3vLTL: A Tool to Generate Automata for Three-valued LTL

Francesco Belardinelli
Imperial College, London, United Kingdom
francesco.belardinelli@imperial.ac.uk

Angelo Ferrando
University of Genoa, Genoa, Italy
angelo.ferrando@unige.it

Vadim Malvone
Telecom Paris, Paris, France
vadim.malvone@telecom-paris.fr

Multi-valued logics have a long tradition in the literature on system verification, including run-time verification. However, comparatively fewer model-checking tools have been developed for multi-valued specification languages. We present 3vLTL, a tool to generate Büchi automata from formulas in Linear-time Temporal Logic (LTL) interpreted on a three-valued semantics. Given an LTL formula, a set of atomic propositions as the alphabet for the automaton, and a truth value, our procedure generates a Büchi automaton that accepts all the words that assign the chosen truth value to the LTL formula. Given the particular type of the output of the tool, it can also be seamlessly processed by third-party libraries in a natural way. That is, the Büchi automaton can then be used in the context of formal verification to check whether an LTL formula is true, false, or undefined on a given model.

1 Introduction

Multi-valued logics have a long tradition in the literature on system verification, as demonstrated by various references [8, 15, 3, 23, 17, 18], and they play a crucial role in run-time verification as well [4, 5]. Of particular interest are three-valued logics, including temporal extensions of Kleene's logic [19], where the third value, in addition to true and false, is interpreted as "unknown" or "unspecified". Such semantics prove especially convenient when constructing smaller abstractions of complex reactive and distributed systems. These abstractions are typically approximations of the original model, containing strictly less information. Consequently, the challenge lies in finding the right trade-off between reducing complexity and minimizing information loss during the abstraction process. In system verification, one of the most widely used temporal logics for specifying requirements is Linear-time Temporal Logic (LTL) [22]. The model checking problem for LTL is typically addressed through automata-theoretic techniques [2]. Given a model M of a transition system and an LTL formula φ, the approach involves generating Büchi automata for both M and the negation of φ. This allows us to determine whether φ is satisfied in M by examining whether the language accepted by the product of these two automata is empty.

Several tools are available now to generate Büchi automata from LTL formulas. Notable examples include [11, 13]. However, to the best of our knowledge, no tool has yet been proposed to directly generate Büchi automata for multi-valued temporal logics.

Contribution. In this paper we present 3vLTL, a tool to generate (generalized) Büchi automata from LTL formulas interpreted on a three-valued semantics. Specifically, given an LTL formula, a set of atomic propositions (representing the automaton alphabet), and a truth value (true, false or undefined), our procedure generates a Büchi automaton that accepts all the words that assign the chosen truth value to the input LTL formula. Furthermore, 3vLTL has the functionality to process our output (i.e., the

automaton) by third-party libraries in a natural way. The present work is motivated by the use of three-valued logics in system verification. Indeed, our tool can be used in several works, such as [20, 16, 24, 25], to provide results for the verification of three-valued *LTL* formulas. Furthermore, our tool is already used in [6]. In this work, the authors present an abstraction-refinement method to partially solve the model checking of multi-agent systems under imperfect information and perfect recall strategies. Note that, a three-valued semantics becomes particularly significant in situations involving imperfect information, as the absence of information can lead to the emergence of a third value. This is particularly evident in autonomous and distributed systems, where a component may not have access to the complete system's information [12].

Related Work. Concerning the three-valued automata technique for *LTL* employed in this work, the most closely related approaches can be found in [21, 10, 9, 26]. Notably, in [9], there is an exploration of a reduction from multi-valued to two-valued *LTL*, but it does not encompass automata-theoretic techniques. Conversely, in [10], an automata-theoretic approach for general multi-valued *LTL* is presented, following the tableau-based construction as outlined in [14]; however, this work is more suitable for on-the-fly verification w.r.t. to our approach. In a different vein, [21] delves into general multi-valued automata, defining lattices, deterministic and non-deterministic automata, as well as their extensions with Büchi acceptance conditions. As part of their theoretical findings, they introduce an automata construction for multi-valued *LTL*, though it lacks a clear explanation of states and transitions. With respect to our work, in [21], the model checking is only briefly discussed, and their approach is tailored more toward multi-valued logics in a broader sense.

To summarize, unlike [21, 10, 9], our proposed approach makes minimal modifications to the automata-theoretic construction for two-valued *LTL* [2] and extends it to a three-valued interpretation.

2 Preliminaries

In this part we present a three-valued semantics for Linear-time Temporal Logic *LTL* and recall the definition of generalized non-deterministic Büchi automata. To fix the notation, we assume that $AP = \{q_1, q_2, \ldots\}$ is the set of *atomic propositions*, or simply atoms. We denote the length of a tuple t as $|t|$, and its i-th element as t_i. For $i \leq |t|$, let $t_{\geq i}$ be the suffix $t_i, \ldots, t_{|t|}$ of t starting at t_i and $t_{\leq i}$ its prefix $t_1, \ldots, t_i$. Notice that we start enumerations with index 1.

Models. We begin by giving a formal definition of Transition Model [2].

Definition 1 (Transition Model) *Given a set AP of atoms, a* Transition Model *is a tuple* $M = \langle S, s_0, \longrightarrow, V \rangle$ *such that (i) S is a finite, non-empty set of* states, *with* initial state $s_0 \in S$; *(ii)* $\longrightarrow \subseteq S \times S$ *is a serial* transition relation; *(iii)* $V : S \times AP \to \{\text{tt}, \text{ff}, \text{uu}\}$ *is the* three-valued labelling function.

A path $p \in S^\omega$ is an infinite sequence $s_1 s_2 s_3 \ldots$ of states where $s_i \longrightarrow s_{i+1}$, for each $i \geq 1$.

Syntax. Here, we recall the syntax of *LTL*.

Definition 2 (LTL) *Formulas in LTL are defined as follows, where* $q \in AP$:
$$\varphi \ ::= \ q \mid \neg\varphi \mid \varphi \land \varphi \mid X\varphi \mid (\varphi U \varphi)$$

The meaning of operators *next X* and *until U* is standard [2]. Operators *release R*, *finally F*, and *globally G* can be introduced as usual: $\varphi R\psi \equiv \neg(\neg\varphi U \neg\psi)$, $F\varphi \equiv \text{tt} U \varphi$, $G\varphi \equiv \text{ff} R\varphi$.

Semantics.   Formally we define the three-valued semantics for *LTL* as follows.

Definition 3 (Satisfaction) *The three-valued satisfaction relation $\models^3$ for a Transition Model M, path $p \in S^\omega$, atom $q \in AP$, $v \in \{\text{tt}, \text{ff}\}$, and formula φ is defined as follows:*

$$((M,s) \models^3 A\psi) = \text{ff} \quad \text{iff} \quad \text{for some path } p \text{ in } M, \ ((M,p) \models^3 \psi) = \text{ff}$$

$$((M,p) \models^3 q) = v \quad \text{iff} \quad V(p_1, q) = v$$

$$((M,p) \models^3 \neg\psi) = v \quad \text{iff} \quad ((M,p) \models^3 \psi) = \neg v$$

$$((M,p) \models^3 \psi \wedge \psi') = \text{tt} \quad \text{iff} \quad ((M,p) \models^3 \psi) = \text{tt} \text{ and } ((M,p) \models^3 \psi') = \text{tt}$$

$$((M,p) \models^3 \psi \wedge \psi') = \text{ff} \quad \text{iff} \quad ((M,p) \models^3 \psi) = \text{ff} \text{ or } ((M,p) \models^3 \psi') = \text{ff}$$

$$((M,p) \models^3 X\psi) = v \quad \text{iff} \quad ((M,p_{\geq 2}) \models^3 \psi) = v$$

$$((M,p) \models^3 \psi U\psi') = \text{tt} \quad \text{iff} \quad \text{for some } k \geq 1, \ ((M,p_{\geq k}) \models^3 \psi') = \text{tt}, \text{ and}$$
$$\text{for all } j, \ 1 \leq j < k \Rightarrow ((M,p_{\geq j}) \models^3 \psi) = \text{tt}$$

$$((M,p) \models^3 \psi U\psi') = \text{ff} \quad \text{iff} \quad \text{for all } k \geq 1, \ ((M,p_{\geq k}) \models^3 \psi') = \text{ff}, \text{ or}$$
$$\text{for some } j \geq 1, \ ((M,p_{\geq j}) \models^3 \psi) = \text{ff}, \text{ and}$$
$$\text{for all } j', \ 1 \leq j' \leq j \Rightarrow ((M,p_{\geq j'}) \models^3 \psi') = \text{ff}$$

In all other cases the value of φ is uu.

Generalized non-deterministic Büchi automaton.   Now, we recall the definition of the class of automota that we will use in our construction and in the tool.

Definition 4 (GNBA) *A generalized non-deterministic Büchi automaton is a tuple $A = \langle Q, Q_0, \Sigma, \pi, \mathscr{F} \rangle$ where (i) Q is a finite set of* states *with $Q_0 \subseteq Q$ as the set of* initial states*; (ii) Σ is an* alphabet*; (iii) $\pi : Q \times \Sigma \to 2^Q$ is the (non-deterministic)* transition function*; (iv) $\mathscr{F}$ is a (possibly empty) set of subsets of Q, whose elements are called* acceptance sets.

Given an infinite run $\rho = q_0 q_1 q_2 \ldots \in Q^\omega$, let $Inf(\rho)$ be the set of states q for which there are infinitely many indices i with $q = q_i$, that is, q appears infinitely often in ρ. Then, run ρ is *accepting* if for each acceptance set $F \in \mathscr{F}$, $Inf(\rho) \cap F \neq \emptyset$, that is, there are infinitely many indices i in ρ with $q_i \in F$. The *accepted language $L(A)$* of automaton A consists of all infinite words $w \in \Sigma^\omega$ for which there exists at least one accepting run $\rho = q_0 q_1 q_2 \ldots \in Q^\omega$ such that for all $i \geq 0$, $q_{i+1} \in \pi(q_i, w_i)$.

3   Automata Construction

In this section we provide a slightly variant of the automata-theoretic approach to the verification of the three-valued linear-time logic *LTL* as proposed in [7]. In particular, in what follows we generalize the construction in [7] for the truth values tt, ff, and uu.

Definition 5 (Closure and Elementarity) *The* closure $cl(\psi)$ *of an LTL formula ψ is the set consisting of all subformulas ϕ of ψ and their negation $\neg\phi$. A set $B \subseteq cl(\psi)$ is* consistent w.r.t. propositional logic *iff for all $\psi_1 \wedge \psi_2, \neg\phi \in cl(\psi)$: (i) $\psi_1 \wedge \psi_2 \in B$ iff $\psi_1 \in B$ and $\psi_2 \in B$; (ii) $\neg(\psi_1 \wedge \psi_2) \in B$ iff $\neg\psi_1 \in B$ or $\neg\psi_2 \in B$; (iii) if $\phi \in B$ then $\neg\phi \notin B$; (iv) $\neg\neg\phi \in B$ iff $\phi \in B$. Further, B is* locally consistent w.r.t. the until operator *iff for all $\psi_1 U \psi_2 \in cl(\psi)$: (i) if $\psi_2 \in B$ then $\psi_1 U \psi_2 \in B$; (ii) if $\neg(\psi_1 U \psi_2) \in B$ then $\neg\psi_2 \in B$; (iii) if $\psi_1 U \psi_2 \in B$ and $\psi_2 \notin B$ then $\psi_1 \in B$; (iv) if $\neg\psi_1, \neg\psi_2 \in B$, then $\neg(\psi_1 U \psi_2) \in B$.*

Finally, B is elementary *iff it is both consistent and locally consistent.*

Note that, unlike the standard construction for two-valued *LTL* [2], we do not require elementary sets to be maximal (i.e., either $\phi \in B$ or $\neg\phi \in B$), but we do require extra conditions (*ii*) and (*iv*) on consistency, and (*ii*) and (*iv*) on local consistency. These extra conditions can be derived in the classic, two-valued semantics, but need to be assumed as primitive here.

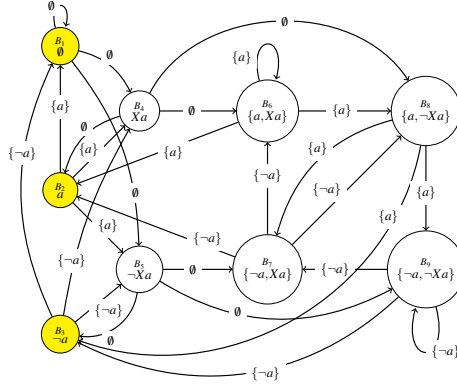Hereafter $Lit = AP \cup \{\neg q \mid q \in AP\}$ is the set of *literals*.

Figure 1: The automaton $A_{\psi,uu}$ for formula $\psi = Xa$. Initial states are marked in yellow.

Definition 6 (Automaton $A_{\psi,v}$) *Let ψ be a formula in LTL. We define the automaton $A_{\psi,v} = \langle Q, Q_0, 2^{Lit}, \pi, \mathscr{F} \rangle$, where $v \in \{tt, ff, uu\}$, as follows: Q is the set of all elementary sets $B \subseteq cl(\psi)$. if $v = tt$ then $Q_0 = \{B \in Q \mid \psi \in B\}$; else if $v = ff$ then $Q_0 = \{B \in Q \mid \neg\psi \in B\}$; otherwise $Q_0 = \{B \in Q \mid \psi \notin B$ and $\neg\psi \notin B\}$. The transition relation π is given by: let $A \subseteq Lit$. If $A \neq B \cap Lit$, then $\pi(B,A) = \emptyset$; otherwise $\pi(B,A)$ is the set of all elementary sets B' of formulas such that for every $X\phi, \psi_1 U \psi_2 \in cl(\psi)$: (i) $X\phi \in B$ iff $\phi \in B'$; (ii) $\neg X\phi \in B$ iff $\neg\phi \in B'$; (iii) $\psi_1 U \psi_2 \in B$ iff $\psi_2 \in B$ or, $\psi_1 \in B$ and $\psi_1 U \psi_2 \in B'$; (iv) $\neg(\psi_1 U \psi_2) \in B$ iff $\neg\psi_2 \in B$ and, $\neg\psi_1 \in B$ or $\neg(\psi_1 U \psi_2) \in B'$. $\mathscr{F} = \{F_{\psi_1 U \psi_2} \mid \psi_1 U \psi_2 \in cl(\psi)\} \cup \{Q\}$, where $F_{\psi_1 U \psi_2} = \{B \in Q \mid \psi_1 U \psi_2 \in B$ implies $\psi_2 \in B$ and $\neg\psi_2 \in B$ implies $\neg(\psi_1 U \psi_2) \in B\}$.*

According to Def. 6, the transition relation operates as follows: when the automaton reads a set A of literals that do not exist in the current state, the transition remains undefined. However, if these literals are present in the state, the automaton proceeds to verify the enabled transitions based on the semantics of the *LTL* operators. It is worth noting that in Def. 6, we must also specify conditions for negated formulas. This is necessary because elementary sets may not necessarily be maximal in this context.

We present an example of automaton for the next operator and truth value undefined.

Example 1 *Consider $\psi = Xa$. The GNBA $A_{\psi,uu}$ in Fig. 1 is obtained as indicated in Def. 6. Namely, the state space Q consists of all elementary sets of formulas contained in $cl(\psi) = \{a, \neg a, Xa, \neg Xa\}$: $B_1 = \emptyset, B_2 = \{a\}, B_3 = \{\neg a\}, B_4 = \{Xa\}, B_5 = \{\neg Xa\}, B_6 = \{a, Xa\}, B_7 = \{a, \neg Xa\}, B_8 = \{\neg a, Xa\}, B_9 = \{\neg a, \neg Xa\}$. The initial states of $A_{\psi,uu}$ are the elementary sets $B \in Q$ with $\psi, \neg\psi \notin B$. That is, $Q_0 = \{B_1, B_2, B_3\}$. The transitions are depicted in Fig. 1. The set $\mathscr{F}$ is $\{Q\}$ as ψ does not contain until operators. Hence, every infinite run in $A_{\psi,uu}$ is accepting.*

Now, we provide a generalization of the main theoretical result proved in [7].

Theorem 1 *For every LTL formula ψ and truth value $v \in \{tt, ff, uu\}$, there exists a GNBA $A_{\psi,v}$ (given as in Def. 6) s.t. $L(A_{\psi,v}) = Paths(\psi,v)$, where $Paths(\psi,v)$ is the set of paths $p \in (2^{Lit})^\omega$ such that $(p \models^3 \psi) = v$. Moreover, the size of $A_{\psi,v}$ is exponential in the size of ψ.*

4   Implementation

Tool Architecture.   The 3vLTL tool[1] developed for this paper aims at generating highly reusable generalized non-deterministic Büchi automata (GNBA) [7]. Hence, instead of generating only a graphical

---

[1] https://github.com/AngeloFerrando/3vLTL

result, 3vLTL produces a machine-readable file which can be easily parsed by third-party tools and libraries as well. From an engineering perspective, a pure graphical representation would help the final user to visualise the generated automaton, but it would not make it accessible for further evaluations.
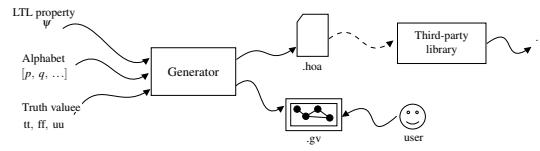

Figure 2: Overview of the tool.

Figure 2 provides an overview of 3vLTL. The tool begins by parsing the user's input, which consists of three essential arguments. The first argument is the LTL property ψ of interest, serving as a guide for generating the GNBA. The second argument represents the alphabet of ψ and informs 3vLTL about the atomic propositions to consider when constructing the automaton. Since the automaton explicitly specifies the atomic propositions associated with its transitions, it is crucial to identify the relevant events of interest. The third argument specifies the truth value against which the LTL formulas are verified. Following the approach proposed in [7], 3vLTL offers support for generating three different GNBA versions. To elaborate further, if tt (representing satisfaction), ff (representing violation), or uu (representing neither satisfaction nor violation) is provided as the third argument, 3vLTL generates the respective GNBA, denoted as $A_{\psi,\text{tt}}$, $A_{\psi,\text{ff}}$, or $A_{\psi,\text{uu}}$, recognizing traces that satisfy, violate, or neither satisfy nor violate ψ. 3vLTL produces two distinct output files. The first file, primarily graphical, contains the GNBA description in the DOT graph description language. The choice of DOT format stems from its widespread usage (supported by many programming languages) and its native compatibility with Graphviz[2].   The second file generated by 3vLTL adheres to the HOA (Hanoi Omega-Automata) format[3], a machine-readable format. This format enjoys support from well-known automata-based libraries, including Spot [11] and LTL3BA [1][4]. This choice enhances compatibility with third-party tools, promoting the broader utility of the GNBA generated by 3vLTL. It is important to note that while 3vLTL operates independently, it seamlessly integrates with existing automata-based tools, ensuring a smooth transition for users and enabling further advancements and applications of the GNBA it produces.

Technical details.   We go further into the detail of the implementation. First of all, 3vLTL has been implemented in Java (version 17). The resulting runnable jar can be directly used off-the-shelf.

3vLTL is divided into three components: input handler, automaton generator, and output handler.

Input handler. 3vLTL handles three input data: the LTL property ψ, the alphabet, and the truth value. While the handling of the second and third arguments is straightforward, the first argument requires a bit more of work. Specifically, a parser has been implemented to parse LTL formulas using Antlr[5], which is directly supported in Java. The resulting visitor for the LTL grammar is not only used to parse the LTL property ψ given in input, but it is also used to extract the corresponding closure $cl(\psi)$.

Automaton generator.   After the LTL property ψ given in input has been successfully parsed, and

---

[2]`https://graphviz.org/`

[3]`http://adl.github.io/hoaf/`

[4]LTL3BA operates on two-valued automata, but its output is defined using three truth values, allowing it to effectively handle run-time verification scenarios. For a comprehensive examination of the distinctions between "undefined" and "unknown" truth values in the context of run-time verification, you can find more detailed information in [12].

[5]`https://www.antlr.org`

the resulting closure $cl(\psi)$ has been generated, the tool proceeds with the generation of the GNBA. The corresponding Java object, instantiation of the custom *Automaton* class, is generated and stored in memory. Inside such object all information about states and transitions, along with details on the initial and accepting states, are stored. In particular, the set of initial states is determined by the last input given to 3vLTL. If the user desires to produce a GNBA to recognise the traces which satisfy ψ, then the initial states in the automaton are all the states containing ψ. Note that, this is possible because the elementary subsets of $cl(\psi)$ which determine the automaton's states are not necessarily maximal, differently form the standard automaton construction. Interestingly, the accepting states in all three cases are the same.

Output handler. Once the GNBA has been generated and the corresponding Java object is stored, 3vLTL moves forward to produce the resulting DOT and HOA output files. Both files are generated using two different custom methods of the *Automaton* class. Such methods pass through all states/transitions, and port these data in the wanted format.

Experiments. To show 3vLTL's scalability, we carried out some experiments w.r.t. the size of the LTL formula given in input. Figure 3 reports the results so obtained. As it is easy to note, the results show 3vLTL is exponential w.r.t. the size of LTL formula; where the size denotes the number of operators in the formula (e.g., the LTL formula XFp has size 2, while $Gp \wedge Fq$ has size 3). Note that, this was expected because the transformation from LTL to GNBA is known to be exponential w.r.t. the size of the LTL formula. So, 3vLTL extends the standard algorithm, but maintains the same complexity.
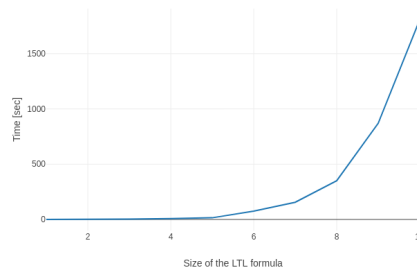

Figure 3: Experimental results.

5 Conclusions

In this paper, we have introduced a tool designed for generating automata from LTL formulas, interpreted within a three-valued semantics framework. To implement this tool, we have closely followed the automata construction methodology outlined in [7]. Looking ahead, our future work entails extending the capabilities of our tool to accommodate more than three truth values. This extension would enable us to create a generator capable of handling multi-valued LTL formulas. Additionally, we envision applying the automata construction and its associated implementation in various domains related to multi-valued logics. One such domain is Runtime Verification, where three-valued LTL also finds relevance. However, it is worth noting that the third value in Runtime Verification serves to maintain the impartiality of the monitor, while in our context, the third value signifies imperfect information about the system. As a result, our approach has the potential to address scenarios involving imperfect information, similar to the approach presented in [12]. Unfortunately, due to space constraints and the paper's primary focus, a comparative analysis with other tools has not been included.

References

[1] Tomáš Babiak, Mojmír Křetínský, Vojtěch Řehák & Jan Strejček (2012): *LTL to Büchi automata translation: Fast and more deterministic*. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, pp. 95–109. Available at `https://doi.org/10.48550/arXiv.1201.0682`.

[2] C. Baier & J. P. Katoen (2008): *Principles of Model Checking*. MIT Press.

[3] T. Ball & O. Kupferman (2006): *An abstraction-refinement framework for multi-agent systems*. In: *LICS06*, IEEE, pp. 379–388, doi:10.1109/LICS.2006.10.

[4] Andreas Bauer, Martin Leucker & Christian Schallhart (2006): *Monitoring of Real-Time Properties*. In S. Arun-Kumar & Naveen Garg, editors: *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, Lecture Notes in Computer Science 4337, Springer, pp. 260–272. Available at `https://doi.org/10.1007/11944836_25`.

[5] Andreas Bauer, Martin Leucker & Christian Schallhart (2007): *The Good, the Bad, and the Ugly, But How Ugly Is Ugly?* In Oleg Sokolsky & Serdar Taşıran, editors: *Runtime Verification*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 126–138, doi:10.1007/978-3-540-77395-5_11.

[6] Francesco Belardinelli, Angelo Ferrando & Vadim Malvone (2023): *An abstraction-refinement framework for verifying strategic properties in multi-agent systems with imperfect information*. Artif. Intell. 316. Available at `https://doi.org/10.1016/j.artint.2022.103847`.

[7] Francesco Belardinelli & Vadim Malvone (2020): *A Three-valued Approach to Strategic Abilities under Imperfect Information*. In Diego Calvanese, Esra Erdem & Michael Thielscher, editors: *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pp. 89–98. Available at `https://doi.org/10.24963/kr.2020/10`.

[8] G. Bruns & P. Godefroid (1999): *Model Checking Partial State Spaces*. In: *Proceedings of the 11th International Conference on Computer Aided Verification (CAV99)*, LNCS 1633, Springer-Verlag, pp. 274–287, doi:10.1007/3-540-48683-6_25.

[9] G. Bruns & P. Godefroid (2003): *Model Checking with Multi-Valued Logics*. Technical Report ITD-03-44535H, Bell Labs.

[10] Marsha Chechik, Benet Devereux & Arie Gurfinkel (2001): *Model-checking in finite state-space systems with fine-grained abstractions using SPIN*. In: *International SPIN Workshop on Model Checking of Software*, Springer, pp. 16–36, doi:10.1007/3-540-45139-0_3.

[11] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault & Laurent Xu (2016): *Spot 2.0 - A Framework for LTL and ω-Automata Manipulation*. In Cyrille Artho, Axel Legay & Doron Peled, editors: *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings*, Lecture Notes in Computer Science 9938, pp. 122–129. Available at `https://doi.org/10.1007/978-3-319-46520-3_8`.

[12] Angelo Ferrando & Vadim Malvone (2022): *Runtime Verification with Imperfect Information Through Indistinguishability Relations*. In Bernd-Holger Schlingloff & Ming Chai, editors: *Software Engineering and Formal Methods - 20th International Conference, SEFM 2022, Berlin, Germany, September 26-30, 2022, Proceedings*, Lecture Notes in Computer Science 13550, Springer, pp. 335–351. Available at `https://doi.org/10.1007/978-3-031-17108-6_21`.

[13] Paul Gastin & Denis Oddoux (2001): *Fast LTL to Büchi Automata Translation*. In Gérard Berry, Hubert Comon & Alain Finkel, editors: *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, Lecture Notes in Computer Science 2102, Springer, pp. 53–65. Available at `https://doi.org/10.1007/3-540-44585-4_6`.

[14] R. Gerth, D. Peled, M. Vardi & P. Wolper (1995): *Simple On-the-fly Automatic Verification of Linear Temporal Logic*. In: *Proceedings of IFIP/WG6.1 Symposium Protocol Specification, Testing and Verification (PSTV95)*, Chapman & Hall, pp. 3–18, doi:10.1007/978-0-387-34892-6_1.

[15] P. Godefroid & R. Jagadeesan (2003): *On the Expressiveness of 3-Valued Models*. In: *Proceedings of the 4th International Conference on Verification, Model Checkig, and Abstract Interpretation (VMCAI03)*, LNCS 2575, Springer-Verlag, pp. 206–222, doi:10.1007/3-540-36384-X_18.

[16] Patrice Godefroid & Nir Piterman (2009): *LTL generalized model checking revisited*. In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*, Springer, pp. 89–104. Available at `https://doi.org/10.1007/s10009-010-0169-3`.

[17] Michael Huth, Radha Jagadeesan & David A. Schmidt (2004): *A domain equation for refinement of partial systems*. Mathematical Structures in Computer Science 14(4), pp. 469–505. Available at `https://doi.org/10.1017/S0960129504004268`.

[18] Michael Huth & Shekhar Pradhan (2004): *Consistent Partial Model Checking*. Electronic Notes in Theoretical Computer Science 73, pp. 45–85. Available at `https://doi.org/10.1016/j.entcs.2004.08.003`.

[19] S. C. Kleene (1952): *Introduction to Metamathematics*. North-Holland.

[20] Beata Konikowska (1998): *A three-valued linear temporal logic for reasoning about concurrency*. ICS PAC, Warsaw, Poland, Tech. Rep.

[21] Orna Kupferman & Yoad Lustig (2007): *Lattice Automata*. In: *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*, pp. 199–213. Available at `https://doi.org/10.1007/978-3-540-69738-1_14`.

[22] A. Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS'77*, IEEE Computer Society, pp. 46–57, doi:10.1109/SFCS.1977.32.

[23] S. Shoham & O. Grumberg (2004): *Monotonic Abstraction-Refinement for CTL*. In: *TACAS04*, pp. 546–560, doi:10.1007/978-3-540-24730-2_40.

[24] Nils Timm & Stefan Gruner (2016): *Parameterised three-valued model checking*. Science of Computer Programming 126, pp. 94–110, doi:10.1016/j.scico.2016.01.006.

[25] Rachel Tzoref & Orna Grumberg (2006): *Automatic refinement and vacuity detection for symbolic trajectory evaluation*. In: *International Conference on Computer Aided Verification*, Springer, pp. 190–204, doi:10.1007/11817963_20.

[26] Stefan J. J. Vijzelaar & Wan J. Fokkink (2017): *Creating Büchi Automata for Multi-valued Model Checking*. In Ahmed Bouajjani & Alexandra Silva, editors: *Formal Techniques for Distributed Objects, Components, and Systems - 37th IFIP WG 6.1 International Conference, FORTE 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DiSCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings*, Lecture Notes in Computer Science 10321, Springer, pp. 210–224. Available at `https://doi.org/10.1007/978-3-319-60225-7_15`.