

Complementing Büchi Automata with a Subset-tuple Construction

Joel Allred

Department of Informatics
University of Fribourg, Switzerland
E-Mail: joel.allred@unifr.ch

Ulrich Ultes-Nitsche

Department of Informatics
University of Fribourg, Switzerland
E-Mail: uun@unifr.ch

Abstract—Complementation of Büchi automata is well known for being difficult. In the worst case, a state-space growth of $(0.76n)^n$ is unavoidable. Recent studies suggest that “simpler” algorithms perform better than more involved ones on practical cases. In this paper, we present a simple “direct” algorithm for complementing Büchi automata. It involves a structured subset construction (using tuples of subsets of states) that produces a deterministic automaton. This construction leads to a complementation procedure that resembles the straightforward complementation algorithm for deterministic Büchi automata, the latter algorithm actually being a special case of our construction.

I. INTRODUCTION

Checking ω -language containment is important in linear-time temporal verification. For ω -languages B and P , testing $B \subseteq P$ is done algorithmically by testing $B \cap \overline{P} = \emptyset$, where \overline{P} is the complement of P . If P is regular and represented by a Büchi-automaton, a Büchi-automaton representing \overline{P} can be constructed effectively. In the worst case, the automaton for \overline{P} has $(0.76n)^n$ many states [1], where n is the number of states of the automaton for P .

Complementation of Büchi automata is difficult because in general, Büchi automata cannot be made deterministic. Complementation of a deterministic Büchi automaton \mathcal{A} is, however, quite simple: construct a copy of \mathcal{A} , remove all accepting states in the copy, make the remaining states in the copy accepting states, make all states in \mathcal{A} non-accepting, and finally allow nondeterministically moving from \mathcal{A} to corresponding states in its copy. So the complement automaton to a deterministic Büchi automaton consists of two deterministic automata (let’s call them an upper and a lower automaton, if we assume that one is drawn below the other) with transitions from upper to the lower automaton but no transitions back from the lower to the upper one. We adapt this construction to nondeterministic automata by constructing a deterministic version of a given automaton that then Büchi-accepts a different ω -language but can still be used in a construction similar to the one for deterministic Büchi-automata to produce a complement automaton. It results again in an automaton consisting basically of two deterministic automata that are then combined by allowing to move nondeterministically from the

upper to the lower one but not vice versa.

The run analysis that we developed independently as some similarities with the analysis by Fogarty, Kupferman, Vardi, and Wilke [2] who elegantly translate the slice-based approach in [3] to a rank-based approach. Instead of unifying interim constructions, we directly construct a complement automaton using the sole analysis of the runs of the original automaton, in a similar way deterministic Büchi automata are complemented.

When developing our construction, the key notion will be that of a *greedy* accepting run that we use in this paper. In short, a greedy run is a run that always aims at reaching an accepting state as quickly as possible. Our construction will consider only greedy runs. By doing so, visits of sets of accepting states in runs of the constructed automaton allow faithfully to identify whether or not they can also occur in a (greedy) run of the given automaton. To consider only greedy runs in a deterministic version of a given Büchi-automaton, we basically conduct a subset construction to which we add sufficient structure to identify greedy runs: the states in the construction are tuples of sets of states of the given automaton, sets of non-accepting states and sets of accepting states are never mixed (to avoid losing too much information [4]).

From the so constructed deterministic automaton, by identifying a particular property of state sets occurring in the state-set tuples (they are either *continued* or *discontinued*), we can construct a nondeterministic one accepting the complement ω -language of the given Büchi automaton in a similar way as for deterministic Büchi automata. The usual complementation procedure for deterministic Büchi automata (Section 3 in [5]) will be a special case of our construction.

II. PRELIMINARIES

Let Σ be an alphabet (set of finite cardinality). The set of all ω -words (infinitely long words) over Σ is designated by Σ^ω . Subsets of Σ^ω are called ω -languages over Σ .

Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, q_{in}, F)$ consists of finite set Q of states, input alphabet Σ , transition function $\delta : Q \times \Sigma \rightarrow 2^Q$, initial state $q_{in} \in Q$, and set $F \subseteq Q$ of accepting states.

δ is extended in the usual way to finite words (instead of only single symbols) and sets of state (instead of only single states).

Let $x = x(0)x(1)x(2)\cdots \in \Sigma^\omega$. A run r of \mathcal{A} on x is a sequence $r(0)r(1)r(2)\cdots$ of states such that $r(0) = q_{in}$ and $r(i+1) \in \delta(r(i), x(i))$, for all $i \geq 0$. Runs on finite words are defined similarly.

Let $\omega(r)$ be the set of all states that recur infinitely often in r . Run r is accepting if and only if $\omega(r) \cap F \neq \emptyset$. Automaton \mathcal{A} Büchi-accepts x if and only if there exists an accepting run of \mathcal{A} on x [6], [7]. The set of all ω -words accepted by \mathcal{A} is the ω -language that \mathcal{A} accepts.

If, for all $q \in Q$ and $a \in \Sigma$, $\delta(q, a)$ is a singleton set or the empty set, then \mathcal{A} is called deterministic. Otherwise it is called nondeterministic. For ω -languages, the language classes accepted by deterministic and nondeterministic automata differ: There exist ω -languages that can only be represented by nondeterministic finite automata [6], [7].

We call a (finite) run $r = r(0)r(1)r(2)\cdots r(n)$ on some finite word *greedy* if it always tries to reach, on its way to $r(n)$, accepting states as quickly as possible. To define greediness formally, let $\alpha : Q \rightarrow \{0, 1\}$ be a function that assigns 1 to a state if and only if it is accepting. α can be extended in the usual way to a function $\alpha : Q^* \rightarrow \{0, 1\}^*$ on sequences of states by its stepwise application.

We interpret words over $\{0, 1\}$ as binary numbers with the most significant bit to the left. We define that $r = r(0)r(1)r(2)\cdots r(n)$ on finite word w is *greedy* if and only if there does not exist a finite run $r' = r(0)r'(1)r'(2)\cdots r'(n-1)r(n)$ on w (note that r and r' coincide in their first and last state) such that $\alpha(r) < \alpha(r')$.

The definition of greediness can be extended to infinite runs r by defining that infinite run r is greedy if and only if all of its finite prefixes are greedy. It is to be noted that the notion of greediness similar to *lexicographically maximal runs*, as presented in [2].

III. THE SUBSET-TUPLE CONSTRUCTION

A. Definitions

Let $\mathcal{A} = (Q, \Sigma, \delta, q_{in}, F)$ be a Büchi automaton. We construct an interim deterministic Büchi-automaton $\mathcal{A}' = (Q', \Sigma, \delta', (\{q_{in}\}), \emptyset)$ from which we will later derive the complement automaton of \mathcal{A} . The state set of \mathcal{A}' contains non-empty disjoint sets of \mathcal{A} -states:¹

$$Q' = \bigcup_{m=1}^{|Q|} \{(S_1, \dots, S_m) \in (2^Q \setminus \{\emptyset\})^m \mid (\forall 1 \leq j < k \leq m : S_j \cap S_k = \emptyset)\}.$$

\mathcal{A}' does not have accepting states as \mathcal{A}' 's acceptance condition will not influence the construction of the complement automaton. We will use some other observation about runs of \mathcal{A}' to define the acceptance condition when finally constructing the complement automaton.

We define transition function $\delta' : Q' \times \Sigma \rightarrow Q'$ subsequently. Let $p' \in Q'$ and let $m = |p'|$. Let $p'(j)$ be the j th component of p' (note that $p'(j)$ is a set of \mathcal{A} -states). For $a \in \Sigma$, we define the a -successor of the j th component of p' to be

$$\sigma(p', j, a) = \delta(p'(j), a) \setminus \bigcup_{k=j+1}^m \delta(p'(k), a).$$

$\sigma(p', j, a)$ contains all \mathcal{A} -states that are a -successors of \mathcal{A} -states in $p'(j)$ and not already contained in $\sigma(p', k, a)$, for $k > j$. From this definition, we get immediately that sets $\sigma(p', j, a)$ and $\sigma(p', k, a)$ are disjoint.

The definition of $\sigma(p', j, a)$ guarantees that if an \mathcal{A} -state could occur in multiple components of a tuple, we will keep it only in the rightmost component. Note that even though $\delta(p'(j), a)$ may not be empty, $\sigma(p', j, a)$ still can be empty. We partition each set $\sigma(p', j, a)$ into non-accepting \mathcal{A} -states and accepting \mathcal{A} -states:

$$\begin{aligned} \sigma_n(p', j, a) &= \sigma(p', j, a) \cap (Q \setminus F), \\ \sigma_a(p', j, a) &= \sigma(p', j, a) \cap F. \end{aligned}$$

As we just partitioned the sets $\sigma(p', j, a)$, the resulting sets are still pairwise disjoint.

We put $\sigma_a(p', j, a)$ to the right of $\sigma_n(p', j, a)$ and remove all empty sets. We define the transition function of \mathcal{A}' :

$$\delta'(p', a) = q',$$

where q' is obtained by removing all empty sets in

$$(\sigma_n(p', 1, a), \sigma_a(p', 1, a), \dots, \sigma_n(p', m, a), \sigma_a(p', m, a))$$

but otherwise keeping the order of the non-empty components.

Lemma 1. *For all reachable states $p' \in Q'$ in \mathcal{A}' and all $1 \leq j < k \leq |p'|$, $p'(j)$ and $p'(k)$ are nonempty and disjoint.*

B. Example

We take the automaton in Figure 1 as an example.

Automaton \mathcal{A} in Figure 1 Büchi-accepts all ω -words over $\{a, b\}$ that contain finitely many occurrences of symbol a .

¹By “ \mathcal{A} -states” we refer to the states of automaton \mathcal{A} .

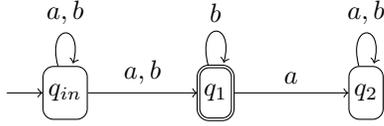


Fig. 1. Büchi automaton \mathcal{A} accepting $\{a, b\}^* \cdot \{b\}^\omega$.

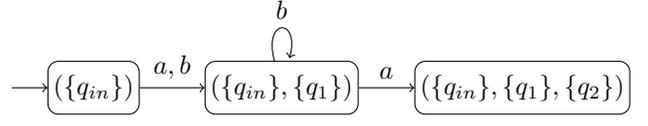


Fig. 4. Initial fragment of \mathcal{A}' .

From \mathcal{A} , we construct \mathcal{A}' stepwise. The initial state of \mathcal{A}' is the 1-tuple that contains \mathcal{A} -state set $\{q_{in}\}$ (see Figure 2).

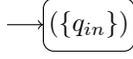


Fig. 2. Initial fragment of \mathcal{A}' .

With respect to \mathcal{A}' 's transition relation, we get

$$\begin{aligned}\sigma(\{q_{in}\}, 1, a) &= \delta(\{q_{in}\}, a) = \{q_{in}, q_1\}, \\ \sigma(\{q_{in}\}, 1, b) &= \delta(\{q_{in}\}, b) = \{q_{in}, q_1\}.\end{aligned}$$

Because $\{q_{in}, q_1\}$ contains non-accepting \mathcal{A} -state q_{in} as well as accepting \mathcal{A} -state q_1 , $\{q_{in}, q_1\}$ is partitioned into two sets:

$$\begin{aligned}\sigma_n(\{q_{in}\}, 1, a) &= \sigma_n(\{q_{in}\}, 1, b) = \{q_{in}\}, \\ \sigma_a(\{q_{in}\}, 1, a) &= \sigma_a(\{q_{in}\}, 1, b) = \{q_1\},\end{aligned}$$

and we get $(\{q_{in}\}, \{q_1\})$ as the successor of $(\{q_{in}\})$ when \mathcal{A}' reads a or b (see Figure 3).

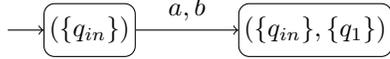


Fig. 3. Initial fragment of \mathcal{A}' .

In the next step, we calculate for symbol a :

$$\sigma(\{q_{in}\}, \{q_1\}, 2, a) = \delta(\{q_1\}, a) = \{q_2\}$$

and

$$\begin{aligned}\sigma(\{q_{in}\}, \{q_1\}, 1, a) &= \delta(\{q_{in}\}, a) \setminus \delta(\{q_1\}, a) \\ &= \{q_{in}, q_1\} \setminus \{q_2\} = \{q_{in}, q_1\}.\end{aligned}$$

As in the previous step, $\{q_{in}, q_1\}$ is partitioned into the two sets $\{q_{in}\}$ and $\{q_1\}$, and we get $(\{q_{in}\}, \{q_1\}, \{q_2\})$ as the a -successor of $(\{q_{in}\}, \{q_1\})$.

Similarly, for symbol b , we calculate

$$\sigma(\{q_{in}\}, \{q_1\}, 2, b) = \delta(\{q_1\}, b) = \{q_1\}$$

and

$$\begin{aligned}\sigma(\{q_{in}\}, \{q_1\}, 1, b) &= \delta(\{q_{in}\}, b) \setminus \delta(\{q_1\}, b) \\ &= \{q_{in}, q_1\} \setminus \{q_1\} = \{q_{in}\}.\end{aligned}$$

No sets need to be partitioned and we get $(\{q_{in}\}, \{q_1\})$ as the b -successor of $(\{q_{in}\}, \{q_1\})$. (see Figure 4).

Now we calculate for symbol a :

$$\begin{aligned}\sigma(\{q_{in}\}, \{q_1\}, \{q_2\}, 3, a) &= \delta(\{q_2\}, a) = \{q_2\}, \\ \sigma(\{q_{in}\}, \{q_1\}, \{q_2\}, 2, a) &= \delta(\{q_1\}, a) \setminus \delta(\{q_2\}, a) \\ &= \{q_2\} \setminus \{q_2\} = \emptyset,\end{aligned}$$

and

$$\begin{aligned}\sigma(\{q_{in}\}, \{q_1\}, \{q_2\}, 1, a) &= \delta(\{q_{in}\}, a) \setminus (\delta(\{q_1\}, a) \cup \delta(\{q_2\}, a)) \\ &= \{q_{in}, q_1\} \setminus \{q_2\} = \{q_{in}, q_1\}.\end{aligned}$$

As previously, $\{q_{in}, q_1\}$ is partitioned into the two sets $\{q_{in}\}$ and $\{q_1\}$, the empty set is removed, and we get $(\{q_{in}\}, \{q_1\}, \{q_2\})$ as the a -successor of $(\{q_{in}\}, \{q_1\}, \{q_2\})$.

Similarly, for symbol b , we calculate

$$\begin{aligned}\sigma(\{q_{in}\}, \{q_1\}, \{q_2\}, 3, b) &= \delta(\{q_2\}, b) = \{q_2\}, \\ \sigma(\{q_{in}\}, \{q_1\}, \{q_2\}, 2, b) &= \delta(\{q_1\}, b) \setminus \delta(\{q_2\}, b) \\ &= \{q_1\} \setminus \{q_2\} = \{q_1\}.\end{aligned}$$

and

$$\begin{aligned}\sigma(\{q_{in}\}, \{q_1\}, \{q_2\}, 1, b) &= \delta(\{q_{in}\}, b) \setminus (\delta(\{q_1\}, b) \cup \delta(\{q_2\}, b)) \\ &= \{q_{in}, q_1\} \setminus \{q_1, q_2\} = \{q_{in}\}.\end{aligned}$$

No sets need to be partitioned and we get $(\{q_{in}\}, \{q_1\}, \{q_2\})$ as the b -successor of $(\{q_{in}\}, \{q_1\}, \{q_2\})$, completing the construction (see Figure 5).

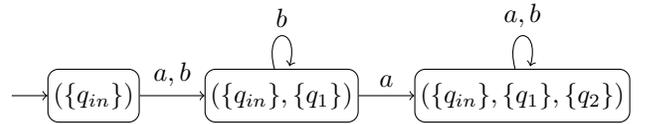


Fig. 5. Interim automaton \mathcal{A}' to Büchi automaton \mathcal{A} of Figure 1.

C. Additional Notation

As pointed out in the previous section, it is important to identify for transitions in \mathcal{A}' which component in a successor

state results from which component in the predecessor state. In addition, it will be important to identify which component will eventually have no successor component any more (the component disappears eventually, it is *discontinued*).

Let a be a symbol in Σ . Let p' and q' be two \mathcal{A}' -states such that $\delta'(p', a) = q'$. Let $1 \leq j \leq |p'|$ and $1 \leq k \leq |q'|$.

If $q'(k) \subseteq \sigma(p', j, a)$, then we write

$$p'(j) \xrightarrow{a} q'(k),$$

indicating that p' 's a -successor q' contains component k because p' contains component j . We extend this definition to finite words $w = a_0 a_1 \dots a_l \in \Sigma^*$ in the usual way:

$$q'_0(j_0) \xrightarrow{w} q'_{l+1}(j_{l+1})$$

if and only if there exist states q'_1, q'_2, \dots, q'_l and indices j_1, j_2, \dots, j_l such that for all i , $0 \leq i \leq l$, we have $q'_i(j_i) \xrightarrow{a_i} q'_{i+1}(j_{i+1})$.

If we take the example from Figure 5, we have, for instance,

$$(\{q_{in}\}, \{q_1\}, \{q_2\})(1) \xrightarrow{a} (\{q_{in}\}, \{q_1\}, \{q_2\})(2),$$

because when reading a in state $(\{q_{in}\}, \{q_1\}, \{q_2\})$, the successor state that is also $(\{q_{in}\}, \{q_1\}, \{q_2\})$ contains $\{q_1\}$ because $\{q_1\} \subseteq \sigma((\{q_{in}\}, \{q_1\}, \{q_2\}), 1, a)$. If we change however the symbol, then

$$(\{q_{in}\}, \{q_1\}, \{q_2\})(2) \xrightarrow{b} (\{q_{in}\}, \{q_1\}, \{q_2\})(2),$$

because when reading b in state $(\{q_{in}\}, \{q_1\}, \{q_2\})$, the successor state that is also $(\{q_{in}\}, \{q_1\}, \{q_2\})$ contains $\{q_1\}$ because $\{q_1\} \subseteq \sigma((\{q_{in}\}, \{q_1\}, \{q_2\}), 2, b)$.

For the remainder of this paper, we always assume that \mathcal{A}' is complete. This can be achieved, for instance, by making \mathcal{A} complete before constructing \mathcal{A}' . It guarantees that for each ω -word x , a unique (\mathcal{A}' is deterministic) run of \mathcal{A}' on x does always exist.

Let $x = x(0)x(1)x(2)\dots \in \Sigma^\omega$. Let $r' = r'(0)r'(1)r'(2)\dots$ be the run of \mathcal{A}' on x .

Let $i \geq 0$ and let $1 \leq j \leq |r'(i)|$. We write

$$r'(i)(j)_\perp$$

to indicate that either there does not exist a k such that $r'(i)(j) \xrightarrow{x(i)} r'(i+1)(k)$, or for each k such that $r'(i)(j) \xrightarrow{x(i)} r'(i+1)(k)$ we have $r'(i+1)(k)_\perp$. The notation $r'(i)(j)_\perp$ is used to indicate that in the run r' of \mathcal{A}' on x , component j of state $r'(i)$ will disappear. Either it disappears immediately (it does not have an $x(i)$ -successor) or it disappears eventually (all its $x(i)$ -successors will disappear). We will say that the component is *discontinued* in \mathcal{A}' 's run on x .

Conversely, we write

$$r'(i)(j)^\top$$

if and only if $r'(i)(j)_\perp$ does not hold (we then say that $r'(i)(j)$ is *continued* in \mathcal{A}' 's run on x). In addition, we write

$$r'(i)(j)_F^\top$$

if and only if $r'(i)(j)^\top$ and $q'(i)(j) \subseteq F$, and

$$r'(i)_F^\top$$

if and only if there exists j such that $r'(i)(j)_F^\top$.

If $r'(i)(j)^\top$, then $r'(i)(j)$ has an $x(i)$ -successor $r'(i+1)(k)$ such that $r'(i+1)(k)^\top$, because otherwise $r'(i)(j)_\perp$ would hold. Therefore, and because of the pairwise disjointness of components in \mathcal{A}' -states (Lemma 1), the number of continued components cannot decrease from one state to the next in \mathcal{A}' 's run on x :

Lemma 2. *Let k be the number of different components $r'(i)(j)$ of state $r'(i)$ such that $r'(i)(j)^\top$, and let l be the number of different components $r'(i+1)(j)$ of state $r'(i+1)$ such that $r'(i+1)(j)^\top$. Then $k \leq l$.*

As an example, let $y = bababa\dots$. The run r' of the automaton in Figure 5 on ω -word y is then:

$$(\{q_{in}\})(\{q_{in}\}, \{q_1\})(\{q_{in}\}, \{q_1\}, \{q_2\})(\{q_{in}\}, \{q_1\}, \{q_2\}) \dots$$

In this run, we now label all components of \mathcal{A}' -states either with $^\top$ or $_\perp$, depending on whether they are continued or not:

$$\begin{aligned} & (\{q_{in}\}^\top)(\{q_{in}\}^\top, \{q_1\}^\top)(\{q_{in}\}^\top, \{q_1\}_\perp, \{q_2\}^\top) \\ & (\{q_{in}\}^\top, \{q_1\}_\perp, \{q_2\}^\top)(\{q_{in}\}^\top, \{q_1\}_\perp, \{q_2\}^\top) \dots \end{aligned}$$

As we could see in the construction of the automaton in Figure 5, $(\{q_{in}\}, \{q_1\}, \{q_2\})(2)$ does not have a successor (i.e. the successor set is the empty set) when reading symbol a , because

$$\begin{aligned} \sigma((\{q_{in}\}, \{q_1\}, \{q_2\}), 2, a) &= \delta(\{q_1\}, a) \setminus \delta(\{q_2\}, a) \\ &= \{q_2\} \setminus \{q_2\} = \emptyset. \end{aligned}$$

Therefore $(\{q_{in}\}, \{q_1\}, \{q_2\})(2)$ is labelled with $_\perp$ in the run, as the run is on an ω -word that contains infinitely many times symbol a , and $(\{q_{in}\}, \{q_1\}, \{q_2\})(2)$ never persists.

The situation changes entirely, when we consider ω -word $z = aabbbb\dots$ (two a s followed by exclusively b s). Run r' of \mathcal{A}' on z is again:

$$(\{q_{in}\})(\{q_{in}\}, \{q_1\})(\{q_{in}\}, \{q_1\}, \{q_2\})(\{q_{in}\}, \{q_1\}, \{q_2\}) \dots$$

However, labelling all components of \mathcal{A}' -states either with $^\top$ or $_\perp$, depending on whether they are continued or not, leads now to:

$$\begin{aligned} & (\{q_{in}\}^\top)(\{q_{in}\}^\top, \{q_1\}^\top)(\{q_{in}\}^\top, \{q_1\}^\top, \{q_2\}^\top) \\ & (\{q_{in}\}^\top, \{q_1\}^\top, \{q_2\}^\top)(\{q_{in}\}^\top, \{q_1\}^\top, \{q_2\}^\top) \dots \end{aligned}$$

Now $(\{q_{in}\}, \{q_1\}, \{q_2\})(2)$ is always labelled with \top because the absence of symbol a in z results in $(\{q_{in}\}, \{q_1\}, \{q_2\})(2)$ always having a successor:

$$\begin{aligned}\sigma((\{q_{in}\}, \{q_1\}, \{q_2\}), 2, b) &= \delta(\{q_1\}, b) \setminus \delta(\{q_2\}, b) \\ &= \{q_1\} \setminus \{q_2\} = \{q_1\}.\end{aligned}$$

Because q_1 is an accepting \mathcal{A} -state, the run can even be labelled as:

$$\begin{aligned}(\{q_{in}\}^\top)(\{q_{in}\}^\top, \{q_1\}_F^\top)(\{q_{in}\}^\top, \{q_1\}_F^\top, \{q_2\}^\top) \\ (\{q_{in}\}^\top, \{q_1\}_F^\top, \{q_2\}^\top)(\{q_{in}\}^\top, \{q_1\}_F^\top, \{q_2\}^\top) \dots\end{aligned}$$

With our new notation, we will show in the next section that run r' of \mathcal{A}' on ω -word x will contain infinitely many states with component labels \top_F if and only if the original automaton \mathcal{A} Büchi-accepts x .

D. Some Properties of the Construction

Let $q' \in Q'$. We will write “ $\bigcup q'$ ” to designate $\bigcup_{i=1}^{|q'|} q'(i)$, i.e. the set of all \mathcal{A} -states that occur in q' . From the definition of the transition function δ' we get immediately:

Lemma 3. $\delta(\bigcup q', a) = \bigcup \delta'(q', a)$.

For the remainder of this section, let $a \in \Sigma$, let $w \in \Sigma^*$, let $x \in \Sigma^\omega$, and let r' be the run of \mathcal{A}' on x .

Lemma 4. $\delta(\{q_{in}\}, w) = \bigcup \delta'(q'_{in}, w)$.

Proof. We prove the lemma by an induction on the length $|w|$ of w . If $|w| = 0$, then \mathcal{A} reaches $\{q_{in}\} = \bigcup q'_{in}$ by reading w . If $|w| > 0$, then $w = va$ with $v \in \Sigma^*$ and $a \in \Sigma$. Assuming $\delta(\{q_{in}\}, v) = \bigcup \delta'(q'_{in}, v)$ we get:

$$\begin{aligned}\delta(\{q_{in}\}, w) &= \delta(\delta(\{q_{in}\}, v), a) = \delta(\bigcup \delta'(q'_{in}, v), a) \\ &= \bigcup \delta'(\delta'(q'_{in}, v), a) = \bigcup \delta'(q'_{in}, w),\end{aligned}$$

where the second equality holds by induction, and the third applies Lemma 3. \square

Lemma 5. *If there are infinitely many j such that $r'(j)$ contains a continued set of accepting states (written: $r'(j)^\top_F$), then \mathcal{A} accepts x .*

Proof. From Lemma 2 we know that the number of continued components cannot decrease when moving forward in a run. Because each state in \mathcal{A}' can have at most m components, where m is the number of states in \mathcal{A} , the number of continued components must remain constant after some position k in run r' . Let this number be l where $1 \leq l \leq m$. So after position k in r' , each state will contain l continued components, and therefore each continued component will have exactly

one continued component as its successor when reading the respective symbol.

So after position k , we can identify l sequences of consecutive continued components in r' . Because there are infinitely many continued components in r' that are sets of accepting \mathcal{A} -states, at least one of these l sequences of consecutive continued components must contain infinitely many continued components that are sets of accepting \mathcal{A} -states (because l is finite).

Therefore there must be sequences $i_1, i_2, i_3 \dots$ of run positions and $j_1, j_2, j_3 \dots$ of component indices such that, for all $n \geq 1$ and $w_n = x(i_n) \dots x(i_{n+1} - 1)$:

$$r'(i_n)(j_n)^\top_F$$

and

$$r'(i_n)(j_n) \xrightarrow{w_n} r'(i_{n+1})(j_{n+1}).$$

Then there exists to each \mathcal{A} -state $q_{n+1} \in r'(i_{n+1})(j_{n+1})$ an \mathcal{A} -state $q_n \in r'(i_n)(j_n)$ such that $q_{n+1} \in \delta(q_n, w_n)$. Applying Koenig's Lemma [8], there exists a sequence p_1, p_2, p_3, \dots of \mathcal{A} -states such that $p_{n+1} \in \delta(p_n, w_n)$ and $p_1 \in \delta(q_{in}, x(0) \dots x(i_1 - 1))$. Therefore there exists a run r of \mathcal{A} on x that contains the states p_n . Because all p_n are accepting \mathcal{A} -states, r is an accepting run and \mathcal{A} accepts x . \square

Lemma 6. *If \mathcal{A} accepts x , then there are infinitely many i such that $r'(i)$ contains a continued set of accepting states (written: $r'(i)^\top_F$).*

Proof. Let r be a greedy accepting run of \mathcal{A} on x and let r' be the run of \mathcal{A}' on x . We can then prove (see below) that there exists a sequence $j_0, j_1, j_2 \dots$ of component indices, $1 \leq j_0, j_1, j_2 \dots \leq m$, such that for all $i \geq 0$:

$$r(i) \in r'(i)(j_i)$$

and

$$r'(i)(j_i) \xrightarrow{x(i)} r'(i+1)(j_{i+1}).$$

Therefore all $r'(i)(j_i)$ are continued. In addition, because r is accepting, infinitely many $r(i)$ are accepting and so are infinitely many $r'(i)(j_i)$, because they contain $r(i)$ and are therefore sets of accepting \mathcal{A} -states. Then infinitely many $r'(i)(j_i)$ are continued and accepting, proving the lemma.

What we still need to prove to complete the proof of the lemma is that $r(i) \in r'(i)(j_i)$ and $r'(i)(j_i) \xrightarrow{x(i)} r'(i+1)(j_{i+1})$. We prove this fact by contradiction: we assume it is not true and prove that then r cannot be greedy.

Because of Lemma 4, there is always a component $r'(i)(j_i)$ in \mathcal{A}' -state $r'(i)$ that contains $r(i)$. If the statement is not true, then the sequence j_0, j_1, j_2, \dots of component indices such that $r(i) \in r'(i)(j_i)$ must contain a position $k \geq 0$ such that

$$r'(k)(j_k) \not\xrightarrow{x(k)} r'(k+1)(j_{k+1}).$$

Let k be the smallest such position. According to the definition of \mathcal{A}' 's transition function δ' , there must be a $j'_k > j_k$ such that

$$r'(k)(j'_k) \xrightarrow{x(k)} r'(k+1)(j_{k+1}).$$

Then there is a sequence $j'_0, j'_1, j'_2, \dots, j'_k$ of component positions such that, for all $1 \leq n \leq k-1$ such that

$$r'(n)(j'_n) \xrightarrow{x(n)} r'(n+1)(j'_{n+1}).$$

$j_0, j_1, j_2, \dots, j_k$ and $j'_0, j'_1, j'_2, \dots, j'_k$ will coincide in a few first indices.² Let l be the first position at which $j_0, j_1, j_2, \dots, j_k$ and $j'_0, j'_1, j'_2, \dots, j'_k$ start to differ. Because $j'_k > j_k$, also $j'_l > j_l$. So we have

$$r'(l-1)(j_{l-1}) \xrightarrow{x(l-1)} r'(l)(j_l),$$

$$r'(l-1)(j_{l-1}) \xrightarrow{x(l-1)} r'(l)(j'_l),$$

and

$$j'_l > j_l.$$

From the definition of \mathcal{A}' 's transition function δ' we get that $r'(l)(j_l)$ contains non-accepting \mathcal{A} -states, but $r'(l)(j'_l)$ contains accepting \mathcal{A} -states. Let $\tilde{r}(0)\tilde{r}(1)\dots\tilde{r}(k+1)$ be the run of \mathcal{A} on $x(0)x(1)\dots x(k)$ such that for all $1 \leq n \leq k+1$

$$\tilde{r}(n) \in r'(n)(j'_n)$$

and

$$\tilde{r}(k+1) = r(k+1).$$

Then $\tilde{r}(0)\tilde{r}(1)\dots\tilde{r}(k+1)$ and $r(0)r(1)\dots r(k+1)$ are runs of \mathcal{A} on $x(0)x(1)\dots x(k)$, and $\tilde{r}(0)\tilde{r}(1)\dots\tilde{r}(k+1)$ visits an accepting state earlier than $r(0)r(1)\dots r(k+1)$ does. So $r(0)r(1)\dots r(k+1)$ cannot be greedy, and consequently r cannot be greedy as $r(0)r(1)\dots r(k+1)$ is a prefix of r , contradicting the choice of r and completing the proof of the lemma. \square

Putting Lemmas 5 and 6 together, we obtain:

Corollary 1. *\mathcal{A} accepts x if and only if r' contains infinitely many \mathcal{A} -states that contain continued sets of accepting \mathcal{A} -states.*

To complete this section, let us briefly look at the size of the construction: we calculate the number of states in \mathcal{A}' . These states are n -tuples of disjoint sets of \mathcal{A} -states, where n ranges from 1 to m and m is the number of states of \mathcal{A} .

²They coincide at least in $j'_0 = j_0 = 1$, because $r'(0)$ is the initial state of \mathcal{A}' that contains a single component (containing the initial state of \mathcal{A}).

The number of ways of partitioning a set of n labelled objects into k non-empty unlabelled subsets can be written as a Stirling number of the second kind [9] :

$$\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n.$$

The number of ways to partition a set of n labelled objects into non-empty unlabelled subsets of any size is the Bell number [10]

$$B_n = \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\}.$$

Since we consider here tuples and not just partitions, we must multiply each term by the number of possible orderings, giving us the ordered Bell number [11]:

$$a(n) = \sum_{k=0}^n k! \left\{ \begin{matrix} n \\ k \end{matrix} \right\}.$$

Now, the ordered Bell number represents the number of tuples of non-empty components one can create using all n elements. In the construction presented above, all elements may not be used. We have to sum up the ordered bell numbers for each subset of Q . The number of tuples (weak-orderings) over m elements, which we write $\#_{tuples}(m)$, is thus the sum of the ordered Bell numbers $a(n)$ with n ranging from 1 to m , multiplied by the number of ways of choosing n elements over m :

$$\#_{tuples}(m) = \sum_{n=1}^m \binom{m}{n} a(n).$$

The following recurrence relation [12]:

$$\sum_{n=0}^{m-1} \binom{m}{n} a(m-n) = 2a(m) - 1$$

leads to a much simpler expression of the size of \mathcal{A}' :

$$\begin{aligned} \#_{tuples}(m) &= \sum_{n=1}^m \binom{m}{n} a(n) \\ &= \sum_{n=0}^m \binom{m}{n} a(n) - \underbrace{\binom{m}{0} a(0)}_{=1} \\ &= \sum_{n=0}^m \binom{m}{m-n} a(n) - 1 \\ &= \sum_{n=0}^m \binom{m}{n} a(m-n) - 1 \\ &= \sum_{n=0}^{m-1} \binom{m}{n} a(m-n) + \underbrace{\binom{m}{m} a(m-m)}_{=0} - 1 \\ &= 2a(m) - 1 \end{aligned}$$

The ordered Bell numbers can be approximated by [13]:

$$a(m) \approx \frac{m!}{2(\ln 2)^{m+1}}$$

and thus the size of the upper part is asymptotically [14]:

$$\#_{tuples}(m) \approx a(m) \approx (0.53m)^m.$$

IV. THE COMPLEMENTATION CONSTRUCTION

A. Additional Indices

With the use of a colouring of the tuple components, we can now extend the automaton \mathcal{A}' described in Section III, to build an automaton \mathcal{A}_c complement to \mathcal{A} . In the process, we create an *upper* (non-accepting) automaton $\hat{\mathcal{A}}$ and a *lower* (accepting) automaton $\check{\mathcal{A}}$ and put them together into automaton \mathcal{A}_c . This colouring takes the form of an index that can take integer values in $[-1, 2]$. The meaning of the colours will be presented further down.

Whereas a state q' of \mathcal{A}' is written (S_1, S_2, \dots, S_m) , a state q of $\hat{\mathcal{A}}$, $\check{\mathcal{A}}$ or \mathcal{A}_c is written as

$$q = ((S_1, c_1), (S_2, c_2), \dots, (S_m, c_m)),$$

where $S_j \in 2^Q \setminus \{\emptyset\}$ and $c_j \in [-1, 2], \forall 1 \leq j \leq m$.

B. Upper (Non-accepting) Part

The upper part $\hat{\mathcal{A}}$ represents the initial part of \mathcal{A}_c and contains no accepting states. The denomination ‘‘upper’’ comes from the fact that the authors generally picture the accepting part of the complement under the non-accepting one. The structure is inspired by the complementation construction for deterministic Büchi automata, where the non-accepting part is simply a non-accepting copy of the original automaton representing everything that can happen in a finite prefix of an ω -word. Accepting runs of \mathcal{A}_c then jump to the lower (accepting) part.

In the upper automaton $\hat{\mathcal{A}} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_{in}, \hat{F})$ we simply append colour -1 to all components of each $\hat{\mathcal{A}}$ -state. States (written (S_1, S_2, \dots, S_m)) of the interim automaton \mathcal{A}' are thus rewritten $((S_1, -1), (S_2, -1), \dots, (S_m, -1))$ in $\hat{\mathcal{A}}$. The transition function should be rewritten accordingly as follows. Let $a \in \Sigma, p' := (S_1, \dots, S_m) \in Q', q' := (S'_1, \dots, S'_{m'}) \in Q'$. If

$$\delta'(p, a) = q$$

where δ' is as in Section III, then for

$$\begin{aligned} \hat{p} &:= ((S_1, -1), \dots, (S_m, -1)) \in \hat{Q} \text{ and} \\ \hat{q} &:= ((S'_1, -1), \dots, (S'_{m'}, -1)) \in \hat{Q}, \end{aligned}$$

$\hat{\delta}$ is defined by:

$$\hat{\delta}(\hat{p}, a) := \hat{q}.$$

As for the other elements:

- $\hat{q}_{in} := ((\{q_{in}\}, -1))$
- $F := \emptyset$

Only $\hat{\mathcal{A}}$ -states will have components with colour -1 .

C. Lower (Accepting) Part

The lower automaton $\check{\mathcal{A}} := (\check{Q}, \Sigma, \check{\delta}, \check{q}_{in}, \check{F})$ can be defined in a similar fashion, but with values $0, 1, 2$ for the colouring indices and a non-empty accepting set \check{F} . The set of accepting states F_c is then defined according to the colouring of the components of the states.

In a nutshell, for a an ω -word $x \in \Sigma^\omega$ we want the run r_c of \mathcal{A}_c on x to be accepting if and only if each greedy run r of the original automaton \mathcal{A} on x either:

- eventually stops visiting states of F , or
- is discontinued (i.e. is finite or becomes non-greedy).

Translated to $\check{\mathcal{A}}$ (where any finite behaviour has already been taken care of in $\hat{\mathcal{A}}$, these constraints become:

- the run does not visit states of F , or
- it is discontinued (i.e. is finite or becomes non-greedy).

We now give some insight into the meaning of the colours. Let i' be the point where run r_c jumps to the lower part, i.e. $i \geq i' \Leftrightarrow r_c(i) \in \check{\mathcal{A}}$. The colours of the components of the states of $\check{\mathcal{A}}$ hold some information on what happens in the greedy runs of \mathcal{A} , ‘‘after’’ i' .

- Colour $c = 0$:
If a tuple component (S_j, c_j) is 0-coloured in $r_c(i)$, i.e. $(S_j, 0) \in r_c(i)$, for an $i \geq i'$, then for each state q in S_j , the greedy run r of \mathcal{A} s.t. $r(i) = q$ has not yet visited an accepting \mathcal{A} -state ‘‘since’’ i' . An $\check{\mathcal{A}}$ -state containing only 0-coloured components can be set accepting, because all greedy runs of \mathcal{A} that may have visited an accepting \mathcal{A} -state between i and i' have disappeared.
- Colour $c = 2$:
If a tuple component (S_j, c_j) is 2-coloured in $r_c(i)$, i.e. $(S_j, 2) \in r_c(i)$, for an $i \geq i'$, then for each state q in S_j , the greedy run r of \mathcal{A} s.t. $r(i) = q$ has visited an accepting \mathcal{A} -state since i' . \mathcal{A}_c -states containing 2-coloured components are kept non-accepting, because that greedy run r has visited accepting \mathcal{A} -states and has not yet disappeared. If it never disappears, then r_c only visits non-accepting states and it is correct to reject r_c in \mathcal{A}_c .
- Colour $c = 1$:
If a tuple component (S_j, c_j) is 1-coloured in $r_c(i)$, i.e. $(S_j, 1) \in r_c(i)$, for an $i \geq i'$, then for each state q in S_j , the greedy run r of \mathcal{A} s.t. $r(i) = q$ has visited an accepting \mathcal{A} -state since i' , but there also exist 2-coloured components that have not yet disappeared. We say that

1-coloured components are *on hold*, meaning that for the moment, we wait until the 2-coloured components have disappeared. When this happens, 1-coloured components become 2-coloured in the following state. \mathcal{A}_c -states that contain only 0-coloured or 1-coloured components are set accepting, because all greedy runs of \mathcal{A} that have visited an accepting \mathcal{A} -state between i and i' have disappeared. If this happens infinitely often, then no greedy run of \mathcal{A} can visit infinitely many accepting \mathcal{A} -states, and it is correct to accept r_c in \mathcal{A}_c .

We define $\check{\mathcal{A}}$ formally.

First, let us write the set of possible states:

$$\check{Q} := \bigcup_{m=1}^{|\mathcal{Q}|} \left\{ ((S_1, c_1), \dots, (S_m, c_m)) \in (2^{\mathcal{Q}} \setminus \{\emptyset\}) \times [0, 2]^m \mid \forall 1 \leq j < k \leq m, S_j \cap S_k = \emptyset \right\}.$$

The transition function $\check{\delta}$ is defined as follows:

$$\check{\delta} : \check{Q} \times \Sigma \rightarrow \check{Q}$$

As earlier, we extend the transition function δ' of Section III to define $\check{\delta}$. Let $\check{p} := ((S_1, c_1), \dots, (S_m, c_m)) \in \check{Q}$ be a state of $\check{\mathcal{A}}$.

Let $p' := (S_1, \dots, S_m)$ be the corresponding state of \mathcal{A}' obtained by removing the c_j 's. Let $a \in \Sigma$ and let $q' = (S'_1, \dots, S'_{m'})$ be the unique state of \mathcal{A}' s.t. $\delta'(p', a) = q'$.

We now define the values of the indices $c'_{j'}$ of the a -successor of \check{p} :

$$\forall 1 \leq j' \leq m', \text{ let } 1 \leq j_{pred} \leq m \text{ be (unique)} \\ \text{s.t. } \check{p}(j_{pred}) \xrightarrow{a} \check{q}(j'),$$

$$\left\{ \begin{array}{l} \text{if } \forall 1 \leq j \leq m : c_j \neq 2 \quad \text{then} \\ \text{if } \exists 1 \leq j \leq m : c_j = 2 \quad \text{then} \end{array} \right\} \left\{ \begin{array}{l} c'_j := 0 \quad \text{if } c_{j_{pred}} = 0 \\ \quad \wedge S'_{j'} \not\subseteq F \\ c'_j := 2 \quad \text{otherwise} \\ c'_j := 0 \quad \text{if } c_{j_{pred}} = 0 \\ \quad \wedge S'_{j'} \not\subseteq F \\ c'_j := 2 \quad \text{if } c_{j_{pred}} = 2 \\ c'_j := 1 \quad \text{otherwise} \end{array} \right.$$

For $\check{q} := ((S'_1, c'_1), \dots, (S'_{m'}, c'_{m'}))$ we define:

$$\check{\delta}(\check{p}, a) := \check{q}.$$

The initial state can be defined as :

$$\check{q}_{in} := ((\{q_{in}\}, 0)).$$

The set \check{F} of accepting states holds all states that do not contain a 2-coloured component :

$$\check{F} := \bigcup \{ ((S_1, c_1), \dots, (S_m, c_m)) \in \check{Q} \mid \forall 1 \leq j \leq m, c_j < 2 \}.$$

D. The Complement Automaton

Applying the construction below, we can join automata $\hat{\mathcal{A}}$ and $\check{\mathcal{A}}$ to build a new automaton \mathcal{A}_c representing a language which is complement to the language of the original automaton \mathcal{A} . The set of states Q_c is the union of the sets of states of both automata and the nondeterministic transition function δ_c is the union of the functions of the two automata plus some extra transitions which unidirectionally link the upper automaton to the lower one. As mentioned earlier, the accepting states are only located in the lower deterministic part of \mathcal{A}_c .

We create the automaton $\mathcal{A}_c = (Q_c, \Sigma, \delta_c, q_c, F_c)$ complement to \mathcal{A} by joining automata $\hat{\mathcal{A}}$ and $\check{\mathcal{A}}$ in the following way:

- $Q_c = \hat{Q} \cup \check{Q}$
- Let $p = ((S_1, c_1), \dots, (S_m, c_m)) \in Q_c$ and $a \in \Sigma$. The nondeterministic transition function

$$\delta_c : Q_c \times \Sigma \rightarrow 2^{Q_c}$$

is defined as follows ³:

$$\left\{ \begin{array}{l} \delta_c(p, a) := \hat{\delta}(p, a) \cup \check{\delta}(((S_1, 0), \dots, (S_m, 0)), a) \\ \quad \text{if } p \in \hat{\mathcal{A}}, \\ \delta_c(p, a) := \check{\delta}(p, a) \text{ otherwise} \end{array} \right.$$

- $q_c := \hat{q}_{in}$
- $F_c := \check{F}$

In order to prove the language equivalence between $L(\mathcal{A})$ and the complement of $L(\mathcal{A}_c)$, we first show the left-to-right direction, i.e. if an ω -word x is accepted by \mathcal{A} then it will not be accepted by \mathcal{A}_c , i.e. all runs of \mathcal{A}_c on x will be rejected.

Lemma 7. *Let $x = x(0)x(1)x(2)\dots \in L(\mathcal{A})$. Then: \forall run r_c of \mathcal{A}_c on x , r_c is non-accepting.*

Proof. Let r' be the unique run of \mathcal{A}' on x . Let r_c be a run of \mathcal{A}_c on x . We show that r_c cannot be accepting. Consider two cases:

- 1) $r_c \subseteq \hat{\mathcal{A}}$, i.e. the run remains in the upper part. Then r_c never visits an accepting state and is thus non-accepting.
- 2) $r_c \cap \hat{\mathcal{A}} \neq \emptyset$, i.e. $\exists l : r(l) \in \check{Q}$. Since $x \in L(\mathcal{A})$ Lemma 6 gives us: $\exists i' \geq l : r(i') \in F$, so $\exists j' : 1 \leq j' \leq |r'(i')|$ s.t. $r'(i')(j') \in F$. Thus, the definition of the transition function δ_c (stemming from $\check{\delta}$) ensures that $r_c(i')(j')$ is 1 or 2-coloured. We consider those two cases:

- a) $r_c(i')(j')$ is 2-coloured: The transition function δ_c gives us that there exists a sequence of components $(S_{i'}, c_{i'})(S_{i'+1}, c_{i'+1}) \dots \in (2^{\mathcal{Q}} \times [-1, 2])^\omega$ s.t.
 - $(S_{i'}, c_{i'}) = r_c(i')(j')$
 - $\forall i \geq i', (S_i, c_i) \in r_c(i)$
 - $\forall i \geq i', (S_i, c_i) \xrightarrow{x(i)} (S_{i+1}, c_{i+1})$

³Transitions from the upper to the lower part are as if originating from entirely 0-coloured states.

- $\forall i \geq i', c_i = 2$ (the colour cannot decrease).

Since each state in $r_c(i)$ for $i \geq i'$ contains a 2-coloured component, none of these states can be accepting and r_c is thus a non-accepting run.

- b) $r_c(i')(j')$ is 1-coloured: Again we consider the sequence of components starting at $r_c(i')(j')$: $(S_{i'}, c_{i'})(S_{i'+1}, c_{i'+1}) \cdots \in (2^Q \times [-1, 2])^\omega$ s.t.
- $(S_{i'}, c_{i'}) = r_c(i')(j')$
 - $\forall i \geq i', (S_i, c_i) \in r_c(i)$
 - $\forall i \geq i', (S_i, c_i) \xrightarrow{x(i)} (S_{i+1}, c_{i+1})$

Additionally, the definition of δ_c ensures that the colour c cannot decrease, so

$$\forall i \geq i', c_i \geq 1.$$

There are now two possibilities, either the colour eventually turns to 2, either it remains 1 forever:

- $\exists k' > i' : c_{k'} = 2$: Since $r(i')$ is continued, and the colour cannot decrease, we have $\forall i \geq k', (S_i, 2) \in r(i)$ and thus $r(i) \notin F_c$. It follows that r_c is a non-accepting run.
- $\forall k \geq i' : c_k = 1$: Here, the colour of the sequence remains 1 forever. By definition of the transition function δ_c , if a 1-coloured component exists at level k in the run, it implies that there exists a 2-coloured component at level $k - 1$. Since $c_k = 1$ is true $\forall k \geq i'$, then $\forall k \geq i' - 1$, there exists a 2-coloured component in $r_c(k)$, making all the states of r_c non-accepting for $k \geq i' - 1$.

So there exists no accepting run of x on \mathcal{A}_c . x is therefore in $\overline{L(\mathcal{A}_c)}$. \square

We now show the opposite direction of the equivalence statement, by proving that an ω -word not accepted by \mathcal{A} can produce an accepting run of \mathcal{A}_c .

Lemma 8. *Let $x = x(0)x(1)x(2) \dots \in \Sigma^\omega \setminus L(\mathcal{A})$. Then \exists run r_c of \mathcal{A}_c on x s.t. r_c is accepting.*

Proof. Let r' be the unique run of \mathcal{A}' on x (such a run exists under the assumption that \mathcal{A}' is complete). By Lemma 5, there exist only finitely many i s.t. $r'(i) \in F$. This means that $\exists i' \geq 0$ s.t. $\forall i \geq i', !r'(i) \in F$. We now construct an accepting run r_c of \mathcal{A}_c on x which jumps to the lower part $\bar{\mathcal{A}}$ exactly at that point i' where r' will have ceased holding sets that are continued and accepting.

Let $r_c = r_c(0)r_c(1)r_c(2) \cdots \in Q_c^\omega$ be a run of \mathcal{A}_c on x s.t. $r_c(i) \in \bar{Q}$ iff $i \geq i'$.

We show, by contradiction, that r_c is accepting. Suppose it is not. Then $\exists i'' \geq i'$ s.t. $\forall i \geq i'', r_c(i) \notin F_c$, i.e. r_c eventually visits only non-accepting states.

By definition of the acceptance condition F_c of \mathcal{A}_c , at least one of components of $r_c(i'')$ is 2-coloured. Formally: $\exists 1 \leq j'' \leq |r_c(i'')|$ s.t. $r_c(i'')(j'')$ is 2-coloured.

We now define the unique greedy run of \mathcal{A} containing $r_c(i'')(j'')$:

Let $(S_0, c_0)(S_1, c_1)(S_2, c_2) \cdots \in (2^Q \times [-1, 2])^\omega$ be a sequence s.t.:

- $\forall k \geq 0, (S_k, c_k) \xrightarrow{x(k)} (S_{k+1}, c_{k+1})$
- $(S_{i'}, c_{i'}) = r_c(i')(j')$ for a unique $j' : 1 \leq j' \leq |r_c(i')|$
- $(S_{i''}, c_{i''}) = r_c(i'')(j'')$.

Since $r_c(i'')(j'')$ is 2-coloured, the definition of δ_c implies that there must exist an $i''' : i' \leq i''' \leq i''$ and a $j''' : 1 \leq j''' \leq |r_c(i''')|$ s.t. $r_c(i''')(j''') \subseteq F^4$, which contradicts the fact that $\forall i \geq i', !r'(i) \in F$. Therefore our assumption that r_c is not an accepting run is contradicted and $x \in L(\mathcal{A}_c)$. \square

As a consequence of lemmas 7 and 8, we have the following:

Corollary 2. *For $x \in \Sigma^\omega$, $x \in L(\mathcal{A}) \Leftrightarrow x \in \overline{L(\mathcal{A}_c)}$.*

This proves the correctness of the construction.

In terms of size of the construction, adding the colouring to \mathcal{A}' , does not introduce more than a 3^k factor for each \mathcal{A}' -state with k components. The number of states of the lower part $\#_{lower}(m)$ is thus bounded by:

$$\#_{lower}(m) \leq \sum_{n=1}^m \binom{m}{n} \sum_{k=0}^n k! \left\{ \begin{matrix} n \\ k \end{matrix} \right\} 3^k$$

A looser (but simpler) upper bound can further be defined (using Stirling's approximation of the factorial [15]):

$$\begin{aligned} \#_{lower}(m) &\leq 3^m \cdot \sum_{n=1}^m \binom{m}{n} \sum_{k=0}^n k! \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \\ &\leq 3^m \cdot 2a(m) \\ &\approx 3^m \cdot 2 \frac{1}{2(\ln 2)^{n+1}} \cdot n! \\ &\approx 3^m \cdot \frac{1}{(\ln 2)^{m+1}} \cdot \sqrt{2\pi m} \cdot \left(\frac{m}{e}\right)^m \\ &= \frac{\sqrt{2\pi m}}{\ln 2} \cdot \left(\frac{3m}{e \ln 2}\right)^m \\ &\approx \sqrt{2\pi n} \cdot (1.59m)^m \\ &= \sqrt{2\pi} \cdot \left(\sqrt[m]{m} \cdot 1.59m\right)^m \end{aligned}$$

⁴In words, this means that if we find 2-coloured component, then at some point in the past of this sequence, an accepting state of \mathcal{A} was visited.

Since $\sqrt[m]{m} \xrightarrow{m \rightarrow \infty} 1$,

$$\#_{lower}(m) \in O((1.59m)^m)$$

So the size of the entire construction $|Q_c|$ is not more than

$$2a(m) - 1 + \sum_{n=1}^m \binom{m}{n} \sum_{k=0}^n k! \binom{n}{k} 3^k.^5$$

and thus:

$$|Q_c| \in O((1.59m)^m)$$

This upper bound is very loose and its refinement is the subject of ongoing research.

We resume our previous example and construct the complement of the automaton of Figure 1.

To avoid cumbersome notation in the following example, a component (S_j, c_j) will be denoted as:

$$\begin{array}{ll} \widehat{S}_j & \text{if } c_j = -1 & \overline{S}_j & \text{if } c_j = 1 \\ S_j & \text{if } c_j = 0 & \underline{S}_j & \text{if } c_j = 2 \end{array}$$

The upper part $\hat{\mathcal{A}}$ of the complement automaton is just the automaton \mathcal{A}' where we append colour -1 to all components (Figure 6).

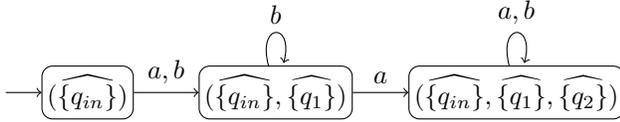


Fig. 6. Upper part $\hat{\mathcal{A}}$ of Büchi automaton \mathcal{A}_c complement to \mathcal{A} of Figure 1.

The lower part $\check{\mathcal{A}}$ is constructed in a similar way, but with the addition of colours 0, 1 and 2. Let's first look at the a -successor in $\check{\mathcal{A}}$ of $(\widehat{\{q_{in}\}})$. The two successor sets are $\{q_{in}\}$ and $\{q_1\}$. By definition of the transition function δ , since $\{q_{in}\} \cap F = \emptyset$, the colour (initially 0) remains 0. For the set $\{q_1\}$, since $\{q_1\} \subseteq F$, the new colour is 2. So the a -successor of $(\widehat{\{q_{in}\}})$ in $\check{\mathcal{A}}$ is $(\widehat{\{q_{in}\}}, \overline{\{q_1\}})$. The same holds for symbol b and we can draw the transition (Figure 7):

Let's now look at the a -successor of this newly created state. The a -successor of $\{q_1\}$ is $\{q_2\}$ and colour 2 remains. The a -successors of $\{q_{in}\}$ are $\{q_{in}\}$ (left successor) and $\{q_1\}$ (right-successor). Since $\{q_{in}\} \cap F = \emptyset$, the colour of $\{q_{in}\}$ remains 0. The colour of $\{q_1\}$ is set to 1 because $\{q_1\} \subseteq F$ and colour 2 already exists in the predecessor state. So the a -successor of $(\widehat{\{q_{in}\}}, \overline{\{q_1\}})$ is finally $(\widehat{\{q_{in}\}}, \overline{\{q_1\}}, \overline{\{q_2\}})$ and we can draw the transition in our automaton (Figure 8).

⁵Size of the upper part (which is equal to the size of \mathcal{A}') plus the size of the lower part.

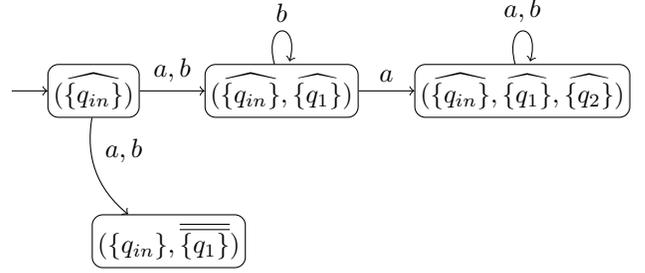


Fig. 7. Interim lower part $\check{\mathcal{A}}$ of Büchi automaton \mathcal{A}_c .

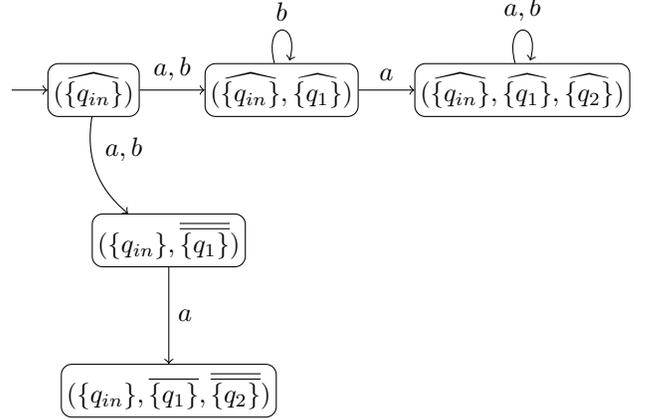


Fig. 8. Interim lower part $\check{\mathcal{A}}$ of Büchi automaton \mathcal{A}_c .

By applying this simple method for all reachable \mathcal{A}_c -states, we get the automaton of Figure 9, where the accepting states are the states of the lower part which do not contain a 2-coloured component. Here it is only the case for state $(\widehat{\{q_{in}\}}, \overline{\{q_1\}}, \overline{\{q_2\}})$.

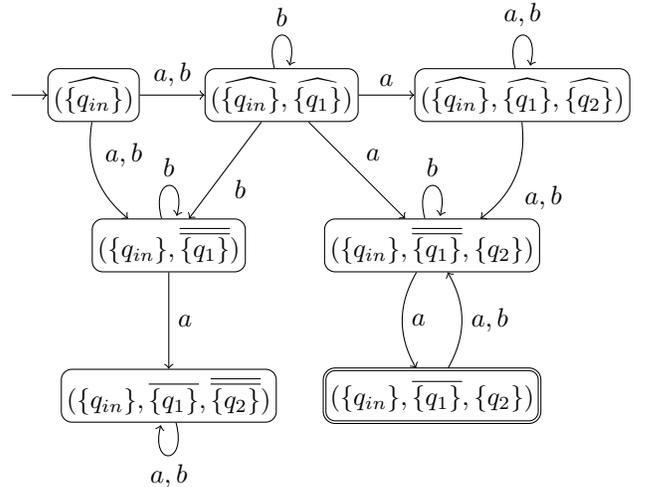


Fig. 9. Automaton \mathcal{A}_c complement to \mathcal{A} .

E. An improved algorithm

The presented construction allows some optimisations to be applied. First, it is possible to join all neighbouring 1-coloured components, as well as all neighbouring 2-coloured components, without changing the language of the constructed automaton, but greatly reducing its size (and thus its worst-case bound).

We define the $join : Q_c \rightarrow Q_c$ function as the function that takes an arbitrary state of Q_c and outputs another state of Q_c where all neighbouring 1-coloured components have been joined together, and all neighbouring 2-coloured components have been joined together. We extend the $join$ function to sets in the usual way:

$$join : 2^{Q_c} \times \Sigma \rightarrow 2^{Q_c}$$

$$join(Q, a) := \bigcup_{q \in Q} join(q)$$

Let $A_c = (Q_c, \Sigma, \delta_c, q_0, F_c)$ be a Büchi automaton as constructed by the algorithm above. Defining a new transition function

$$\delta_{join} : Q_c \times \Sigma \rightarrow Q_c$$

$$\delta_{join}(q, a) := join(\delta_c(q, a)).$$

allows us to consider the automaton

$$\mathcal{A}_{join} := (Q_c, \Sigma, \delta_{join}, q_0, F).$$

We here prove the language equivalence between \mathcal{A} and \mathcal{A}_{join} .

Lemma 9. *Let r be a run of \mathcal{A} on an ω -word x . Let r' be a sequence of states such that $join(r(i)) = r'(i)$, $\forall i \geq 0$. Then r' is a run of \mathcal{A}_{join} on x .*

Proof. We show this by induction on the length of prefixes of x . First, it is clear that $r(0) = r'(0)$ because applying $join$ to a state of the upper part has no effect. So $r'(0)$ is a run of \mathcal{A}' on ϵ .

For the induction step, by hypothesis we have that $r'(0) \dots r'(i)$ is a run of $x(0) \dots x(i-1)$. Then we get:

$$\begin{aligned} r'(i+1) &= join(r(i+1)) \\ &\in join(\delta(r(i), x(i))) \\ &= \delta_{join}(r(i), x(i)) \\ &= \delta_{join}(join(r(i)), x(i)) \\ &= \delta_{join}(r'(i), x(i)) \end{aligned}$$

So $r'(0) \dots r'(i+1)$ is indeed a run of \mathcal{A}' on x . \square

The following corollary holds trivially:

Corollary 3. *For any $i \geq 0$, $r(i)$ contains the same 1-coloured states as $r'(i)$. Also, $r(i)$ contains the same 2-coloured states as $r'(i)$.*

The ‘‘equivalence’’ of r and r' in terms of acceptance, can therefore be stated:

Lemma 10. *\mathcal{A} accepts r if and only if \mathcal{A}' accepts r' .*

Proof. We first prove that r is accepting implies that r' is accepting. Let $x \in \Sigma^\omega$. Let r be an accepting run of \mathcal{A} on x . Then it visits infinitely many accepting states of \mathcal{A} , which contain no 2-coloured component. By Corollary 3, r' also visits infinitely many states that contain no 2-coloured component. r' is thus accepting. The same argument holds in the other direction. \square

In order to reduce the state space of the lower part, one can apply a further optimisation. That is, a 2-coloured component immediately followed by a 1-coloured component can be joined into a single 2-coloured component. Intuitively, this can be done because switching a 1-coloured component to colour 2 only delays the appearance of the next accepting state (of Q_c). Since, by the structure of δ_{join} such an operation can only be done finitely many times, we only introduce finitely many delays, and thus do not turn an accepting run of \mathcal{A}_{join} into a non-accepting run. This is proven formally below:

We define a function $join_{2-1} : Q_c \rightarrow Q_c$ which applies the substitution described above recursively until there is no 1-coloured component following a 2-coloured component. $join_{2-1} : 2^{Q_c} \rightarrow 2^{Q_c}$ is extended to sets in the usual way. A new transition function can be defined:

$$\delta_{2-1} : Q_c \times \Sigma \rightarrow Q_c$$

$$\delta_{2-1}(q, a) := join_{2-1}(\delta_{join}(q, a)).$$

The corresponding automaton

$$\mathcal{A}_{2-1} := (Q = \hat{Q} \cup \check{Q} \cup \{s\}, \Sigma, \delta_{2-1}, q_0, F)$$

contains states that have no two neighbouring 1-coloured components, no two neighbouring 2-coloured components, and no 1-coloured component immediately following a 2-coloured component.

To show language equivalence, we first need to prove some general properties of the colourings in a run on the constructed automaton. Let $x \in \Sigma^\omega$ be an ω -word and $r = r(0)r(1)r(2) \dots \in Q^\omega$ be a run of \mathcal{A} on x such that $\exists k > 0$ such that $r(i) \in \check{Q} \Leftrightarrow i \geq k$. That is, l is the point where r reaches the lower part \check{Q} .

This lemma states that, in the lower part \check{Q} , the number of 0-coloured elements cannot increase from one state to the next.

Lemma 11. *Let m_i be the number of 0-coloured elements in state $r(i)$. Then for $i \geq l$, $m_i \geq m_{i+1}$.*

Proof. By definition of $\check{\delta}$ in the construction algorithm, a 0-coloured component can have at most one 0-coloured succes-

sor. Therefore the number of 0-coloured components cannot increase from a state of the run r to the next. \square

By boundedness of the number of components, it follows immediately that there are only finitely steps where the number of 0-coloured components can decrease.

Corollary 4. *In r , there are finitely many i where $m_i > m_{i+1}$.*

The following lemma shows language equivalence.

Lemma 12. $L(\mathcal{A}) = L(\mathcal{A}_{2-1})$.

Proof. We first prove that $L(\mathcal{A}_{join}) \subseteq L(\mathcal{A}_{2-1})$. Let $x \in L(\mathcal{A}_{join})$ and let r be the run of \mathcal{A}_{join} on x . Consider the run r' of x such that $r'(i) := join_{2-1}(r(i))$, $\forall i \geq 0$. One can see, by looking at the transition function δ , that the situation where a 1-coloured component immediately follows a 2-coloured component can only arise after a 0-coloured component disappears (because 2-coloured components are inherently “older” than 1-coloured components, and that newly created components (emerging from a 0-coloured component) are placed to the left of older components). In r' , these two components are merged. And the next occurrence of a 2-coloured component followed by a 1-coloured component is necessarily the result of another 0-coloured component disappearing. By Corollary 4, this can only happen finitely many times. Since r is accepting, all states that are in 1- or 2-coloured components eventually disappear. In r' , this disappearing can be delayed by the union of 1-coloured elements into 2-coloured elements, but only by a finite amount of steps. Therefore, all states present in a 1- or 2-coloured component eventually disappear also, and at this point an accepting state is marked. So r' contains infinitely many accepting states.

In the other direction of proof, suppose r' is accepting, then all states contained in 2-coloured components eventually disappear (each time an accepting state of \mathcal{A}_{2-1} is visited). The set of these states is a superset of the set of states in 2-coloured components that disappear in r . So $\forall i$ such that $r'(i) \in F$, then $r(i) \in F$. r' is an accepting run of \mathcal{A}_{2-1} immediately implies that r is an accepting run of \mathcal{A}_{join} .

Since $L(\mathcal{A}) = L(\mathcal{A}_{join})$, we have $L(\mathcal{A}) = L(\mathcal{A}_{2-1})$. \square

The following lemma states that it is sufficient to consider tuples of which the first component is either 0-coloured, or 2-coloured. In other words, if the first component of a state is of colour 1, that component can be coloured 2 and joined with the following 2-coloured component, if such a next component exists. We must define a new function f (for first component) $f : Q_c \rightarrow Q_c$ that performs this operation on a state $p := ((S_0, c_0), (S_1, c_1), \dots, (S_m, c_m))$:

- If $c_0 = 1$ and $c_1 = 0$, then
 $f(p) := ((S_0, 2), (S_1, c_1), \dots, (S_m, c_m))$

- If $c_0 = 1$ and $c_1 = 2$, then
 $f(p) := ((S_0 \cup S_1, 2), (S_2, c_2), \dots, (S_m, c_m))$
- otherwise (and if $|p| < 2$): $f(p) = p$.

$f : 2^{Q_c} \rightarrow 2^{Q_c}$ is extended to sets in the usual way.

We define the transition function:

$$\begin{aligned} \delta_f : Q_c \times \Sigma &\rightarrow Q_c \\ \delta_f(q, a) &:= f(\delta_{2-1}(q, a)). \end{aligned}$$

and the automaton:

$$\mathcal{A}_f := (Q = \hat{Q} \cup \check{Q} \cup \{s\}, \Sigma, \delta_f, q_0, F).$$

It is easy to see (using an argument as in 9 if necessary), that for a run $r = r(0) \cdot r(1) \cdot r(2) \dots$ of \mathcal{A} on an ω -word $x \in \Sigma^\omega$, the run $r' := f(r(0)) \cdot f(r(1)) \dots$ is a run of \mathcal{A}_f on x .

Lemma 13. $L(\mathcal{A}) = L(\mathcal{A}_f)$.

Proof. We first show that $L(\mathcal{A}_{2-1}) \subseteq L(\mathcal{A}_f)$. Let $x \in L(\mathcal{A}_{2-1})$ and r be an accepting run of \mathcal{A}_{2-1} on x . Let $r' = f(r(0)) \cdot f(r(1)) \dots$ be a run of \mathcal{A}_f on x . We use the same argument as in the proof of Lemma 12. A 1-coloured component on the leftmost position can only be the result of a 0-coloured component that has previously disappeared. Therefore, the switching of this 1-coloured component to colour 2 (and possible merging with the next 2-coloured component if it exists), can only happen finitely many times, and as such does not alter the acceptance of the run. So r is accepting if and only if r' is accepting.

For the other direction $L(\mathcal{A}_{2-1}) \supseteq L(\mathcal{A}_f)$, let $x \in L(\mathcal{A}_f)$. Let r be an accepting run of \mathcal{A}_{2-1} on x and $r' = f(r(0)) \cdot f(r(1)) \dots$ be a run of \mathcal{A}_f on x . $\forall i \geq 0$, if $r'(i)$ contains no 2-coloured component, then $r(i)$ either. So if r' is accepting (and contains infinitely many states that have no 2-coloured component), then r is also accepting.

As a consequence, $L(\mathcal{A}) = L(\mathcal{A}_{1-2}) = L(\mathcal{A}_f)$. \square

These optimisations reduce the set of possible states in the lower part. We first must count the number $col(k)$ of different ways of colouring a tuple of size k (which we henceforth call a k -tuple). Let $col_0(k)$, $col_1(k)$ and $col_2(k)$ be the number of colourings of a k -tuple which leftmost component is coloured 0, 1 or 2, respectively. The above optimisations (namely “join”, “2 – 1” and “f”) yield the following recurrence relations, valid for $k \geq 1$:

- $col_0(k+1) = col_0(k) + col_1(k) + col_2(k) = col(k)$
- $col_1(k+2) = col_0(k+1) + col(k)$
- $col_2(k+1) = col_0(k)$

The number of colourings can then be recursively evaluated, $\forall k \geq 1$:

$$\begin{aligned}
col(k+3) &= col_0(k+3) + col_1(k+3) + col_2(k+3) \quad (1) \\
&= col(k+2) + col_0(k+2) + col_2(k+2) + col_0(k+2) \quad (2) \\
&= col(k+2) + 2col(k+1) + col_2(k+2) \quad (3) \\
&= col(k+2) + 2col(k+1) + col_0(k+1) \quad (4) \\
&= col(k+2) + 2col(k+1) + col(k) \quad (5)
\end{aligned}$$

From this, we can deduce that, $\forall k \geq 1$:

$$col(k+2) \leq col(k+1) + col(k)$$

Reintroduced in equation 5, this gives us:

$$\begin{aligned}
col(k+3) &= col(k+2) + 2col(k+1) + col(k) \\
&= col(k+2) + col(k+1) + \underbrace{col(k+1) + col(k)}_{\geq col(k+2)} \\
&\geq col(k+2) + col(k+1) + col(k+2) \\
&= 2col(k+2) + col(k+1) \\
&\geq 2col(k+2)
\end{aligned}$$

This result can be used in equation 5 again to bound the increase of col :

$$\begin{aligned}
col(k+3) &= col(k+2) + \underbrace{2col(k+1)}_{\leq col(k+2)} + \underbrace{col(k)}_{\leq 1/2col(k+1)} \\
&\leq col(k+2) + col(k+2) + \underbrace{1/2col(k+1)}_{1/4col(k+2)} \\
&\leq 2.25 \cdot col(k+2)
\end{aligned}$$

We are left to find the initial values of col in order to have a valid bound. There are 2 colourings of tuples of size 1:

- 1) $((S_1, 0))$
- 2) $((S_1, 2))$

There are 4 colourings of tuples of size 2:

- 1) $((S_1, 0), (S_2, 0))$
- 2) $((S_1, 0), (S_2, 1))$
- 3) $((S_1, 0), (S_2, 2))$
- 4) $((S_1, 2), (S_2, 0))$

And there are 9 colourings of tuples of size 3:

- 1) $((S_1, 0), (S_2, 0), (S_3, 0))$
- 2) $((S_1, 0), (S_2, 0), (S_3, 1))$
- 3) $((S_1, 0), (S_2, 0), (S_3, 2))$
- 4) $((S_1, 0), (S_2, 1), (S_3, 0))$
- 5) $((S_1, 0), (S_2, 1), (S_3, 2))$
- 6) $((S_1, 0), (S_2, 2), (S_3, 0))$
- 7) $((S_1, 2), (S_2, 0), (S_3, 0))$

- 8) $((S_1, 2), (S_2, 0), (S_3, 1))$
- 9) $((S_1, 2), (S_2, 0), (S_3, 2))$

We sum up:

- $col(1) = 2$
- $col(2) = 4$
- $col(3) = 9$

So from these initial values, and the recurrence relation calculated above, we get:

$$col(k) \leq 2.25^k \quad \forall k \geq 0.$$

The number of states in the lower part of the optimised automaton is then:

$$\#_{lower}(m) \leq \sum_{n=1}^m \binom{m}{n} \sum_{k=0}^n k! \binom{n}{k} 2.25^k$$

and we may recalculate the approximation:

$$\#_{lower}(m) \in O\left(\left(\frac{2.25}{3} \cdot 1.59m\right)^m\right) = O\left((1.195m)^m\right)$$

F. Further possibilities for Optimization

Without further proof here, it is easy to see that there is some room for other language-preserving operations. For instance in the case where automaton \mathcal{A} is complete, we observe that \mathcal{A}_c -states of which the rightmost component has colour 2 can be ignored, as well as all their successors. This is because the completeness enforces that “branches” of the execution tree can only “die” if at some point, the deletion process described in section III removes the component. Since the deletion process is done right-to-left, this is only possible if there exists another component on the right, which is not the case here. Therefore a rightmost branch is persistent and if its colour is 2, this colour never disappears and future states can never become accepting.

Taking our previous example, since \mathcal{A} is complete, optimization directly leads to the automaton of Figure 10 (applied at construction time and not as a reduction from Figure 9).

G. Benchmark results

In this section, we give the output of our performance test against other complementation constructions. For these experiments, we have implemented the algorithm presented in Section III, with the “2-1” optimisation of Section IV-E. The algorithm was implemented in *Java* and then transformed into a plugin for the *GOAL*⁶ software. It was checked against usual complementation methods: slice, rank and Safra-Piterman,

⁶<http://goal.im.ntu.edu.tw/>

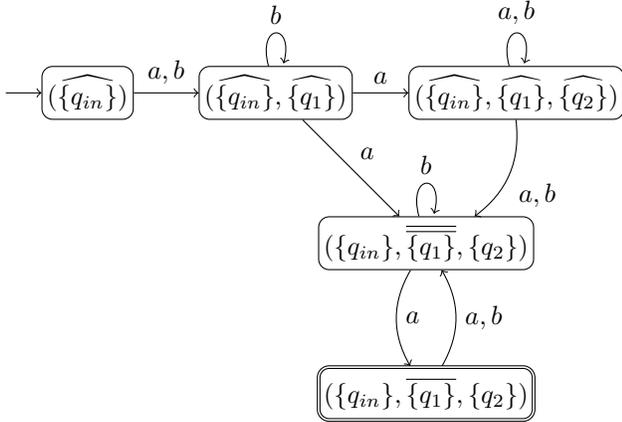


Fig. 10. Reduced automaton \mathcal{A}_c complement to \mathcal{A} .

which are all already implemented in *GOAL*. No specific optimisation (reduction of dead states, maximising of the acceptance condition, ...) was used for either of the algorithm. It is clear that applying such optimisations will often reduce the output size, as well on our construction than on the algorithms it is tested against.

These preliminary benchmarks only take into account the number of states of the constructed automaton. In order to compare the presented algorithm (that we name here “Tuple”) with each other, we wrote a *GOAL* script that generates random Büchi automata with following properties:

- Number of states: 6
- Number of transitions: 24
- Size of (propositional) alphabet: 2
- Density of accepting states: 0.3

The script applies the tuple construction on one side, and the other algorithm on the other side, and outputs the number of states and transitions for each construction. You will note that the complementation time is not considered here. The reason is that computation time depends too much on the *GOAL* framework and on the way the various algorithms are implemented.

As it is suggested in [16], the Safra-Piterman [17] algorithm produces the least states than any other construction to date. Here are the results for 1000 generated automata (we print the average number of states and transitions):

	Safra-Piterman	Tuple
av. # states	180	199
av. # transitions	721	850
# wins	555	445

The *wins* line shows how many times the algorithm has beaten its counterpart. Interestingly, the tuple construction

beats Safra-Piterman nearly half of the time. This shows that our preliminary construction shows good potential on random automata and motivates further research on efficiency.

We also deliver some test results against other algorithms implemented in *GOAL*. Because these other algorithms take a lot more time, the test is only done on 100 automata:

Slice (with turn-wise cut-point optimisation) [2] :

	Slice	Tuple
av. # states	3453	227
av. # transitions	18571	957
# wins	0	100

Rank (with tight-rank and turn-wise cut-point optimisations) [18]:

	Rank	Tuple
av. # states	1762	192
av. # transitions	33233	829
# wins	0	100

Our construction is clearly more efficient one practical cases than those two approaches.

H. An Alternative Construction

The worst-case bounds discussed above can be improved, as it was shown in [1] that the $O(0.76m)^m$ bound is tight. Therefore, there exist complementation methods that do not exceed that bound. We can improve the bound of our construction by modifying the transition function. It is possible to consider only one 2-coloured component at a time. The state-space of the lower part is then loosely bounded by:

$$\#_{lower}(m) \leq \sum_{n=1}^m \binom{m}{n} \sum_{k=0}^n k! \binom{n}{k} 1.62^k$$

A very loose upper bound for this expression is $O(0.86m)^m$, and there is strong evidence (by computation of actual values of the expression) that it can be bounded by $O(0.76m)^m$. Demonstrations of these statements will be the subject of a future paper. We may nevertheless already mention that experimental results lead to observe that this alternative construction does not behave as well as the primary (optimised) construction.

V. CONCLUSION

We have presented here a direct complementation construction for Büchi automata. The goal was not to find an efficient algorithm, such a search would have been vain because of the theoretical limits of complementation. We however strongly believe that some practical construction can stem from a better

comprehension of the internal mechanism of the complement operation on Büchi automata. Our algorithm already outperforms most complementation constructions, and still holds possibilities of improvement.

Constructed as a concatenation of two deterministic automata, the resulting complement automaton has the nice property of being deterministic in the limit. As a side note, the non-determinism degree of the complement is 2. Another interesting property is the fact that the simple complementation algorithm for deterministic Büchi automata is a special case of our construction.

Future prospects include the tightening of the complexity bounds, as well as further improvements to the algorithm's efficiency.

REFERENCES

- [1] S. Schewe, "Büchi complementation made tight," in *STACS*, ser. LIPIcs, S. Albers and J.-Y. Marion, Eds., vol. 3. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009, pp. 661–672.
- [2] S. Fogarty, O. Kupferman, T. Wilke, and M. Y. Vardi, "Unifying büchi complementation constructions," *Logical Methods in Computer Science*, vol. 9, no. 1, 2013.
- [3] D. Kähler and T. Wilke, "Complementation, disambiguation, and determinization of büchi automata unified," in *ICALP (1)*, ser. Lecture Notes in Computer Science, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfssdóttir, and I. Walukiewicz, Eds., vol. 5125. Springer, 2008, pp. 724–735.
- [4] U. Ultes-Nitsche, "A power-set construction for reducing Büchi automata to non-determinism degree two," *Information Processing Letters (IPL)*, vol. 101, no. 3, pp. 107–111, February 2007.
- [5] F. Nießner, U. Nitsche, and P. Ochsenschläger, "Deterministic ω -regular liveness properties," in *Proceedings of the 3rd International Conference on Developments in Language Theory (DLT'97)*, S. Bozapalidis, Ed., Thessaloniki, Greece, 1998, pp. 237–247.
- [6] J. R. Büchi, "On a decision method in restricted second order arithmetic," in *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science 1960*, E. Nagel et al., Eds. Stanford University Press, 1962, pp. 1–11.
- [7] W. Thomas, "Automata on infinite objects," in *Formal Models and Semantics*, ser. Handbook of Theoretical Computer Science, J. van Leeuwen, Ed., vol. B. Elsevier, 1990, pp. 133–191.
- [8] H. Hoogeboom and G. Rozenberg, "Infinitary languages: Basic theory and applications to concurrent systems," in *Current Trends in Concurrency*, ser. Lecture Notes in Computer Science, J. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds., vol. 224. Springer Verlag, 1986, pp. 266–342.
- [9] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994.
- [10] D. E. Knuth, "The art of computer programming, volume 1 (3rd ed.): fundamental algorithms," 1997.
- [11] R. Sprugnoli, "Riordan arrays and combinatorial sums," *Discrete Math.*, vol. 132, no. 1-3, pp. 267–290, Sep. 1994. [Online]. Available: [http://dx.doi.org/10.1016/0012-365X\(92\)00570-H](http://dx.doi.org/10.1016/0012-365X(92)00570-H)
- [12] O. A. Gross, "Preferential arrangements," no. 69, pp. 4–8, 1962.
- [13] J. Barthelemy, "An asymptotic equivalent for the number of total preorders on a finite set," *Discrete Mathematics*, vol. 29, no. 3, pp. 311 – 313, 1980. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/0012365X80901594>
- [14] M. Vardi, "Expected properties of set partitions," The Weizmann Institute of Science, Tech. Rep., 1980.
- [15] J. Stirling, *Methodus Differentialis, sive tractatus de summation et interpolation serierum infinitarum*, London, Ed., 1730.
- [16] M.-H. Tsai, S. Fogarty, M. Y. Vardi, and Y.-K. Tsay, "State of Büchi complementation," in *CIAA'10*, 2010, pp. 261–271.
- [17] N. Piterman, "From nondeterministic Büchi and streett automata to deterministic parity automata," in *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, Seattle, WA, 2006, pp. 255–264. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1157735.1158062>
- [18] O. Kupferman and M. Y. Vardi, "Module checking," in *CAV'96*, ser. Lecture Notes in Computer Science, R. Alur and T. A. Henzinger, Eds., vol. 1102. New Brunswick, N.J.: Springer Verlag, 1996, pp. 75–86.
- [19] C. Göttel, "Implementation of an Algorithm for Büchi Complementation," Bachelor thesis, University of Fribourg, Switzerland, 2013.
- [20] M. Michel, "Complementation is more difficult with automata on infinite words," CNET, Paris, Tech. Rep., 1988.
- [21] W. Thomas, "Handbook of formal languages, vol. 3," G. Rozenberg and A. Salomaa, Eds. New York, NY, USA: Springer-Verlag New York, Inc., 1997, ch. Languages, Automata, and Logic, pp. 389–455. [Online]. Available: <http://dl.acm.org/citation.cfm?id=267871.267878>