = \neg\varphi \,\mathsf{R}\, \neg\psi$ between until and release is preserved, since we have $\neg\, \mathsf{X}_s\, \varphi = \mathsf{X}_w\, \neg\varphi$.

Definition 7. *Using the weak and strong next operator, the until and release operators have the following expansion laws:*

$$\varphi \,\mathsf{U}\, \psi \equiv \psi \vee (\varphi \wedge \mathsf{X}_s(\varphi \,\mathsf{U}\, \psi)) \qquad \varphi \,\mathsf{R}\, \psi \equiv \psi \wedge (\varphi \vee \mathsf{X}_w(\varphi \,\mathsf{R}\, \psi))$$

By definition $\varphi \,\mathsf{U}\, \psi$ is the least solution of its expansion law and requires that ψ is evaluated at some point, while $\varphi \,\mathsf{R}\, \psi$ is the greatest solution of its expansion law and does not require that φ is evaluated at some point.

Due to the additional requirement on $\varphi \,\mathsf{U}\, \psi$, its expansion can not ignore ψ indefinitely. In disjunctive normal form, only the clauses of finite length are considered: the infinite clause $c = \varphi \wedge \mathsf{X}_s(c)$ is not included in the evaluation.

In Boolean Kripke models, these expansions work as expected. In a path where $\varphi \,\mathsf{U}\, \psi$ encounters a *false* transition, the strong next operator ensures that the property becomes *false* if ψ has not been *true* yet. The strong next requires ψ to hold at some point. Similarly when $\varphi \,\mathsf{R}\, \psi$ encounters a *false* transition, the weak next operator ensures that the property becomes *true* even when φ has not been *true* yet. The weak next allows φ to never hold.

3.4 Paths with false transitions

The definitions of the LTL operators given in the previous sections are correct for a single Boolean or quasi-Boolean path, but can give incorrect results when universally or existentially quantifying over all paths in a Kripke model. This becomes apparent when we consider paths with *false* transitions.

In principle, paths with *false* transitions can be safely ignored if their first *false* transition originates from a non-deadlock state. However, quasi-Boolean logics allow for transitions that are only partially *false* and states that are only partially deadlocked. In the following we investigate paths with *false* transitions to exemplify the issue and reach a more general solution

In Fig. 2a we see a Kripke model with each state labelled by the propositions that are *true* in that state, while propositions that are not part of the label are *false*. All transitions drawn in the figure have the transition value *true*, while

(a) Single path (b) Steering

Figure 2: Example Kripke models

omitted transitions have the transition value *false*. The small incoming arrow indicates the initial state of the model.

If we only consider paths without *false* transitions, then $\varphi \mathsf{U} \psi$ holds universally, but $\psi \mathsf{R} \varphi$ does not hold existentially. This follows from the only path $\pi_1 = \varphi, \psi, \psi, ...$ without *false* transitions. (States are uniquely identified by their propositions in this example.) If we allow paths to include *false* transitions, then we should also consider the path $\pi_2 = \varphi, \varphi, ...$ among others.

Quantifying over all paths, irrespective of transition values, would give incorrect results. The property $\psi \mathsf{R} \varphi$ would hold existentially, since it holds for π_2. (The execution effectively halts after the first φ in the path by taking a *false* transition, and φ is never released.) The property $\varphi \mathsf{U} \psi$ would not hold universally, since it does not hold for π_2. (The execution halts, and ψ will never hold.) We need to adapt the definitions of the LTL operators if we want to use them on a multi-valued Kripke model.

3.5 Maximality

Paths with *false* transitions can give incorrect results. The same problem applies to multi-valued transitions that are only partially *true*. To get correct results we need to consider to what extent a transition is allowed to stop the execution; this is done by taking into account the other transitions from the same state. For the specific case of a *false* transition this means that we only halt the execution to the extent we can not make progress through any of the other transitions. This requirement is formalised using the notion of maximality.

Definition 8. *Given a multi-valued Kripke model* $M = \langle \mathcal{L}, AP, S, s_0, R, \Theta \rangle$*, for the transition from state* $s_1 \in S$ *to* $s_2 \in S$*, the predicate* other*, the predicate* halt*, and the maximality* max *are defined as:*

$$\begin{aligned} \text{other}(s_1, s_2) &= \bigvee\nolimits_{o \in S \setminus \{s_2\}} R(s_1, o) \\ \text{halt}(s_1, s_2) &= \neg\text{other}(s_1, s_2) \wedge \neg R(s_1, s_2) \\ \max(s_1, s_2) &= \neg\text{other}(s_1, s_2) \vee \quad R(s_1, s_2) \end{aligned}$$

Maximality of a transition is defined by its own value, and the values of other transitions from the same state. Looking at the border cases, a *true* transition is always maximal, but the maximality of a *false* transition depends on the other

transitions. This ensures that halting the execution, by taking a *false* transition from the current state, depends on the degree to which the current state is a deadlock state. In addition, maximality is equal to the transition value if any other transition is *true*, or equal to *true* if all other transitions are *false*.

Including the value s_2 in the disjunction over $o \in S \backslash \{s_2\}$ in the definition of other results in an alternative definition for maximality: $\max'(s_1, s_2) = \max(s_1, s_2) \wedge (R(s_1, s_2) \vee \neg R(s_1, s_2))$. The definitions coincide for Boolean logic, but not for multi-valued logics without excluded middle. We assume $R(s_1, s_2) \vee \neg R(s_1, s_2) = \mathit{true}$ for each transitions of the Kripke model by using the original definition for maximality; otherwise, we would incorrectly test for excluded middle and fail for any transition value other than *true* or *false*.

We revise the definitions of our LTL operators to require maximality; this is comparable to requiring fair paths under a fairness condition. A property under fairness in the universal case requires a path to be not fair or uphold the property, while the existential case requires a path to be fair and uphold the property. We can similarly change our definitions to require maximality of transitions in addition to the original requirements.

In a multi-valued setting, maximality can not be evaluated separately from the LTL property for the path as a whole, but needs to be evaluated simultaneously with the LTL property for each individual transition. This is necessary, since a violation of a property at state s_n of a path, should only be influenced by the maximality of the path up to s_n. This requires us to choose between existential or universal quantification of our LTL formulas and modify our definitions accordingly to include maximality. In the following we assume existential quantification, since Büchi automata are used to search for counterexamples. Imposing maximality on top of Def. 7 gives us the following existential definitions:

Definition 9. *Given a path $\pi = s_1, s_2, ...$ in a multi-valued Kripke model $M = \langle \mathcal{L}, AP, S, s_0, R, \Theta \rangle$ such that $s_1, s_2, ... \in S$. The strong next and weak next have the following existential definitions:*

$$[\mathsf{X}_s\, \varphi]_1 = R(s_1, s_2) \wedge [\varphi]_2 \qquad [\mathsf{X}_w\, \varphi]_1 = \max(s_1, s_2) \wedge (\mathrm{halt}(s_1, s_2) \vee [\varphi]_2)$$

We can define the strong next operator analogous to weak next as $\max(s_1, s_2) \wedge (R(s_1, s_2) \wedge [\varphi]_2)$, but this reduces to the definition given above since $\max(s_1, s_2) \wedge R(s_1, s_2) = R(s_1, s_2)$. The strong next operator already works correctly in the existential case for *false* transitions. In the definition of the weak next operator, $\mathrm{halt}(s_1, s_2)$ replaces the $\neg R(s_1, s_2)$ of the original definition to prevent introducing a test for noncontradiction: $\max(s_1, s_2) \wedge \neg R(s_1, s_2) = \mathrm{halt}(s_1, s_2) \vee (R(s_1, s_2) \wedge \neg R(s_1, s_2))$. We assume noncontradiction for each transition of the Kripke model by taking $R(s_1, s_2) \wedge \neg R(s_1, s_2) = \mathit{false}$. Note that indeed $\max(s_1, s_2) \wedge \mathrm{halt}(s_1, s_2) = \mathrm{halt}(s_1, s_2)$.

4 Steering logic

We take a closer look at the nine-valued lattice of Fig. 1c. Our motivation for developing this logic and the theory of this paper is to investigate multi-valued

abstraction in the context of steerability: guiding the execution of a program to avoid bugs. This can for example be done by the scheduler of the operation system or by instrumenting the original program. Values of the nine-valued lattice should be interpreted as values indicating the steerability of transitions. They can be attached to transitions using quasi-Boolean guards in the modelling language: a quasi-Boolean expression that determines the transition value.

4.1 Semantics

The lattice of Fig. 1c can be used to encode steerability information in a model. Values of the lattice are effectively subsets of $\{t, f, T, F\}$ with N being the empty set, and A being the complete set. We use the convention that lowercase letters indicate truth under steering and uppercase letters indicate truth by default. Negation is defined as exchanging T with F and t with f in the subset. Keep in mind that while the subset construction is helpful to understand the semantics behind the truth values, the subsets are indivisible as truth values of the logic.

The intuition of the individual values is that T indicates a transition that is enabled by default: during execution it can be non-deterministically chosen to further the execution. A value t indicates a transition that can be enabled when controlling the execution: if we want this transition to be considered, we will have to influence the execution. Similarly F is a transition that is disabled by default, while f can be disabled when controlling the execution.

We could use all possible subsets of these base values to form a lattice, but we can reduce the number of values by adding a restriction: if a subset contains an uppercase value, then it also needs to contain the corresponding lowercase value. For example, we do not allow the value T, but do allow the value Tt. The reason for this restriction is that a transition that is enabled by default can be trivially enabled when controlled, simply by not exerting any influence.

To indicate a steerable transition we can also use the values tf, Ttf, and Ftf. They respectively indicate a transition that: can be enabled or disabled when controlled (tf); is enabled by default, but can be disabled when controlled (Ttf); and is disabled by default, but can be enabled when controlled (Ftf). Using these values in a multi-valued Kripke model enables us to detect how a property is influenced by the ability to steer an execution. For example, a property with the value tf can be enforced or broken using steering, while a value Ttf holds by default, but can be broken using steering.

4.2 Example

To demonstrate the necessity of maximality, we give an example using steering logic. In Fig. 2b we have a state space with multi-valued transitions: labeled transitions have the value as depicted, unlabelled transitions have the value *true*, and omitted transitions have the value *false*. The t transition can be enabled by steering, while the f transition can be disabled.

If we evaluate the existential property $\varphi \mathsf{R} \psi$ in the initial state, then we have two infinite paths without *false* transitions: one taking the t transition and the

other taking the f transition. Paths with *false* transitions are ignored a priori: without maximality they lead to incorrect results and with maximality they have no influence. For the remaining two paths, with or without maximality, $\varphi \,\mathsf{R}\, \psi$ is *false* after the t transition and *true* after the f transition.

To calculate $\varphi \,\mathsf{R}\, \psi$ without maximality we use Def. 6. The path over the t transition gives $[\varphi \,\mathsf{R}\, \psi]_1 = \mathit{true} \wedge (\mathit{false} \vee (\neg t \vee \mathit{false})) = f$. The path over the f transition gives $[\varphi \,\mathsf{R}\, \psi]_1 = \mathit{true} \wedge (\mathit{false} \vee (\neg f \vee \mathit{true})) = \mathit{true}$. The *true* result of the second path suggests that $\varphi \,\mathsf{R}\, \psi$ holds, irrespective of how we steer; but it forgoes that, with the f transition disabled, it is ignored in favour of any other transition, such as the t transition. The initial state is never a deadlock state: we are not allowed to halt the execution by disabling the f transition.

In comparison, to calculate $\varphi \,\mathsf{R}\, \psi$ with maximality we use Def. 9. The path over the t transition gives $[\varphi \mathsf{R} \psi]_1 = \mathit{true} \wedge (\mathit{false} \vee ((\neg f \vee t) \wedge ((\neg t \wedge \neg f) \vee \mathit{false}))) = f$. The path over the f transition gives $[\varphi \,\mathsf{R}\, \psi]_1 = \mathit{true} \wedge (\mathit{false} \vee ((\neg t \vee f) \wedge ((\neg t \wedge \neg f) \vee \mathit{true}))) = f$. The f result of the second path correctly models that by disabling the f transition we can steer to ignore this path in favour of others. The maximality in the path over the f transition correctly models the influence of the other transitions on our ability to halt the execution.

5 Creating Büchi automata

Explicit state LTL model checking verifies a property by searching for counterexamples in a state space: the product of a Kripke model describing the program, and a Büchi automaton encoding the negation of the property. Counterexamples are paths ending in an accepting cycle of the state space: cycles containing states that have been marked as accepting in the Büchi automaton.

The truth of a counterexample is the conjunction of its transition values, while multiple counterexamples can be combined using disjunction. The negation of this disjunction is the truth of the property. If no accepting cycles with truth larger than *false* are found, then the property is *true* for the model.

5.1 The algorithm

Our algorithm for generating multi-valued Büchi automata is an adaptation of the algorithm presented in [8]. It starts with a graph consisting of a single node containing a single proof obligation: the LTL formula under verification. Nodes in the graph are then iteratively expanded by creating new transitions to new nodes. The transitions contain requirements on atomic propositions of the current state, while the new nodes contain proof obligations for the next state.

Our algorithm differs from [8] in that we cannot use all of the transitions in the multi-valued Kripke model in their positive form: we also require support for calculating $\neg R(s_1, s_2)$ and $\neg \mathrm{other}(s_1, s_2)$. In addition we use the more conventional method of evaluating atomic propositions of the Kripke model using the transitions of the Büchi automaton, instead of its states.

The Kripke model is modified to include the atomic propositions r and o. After each transition from a state $s_1 \in S$ to a state $s_2 \in S$, the model ensures that $r = R(s_1, s_2)$ and $o = \text{other}(s_1, s_2)$. These propositions are then used to calculate $\max(s_1, s_2)$ and $\text{halt}(s_1, s_2)$. This can require duplication of the original state if there are multiple incoming transitions for which r and o do not agree. When interleaved with the Büchi automaton all transitions of the Kripke model are *true*; only r will be equal to $R(s_1, s_2)$.

Given a multi-valued Kripke model with atomic propositions AP and quasi-Boolean logic $\mathcal{L} = \langle L, \leq, \neg \rangle$, we want to create a Büchi automaton to verify whether a temporal property $\neg\varphi$ holds universally for the complete state space of the model. The algorithm creates a proof graph on the basis of which we can construct the Büchi automaton.

Definition 10. *A proof graph is a tuple* $\mathcal{G} = \langle N, n_0, T, R \rangle$ *with* $N : \mathcal{P}(\mathcal{P}(LTL))$ *a set of proof nodes,* n_0 *the initial node,* $T : \mathcal{P}(\mathcal{P}(LTL))$ *a set of proof transitions and* $R : N \times N \to T$ *a transition relation.*

Each node and transition is a set of proof obligations: a set of LTL formulas that need to be verified. Initially proof nodes are related using $\{false\}$ transitions. We can assume without loss of generality that all LTL formulas are in release positive normal form.

The algorithm starts by creating a single initial node $\{\varphi\}$ in the proof graph, with φ the counterexample we are searching for. This initial node will be made the current node, making it the first node up for expansion. (In the following t and f are variables and should not be considered as truth values.)

(a) Expand O_1 (b) Match f

Figure 3: Preliminaries

A node $n = O_1$ is expanded by creating an initial transition $t = O_1$ from n to a destination $d = \emptyset$, as shown in Fig. 3a. Starting with this initial transition, a transition t from a node n to a destination d is processed by removing an obligation $f \in t$ from t and executing the following rules by matching on the formula f. When this results in a split, a copy t' of t is created to a corresponding copy d' of d. Processing continues on t, or in case of a split on both t and t', until only literals (p or $\neg p$, with $p \in AP$) and truth values ($l \in L$) remain.

$\varphi \wedge \psi$	Add φ and ψ to t.
$\varphi \vee \psi$	Split t, add φ to t, add ψ to t'.
$\mathsf{X}_s\, \varphi$	Add r and φ to d.
$\mathsf{X}_w\, \varphi$	Split t, add $\neg r$ and $\neg o$ to d, add $r \vee \neg o$ and φ to d'.

$\varphi \,\mathsf{U}\, \psi$ Split t, add φ and $\mathsf{X}_s(\varphi \,\mathsf{U}\, \psi)$ to t, add ψ to t'.
$\varphi \,\mathsf{R}\, \psi$ Split t, add ψ and $\mathsf{X}_w(\varphi \,\mathsf{R}\, \psi)$ to t, add φ and ψ t'.

Applying the rules on an obligation f of transition t from O_1 to O_2, as depicted in Fig. 3b, will result in the transitions of Fig. 4. With regard to the temporal operators we effectively follow Def. 7 and 9 when put in disjunctive normal form.

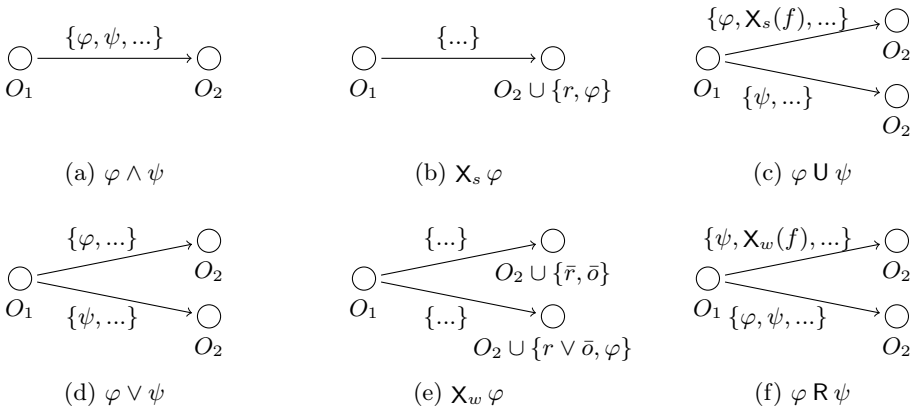


Figure 4: Splitting transitions

After processing the current node, the algorithm checks for optimisations. Transitions that are inconsistent in their proof obligations, such that their conjunction results in *false*, are removed. Truth values in a transition are combined into a single value using conjunction. Nodes with identical proof obligations are combined; and multiple transitions $t_1, ..., t_n$ between the same two nodes are replaced by a single transition $\{\bigwedge(t_1) \vee ... \vee \bigwedge(t_n)\}$.

Expansion continues in a depth-first manner by following one of the transitions to a new current node. Only nodes that have not been visited before are considered. The algorithm stops when all nodes have been visited.

5.2 The multi-valued Büchi automaton

Having constructed a proof graph, we can create the multi-valued Büchi automaton $\langle \mathcal{L}, \Sigma, Q, q_0, \delta, F \rangle$ required for interleaving with the multi-valued Kripke model $\langle \mathcal{L}, AP, S, s_0, R, \Theta \rangle$. We create a state q_k for each node n_k of the proof graph; the initial node n_0 corresponds to the initial Büchi state q_0.

Transitions between Büchi states correspond to transitions between proof states. The transition relation $\delta : Q \times \Sigma \times Q \to L$ of the Büchi automaton returns a conjunction of the proof obligations contained in the corresponding transition between proof nodes. In the case of multi-valued Kripke models we use $\Sigma = L^{AP}$,

such that $\delta : Q \times L^{AP} \times Q \to L$. We can therefore define the transition relation as $\delta(q_s, \sigma, q_t) = \sigma\left(\bigwedge(R(n_s, n_t))\right)$ with $\sigma : L^{AP}$ being used as a mapping from atomic propositions to truth values. (When applied to an expression, each occurrence of an atomic proposition is replaced by its corresponding truth value.)

The construction of the proof graph for an LTL property φ might suggest that any infinite path in the graph corresponds to a proof of φ. This is however not the case for the until operator, and the reason why accepting states of the Büchi automaton are significant. When evaluating $\varphi \,\mathsf{U}\, \psi$ in the Boolean setting, its definition requires that ψ becomes *true* at some point during the execution. For the multi-valued setting this means that in the disjunctive normal form of the until operator we do not consider the conjunct that is the infinite conjunction of φ. This is enforced in the Büchi automaton by creating an acceptance set F for each sub-formula of the form $\varphi \mathsf{U} \psi$, such that $q_k \in F$ iff $\varphi \mathsf{U} \psi \notin n_k$ or $\psi \in n_k$. An accepting run of the Büchi automaton should pass infinitely often through at least one member of each acceptance set.

Our definition of multi-valued Büchi automata only allows for one acceptance set. This limitation simplifies the requirements on the model checker looking for accepting loops in the combined state space. It is straightforward to convert a Büchi automaton with multiple acceptance sets to one with a single acceptance set, by putting multiple copies of the original Büchi automaton in sequence: make one copy for each acceptance set, and have transitions move from one copy to the next after reaching an accepting state for the current acceptance set. For further details see [3] , where generalised non-deterministic Büchi automata (GNBA) are transformed into non-deterministic Büchi automata (NBA).

Theorem 1. *The product state space of a modified Kripke model and a multi-valued Büchi automaton as described in section 5 encodes the given LTL property $\neg\varphi$ such that the disjunction of all counterexamples is the truth of φ.*

Proof (Sketch). This follows directly from Def. 7 and 9. The algorithm ensures that each accepting path through the Büchi automaton corresponds to a conjunct of φ in disjunctive normal form. Finite conjuncts correspond to paths diverging on an accepting state $\emptyset$ with a *true* self-loop. Only paths of conjuncts that require a $\gamma \,\mathsf{U}\, \psi$ but do not consider ψ at some point, are not accepting. □

The resulting multi-valued Büchi automaton can be used to verify the property φ over the multi-valued Kripke model when the transitions of the Kripke model in the interleaved state space are all valued as *true*. The actual transition value is derived from atomic propositions in the target Kripke state by the Büchi automaton. (Transitions of the Büchi automaton can be described as a conjunction of this transition value and an additional truth value derived from the original atomic propositions of the Kripke model.) These additional atomic propositions can however increase the size of the Kripke model.

We can opt for an implementation that slightly deviates from the normal definition of a Büchi automaton. Instead of encoding the transition values with atomic propositions in the target Kripke state, we can use atomic propositions of the Büchi automaton to signal the Kripke model what value we require for

its next transition. The Kripke model can then directly calculate these values from its state, without having to resort to additional bookkeeping. Changes to the size of the Büchi automaton are negligible, since the additional state of the atomic propositions is directly related to the original destination Büchi state. This alternate implementation might even result in a reduced size for the Büchi automaton when there are original states that only differ in their calculation of the transition value.

6 Future work

We are implementing the presented algorithm to facilitate multi-valued model checking in the (distributed) SpinJa model checker [6,12]. Together with the steering logic that was considered in Section 4, this will allow us to investigate execution steering based on abstract models [11], building on our previous implementation of multi-valued model checking [13,2].

References

1. J.O. Andrade and Y. Kameyama. Efficient multi-valued bounded model checking for LTL over quasi-boolean algebras. *IEICE Trans.*, 95-D(5):1355–1364, 2012.
2. R. Augustijn. Multivalued logics and hyper transitions in SpinJa. Master's thesis, Vrije Universiteit Amsterdam, 2015.
3. C. Baier and J.P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4. G. Bruns and P. Godefroid. Model checking with multi-valued logics. In *ICALP*, volume 3142 of *LNCS*, pages 281–293. Springer, 2004.
5. M. Chechik, B. Devereux, and A. Gurfinkel. Model-checking infinite state-space systems with fine-grained abstractions using spin. In *SPIN*, volume 2057 of *LNCS*, pages 16–36. Springer, 2001.
6. M. de Jonge and T.C. Ruys. The SpinJa model checker. In *SPIN*, volume 6349 of *LNCS*, pages 124–128. Springer, 2010.
7. M. Fitting. Bilattices and the theory of truth. *Journal of Philosophical Logic*, 18:225–256, 1989.
8. R. Gerth, D.A. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV*, volume 38 of *IFIP*, pages 3–18. Chapman & Hall, 1995.
9. O. Kupferman and Y. Lustig. Lattice automata. In *VMCAI*, volume 4349 of *LNCS*, pages 199–213. Springer, 2007.
10. Y. Meller, O. Grumberg, and S. Shoham. A framework for compositional verification of multi-valued systems via abstraction-refinement. In *ATVA*, volume 5799 of *LNCS*, pages 271–288. Springer, 2009.
11. S.J.J. Vijzelaar and W.J. Fokkink. Multi-valued simulation and abstraction using lattice operations. *ACM Trans. Embedded Comput. Syst.*, 16(2):42:1–42:26, 2017.
12. S.J.J. Vijzelaar, C. Verstoep, W.J. Fokkink, and H.E. Bal. Distributed MAP in the SpinJa model checker. In *PDMC*, volume 72 of *EPTCS*, pages 84–90, 2011.
13. S.J.J. Vijzelaar, C. Verstoep, W.J. Fokkink, and H.E. Bal. Bonsai: Cutting models down to size. In *PSI*, volume 8974 of *LNCS*, pages 361–375. Springer, 2014.