MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Tight Omega-automata

Master's Thesis

MAREK JANKOLA

Brno, Spring 2023

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Tight Omega-automata

Master's Thesis

MAREK JANKOLA

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

Department of Computer Science

Brno, Spring 2023

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

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.

Marek Jankola

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

Acknowledgements

Many thanks to my advisor Jan Strejček for his valuable insights and feedback on the thesis work. Further, I would like to thank the members of Infinists research group for their motivation and help. Lastly, I thank my family and Klára for being understanding and providing me with unending support.

Abstract

Tight automata are useful in providing the shortest counterexample in LTL model checking or in constructing a maximally satisfying strategy in LTL strategy synthesis. This thesis extends the formalism of tight state-based Büchi automata (BA) to the universal class of ω-automata, tight transition-based Emerson-Lei automata (TELA). We provide a new approach for transforming an arbitrary transition-based Büchi automaton (TBA) into a tight TBA. Furthermore, we adjust this approach to convert Rabin automata and show that the construction is very close to the optimal one concerning the rise in the number of states. We prove the correctness of all algorithms included in the work.

The work further contains an implementation of the new approach for converting BAs to tight BAs in a tool called Tightener. We show which existing techniques for reducing the state-space of an automaton do not violate tightness, and we use them for postprocessing of our resulting automata. The last part experimentally compares the new approach with existing tools.

Keywords

Büchi automata, Emerson-Lei automata, model checking, strategy synthesis, omega-automata, shortest counterexample, tight Büchi automata

Contents

1 Introduction

Techniques such as LTL model checking or LTL strategy synthesis take as an input a formal model of a system, for example, a Kripke structure and an LTL formula describing a requirement for the system. In both cases, the tool outputs an infinite word without loss of generality. It represents a counterexample in model checking (if the model violates the property) or a strategy in strategy synthesis.

When providing an LTL formula, the tool converts the formula into an ω-automaton and constructs the product with the model. If the product is represented explicitly, the tool explores the state space looking for an accepting run over a *lasso-shaped* word. Infinite words $w = uv^\omega$ are called lasso-shaped (ultimately periodic), and they can be represented finitely by their *stem* u and *loop* v. Observe that lasso-shaped word have multiple such finite representations, for instance $c(ab)^\omega = ca(ba)^\omega = cab(abab)^\omega$. Some applications [1, 2] require the lasso-shaped output to have the shortest stem and loop out of all possible outcomes. By that, a model checker can provide the shortest counterexample, which is more readable by the developer, or a strategy synthesiser can provide a maximally satisfying strategy.

Notice that if we represent runs of an automaton as a sequence of transitions, we can also talk about lasso-shaped runs/structures. The evident approach to this problem would be to explore the state space and provide the shortest accepting lasso-shaped structure. However, this approach is flawed. Let us assume that the accepting lasso-shaped run with the shortest stem and loop is run over word $cc(ab)^\omega$. The tool would provide $cc(ab)^\omega$ as the shortest output. However, there might be an accepting run with the stem and loop over $ccaaa(a)^\omega$, which has a shorter equivalent representation $cc(a)^\omega$.

Tight ω-automata is the class of ω-automata with the advantage that the tool can look for the shortest accepting lasso-shaped structure to provide the shortest finite representation of the model's lasso-shaped behaviour. In other words, each lasso-shaped word has an accepting lasso-shaped run with the same length of the shortest finite representation as the length of the shortest finite representation of the word.

1.1 Related work

Kupferman and Vardi [3] were the first to define this property for automata over finite words used to verify safety properties. Later Schuppan and Biere [1] extended this formalism to tight nondeterministic Büchi automata (BA), in other words, to automata over infinite words. Further, they proved that a translating algorithm from LTL to BA, called CGH [4], constructs tight BAs. As a part of their work, they extended CGH to work with past LTL operators.

The natural question of whether a formalism with greater expressive power than LTL (for instance, ω-automata) can be translated to tight automata arose. Schuppan [5] answered the question in his doctoral thesis by showing the problem of transforming an arbitrary BA into a tight BA is solvable.

Ehlers [6] set the upper bound of the rise of states to $2^{\mathcal{O}(n^2)}$ for the transformation of BAs. In our previous work [7], we improved the upper bound to $\mathcal{O}(n^{4n+6})$ and implemented the first practical tool that translates BAs to tight BAs as a part of the bachelor thesis.

1.2 Overview

This thesis discusses properties of the universal tight ω-automata and proposes new approaches to the problem of converting automata to tight automata. In Chapter 2, we set the necessary preliminaries. Chapter 3 extends the formalism of tight BA to the universal class of ω-automata (transition-based Emerson-Lei automata). It further studies properties of the newly defined class.

Chapter 4 discusses a new approach of converting BAs to tight BAs. It introduces all the problems faced during the construction and motivates each step of the algorithm. Further, it contains the proof of correctness and the analysis of the rise in the state space. The new approach naturally extends to converting Rabin automata to tight Rabin automata, which we present in Chapter 5. Whatsmore, we prove that the construction for Rabin automata is nearly optimal, and there is not much space for improvement. Chapters 6 and 7 present the practical optimizations, implementation details, and experimental results.

2 Preliminaries

This chapter recalls the basic notation and terminology used in this thesis.

2.1 Formal ω-languages

An *alphabet* Σ is an arbitrary non-empty finite set. A set of infinite words over Σ is *ω-Language*. Further, ω is an operator that can be applied to finite words or languages of finite words. It then denotes an infinite repetition of the word or an infinite concatenation of words from the language, respectively. For instance $(ab)^\omega = ababab\ldots$, $(abb)^\omega \in \{ab, b\}^\omega$.

2.2 Transition-based Emerson-Lei automata (TELA)

Here we define the type of ω-automata with universal acceptance condition. Any of Büchi, coBüchi, Street, and Rabin automata (and many more) can be seen as Emerson-Lei automata.

Definition 1. *A* nondeterministic transition-based Emerson-Lei automaton (TELA) $\mathcal{A}$ *is a tuple* $(Q, \Sigma, M, \delta, I, \varphi)$*, where*

- Q *is a finite set of* states*,*
- Σ *is a finite* alphabet*,*
- M *is a finite set of* acceptance marks*,*
- $\delta \subseteq Q \times \Sigma \times 2^M \times Q$ *is a transition relation,*
- $I \subseteq Q$ *is a set of* initial states*, and*
- φ *is an acceptance formula given by the following abstract syntax, with* m *ranging over* M*.*

$$\varphi \ ::= \ \mathsf{t} \mid \mathsf{f} \mid \mathsf{Inf}(m) \mid \mathsf{Fin}(m) \mid (\varphi \land \varphi) \mid (\varphi \lor \varphi)$$

An *run* of $\mathcal{A}$ over an infinite word $u = u_0u_1 \ldots \in \Sigma^\omega$ is an infinite sequence $\rho = (q_0, u_0, M_0, q_1)(q_1, u_1, M_1, q_2) \ldots \in \delta^\omega$ of consecutive transitions starting in an initial state $q_0 \in I$. By ρ_i, we denote the transition (q_i, u_i, M_i, q_{i+1}) from ρ. States q_i and q_{i+1} are the *start* and the *destination* state of the transition, respectively. Further, $\mathit{Inf}(\rho)$ denotes the set of marks that ρ visits infinitely often, and $\mathit{Fin}(\rho)$ is its complement. Every run satisfies t (true) and no run satisfies f (false). A run ρ satisfies $\mathsf{Inf}(m)$ if $m \in \mathit{Inf}(\rho)$, and satisfies $\mathsf{Fin}(m)$ otherwise. A run is *accepting* if it satisfies φ. An automaton *accepts* a word u if there exists an accepting run over this word. A *language* of automaton $\mathcal{A}$ is the set $L(\mathcal{A})$ of all words in Σ^ω accepted by $\mathcal{A}$. Automata $\mathcal{A}, \mathcal{B}$ are *equivalent* if $L(\mathcal{A}) = L(\mathcal{B})$.

In the following, *word* without any adjective refers to an infinite word. However, we work also with finite words. A *path* in $\mathcal{A}$ from a state q_0 to a state q_n over a finite word $r = r_0r_1 \ldots r_{n-1} \in \Sigma^*$ is a finite sequence $\sigma = (q_0, r_0, M_0, q_1) \ldots (q_{n-1}, r_{n-1}, M_{n-1}, q_n) \in \delta^n$ of consecutive transitions. We write that the path σ has the form $q_0 \xrightarrow{r}_{M'} q_n$ and that q_n is *reachable from* q_0 *over* r visiting the set of acceptance marks $M' = M_0 \cup M_1 \cup \cdots \cup M_{n-1}$. If $|M| = 1$, for clearer notation we write $q \xrightarrow{u}_m p$ instead of $q \xrightarrow{u}_{\{m\}} p$.

For a word or a run $u = u_0u_1 \ldots$, by $u_{i..}$ we denote its suffix $u_iu_{i+1} \ldots$ and by $u_{i,j}$, for $i < j$, we denote its infix $u_iu_{i+1} \ldots u_{j-1}$.

The thesis intensively works with *lasso-shaped* words and runs, which are sequences of the form $s.l^\omega$, where s is called a *stem* and $l \neq \epsilon$ is called a *loop*. Further, s is a *mininal stem* and l is a *minimal loop* of a lasso-shaped sequence $u = s.l^\omega$ if for each s', l' satisfying $u = s'.l'^\omega$ it holds $|s| + |l| \leq |s'| + |l'|$.

The minimal stem and the minimal loop of a lasso-shaped sequence u is denoted by $\mathit{minS}(u)$ and $\mathit{minL}(u)$, respectively. Moreover, $\mathit{minSL}(u)$ denotes the pair $(\mathit{minS}(u), \mathit{minL}(u))$ and we set $|\mathit{minSL}(u)| = |\mathit{minS}(u)| + |\mathit{minL}(u)|$.

2.3 Büchi and Rabin automata

In the thesis, we often work with certain subclasses of TELA since they have a simpler form of an acceptance condition, and we can construct smaller tight automata for them while preserving the same expressive power [8].

2.3.1 Büchi and generalized Büchi automata

Firstly, we define these automata with transition-based acceptance. Then we also introduce state-based acceptance because Schuppan's tightening construction [5] works with this representation. Further in the later chapters, we extend our algorithms to state-based acceptance, so we can compare easier.

- A *nondeterministic transition-based Büchi automaton* (*TBA*) $\mathcal{A}$ is a TELA $(Q, \Sigma, \{m\}, \delta, I, \mathsf{Inf}(m))$.
- A *nondeterministic transition-based generalized Büchi automaton* (*TGBA*) $\mathcal{A}$ is a TELA $(Q, \Sigma, \{m_0, m_1 \ldots, m_k\}, \delta, I, \bigwedge_{0 \le i \le k} \mathsf{Inf}(m_i))$.

Further, we define a state-based acceptance version of these automata.

- A *nondeterministic state-based Büchi automaton* (*BA*) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, I, F)$. Where:
  - Q, Σ and I are the same as in TELA,
  - $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, and
  - $F \subseteq Q$ is a set of accepting states.

  A run $\rho = (q_0, u_0, q_1)(q_1, u_1, q_2) \ldots \in \delta^\omega$ is accepting if there are infinitely many transitions (q, u, p) such that $q \in F$.
- A *nondeterministic state-based generalized Büchi automaton* (*GBA*) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, I, \mathcal{F})$. Where Q, Σ, δ, I are the same as in BA and $\mathcal{F} = \{F_0, F_1, \ldots, F_k\}$, such that for every $0 \le i \le k$, it holds that $F_i \subseteq Q$. A run ρ is accepting if, for each $0 \le i \le k$, there is a state $q \in F_i$, such that ρ visits q infinitely often.

Examples of all automata presented in this subsection are in Figure 2.1.

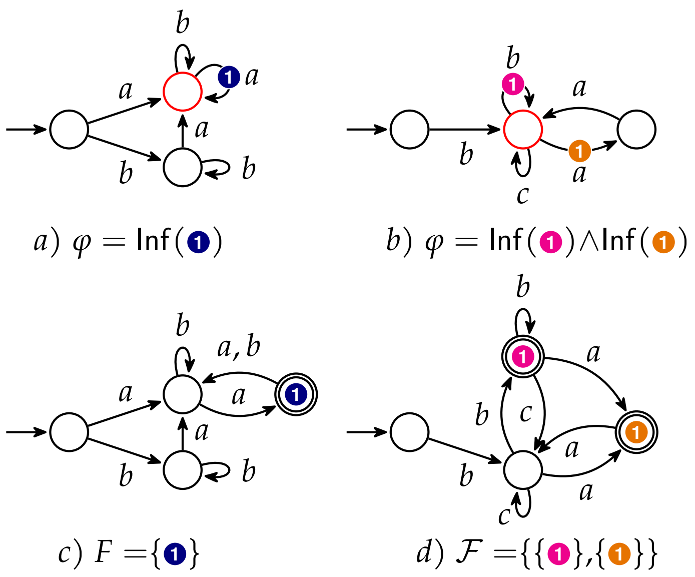

a) $\varphi = \mathsf{Inf}(❶)$
b) $\varphi = \mathsf{Inf}(❶) \wedge \mathsf{Inf}(❶)$
c) $F = \{❶\}$
d) $\mathcal{F} = \{\{❶\},\{❶\}\}$
a
b
1
c

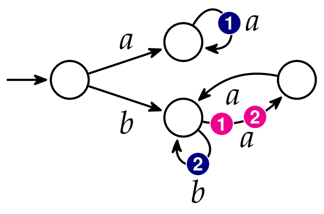
Figure 2.1: Automata in a) and c) are examples of TBA and BA, respectively. They both recognize the same language. The acceptance condition of a) is φ, and the set of accepting states of c) is F. The automata accept words which contain infinitely many symbols a. Often, translation of TBA to BA requires adding a new state for every marked transition. For example, we cannot just mark the red state from a) as accepting since the automaton would accept $a(b)^\omega$, which is not accepted by the TBA. Therefore, when using transition-based acceptance, we can work with smaller automata.
Similarly, automata in b) and d) are examples of TGBA and GBA. Again, they accept the same language and φ is the acceptance condition of the TGBA and $\mathcal{F}$ determines the sets of accepting states of the GBA. As in the former example, we can again work with smaller automata when using transition-based acceptance.

2.3.2 Rabin automata

As it turns out, it is straightforward to generalize the tightening algorithm presented in Chapter 4 for Büchi automata to Rabin automata. Moreover, it results in smaller state space. Furthermore, tools such as Rabinizer [9] can effectively construct a Rabin automaton from an LTL formula.

A *nondeterministic transition-based Rabin automaton* (*TRA*) $\mathcal{A}$ is a TELA $(Q, \Sigma, \{f_0, i_0, f_1, i_1 \ldots, f_k, i_k\}, \delta, I, \bigvee_{0 \leq j \leq k}(\mathsf{Fin}(f_j) \wedge \mathsf{Inf}(i_j)))$. Further, we naturally group the marks together and call each pair (f_j, i_j) a *Rabin pair*.

An example of TRA is in Figure 2.2.



$$\varphi = (\mathsf{Fin}(❶) \wedge \mathsf{Inf}(❶)) \vee (\mathsf{Fin}(❷) \wedge \mathsf{Inf}(❷))$$

Figure 2.2: In the figure, we present an example of TRA. The automaton must loop over a^ω or b^ω and cannot combine them in the infinite suffix.

3 Tight TELA

As mentioned, the tightness property was first defined for automata over finite words [3]. Further, it was extended to Büchi and generalized Büchi automata [1]. Tightness requires the existence of a specific run for every lasso-shaped word from the language accepted by a tight automaton. This requirement is independent of the acceptance method. Therefore, extending this property to automata with an arbitrary acceptance formula is straightforward.

Definition 2. *A TELA $\mathcal{A}$ is* tight *iff for each lasso-shaped word $u \in L(\mathcal{A})$ there exists an accepting lasso-shaped run ρ over u satisfying $|minSL(u)| = |minSL(\rho)|$. We call such runs* tight.

In our previous work [7], we have shown a few properties of lasso-shaped words and tight BA. For completeness, we introduce them here and adjust them to tight TELA if necessary.

Lemma 1. *Let u be a lasso-shaped word and $minSL(u) = (r, s)$. For every r', s' such that $u = r's'^{\omega}$, it holds that $r' = r$ and $s' = s$ if $|r| + |s| = |r'| + |s'|$.*

Proof. Given in [7]. □

In other words, for every lasso-shaped word u, $minS(u)$ and $minL(u)$ are unambiguous. Therefore, the relations *minS*, *minL*, *minSL* can be used as functions.

Another property we have shown is that there are some languages for which there is no tight BA with one initial state. This claim also holds for tight TELA. A similar counterexample as used in our previous work [7] is feasible with slightly different argumentation.

Lemma 2. *There exists a TELA $\mathcal{A}$, for which there does not exist any tight TELA $\mathcal{A}'$ with precisely one initial state and satisfying $L(\mathcal{A}) = L(\mathcal{A}')$.*

Proof. Consider a TBA (also a TELA) given by Figure 3.1. Let us assume there is a tight TELA $\mathcal{A}'$ with one initial state q_0 and satisfying $L(\mathcal{A}') = L(\mathcal{A})$. $\mathcal{A}'$ must accept a^{ω} and b^{ω}. Furthermore, since $|minSL(a^{\omega})| = |minSL(b^{\omega})| = 1$ and $\mathcal{A}'$ is tight, there must exist self-loops over a and b in q_0. However, $\mathcal{A}'$ then accepts for instance $a(b)^{\omega} \notin L(\mathcal{A})$, which is a contradiction. □

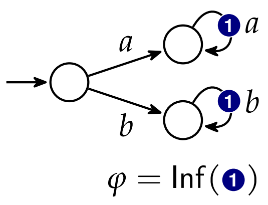


Figure 3.1: In the figure, there is a TELA $\mathcal{A}$ accepting language $\{a^\omega, b^\omega\}$.

3.1 Boundaries for the repetition of the minimal loop

As the (un)tightness of an automaton depends purely on lasso-shaped words accepted by the automaton and the corresponding accepting runs, we turn our attention to these words. The completeness and efficiency of our tightening algorithm intensely depend on the possible number of repetitions of the minimal loop in the smallest lasso-shaped run over a lasso-shaped word. In our previous work, we have shown that in BA, for each lasso-shaped word, there is a lasso-shaped run that, in each period, repeats the minimal loop of the word at most n times (see Lemma 2 [7]). Lemma 3 below extends this result to TELA.

We introduce a notion of *significant* states that are used to detect infinite suffices of a lasso-shaped run ρ over a repetition of a finite word.

Definition 3. *Let $\rho = (q_0, u_0, M_0, q_1)(q_1, u_1, M_1, q_2)\ldots$ be a lasso-shaped run over a lasso-shaped word $u = wv^\omega$. We say that a start state q_i at a position i is v-*significant *iff $\rho_{i..}$ is a run over v^ω.*

In other words, a sequence of v-significant states starts after reading a finite prefix w by some lasso-shaped run and between each consecutive pair of them there is the finite word v. The definition is illustrated in Figure 3.3. A scheme figure for Lemma 3 is in Figure 3.2.

Further, let ρ be a run $\rho = \rho_0\tau\rho_1$, we say that a finite path τ *induces* v-significant state q at a position i if $\tau_i = (q_i, a_i, M_i, q_{i+1})$ and $q = q_i$. In particular, if $\tau = (q_1, a_1, M_1, q_2)\ldots(q_n, a_n, M_n, q_{n+1})$, then τ does not induce q_{n+1} even if it is a v-significant state.

Lemma 3. *Let $\mathcal{A}$ be a TELA with n states and let M be its set of acceptance marks, without loss of generality $|M| > 0$. Then for each lasso-shaped word*

$wv^\omega \in L(\mathcal{A})$ there exist $0 \leq x \leq |M|+1$ states $s, l_1, \dots, l_x$ and paths $\kappa, \pi, \tau_1 \dots \tau_x$, such that

- *κ has the form $q_I \xrightarrow{w}_{M_\kappa} s$ for some initial state q_I,*
- *π has the form $s \xrightarrow{v^i}_{M_\pi} l_1$ for some $0 \leq i \leq n$, further*
- *for each $0 < y < x$, τ_y has the form $l_y \xrightarrow{v^{j_y}}_{M_y} l_{y+1}$ for some $j_y > 0$,*
- *τ_x has the form $l_x \xrightarrow{v^{j_x}}_{M_x} l_1$ for some $j_x > 0$, and*
- *$j_1 + j_2 + \dots + j_x \leq (n-i) \cdot |M|$.*

Further, the lasso-shaped run $\kappa\pi(\tau_1\tau_2 \dots \tau_x)^\omega$ is an accepting run over wv^ω.

Proof. Let $wv^\omega \in L(\mathcal{A})$ be a lasso-shaped word and let $k = |w|$ and $o = |v|$. Further, let $\rho = (q_0, u_0, M_0, q_1)(q_1, u_1, M_1, q_2)\dots$ be an accepting run of $\mathcal{A}$ over vw^ω. We focus on v-significant states of this run, i.e. $q_k, q_{k+o}, q_{k+2o}, \dots$ reached by the run after reading $w, w.v, w.v^2, \dots$ as depicted in Figure 3.3.

There exists an index c such that $\rho_{c..}$ contains precisely marks from $Inf(\rho)$ and accordingly, it does not contain marks $Fin(\rho)$. Further, there exist two other indices $c < c_0 < c_1$, such that ρ_{c_0,c_1} contains all marks from $Inf(\rho)$ and it starts and ends with the same v-significant state. This implies that $\rho_{0,c_0}(\rho_{c_0,c_1})^\omega$ is an accepting lasso-shaped run over wv^ω.

We can now divide such a lasso-shaped run into a few parts: $\kappa, \pi, \tau_1, \dots, \tau_x$ and shorten it while preserving that it is still an accepting run over wv^ω. Firstly, let κ be the part of ρ over w. In other

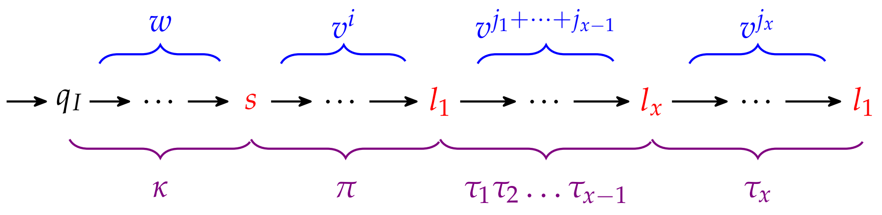


Figure 3.2: A picture illustrates a decomposition of a lasso-shaped run $\rho = \kappa\pi(\tau_1\tau_2\dots\tau_x)^\omega$ over a lasso-shaped word wv^ω given in Lemma 3.

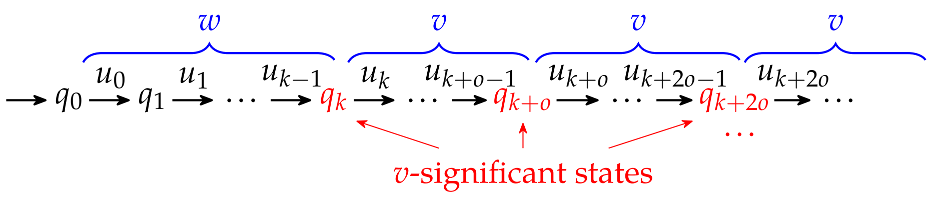


Figure 3.3: A picture illustrates a run ρ over a lasso-shaped word $u = u_0u_1\ldots$. Red colour highlights v-significant states.

words, it is a path of the form $q_I \xrightarrow{w}_{M_\kappa} s$, where q_I is an initial state, and s is a state of $\mathcal{A}$. Further, let π be a path of the form $s \xrightarrow{v^i}_{M_\pi} l_1$. Let l_1 be the start state of the transition at the index c_0. We can shorten π so that $i \leq n$. From the perspective of acceptance, it does not matter whether π contains any acceptance marks. Therefore, if π induces two same v-significant states, we can simply exclude the path between them, i.e., $i \leq n$.

Further, we construct and shorten the division $\tau_1\tau_2\ldots\tau_x$ of the loop ρ_{c_0,c_1}. For each $0 < y < x$, we construct τ_y to be a path of the form $l_y \xrightarrow{v^{j_y}}_{M_y} l_{y+1}$ for some $j_y > 0$. Moreover, we put another requirement to τ_y. We require the part between the last v-significant state of τ_y and l_{y+1} to contain a mark from $\mathit{Inf}(\rho)$ that is not contained by any previous τ_i (Figure 3.4). Since, $|\mathit{Inf}(\rho)| \leq |M|$, then number of parts in such division is at most $|M|$, i.e. $x \leq |M| + 1$. In the perspective of satisfying the acceptance condition (seeing every state from $\mathit{Inf}(\rho)$), only the

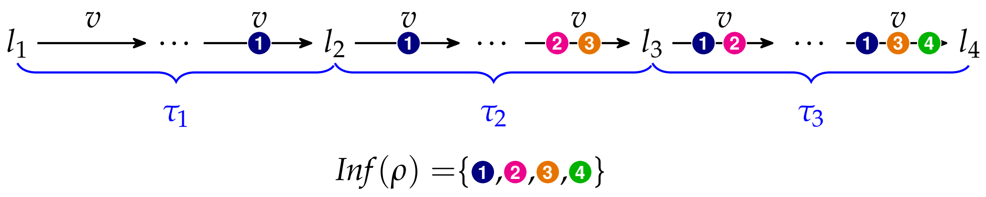


Figure 3.4: The picture illustrates the second constraint given to τ_y. In this case we divide the loop into $\tau_1, \tau_2, \tau_3, \tau_4$, where τ_y ($y < 4$) corresponds to the third type of division from the lemma. Notice that the last period of v in every τ_y contains a new mark from $\mathit{Inf}(\rho)$ that is not contained by any previous τ.

path between the last significant state and l_{y+1} matters. Therefore we can consider that τ_y does not induce two same v-significant states. Otherwise, the path between them can be omitted. This implies $j_y \leq n$.

Finally, let τ_x be the path of the form $l_{x-1} \xrightarrow{v^{j_x}}_{M_x} l_1$. Because of the same argument as in the paragraph with π, $j_x \leq n$. Thanks to the argumentation in this and previous paragraphs, we know that $j_1 + \cdots + j_x \leq n \cdot x$.

The last part left to show is that this boundary for $j_1 + \cdots + j_x$ can be lowered to $(n-i) \cdot |M|$. The bound naturally comes after showing that without loss of generality, we can assume that the set of v-significant states that π induces is disjoint with the set of induced v-significant states by $\tau_1 \ldots \tau_x$. Let us assume, that π and some $\tau_{y'}$ induce the same v-significant state q. Let us divide $\tau_{y'}$ into two parts: $l_{y'} \xrightarrow{v^g}_{M_{y'_1}} q$ and $q \xrightarrow{v^h}_{M_{y'_2}} l_{y'+1}$. Let us denote these paths as σ_0 and σ_1. Accordingly, let us divide π into two parts: $s \xrightarrow{v^{g'}}_{M_{\pi_1}} q$ and $q \xrightarrow{v^{h'}}_{M_{\pi_2}} l_1$. Let us denote these parts as v_0 and v_1. The run $\kappa.v_0.v_1.\tau_1 \ldots \tau_{y'-1}.\sigma_0(\sigma_1.\tau_{y'+1} \ldots \tau_x.\tau_1 \ldots \sigma_0)^\omega$ is clearly an accepting run over wv^ω. Since both v_1 and σ_1 start with q, we can omit the part $v_1.\tau_1 \ldots \tau_{y'-1}.\sigma_0$ and get an accepting run $\kappa.v_0(\sigma_1.\tau_{y'+1} \ldots \tau_x.\tau_1 \ldots \sigma_0)^\omega$ over u. Thus, without loss of generality $j_1 + \cdots + j_x \leq (n-i) \cdot x = (n-i) \cdot (|M|+1)$.

Finally, we can push the boundary to $(n-i) \cdot |M|$ by showing that τ_1 and τ_x are inducing disjoint sets of v-significant states. Let us assume that there is a v-significant state q that is both induced by τ_1 and τ_x. As previously, we can divide the run into $\kappa.\pi(v_0.v_1.\tau_2 \ldots \tau_{x-1}\sigma_0.\sigma_1)^\omega$. Since v_1 starts with q and σ_0 ends with q, we can shorten the paths to $\kappa.\pi.v_0(v_1.\tau_2 \ldots \tau_{x-1}\sigma_0)^\omega$ obtaining an accepting run over wv^ω. The last paragraph implies the boundary $j_1 + \cdots + j_x \leq (n-i) \cdot |M|$. $\square$

The following claim sets an important upper boundary and is a direct result of Lemma 3.

Claim 1. *Let $\mathcal{A}$ be a TELA and $u \in L(\mathcal{A})$ be a lasso-shaped word. There exists a lasso-shaped accepting run $\rho = \kappa\pi(\tau)^\omega$ over u in $\mathcal{A}$. Further, let n be the number of minL(u)-significant states induced by π and τ, then $n \leq |M| \cdot |Q|$.*

Proof. Since $u = minS(u)minL(u)^\omega$, the existence of the lasso-shaped run ρ directly follows as a side result from Lemma 3. Furthermore, $|\pi\tau| \leq |minL(u)|^{k+i}$, where $k \leq (|Q| - i) \cdot |M|$. Therefore, $k + i \cdot |M| \leq |Q| \cdot |M|$, since $|M| > 0$ (assumption of Lemma 3) $k + i \leq k + i \cdot |M| \leq |Q| \cdot |M|$ holds.

The last thing that needs to be shown is that $\pi\tau$ does not induce more than $k + i$ $minL(u)$-significant states. Let $s_1, s_2, \dots s_i, l_1, \dots l_k$ be the $minL(u)$-significant states induced by $\pi\tau$, such that

$$\pi = s_1 \xrightarrow{minL(u)} s_2 \xrightarrow{minL(u)} \dots \xrightarrow{minL(u)} s_i \xrightarrow{minL(u)} l_1$$

$$\tau = l_1 \xrightarrow{minL(u)} l_2 \xrightarrow{minL(u)} \dots \xrightarrow{minL(u)} l_k \xrightarrow{minL(u)} l_1$$

Let us assume that there is another induced $minL(u)$-significant state q. Without loss of generality, $l_j \xrightarrow{w} q$ and $q \xrightarrow{v} l_{j+1}$ are in $\pi\tau$ with $v, w \neq \epsilon$ and $wv = minL(u)$. Since q, l_j and l_{j+1} are $minL(u)$-significant states, $\pi\tau$ continues over $minL(u)$ from them, therefore $vw = wv = minL(u)$. Without loss of generality, let $|v| > |w|$, then w is a prefix of v. Therefore, $minL(u)^\omega = w^\omega$, which is a contradiction with $minL(u)$ being minimal. □

The next chapter reveals that the better the upper boundary we have on the number of $minL(u)$-significant states in the shortest lasso-shaped run over each lasso-shaped word, the smaller set of states our tightening algorithm constructs. Although the currently proposed upper boundary $|M| \cdot |Q|$ might seem high, Lemma 4 shows it is tight.

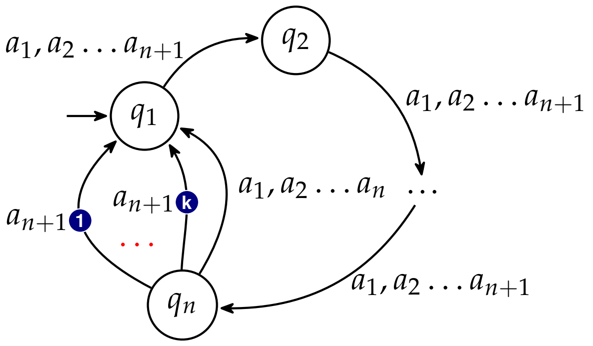


Figure 3.5: TELA $\mathcal{A}_{n,k}$, with $(a_1 a_2 \dots a_{n+1})^\omega \in L(\mathcal{A}_{n,k})$. The red dots represent the $|M| = k$ transitions over a_{n+1} each marked with a different mark.

In the proof of the following lemma, we present a TELA that accepts the word $(a_1a_2\dots a_{n+1})^\omega$ and for which every accepting lasso-shaped run has at least $|Q|\cdot|M|$ $minL(u)$-significant states induced by its minimal stem and loop.

Firstly, let us propose the following construction. Let $n,k\geq 1$, then TGBA $\mathcal{A}_{n,k} = (\{q_1,\dots,q_n\},\{a_1,a_2,\dots,a_{n+1}\},\{m_1,\dots,m_k\},\delta,\{q_1\},\varphi)$, where $\varphi = \bigwedge_{1\leq i\leq k}\mathsf{Inf}(m_i)$ and δ is given by Figure 3.5.

Lemma 4. *For each $n,k\geq 1$, there is a lasso-shaped word $u\in L(\mathcal{A}_{n,k})$, such that every accepting lasso-shaped run $\rho=\kappa\pi(\tau)^\omega$ of $\mathcal{A}_{n,k}$ over u satisfies that $\pi\tau$ induces at least $|M|\cdot|Q|$ $minL(u)$-significant states.*

Proof. Constructed $\mathcal{A}_{n,k}$ clearly accepts $(minL(u))^\omega = (a_1a_2\dots a_{n+1})^\omega$. An accepting run $\rho=\tau^\omega$ over u with the minimal $|\tau|$ is of the following form. It reads $minL(u)$ from q_1 and gets to q_2. The only option is then to again read $minL(u)$ from q_2 and get to q_3. After reading $n-1$ times $minL(u)$ it gets to q_n. From there, it can see only one of the accepting marks while reading $minL(u)$. Hence it takes precisely n repetitions of $minL(u)$ to be able to again take a transition over a_1 from q_1. After this path of length $|minL(u)|\cdot n$ only one accepting mark can be seen. Therefore, we need to concatenate $|M|$ such paths (each seeing a different mark) to accept this word. Therefore $|minL(\rho)| = |minL(u)^{n\cdot|M|}|$ and it is easy to see, that τ induces at least $|Q|\cdot|M|$ $minL(u)$-significant states. $\square$

The previously stated properties were formulated for TELA. However, for completeness of the tightening algorithms that we present in the next chapters, we need one more property that we formulate only for TRA. We use it also for TBA, since TBA can be seen as TRA with only on Rabin Pair (f,i) (and none transition is marked by f).

Let $q_0,q_1,\dots,q_k$ be a sequence of states. We say that they are *pairwise different* if for each $0\leq i,j\leq k$, it holds that if $i\neq j$, then $q_i\neq q_j$. In other words, all the states are unique.

Lemma 5. *Let $\mathcal{R}$ be a TRA. For every lasso-shaped word $u\in L(\mathcal{R})$, there is an accepting lasso-shaped run $\rho=\kappa\pi(\tau)^\omega$ in $\mathcal{R}$, such that κ,π and τ are paths over $minS(u), minL(u)^i$ and $minL(u)^k$, respectively.*

Furthermore, $k+i\leq|Q|$ and for every finite word v, all v-significant states that $\pi\tau$ induces are pairwise different.

Proof. A lasso-shaped run is accepting in $\mathcal{R}$, if it satisfies at least one $\mathsf{Fin}(m_f) \wedge \mathsf{Inf}(m_i)$. For that, it needs to have m_i in its loop. Therefore, similarly as in the proof of Lemma 3, there is an accepting run ρ' over u and indices c_0, c_1 such that $\rho = \rho'_{0,c_0}(\rho'_{c_0,c_1})^\omega$ is accepting over u and ρ'_{c_0,c_1} contains some m_i. If we now shortened ρ as in the proof of Lemma 3 we would need to keep only m_i in the loop, therefore, we can set $x = 2$ and there are paths $\kappa, \pi, \tau_1, \tau_2$, such that $\rho = \kappa\pi(\tau_1\tau_2)^\omega$. Furthermore κ, π and $\tau_1\tau_2$ are paths over $minS(u), minL(u)^i$ and $minL(u)^k$, where $i + k \leq |Q|$. We further denote $\tau_1\tau_2$ by τ.

Let us show the rest of the lemma by contradiction. Assume, that there are two indices $|\kappa| \leq i' < j' < |\kappa\pi\tau|$ and $q_{i'} = q_{j'}$ are v-significant states induced by $\pi\tau$. Let us consider two cases:

- They are both in π or there is no acceptance mark in $\tau_{i',j'}$. We can omit the path between $q_{i'}$ and $q_{j'}$ and obtain $\rho'' = \rho_{0,i'}\rho_{j'..}$, an accepting run over u does not inducing $q_{i'}$.
- They are both in τ and there is a acceptance mark in $\tau_{i',j'}$. We can shorten τ into $\tau' = \tau_{i',j'}$. It does not induce $q_{j'}$ and $\rho'' = \rho_{0,i'}(\rho_{i',j'})^\omega$ is an accepting run over u.

□

4 Tightening of TBA

There are a few already known approaches for the transformation of BAs to tight BAs. The doctoral thesis of Schuppan [5] presents the first prescription. It is not an algorithm per se. Therefore, it does not make sense to analyze its complexity. It transforms GBA to tight GBA. The algorithm in this thesis for simplicity works with TBA, but one can extend it to TGBA accordingly.

Another difference is that we work with explicit automaton representation rather than symbolic one. The advantage is that it enables tight automata to be used in explicit-state model checking [10] without exhaustive translation of symbolic representation to explicit one. Furthermore, authors of maximally satisfying strategy synthesis [2] assume an explicit tight BA on the input, again advantageous to our translating algorithm.

Ehlers [6] presented another approach to the tightening of BA. His work is rather theoretical and sets an upper bound on the number of states for the construction. He combines two other constructions with his final modification of the automaton to make it tight. His approach uses a technique by Clabrix et al. [11] to create a finite automaton accepting $u\$v$ iff the original automaton accepted uv^ω. Further, a technique by Farzan et al. [12] is used to convert the finite automaton back to Büchi (extracting \$ from the alphabet) together with Ehlers modifications to keep the accepting loops tight.

Ehlers claims that the raise of the number of states of the tight automaton can be bounded by $2^{O(n^2)}$, where n is the number of states from the original automaton.

Finally, our previous approach to this problem was an algorithm for translating explicit-state BAs to tight BAs, presented in the bachelor thesis [7]. Regarding the size of the state space, we claim there is $\mathcal{O}(n^{4n+6})$ raise in the worst case. The idea of the algorithm is also complex and requires a comprehensive description. Although the state complexity is already better than Ehlers's upper bound estimation, this chapter presents a new and even more efficient algorithm with a more concise description.

4.1 Intuition

This section aims to introduce the main intuition behind the tightening algorithm. Firstly, we show how we use Lemma 5 to ensure the completeness of the construction. After that, we present three issues with the simple approach and how we fix them in the final algorithm. Once a reader has the motivation for every logical step, the last subsection introduces an informal description of the final algorithm.

We work with TBA. However, the construction can be easily adjusted to state-based acceptance. To make a TBA tight, we must ensure that for each lasso-shaped word u accepted by the input automaton, there exists an accepting run ρ inducing precisely one $minL(u)$-significant state in its minimal stem and loop.

Let $\mathcal{B}$ be a TBA, and $u \in L(\mathcal{B})$ be a lasso-shaped word. From Lemma 5, there is an accepting lasso-shaped run $\rho = \kappa\pi(\tau)^\omega$ over u of the following form

$$q_0 \xrightarrow{minS(u)} s_1 \xrightarrow{minL(u)} s_2 \cdots s_i \xrightarrow{minL(u)} (l_1 \xrightarrow{minL(u)} l_2 \cdots l_k \xrightarrow{minL(u)} l_1)^\omega$$

where $i + k \leq |Q|$. Furthermore, states $s_1, \ldots, s_i, l_1, \ldots, l_k$ are pairwise different since they are $minL(u)$-significant and they are induced by $\pi\tau$.

The algorithm aims to construct a new state for every permutation of such states and create a loop over $minL(u)$ in the new state. In other words, ensure the existence of a run of the following form

$$q_0 \xrightarrow{minS(u)} (s_1, s_2 \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{minL(u)} s_1, s_2, \ldots, s_i[l_1, \ldots, l_k])^\omega$$

An evident approach to cover all finite words (and by that cover also $minL(u)$) between states $s_1, s_2, \ldots, l_k$ would be constructing finite automata accepting all finite words over which we can get from s_1 to s_2, from s_2 to s_3, etc. and then do a parallel composition. That is an idea of our previous approach from the bachelor thesis [7]. In this thesis, we dig a bit more into the composition of such automata and show that it is not necessary to construct the whole state space of the product automaton representing the parallel composition.

4.1.1 Composition of significant states

Let us have a TBA with a set of states Q. For every pairwise different sequence of states $s_1, \dots, s_i, l_1, \dots, l_k \in Q$, and $i + k \leq |Q|$, we create a new *composite state* labelled by $s_1, s_2, \dots, s_i[l_1, \dots, l_k]$. Before squared brackets, there are *stem states*, and in the brackets, there are *loop states*.

The idea is that this state would represent a composition of the original states. In other words, if there are transitions:

$$s_1 \xrightarrow{a} r_1, s_2 \xrightarrow{a} r_2, \dots s_i \xrightarrow{a} r_i, l_1 \xrightarrow{a} t_1 \dots, l_k \xrightarrow{a} t_k$$

then in the resulting automaton, we construct a transition:

$$s_1, s_2, \dots, s_i[l_1, \dots, l_k] \xrightarrow{a} r_1, r_2, \dots, r_i[t_1, \dots, t_k]$$

Notice that constructing only transitions where the sequence of states $r_1, r_2, \dots, r_i, t_1, \dots, t_k$ is also pairwise different is correct. It follows from Lemma 5. Consider a lasso-shaped word u accepted by a TBA $\mathcal{B}$, such that $minL(u) = av$, then there is a run $\rho = \kappa\pi\tau^\omega$ in $\mathcal{B}$ with the properties from the lemma. After making synchronously a transition over a from $s_1, \dots, s_i, l_1, \dots, l_k$, we reach $r_1, \dots, r_i, t_1, \dots, t_k$, that are va-significant states induced by $\pi\tau$. Therefore, by the lemma, they are pairwise different. By this observation, we spare a lot of states since it is correct to construct only composite states. An example of this construction is in Figure 4.1.

A reader might now recall Claim 1 and Lemma 4. They provide the tight bound on the number of $minL(u)$-significant states in a lasso-shaped run. From the idea behind the composition, one can see this

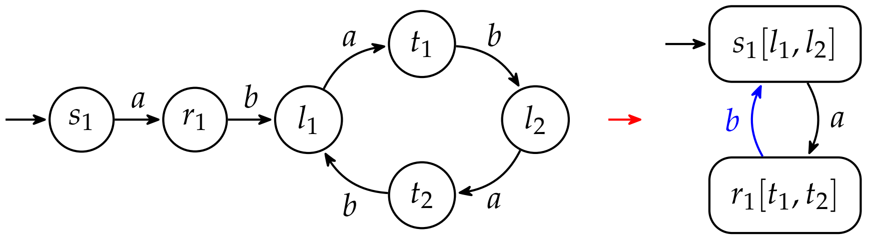


Figure 4.1: An example of the composition of possible significant states. The blue transition is further discussed in the later subsections. Notice, that in the component on the right, there is a tight run over $(ab)^\omega$.

is also an upper bound on the number of composite states possibly needed to cover all lasso-shaped words. The bound is $|M| \cdot |Q|$, but TBAs have $|M| = 1$. Therefore, in the worst case, we must create a composite state from every variation of at most $|Q|$ pairwise different states.

4.1.2 The problem of enclosing a loop

In this part of the chapter, the idea of the algorithm might start to crystallize. We present three issues that need to be considered during the construction. First, we show that building a simple parallel synchronous composition is not sufficient.

Consider a TBA from Figure 4.1. There are transitions $s_1 \xrightarrow{a} r_1, l_1 \xrightarrow{a} t_1, l_2 \xrightarrow{a} t_2$, so we create a composite states $s_1[l_1, l_2]$ and $r_1[t_1, t_2]$, and add transition $s_1[l_1, l_2] \xrightarrow{a} r_1[t_1, t_2]$. If we simply do the same for the transitions over b, we would add transition $r_1[t_1, t_2] \xrightarrow{b} l_1[l_2, l_1]$, but $l_1[l_2, l_1]$ is not a composite state. We would never obtain the *tight* loop. However, states like $l_1[l_2, l_1]$ are helpful because they might signal that we need to enclose the loop.

States with the same last stem state and last loop state might signal that we moved by an entire minimal loop of some word. Formally, states $s_1, \ldots, l_1[l_2, \ldots, l_1]$ are called *enclosing* states.

Further, the *rotated version* of a composite state $s_1, s_2, \ldots, s_i[l_1, \ldots, l_k]$ is the enclosing state $s_2, \ldots, l_1[l_2, \ldots, l_1]$. Or if the composite state does not have a stem, i.e. it has form $[l_1, l_2 \ldots, l_k]$, then the *rotated version* is the composite state $[l_2, \ldots, l_k, l_1]$. Notice that every composite state has precisely one rotated version.

We fix the presented issue by the following condition that we call *enclosing of a loop*. Consider that a composite state $s_1, \ldots, s_i[l_1, \ldots, l_k]$ can reach a composite state $r_1, \ldots, r_i[t_1, \ldots, t_k]$. If a transition from $r_1, \ldots, r_i[t_1, \ldots, t_k]$ to the rotated version of $s_1, \ldots, s_i[l_1, \ldots, l_k]$ should be added, we lead the transition also to $s_1, \ldots, s_i[l_1, l_2, \ldots l_k]$.

This step can be illustrated by the example in Figure 4.1. At some point, we should add a transition from $r_1[t_1, t_2]$ to $l_1[l_2, l_1]$ over b. However, our algorithm notices that $l_1[l_2, l_1]$ is the rotated version of $s_1[l_1, l_2]$. It further checks that $r_1[t_1, t_2]$ is reachable from $s_1[l_1, l_2]$. Therefore, it creates the blue transition to $s_1[l_1, l_2]$. Naturally, it does not add $l_1[l_2, l_1]$ to the constructed component since it is an enclosing state (it is not a composite state).

4.1.3 The problem of accepting transitions

Consider a TBA in Figure 4.2. We construct the composite states the same way as in the previous subsection. The question is, how to correctly mark the transitions so that we accept $(\{c\}^*.\{a\}.\{b\})^\omega$ with *tight* loops and do not add any new behaviour that was not in the original automaton.

A naive approach to the problem would be to mark the transitions for which there is at least one marked transition between the loop states in the original automaton. For instance, when constructing transition $s_1[l_1, l_2] \xrightarrow{a}_m r_1[t_1, t_2]$, we mark it, because there is a mark on transition $l_1 \xrightarrow{a}_m t_1$ and both l_1, t_1 are loop states. However, this approach is incorrect since we would need to mark the self-loop in $s_1[l_1, l_2]$ over c as depicted in Figure 4.2. The resulting TBA would then accept c^ω, which is not accepted by the original TBA on the left.

The problem is that $s_1[l_1, l_2]$ can never reach its rotated version over c^ω. During the construction, $s_1[l_1, l_2]$ can get over c only back to itself, not to $l_1[l_2, l_1]$. We can only add an accepting transition when enclosing a loop to ensure that once a run passes the mark, it could reach the rotated version of the state where it started.

We fix the problem in the following way. During the construction of the parallel composition, when the following transitions exist in the

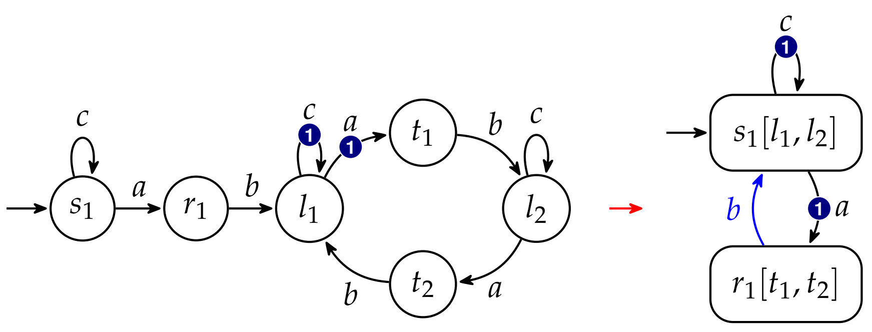


Figure 4.2: An example of an incorrect marking of the accepting transitions.

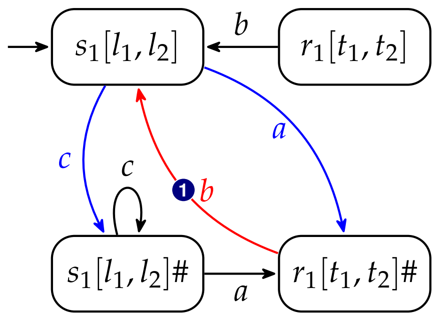


Figure 4.3: Constructing a TBA from the left TBA from Figure 4.2 with the presented fix and correct marking.

original automaton

$$s_1 \xrightarrow{a} r_1, s_2 \xrightarrow{a} r_2, \ldots, s_i \xrightarrow{a} r_i, l_1 \xrightarrow{a} t_1 \ldots l_j \xrightarrow{a}_m t_j, \ldots l_k \xrightarrow{a} t_k$$

we add the following transition in the constructing component

$$s_1, s_2, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{a} r_1, r_2, \ldots, r_i[t_1, \ldots, t_k]\#$$

The composite states with # represent that we have seen an accepting mark. Further, we only add a marked transition when enclosing a loop, and at least one of the following conditions is satisfied: We are in a # state, or there is an accepting mark between the loop states. We demonstrate the fixed construction from Figure 4.2 in Figure 4.3.

The blue transitions from $s_1[l_1, l_2]$ are added to # copies, because transitions $l_1 \xrightarrow{c}_m l_1$ and $l_1 \xrightarrow{a}_m t_1$ are marked. The red transition is added when enclosing the loop from the # state. It is marked because it leads from # copy. Therefore, when a potential run sees the mark it could reach the rotated version of $s_1[l_1, l_2]$. Notice that the constructed automaton does not accept the same language as the left TBA from Figure 4.2, which is fine since the goal was to build a tight loop for words from $(\{c\}^*.\{a\}.\{b\})^\omega$ and do not add any new behaviour.

4.1.4 The problem of overlapping loops

We first demonstrate the problem by the example in Figure 4.4. The approach that we presented in Subsection 4.1.3 results in the automaton presented in Figure 4.5, when processing $[l_1, l_2, l_3, l_4]$. The

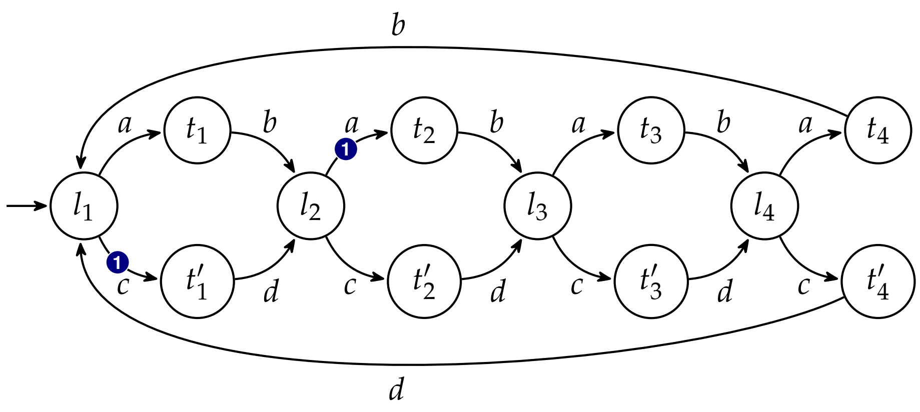


Figure 4.4: The example displays a TBA for which the approach from Subsection 4.1.3 is not correct.

blue transitions over a, b from $[l_1, l_2, l_3, l_4]$ are added to $[t_1, t_2, t_3, t_4]$# and $[t'_1, t'_2, t'_3, t'_4]$#, because transitions $l_2 \xrightarrow{a}_m t_2$ and $l_1 \xrightarrow{c}_m t'_1$ are marked. The red transitions are constructed when enclosing the loops because $[t_1, t_2, t_3, t_4]$# and $[t'_1, t'_2, t'_3, t'_4]$# can reach the rotated version of $[l_1, l_2, l_3, l_4]$ over b and d.

A component from Figure 4.5 (let us refer to it as $\mathcal{A}$) accepts $(abcd)^\omega$, which is not accepted by the original automaton $\mathcal{B}$. The problem is that when taking a path in $\mathcal{B}$ over ab, we get from l_1 to l_2. Further, if we take a path from l_2 over cd, we get to l_3. This repeats when we return to l_1 after another iteration over $abcd$. However, the blue tran-

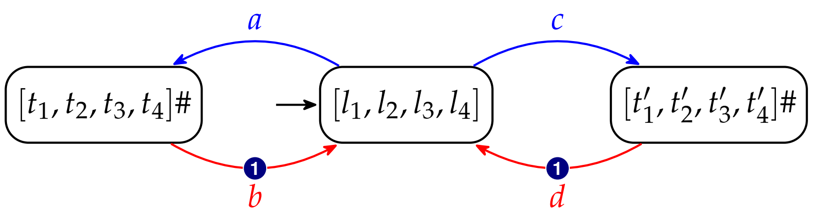


Figure 4.5: Constructing a component of TBA for a composite state $[l_1, l_2, l_3, l_4]$ from the TBA given in Figure 4.4 using approach from Subsection 4.1.3

sitions in $\mathcal{A}$ lead into # copies because of the accepting transitions $l_1 \xrightarrow{c}_m t'_1$ and $l_2 \xrightarrow{a}_m t_2$ that are never taken.

The core of the issue is that when finishing an iteration over *abcd* in $\mathcal{A}$, the path goes through two accepting transitions, each with a mark caused by a different marked transition in $\mathcal{B}$. In other words, the path over *abcd* contains transition $[t_1, t_2, t_3, t_4]\# \xrightarrow{b}_m [l_1, l_2, l_3, l_4]$ that is marked because of $l_2 \xrightarrow{a}_m t_2$ and transition $[t'_1, t'_2, t'_3, t'_4]\# \xrightarrow{d}_m [l_1, l_2, l_3, l_4]$ that is marked because of $l_1 \xrightarrow{c}_m t'_1$.

More generally, assume we constructed the following accepting loop using the approach described in the previous subsection

$$\tau = s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{u}_m s_1. \ldots, s_i[l_1, \ldots, l_k]$$

The intuitive algorithm from subsection 4.1.3 would be correct if τ were *associated* with some transition $t = s'_1, \ldots, s'_i[l'_1, \ldots, l'_k] \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]\#$. Meaning that for every occurrence of a marked transition in τ, there is an occurrence of t before with no other marked transition in between.

More formally, for every $j \leq |\tau|$, if $\tau_j = r'_1, \ldots, r'_i[t'_1, \ldots, t'_k]\# \xrightarrow{a}_m r''_1, \ldots, r''_i[t''_1, \ldots, t''_k]$, then there is $i \leq j$, where $\tau_i = t$ and $\tau_{i,j}$ does not contain a marked transition.

In the following paragraph, we intuitively explain why we want every loop associated with some t. Let τ be associated with some $t = s'_1, \ldots, s'_i[l'_1, \ldots, l'_k] \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]\#$. Then after taking a few transitions from s_1, we are in s'_1 in the corresponding path in the original automaton. Since t leads into a # copy, some accepting transition

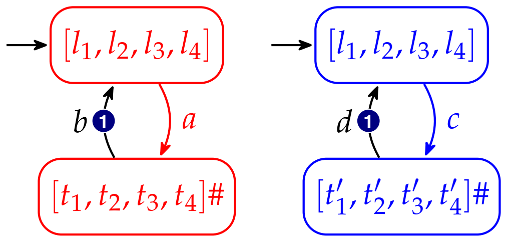


Figure 4.6: Demonstrating the fix from Subsection 4.1.4 on the example from Figure 4.4. The red copy is added because of the red transition and the blue copy is added because of the blue transition.

$r'_1, \dots, r'_i[t'_1, \dots, t'_k]\# \xrightarrow{a}_m r''_1, \dots, r''_i[t''_1, \dots, t''_k]$ must come after t in τ. The accepting transitions are only added when enclosing. Therefore r'_1 can move to r''_2 over a in the original automaton. So in the corresponding path, we have moved from s'_1 to r''_2. Since τ is associated with t, before moving into # copies and seeing another accepting mark, we must go through t first. Therefore, we get from r''_2 to s'_2. If we repeat this again (and we will because there is another accepting mark), we end up in s'_3. Therefore, when repeating τ infinitely, the corresponding run goes infinitely often through all the loop transitions of t, i.e. transitions $l'_1 \xrightarrow{a} r_1, \dots, l'_k \xrightarrow{a} r_k$. Furthermore, since t goes into a # copy, there is some $l'_j \xrightarrow{b}_m t_j$ in the original automaton. Therefore the corresponding run in the original automaton will also be accepting.

The following fix ensures that every accepting loop is associated with some t. When adding $r_1, \dots, r_i[t_1, \dots, t_k] \xrightarrow{a} s_1, \dots, s_i[l_1, \dots, l_k]\#$, we copy every composite state that can reach $r_1, \dots, r_i[t_1, \dots, t_k]$ and add the transition into the copy instead. Further, we finish the parallel synchronous composition in the copy without constructing any more transitions from composite states into # states. The fix is demonstrated on the example from Figure 4.4 in Figure 4.6. Further, a general scheme of the fix on the general case is presented in Figure 4.7.

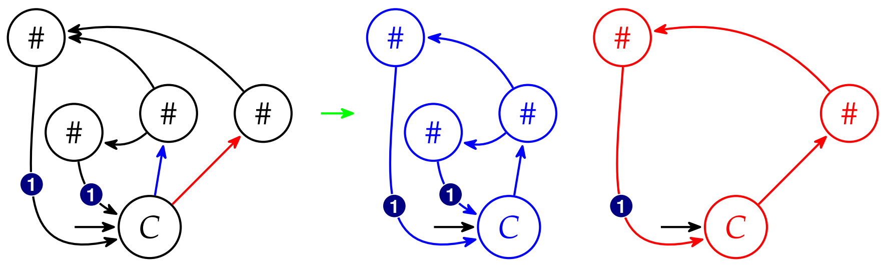


Figure 4.7: Scheme of the general problem and the fix from Subsection 4.1.4. The blue transition is associated with loops in the blue component and the red transition is associated with the red loop.

4.1.5 Intuitive description of the tightening algorithm

In this subsection, we combine the parallel composition of the composite states with all the fixes for the presented problems and introduce an intuitive description of the tightening algorithm.

Assume a TBA $\mathcal{B}$ on the input. The algorithm gradually adds new components into $\mathcal{B}$ and, in the end, returns a modified tight version of the original automaton denoted as $\mathcal{B}'$. It iterates through all $0 \leq i < |Q|$ and $1 \leq k \leq |Q|$, such that $2 \leq i + k \leq |Q|$. For a concrete pair i, k, it does the following:

- Firstly, it constructs a parallel synchronous composition of the composite states into some new automaton $\mathcal{C}$ and collects set *AssociatedTransitions* of transitions that might be associated with some loop. In other words, it goes through all not yet visited states $s_1, \dots, s_i[l_1, \dots, l_k]$ or $s_1, \dots, s_i[l_1, \dots, l_k]\#$ and constructs the following transitions in $\mathcal{C}$.

  If $s_1 \xrightarrow{a} r_1, \dots, s_i \xrightarrow{a} r_i, l_1 \xrightarrow{a} t_1, \dots, l_k \xrightarrow{a} t_k$ and $r_1, \dots, r_i[l_1, \dots, l_k]$ is a composite state (Subsection 4.1.1),

  - it adds $s_1, \dots, s_i[l_1, \dots, l_k]\# \xrightarrow{a} r_1, \dots, r_i[t_1, \dots, t_k]\#$, if composite state $s_1, \dots, s_i[l_1, \dots, l_k]\#$ is marked (Subsection 4.1.3),
  - or else, it adds $t = s_1, \dots, s_i[l_1, \dots, l_k] \xrightarrow{a} r_1, \dots, r_i[t_1, \dots, t_k]\#$, if $\exists j. l_j \xrightarrow{a}_m t_j$ in $\mathcal{B}$ (Subsection 4.1.3), further, it adds t into *AssociatedTransitions* (Subsection 4.1.4),
  - or else, it adds $s_1, \dots, s_i[l_1, \dots, l_k] \xrightarrow{a} r_1, \dots, r_i[t_1, \dots, t_k]$ (Subsection 4.1.1).

- Secondly, it encloses all loops in $\mathcal{C}$. Let $t = r_1, \dots, r_i[t_1, \dots, t_k] \xrightarrow{a}_m s_1, \dots, s_i[l_1, \dots, l_k]$ denote not yet constructed transition. For each $s_1, \dots, s_i[l_1, \dots, l_k]$ that can reach state $r_1, \dots, r_i[t_1, \dots, t_k]$ and (Subsection 4.1.2)

  - $r_1 \xrightarrow{a} s_2, \dots, r_{i-1} \xrightarrow{a} s_i, r_i \xrightarrow{a} l_1, \dots, t_{k-1} \xrightarrow{a} l_k, t_k \xrightarrow{a} l_1$ and composite state $r_1, \dots, r_i[t_1, \dots, t_k]\#$ is marked with #, add t into $\mathcal{C}$ (Subsection 4.1.2), or

  - $r_1 \xrightarrow{a} s_2, \ldots, r_{i-1} \xrightarrow{a} s_i, r_i \xrightarrow{a} l_1, \ldots, t_{k-1} \xrightarrow{a} l_k, t_k \xrightarrow{a} l_1$ and $\exists j. t_j \xrightarrow{a}_m l_{j+1}$ in $\mathcal{B}$, add t into $\mathcal{C}$ (Subsection 4.1.2) and add t into *AssociatedTransitions* (Subsection 4.1.4). This edge case covers finite words for which the marked transition is over the last symbol. Therefore, we do not pass to # copies, but we directly enclose the loop with marked transition. It needs to be also added to *AssociatedTransitions*.

- The algorithm goes through every t from *AssociatedTransitions* and constructs automata components $\mathcal{C}_t$. It is a copy of $\mathcal{C}$ without transitions from *AssociatedTransitions*. The only transition from the set that it contains is t. (Subsection 4.1.4).

- Lastly, it goes through the components $\mathcal{C}_t$. The whole component $\mathcal{C}_t$ is added to $\mathcal{B}$ as fresh states and transitions. Further, it goes through every state $s_1, \ldots, s_i[l_1, \ldots, l_k]$ from $\mathcal{C}_t$ and does the following

  - if s_1 is an initial state in $\mathcal{B}$, then it sets $s_1, \ldots, s_i[l_1, \ldots, l_k]$ as an initial state too,
  - if there is a transition $q \xrightarrow{a} s_1$ in $\mathcal{B}$, it adds transition $q \xrightarrow{a} s_1, \ldots, s_i[l_1, \ldots, l_k]$ into $\mathcal{B}$.

  The previous procedure ensures that everything that could reach s_1, will now be able to reach $s_1, \ldots, s_i[l_1, \ldots, l_k]$. By that, we ensure that if there was a path over some $minS(u)$ into s_1, there will also be a path into $s_1, \ldots, s_i[l_1, \ldots, l_k]$.

One can see the individual parts from each subsection working together to construct a 'tight component' as motivated at the beginning of Section 4.1. The red part in the first step is implementing the idea of composite states introduced in Subsection 4.1.1. The blue part is the fix needed to remember that we have passed an accepting mark, presented in Subsection 4.1.3. The purple part from the second step is the idea of enclosing the loops from Subsection 4.1.2 together with the fix, where we enclose only with an accepting transition from Subsection 4.1.3. Lastly, the orange part is a fix to ensure that every loop is associated with some transition - presented in Subsection 4.1.4.

4.2 Correctness of the tightening algorithm

We formalize the idea presented in the previous section in the form of a pseudocode.

4.2.1 Pseudocode

```
Function TighteningTBA($\mathcal{B}$)
  Input: A TBA $\mathcal{B} = (Q, \Sigma, \{m\}, \delta, I, \mathsf{Inf}(m))$
  Output: Tight TBA $\mathcal{B}' = (Q', \Sigma, \{m\}, \delta', I', \mathsf{Inf}(m))$
1 SCCs ← Compute strongly connected components
2 for $SCC$ in $SCCs$ do
3     for $2 \leq n \leq |Q|$ do
4         for $i + k = n$, where $i \geq 0$ and $k > 0$ do
5             $(\mathcal{C}, AssociatedTransitions) \leftarrow PSC(\mathcal{B}, i, k)$
6             for $s_1, \dots, s_i[l_1, \dots, l_k]$ from $\mathcal{C}$ do
7                 for $r_1, \dots, r_i[t_1, \dots, t_k]$ that $s_1, \dots, s_i[l_1, \dots, l_k]$ can reach
                    in $\mathcal{C}$ do
8                     if $r_1 \xrightarrow{a} s_2, \dots, r_i \xrightarrow{a} l_1, t_1 \xrightarrow{a} l_2, \dots, t_k \xrightarrow{a} l_1 \in \delta$ then
9                         $t = r_1, \dots, r_i[t_1, \dots, t_k] \xrightarrow{a}_m s_1, \dots, s_i[l_1, \dots, l_k]$
10                        if $r_1, \dots, r_i[t_1, \dots, t_k]\#$ is marked with $\#$ then
11                            Add $t$ to $\mathcal{C}$
12                        else
13                            if $\exists j.t_j \xrightarrow{a}_m l_{j+1} \in \delta$ or $t_k \xrightarrow{a}_m l_1 \in \delta$ then
14                                Add $t$ to $\mathcal{C}$
15                                Add $t$ to $AssociatedTransitions$
16            for $t$ in $AssociatedTransitions$ do
17                $\mathcal{C}_t \leftarrow Copy(\mathcal{C}, AssociatedTransitions, t)$
18                Add $\mathcal{C}_t$ into $\mathcal{B}'$
19                for $s_1, \dots, s_i[l_1, \dots, l_k]$ in $\mathcal{C}_t$ do
20                    if $s_1 \in I$ then
21                        Add $s_1, \dots, s_i[l_1, \dots, l_k]$ to $I'$
22                    for $q \xrightarrow{a} s_1 \in \delta$ do
23                        Add $q \xrightarrow{a} s_1, \dots, s_i[l_1, \dots, l_k]$ to $\delta'$
24 return $\mathcal{B}'$
```

Algorithm 4.8: A pseudocode for the tightening algorithm of TBA.

We extract the most straightforward parts from the pseudocode presented in Algorithm 4.8 to keep the algorithm clean and short. The excluded parts are marked in blue colour. For completeness, let us comment on them more.

A function $PSC(\mathcal{B}, i, k)$ constructs a parallel synchronous composition for composite states with a stem of length i and a loop of length k. The loop states are taken only from the same strongly connected component. Therefore we compute SCCs first (we omit a SCC from arguments of PSC, to keep it short). The stem states are such that they can reach the loop states. Furthermore, if there is a marked transition between some loop states, i.e. $\exists j. l_j \xrightarrow{a}_m t_j$, we add a transition without any acceptance mark to the state marked with # instead. Also, it adds every transition $t = s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]\#$ into *AssociatedTransitions*. Naturally, transitions from # states lead only to # states.

A function $Copy(\mathcal{C}, AssociatedTransitions, t)$ for a component $\mathcal{C}$ and a transition $t \in AssociatedTransitions$ creates a copy of $\mathcal{C}$ without transitions from *AssociatedTransitions*. Lastly, it adds t into the copy.

The last blue line creates fresh states and transitions between them according to $\mathcal{C}_t$ and adds them into Q' and δ'.

4.2.2 Proof of the correctness and analysis of complexity

Further, in the remaining part of the chapter, we prove the correctness of the algorithm, i.e. the TBA we construct is tight, and it accepts the same language as the original one. At the end of this section, we analyze the complexity.

Lemma 6. *Let $\mathcal{B}$ be an arbitrary TBA and $\mathcal{B}'$ be the automaton returned by TighteningTBA($\mathcal{B}$). Let τ be a path in $\mathcal{B}'$ of the one of the following forms*

$$s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{u} r_1, \ldots, r_i[t_1, \ldots, t_k]$$

$$s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{u} r_1, \ldots, r_i[t_1, \ldots, t_k]\#$$

Then there are paths $s_1 \xrightarrow{u} r_1, \ldots, s_i \xrightarrow{u} r_i, l_1 \xrightarrow{u} t_1, \ldots, l_k \xrightarrow{u} t_k$ in $\mathcal{B}$.

Proof. Since all transitions in τ are not marked, they are constructed by $PSC(\mathcal{B}, i, k)$ at line 5. We will show the lemma by induction with respect to the length of u.

Base: Let $|u| = 0$, then the lemma holds since there are trivial paths of length 0 in each state of $\mathcal{B}$.

Inductive step: Let $|u| > 0$ and let us assume that the proposition holds for paths over words of length $|u| - 1$. Let τ be a path over $u = va$, where $|v| = |u| - 1$. Further, we decompose τ into $\beta\gamma$. Path β can now be of two forms, either $\beta = s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{v} r'_1, \ldots, r'_i[t'_1, \ldots, t'_k]$ or $\beta = s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{v} r'_1, \ldots, r'_i[t'_1, \ldots, t'_k]\#$. In both cases, the inductive assumption can be applied, and there are paths $s_1 \xrightarrow{v} r'_1, \ldots, s_i \xrightarrow{v} r'_i, l_1 \xrightarrow{v} t'_1, \ldots, l_k \xrightarrow{v} t'_k$ in $\mathcal{B}$.

Let us now focus on γ. It can also be of few forms:

- $\gamma = r'_1, \ldots, r'_i[t'_1, \ldots, t'_k] \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]$, such transition was constructed in $PSC(\mathcal{B}, i, k)$, only if paths $r'_1 \xrightarrow{a} r_1, \ldots, r'_i \xrightarrow{a} r_i, t'_1 \xrightarrow{a} t_1, \ldots, t'_k \xrightarrow{a} t_k$ exist in $\mathcal{B}$.

- $\gamma = r'_1, \ldots, r'_i[t'_1, \ldots, t'_k] \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]\#$, such transition was constructed in $PSC(\mathcal{B}, i, k)$, only if paths $r'_1 \xrightarrow{a} r_1, \ldots, r'_i \xrightarrow{a} r_i, t'_1 \xrightarrow{a} t_1, \ldots, t'_j \xrightarrow{a}_m t_j, \ldots, t'_k \xrightarrow{a} t_k$ exist in $\mathcal{B}$.

- $\gamma = r'_1, \ldots, r'_i[t'_1, \ldots, t'_k]\# \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]\#$, such transition was constructed in $PSC(\mathcal{B}, i, k)$, only if paths $r'_1 \xrightarrow{a} r_1, \ldots, r'_i \xrightarrow{a} r_i, t'_1 \xrightarrow{a} t_1, \ldots, t'_k \xrightarrow{a} t_k$ exist in $\mathcal{B}$.

If we concatenate the paths that exist in $\mathcal{B}$ thanks to β and γ, we obtain the required paths. □

Lemma 7. *Let $\mathcal{B}$ be an arbitrary TBA and $\mathcal{B}'$ be the automaton returned by TighteningTBA$(\mathcal{B})$. Let τ be a path in $\mathcal{B}'$ of the following form*

$$s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{u}_m r_1, \ldots, r_i[t_1, \ldots, t_k]$$

Then there is a path $s_1 \xrightarrow{u} r_j$ or $s_1 \xrightarrow{u} t_j$ in $\mathcal{B}$.

Proof. We will show the lemma by induction with respect to the number of marked transitions in τ. Let n be the number.

Base: Let $n = 0$, then the lemma holds since from Lemma 6, there is a path $s_1 \xrightarrow{u} r_1$.

Inductive step: Let $n \leq 0$ and let us assume that the proposition holds for paths with n marked transitions. We will show, that it holds also for paths with $n+1$ transitions.

Let $\tau = \beta\gamma$, such that τ contains $n+1$ marked transitions. Further, let β be a path over $v_0 a$, containing n marked transitions and ending with some $s'_1, \dots, s'_i[l'_1, \dots, l'_k] \xrightarrow{a}_m r'_1, \dots, r'_i[t'_1, \dots, t'_k]$. From an inductive assumption there is a path $s_1 \xrightarrow{v_0 a} r'_j$ or $s_1 \xrightarrow{v_0 a} t'_j$.

Path γ contains precisely one marked transition t, we decompose γ based on t. Let $\gamma = \sigma_0 t \sigma_1$, where $t = r''_1, \dots, r''_i[t''_1, \dots, t''_k] \xrightarrow{b}_m r'''_1, \dots, r'''_i[t'''_1, \dots, t'''_k]$. Further, σ_0 is a path over v_1 and starts with $r'_1, \dots, r'_i[t'_1, \dots, t'_k]$ and path σ_1 over v_2 ends with $r_1, \dots, r_i[t_1, \dots, t_k]$.

Because of σ_0 and Lemma 6 there are paths $r'_j \xrightarrow{v_1} r''_j$ and $t'_j \xrightarrow{v_1} t''_j$ (based on where the path from $\mathcal{B}$ for β ends).

Since t is marked, it was constructed by one of the conditions at lines 10 or 13. Therefore, there are transition $r''_1 \xrightarrow{b} r'''_2, \dots, r''_i \xrightarrow{b} t'''_1, t''_1 \xrightarrow{b} t'''_2, \dots, t''_k \xrightarrow{b} t'''_1$ in $\mathcal{B}$. So we can take one of the transitions over b in $\mathcal{B}$ and end up in r'''_{j+1}, or t'''_{j+1}, or t'''_1, depending on j.

Similarly as for σ_0, because of σ_1 and Lemma 6, there are paths $r'''_{j+1} \xrightarrow{v_2} r_{j+1}$, $t'''_{j+1} \xrightarrow{v_2} t_{j+1}$ and $t'''_1 \xrightarrow{v_2} t_1$ in $\mathcal{B}$. Depending on so far constructed path, we can concatenate one of the paths and obtain a path from s_1 to some r_{j+1}, t_{j+1} or t_1 over u in $\mathcal{B}$. □

Lemma 8. *Let $\mathcal{B}$ be an arbitrary TBA and $\mathcal{B}'$ be the automaton returned by TighteningTBA$(\mathcal{B})$. Let q_0 be an initial state and $\pi\tau$ be a path in $\mathcal{B}'$ of the following form*

$$q_0 \xrightarrow{v} s_1, \dots, s_i[l_1, \dots, l_k] \xrightarrow{u}_m s_1, \dots, s_i[l_1, \dots, l_k]$$

Then $vu^\omega \in L(\mathcal{B})$.

Proof. We construct an accepting run over vu^ω in $\mathcal{B}$. Let us first focus on π part of $\pi\tau$ path. We firstly show that $q_0 \xrightarrow{v} s_j$ or $q_0 \xrightarrow{v} l_j$ exists in $\mathcal{B}$. It follows from lines 19-23 and Lemma 7.

If $|v| = 0$, then $s_1, \dots, s_i[l_0, \dots, l_k] = q_0$ and it is an initial state in $\mathcal{B}'$. Thanks to the condition at lines 20-21, it is only possible if s_1 is an initial state. Therefore we can start the run in $\mathcal{B}$ in s_1.

If $|v| > 0$, then there is path $q_0 \xrightarrow{v_0} q \xrightarrow{a} s'_1, \ldots, s'_i[l'_1, \ldots, l'_k] \xrightarrow{v_1}_m s_1, \ldots, s_i[l_1, \ldots, l_k]$, such that $v = v_0 a v_1$, $a \in \Sigma$ and m might be there or not. Path $q_0 \xrightarrow{v_0} q$ is already from $\mathcal{B}$. Further, a transition $q \xrightarrow{a} s'_1$ is possible because of the for cycle at lines 22-23. Lastly, since $s'_1, \ldots, s'_i[l'_1, \ldots, l'_k] \xrightarrow{v_1}_m s_1, \ldots, s_i[l_1, \ldots, l_k]$ is in $\mathcal{B}'$, from Lemma 7 there are paths $s'_1 \xrightarrow{v_1} s_j$ or $s'_1 \xrightarrow{v_1} l_j$ in $\mathcal{B}$.

Now we focus on τ part of the path. Since τ contains at least one accepting mark but it starts in a state that is not marked with #, there must be one of the two cases:

- There is transition $t = s'_1, \ldots, s'_i[l'_1, \ldots, l'_k] \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]\#$ in τ (let us abuse the notation and by $s'_1, \ldots, s'_i[l'_1, \ldots, l'_k]$ denote a different state than in the previous paragraph). Notice that $t \in \mathit{AssociatedTransitions}$ after line 5. Therefore, $\mathcal{C}_t$ was added into $\mathcal{B}'$ at some point at line 18 and τ must be in this $\mathcal{C}_t$. Further, because of *Copy* function at line 17, the only transition from composite states to # copies that τ contain is t. It also does not contain any accepting marks added at line 14 since they are also in *AssociatedTransitions*. So τ is *associated* with t.

  Without loss of generality, let us assume that we have constructed a path from q_0 to s_j in the π part of the proof. To get the first time to t, there must be some part of τ of the following form $s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{u_0} s'_1, \ldots, s'_i[l'_1, \ldots, l'_k]$. From Lemma 6, there is a path in $\mathcal{B}$ from s_j to s'_j. Then there is t in τ, so we can move from s'_j to r_j.

  Now, there must be a marked transition before again seeing t. Since the algorithm added the marked transition at lines 10-11, the condition at line 8 must have been satisfied. Therefore, from Lemma 7 we construct a path to s'_{j+1} before again seeing t. As you can see, we can repeat τ infinitely and every time we see t, we move from s'_j to s'_{j+1}, then to s'_{j+2} and so on. Since we are always moving by one 'iteration', we will see transitions $l'_1 \xrightarrow{a} t_1, \ldots, l'_k \xrightarrow{a} t_k$ infinitely often. At least one of them is marked. Therefore, such constructed run is accepting over vu^ω in $\mathcal{B}$.

- There is transition $t = s'_1, \dots, s'_i[l'_1, \dots, l'_k] \xrightarrow{a}_m r_1, \dots, r_i[t_1, \dots, t_k]$ in τ. Since t does not go from # copy to composite state, it was added at lines 14-15. Therefore, when removing all transitions from *AssociatedTransitions* at line 17, we also remove all # copies. Therefore τ does not contain any # copies. Further, it is easy to see that the only marked transition that τ can now contain is t.

  Without loss of generality, let us assume that we have constructed a path from q_0 to s_j in the π part of the proof. To get the first time to t, there must be some part of τ of the following form $s_1, \dots, s_i[l_1, \dots, l_k] \xrightarrow{u_0} s'_1, \dots, s'_i[l'_1, \dots, l'_k]$. From Lemma 6, there is a path in $\mathcal{B}$ from s_j to s'_j.

  When taking t, we must move from s'_j to r_{j+1} because of the condition at line 8. Then, from Lemma 6 we get into s'_{j+1} before again taking t. After taking t we again raise the index to r_{j+2}.

  When repeating τ infinitely, we see every transition $l'_1 \xrightarrow{a} t_2, \dots, l'_{k-1} \xrightarrow{a} t_k, l'_k \xrightarrow{a} t_1$ infinitely often. Because of the condition at line 13, at least one of them is marked. Therefore, such constructed run is accepting over vu^ω in $\mathcal{B}$.

□

The language equality of the input and output automata is now a direct consequence of Lemma 8.

Theorem 1. *Let $\mathcal{B}$ be an arbitrary TBA and $\mathcal{B}'$ be the automaton returned by TighteningTBA($\mathcal{B}$), then $L(\mathcal{B}) = L(\mathcal{B}')$.*

Proof. Since in the construction of $\mathcal{B}'$, we only add some components to $\mathcal{B}$, the inclusion $L(\mathcal{B}) \subseteq L(\mathcal{B}')$ is trivial.

Further, we show $L(\mathcal{B}') \subseteq L(\mathcal{B})$. Two automata accept the same language if they accept the same lasso-shaped words [11]. Hence, it suffices to show that for every lasso-shaped word $w \in L(\mathcal{B}')$, it holds that $w \in L(\mathcal{B})$. For every lasso-shaped word, there is an accepting lasso-shaped run $\rho = \kappa\pi\tau^\omega$ - Claim 1.

The most general case of accepting lasso-shaped runs that are not in $\mathcal{B}$, but are in $\mathcal{B}'$ is the following:

$$q_0 \xrightarrow{v} (s_1, \dots, s_i[l_1, \dots, l_k] \xrightarrow{u}_m s_1, \dots, s_i[l_1, \dots, l_k])^\omega$$

where $w = uv^\omega$. From Lemma 8, $vu^\omega \in L(\mathcal{B})$ holds. □

Theorem 2. *Let $\mathcal{B}$ be an arbitrary TBA and $\mathcal{B}'$ be the automaton returned by TighteningTBA$(\mathcal{B})$, then $\mathcal{B}'$ is tight.*

Proof. Assume a lasso-shaped word $w \in L(\mathcal{B}')$, from Theorem 1, it holds that $w \in L(\mathcal{B})$. From Claim 1, there is a lasso-shaped run ρ, with at most $|Q|$ $minL(u)$-significant states $s_1, \dots, s_i, l_1, \dots, l_k$.

Since there is a path $q_1 \xrightarrow{minS(w)} s_1$ in $\mathcal{B}$, then thanks to lines 19-23 in *TighteningTBA*, there is also a path $\pi = q_0 \xrightarrow{minS(w)} s_1, \dots, s_i[l_1, \dots, l_k]$ in $\mathcal{B}'$.

Further, without loss of generality, we can assume that ρ is the one from Lemma 5. Therefore, if we move synchronously over some prefix of $minL(u)$ from $s_1, \dots, s_i, l_1, \dots, l_k$ in ρ we end up in a pairwise different sequence of states $r_1, \dots, r_i, t_1, \dots, t_k$.

Since ρ is accepting, there is some prefix u_0 of $minL(u)$, such that $\exists j. l_j \xrightarrow{u_0} t_j \xrightarrow{a}_m l'_j$. Further, let u_0 be the shortest prefix, over which, if we take synchronously transitions from states $l_1, \dots, l_k$ in $\mathcal{B}$ we arrive at the first marked transition.

From the construction of the parallel synchronous composition at line 5, there is a path $s_1, \dots, s_i[l_1, \dots, l_k] \xrightarrow{u_0} r_1, \dots, r_i[t_1, \dots, t_k]$ in $\mathcal{B}'$. Let us continue the proof based on two cases:

- $u_0a = minL(u)$, in this case, transitions over a are the last transitions of paths $s_1 \xrightarrow{u_0a} s_2, \dots, s_{i-1} \xrightarrow{u_0a} s_i, s_i \xrightarrow{u_0a} l_1 \dots l_k \xrightarrow{u_0a} l_1$. Furthermore, $s_1, \dots, s_i[l_1, \dots, l_k]$ can reach $r_1, \dots, r_i[t_1, \dots, t_k]$ over u_0 and transition $l_j \xrightarrow{a}_m l_{j+1}$ is marked. Therefore, condition at line 13 is satisfied and transition $t = r_1, \dots, r_i[t_1, \dots, t_k] \xrightarrow{a}_m s_1, \dots, s_i[l_1, \dots, l_k]$ is added to $\mathcal{C}$. Furthermore this cycle over u_0a, is added at line 17 to $\mathcal{B}'$ as a part of $\mathcal{C}_t$.

- $u_0a \neq minL(u)$, in this case, there is a transition constructed by the parallel synchronous composition at line 5 of the form $t = r_1, \dots, r_i[t_1, \dots, t_k] \xrightarrow{a} s'_1, \dots, s'_i[l'_1, \dots, l'_k]\#$. Further there are transitions over the rest of $minL(u)$ in the # copies until the last symbol of $minL(u)$, lets denote it b. There are transitions $r'_1 \xrightarrow{b} s_2, \dots, r'_{i-1} \xrightarrow{b} s_i, r_i \xrightarrow{b} l_1, \dots t'_k \xrightarrow{b} l_1$ in $\mathcal{B}$, since b is the last symbol of $minL(u)$. Furthermore, $r'_1, \dots, r'_i[t'_1, \dots, t'_k]\#$ is reachable from $s_1, \dots, s_i[l_1, \dots, l_k]$, therefore condition at line 10 is triggered and the run can get back to $s_1, \dots, s_i[l_1, \dots, l_k]$ with a marked transition over b. This cycle over $minL(u)$, is added at line 17 to $\mathcal{B}'$ as a part of $\mathcal{C}_t$.

□

Claim 2. *Let $\mathcal{B}$ be a TBA with n states over an alphabet Σ, and $\mathcal{B}'$ be a tight TBA returned by TighteningTBA($\mathcal{B}$). Then the number of states of $\mathcal{B}'$ is at most*

$$\mathcal{O}(\sum_{j=2}^{n} \sum_{\substack{|i+k=j \\ k>0}} (\binom{n}{i+k} \cdot (i+k)!)^3 \cdot |\Sigma|)$$

Proof. For every $2 \leq j \leq n$ and for every i, k, where $k > 0$ and $i + k = j$ our algorithm constructs in the worst case a composite state for every pairwise different sequence of states $s_1, \dots, s_i, l_1, \dots, l_k$. Given i, k, we need to pick a combination without repetition of $i + k$ states, therefore $\binom{n}{i+k}$ states. Further, we have to multiply it by the number of permutations we can do on the set of picked states, resulting in $\binom{n}{i+k} \cdot (i+k)!$. That is the number of composite states we can build on a given i, k. We can also have their # copies, but in the worst case it just doubles the state space, so it gets cancelled when using $\mathcal{O}$-notation. Lastly, we can copy them in the *Copy* function at line 17. In the worst case, we copy each composite state for every transition between them. There is at most $(\binom{n}{i+k} \cdot (i+k)!)^2 \cdot |\Sigma|$ transitions between them. Resulting in at most $(\binom{n}{i+k} \cdot (i+k)!)^3 \cdot |\Sigma|$ copies. □

Assuming that $|\Sigma|$ is, in most practical cases, significantly smaller than $n!$, after simplification, one can see that the leading function is $(n!)^3$, which is better than the so far reached upper boundary to this problem $\mathcal{O}(n^{4n+6})$. The following chapter shows how Rabin acceptance condition can be used to lower the leading function to $n!$.

5 Tightening of TRA

Since we extended the formalism of tight ω-automata to the whole class of TELA, we can study the tightening of its arbitrary subclasses. The subclass we focus on in this chapter is TRA. As we present later, the algorithm for transforming TRAs to tight TRAs has better theoretical state complexity than the tightening of TBA. Furthermore, it requires only minor modifications of the algorithm presented in Chapter 4. This chapter's most important result is the one at the end, where we show that there is just a small gap between the presented construction and the optimal one.

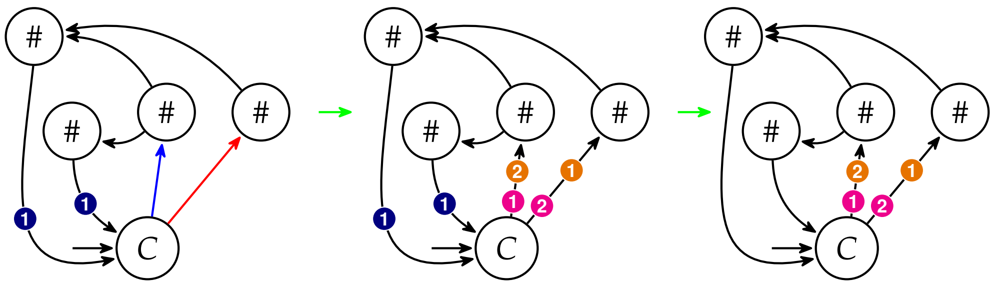
The algorithm we present is based on the following trivial observation. If a lasso-shaped run $\pi\tau^\omega$ is accepting over a lasso-shaped word in some TRA $\mathcal{A}$, then there is some Rabin pair (f, i), such that τ does not contain f. Let us focus on this concrete Rabin pair (f, i). We allow a transition $s_1, \dots, s_i[l_1, \dots, l_k] \xrightarrow{a} r_1, \dots, r_i[t_1, \dots, t_k]$, when constructing a parallel composition only if there is no mark from f on the transitions between the loop states. If we modify the parallel composition by the previous claim, we obtain a TBA for a pair (f, i).

5.1 Formal description of the algorithm

5.1.1 Improvement of the construction using Rabin pairs

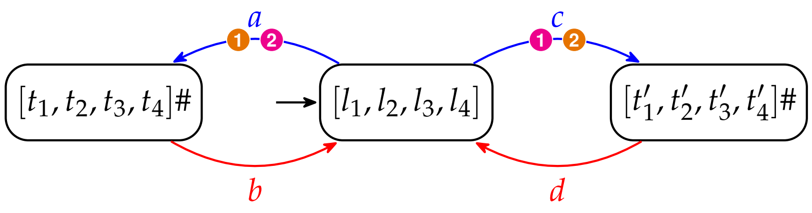
When transforming a TRA into a tight TRA, we can use a slightly modified version of Algorithm 4.8. The place where we use Rabin pairs to spare some states is *Copy* function at line 17. This function is used to solve the problem from Section 4.1.4. It ensures that every accepting lasso-shaped run has its loop associated with some transition t. *TighteningTBA* does it in a way that it copies the parallel composition $\mathcal{C}$ and excludes every transition from *AssociatedTransitions* from the copy, except for t. We can now use Rabin pairs to avoid copying $\mathcal{C}$ many times.

Instead of copying, we create a new Rabin pair (f_t, i_t) for every transition from *AssociatedTransitions*. Further, we mark t with i_t, and every other transition from *AssociatedTransitions* by f_t. Lastly, we remove every original mark from $\mathcal{C}$.



$\varphi = (\mathsf{Fin}(❷) \wedge \mathsf{Inf}(❷)) \vee (\mathsf{Fin}(❶) \wedge \mathsf{Inf}(❶))$

Figure 5.1: The same scheme as in Figure 4.7 using the modified fix with Rabin pairs. Firstly, we assign fresh Rabin pairs to blue and red transitions. Further, as the last step we remove the original marks. The final acceptance condition is φ. Notice, that there are the same accepting loops as in the fixed version of Figure 4.7.



$\varphi = (\mathsf{Fin}(❷) \wedge \mathsf{Inf}(❷)) \vee (\mathsf{Fin}(❶) \wedge \mathsf{Inf}(❶))$

Figure 5.2: Using the fix with Rabin pairs on the concrete example in Figure 4.5.

The fix enforces an accepting run to contain some i_t in the loop and not contain any f_t. Since every accepting run of $\mathcal{C}$ must pass some of the transitions from *AssociatedTransitions*, in the loop, it may go only over t, since that is the only one not containing f_t. By that, we ensure that every loop of an accepting run is associated with some t.

5.1.2 Pseudocode

Algorithm 5.3 is the formal pseudocode of the tightening algorithm.

```
Function TighteningTRA($\mathcal{R}$)
  Input: A TRA $\mathcal{R} = (Q, \Sigma, \{f_0, i_0, f_1, i_1 \dots, f_m, i_m\}, \delta, I, \bigvee_{j\in\{0,\dots,m\}}(\mathsf{Fin}(f_j) \wedge \mathsf{Inf}(i_j)))$
  Output: Tight TRA $\mathcal{R}' = (Q', \Sigma, M', \delta', I', \varphi)$
  SCCs $\leftarrow$ Compute strongly connected components
  $M' \leftarrow \{f_0, i_0, f_1, i_1 \dots, f_m, i_m\}$
  $\varphi \leftarrow \bigvee_{j\in\{0,\dots,m\}}(\mathsf{Fin}(f_j) \wedge \mathsf{Inf}(i_j)))$
  for $0 \leq j \leq m$ do
      for $SCC$ in $SCCs$ do
          for $2 \leq n \leq |Q|$ do
              for $i + k = n$, where $i \geq 0$ and $k > 0$ do
                  $(\mathcal{C}, AssociatedTransitions) \leftarrow PSCRabin(\mathcal{R}, i, k, f_j, i_j)$
                  for $s_1, \dots, s_i[l_1, \dots, l_k]$ from $\mathcal{C}$ do
                      for $r_1, \dots, r_i[t_1, \dots, t_k]$ that $s_1, \dots, s_i[l_1, \dots, l_k]$ can reach in $\mathcal{C}$ do
                          if $r_1 \xrightarrow{a} s_2, \dots, r_i \xrightarrow{a} l_1, t_1 \xrightarrow{a} l_2, \dots, t_k \xrightarrow{a} l_1 \in \delta$ then
                              $t = r_1, \dots, r_i[t_1, \dots, t_k] \xrightarrow{a} s_1, \dots, s_i[l_1, \dots, l_k]$
                              if $r_1, \dots, r_i[t_1, \dots, t_k]\#$ is marked with $\#$ then
                                  Add $t$ to $\mathcal{C}$
                              else
                                  if $\exists o.t_o \xrightarrow{a}_{M_o} l_{o+1} \in \delta$ and $i_j \in M_o$, $f_j \notin M_o$ then
                                      Add $t$ to $\mathcal{C}$
                                      Add $t$ to $AssociatedTransitions$
                  for $t$ in $AssociatedTransitions$ do
                      Add $f_t, i_t$ to $M'$ and to $\varphi$
                      Mark $t$ by $i_t$
                      Mark every $t' \in AssociatedTransitions$, s.t. $t \neq t'$ by $f_t$
                  Add $\mathcal{C}$ into $\mathcal{R}'$
                  for $s_1, \dots, s_i[l_1, \dots, l_k]$ in $\mathcal{C}$ do
                      if $s_1 \in I$ then
                          Add $s_1, \dots, s_i[l_1, \dots, l_k]$ to $I'$
                      for $q \xrightarrow{a} s_1 \in \delta$ do
                          Add $q \xrightarrow{a} s_1, \dots, s_i[l_1, \dots, l_k]$ to $\delta'$
  return $\mathcal{R}'$
```

Algorithm 5.3: A pseudocode for the function that transforms an arbitrary TRA to a tight TRA.

The blue colour highlights the changes compared to Algorithm 4.8. We further explain some of the changes.

In for cycle at line 4, we go through all Rabin pairs of the original automaton. It is because every accepting lasso-shaped run satisfies at least one cube of Rabin pair, i.e. $\mathsf{Fin}(f_m) \wedge \mathsf{Inf}(i_m)$. There is no transition with f_m and at least one transition with i_m in its loop. Therefore, the accepting runs are similar to those in TBA (it is sufficient to see one mark in the loop). If we modify the parallel synchronous composition (do not allow f_m in the loop), we can tighten the automaton as if it was TBA for a given Rabin pair.

We construct a parallel synchronous composition similarly as in Algorithm 4.8 with only a few changes. Let $PSCRabin(\mathcal{R}, i, k, f_m, i_m)$ denote this modified version of an abstract function $PSC(\mathcal{B}, i, k)$, where (f_m, i_m) is a given Rabin Pair. We modify the description of the composition from Section 4.1.5 in the following way:
If $s_1 \xrightarrow{a}_{M_1} r_1, \ldots, s_i \xrightarrow{a}_{M_i} r_i, l_1 \xrightarrow{a}_{M'_1} t_1, \ldots, l_k \xrightarrow{a}_{M'_k} t_k$ are in $\mathcal{R}$, a state $r_1, \ldots, r_i[l_1, \ldots, l_k]$ is a composite state and $f_m \notin M'_j$ for all $1 \leq j \leq k$,

- it adds $s_1, \ldots, s_i[l_1, \ldots, l_k]\# \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]\#$, if composite state $s_1, \ldots, s_i[l_1, \ldots, l_k]\#$ is marked,
- or else, it adds $t = s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]\#$, if $\exists j. l_j \xrightarrow{a}_{M'_j} t_j$, such that $i_m \in M'_j$, further, the algorithm adds t into *AssociatedTransitions*,
- or else, it adds $s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{a} r_1, \ldots, r_i[t_1, \ldots, t_k]$.

The change at line 12 is that we do not mark t during the enclosing. If the condition at line 16 is satisfied, it is marked later since it is added to *AssociatedTransitions*. If it is not satisfied, then t goes from some # state into a non-marked state. Therefore, some t' that goes from a non-marked state into a # state must be before t in every run. Because of *PSCRabin*, the transition t' is in *AssociatedTransitions*, and we mark it later at line 21.

We replace a function *Copy* from Algorithm 4.8 with lines 19-22. A more detailed motivation for this is in Subsection 5.1.1. Line 20 adds a new Rabin Pair into $\mathcal{R}'$ for a transition t. It further adds cube $\mathsf{Fin}(f_t) \wedge \mathsf{Inf}(i_t)$ into the acceptance condition φ. Lines 21-22 mark t with i_t and all other transitions from *AssociatedTransitions* with f_t.

5.2 Proof of the correctness and analysis of complexity

Theorem 3. *Let $\mathcal{R}$ be an arbitrary TRA and $\mathcal{R}'$ be the automaton returned by TighteningTRA$(\mathcal{R})$, then $L(\mathcal{R}) = L(\mathcal{R}')$.*

Proof. The inclusion $L(\mathcal{R}) \subseteq L(\mathcal{R}')$ is trivial. Let us show the inclusion $L(\mathcal{R}') \subseteq L(\mathcal{R})$.

Let $w = uv^\omega$ be a lasso-shaped word, such that $w \in L(\mathcal{R}')$. It is sufficient to consider only lasso-shaped words [11]. Without loss of generality, we can assume that there is an accepting lasso-shaped run ρ over w of the following form

$$q_0 \xrightarrow{v}_{M_s} (s_1, \dots, s_i[l_1, \dots, l_k] \xrightarrow{u}_{M_l} s_1, \dots, s_i[l_1, \dots, l_k])^\omega$$

where $w = uv^\omega$. Notice that M_s does not matter.

If we created parallel synchronous composition from Algorithm 4.8, i.e. $PSC(\mathcal{R}_{i_m}, i, k)$, where we would treat i_m as the mark of TBA $\mathcal{R}_{i_m}$, we would obtain more transitions than in $PSC(\mathcal{R}, i, k, f_m, i_m)$, since we just add more constraints into the construction. Therefore Lemma 6 also holds, when reformulated to *TighteningTRA*$(\mathcal{R})$.

The only part, where we mark transitions are lines 19-22. Therefore, M_l can contain only some f_t or i_t marks, where $t \in$ *AssociatedTransitions*. Furthermore, it can contain only one i_t mark. It is because, every other mark $i_{t'}$ is on some other transition $t' \in$ *AssociatedTransitions*. But line 22 puts f_t and $f_{t'}$ on transitions t' and t, respectively. Therefore, it would contain also f_t and $f_{t'}$ and ρ would not be accepting. As a result, the loop of ρ is associated with t, since every other $t' \in$ *AssociatedTransitions* contain f_t.

Since ρ visits t infinitely often, it must always pass some transition from enclosing a loop before visiting t again, i.e. transitions from line 12. Therefore, the same arguments as in proof Lemma 7 can be applied and it can be also reformulated with *TighteningTRA*$(\mathcal{R})$. The proof of Lemma 8 uses Lemma 6, Lemma 7 and arguments that clearly hold also for *TighteningTRA* (the loop is associated with t). Therefore, we can use Lemma 8 to show, that $w \in L(\mathcal{R}')$. □

Theorem 4. *Let $\mathcal{R}$ be an arbitrary TRA and $\mathcal{R}'$ be the automaton returned by TighteningTRA$(\mathcal{R})$, then $\mathcal{R}'$ is tight.*

Proof. Let $w \in L(\mathcal{R}')$ be a lasso-shaped word. From Theorem 3, $w \in L(\mathcal{R})$ and from Lemma 5 there is an accepting lasso-shaped run $\rho = \kappa\pi\tau^\omega$ over w. Let $s_1, \ldots, s_i$ and $l_1, \ldots, l_k$ be the $minL(w)$-significant states from π and τ, respectively. Furthermore, there is a Rabin pair (f_m, i_m), such that τ contains i_m and does not contain f_m.

Let $minL(w) = va$. After moving by v from $s_1, \ldots, s_i, l_1, \ldots, l_k$ in ρ, we end up in av-significant states $r_1, \ldots, r_i, t_1, \ldots, t_k$. From Lemma 5, they are pairwise different.

Therefore, in the parallel synchronous composition returned by $PSCRabin(\mathcal{R}, i, k, f_m, i_m)$, there is a path $\tau = s_1, \ldots, s_i[l_1, \ldots, l_k] \xrightarrow{v} r_1, \ldots, r_i[t_1, \ldots, t_k]$.

Based on whether, there is i_m on some of the paths over v or on some of the last transitions over a in $\mathcal{R}$, state $r_1, \ldots, r_i[t_1, \ldots, t_k]$ is marked with # or not.

- If it is marked, then there was a transition t from non-marked state to a # state. After lines 19-22, there is a mark i_t on t, since $t \in AssociatedTransitions$. Furthermore, we can take the other transitions in # copies until reaching $r_1, \ldots, r_i[t_1, \ldots, t_k]$# after taking t. Therefore there is no other transition t' from *AssociatedTransitions*. So τ does not contain f_t. Paths over a from $\mathcal{R}$ satisfy condition at line 11 and transition $r_1, \ldots, r_i[t_1, \ldots, t_k] \xrightarrow{a} s_1, \ldots, s_i[l_1, \ldots, l_k]$ is added to $\mathcal{C}$. Thanks to lines 24-28, it is now straightforward to construct a run ρ, such that $|minSL(w)| = |minSL(\rho)|$.

- If it is not marked, then i_m was not on paths over v in $\mathcal{R}$. Therefore one of the transitions over a must contain i_m. The condition at line 16 is satisfied and transition $t = r_1, \ldots, r_i[t_1, \ldots, t_k] \xrightarrow{a} s_1, \ldots, s_i[l_1, \ldots, l_k]$ is constructed.

  Further, t is added into *AssociatedTransitions* and lines 20-22 mark t with i_t. It is easy to see, that such loop does not contain f_t and thanks to lines 24-28, it is now straightforward to construct a run ρ, such that $|minSL(w)| = |minSL(\rho)|$.

□

Claim 3. *Let $\mathcal{R}$ be a TRA with n states, $|\mathcal{F}|$ be the number of Rabin pairs, and $\mathcal{R}'$ be a tight TRA returned by $TighteningTRA(\mathcal{R})$. Then the number* *of states of $\mathcal{R}'$ is at most*

$$\mathcal{O}(|\mathcal{F}| \cdot \sum_{j=2}^{n} \sum_{\substack{|i+k=j \\ k>0}} \cdot \binom{n}{i+k} \cdot (i+k)!)$$

Proof. The new states that we add, when constructing a tight TRA for a given Rabin pair (f_m, i_m) are only those constructed by $PSC(\mathcal{R}, i, k, f_m, i_m)$. The number of such states is at most $\mathcal{O}(\sum_{j=2}^{n} \sum_{\substack{|i+k=j \\ k>0}} \cdot \binom{n}{i+k} \cdot (i+k)!)$ as shown in the proof of Claim 2. We do such construction for every Rabin pair, resulting in $\mathcal{O}(|\mathcal{F}| \cdot \sum_{j=2}^{n} \sum_{\substack{|i+k=j \\ k>0}} \cdot \binom{n}{i+k} \cdot (i+k)!)$. □

5.3 Optimality of the construction

By the following lemma, we show that for each n, there exists a TRA with $n+1$ states whose tight version must have at least one new state for every pairwise different sequence of its states. Furthermore, it is easy to see that the constructed automaton can be transformed into TBA without the rise of states. Therefore the lower bound holds also for the tightening of TBA.

Lemma 9. *Given $n \in \mathbb{N}$, there is a TRA $\mathcal{R}$ with $n+1$ states, such that for every tight TRA $\mathcal{R}'$ with a set of states Q', the following holds. If $L(\mathcal{R}) = L(\mathcal{R}')$, then*

$$|Q'| \geq \sum_{j=2}^{n} \sum_{\substack{|i+k=j \\ k>0}} \cdot \binom{n}{i+k} \cdot (i+k)!$$

Proof. Given n, let us construct the concrete TRA $\mathcal{R}$ with $n+1$ states. Firstly, let Σ be the alphabet of the language $L(\mathcal{R})$, we gradually add new symbols into the alphabet during the construction. Further, we take the extra state q_0 and set it as the only initial state, i.e. $I = \{q_0\}$. For every i, k, such that $i + k \leq n$ and $i \geq 0$ and $k > 0$, the construction proceeds in the following way:

- We add two fresh symbols c, c_p to Σ.

- We pick $i+k$ pairwise different states in the order $s_1, \dots, s_i[l_1, \dots, l_k]$ that have not been processed yet and construct the transitions as displayed in Figure 5.4:

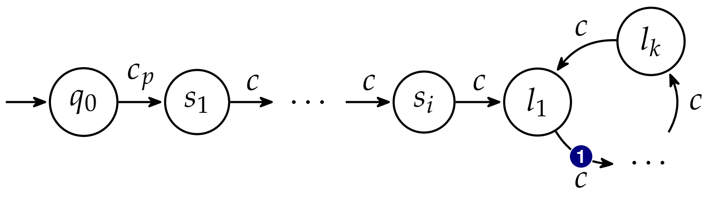


Figure 5.4: The construction of the transitions for a given $s_1, \dots, s_i[l_1, \dots, l_k]$.

- Let $\hat{m}$ be a fresh mark (the blue one displayed in Figure 5.4), add $\mathsf{Inf}(\hat{m})$ to an accepting condition. In other words, create a new Rabin pair $(\hat{f}, \hat{m})$ and add cube $\mathsf{Fin}(\hat{f}) \wedge \mathsf{Inf}(\hat{m})$. Where none transition is marked by $\hat{f}$.

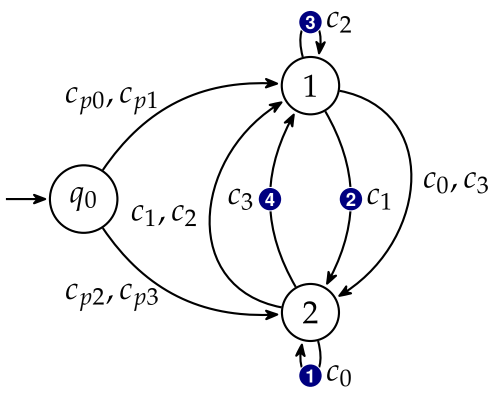


Figure 5.5: An example of the construction for $n = 2$. The orders, for which we construct the transitions with accepted words are: $1[2](c_{p0}c_0(c_0)^\omega)$, $[1,2](c_{p1}(c_1c_1)^\omega)$, $2[1](c_{p2}c_2(c_2)^\omega)$, $[2,1](c_{p3}(c_3c_3)^\omega)$

An example of the construction for $n = 2$ is displayed in Figure 5.5. Let us now prove the claim in the lemma. For each $s_1, \dots, s_i[l_1, \dots, l_k]$, we have added a new word $c_{pref}c^\omega$ to the language. To prove the lemma, it is sufficient to consider $i + k > 1$, therefore, there is no accepting run ρ with $|minSL(\rho)| = 2$ in $\mathcal{R}$. Hence, for each such combination, we will need to have a state in $\mathcal{R}'$ reachable over c_{pref} and with an accepting loop over c. We claim, that for every $s_1, \ldots, s_i[l_1, \ldots, l_k]$ such state must be different. Let us show it by contradiction.

Let us assume, that $\mathcal{R}'$ is tight and $L(\mathcal{R}) = L(\mathcal{R}')$. Since $\mathcal{R}'$ is tight, then there is a state for each $s_1, \ldots, s_i[l_1, \ldots, l_k]$ with a self-loop over c. Let us assume, that there is one state for $s_1, \ldots, s_i[l_1, \ldots, l_k]$ and $r_1, \ldots, r_{i'}[t_1, \ldots, t_{k'}]$ with self-loops over c and c' (both must be accepting transitions).

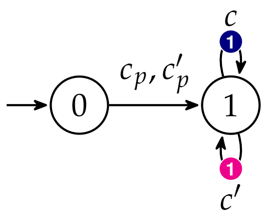


Figure 5.6: Contradictory condition, we assume that holds.

We now distinguish two cases:

- Case $\{l_1, \ldots, l_k\} \neq \{t_1, \ldots, t_{k'}\}$: Without loss of generality, let $t_j \notin \{l_1, \ldots, l_k\}$. We can now construct an accepting run in $\mathcal{R}'$ that has no equivalent in $\mathcal{R}$. In $\mathcal{R}'$, we can take a transition over c'_p and combine c or c' in any way and obtain an accepting run. In $\mathcal{R}$, after taking the transition over c'_p, the run must be in r_1. It can now transit over a sequence of c' into t_j. From there, we cannot take a transition over c since it is not in $\{l_1, \ldots, l_k\}$. Hence, a run over such word does not exist in $\mathcal{R}$, therefore $L(\mathcal{R}) \neq L(\mathcal{R}')$, which is a contradiction with our initial assumptions.

- Case $\{l_1, \ldots, l_k\} = \{t_1, \ldots, t_{k'}\}$: In this case we can combine arbitrarily c and c' in the loop of $\mathcal{R}$ as well as in $\mathcal{R}'$. Since in the lasso-shaped structure in $\mathcal{R}$ the only marked transitions are those starting in l_1 and t_1, we distinguish two more subcases:

  – Subcase $l_1 = t_1$: Using the orders from $s_1, \ldots, s_i[l_1, \ldots, l_k]$ and $r_1, \ldots, r_{i'}[t_1, \ldots, t_{k'}]$, let l_j and t_j be the first states such $l_j \neq t_j$ (with the possible exception of prefices). Since the sets of loop states equal, there are l_m and $t_{m'}$, such that $l_m = t_j$ and $t_{m'} = l_j$. We can now construct a path in $\mathcal{R}$, in which we can repeat the part between l_j and $(t_{m'} = l_j)$ infinitely and obtain a non-accepting run. Since $\mathcal{R}$ is deterministic, it is the only run over this word. As $\{c'_p\}.\{c, c'\}^\omega \subseteq L(\mathcal{R}')$, we get $L(\mathcal{R}') \neq L(\mathcal{R})$, which is again a contradiction.

$$l_1 \xrightarrow{c}_{\hat{m}} \dots \xrightarrow{c} l_j \xrightarrow{c} l_{j+1} \xrightarrow{c} \dots l_m = t_j \xrightarrow{c'} t_{j+1} \xrightarrow{c'} \dots \xrightarrow{c'} t_{m'} = l_j \xrightarrow{c} l_{j+1}$$

– Subcase $l_1 \neq t_1$: Let us construct a run in $\mathcal{R}$. It starts with c'_p and transfers to t_1 by repeating c'. From there, it starts taking transitions over c until reaching l_1. None of these transitions is marked since none of the states in between are l_1. From l_1, we start taking transitions over c' until reaching t_1. Again they are not marked. By repeating this procedure, we obtain a non-accepting run in $\mathcal{R}$. Since $\mathcal{R}$ is deterministic, it is the only run over this word. As $\{c'_p\}.\{c, c'\}^\omega \subseteq L(\mathcal{R}')$, we get $L(\mathcal{R}') \neq L(\mathcal{R})$, which is again a contradiction.

$$t_1 \xrightarrow{c'} t_j \xrightarrow{c'} \dots \xrightarrow{c'} t_{i'} \xrightarrow{c'} l_1 \xrightarrow{c} l_j \xrightarrow{c} \dots \xrightarrow{c} l_{j'} \xrightarrow{c} t_1$$

We have covered all possibilities and shown that all of them lead to a contradiction. Therefore, every tight TRA $\mathcal{R}'$, such that $L(\mathcal{R}') = L(\mathcal{R})$ must have at least one state for every $s_1, \dots, s_i[l_1, \dots, l_k]$. Resulting in

$$|Q'| \geq \sum_{j=2}^{n} \sum_{\substack{|i+k=j \\ k>0}} \cdot \binom{n}{i+k} \cdot (i+k)!$$

□

The result of the previous lemma is that there is only a tiny gap between the size of tight TRAs constructed by Algorithm 5.3 and the size of tight TRAs constructed by an optimal algorithm. The only difference in the upper bound given by Algorithm 5.3 and the lower bound given by Lemma 9 is the number $|\mathcal{F}|$. Compared to $n!$, the number of Rabin pairs is usually far lower, therefore there is not much space for improvements in theory. For completeness, in the upper bound, the formula is also multiplied by a constant c. However, it has no influence on the asymptotic complexity of the construction.

6 Reduction of tight automata

In our implementation of the tightening algorithm, we use existing methods that can reduce the state space of an automaton after the transformation. In this chapter, we formulate an exact argument on why the postprocessing we use does not violate tightness. Furthermore, there are some standard techniques that we cannot use. For these, we show a counterexample in the form of a tight automaton such that it is not tight after the reduction.

6.1 Preorders and quotients

Clemente and Mayr [13] created a collection of techniques for reducing the state space and number of transitions of non-deterministic automata over finite and infinite words. We provide the necessary definitions from their work and show what techniques can be used on tight automata.

A *preorder* $\sqsubseteq$ is a reflexive and transitive relation and a *strict partial order* $\sqsubset$ is an irreflexive, asymmetric and transitive relation. Every preorder defines an *induced equivalence* $\approx$ such that $\approx = \sqsubseteq \cap \sqsupseteq$. Given a state q, we denote by $[q]$ its equivalence class with respect to a fixed equivalence $\approx$. Furthermore, for every $P \subseteq Q$, by $[P]$ we denote the set of all equivalence classes of states in P, i.e. $[P] = \{[q] \mid q \in P\}$.

6.2 Good for quotienting relations

We focus on so-called quotienting, which is based on computing a simulation relation on the set of states and then merging the states from the same equivalence class. Given a BA $\mathcal{B} = (Q, \Sigma, \delta, I, F)$ and a preorder $\sqsubseteq$ on Q with its induced equivalence $\approx$, the *quotient* of $\mathcal{B}$ is the BA $\mathcal{B}/\sqsubseteq = ([Q], \Sigma, \delta', [I], [F])$, where $([q], a, [p]) \in \delta'$ whenever $(q, a, p) \in \delta$.

Definition 4. [13] *Given a preorder $\sqsubseteq$ and an BA $\mathcal{B}$, we say that $\sqsubseteq$ is* good for quotienting (GFQ) *if $L(\mathcal{B}) = L(\mathcal{B}/\sqsubseteq)$.*

Lemma 10. *Let $\mathcal{B}$ be a tight BA and let $\sqsubseteq$ be a GFQ preorder. An automaton $\mathcal{B}/\sqsubseteq$ is tight and $L(\mathcal{B}) = L(\mathcal{B}/\sqsubseteq)$.*

Proof. The language equivalence trivially follows from the definition of GFQ. Let us assume a lasso-shaped word u from $L(\mathcal{B})$. Then there exists an accepting run over u of $\mathcal{B}$ with $|minSL(u)| = |minSL(\rho)|$. Let ρ be of the form $\pi(\tau)^\omega$, where

$$\pi = q_0 \xrightarrow{minS(u)} l,\ \tau = l \xrightarrow{minL(u)} l$$

Clemente and Mayr show that if there is a run in the original automaton, then there is a run through the equivalence classes of the corresponding states in the quotient [13]. Hence there is an accepting run $\rho' = \pi'(\tau')^\omega$ in $\mathcal{B}/\sqsubseteq$, where

$$\pi' = [q_0] \xrightarrow{minS(u)} [l],\ \tau' = [l] \xrightarrow{minL(u)} [l]$$

It is easy to see that $|minSL(\rho')| \leq |minSL(\rho)|$. Therefore, the automaton is tight. □

The result of the previous lemma is that any GFQ relation can be used to reduce the state space of a tight automaton while preserving the tightness. Many relations are GFQ, for example, *direct* [14] and *delayed* [15] simulations. One can compute such relations in polynomial time [13]. Another approach that computes GFQ relation is described in Section 4.2 of Babiak et al. [16]. It is based on Moore's classical algorithm for minimizing finite automata. All of these reductions are implemented in ω-automata library Spot [17].

6.3 Good for pruning relations

While quotienting reduces the automaton by merging some of its states, so-called pruning excludes some transitions. Not only does it reduce the number of transitions, but after pruning, some of the states might remain unreachable or be unable to reach an accepting cycle. Therefore we can lower the state space too. The high-level idea is to compute relations on transitions implying that a transition is 'covered' by the other. Although pruning is a powerful tool to reduce omega-automata, we show that for some commonly used relations it does not preserve tightness.

Definition 5. [13] *Let $\mathcal{B} = (Q, \Sigma, \delta, I, F)$ be a BA, let $P \subseteq \delta \times \delta$ be a relation on δ, and let max P be the set of maximal elements of P, i.e.*

$$max\ P = \{(p, a, r) \in P \mid \nexists (p', a', r').((p, a, r), (p', a', r')) \in P\}$$

The pruned automaton *is defined as $Prune(\mathcal{B}, P) = (Q, \Sigma, \delta', I, F)$, where $\delta' = max\ P$.*

Definition 6. [13] *A relation $P \subseteq \delta \times \delta$ is* good for pruning *(GFP) if $L(Prune(\mathcal{B}, P)) = L(\mathcal{B})$, for every BA $\mathcal{B}$.*

The GFP relations that Clemente and Mayr [13] study, combine simulation relations on the states of $\mathcal{B}$. We present our results on forward simulations, but the same counterexample is feasible for backward simulations. Forward direct simulation on a pair of states (q_0, p_0) can be imagined as a game of two players, Spoiler and Duplicator. In the game, the players iteratively construct two traces π_0 and π_1 starting in q_0 and p_0. In the i-th turn and in the configuration (q_i, p_i), Spoiler picks a transition $q_i \xrightarrow{a} q_{i+1}$ and it is added to π_0. Afterwards, Duplicator responds by picking a matching transition $p_i \xrightarrow{a} p_{i+1}$ and it is added to π_1. The next configuration for $(i+1)$-turn is (q_{i+1}, p_{i+1}). The Duplicator wins *direct forward simulation* $\sqsubseteq^{di}$ iff $\forall i. q_i \in F \Rightarrow p_i \in F$. In other words, in every step, Spoiler moves to an accepting state, Duplicator must too. Finally, $q_0 \sqsubseteq^{di} p_0$ holds iff Duplicator has a winning strategy for a game starting in (q_0, p_0).

Furthermore, regarding forward simulations and property GFP, Clemente and Mayr [13] define a more important relation $\subseteq^{di}$. It is defined similarly as $\sqsubseteq^{di}$, but Spoiler firstly reveals his whole infinite path, and then Duplicator must match the path. In other words, $q \subseteq^{di} p$ holds if, for every infinite word $w = u_0, u_1, \ldots$ and every run $q \xrightarrow{u_0} q_1 \xrightarrow{u_1} q_2 \ldots$, there is a run $p \xrightarrow{u_0} p_1 \xrightarrow{u_1} p_2 \ldots$, such that $\forall i. q_i \in F \Rightarrow p_i \in F$.

They build relations on transitions from those on the states in the following way. Let $R_b, R_f \subseteq Q \times Q$, then

$$P(R_b, R_f) = \{((p, a, r), (p', a, r')) \in \delta \times \delta \mid p\ R_b\ p' \text{ and } r\ R_f\ r'\}$$

Clemente and Mayr (Theorem 5.3) [13], show that for every strict partial order R, such that $R \subseteq \subseteq^{di}$, relation $P(id, R)$ is GFP. Let $q \sqsubset^{di} p$ if

$q \sqsubseteq^{di} p$ and $p \not\sqsubseteq^{di} q$. It is easy to see that $\sqsubset^{di} \subseteq \subseteq^{di}$ and every reasonable relation on states given by forward simulation subsumes $\sqsubset^{di}$ (Figure 3 [13]). Then especially, $P(id, \sqsubset^{di})$ is GFP, which is a long known relation commonly used for pruning. By the following lemma, we show that every strict partial order R between $\sqsubset^{di}$ and $\subseteq^{di}$, that could be used in $P(id, R)$ to prune transitions, violates tightness.

Lemma 11. *For every strict partial order R, such that $\sqsubset^{di} \subseteq R \subseteq \subseteq^{di}$, relation $P(id, R)$ is GFP and there is a tight BA $\mathcal{B}$ such that $Prune(\mathcal{B}, P(id, R))$ is not tight.*

Proof. Observation that $P(id, R)$ is GFP directly follows from Theorem 5.3 [13]. Consider tight BA $\mathcal{B}$ given by Figure.

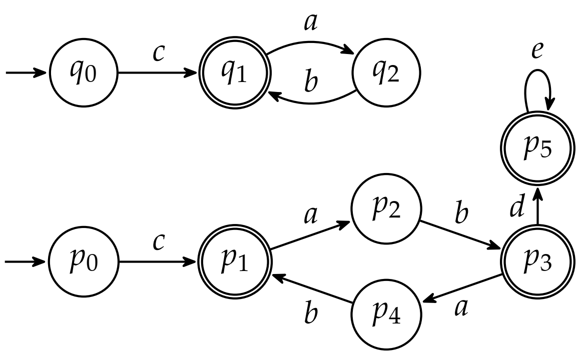


Figure 6.1: Tight BA $\mathcal{B}$ for which $Prune(\mathcal{B}, P(id, R))$ is not tight.

It is easy to see that $q_1 \sqsubset^{di} p_1$, since Spoiler can only loop over ab infinitely and every even transition ends in the accepting state. Duplicator can match this too by the cycle $(p_1 \xrightarrow{a} p_2 \xrightarrow{b} p_3 \xrightarrow{a} p_4 \xrightarrow{b} p_1)^\omega$. Furthermore, $p_1 \not\sqsubseteq^{di} q_1$, since Spoiler can take the transition over d, which Duplicator cannot match.

Since $\sqsubset^{di} \subseteq R$ then $q_1 \; R \; p_1$ holds. An easy observation is that $q_0 \; id \; p_0$. Therefore, $((q_0, c, q_1), (p_0, c, p_1)) \in P(id, R)$. Set *max* $P(id, R)$ does not contain (q_0, c, q_1) because there exists (p_0, c, p_1) that 'covers' it. In conclusion $Prune(\mathcal{B}, P(id, R))$ does not contain (q_0, c, q_1) in its $\delta = max\ P(id, R)$. But then, there is no tight run over $c(ab)^\omega$. □

7 Implementation and evaluation

We implemented the algorithm for tightening BA into a tool called **Tightener**. Even though the presented algorithm from Chapter 4 works with TBA, we adjusted it to work with BA. The reason is that the practical cases that work with tight automata use state-based acceptance [2, 1]. Furthermore, the only known implemented tool [7] for transforming automata to tight automata is the one from our previous work, which works with BA. Therefore, we can compare the new implementation with the old one on the same dataset. Our implementation also allows other types of automata as input but outputs BA.

7.1 Implementation details

The language we use for the implementation is Python 3.8.15. It is built upon a library for ω-automata - Spot [17]. The version of Spot used in the implementation is 2.11.4. Spot provides state-of-the-art LTL to BA conversion [18], which we can connect with our tightening algorithm. We evaluate such a method of obtaining tight automata from LTL in Section 7.2, where we compare it with the direct construction of tight automata from LTL using the CGH algorithm [4]. Another advantage is the postprocessing of the automaton. For the state space reduction, we use Spot's *reduce_direct_sim_sba*(), which computes a direct simulation described by Clemente and Mayr [13] and then constructs the quotient. Such simulation relation is GFQ. Therefore as proved in Lemma 10 it preserves tightness.

One issue with Spot is that it does not support multiple initial states, and by Lemma 2, we usually need more than one initial state. Therefore, we add a new initial state and connect it by a fresh symbol to all initial states, when doing the postprocessing. Afterwards, we remove such symbol and state from the resulting automaton. When outputting the tight BA in HOA format [19], the initial states are part of the output since HOA supports multiple initial states.

Source code of **Tightener** is in the attachments of this work[1]. The project contains two representations of the tool. The first one is Jupyter

1. It is also available at https://gitlab.fi.muni.cz/xjankola/tightener-diploma-thesis

Notebook, which includes the tool's whole code, divided logically into cells. There is also a closer description of each part. The second one is a Python project. The requirements and how to install them are listed in the repository. It needs a Python interpreter of version 3.8 and more.

Flag	Description
-h, - -help	Outputs possible inputs and their description
-r, - -reduces	Uses quotienting on the output automaton
-o - -outputHOA	Outputs the tight automaton in HOA format
-f, - -formula	"input" is in the form of a LTL formula, for example "F(p1|p2)"
-F, - -file	"input" is in the form of a path to .txt file containing LTL formulae
-a, - -HOA	"input" is in the form of a path to .aut file containing an input automaton in HOA format

Table 7.1: Possible flags for **Tightener** and their description.

The project can then be run from the folder *Tightener_project* using the following line: **python Tightener.py [flags] "input"**. An input has one of the forms based on the flags as depicted in Table 7.1.

Input can be an LTL formula in the syntactic form defined by Spot[2], a file with LTL formulae, or an automaton in HOA format. The output is a graph in dot format[3]. Or as an alternative (when setting the **-o** flag) one can output the automaton in HOA format.

7.2 Evaluation

The known motivation usages for tight automata (model checking [10], synthesis of strategies [2]) work with a given property specification. The specification is usually modelled as an LTL formula or directly as BA. The formalism of the specification is then translated into a tight BA. We divide the evaluation of our algorithm into three questions based on the input formalism for our algorithm.

2. Correct syntactic form can be found at https://spot.lre.epita.fr/ioltl.html
3. Can be visualized for example at https://dreampuf.github.io/GraphvizOnline/

- (**Q1**) Given a BA, can our algorithm construct smaller tight BAs than the known implementations?
- (**Q2**) How much can the state space be lowered when using the presented postprocessing?
- (**Q3**) Given an LTL formula, is an approach of firstly using state-of-the-art LTL translation to BA and then using Tightener to make it tight advantageous in some cases, rather than constructing a tight automaton directly from LTL?

All experiments ran on the same machine with Intel Core i7-8750H CPU and with 16GB RAM. The time-out threshold was set to 10 minutes.

7.2.1 Experiments with respect to Q1 and Q2

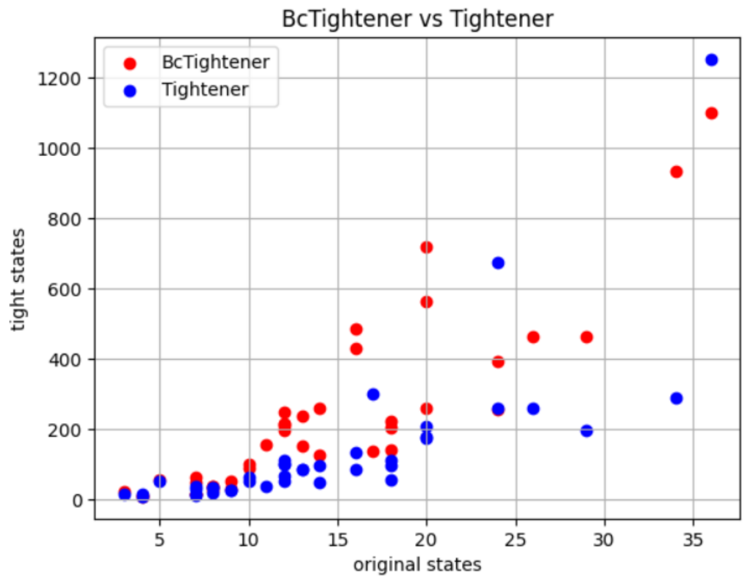


Figure 7.1: Numbers of states of the tight BAs compared to the original BAs.

The only known practical implementation of directly translating a BA into a tight BA is the implementation developed as a part of the

Bachelor thesis of the author of this Master thesis [7]. Let us call the tool by **BcTightener**. Since the idea of the algorithm extensively uses finite automata, BcTightener is written in Java with use of a well-supported finite automata library. However, it cannot work with the HOA representation (for neither input nor output automata). Therefore, we cannot test it on a data set generated by Spot. Instead, we use the dataset introduced in the Bachelor thesis [7] for the evaluation of experiments on BcTightener. It consists of more than 40 BAs with number of states ranging between 2-40.

There were no time-outs by any of the compared tools in this subsection. In the following two figures, we compare BcTightener with all optimisations presented in the Bachelor thesis against Tightener without postprocessing. As you can see in Figure 7.1, both tools have similar raise of state space with respect to the state space of the input automata. However, comparing the state spaces of individual automata constructed by the tools on same inputs. Figure 7.2 shows that Tightener outperforms BcTightener in most cases.

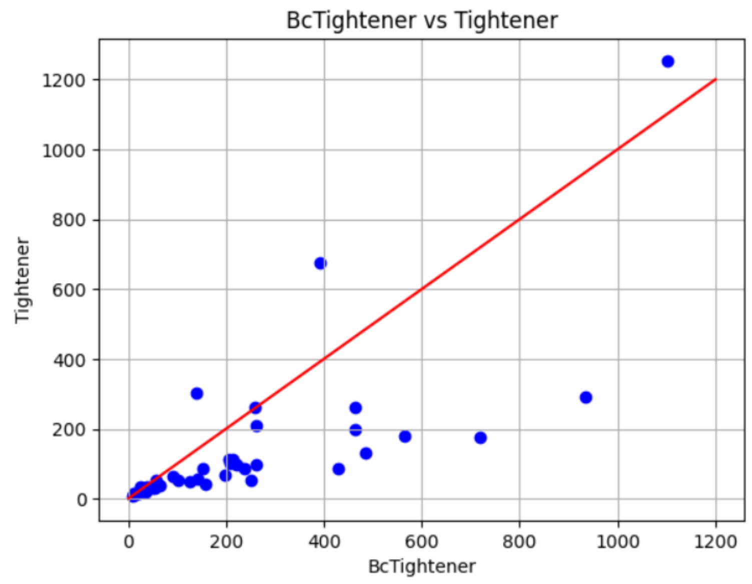


Figure 7.2: Comparison of BcTightener and Tightener.

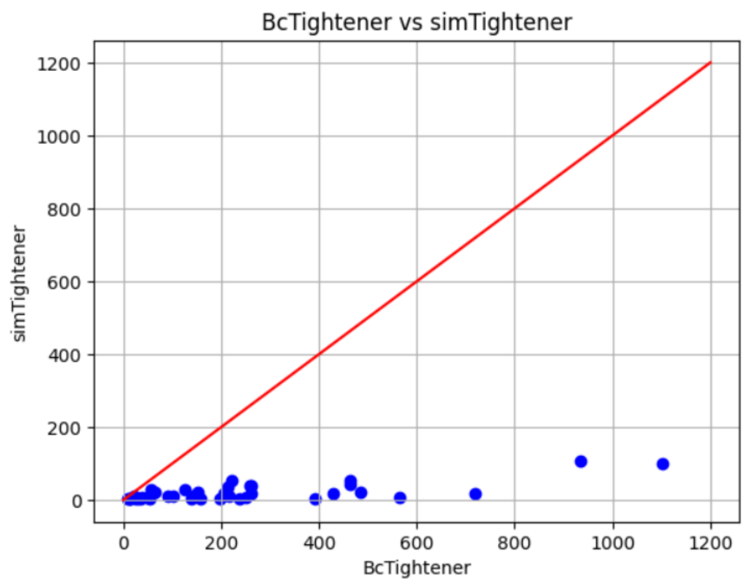


Figure 7.3: Comparison of BcTightener and Tightener with direct simulation postprocessing.

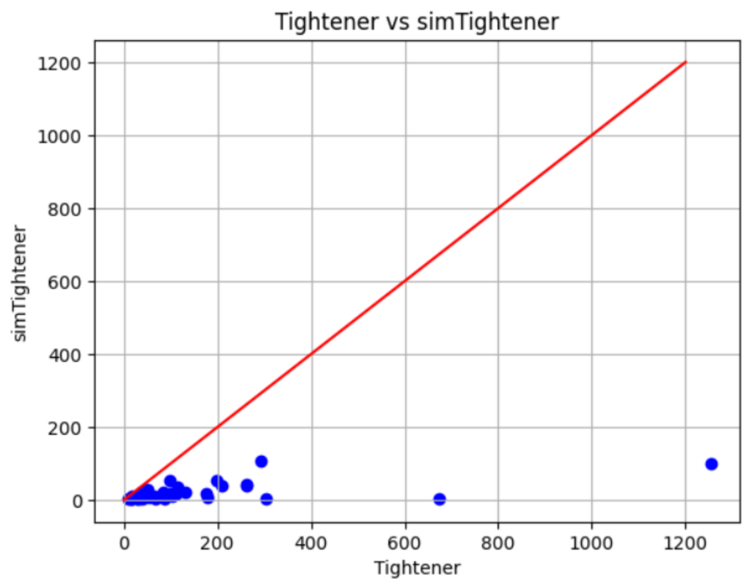


Figure 7.4: Comparison of Tightener and simTightener to provide an answer for Q2.

Figure 7.3 presents the most substantial result. By **simTightener** we call Tightener with reduced state space using quotienting by direct simulation. It shows that we can lower the state space significantly compared to the so far known approach. Furthermore, Figure 7.4 presents that the proposed postprocessing using direct simulation reduces the state space of tight BA remarkably.

In conclusion, we formulate an answer to Q1 and Q2. Our tool Tightener outperforms BcTightener as the only other known tool for conversion of BA to tight BA. Furthermore, the results from Figure 7.3 and Figure 7.4 show that our proposed postprocessing by quotienting reduces the state space of tight BAs significantly. Another advantage is that this postprocessing is universal and does not depend on the tightening algorithm. Hence, it can be used by any tool that produces tight BA.

7.2.2 Experiments with respect to Q3

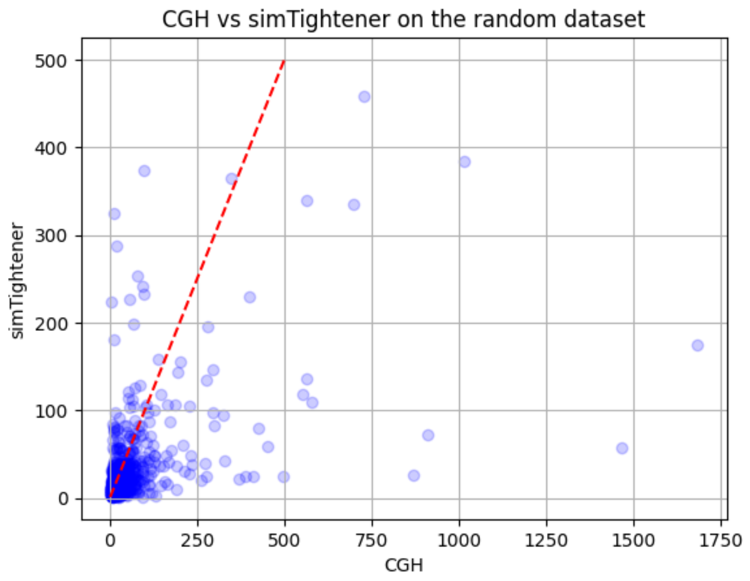


Figure 7.5: Comparison of simTightener and CGH on the random dataset.

The well-known LTL translator to tight BA is an algorithm from Clarke et al.'s work [4]. Schuppan and Biere proved that it constructs tight BAs [1]. Since it produces symbolic automata, we implemented an explicit construction from these symbolic representations. We used SMT solver Z3 [20] to prune unreachable and contradictory states and count the number of reachable states. In the experiments, we refer to the results of this algorithm as **CGH**.

Regarding our tool, we translate an LTL formula using Spot to BA. Afterwards, we convert it using Tightener to tight BA. Furthermore, we reduce the state space using direct simulations. We evaluate the experiments on two datasets.

One of the datasets is composed of 952 formulae, and it was generated using **rand_ltl** Spot's function for generating random formulae. It is stored in *ltlDataSet_random.txt* in our repository.

The second dataset consists of 219 formulae with patterns from literature[21, 22, 23, 24, 25, 26, 27, 28, 29]. We obtained it from Spot by **gen_ltl**. It is stored in *ltlDataSet_pattern.txt* in our repository.

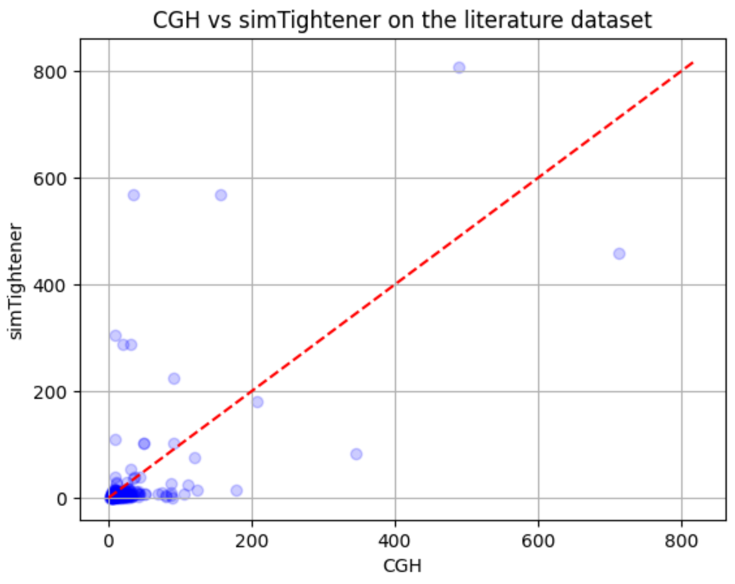


Figure 7.6: Comparison of simTightener and CGH on the dataset from literature.

Both graphs in Figure 7.5 and Figure 7.6 show that direct translation CGH from LTL to tight BA proves less efficient in many cases. However, simTightener constructs larger BAs for some formulae. The following table summarizes the concrete numbers.

Tool	Rand.	Rand. (%)	Lit.	Lit. (%)
simTightener	611 (32)	64%	164 (10)	75%
CGH	317	33%	45	21%
Tightener	383 (32)	40%	125 (10)	57%
CGH	540	57%	80	37%

Table 7.2: The table consists of four comparisons. We compared Tightener with and without postprocessing to CGH. The absolute values are the number of cases in which the tool provided tight BA with fewer states. The red numbers indicate the number of time-outs. Notice that CGH had no time-outs. The percentage columns display the number of better cases relative to the size of the dataset. The second and the fourth row represent the same tool, but they just differ in the comparisons of CGH to Tightner and simTightener, respectively. The blue highlights the winners of the comparison.

Even though Tightener lost one comparison, one can see that when using the postprocessing, simTightener won all the comparisons. Furthermore, in the comparison, where Tightener lost, it could construct smaller BAs in 40% of the cases, which is a significant number of automata.

To conclude this section, we discuss the answer for Q3. There are cases where simTightener produces larger automata than CGH. However, it constructs smaller automata on average. Furthermore, as shown in Figure 7.5, in many instances in which simTightener outputs smaller BA, the automaton produced by CGH has significantly more states. Our tool simTightener performed better and experimentally proved that translating LTL using a state-of-the-art translator and then tightening the automaton is probably more effective than direct translation to a tight automaton. In practice, using both tools and picking the best result is advantageous.

8 Conclusion

In this work, we have extended the property of tight automata to tight TELA, which is the universal class of ω-automata. We proved the existence of a lasso-shaped accepting run for each lasso-shaped word with a stated upper bound on a number of pairwise different v-significant states, which is essential for the completeness of our algorithm. Further, we showed that this boundary is tight. Later, we presented a new approach for converting TBAs to tight TBAs. We proved that this approach is correct and that the asymptotical rise of state space is $O(f)$, where the leading function of f is $(n!)^3$, which is the smallest upper bound so far reached. Furthermore, we have extended this construction for transforming TRAs to tight TRAs. We have again proved the correctness and analyzed the state space rise. We have also shown a lower bound for the number of states needed to have a tight TRA. The only gap between the upper and lower bound is the number of Rabin pairs $|\mathcal{F}|$.

The experiments showed that our new tool Tightener performed much better than our previous tool BcTightener in converting BA to tight BA. When transforming LTL to tight BA, tool CGH constructed larger automata in more cases than Tightener grouped with Spot's LTL translator. Therefore, the proposed approach of first translating an LTL formula and then tightening the automaton is experimentally a better approach for an explicit LTL model checking than a direct translation of LTL to a tight automaton.

8.1 Future work

As we can see from Chapter 5, there is little space left for improvement in the worst-case input, not only for TRA but also for TBA, since the automaton used in Lemma 9 for showing the lower bound was also TBA. However, in practice, we see a few ways for improvement. The first, more obvious way is to observe a certain useful LTL formulae subclass and the BAs constructed after translation. For instance, there might be some subclass where the repetition of the minimal loop that violates the tightness happens only in the stem of runs.

The second way is that we do not have to construct the whole state space of the tight automaton. In both LTL model checking and LTL strategy synthesis, a tight automaton is used so the product of the model and the ω-automaton does not add some 'unnecessary long lasso-shaped structures'. We want to research some way of detecting the repetition of the minimal loop on-the-fly, during the construction of the product. The possible way might be to label the states of ω-automaton by LTL subformulae that they satisfy (some LTL translators can do that). When constructing the product, we could use this symbolical labelling to detect what formula is implied by another one. That way, we could notice the paths over the same words from two different states.

Bibliography

1. SCHUPPAN, Viktor; BIERE, Armin. Shortest Counterexamples for Symbolic Model Checking of LTL with Past. In: HALBWACHS, Nicolas; ZUCK, Lenore D. (eds.). *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*. Springer, 2005, vol. 3440, pp. 493–509. Lecture Notes in Computer Science. Available also from: `https://doi.org/10.1007/978-3-540-31980-1%5C_32`.
2. TUMOVA, Jana; MARZINOTTO, Alejandro; DIMAROGONAS, Dimos V.; KRAGIC, Danica. Maximally satisfying LTL action planning. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 1503–1510. Available from DOI: `10.1109/IROS.2014.6942755`.
3. KUPFERMAN, Orna; VARDI, Moshe Y. Model Checking of Safety Properties. In: HALBWACHS, N.; PELED, D. (eds.). *Proceedinfs of the 11th International Conference on Computer Aided Verification (CAV'99)*. Springer-Verlag, 1999, vol. 1633, pp. 172–183. Lecture Notes in Computer Science.
4. CLARKE, E.; GRUMBERG, O.; HAMAGUCHI, K. Another Look at LTL Model Checking. 1993. Available from DOI: `10.21236/ada277568`.
5. SCHUPPAN, Victor. *Liveness checking as safety checking to find shortest counterexamples to linear time properties*. 2006. Available also from: `http://www.schuppan.de/viktor/VSchuppan-PhD-2006.pdf`.
6. EHLERS, Rüdiger. How Hard Is Finding Shortest Counter-Example Lassos in Model Checking? In: BEEK, Maurice H. ter; MCIVER, Annabelle; OLIVEIRA, José N. (eds.). *Formal Methods – The Next 30 Years*. Cham: Springer International Publishing, 2019, pp. 245–261. ISBN 978-3-030-30942-8.

7. JANKOLA, Marek. *Transformation of Nondeterministic Büchi Automata to Tight Automata* [*online*]. 2021 [cit. 2023-05-10]. Available also from: `https://is.muni.cz/th/o0p4b/`. SUPERVISOR : Jan Strejček.

8. KUPFERMAN, Orna; MORGENSTERN, Gila; MURANO, Aniello. Typeness for omega-Regular Automata. In: 2004, vol. 17, pp. 324–338. Available from DOI: `10.1142/S0129054106004157`.

9. KŘETÍNSKÝ, Jan; MEGGENDORFER, Tobias; SICKERT, Salomon; ZIEGLER, Christopher. Rabinizer 4: From LTL to Your Favourite Deterministic Automaton. In: CHOCKLER, Hana; WEISSENBACHER, Georg (eds.). *Proceedings of the 30th International Conference on Computer Aided Verification (CAV'18)*. Springer, 2018, vol. 10981, pp. 567–577. Lecture Notes in Computer Science.

10. HOLZMANN, Gerard J. Explicit-State Model Checking. In: *Handbook of Model Checking*. Ed. by CLARKE, Edmund M.; HENZINGER, Thomas A.; VEITH, Helmut; BLOEM, Roderick. Cham: Springer International Publishing, 2018, pp. 153–171. ISBN 978-3-319-10575-8. Available from DOI: `10.1007/978-3-319-10575-8_5`.

11. CALBRIX, Hugues; NIVAT, Maurice; PODELSKI, Andreas. Ultimately periodic words of rational ω-languages. In: BROOKES, Stephen; MAIN, Michael; MELTON, Austin; MISLOVE, Michael; SCHMIDT, David (eds.). *Mathematical Foundations of Programming Semantics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 554–566. ISBN 978-3-540-48419-6.

12. FARZAN, Azadeh; CHEN, Yu-Fang; CLARKE, Edmund M.; TSAY, Yih-Kuen; WANG, Bow-Yaw. Extending Automated Compositional Verification to the Full Class of Omega-Regular Languages. In: RAMAKRISHNAN, C. R.; REHOF, Jakob (eds.). *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 2–17. ISBN 978-3-540-78800-3.

13. CLEMENTE, Lorenzo; MAYR, Richard. Efficient reduction of nondeterministic automata with application to language inclusion testing. *Logical Methods in Computer Science ; Volume 15*. 2019,

Issue 1, 1860–5974. Available from DOI: `10.23638/LMCS-15(1:12)2019`.

14. DILL, David L.; HU, Alan J.; WONG-TOI, Howard. Checking for language inclusion using simulation preorders. In: LARSEN, Kim G.; SKOU, Arne (eds.). *Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 255–265. ISBN 978-3-540-46763-2.
15. ETESSAMI, Kousha; WILKE, Thomas; SCHULLER, Rebecca A. Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. *SIAM Journal on Computing*. 2005, vol. 34, no. 5, pp. 1159–1175. Available from DOI: `10.1137/S0097539703420675`.
16. BABIAK, Tomáš; BADIE, Thomas; DURET-LUTZ, Alexandre; KŘETÍNSKÝ, Mojmír; STREJČEK, Jan. Compositional Approach to Suspension and Other Improvements to LTL Translation. In: BARTOCCI, Ezio; RAMAKRISHNAN, C. R. (eds.). *Model Checking Software*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 81–98.
17. DURET-LUTZ, Alexandre; KORDON, Fabrice; POITRENAUD, Denis; RENAULT, Etienne. Heuristics for Checking Liveness Properties with Partial Order Reductions. In: *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*. Springer, 2016, vol. 9938, pp. 340–356. Lecture Notes in Computer Science. Available from DOI: `10.1007/978-3-319-46520-3_22`.
18. BABIAK, Tomáš; KŘETÍNSKÝ, Mojmír; ŘEHÁK, Vojtěch; STREJČEK, Jan. LTL to Büchi Automata Translation: Fast and More Deterministic. In: *Proc. of the 18th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*. Springer, 2012, vol. 7214, pp. 95–109. LNCS.
19. BABIAK, Tomáš; BLAHOUDEK, František; DURET-LUTZ, Alexandre; KLEIN, Joachim; KŘETINSKÝ, Jan; MÜLLER, David; PARKER, David; STREJČEK, Jan. The Hanoi Omega-Automata Format. In: *Proceedings of the 27th Conference on Computer Aided Verification (CAV'15)*. Springer, 2015, vol. 8172,

pp. 442–445. Lecture Notes in Computer Science. See also `http://adl.github.io/hoaf/`.

20. MOURA, Leonardo de; BJØRNER, Nikolaj. Z3: An Efficient SMT Solver. In: RAMAKRISHNAN, C. R.; REHOF, Jakob (eds.). *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

21. GELDENHUYS, Jaco; HANSEN, Henri. Larger Automata and Less Work for LTL Model Checking. In: *Proceedings of the 13th International SPIN Workshop (SPIN'06)*. Springer, 2006, vol. 3925, pp. 53–70. Lecture Notes in Computer Science.

22. CICHOŃ, Jacek; CZUBAK, Adam; JASIŃSKI, Andrzej. Minimal Büchi Automata for Certain Classes of LTL Formulas. In: *Proceedings of the Fourth International Conference on Dependability of Computer Systems (DEPCOS'09)*. IEEE Computer Society, 2009, pp. 17–24.

23. DWYER, Matthew B.; AVRUNIN, George S.; CORBETT, James C. Property Specification Patterns for Finite-state Verification. In: ARDIS, Mark (ed.). *Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP'98)*. New York: ACM Press, 1998, pp. 7–15.

24. ETESSAMI, Kousha; HOLZMANN, Gerard J. Optimizing Büchi Automata. In: PALAMIDESSI, C. (ed.). *Proceedings of the 11th International Conference on Concurrency Theory (Concur'00)*. Pennsylvania, USA: Springer-Verlag, 2000, vol. 1877, pp. 153–167. Lecture Notes in Computer Science.

25. HOLEČEK, Jan; KRATOCHVÍLA, Tomáš; ŘEHÁK, Vojtech; ŠAFRÁNEK, David; ŠIMEČEK, Pavel. *Verification Results in Liberouter Project*. 2004-09. Tech. rep., 03, 32pp. CESNET.

26. PELÁNEK, Radek. BEEM: benchmarks for explicit model checkers. In: *Proceedings of the 14th international SPIN conference on Model checking software*. Springer-Verlag, 2007, pp. 263–267. Lecture Notes in Computer Science.

27. SOMENZI, Fabio; BLOEM, Roderick. Efficient Büchi Automata for LTL Formulæ. In: *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*. Chicago, Illinois, USA: Springer-Verlag, 2000, vol. 1855, pp. 247–263. Lecture Notes in Computer Science.

28. GASTIN, Paul; ODDOUX, Denis. Fast LTL to Büchi Automata Translation. In: BERRY, G.; COMON, H.; FINKEL, A. (eds.). *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*. Paris, France: Springer-Verlag, 2001, vol. 2102, pp. 53–65. Lecture Notes in Computer Science.

29. TABAKOV, Deian; VARDI, Moshe Y. Optimized Temporal Monitors for SystemC. In: *Proceedings of the 1st International Conference on Runtime Verification (RV'10)*. Springer, 2010, vol. 6418, pp. 436–451. Lecture Notes in Computer Science.