hash=dc54e95d84373f969e24b22beac95709family=Poitrenaud, fami-
lyi=P., given=Denis, giveni=D. hash=35e7b914369bc13462bde4e94dc07d9
ily=Klein, familyi=K., given=Joachim, giveni=J.hash=fcc008d91a39a6fed3b337d4a26ab147
ily=Kretínský, familyi=K., given=Jan, giveni=J.hash=9e35d809f21efe4b8c2e8b56e06e7bac-
family=Müller, familyi=M., given=David, giveni=D.hash=85aa04746604480ce47f280e16705
family=Parker, familyi=P., given=David, giveni=D.hash=4a81f3074d6f2168a6f43828d93e0d
ily=Strejcek, familyi=S., given=Jan, giveni=J.

Masaryk University
Faculty of Informatics

Transformation of Nondeterministic Büchi Automata to Slim Automata

Bachelor's Thesis

Pavel Šimovec

Brno, Spring 2021

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Pavel Šimovec

Advisor: doc. RNDr. Jan Strejček, Ph.D.

Acknowledgements

RNDr. František Blahoudek, Ph.D.

Abstract

abstract

Keywords

keyword1, keyword2, ...

Contents

List of Figures

1 Introduction

Büchi automaton is a finite machine over infinite words. It has been a topic of research for almost 60 years. There were discovered various kinds of similar machines with different properties and use cases. Non-deterministic Büchi in general are not well suitable for model checking or reinforcement, but we can construct non-deterministic Büchi automata with a special property - GFM, that makes the automata suitable. We will focus on slim automata [1]. Slim automata are specially constructed from Büchi automata. This kind of automaton was defined by it's construction in [source] and is good for MDP [main source]. We implement the proposed algorithm and it's second variant that we call weak [source private conversation]. We extend the algorithm for generalised Büchi automata. Then we evaluate resulting size of automata and we compare it with different tool to create slim automata and with other kinds of automata.

... slim automata are specially constructed Büchi automata.

2 Preliminaries

In this chapter we define a Büchi automaton and its generalized version. Then we continue with breakpoint algorithm. It allows us to introduce slim automata by its construction, which builds on the breakpoint one. Finally we generalize slim automaton construction to work with generalized Büchi automata.

We will need to know that on *alphabet* is a set of letters, an *ω-word* $w \in \Sigma^\omega$ is an infinite sequence of letters, and a *language* $L \subseteq \Sigma^\omega$ is a set of ω-words.

2.1 Büchi Automaton

A Büchi automaton is a theoretical finite-state machine used to define ω-languages. It decides which infinitely long words (ω-words) belong to its language.

A *transition-based Büchi automaton* (*TBA*) is a tuple $\mathcal{A} \overset{\text{def}}{=} (\Sigma, Q, q_i, \Delta, \Gamma)$, where

- Σ is a non-empty finite *alphabet*,
- Q is a non-empty finite set of *states*,
- $q_i \in Q$ is the initial state of $\mathcal{A}$.
- We write the set of *transitions* as $\Delta \subseteq Q \times \Sigma \times Q$. Intuitivelly, a transition (s, a, t) directionally connects the states s and t with the letter a.
- $\Gamma \subseteq \Delta$ is a set of *accepting transitions*.

A *run* r of $\mathcal{A}$ is an infinite sequence of transitions $r \overset{\text{def}}{=} t_0 t_1 \ldots \in \Delta^\omega$, where $t_i = (s_i, a_i, s_{i+1})$, such that $q_0 = q_i$. A run of $\mathcal{A}$ is *accepting* iff it contains infinitely many accepting transitions.

Finally, we define the *language* $L(\mathcal{A}) \subseteq \Sigma^\omega$ recognized by the automaton $\mathcal{A}$. An ω-word $w \in \Sigma^\omega$ belongs to $L(\mathcal{A})$ iff there exists an accepting run of $\mathcal{A}$ over the word w.

2.2 Generalized Büchi Automaton

A *transition-based Generalized Büchi automaton* (TGBA) is a tuple $\mathcal{A} \overset{\text{def}}{=} (\Sigma, Q, q_i, \Delta, G)$, where $\emptyset \subseteq G \subseteq 2^{\Delta}$ contains sets of accepting conditions and the rest is defined as for TBA. A run of $\mathcal{A}$ is *accepting* iff it contains infinitely many accepting transitions *for each* $\Gamma \in G$. TBA can be seen as a special case of TGBA with $|G| = 1$

2.3 Breakpoint Automaton

We want to define *slim Büchi automaton* (slim automaton) through its construction which is based on breakpoint construction. Breakpoint (BP) automata are a building block for slim automata. BP automata are constructed from BA and are deterministic, but their language is only a subset of the language from original BA.

Construction Let us fix a Büchi Automaton $\mathcal{A} \overset{\text{def}}{=} (\Sigma, Q, q_i, \Delta, \Gamma)$.

By 3^Q we denote the set $\{(S, S') \mid S' \subsetneq S \subseteq Q\}$ and by 3^Q_+ we denote $\{(S, S') \mid S' \subseteq S \subseteq Q\}$.

Given the set of transitions, we define the function $\delta\colon 2^Q \times \Sigma \to 2^Q$ as $\delta\colon (S, a) \overset{\text{def}}{=} \{q' \in Q \mid \exists q \in S.(q, a, q') \in \Delta\}$. We define $\gamma : 2^Q \times \Sigma \to 2^Q$ analogously from Γ as $\gamma : (S, a) \overset{\text{def}}{=} \{q' \in Q \mid \exists q \in S.(q, a, q') \in \Gamma\}$.

By definitions, δ and γ are deterministic transition functions.

Using Δ, we define the raw breakpoint transition $\rho_\Gamma\colon 3^Q \times \Sigma \to 3^Q_+$ as

$$\rho_\Gamma((S, S'), a) \overset{\text{def}}{=} (\delta(S, a), \delta(S', a) \cup \gamma(S, a))$$

We follow the set of reachable states in the first set and the states that are reachable while passing at least one of the accepting transitions in the second set. The transitions of the breakpoint automaton $\mathcal{D}$ follow ρ with an exception: they reset the second set to the empty set when it equals the first; the resetting transitions are accepting. Formally, the breakpoint automaton $\mathcal{D}$ is $\overset{\text{def}}{=} (\Sigma, 3^Q, (q_i, \emptyset), \delta_D, \gamma_D)$ $\rho\colon ((S, S'), a) = (R, R')$, then there are three cases:

1. if $R = \emptyset$, then $\delta_D((S, S'))$ is undefined (or, if a complete automation is preferred, maps to a rejecting sink),

2. else, if $R \neq R'$, then $\delta_D((S, S'), a) = (R, R')$ is a non-accepting transition, and $\gamma_D((S, S'), a)$ is undefined.
3. otherwise $\delta_D((S, S') = \gamma_D((S, S'), a) = (R, \emptyset)$ is an accepting transition.

Figure 2.1) shows application of this construction.

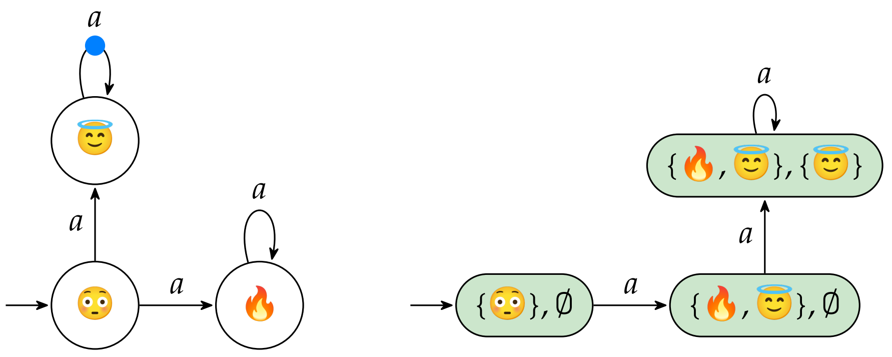

Figure 2.1: Example of breakpoint automaton $\mathcal{D}$ (right) non-equivalent with the original Buchi Automaton $\mathcal{A}$ (left). The example demonstrates that $L(\mathcal{D}) \subseteq L(\mathcal{A})$ as the construction did not generate any accepting transition. Therefore original $L(\mathcal{A}) = \{a^\omega\}$, but $L(\mathcal{D})$ is empty.

3 Slim Automata Construction

Slim automata are BP automata enriched with additional transitions. As a result they are non-deterministic, GFM[1] and equivalent to input automaton.

In this chapter we define transitions for *strong slim* (γ_p) and *weak slim* (γ_p) automata, $\gamma_w, \gamma_p : 3^Q \times \Sigma \to 3^Q$, that promote the second set of a breakpoint construction to the first set as follows.

1. if $\delta_S(S', a) = \gamma_S(S, a) = \emptyset$, then $\gamma_p((S, S'), a)$ and $\gamma_w((S, S'), a)$ are undefined, and
2. otherwise $\gamma_p : ((S, S'), a) = (\delta(S', a) \cup \gamma(S, a), \emptyset)$ and $\gamma_w : ((S, S'), a) = (\delta(S', a), \emptyset)$

$\mathcal{S} \stackrel{\text{def}}{=} (\Sigma, 3^Q, (q_i, \emptyset), \Delta_S, \Gamma_S)$ is slim, when $\Delta_S = \Delta_D \cup \Gamma_p$ is set of transitions generated by δ_D and γ_p, and $\Gamma_S = \Gamma_D \cup \Gamma_p$ is set of accepting transitions, that is generated by γ_D and γ_p. $L(\mathcal{S}) = L(\mathcal{A})$. The equivalence was proven in [1].

Alternatively, similarly defined using γ_w instead of γ_p, automaton $\mathcal{W} \stackrel{\text{def}}{=} (\Sigma, 3^Q, (q_i, \emptyset), \Delta_W, \Gamma_W)$ is slim when $\Delta_W = \Delta_D \cup \Gamma_w$ is set of transitions generated by δ_D and γ_w, and $\Gamma_W = \Gamma_D \cup \Gamma_w$ is set of accepting transitions, that is generated by γ_D and γ_w. $L(\mathcal{S}) = L(\mathcal{A})$ and $L(\mathcal{S}) = L(\mathcal{A})$. (proof would go similarly like the one for strong slim)

1. Good for Markov decision processes [1]

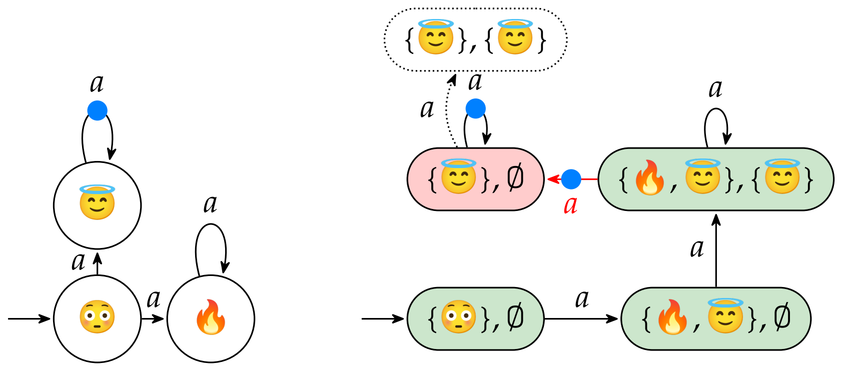

Figure 3.1: Slim automaton (right) and the original Buchi Automaton from Figure 2.1(left)

4 Slim Automaton Construction Generalized to TGBA

In this chapter, we discuss slim automata equivalent to a TGBA $\mathcal{T} \overset{\text{def}}{=} (\Sigma, Q, q_i, \Delta, G)$. One possibility is to *degeneralize* $\mathcal{T}$ and to use previously mentioned algorithm in section 2.3. In the rest of this chapter we introduce a direct construction of slim TGBA equivalent to $\mathcal{T}$.

Extended slim construction (We will simulate the original automaton by checking its accepting conditions on by one. In the original automaton have to go through an accepting transition of each accepting condition $g \in G = \{G_0, G_1, \dots, G_k\}$ infinitely many times. In new automaton we have just one accepting condition and a layer for each original accepting condition. Going through original accepting transitions of layer that we are looking up promotes us to another layer. From the last layer we get back to first layer. Only the transitions that move us layer up are accepting. As we check all accepting conditions of the original automaton, the new automaton will be equivalent to the original one.)

We need to make sure we go infinitely many times trough each accepting subset $g \in G$. To achieve this, we will go through each subset one by one, using original algorithm. We will keep track of *levels* $\overset{\text{def}}{=} \{0, 1, \dots, |G| - 1\}$ in the names of states. Let $|G|$ be number of *levels* and $i \in N, i < |G|$ the current level. At each level i, we look at ith subset of G. We use same steps as in classic breakpoint construction, but on each accepting transition the new state will be leveled up to $(i+1) \bmod |G|$, otherwise the target state has the same level. Our new automaton simulates $\mathcal{T}$, as it accepts a word if it cycles through all levels. If $|G| = 0$, we return a trivially accepting automaton

We can use the core of previous construction and just to extend it with levels. Let $up(x) \overset{\text{def}}{=} (x+1) \bmod |G|$ and let

$P := 3^Q \times levels.$

Let $(S, S') \in 3^Q$ and let $i \in levels$, by P we denote a state $P = (S, S', i)$.

We define γ_i from Γ_i for all $i \in \mathit{levels}$ in the same way we did for γ from Γ and it allows us to easily define the raw generalized breakpoint transitions ρ_{Γ_i}: similarly as ρ_Γ using γ_i instead of γ.

The generalized breakpoint automaton $\mathcal{D} = (\Sigma, 3^Q \times \mathcal{N}, (q_i, \emptyset, 0), \delta_B, \gamma_B)$ is defined such that, when $\delta_R\colon (P, a) \rightarrow (R, R', j)$, then there are three cases:

1. if $R = \emptyset$, then $\delta_B(P, a)$ is undefined,
2. else, if $R \neq R'$, then $\delta_B(P, a) = (R, R', i)$ is a non-accepting transition,
3. otherwise $\gamma_B(P, a) = \delta_B(P, a) = (R, \emptyset, up(i))$.

1. if $\delta(S', a) = \gamma_i(S, a) = \emptyset$, then $\gamma_p(P, a)$ is undefined, and
2. otherwise $\gamma_p\colon (P, a) = (\delta(S', a) \cup \gamma_i(S, a), \emptyset, up(i))$. (Alternatively, for a weak slim automaton we do not include transitions $\gamma_i(S, a)$)

$\mathcal{S} \overset{\text{def}}{=} (\Sigma, P, (q_i, \emptyset, 0), \Delta_p, \Gamma_p))$ is slim, when Δ_p is set of transitions generated by δ_b and γ_p, and Γ_p is set of accepting transitions, that is generated by γ_b and γ_p. We construct weak slim automata

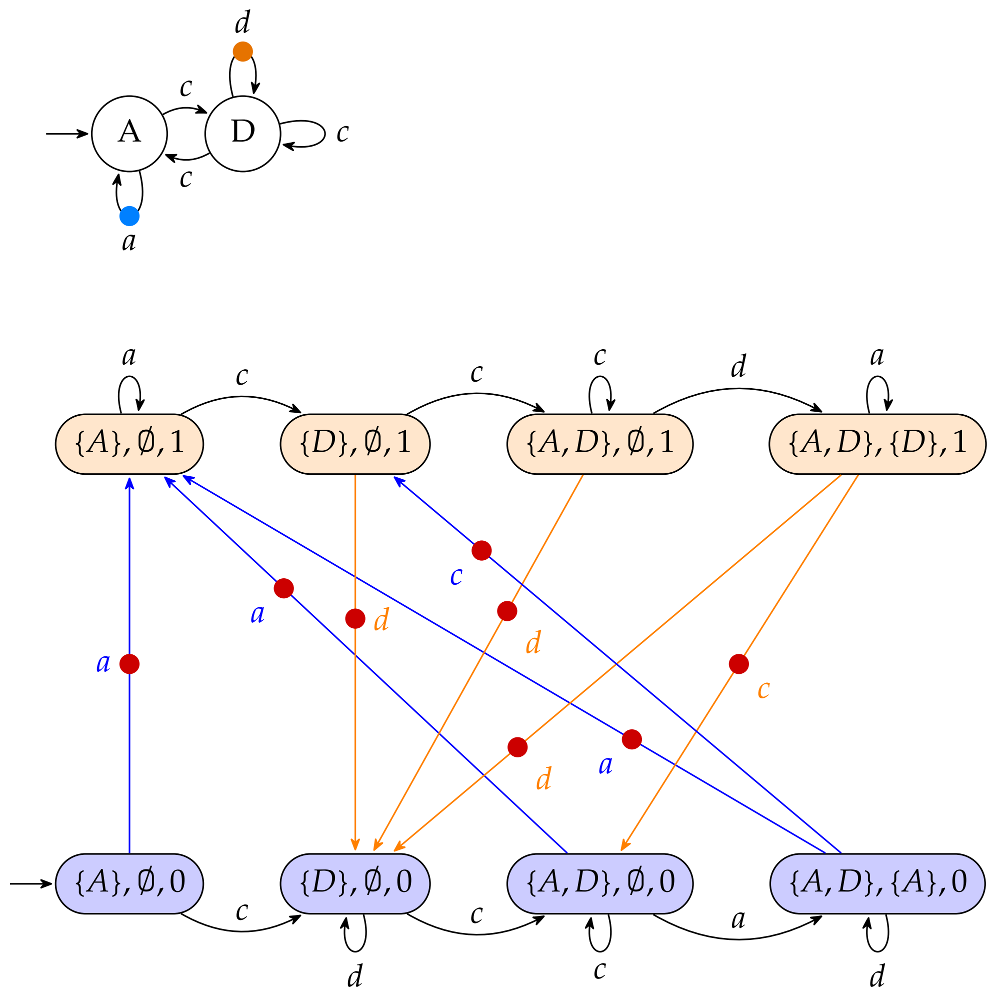

Figure 4.1: The original TGBA (top) and slim automaton with colored states emphasizing different levels (bottom)

5 Implementation

I have implemented the generalized construction of slim automata in both weak and strong version (3) (4). I have also added option to create breakpoint automata (2.3)(2.3).

5.1 Technologies/Tools

The implementation is inside seminator which is implemented in C++17 builds on Spot library.

5.1.1 Seminator

Seminator is a Linux command-line tool which can be run with the `seminator` command. The tool transforms transition-based generalized B̈uchi automata (TGBAs) intoequivalent semi-deterministic automata. [3]

The tool expects the input automaton in the Hanoi Omega-Automata(HOA) format [4] on the standard input stream, but it can also read the input automaton from a file.

5.1.2 Spot

Spot is a C++ library with Python bindings and an assortment of command-line tools designed to manipulate LTL and ω-automata in batch. [5]

Relevant spot tools:

ltl2tgba The ltl2tgba tool translates LTL or PSL formulas into different types of automata. [6]

autfilt The autfilt tool can filter, transform, and convert a stream of automata. [7]

ltlcross ltlcross is a tool for cross-comparing the output of LTL-to-automata translators. [8]

5.2 Create Slim Automata Using Seminator

By default, seminator creates sDBA. To create a slim automaton we need to add –slim option.

Options By default, –slim tries all reasonable combinations of options, optimizes the output and chooses an automaton with the smallest number of states.

Example 1 Transform automaton.hoa to a slim automaton.

```
$./seminator --slim -f automaton.hoa
```

There are several options to specify how we construct the automata.

For example `seminator --slim --strong --optimizations=0 --via-tgba` generates output according to algorithm in 2.5. (Using `--via-tba` converts input to tba first) With automaton

Example 2 Transform automaton.hoa to unoptimized strong slim automaton

```
$./seminator --slim --strong --via-tgba --
    optimalizations=0 -f automaton.hoa
```

`--slim` to generate slim automaton

`--weak` use only weak slim algorithm

`--strong` use only strong slim algorithm

Neither weak or strong option specified - try both options and choose the one with smaller automaton.

`--via-tba` transform input automaton to tba (2.1) first

`--via-tgba` does not modify input automaton to tba.

Neither `--via-tba` nor `--via-tgba`: try both options, choose the smallest automaton

Postprocess optimalizations are enabled by default.

5.3 Implementation of Slim Automata inside Seminator

I have implemented the generalized slim construction and its options mentioned in previous section 3.2. Furthermore, I have added an option to create breakpoint automata.

There already was basis for breakpoint construction in seminator, inside class `bp_twa`. As we can see in sections 2.3 and 3, slim automata construction builds on breakpoint automata construction.

That allows us to simply extend the `bp_twa` class. We create class `slim` that inherits from `bp_twa`. In the `slim` class we build breakpoint automaton using `compute_successors` method. Then we extend the method by adding accepting transitions γ_p, respectively γ_w according to section 2.4, whenever we receive `--slim` option.

Then we extend main function to recognize our desired CLI options.

As seminator didn't offer a command line option to create a breakpoint automaton, I have added the option `--bp` for comparison.

5.4 Testing and Verification

Implemented tests are basic, only language equivalence is checked. ltlcross and ltl2tgba tools are used. The tests use random LTL formulas that were already generated, the LTL formulas are transformed into automata in HOA format by ltl2tgba. Then the tool ltlcross cross-compares the automaton with *seminator –slim* with all supported [not yet] additional parameters.

Only `seminator --slim --strong --via-tba` (and without optimalizations) is proved, as it follows construction from [1] which is proved.

5.5 How to Install Seminator

(jeste nevim kde tuto sekci dat, jestli ma mit tento nazev, co vsechno tady bude treba dat... no a jeste ro pak upravim podle toho jak to bude odevzdane v zipu)

To install the tool we need install spot and to run

```
autoreconf -i && ./configure && make.
```

5.6 Future of Implementation

Implementation: python bindings, optimizations of slim construction (especially from TGBA)

tests/verification: There should be another kind of tests - to check if our slim automata simulate the input automata (so the GFM property is not broken)

Subject of following research, that is out of scope of this thesis, could be to verify if spot optimizations do not break the simulation property.

6 Evaluation

Evaluation part builds on seminator-evaluation. We compare amount of states of output automata. We compare the data on 2 dataset. First dataset are 20 literature formulas, second dataset is 500 automata that were randomly generated.

6.1 Seminator --slim

In this section we compare automaton size generated by seminator –slim. We compare weak against slim and via-tba against via-tgba.

6.1.1 Comparisons among Unoptimized Configurations

In this subsection we compare base unoptimized seminator options.

Table 6.1: literature: unoptimized seminator --slim

seminator	weak		strong	
literature	size	time(s)	size	time(s)
via tba	5x51	21x9	37x0	1x85
via tgba	5x88	21x0	40x8	1x74

Table 6.2: random: unoptimized seminator --slim

seminator	weak		strong	
random	size	time(s)	size	time(s)
via tba	551	219	370	185
via tgba	588	210	408	174

Transforming automata to TBA first yields smaller automata. This might be caused by Spot having well optimized algorithm for degeneralization. Slim algorithm for TGBA proposed in this paper is naive, without any kind of optimizations, and it degeneralizes the automaton during the process.

Using weak slim algorithm creates smaller slim automata than the strong one.

6.1.2 Post-Optimized

In this subsection we post-optimize results using autfilt.

literature

seminator	weak		strong		best	
literature	size	time(s)	size	time(s)	size	time(s)
via tba	551	219	370	185	370	404
via tgba	588	210	408	174	402	384
best	551	429	370	359	365	788

500 randomly generated automata

seminator	weak		strong		best	
random	size	time(s)	size	time(s)	size	time(s)
via tba	8923	443	7404	476	7219	919
via tgba	10130	654	8500	591	8247	1245
best	8751	1097	7285	1067	7088	2164

From 4 base options; after applying post-optimizations strong slim algorithm surpasses weak one by resulting automaton size, even if it has worse results without the post-optimizations. Degeneralizing the automata as a first step still has smaller results. From 4 base options, strong slim algorithm via-tba creates smallest automata on average. Transforming input automata to tba first creates results which are close to best ones. If execution time is not a concern

Minimal hits for random automata weak x strong

literature	unique minimal hits	minimal hits
weak	4	9
strong	11	16

random	unique minimal hits	minimal hits
weak	68	202
strong	296	430

Minimal hits for random automata via-tba versus via-tgba

Let us note that 13/20 formulas from literature and 391/500 formulas from random dataset create automata that are already TBA.

literature	unique minimal hits	minimal hits
via-tba	7	20
via-tgba	0	13

random	unique minimal hits	minimal hits
via-tba	91	489
via-tgba	9	407

6.2 Slim Automata Produced Seminator versus ePMC

popsat co to je ePMC+zdroj prvne jedna vybrana moznost

At first we compare best working basic paramaters (parameters that try only 1 option) of each tool to create smallest automata

random	size	time(s)
epmc acc	9270	
seminator tba strong	6840	

random	size	time(s)
epmc acc	9270	
seminator tba strong	6840	

Now let us compare smallest automata of each tool and to see how smaller automata get by combining these 2 tools.

literature	size	time(s)
epmc best	536	
seminator best	365	
seminator+epmc best	349	

random	size	time(s)
epmc best	10197	
seminator best	7133	
seminator+epmc best	7060	

literature	unique minimal hits	minimal hits
epmc	1	5
seminator	15	19

random	unique minimal hits	minimal hits
epmc	35	155
seminator	344	464

3 (2.5) prvne tba zkusit pro epmc

6.3 Compare with semi-deterministic Automata

random	size	time(s)
ltl2tgba	3000cca	
seminator default	3700	
seminator slim best	7060	

6.4 To Put Somewhere in This Chapter

If the automata optimizations by spot's `autfilt` tool break the simulation property, the results in following Evaluation chapter are pointless, as they are built on such assumption.

7 Conclusion

Bibliography

1. HAHN, Ernst Moritz; PEREZ, Mateo; SCHEWE, Sven; SOMENZI, Fabio; TRIVEDI, Ashutosh; WOJTCZAK, Dominik. Good-for-MDPs Automata for Probabilistic Analysis and Reinforcement Learning. In: BIERE, Armin; PARKER, David (eds.). *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*. Springer, 2020, vol. 12078, pp. 306–323. Lecture Notes in Computer Science. Available from DOI: `10.1007/978-3-030-45190-5\_17`.
2. DURET-LUTZ, given=Alexandre, giveni=A. SPOT: An Extensible Model Checking Library Using Transition-Based Generalized Büchi Automata. In: DEGROOT, Doug; HARRISON, Peter G.; WIJSHOFF, Harry A. G.; SEGALL, Zary (eds.). *12th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004), 4-8 October 2004, Vollendam, The Netherlands*. IEEE Computer Society, 2004, pp. 76–83. Available from DOI: `10.1109/MASCOT.2004.1348184`.
3. BLAHOUDEK, František; DURET-LUTZ, Alexandre; STREJČEK, Jan. Seminator 2 Can Complement Generalized Büchi Automata via Improved Semi-Determinization. In: *Proceedings of the 32nd International Conference on Computer-Aided Verification (CAV'20)*. Springer, 2020, vol. 12225, pp. 15–27. Lecture Notes in Computer Science. Available from DOI: `10.1007/978-3-030-53291-8_2`.
4. BABIAK, Tomás; BLAHOUDEK, Frantisek; DURET-LUTZ, given=Alexandre, giveni=A. The Hanoi Omega-Automata Format. In: KROENING, Daniel; PASAREANU, Corina S. (eds.). *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*. Springer, 2015, vol. 9206, pp. 479–486. Lecture Notes in Computer Science. Available from DOI: `10.1007/978-3-319-21690-4\_31`.
5. DURET-LUTZ, Alexandre; LEWKOWICZ, Alexandre; FAUCHILLE, Amaury; MICHAUD, Thibaud; RENAULT, Étienne; XU, Laurent. Spot 2.0 — A Framework for LTL and $$\omega $$-Automata

Manipulation. In: ARTHO, Cyrille; LEGAY, Axel; PELED, Doron (eds.). *Automated Technology for Verification and Analysis*. Cham: Springer International Publishing, 2016, pp. 122–129. ISBN 978-3-319-46520-3.

6. ROOT. [N.d.]. Available also from: `https://spot.lrde.epita.fr/ltl2tgba.html`.
7. ROOT. [N.d.]. Available also from: `https://spot.lrde.epita.fr/autfilt.html`.
8. ROOT. [N.d.]. Available also from: `https://spot.lrde.epita.fr/ltlcross.html`.