Deciding S1S: Down the Rabbit Hole and Through the Looking Glass

Vojtěch Havlena, Ondřej Lengál, and Barbora Šmahlíková

Faculty of Information Technology, Brno University of Technology, Czech Republic

Abstract. Monadic second-order logic of one successor (S1S) is a logic for specifying ω-regular languages in a concise way. In this paper, we revisit the classical decision procedure based on translating S1S formulae into Büchi automata and employ state-of-the-art algorithms for their manipulation, in particular complementation and size reduction. We compare our implementation to the one based on loop-deterministic finite automata and observe cases where the classical approach scales better.

1 Introduction

The study of formalisms allowing reasoning about ω-regular languages still attracts a lot of attention. For instance, ω-regular languages are often used for specifying properties of reactive systems via the formalisms of linear-time temporal logics such as LTL [1] or QPTL [2]. In addition to that, ω-regular languages have also been used for formal verification of programs [3] and, recently, in the context of automated theorem proving, for reasoning about properties of Sturmian words [4, 5]. A prominent logic allowing to describe the whole class of ω-regular properties is *monadic second-order logic of one successor* (S1S). The decidability of S1S was proven by Büchi in 1962 by introducing a connection of the logic with automata over infinite words called Büchi automata (BAs) [6]. S1S offers immense succinctness for the price of nonelementary worst-case complexity.

The many applications of ω-regular languages, often represented using BAs, together with BAs' nice theoretical properties have attracted a lot of attention towards developing efficient algorithms for their manipulation. Unlike the ones for automata over finite words, algorithms for BAs are often much more involved. In particular, the problem of efficiently complementing BAs has been approached from several sides [2, 7–29] and so has been the problem of BA reduction [30–33].

In this paper we revisit the original automata-based decision procedure for S1S and exploit state-of-the-art approaches for handling BAs, in particular approaches for their reduction and techniques of complementation, to obtain an efficient decision procedure. We summarize our observations with the implementation, identify the bottlenecks, and provide an experimental comparison with an approach deciding S1S based on deterministic-loop automata [34].

2 Preliminaries

Functions, words, and alphabets. We use ω to denote the first infinite ordinal $\omega = \{0, 1, \ldots\}$. An (infinite) word α over alphabet Σ is represented as a function $\alpha\colon \omega \to \Sigma$ where the i-th symbol is denoted as α_i. We abuse notation and sometimes also represent α as an infinite sequence $\alpha = \alpha_0\alpha_1 \ldots$ We use Σ^ω to denote the set of all infinite words over Σ.

Büchi automata. A (nondeterministic) *Büchi automaton* (BA) over Σ is a quadruple $\mathcal{A} = (Q, \delta, I, F)$ where Q is a finite set of *states*, δ is a *transition function* $\delta\colon Q \times \Sigma \to 2^Q$, and $I, F \subseteq Q$ are the sets of *initial* and *accepting* states respectively. We sometimes treat δ as a set of transitions of the form $p \xrightarrow{a} q$, for instance, we use $p \xrightarrow{a} q \in \delta$ to denote that $q \in \delta(p, a)$. A *run* of $\mathcal{A}$ from $q \in Q$ on an input word α is an infinite sequence $\rho\colon \omega \to Q$ that starts in q and respects δ, i.e., $\rho(0) = q$ and $\forall i \geq 0\colon \rho(i) \xrightarrow{\alpha_i} \rho(i+1) \in \delta$. Let $\inf(\rho)$ denote the states occurring in ρ infinitely often. We say that ρ is *accepting* iff $\inf(\rho) \cap F \neq \emptyset$. A word α is accepted by $\mathcal{A}$ if there is an accepting run ρ of $\mathcal{A}$ from some initial state, i.e., $\rho(0) \in I$. The set $\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}$ is called the *language* of $\mathcal{A}$.

Simulation. The *(maximum) direct simulation* on $\mathcal{A}$ is the relation $\preceq_{di} \subseteq Q \times Q$ defined as the largest relation s.t. $p \preceq_{di} q$ implies (i) $p \in F \Rightarrow q \in F$ and (ii) $p \xrightarrow{a} p' \in \delta \Rightarrow \exists q' \in Q\colon q \xrightarrow{a} q' \in \delta \wedge p' \preceq_{di} q'$ for each $a \in \Sigma$.

3 Monadic Second-order Logic of One Successor (S1S)

In this section we briefly introduce monadic second-order logic of one successor, denoted as S1S, used for expressing ω-regular properties of linear structures.

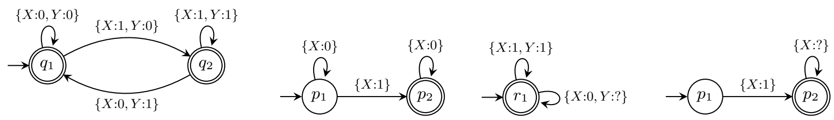
3.1 Syntax and Semantics

In this paper we build S1S formulae from *atomic formulae* of the form (i) $0 \in X$, (ii) $X \subseteq Y$, (iii) $X = \mathit{Succ}(Y)$, and (iv) $\mathit{Sing}(X)$ where X and Y are *second-order* variables. Formulae are then obtained as a Boolean combination of atomic formulae and existential quantification. Other connectives and universal quantification can be obtained as a syntactic sugar, e.g., we can define $\varphi \to \psi$ to denote $\neg\varphi \vee \psi$ and $\forall X.\, \varphi$ to denote $\neg\exists X.\, \neg\varphi$.

S1S formulae are interpreted over the set of natural numbers. In particular, second-order variables range over (possibly infinite) subsets of ω. For an S1S formula $\varphi(\mathbb{X})$ with free variables $\mathbb{X}$ an *assignment* is a mapping $\sigma\colon \mathbb{X} \to 2^\omega$. The *satisfaction* of an atomic formula φ by an assignment σ, denoted as $\sigma \models \varphi$, is inductively defined as follows: (i) $\sigma \models 0 \in X$ iff 0 is in $\sigma(X)$, (ii) $\sigma \models X \subseteq Y$ iff $\sigma(X)$ is a subset of $\sigma(Y)$, (iii) $\sigma \models X = \mathit{Succ}(Y)$ iff $\sigma(X) = \{y + 1 \mid y \in \sigma(Y)\}$, and (iv) $\sigma \models \mathit{Sing}(X)$ iff $|X| = 1$. Satisfaction of an S1S formula by σ is then defined inductively as usual. Formula φ is called *satisfiable* if there is an assignment σ such that $\sigma \models \varphi$.

3.2 Encoding Models as Words

The first step towards automata-based decision procedure is an encoding of assignments as words. In the following, we fix a formula φ with free variables $\mathbb{X}$. A symbol ξ over $\mathbb{X}$ is a mapping $\xi\colon \mathbb{X} \to \{0, 1\}$, e.g., $\xi = \{X{:}0, Y{:}1\}$. We use $\Sigma_{\mathbb{X}}$ to denote the set of all symbols over $\mathbb{X}$. Furthermore, for a set of variables $\mathbb{Y}$, we define the *projection* of ξ wrt. $\mathbb{Y}$ as $\pi_{\mathbb{Y}}(\xi) = \xi_{|\mathbb{X}\setminus\mathbb{Y}}$. An assignment σ of φ is then encoded as the word α^σ over $\Sigma_{\mathbb{X}}$ s.t. for each variable $X \in \mathbb{X}$ and for each $i \in \omega$ the following two conditions hold (i) if $i \in \sigma(X)$ then α^σ_i contains $X{:}1$ and (ii) if $i \notin \sigma(X)$ then α^σ_i contains $X{:}0$. The language of the formula φ is then defined as $\mathcal{L}(\varphi) = \{\alpha^\sigma \mid \sigma \text{ is a model of } \varphi\}$.



(a) $Y = Succ(X)$ (b) $Sing(X)$ (c) $X \subseteq Y$ (d) $0 \in X$

Fig. 1: BAs for atomic formulae.

3.3 Automata-based Decision Procedure

The automata-based decision procedure for S1S takes an input formula φ and inductively builds the BA $\mathcal{A}_\varphi$ accepting the same language as φ. Checking satisfiability of φ is then equivalent to testing whether $\mathcal{L}(\mathcal{A}_\varphi) \neq \emptyset$. The automaton $\mathcal{A}_\varphi$ is defined as follows: (i) If φ is an atomic formula, then $\mathcal{A}_\varphi$ is a predefined BA (see Fig. 1). (ii) If $\varphi = \psi_1 \wedge \psi_2$, then, in the first step, both $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_1}$ are adjusted to accept the original models extended to symbols over $\Sigma_{\mathbb{X}_1 \cup \mathbb{X}_2}$ ($\mathbb{X}_1$ and $\mathbb{X}_2$ are the free variables of ψ_1 and ψ_2 respectively). This step is called *cylindrification* and can be implemented by modifying the transition functions of $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_1}$. In particular, for $\mathcal{A}_{\psi_1}$, each transition over a symbol ξ is replaced by multiple transitions over all symbols $\xi' \in \Sigma_{\mathbb{X}_1 \cup \mathbb{X}_2}$ s.t. $\pi_{\mathbb{X}_1}(\xi') = \xi$. The BA $\mathcal{A}_\varphi$ is then obtained as $\mathcal{A}_\varphi = \mathcal{A}'_{\psi_1} \cap \mathcal{A}'_{\psi_2}$ where $\mathcal{A}'_{\psi_1}$ and $\mathcal{A}'_{\psi_2}$ are cylindrified BAs and $\cap$ is the standard operation of intersection of two BAs. (iii) If $\varphi = \psi_1 \vee \psi_2$ then $\mathcal{A}_\varphi = \mathcal{A}'_{\psi_1} \cup \mathcal{A}'_{\psi_2}$ where $\mathcal{A}'_{\psi_1}$ and $\mathcal{A}'_{\psi_2}$ are cylindrified BAs and $\cup$ is the standard operation of union over BAs. (iv) If $\varphi = \neg\psi$, then $\mathcal{A}_\varphi = \mathcal{A}_\psi^{\complement}$ where $\mathcal{A}^{\complement}$ denotes the BA accepting $\Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$. (v) If $\varphi = \exists X.\, \psi$, then $\mathcal{A}_\varphi = \pi_X(\mathcal{A}_\psi)$ where $\pi_X(\mathcal{A})$ is the BA obtained from $\mathcal{A}$ by modifying its transition function, applying $\pi_{\{X\}}$ on the symbol of each transition.

Handling of first-order variables. Although the definition of S1S presented in Section 3.1 uses second-order variables only, first-order variables (denoted by lowercase letters) can be handled using the support of the *Sing* predicate as follows: $\exists x.\, \varphi$ is transformed into $\exists X.\, \varphi \wedge Sing(X)$ and $\forall x.\, \varphi$ is transformed into $\forall X.\, Sing(X) \rightarrow \varphi$.

4 Implementation of the Decision Procedure

In this section, we focus on details related to our prototype implementation of the S1S decision procedure. The implemented tool, called ALICE, is written in PYTHON and is publicly available on GITHUB[1].

Automata-based decision procedure. ALICE implements the classical decision procedure as described in Section 3.3. In particular, it uses the two-copy product construction for intersection and performs union by simply uniting the input automata (making sure they have disjoint sets of states). BA complementation (corresponding to using negation in the input formula) is performed either by

[1] `https://github.com/barbora4/projektova-praxe`

Schewe's optimal construction [10, Section 3] improving the original rank-based construction [11,23] or by determinization-based complementation implemented within Spot [35]. Although the used complementation algorithms meet the lower bound of BA complementation $2^{\mathcal{O}(n \log n)}$, the complexity is still a bottleneck of the decision procedure. Note that development of efficient complementation algorithms for BAs is still a hot topic of current research [8,13]. In order to avoid the state explosion during complementation, we keep the automata as small as possible using (i) *lightweight reductions*, such as quotienting wrt. the direct simulation equivalence, i.e., two states p, q are merged if $p \preceq_{di} q$ and $q \preceq_{di} p$, or disconnecting little brother states [33], i.e., if there are transitions $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$ with $q \preceq_{di} r$, we can remove the transition $p \xrightarrow{a} q$ from the automaton, and (ii) *heavyweight reductions*, based on a 10-step lookahead simulation relation combined with advanced transition pruning, implemented in the tool Rabit [30].

Alphabet handling. When working with BAs, the number of states is not the only issue. Recall from Section 3.2 that if we consider a formula with n free variables, there are 2^n symbols that can occur in the corresponding automaton. For this reason we implement *symbolic* handling of symbols using a "don't care" flag (denoted by "?"). For instance two transitions $p \xrightarrow{\xi_1} q$ and $p \xrightarrow{\xi_2} q$ where $\xi_1 = \{X{:}1, Y{:}0\}$ and $\xi_2 = \{X{:}1, Y{:}1\}$ are represented by a single transition $p \xrightarrow{\kappa} q$ where $\kappa = \{X{:}1, Y{:}?\}$. In future, we might consider handling alphabets via *binary decision diagrams* in the similar way as Mona [36].

5 Experimental Evaluation

In this section, we compare our tool with, to the best of our knowledge, the only other existing implementation of a decision procedure for S1S, which is based on loop-deterministic finite automata (denoted as L-DFA) [34]. The evaluation uses a benchmark that consists of 26 hand-crafted S1S formulae obtained from [34]. We compared the approaches with respect to the number of states of the automaton $\mathcal{A}_\varphi$ (either BA or L-DFA) corresponding to the formula φ.[2]

In the comparison, we use the following three settings of our tool: Alice-Rank denotes the setting with Schewe's complementation and reduction by Rabit, Alice-Spot denotes Spot's complementation and reduction by Rabit, and, lastly, Alice-Spot-Light denotes Spot's complementation and lightweight reduction. The timeout (TO) was set to 1 hour. Selected results are in Table 1.

Discussion. Our tool usually gives better results than L-DFA in terms of state count, as shown in Table 1. In particular, for the case of Alice-Spot, the state counts of the resulting automata were in the vast majority of cases lower than for L-DFA. There were only two worse cases, one of them being formula 23 that did not finish in a day (complementing an automaton having 33 states). Furthermore, parametric formulae 18–20 are worth noticing; the number of states of L-DFA

[2] We do not compare other measurements such as the execution time or the sum of sizes of all automata obtained during the construction of $\mathcal{A}_\varphi$, because we were not able to obtain the L-DFA tool and [34] does not provide these values.

Table 1: Comparison of ALICE and L-DFA on S1S formulae. In addition to the atomic formulae from Section 3.1, ALICE also considers $x < y$ to be atomic.

	Formula	ALICE-RANK	ALICE-SPOT-LIGHT	ALICE-SPOT	L-DFA
1.	$(x \in Y \land x \notin Z) \lor (x \in Z \land x \notin Y)$	2	5	2	9
2.	$\neg\exists x.((x \in Y \land x \notin Z) \lor (x \in Z \land x \notin Y))$	1	1	1	9
3.	$\mathit{after}(X,Y) := \forall x.(x \in X \to \exists y.(y > x \land y \in Y))$	5	3	3	9
4.	$\mathit{fair}(X,Y) := \mathit{after}(X,Y) \land \mathit{after}(Y,X)$	24	5	5	9
5.	$\forall X.(\mathit{fair}(X,Y) \to \mathit{fair}(Y,Z))$	OOM	29	21	14
6.	$\mathit{suc}(x,y) := x < y \land \forall z.(\neg x < z \lor \neg z < y))$	3	4	3	10
18.	$\mathit{offset}(X,Y) := \forall i \forall j.(\mathit{suc}(i,j) \land i \in X \to j \in Y)$	2	2	2	11
19.	$\mathit{offset}(X,Y) \land \mathit{offset}(Y,Z) \land \mathit{offset}(Z,X)$	8	8	8	107
20.	$\mathit{offset}(V,W) \land \mathit{offset}(W,X) \land \mathit{offset}(X,Y) \land \mathit{offset}(Y,Z) \land \mathit{offset}(Z,V)$	32	32	32	2331
22.	$\mathit{insm}(i,j,U,V,W) := (j \in U \to i \in V \lor i \in W)$	8	13	8	15
23.	$\forall i \forall j(\mathit{suc}(i,j) \to \mathit{insm}(i,j,U,V,Z) \land \mathit{insm}(i,j,V,X,Y) \land \mathit{insm}(i,j,X,Y,V) \land \mathit{insm}(i,j,Y,Z,X) \land \mathit{insm}(i,j,Z,U,Y))$	OOM	TO	TO	198
26.	$\forall x \forall y.(x < y \land y \in X \land y \in Y) \land \forall x \forall y.(x < y \land y \in X \land y \notin Y) \land \forall x \forall y.(x < y \land y \notin X \land y \in Y) \land \forall x \forall y.(x < y \land y \notin X \land y \notin Y)$	21	11	11	18

grows much faster than in our case. If we compare the variants ALICE-RANK and ALICE-SPOT, the setting ALICE-SPOT gives overall better results—e.g., for formula 5, ALICE-RANK ran out of memory (OOM), yet ALICE-SPOT yields an automaton having 21 states. On the other hand, lightweight reduction behaves surprisingly well: ALICE-SPOT outperforms ALICE-SPOT-LIGHT just in 7 cases (most significantly on formulae 7, 9, and 22).

By analyzing the results, we found that the bottleneck of our approach is indeed BA complementation—it caused all the TOs and OOMs in the benchmark. For instance the TO in result of ALICE-SPOT for formula 23 is caused by complementing a BA with 33 states. To keep the sizes of automata small, their reduction is a crucial operation. Therefore, as a future work, we would like to investigate state-of-the-art techniques for BA complementation and identify the most suitable approach in connection with advanced minimization techniques. Although ALICE often produces smaller automata than L-DFA, the number of states is not the only possible measure: with a missing run time the comparison is incomplete, since dealing with BAs is usually harder than dealing with DFAs.

As far as we know, ALICE is the only off-the-shelf publicly available S1S solver. We intend to use it in the following settings: (i) educational (students input S1S formulae and observe the corresponding BAs) and (ii) research (we wish to study the structure of the created BAs and search for potential heuristics).

Acknowledgment. This work has been supported by the Czech Science Foundation project 19-24397S and the FIT BUT internal project FIT-S-20-6427.

References

1. Vardi, M.Y., Wolper, P.: An Automata-Theoretic Approach to Automatic Program Verification. In: Proceedings of the First Symposium on Logic in Computer Science, IEEE (1986) 322–331
2. Sistla, P.A., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. In: Automata, Languages and Programming, Springer (1985) 465–474
3. Heizmann, M., Hoenicke, J., Podelski, A.: Termination analysis by learning terminating programs. In: Proc. of CAV'14, Springer (2014) 797–813
4. Oei, R., Ma, D., Schulz, C., Hieronymi, P.: Pecan: An automated theorem prover for automatic sequences using büchi automata. CoRR **abs/2102.01727** (2021)
5. Hieronymi, P., Ma, D., Oei, R., Schaeffer, L., Schulz, C., Shallit, J.O.: Decidability for Sturmian words. CoRR **abs/2102.08207** (2021)
6. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proc. of International Congress on Logic, Method, and Philosophy of Science 1960, Stanford Univ. Press, Stanford (1962)
7. Gurumurthy, S., Kupferman, O., Somenzi, F., Vardi, M.Y.: On complementing nondeterministic Büchi automata. In Geist, D., Tronci, E., eds.: Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, L'Aquila, Italy, October 21-24, 2003, Proceedings. Volume 2860 of Lecture Notes in Computer Science., Springer (2003) 96–110
8. Chen, Y., Havlena, V., Lengál, O.: Simulations in rank-based Büchi automata complementation. In Lin, A.W., ed.: Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings. Volume 11893 of Lecture Notes in Computer Science., Springer (2019) 447–467
9. Chen, Y., Heizmann, M., Lengál, O., Li, Y., Tsai, M., Turrini, A., Zhang, L.: Advanced automata-based algorithms for program termination checking. In Foster, J.S., Grossman, D., eds.: Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018, ACM (2018) 135–150
10. Schewe, S.: Büchi complementation made tight. In Albers, S., Marion, J., eds.: 26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings. Volume 3 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2009) 661–672
11. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Log. **2**(3) (2001) 408–429
12. Allred, J.D., Ultes-Nitsche, U.: A simple and optimal complementation algorithm for Büchi automata. In: Proceedings of the Thirty third Annual IEEE Symposium on Logic in Computer Science (LICS 2018), IEEE Computer Society Press (July 2018) 46–55
13. Blahoudek, F., Duret-Lutz, A., Strejček, J.: Seminator 2 can complement generalized Büchi automata via improved semi-determinization. In: Proceedings of the 32nd International Conference on Computer-Aided Verification (CAV'20). Volume 12225 of Lecture Notes in Computer Science., Springer (July 2020) 15–27
14. Tsai, M.H., Fogarty, S., Vardi, M.Y., Tsay, Y.K.: State of Büchi complementation. In Domaratzki, M., Salomaa, K., eds.: Implementation and Application of Automata, Berlin, Heidelberg, Springer Berlin Heidelberg (2011) 261–271

15. Li, Y., Turrini, A., Zhang, L., Schewe, S.: Learning to complement Büchi automata. In: Proc. of VMCAI'18, Springer (2018) 313–335
16. Vardi, M.Y., Wilke, T., Kupferman, O., Fogarty, S.J.: Unifying Büchi Complementation Constructions. Logical Methods in Computer Science **9** (2013)
17. Blahoudek, F., Heizmann, M., Schewe, S., Strejcek, J., Tsai, M.: Complementing semi-deterministic Büchi automata. In Chechik, M., Raskin, J., eds.: Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings. Volume 9636 of Lecture Notes in Computer Science., Springer (2016) 770–787
18. Sistla, A.P., Vardi, M.Y., Wolper, P.: The Complementation Problem for Büchi Automata with Applications to Temporal Logic. Theoretical Computer Science **49**(2-3) (1987) 217–237
19. Fogarty, S., Vardi, M.Y.: Büchi complementation and size-change termination. In: Proc. of TACAS'09, Springer (2009) 16–30
20. Fogarty, S., Vardi, M.Y.: Efficient Büchi Universality Checking. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer (2010) 205–220
21. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proc. of LICS'06, IEEE (2006) 255–264
22. Kähler, D., Wilke, T.: Complementation, disambiguation, and determinization of Büchi automata unified. In: Proc. of ICALP'08, Springer (2008) 724–735
23. Friedgut, E., Kupferman, O., Vardi, M.: Büchi complementation made tighter. International Journal of Foundations of Computer Science **17** (2006) 851–868
24. Vardi, M.Y.: The Büchi complementation saga. In: Proc. of STACS'07, Springer (2007) 12–22
25. Breuers, S., Löding, C., Olschewski, J.: Improved Ramsey-based Büchi complementation. In: Proc. of FOSSACS'12, Springer (2012) 150–164
26. Li, Y., Vardi, M.Y., Zhang, L.: On the power of unambiguity in Büchi complementation. In Raskin, J.F., Bresolin, D., eds.: Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, Brussels, Belgium, September 21-22, 2020. Volume 326 of Electronic Proceedings in Theoretical Computer Science., Open Publishing Association (2020) 182–198
27. Kurshan, R.P.: Complementing deterministic Büchi automata in polynomial time. J. Comput. Syst. Sci. **35**(1) (1987) 59–71
28. Havlena, V., Lengál, O.: Reducing (to) the ranks: Efficient rank-based Büchi automata complementation (technical report). Technical report (2020) `https://arxiv.org/abs/2010.07834`.
29. Karmarkar, H., Chakraborty, S.: On minimal odd rankings for Büchi complementation. In Liu, Z., Ravn, A.P., eds.: Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings. Volume 5799 of Lecture Notes in Computer Science., Springer (2009) 228–243
30. Abdulla, P.A., Chen, Y., Clemente, L., Holík, L., Hong, C., Mayr, R., Vojnar, T.: Advanced Ramsey-based Büchi automata inclusion testing. In: Proc. of CONCUR'11, Springer (2011) 187–202
31. Etessami, K.: A hierarchy of polynomial-time computable simulations for automata. In: Proc. of CONCUR'02, Springer (2002) 131–144
32. Gurumurthy, S., Bloem, R., Somenzi, F.: Fair simulation minimization. In: Proc. of CAV'02, Springer (2002) 610–623

33. Bustan, D., Grumberg, O.: Simulation based minimization. ACM Transactions on Computational Logic **4** (03 2000)
34. Barth, S.: Deciding monadic second order logic over ω-words by specialized finite automata. In Ábrahám, E., Huisman, M., eds.: Integrated Formal Methods, Cham, Springer International Publishing (2016) 245–259
35. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, É., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In Artho, C., Legay, A., Peled, D., eds.: Automated Technology for Verification and Analysis, Cham, Springer International Publishing (2016) 122–129
36. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA implementation secrets. Int. J. Found. Comput. Sci. **13**(4) (2002) 571–586