MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Transformation of Nondeterministic Büchi Automata to Slim Automata

BACHELOR'S THESIS

Pavel Šimovec

Brno, Fall 2020

Replace this page with a copy of the official signed thesis assignment and a copy of the Statement of an Author.

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: RNDr. František Blahoudek, Ph.D.; doc. RNDr. Jan Strejček, Ph.D.

Acknowledgements

ack

Abstract

abstract

Keywords

keyword1, keyword2, …

Contents

List of Figures

1 Introduction

... slim automata are specially constructed Büchi automata...

2 Preliminaries

In this ~~section~~ 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 automation 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 ~~$\mathcal{A} = (\Sigma, Q, q_i, \Delta, \Gamma)$~~ $\mathcal{A} \stackrel{def}{=} (\Sigma, Q, q_i, \Delta, \Gamma$ 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 ~~two states inside Q with a letter from alphabet Σ~~ 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 ~~$r = (q_0, a_0, q_1)(q_1, a_1, q_2)(q_2, a_2, q_3)\ldots \in$~~ $r \stackrel{def}{=} t_0t_1\ldots \in \Delta^\omega$, where $t_i = (s_i, a_i, s_{i+1})$, such that $q_0 = q_i$. A run of ~~TBA~~ $\mathcal{A}$ is *accepting* iff it contains infinitely many accepting transitions.

Finally, we define the *language* ~~$L_A \in \Sigma^\omega$~~ $L(\mathcal{A}) \subseteq \Sigma^\omega$ recognized by the automaton $\mathcal{A}$. An ω-word $w \in \Sigma^\omega$ belongs to ~~L_A~~ $L(\mathcal{A})$ 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 ~~$\mathcal{A} = (\Sigma, Q, q_i, \Delta, G)$ is a modified TBA, where $G = \{\Gamma_0, \Gamma_1, \dots, \Gamma_{|G|-1}\} \subseteq 2^\Delta$~~ $\mathcal{A} \stackrel{def}{=} (\Sigma, Q, q_i, \Delta, G)$, 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* ~~((element)) of G. Büchi Automata~~ $\Gamma \in G$. TBA can be seen as a special case of TGBA with $|G| = 1$

2.3 Breakpoint Automaton

We want to define *slim GFM*[1] *Büchi automaton* (slim automaton) through its construction which is based on breakpoint construction.

Construction Let us fix a Büchi Automaton ~~$\mathcal{A} = (\Sigma, Q, q_i, \Delta, \Gamma)$. We define the variations of subset and breakpoint constructions that are used to define semi-deterministic GFM automata (semi-deterministic automata) - which we use in our evaluation for comparison - and the slim automata we construct~~ $\mathcal{A} \stackrel{def}{=} (\Sigma, Q, q_i, \Delta, \Gamma)$. We want to construct a deterministic automaton $\mathcal{D}$ such that $L(\mathcal{D}) \subseteq L(\mathcal{A})$ We denote 3^Q by the set $\{(S, S') \mid S' \subsetneq S \subseteq Q\}$ and 3^Q_+ by $\{(S, S') \mid S' \subseteq S \subseteq Q\}$.

~~Let $3^Q := \{(S, S') \mid S' \subsetneq S \subseteq Q\}$ and $3^Q_+ := \{(S, S') \mid S' \subseteq S \subseteq Q\}$.~~
We define the ~~subset~~ notation for the transitions and accepting transitions as ~~$\delta_S, \gamma_S : 2^Q \times \Sigma \to 2^Q$~~ $\delta, \gamma : 2^Q \times \Sigma \to 2^Q$ with

~~$\delta : (S, a) \mapsto \{q' \in Q \mid \exists q \in S.(q, a, q') \in \Delta\}$~~ $\delta : (S, a) \stackrel{def}{=} \{q' \in Q \mid \exists q \in S.(q, a, q') \in \Delta\}$
and
~~$\gamma : (S, a) \mapsto \{q' \in Q \mid \exists q \in S.(q, a, q') \in \Gamma\}$~~ $\gamma : (S, a) \stackrel{def}{=} \{q' \in Q \mid \exists q \in S.(q, a, q') \in \Gamma\}$

(? Let us note that δ and γ are deterministic transitions.)

We define the raw breakpoint transition ~~$\rho : 3^Q \times \Sigma \to 3^Q_+$ as $((S, S'), a) \mapsto (\delta(S, a), \delta(S', a) \cup \gamma$~~
~~In this construction, we~~ $\rho_\Gamma : 3^Q \times \Sigma \to 3^Q_+$ as

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

1. Good for Markov decision processes [+zdroj]

We follow the set of reachable states (first set) and the states that are reachable while passing at least one of the accepting transitions (second set). ~~To turn this into a breakpoint automaton , we~~ 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 ~~transitions where we reset the second set are exactly the acceptingones~~resetting transitions are accepting. The breakpoint automaton ~~$\mathcal{D} = (\Sigma, 3^Q, (q_i, \emptyset), \delta_B, \gamma_B)$~~ $\mathcal{D} \overset{def}{=} (\Sigma, 3^Q, (q_i, \emptyset), \delta_D, \gamma_D)$ is defined such that, when ~~$\rho: ((S, S'), a) \to (R, R')$~~ $\rho: ((S, S'), a$ then there are three cases:

1. if $R = \emptyset$, then $\delta_B((S, S'))$ is undefined (or, if a complete automation is preferred, maps to a rejecting sink),
2. else, if $R \neq R'$, then ~~$\delta_B((S, S'), a) \to (R, R')$~~ $\delta_B((S, S'), a) = (R, R')$ is a non-accepting transition, and $\gamma_d((S, S'), a)$ is undefined.
3. otherwise ~~$\delta_B, \gamma_B : \delta_B((S, S'), a) \to (R, \emptyset)$~~ $\delta_D((S, S') = \gamma_D((S, S'), a) = (R, \emptyset)$ is an accepting transition.

~~Breakpoint automata are insufficient to decide all GFM languages.~~ ??????? On the other hand, semi-deterministic automata decide superset of such language. We are going to define a few more transitions on top of breakpoint construction which allow us to construct slim automata that decide exactly the class of GFM languages.

2.4 Slim Automata Construction [separate chapter?]

~~Let us~~ Breakpoint automata constructed as presented in the previous section are not always equivalent to the input automaton.

In this section we define transitions $\gamma_w, \gamma_p : 3^Q \times \Sigma \to 3^Q$ 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 ~~$\gamma_p : ((S, S'), a) \to (\delta(S', a) \cup \gamma(S, a), \emptyset)$ and $\gamma_w : ((S, S'), a) \to (\delta(S', a), \emptyset$~~ $\gamma_p : ((S, S'), a) = (\delta(S', a) \cup \gamma(S, a), \emptyset)$ and $\gamma_w : ((S, S'), a) = (\delta(S', a), \emptyset)$

$\cancel{\mathcal{S} = (\Sigma, 3^Q, (q_i, \emptyset), \Delta_p, \Gamma_p)}$ $\mathcal{S} \overset{def}{=} (\Sigma, 3^Q, (q_i, \emptyset), \Delta_S, \Gamma_S)$ is slim, when $\cancel{\Delta_p}$ $\Delta_S = \Delta_D \cup \Gamma_p$ is set of transitions generated by $\cancel{\delta_b}$ δ_D and γ_p, and $\cancel{\Gamma_p}$ $\Gamma_S = \Gamma_D \cup \Gamma_p$ is set of accepting transitions, that is generated by $\cancel{\gamma_b}$ γ_D and γ_p. $L(\mathcal{S}) = L(\mathcal{A})$ (proof in text with original definition)

Alternatively, similarly defined using γ_w instead of γ_p, automaton $\cancel{\mathcal{W} = (\Sigma, 3^Q, (q_i, \emptyset), \Delta_w, \Gamma_w)}$ ~~is slim~~ $\mathcal{W} \overset{def}{=} (\Sigma, 3^Q, (q_i, \emptyset), \Delta_w, \Gamma_w)$ is slim a and $L(\mathcal{S}) = L(\mathcal{A})$. (no proof yet)

2.5 Slim Automaton Construction Generalized to TGBA

We want to construct a slim automaton from TGBA $\cancel{\mathcal{T} = (\Sigma, Q, q_i, \Delta, G)}$ $\mathcal{T} \overset{def}{=} (\Sigma, Q, q_i, \Delta, G)$. One possibility is to *degeneralize* $\mathcal{T}$ and to use previously mentioned algorithm in section 2.3. Another way is to extend slim automaton construction to TGBA.

extended slim construction 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~~ $\cancel{(:= \{0, 1, \ldots, |G| - 1\})}$ $levels \overset{def}{=} \{0, 1, \ldots, |G| - 1\}$ in the names of states. Let $|G|$ be number of ~~levels~~ $levels$ 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 iff 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

$up(x) \overset{def}{=} (x + 1) \bmod |G|$

$\cancel{P := 3^Q \times \text{levels}}$ ~~and~~ $\cancel{P_+ := 3^Q_+ \times \text{levels}}$ $P := 3^Q \times levels$ (?nepotrebuju and $P_+ := 3^Q_+ \times levels$)

We define γ_i similarly like γ, we just use Γ_i instead of Γ

$\cancel{\text{up}(x) = (x + 1) \bmod |G|}$

~~We~~ 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.

~~$\delta_R : P \times \Sigma \to P_+$ as $((S, S', i), a) \to (\delta(S, a), \delta(S', a) \cup \gamma_i(S, a), j)$~~

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

1. if $R = \emptyset$, then $\delta_B((S, S', i))$ is undefined,
2. else, if $R \neq R'$, then ~~$\delta_B : ((S, S', i), a) \to (R, R', i)$~~ $\delta_B : ((S, S', i), a) = (R, R', i)$ is a non-accepting transition,
3. otherwise ~~$\delta_B, \gamma_B : \delta_B((S, S', i), a) \to (R, \emptyset, \mathrm{up}(i))$~~ $\delta_B, \gamma_B : \delta_B((S, S', i), a) = (R, \emptyset, up(i$

$\gamma_p : P \times \Sigma \to P$

1. if $\delta(S', a) = \gamma_i(S, a) = \emptyset$, then $\gamma_p((S, S', i), a)$ is undefined, and
2. otherwise ~~$\gamma_p : ((S, S', i), a) \to (\delta(S', a) \cup \gamma_i(S, a), \emptyset, \mathrm{up}(i))$~~ $\gamma_p : ((S, S', i), a) = (\delta(S', a$

~~$\mathcal{S} = (\Sigma, P, (q_i, \emptyset, 0), \Delta_p, \Gamma_p))$~~ $\mathcal{S} \stackrel{def}{=} (\Sigma, P, (q_i, \emptyset, 0), \Delta_p, \Gamma_p))$ 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.

3 Implementation

My goal was to implement algorithm (hlavni zdroj) that takes TBA as an input and yields slim automata (2.4) inside a tool called *seminator*. Optional goal was to generalize the algorithm for TGBA input (2.5).

[mklokocka thesis] Seminator is implemented in C++ over the Spot library [24-mklokocka]. Seminator is a Linux command-line tool which can be run with the seminator command. There are several optional arguments that allow the user to express a preference for a certain type of output, the ability to turn off the post-optimizations provided by Spot, and several options for determining the behavior of the algorithm itself (whether to use Spot's degeneralization algorithms or run the algorithm on unmodified input, useful mostly for testing the tool). The tool expects the input automaton in the Hanoi Omega-Automata (HOA) format [25-mklokocka] on the standard input stream, but it can also read the input automaton from a file. To install we need install spot and to run "'autoreconf -i && ./configure && make"'.

3.1 Technologies

3.2 ~~Implementation inside~~ Create Slim Automata Using Seminator

Options Options to create slim automata in different ways

--slim to generate slim automata by, defaults to unoptimized, "strong" slim algorithm

--weak use "weak"-slim algorithm instead

--best try weak and strong, optimize outputs with spot and choose the one with smaller automaton [delete and use as default]

(add --strong to generate just automata just by strong slim algorithm) [not implemented yet] neither –weak nor –strong specified try both, optimize and choose smaller result

--via-tba transform input automaton to tba first

--via-tgba [not implemented] transform input automaton to tgba first neither --via-tba nor --via-tgba: try both options, choose smallest automaton

postprocess optimalizations [not implemented] should be used be as a default option, use an option to disable

Example Transform automaton.hoa to a slim automaton.

```
$ automaton.hoa | ./seminator --slim > slim.hoa
```

3.3 Implementation of Slim Automata inside Seminator

3.4 Testing and Verification

Implemented tests are basic, only language equivalence is checked. ltlcross and ltl2tgba tools are used. I use random ltls that were already generated, the ltls 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 with no optimalizations) is proved, as it follows construction from [main source] that is proved.

There should be another kind of tests - if slim automaton simulates the input automaton (so GFM property is kept) Subject following research,that is out of scope of this thesis, could be to verify if spot optimizations do not break this property. If the automata optimized by spot's emphautfilt tool break simulation property, then the results in following Evaluation chapter, which are built on such assumption, are pointless.

4 Evaluation

4.1 Alternative Algorithm

4.2 Different Implementation - ePMC

4.3 Semi-deterministic Automata

5 Conclusion