BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2021
http://excel.fit.vutbr.cz

Deciding S1S Using Büchi Automata

Barbora Šmahlíková*

Abstract
Automata on infinite words were introduced as a tool for proving the decidability of problems in various logics. They enable finite representation of a set of infinite words with a regular structure. The aim of this work is to show the process of translating a formula in monadic second-order logic of one successor (S1S) to a corresponding Büchi automaton and deciding its satisfiability/validity. We then compare our implementation with the implementation using loop deterministic finite automata for various formulae. The efficiency is measured mainly by the state count of the final automaton for each formula. Our implementation gives better results in the overall majority of the tested formulae in terms of the number of states of the final automaton.

Keywords: S1S — Büchi automata — decision procedure

Supplementary Material: Downloadable code

*xsmahl00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Monadic second-order logic of one successor (S1S) is of interest in different applications, mostly in formal verification. The verification algorithm relies on a translation of a formula into an equivalent automaton. In this paper, we describe a decision procedure for S1S that performs a translation of an S1S formula into a Büchi automaton and analyse it in order to establish satisfiability/validity of the original formula. Büchi automata were introduced during the 1960s in order to develop a decision procedure for S1S [1]. Büchi automata characterize the entire class of ω-regular languages, i.e., languages containing infinite words with a regular structure. Moreover, every S1S-definable language is Büchi recognizable and vice versa [2].

Automata on infinite words, also called ω-automata, have gained importance since their first definition in the 1960s. Apart from the theoretical point of view, they are also important for the specification and verification of reactive systems that are not supposed to terminate at some point in time (for example operating systems). ω-automata enable finite representation of a set of infinite words. Apart from Büchi automata, there are other automata models capable of describing the entire class of ω-regular languages, such as parity, Rabin, Streett, and Muller automata.

Besides verification and synthesis, an important application of connection between logic and automata is to decide the satisfiability problem of various logics (e.g. LTL, QPTL, S1S, ...).

In this paper, we describe the implementation of translating any S1S formula into a corresponding Büchi automaton. We then compare the generated state space of our implementation with as far as we know the only other existing implementation of a decision procedure for S1S, based on loop deterministic finite automata [3].

2. Preliminaries

2.1 Monadic Second-Order Logic of One Successor (S1S)

Monadic second-order logic of one successor is a logic interpreted over natural numbers that allows us to manipulate positions directly, through second-order variables, which store sets of positions, and through a successor operation *Succ*.

Monadic second-order logic is a fragment of the full second-order logic allowing quantification only over unary relations – sets. One successor means that we only have a single successor operation.

Terms are defined by the following grammar:

$$t ::= X \,|\, Succ(X),$$

where X is a second-order variable and $Succ(X)$ is a successor operation.

Formulae are defined by the following grammar:

$$\varphi ::= 0 \in X \,|\, X_1 \subseteq X_2 \,|\, X_1 = Succ(X_2) \,|\, Sing(X) \,|$$

$$\neg\varphi \,|\, \varphi_1 \wedge \varphi_2 \,|\, \varphi_1 \vee \varphi_2 \,|\, \exists X.\varphi$$

We also allow using other usual boolean connectives:

- $\forall X.\varphi := \neg\exists X.\neg\varphi$
- $\varphi_1 \rightarrow \varphi_2 := \neg\varphi_1 \vee \varphi_2$
- $\varphi_1 \leftrightarrow \varphi_2 := (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$

Let V be a set of second-order variables. A *second-order valuation* is a function $\sigma : V \rightarrow 2^{\mathbb{N}}$ which assigns a set of natural numbers to each second-order variable.

The value of a term is then defined as follows:

- $\sigma \models 0 \in X \iff 0 \in \sigma(X)$
- $\sigma \models X_1 \subseteq X_2 \iff \sigma(X_1) \subseteq \sigma(X_2)$
- $\sigma \models X_1 = Succ(X_2) \iff \sigma(X_1) = \{y \,|\, y = x+1, x \in \sigma(X_2)\}$
- $\sigma \models Sing(X) \iff |X| = 1$, i.e. X is a singleton
- $\sigma \models \neg\varphi \iff \sigma \not\models \varphi$
- $\sigma \models \varphi_1 \wedge \varphi_2 \iff \sigma \models \varphi_1$ and $\sigma \models \varphi_2$
- $\sigma \models \varphi_1 \vee \varphi_2 \iff \sigma \models \varphi_1$ or $\sigma \models \varphi_2$
- $\sigma \models \exists X.\varphi \iff \exists A \subseteq \mathbb{N} : \sigma \triangleleft \{X \mapsto A\} \models \varphi$, where $\sigma \triangleleft \{X \mapsto A\}(Y) = \begin{cases} A & \text{for } Y = X \\ \sigma(Y) & \text{otherwise} \end{cases}$

Formula φ is *satisfiable* if there is a valuation σ such that $\sigma \models \varphi$.

2.2 Büchi automata

In this section we focus on necessary definitions related to Büchi automata.

Definition 1. Let Σ be a nonempty finite alphabet. An *ω-word* (or *infinite word*) is an infinite sequence of symbols of alphabet Σ. Σ^ω then denotes a set of all infinite words over Σ. A set of ω-words over a given alphabet is called an *ω-language*.

Definition 2. A (nondeterministic) *Büchi automaton* (BA) is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$, where

- Q is a nonempty finite set of states,
- Σ is a nonempty finite alphabet,
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function,
- $Q_0 \subseteq Q$ is a set of initial states, and

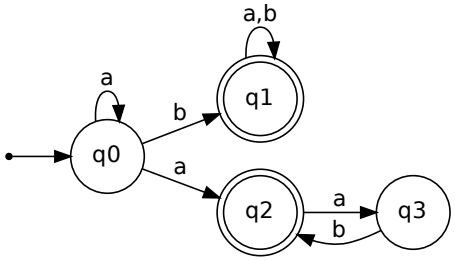


Figure 1. Büchi automaton. $Q = \{q_0, q_1, q_2, q_3\}$; $\Sigma = \{a, b\}$; $\delta(q_0, a) = \{q_0, q_2\}$, $\delta(q_0, b) = \{q_1\}$, $\delta(q_1, a) = \{q_1\}$, $\delta(q_1, b) = \{q_1\}$, $\delta(q_2, a) = \{q_3\}$, $\delta(q_2, b) = \emptyset$, $\delta(q_3, a) = \emptyset$, $\delta(q_3, b) = \{q_2\}$; $Q_0 = \{q_0\}$; $F = \{q_1, q_2\}$.

- $F \subseteq Q$ is a set of final states.

A *run* of A on an ω-word $a_0a_1a_2... \in \Sigma^\omega$ is an infinite sequence $\rho = q_0q_1q_2...$, such that $q_i \in Q$ for every $i \geq 0$, $q_0 \in Q_0$, and $q_{i+1} \in \delta(q_i, a_i)$ for every $i \geq 0$.

Let $inf(\rho) = \{q \in Q \,|\, q = q_i$ for infinitely many i's$\}$ be the set of states that occur in ρ infinitely often. A run ρ is accepting if $inf(\rho) \cap F \neq \emptyset$. A Büchi automaton *accepts* an ω-word $w \in \Sigma^\omega$ if it has an accepting run on w. The language recognized by a Büchi automaton A is the set $L_\omega(A) = \{w \in \Sigma^\omega \,|$ w is accepted by $A\}$.

Definition 3. Languages that can be recognized by Büchi automata are called *ω-regular languages* and can be defined by *ω-regular expressions*.

The automaton in Figure 1 recognizes the language described by the following ω-regular expression: $a^*b(a+b)^\omega + a^+(ab)^\omega$.

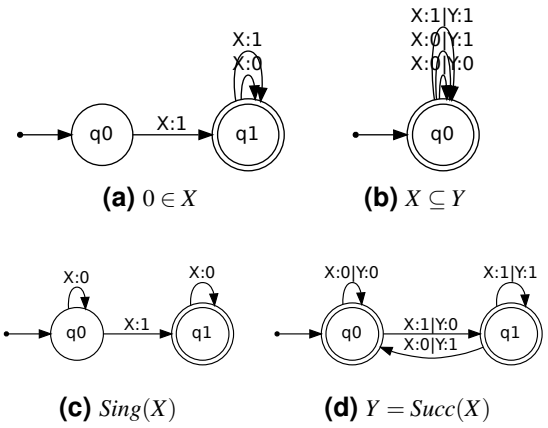
Closure properties of the class of ω-regular languages are summarized by the following theorem.

Theorem 1. *ω-regular languages are closed under union, intersection, and complementation* [2].

3. A decision procedure for S1S

Büchi automata were originally introduced in order to develop a decision procedure for S1S. In this section, we show how we can inductively construct a Büchi automaton for every S1S formula.

Let $A = (Q, \Sigma, \delta, Q_0, F)$ be a BA. A symbol of Σ over set of variables $\mathbb{X}$ is a function with the signature $\mathbb{X} \rightarrow \{0, 1\}$. An ω-word $a_0a_1a_2... \in \Sigma^\omega$ encodes a set X. If a_n contains X: 1, then $n \in X$. Otherwise, $n \notin X$.



(a) $0 \in X$ **(b)** $X \subseteq Y$

(c) $Sing(X)$ **(d)** $Y = Succ(X)$

Figure 2. Automata for atomic formulae

Construction of BAs from atomic formulae is shown in Figure 2. Additionally, we show how to inductively construct a BA for any S1S formulae:

- $\neg\varphi$: Let A_φ be the automaton constructed for φ. We obtain the automaton for $\neg\varphi$ by constructing the automaton accepting $\Sigma^\omega \setminus L(A_\varphi)$.
- $\varphi_1 \wedge \varphi_2$: Let A_{φ_1} and A_{φ_2} be the automata constructed for φ_1 and φ_2, respectively. We obtain the automaton for $\varphi_1 \vee \varphi_2$ by extending the alphabet and constructing the automaton accepting $L(A_{\varphi_1}) \cap L(A_{\varphi_2})$.
- $\varphi_1 \vee \varphi_2$: Let A_{φ_1} and A_{φ_2} be the automata constructed for φ_1 and φ_2, respectively. We obtain the automaton for $\varphi_1 \vee \varphi_2$ by constructing the automaton accepting $L(A_{\varphi_1}) \cup L(A_{\varphi_2})$.
- $\exists X.\varphi$: Let A_φ be the automaton constructed for φ. The automaton for $\exists X.\varphi$ is then constructed by eliminating symbol X from the input alphabet. That means that we replace each transition $\delta(q,a) = q'$ by $\delta(q, a \setminus \{X:0, X:1\})$.

An S1S formula is satisfiable if there exists an accepting run on corresponding Büchi automaton. Otherwise, when there is no accepting run, i.e. the automaton recognizes an empty language, the formula is unsatisfiable. We can also test the validity of the formula. A formula φ is valid iff $\neg\varphi$ is unsatisfiable.

4. Implementation

The tool constructing a Büchi automaton from an S1S formula[1] was written in Python. It implements the decision procedure described in Section 3. The tool can be used as it is, without any external tools or libraries. However, during our experiments, we also used the tool RABIT [4] for heavyweight automata reduction and SPOT [5] for comparing the results using another algorithm for BA complementation.

For union and intersection, the tool uses standard algorithms for Büchi automata [2]. Complementation is performed either by Schewe's optimal construction [6] improving the original rank-based construction [7, 8] or by determinization-based complementation implemented within SPOT [5]. 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 Büchi automata is still a hot topic of current research [9, 10].

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 [11] or disconnecting little brother states [12], and (ii) *heavyweight reductions*, based on a 10-step lookahead simulation relation combined with advanced transition pruning, implemented in the tool RABIT [4]. Apart from that, we also used Tarjan's algorithm to identify strongly connected components of an automaton. We can remove all states in a strongly connected component if there are no transitions to another strongly connected component, and it contains no final states, or it consists of only one final state with no transitions going both from and to this state. We can also decide the satisfiability of a formula using this algorithm.

We defined S1S using only second-order variables in Section 2.1. We can treat a first-order variable x as a second-order variable X and then intersect the automaton with $Sing(X)$.

We can see an example of the final automaton for one of the tested formulae in Figure 3. Note that we use a special symbol '?' to show that we do not care about the value assigned to a variable in a certain transition and both options are possible. This way, we can significantly reduce the number of transitions we are working with.

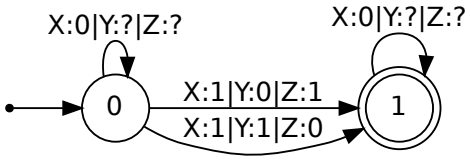


Figure 3. Automaton for the S1S formula $(x \in Y \wedge x \notin Z) \vee (x \in Z \wedge x \notin Y)$

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

Formulae in Table 1, which we used for our experiments, also contain a predicate $x < y$. This predicate can be expressed as the following formula [13]:

$$X < Y := Sing(X) \wedge Sing(Y) \wedge \neg(X \subseteq Y) \wedge \forall Z\ ((Y \subseteq Z\ \wedge \forall U\ (\forall V\ ((V\ = Succ(U)) \rightarrow V\ \subseteq Z) \rightarrow U\ \subseteq Z)) \rightarrow X \subseteq Z)$$

Even though it is not necessary to implement an atomic automaton for this predicate, we have implemented it in order to improve the efficiency of the decision procedure. The automaton is shown in Figure 4.

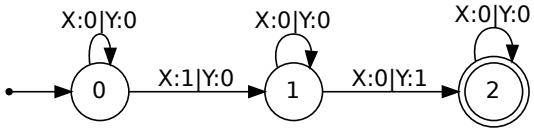


Figure 4. Atomic automaton for $x < y$

5. Experiments

In this section, we compare the efficiency of using loop deterministic finite automata (L-DFA) [3] and Büchi automata within our implementation for various S1S formulae. The efficiency is measured mainly in the state count of the final automaton for each formula. The formulae along with the resulting state count of L-DFA were taken from [3]. The results are shown in Table 1.

Implementation based on BAs seems to give better results than on L-DFAs in terms of state count, as shown in Table 1. Especially when using SPOT [5] for complementation, the state count of the resulting automata were lower in the majority of cases compared to L-DFAs. There were only two worse results, one of them being the formula that did not finish in a day. However, it is not only the number of states of the automata that we should focus on. Unfortunately, we do not have access to the implementation of the operations over L-DFAs used in [3] so we cannot properly compare the efficiency of those algorithms with our implementation, but judging by the results, we can assume that complementation is not such a demanding problem for L-DFAs.

The positive conclusion is that the implementation using BAs and L-DFAs seems to be complementary. For the formulae where the complementation of an automaton with a lot of states is required, the implementation on L-DFAs is able to give at least some result. On the other hand, in formula 20, the L-DFA has 2 331 states, which is much more than the corresponding BA with 32 states. The state count for formulae with intersections of offsets (following the pattern in formulae 19 and 20) grows exponentially for BAs, that is, the final BA has 64 states for the intersection of 6 offsets, 128 for 7 offsets, etc., but the corresponding L-DFA seems to grow much quicker.

6. Conclusions

In this paper, we presented the connection between S1S and Büchi automata. We described the syntax and semantics of S1S and the decision procedure which shows how to inductively construct a corresponding Büchi automaton for any S1S formula. We then compared our approach with the one using loop deterministic finite automata.

When compared to the implementation using loop deterministic finite automata, Büchi automata offer an efficient decision procedure for S1S formulae in terms of state count of the resulting automaton. The problem lies in complementation, which is a difficult and complex problem for automata with a lot of states.

It seems that BAs and L-DFAs complement each other when using as a decision procedure for S1S formulae. Implementation on L-DFAs gives better results (in terms of the number of states) when complementation of a BA with a lot of states is required and on the contrary, implementation on BAs seems to give better results when we do not need to complement a big automaton.

In this work, the comparison of the efficiency of BAs and L-DFAs was based mainly on the number of states of the final automaton, but as we discussed in Section 5, we should not make further conclusions based only on these data. For future work, it would be interesting to compare the efficiency of algorithms implemented on both types of automata in terms of their time complexity. We could also try to use some other algorithms and their optimizations or external tools for reduction and complementation of Büchi automata.

Acknowledgements

I would like to thank Ing. Ondřej Lengál, Ph.D. and Ing. Vojtěch Havlena for their ideas, help, advice, patience and support throughout this project.

References

[1] J. Richard Büchi. On a decision method in restricted second order arithmetic. In *The Collected Works of J. Richard Büchi*, pages 425–435. Springer, New York, NY.

	Formula	State count		
		BA, Schewe's complementation	BA, complementation using Spot	L-DFA
1	$(x \in Y \wedge x \notin Z) \vee (x \in Z \wedge x \notin Y)$	**2**	**2**	9
2	$\neg\exists x.((x \in Y \wedge x \notin Z) \vee (x \in Z \wedge x \notin Y))$	**1**	**1**	9
3	$after(X,Y) := \forall x.(x \in X \Rightarrow \exists y.(y > x \wedge y \in Y))$	5	**3**	9
4	$fair(X,Y) := after(X,Y) \wedge after(Y,X)$	24	**5**	9
5	$\forall X.(fair(X,Y) \Rightarrow fair(Y,Z))$	Out of memory	21	**14**
6	$suc(x,y) := x < y \wedge \forall z.(\neg x < z \vee \neg z < y))$	**3**	**3**	10
7	$suc2(x,y) := \exists z.(suc(x,z) \wedge suc(z,y))$	**4**	**4**	10
8	$suc4(x,y) := \exists z.(suc2(x,z) \wedge suc2(z,y))$	**6**	**6**	10
9	$suc8(x,y) := \exists z.(suc4(x,z) \wedge suc4(z,y))$	**10**	**10**	13
10	$inf(X) := \forall u \exists v.(u < v \wedge v \in X)$	**3**	**3**	9
11	$inf(X) \vee inf(Y)$	**5**	**5**	9
12	$(inf(U) \vee inf(V)) \Rightarrow (inf(X) \vee inf(Y))$	9	**7**	9
13	$\exists U.((inf(U) \vee inf(V)) \Rightarrow (inf(X) \vee inf(Y)))$	8	**7**	9
14	$infsuc(X,Y) := \forall u \exists x,y.(u < x \wedge suc(x,y) \wedge x \in X \wedge y \in Y)$	**4**	5	18
15	$\exists Y.(infsuc(X,Y) \wedge infsuc(Z,Y))$	19	**6**	20
16	$zeroin(X) := \exists u.(u \in X \wedge \neg\exists v.(v < u))$	**2**	**2**	6
17	$alter(X) := zeroin(X) \wedge \forall x,y.(suc(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X))$	**2**	**2**	12
18	$offset(X,Y) := \forall i \forall j.(suc(i,j) \wedge i \in X \Rightarrow j \in Y)$	**2**	**2**	11
19	$offset(X,Y) \wedge offset(Y,Z) \wedge offset(Z,X)$	**8**	**8**	107
20	$offset(V,W) \wedge offset(W,X) \wedge offset(X,Y) \wedge offset(Y,Z) \wedge offset(Z,V)$	**32**	**32**	2331
21	$\exists Y.(offset(X,Y) \wedge offset(Y,Z))$	**4**	**4**	29
22	$insm(i,j,U,V,W) := (j \in U \Rightarrow i \in V \vee i \in W)$	**8**	**8**	15
23	$\forall i \forall j(suc(i,j) \Rightarrow insm(i,j,U,V,Z) \wedge insm(i,j,V,X,Y) \wedge insm(i,j,X,Y,V) \wedge insm(i,j,Y,Z,X) \wedge insm(i,j,Z,U,Y))$	Out of memory	Timeout	**198**
24	$\forall x \forall y.(x < y \wedge y \in X \wedge y \in Y)$	**3**	**3**	9
25	$\forall x \forall y.(x < y \wedge y \in X \wedge y \in Y) \wedge \forall x \forall y.(x < y \wedge y \in X \wedge y \notin Y)$	7	**4**	9
26	$\forall x \forall y.(x < y \wedge y \in X \wedge y \in Y) \wedge \forall x \forall y.(x < y \wedge y \in X \wedge y \notin Y) \wedge \forall x \forall y.(x < y \wedge y \notin X \wedge y \in Y) \wedge \forall x \forall y.(x < y \wedge y \notin X \wedge y \notin Y)$	21	**11**	18

Table 1. Comparison of resulting state count of Büchi automata and loop deterministic finite automata for various S1S formulae

[2] Bernd Finkbeiner. Automata, games, and verification. Available on: https://www.react.uni-saarland.de/teaching/automata-games-verification-15/downloads/notes.pdf, 2015.

[3] Stephan Barth. Deciding monadic second order logic over ω-words by specialized finite automata. In Erika Ábrahám and Marieke Huisman, editors, *Integrated Formal Methods*, pages 245–259, Cham, 2016. Springer International Publishing.

[4] Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr, and Tomáš Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In *Proc. of CONCUR'11*, pages 187–202. Springer, 2011.

[5] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Étienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In Cyrille Artho, Axel Legay, and Doron Peled, editors, *Automated Technology for Verification and Analysis*, pages 122–129, Cham, 2016. Springer International Publishing.

[6] Sven Schewe. Büchi complementation made tight. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[7] O. Kupferman and Moshe Vardi. Weak alternating automata are not that weak. In *ACM Transactions on Computational Logic*, volume 2, pages 147–158, 07 1997.

[8] Ehud Friedgut, Orna Kupferman, and Moshe Vardi. Büchi complementation made tighter. *International Journal of Foundations of Computer Science*, 17:851–868, 2006.

[9] Yu-Fang Chen, Vojtěch Havlena, and Ondřej Lengál. Simulations in rank-based Büchi automata complementation. In Anthony Widjaja Lin, editor, *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*, pages 447–467. Springer, 2019.

[10] František Blahoudek, Alexandre Duret-Lutz, and Jan Strejček. 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*, pages 15–27. Springer, July 2020.

[11] Lucian Ilie, Gonzalo Navarro, and Sheng Yu. *On NFA Reductions*, pages 112–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[12] Doron Bustan and Orna Grumberg. Simulation based minimization. *ACM Transactions on Computational Logic*, 4, 03 2000.

[13] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://tata.gforge.inria.fr/, 2007. release October, 12th 2007.