Counter-queue automata with an application to a meaningful extension of ω-regular languages*

David Barozzini[1], Dario Della Monica[2,3], Angelo Montanari[1], and Pietro Sala[4]

[1] Dept. of Mathematics, Computer Science, and Physics, University of Udine, Italy
dbaro13@gmail.com,angelo.montanari@uniud.it
[2] Dept. Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain
ddellamo@ucm.es
[3] Dept. of Electrical Engineering and Information Technology,
University "Federico II" of Napoli, Italy
dario.dellamonica@unina.it
[4] Dept. of Computer Science, University of Verona, Italy
pietro.sala@univr.it

Abstract. In this paper, we introduce a new class of automata over infinite words (counter-queue automata) and we prove the decidability of their emptiness problem. Then, we define an original extension of ω-regular languages, called ωT-regular languages, that captures meaningful languages that neither belong to the class of ω-regular languages nor to other extensions of it proposed in the literature, and we show that counter-queue automata are expressive enough to encode them.

1 Introduction

In this paper, we introduce a new class of automata, called counter-queue automata (CQ automata for short), and we show that their emptiness problem is decidable in 2ETIME.

CQ automata are finite state automata on ω-words, provided with a finite number of queues. Their acceptance condition imposes strong requirements on the contents and the management of queues in successful computations. In particular, infinitely many distinct numbers must be inserted in each queue and each of them must be removed and added back infinitely often.

The proof of the decidability of the emptiness problem for CQ automata benefits from known results about multi-pushdown automata. It can be decomposed into four fundamental steps. We first show that, without loss of generality, we can restrict ourselves to a proper subclass of CQ automata with a simplified transition function (we call them simple CQ automata). Then, we introduce a new two-player game (CQ game) and we show that we can associate a CQ game with each simple CQ automaton. Next, we introduce the notion of winning witness and we prove that CQ games can be decided by checking the existence of winning witnesses. Finally, we show that the existence of such a witness can be verified by checking for emptiness a suitable multi-pushdown automaton.

* D. Della Monica acknowledges the financial support from a Marie Curie INdAM-COFUND-2012 Outgoing Fellowship. A. Montanari acknowledges the support from the Italian GNCS Project *"Logics and Automata for Interval Model Checking"*.

As a matter of fact, the idea of CQ automata came to our mind during an investigation of possible extensions of regular languages of infinite words (ω-regular languages [8,9,11]). Recent work by Bojańczyk, Colcombet, and others made it clear that ω-regular languages can actually be extended in various ways, preserving their decidability and some of their closure properties [4,5,6]. As an example, ω-regular languages can be extended to allow one to constrain the distance between consecutive occurrences of a given symbol to be bounded, a promptness condition which turns out to be fairly natural in a number of application domains [1,10].

The proposed extensions of ω-regular languages pair the Kleene star $(.)^*$ with bounding/unbounding variants of it. In particular, the bounding exponent $(.)^B$ (aka B-constructor) constrains parts of the input word to be of bounded size, while the unbounding exponent $(.)^S$ (aka S-constructor) forces parts of the input word to be arbitrarily large. These two extensions have been studied both in isolation (ωB- and ωS-regular expressions) and in conjunction (ωBS-regular expressions) [6]. Among other things, it has been shown that the complement of an ωB-regular language is an ωS-regular one and vice versa; moreover, ωBS-regular languages, featuring B- and S-constructors, strictly extend both ωB- and ωS-regular languages and are not closed under complementation.

Here, we focus on those ω-languages which are complements of ωBS-regular ones, but do not belong to the class of ωBS-regular languages. Starting with a paradigmatic example of one such language given in [6], we identify a meaningful extension of ω-regular languages, that includes such a language, which is obtained by adding a new, fairly natural constructor $(.)^T$ (named T-constructor) to the standard constructors of ω-regular expressions. Then, we show that CQ automata are expressive enough to capture the resulting class of ω-languages, called ωT-regular languages, by providing an encoding of ωT-regular expressions into them.

The rest of the paper is organized as follows. In Section 2, we formally define CQ automata. Then, in Section 3 we prove that their emptiness problem is decidable in 2ETIME. Finally, in Section 4, we introduce the class of ωT-regular languages and provide their encoding into CQ automata. Conclusions give a short assessment of the work done and illustrate future research directions.

2 Counter-queue automata

In this section, we introduce CQ automata; then, in the next section we will show that their emptiness problem is decidable.

To start with, we introduce the notion of queue (of natural numbers) devoid of repetitions: a *queue* q is a finite word over $\mathbb{N}$ such that all of its elements are different. We denote the empty queue by $\varnothing$. Given a queue q, we denote by $q[i]$ the i-th number in q. Moreover, we denote the set of the elements of q and the maximum among them by $Set(q)$ and $\max(q)$, respectively. Formally, $Set(q) = \{n \in \mathbb{N} : \exists i. q[i] = n\}$ and $\max(q) = \max(Set(q))$ if $Set(q) \neq \varnothing$, $\max(q) = -1$ otherwise. The first and the last element of q can be selected by means of the usual *front* and *back* operations: $front(q) = q[1]$ and $back(q) = q[|q|]$ if $Set(q) \neq \varnothing$,

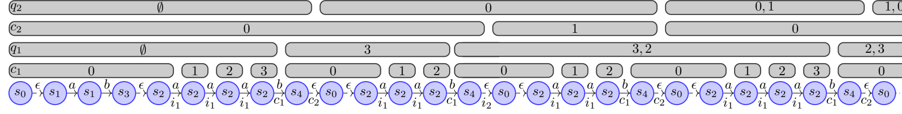

Fig. 2. A prefix of a computation of the automaton in Figure 1. A configuration is characterised by a circle (state) and the rounded-corner rectangles above it (counter-queue configuration). For each i, with $1 \leqslant i \leqslant N = 2$, c_i (resp., q_i) is one of its counter (resp., queue) components.

$front(q) = back(q) = -1$ otherwise. The *enqueue* operation has to satisfy the uniqueness constraint on the elements of q: for every $n \in \mathbb{N}$, $enqueue(q, n) = q \cdot n$ if $n \notin Set(q)$, $enqueue(q, n) = q$ otherwise. The *dequeue* operation is defined as usual: $dequeue(q) = q[2] \ldots q[|q|]$. We denote by $\mathcal{Q}$ the set of all queues.

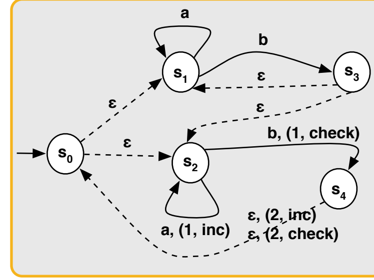

Fig. 1. An example of a CQ automaton.

A *counter-queue automaton* (CQ automaton—see Fig. 1) is a quintuple $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$, where S is a finite set of states, Σ is a finite alphabet, $s_0 \in S$ is the initial state, N is a natural number, and $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S \times (\{1, \ldots, N\} \times \{no_op, inc, check\})$ is a transition relation such that, for each $(s, \sigma, s', (k, no_op)) \in \Delta$, it holds that $k = 1$. Given a CQ automaton $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$, a *configuration* of $\mathcal{A}$ is a pair $c = (s, C)$, where $s \in S$ and $C \in (\mathbb{N} \times \mathcal{Q})^N$ is a *counter-queue* configuration. For $i \in \{1, \ldots, N\}$, we denote by $C[i] = (n_i, q_i)$ the i-th component of a counter-queue configuration C, where n_i and q_i are its *counter* and *queue* components, respectively. In the following, we will often refer to n_i as $counter(C[i])$ and to q_i as $queue(C[i])$.

Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$. We define a ternary relation $\rightarrow_{\mathcal{A}}$ over pairs of configurations and symbols in $\Sigma \cup \{\epsilon\}$ such that for all configuration pairs $(s, C), (s', C')$ and $\sigma \in \Sigma \cup \{\epsilon\}$, $(s, C) \rightarrow^{\sigma}_{\mathcal{A}} (s', C')$ if, and only if, there exists $\delta = (s, \sigma, s', (k, op)) \in \Delta$ such that $C[k'] = C'[k']$ for all $k' \neq k$, and

- if $op = no_op$, then $C[k] = C'[k]$;
- if $op = inc$, then $counter(C'[k]) = counter(C[k]) + 1$ and $queue(C'[k]) = queue(C[k])$;
- if $op = check$, then $counter(C'[k]) = 0$; moreover,
  - if $counter(C[k]) = front(queue(C[k]))$, then $queue(C'[k]) = enqueue(dequeue(queue(C[k])), counter(C[k]))$;
  - if $counter(C[k]) \neq front(queue(C[k]))$, then $queue(C'[k]) = enqueue(queue(C[k]), counter(C[k]))$.

In such a case, we say that $(s, C) \rightarrow^{\sigma}_{\mathcal{A}} (s', C')$ via δ. Let $\rightarrow^{*}_{\mathcal{A}}$ be the reflexive and transitive closure of $\rightarrow^{\sigma}_{\mathcal{A}}$ (where we abstract away symbols in $\Sigma \cup \{\epsilon\}$). The *initial configuration* of $\mathcal{A}$ is the pair (s_0, C_0), where for each $k \in \{1, \ldots, N\}$ we have $C_0[k] = (0, \varnothing)$. A *computation* of $\mathcal{A}$ is an infinite sequence of configurations

$\mathcal{C} = (s_0, C_0)(s_1, C_1)\ldots$, where, for all $i \in \mathbb{N}$, $(s_i, C_i) \rightarrow_{\mathcal{A}}^{\sigma_i} (s_{i+1}, C_{i+1})$ for some $\sigma_i \in \Sigma \cup \{\epsilon\}$ (see Figure 2). Given two configurations (s_i, C_i) and (s_j, C_j) in $\mathcal{C}$, with $i \leqslant j$, we say that (s_j, C_j) is *ϵ-reachable* from (s_i, C_i), written $(s_i, C_i) \rightarrow_{\mathcal{A}}^{*\epsilon} (s_j, C_j)$, if $(s_{j'-1}, C_{j'-1}) \rightarrow_{\mathcal{A}}^{\epsilon} (s_{j'}, C_{j'})$ for all $j' \in \{i+1, \ldots, j\}$. Given a computation $\mathcal{C}$ of $\mathcal{A}$ and an ω-word $w \in \Sigma^\omega$, we say that w is *$\mathcal{C}$-induced* if there is an increasing function $f : \mathbb{N} \backslash \{0\} \rightarrow \mathbb{N}$ such that:

- $(s_0, C_0) \rightarrow_{\mathcal{A}}^{*\epsilon} (s_{f(1)}, C_{f(1)})$, and
- for all $i \geqslant 1$, $(s_{f(i)}, C_{f(i)}) \rightarrow_{\mathcal{A}}^{w[i]} (s_{f(i)+1}, C_{f(i)+1}) \rightarrow_{\mathcal{A}}^{*\epsilon} (s_{f(i+1)}, C_{f(i+1)})$.

A computation $\mathcal{C}$ of $\mathcal{A}$ is *accepting* if and only if:

(**ac1**) there exists a $\mathcal{C}$-induced ω-word w;
(**ac2**) for all $k \in \{1, \ldots, N\}$, $\lim_{i \rightarrow +\infty} |queue(C_i[k])| = +\infty$;
(**ac3**) for all $k \in \{1, \ldots, N\}$, $i \in \mathbb{N}$, and $n \in Set(queue(C_i[k]))$, it holds that $|\{i' \in \mathbb{N} \mid back(queue(C_{i'}[k])) = n\}| = +\infty$.

In such a case, we say that w is *accepted* by $\mathcal{A}$. Notice that condition (**ac2**) forces the insertion of infinitely many (distinct) numbers in each queue and condition (**ac3**), together with condition (**ac2**), guarantees that each of them occurs, that is, is removed and added back, infinitely often.

We denote by $\mathcal{L}(\mathcal{A})$ the set of all ω-words $w \in \Sigma^\omega$ that are accepted by $\mathcal{A}$, and we say that $\mathcal{A}$ *accepts* the language $\mathcal{L}(\mathcal{A})$.

3 Decidability of the emptiness problem for CQ automata

We now prove that the emptiness problem for *CQ* automata is decidable in 2ETIME. The proof consists of four main steps: (i) we replace general *CQ* automata by simple ones; (ii) we associate a two-player game (*CQ* game) with each simple *CQ* automaton; (iii) we prove that *CQ* games can be decided by checking the existence of winning witnesses; (iv) we show that the latter condition can be verified by checking for emptiness a suitable multi-pushdown automaton.

From CQ automata to simple CQ automata. W.l.o.g., from now on we restrict our attention to simple *CQ* automata. A *CQ* automaton $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ is *simple* if, and only if, for each $s \in S$ either $|\{(s, \sigma, s', (k, op)) \in \Delta\}| = 1$ or $op = no_op$, $k = 1$, and $\sigma = \epsilon$ for all $(s, \sigma, s', (k, op)) \in \Delta$. Basically, a simple *CQ* automaton has two kinds of state: those in which it can fire exactly one action and those in which it makes a nondeterministic choice. Moreover, for all pairs of configurations $(s, C), (s', C')$ such that $(s, C) \rightarrow_{\mathcal{A}}^{\sigma} (s', C')$, the transition $\delta \in \Delta$ that has been fired in (s, C) is uniquely determined by s and s'. By exploiting *ϵ-transitions*, that is, transitions of the form $(s, \epsilon, s', (k, op))$, and by adding a suitable number of states, it can be easily shown that every *CQ* automaton $\mathcal{A}$ may be turned into a simple one $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

The set of states of a simple *CQ* automaton can be partitioned in four subsets: (*i*) the set of states s from which only one transition of the form $(s, \sigma, s', (k, check))$ can be fired ($check_k$ states); (*ii*) the set of states s from which only one transition of the form $(s, \sigma, s', (k, inc))$ can be fired (inc_k states); (*iii*) the set of states s from which only one transition of the form $(s, \sigma, s', (1, no_op))$, with $\sigma \neq \epsilon$, can be

fired (*sym* states); (*iv*) the set of states s from which possibly many transitions of the form $(s, \epsilon, s', (1, no_op))$ can be fired (*choice* states).

Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ be a (simple) CQ automaton. A *partial computation* of $\mathcal{A}$ is a *finite sequence* $\mathcal{P} = (s_0, C_0) \ldots (s_n, C_n)$ such that, for all $i \in \{0, \ldots, n-1\}$, $(s_i, C_i) \rightarrow_{\mathcal{A}}^{\sigma_i} (s_{i+1}, C_{i+1})$, for some $\sigma_i \in \Sigma \cup \{\epsilon\}$. If (s_0, C_0) is the initial configuration of $\mathcal{A}$, then $\mathcal{P}$ is a *prefix computation* of $\mathcal{A}$. We denote by $Partial_{\mathcal{A}}$ and $Prefixes_{\mathcal{A}}$ the sets of all partial computations and prefix computations of $\mathcal{A}$, respectively. Clearly, $Prefixes_{\mathcal{A}} \subseteq Partial_{\mathcal{A}}$. Given a prefix computation $\mathcal{P} = (s_0, C_0) \ldots (s_n, C_n)$ and a partial computation $\mathcal{P}'$ $= (s'_0, C'_0) \ldots (s'_m, C'_m)$, we say that $\mathcal{P}$ *can be extended with* $\mathcal{P}'$ if, and only if, $\mathcal{P}'' = \mathcal{P} \cdot \mathcal{P}' = (s_0, C_0) \ldots (s_n, C_n)(s'_0, C'_0) \ldots (s'_m, C'_m)$ is a prefix computation ($\mathcal{P}''$ is said to be an *extension* of $\mathcal{P}$).

Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ and $\mathcal{P} = (s_0, C_0) \ldots (s_n, C_n) \in Partial_{\mathcal{A}}$. For all $s \in S$, it holds that if $(s_n, C_n) \rightarrow_{\mathcal{A}}^{\sigma} (s, C)$, for some counter-queue configuration C and some $\sigma \in \Sigma \cup \{\epsilon\}$, then C is uniquely determined by s_n, s, and C_n, that is, there is no $C' \neq C$ such that $(s_n, C_n) \rightarrow_{\mathcal{A}}^{\sigma'} (s, C')$, for any σ'.

From simple CQ automata to CQ games. We now associate a two-player game, called *CQ game*, with every CQ automaton $\mathcal{A}$. The configurations of the game are the prefix computations of $\mathcal{A}$. The initial configuration is the shortest (non-empty) prefix computation of $\mathcal{A}$, namely, $\mathcal{P}_0 = (s_0, C_0)$. Let $i \in \mathbb{N}$ be the current round and $\mathcal{P}_i = (s_0, C_0) \ldots (s_n, C_n)$ be the current game configuration. The first player (*Spoiler*) moves by choosing a priority $p_i \in \{check_k, max_k | 1 \leqslant k \leqslant N\} \cup \{sym\}$; the second one (*Duplicator*) replies with a partial computation $\mathcal{Q}_i = (s'_0, C'_0) \ldots (s'_m, C'_m)$ such that (i) $\mathcal{P}_i$ can be extended with $\mathcal{Q}_i$; (ii) if $p_i = check_k$, for some $k \in \{1, \ldots, N\}$, then there is $j \in \{0, \ldots, m-1\}$ such that $front(queue(C'_j[k])) = back(queue(C'_{j+1}[k]))$; (iii) if $p_i = max_k$, for some $k \in \{1, \ldots, N\}$, then there is $j \in \{0, \ldots, m-1\}$ such that $back(queue(C'_{j+1}[k])) > \max(queue(C'_j[k]))$; (iv) if $p_i = sym$, then there is $j \in \{0, \ldots, m\}$ such that a non-ϵ-transition can be fired from s'_j.

A play $\mathcal{P}\ell$ of a CQ game is a sequence of pairs $(\mathcal{P}_0, p_0)(\mathcal{P}_1, p_1) \ldots$, where, at each round $i \in \mathbb{N}$, p_i is Spoiler's move and $\mathcal{P}_{i+1}$ is the result of the extension of $\mathcal{P}_i$ with Duplicator's move $\mathcal{Q}_i$. Given a play $\mathcal{P}\ell$, we denote by $\mathcal{P}\ell(n)$ the finite prefix of $\mathcal{P}\ell$ of length n, and by $\mathcal{P}\ell[n]$ the n-th element of $\mathcal{P}\ell$. Let $\mathcal{P}lay_{\mathcal{A}}$ be the set of all possible finite prefixes of all possible plays of the CQ game on $\mathcal{A}$. Duplicator wins a play of the CQ game if, and only if, the play is infinite, that is, she is able to reply to Spoiler's move at every round. A *strategy* for Duplicator in the CQ game on $\mathcal{A}$ is a function $str : \mathcal{P}lay_{\mathcal{A}} \rightarrow Partial_{\mathcal{A}}$. In a play $\mathcal{P}\ell = (\mathcal{P}_0, p_0)(\mathcal{P}_1, p_1) \ldots$, Duplicator acts according to str if for all $i \in \mathbb{N}$, $\mathcal{P}_{i+1} = \mathcal{P}_i \cdot str(\mathcal{P}\ell(i))$, that is, $\mathcal{P}_{i+1}$ is the result of the extension of $\mathcal{P}_i$ with $str(\mathcal{P}\ell(i))$. A strategy str for Duplicator is winning if, and only if, Duplicator wins every play in which she acts according to str. The proof of Lemma 1 below is straightforward, and thus omitted.

Lemma 1. *Let $\mathcal{A}$ be a CQ automaton. It holds that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ if and only if there is a winning strategy for Duplicator in the CQ game on $\mathcal{A}$.*

From CQ games to winning witnesses. We show now how to decide CQ games making use of the notion of winning witness.

Definition 1 (winning witness). *Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ be a CQ automaton and $\mathcal{P} = (s_0, C_0) \dots (s_n, C_n) \in Prefixes_{\mathcal{A}}$. $\mathcal{P}$ is a* winning witness *if, and only if, there exist $2N + 3$ indexes $begin < b_1 < e_1 < \dots < b_N < e_N < limit < end$ such that $0 \leqslant begin$, $end \leqslant n$, and the following conditions hold:*
1. *s_{begin} is a state from which a non-ϵ-transition can be fired;*
2. *$s_{begin} = s_{end}$ and, for each $k \in \{1, \dots, N\}$, s_{b_k} is an inc_k state, $s_{b_k} = s_{e_k}$, and s_j is not a $check_k$ state for any j with $b_k \leqslant j \leqslant e_k$;*
3. *for each $k \in \{1, \dots, N\}$, there is j, with $e_N < j < limit$, such that s_j is a $check_k$ state;*
4. *let $J = \{j \mid 0 \leqslant j \leqslant limit$ and s_j is a $check_k$ state for some $k\}$; there exists a set of indexes $\overline{J} = \{\overline{b}_j, \overline{e}_j \mid j \in J\}$ such that for all $j \in J$, with s_j a $check_k$ state, (i) $limit < \overline{b}_j < \overline{e}_j < end$, (ii) for all $j' \neq j$, either $\overline{e}_j < \overline{b}_{j'}$ or $\overline{e}_{j'} < \overline{b}_j$, (iii) $s_{\overline{b}_j}$ and $s_{\overline{e}_j}$ are $check_k$ states and there are no other $check_k$ states between them, and (iv) there are exactly $counter(C_j[k])$ inc_k states between $s_{\overline{b}_j}$ and $s_{\overline{e}_j}$.*

Indexes $begin, b_1, e_1, \dots, b_N, e_N, limit$, and end are referred to as the *milestones* of a winning witness. A winning witness can be seen as a finite representation of a winning strategy, as stated by the following lemma, which links the existence of a winning strategy for Duplicator in the CQ game on $\mathcal{A}$ to the existence of a winning witness (the proof can be found in [3]).

Lemma 2. *Let $\mathcal{A}$ be a CQ automaton. Then, Duplicator has a winning strategy in the CQ game on $\mathcal{A}$ if and only if $Prefixes_{\mathcal{A}}$ contains a winning witness.*

From winning witnesses to multi-pushdown automata. Thanks to Lemma 1 and Lemma 2, deciding the emptiness problem for a CQ automaton $\mathcal{A}$ amounts to searching $Prefixes_{\mathcal{A}}$ for a winning witness. Since we restricted ourselves to simple CQ automata, we can safely identify elements of $Prefixes_{\mathcal{A}}$ with their sequence of states and thus, by slightly abusing the notation, we write, for instance, $s_0 s_1 \dots s_n \in Prefixes_{\mathcal{A}}$ for $(s_0, C_0) \dots (s_n, C_n) \in Prefixes_{\mathcal{A}}$. Given a CQ automaton $\mathcal{A}$, let $\mathcal{L}_{\mathsf{ww}}(\mathcal{A})$ be the language of finite words over the alphabet S (the set of states of $\mathcal{A}$) that are winning witnesses in $Prefixes_{\mathcal{A}}$. It is easy to see that $\mathcal{L}(\mathcal{A}) \neq \varnothing$ if and only if $\mathcal{L}_{\mathsf{ww}}(\mathcal{A}) \neq \varnothing$. In what follows, for a CQ automaton $\mathcal{A}$ we build a multi-pushdown automaton whose language is exactly $\mathcal{L}_{\mathsf{ww}}(\mathcal{A})$. Since the emptiness problem for multi-pushdown automata is decidable, so is the one for CQ automata.

Multi-pushdown automata generalize pushdown ones by featuring more than one stack [7]. At each transition, the automaton can write on any stack, possibly more than one, but it reads only from the first non-empty one. Formally, a *multi-pushdown automaton* (*MPDA* for short) is a tuple $\mathcal{M} = \langle n, Q, \Sigma, \Gamma, \delta, q_0, F, Z_0 \rangle$, where $n \geqslant 1$ is the number of stacks, Q is a finite set of states, Σ and Γ are finite alphabets, referred to as the input and the stack alphabet, respectively,

$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times (\Gamma^*)^n$ is the transition relation, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $Z_0 \in \Gamma$ is the initial stack symbol.

Let $\mathcal{M} = \langle n, Q, \Sigma, \Gamma, \delta, q_0, F, Z_0 \rangle$ be an MPDA. A *configuration* of $\mathcal{M}$ is a $(n+2)$-tuple $\langle q, w, \gamma_1, \ldots, \gamma_n \rangle$, where $q \in Q$, $w \in \Sigma^*$, and $\gamma_i \in \Gamma^*$, for $i \in \{1, \ldots, n\}$. We define a binary relation $\vdash_{\mathcal{M}}$ over pairs of configurations as follows: $\langle q, aw, \varepsilon, \ldots, \varepsilon, A\gamma_i, \ldots, \gamma_n \rangle \vdash_{\mathcal{M}} \langle q', w, \alpha_1, \ldots, \alpha_{i-1}, \alpha_i\gamma_i, \ldots, \alpha_n\gamma_n \rangle$ if and only if $(q, a, A, q', (\alpha_1, \ldots, \alpha_n)) \in \delta$. Intuitively, the automaton pops the first symbol from the first non-empty stack, reads the first letter of the (current) input word, moves from state q to state q', and pushes strings $\alpha_1, \ldots, \alpha_n$ in the stacks. We denote by $\vdash^*_{\mathcal{M}}$ the transitive closure of $\vdash_{\mathcal{M}}$. A word $w \in \Sigma^*$ is *accepted* by $\mathcal{M}$ if, and only if, $\langle q_0, w, \varepsilon, \ldots, \varepsilon, Z_0 \rangle \vdash^*_{\mathcal{M}} \langle q, \varepsilon, \gamma_1, \ldots, \gamma_n \rangle$ for some $q \in F$. The language of $\mathcal{M}$, denoted by $\mathcal{L}(\mathcal{M})$, is the set of words accepted by $\mathcal{M}$.

From now on, we denote by σ the symbol (in Σ) read from the input word w and by γ the symbol (in Γ) read from the stack. Moreover, unless we explicitly say the opposite, we assume that the symbol popped from the stack is immediately pushed back in. In particular, by saying "do nothing" we mean "push γ back in the same stack you read it from and do nothing else" (notice that this is possible because we will make use of an MPDA whose stacks store, to a large extent, disjoint subsets of symbols in Γ).

Theorem 1 ([2,7]). *The emptiness problem for MPDA is 2ETIME-complete.*

Lemma 3 ([7]). *Let $\mathcal{L} = \mathcal{L}(\mathcal{M})$ for some MPDA $\mathcal{M}$ and $\mathcal{L}'$ be a regular language. Then, there exists an MPDA $\mathcal{M}'$ such that $\mathcal{L}(\mathcal{M}') = \mathcal{L} \cap \mathcal{L}'$.*

W.l.o.g., in what follows we restrict our attention to winning witnesses for which the sets of indexes required by items 3 and 4 of Definition 1 are ordered. More precisely (we borrow the notation from Definition 1), we assume that (i) there are N indexes $c_1 < \ldots < c_N$, with $e_N < c_1$ and $c_N < limit$, such that s_{c_k} is a $check_k$ state for each $k \in \{1, \ldots, N\}$ (this requirement strengthens the one imposed by item 3 of Definition 1), and (ii) for all $j', j'' \in J$, with $j' < j''$, such that $s_{\overline{b}_{j'}}$ and $s_{\overline{b}_{j''}}$ are, respectively, $check_{k'}$ and $check_{k''}$ states, it holds that $\overline{b}_{j'} < \overline{b}_{j''}$ if and only if $k' < k''$ (this requirement strengthens the one imposed by item 4 of Definition 1). Notice that if $k' = k''$, then $\overline{b}_{j''} < \overline{b}_{j'}$. It is easy to check that, given a CQ automaton $\mathcal{A}$, $Prefixes_{\mathcal{A}}$ contains a winning witness, as specified by Definition 1, if and only if it contains a winning witness satisfying

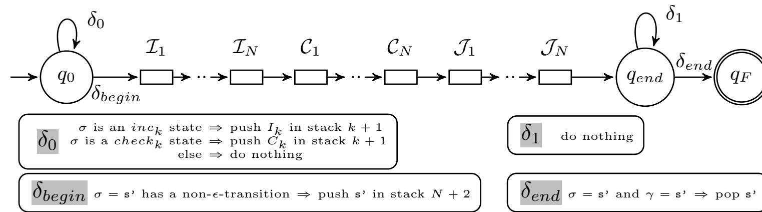

Fig. 3. A graphical account of $\hat{\mathcal{M}}$ (part 1).

the additional ordering properties above. Thus, Lemma 2 holds with the new definition of winning witness as well.

Given a CQ automaton $\mathcal{A}$, we build an MPDA $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}_{\mathsf{ww}}(\mathcal{A})$ as follows. We first build an MPDA $\hat{\mathcal{M}}$, whose input alphabet is the set of states of $\mathcal{A}$, which accepts winning witnesses. Unfortunately, such an automaton might also accept winning witnesses not belonging to $Prefixes_{\mathcal{A}}$. However, since $Prefixes_{\mathcal{A}}$ is a regular language, thanks to Lemma 3, there exists an MPDA $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\hat{\mathcal{M}}) \cap Prefixes_{\mathcal{A}} = \mathcal{L}_{\mathsf{ww}}(\mathcal{A})$.

Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ be a CQ automaton. $\hat{\mathcal{M}} = \langle n, Q, \Sigma_{\hat{\mathcal{M}}}, \Gamma, \delta, q_0, F, Z_0 \rangle$ is defined as follows. We set $n = N+2$, $\Sigma_{\hat{\mathcal{M}}} = S$, and $\Gamma = S \cup \{I_k, C_k\}_{k=1}^{N} \cup \{Z_0\}$. The remaining components of $\hat{\mathcal{M}}$ are described in Figures 3 and 4. In particular, the transition relation δ forces the automaton to behave as follows (steps 1-5):

1. it nondeterministically guesses *begin* and it stores s_{begin} in the last stack in order to check, at a later stage, that $s_{begin} = s_{end}$;
2. for each $k \in \{1, \dots, N\}$, it nondeterministically guesses b_k and e_k and it stores s_{b_k} in the first stack in order to check that $s_{b_k} = s_{e_k}$;
3. it checks for the existence, after e_N, of $check_k$ states, for $k = 1, \dots, N$, in the desired order;
4. until *limit* is reached, whenever it reads an inc_k (resp., $check_k$) state for some $k \in \{1, \dots, N\}$, it pushes I_k (resp., C_k) in the $(k+1)$-th stack;
5. once *limit* is reached, it checks whether the stacks can be emptied, by popping I_k, when an inc_k state is read, and C_k, when a $check_k$ state is read.

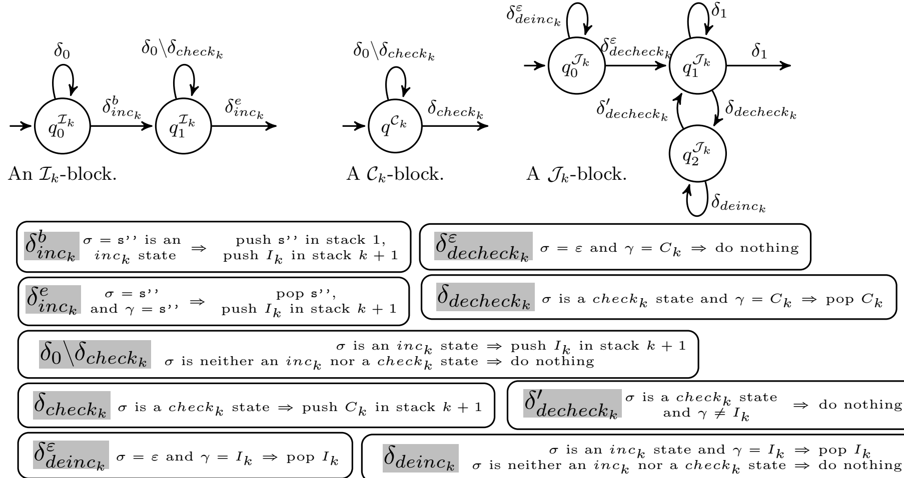

Fig. 4. A graphical account of $\hat{\mathcal{M}}$ (part 2): $\mathcal{I}_k$-, $\mathcal{C}_k$-, and $\mathcal{J}_k$-blocks, for $k \in \{1, \dots, N\}$.

Figure 3 gives a high-level pictorial account of the behavior of the MPDA. Steps 1 and 4 above are easy to implement; steps 2, 3, and 5 are dealt with

by separate modules, namely, $\mathcal{I}$-, $\mathcal{C}$-, and $\mathcal{J}$-blocks, respectively (Figure 4). Transitions of $\hat{\mathcal{M}}$ are depicted by labeled edges, whose labels have the form $\delta_{\mathtt{x}}$, which denote sets of transitions. An explicit account of their intended meaning is given in the pictures. Transitions labeled with δ_{begin} and δ_{end} (see Figure 3) force the automaton to behave exactly as described in step 1 above, while δ_0 in both Figure 3 and Figure 4 captures the behavior described in step 4. Let us consider, for instance, the loop edge labeled with δ_0 in Figure 3. It denotes the following transitions:

- for each inc_k state σ of $\mathcal{A}$, $(q_0, \sigma, \lambda_i, q_0, (\epsilon, \ldots, \epsilon, \lambda_i, \epsilon, \ldots, \epsilon, I_k, \epsilon, \ldots, \epsilon)) \in \delta$, for $i \in \{1, \ldots, N\}$ and $\lambda_i \in \{I_i, C_i\}$ (we are assuming that $i < k$; similar transitions exist for $i > k$ and $i = k$);
- for each $check_k$ state σ of $\mathcal{A}$, $(q_0, \sigma, \lambda_i, q_0, (\epsilon, \ldots, \epsilon, C_k\lambda_i, \epsilon, \ldots, \epsilon)) \in \delta$, for $i \in \{1, \ldots, N\}$ and $\lambda_i \in \{I_i, C_i\}$ (for the sake of completeness, we are assuming here, unlike the previous item, that $i = k$; similar transitions exist for $i < k$ and $i > k$);
- for each state σ of $\mathcal{A}$ that is neither an inc_k nor a $check_k$ state, $(q_0, \sigma, \lambda_i, q_0, (\epsilon, \ldots, \epsilon, \lambda_i, \epsilon, \ldots, \epsilon)) \in \delta$, for $i \in \{1, \ldots, N\}$ and $\lambda_i \in \{I_i, C_i\}$.

The internal structure of $\mathcal{I}$-, $\mathcal{C}$-, and $\mathcal{J}$-blocks is depicted in Figure 4. An $\mathcal{I}_k$*-block*, with $k \in \{1, \ldots, N\}$, is used to check whether there is a cycle that starts and ends at an inc_k state, and visits no $check_k$ states. A $\mathcal{C}_k$*-block*, with $k \in \{1, \ldots, N\}$, is used to verify whether there is a $check_k$ state before *limit*. Finally, a $\mathcal{J}_k$*-block*, with $k \in \{1, \ldots, N\}$, is used to check item 4 of Definition 1: first, it pops the increments that were not checked yet from stack $k+1$, and then it nondeterministically checks whether the condition can be satisfied by trying to empty stack $k+1$.

Theorem 2. *The emptiness problem for CQ automata is decidable in 2ETIME.*

4 Encoding of ωT-regular languages into CQ automata

In this section, we first introduce ωT-regular languages and then we show how to encode them into CQ automata.

ωT-regular languages. A standard way to define ω-regular languages is via ω-regular expressions. An *ω-word* can be seen as the concatenation of a finite prefix, belonging to a regular language, and an infinite sequence of finite words (we call each of them an *ω-iteration*), also belonging to a regular language. We focus on ω-iterations consisting of finite sequences of symbols generated by an occurrence of the Kleene star operator $(.)^*$, aka *$*$-constructor*, in the scope of the *ω-constructor* $(.)^\omega$. As an example, the ω-regular expression $(b \cdot a^*)^\omega$ generates the language of ω-words featuring an infinite sequence of finite words consisting of one b followed by a finite (possibly empty) sequence of a's. Given an ω-regular expression E featuring an occurrence of the $*$-constructor (sub-expression R^*) in the scope of the ω-constructor and an ω-word w belonging to the language of E, we refer to the sequence of the sizes of the (maximal) blocks of consecutive iterations of R in the different ω-iterations as the *(sequence of) exponents of R in (the ω-iterations*

of) w. As an example, the ω-word $w = bbabaabaaa \ldots (= ba^0ba^1ba^2ba^3 \ldots)$, generated by $(b \cdot a^*)^\omega$, is characterized by the sequence of exponents $0, 1, 2, 3, \ldots$. A limitation of ω-regular languages is that there is no way to constrain the behaviour (in the limit) of such a sequence of exponents.

Some extensions of ω-regular expressions have been proposed in the last decade to overcome this limitation (see, e.g., [6]). ωB-regular expressions are obtained from ω-regular ones by adding a variant of $(.)^*$, called *B-constructor* and denoted by $(.)^B$, to be used in the scope of $(.)^\omega$. The B-constructor constrains the argument R of the expression R^B to be repeated in each ω-iteration a number of times less than a given bound fixed for the whole ω-word (boundedness). As the bound may vary from word to word, the language is not ω-regular. Similarly, ωS-regular expressions are obtained from ω-regular ones by adding another variant of $(.)^*$, called *S-constructor* and denoted by $(.)^S$, to be used in the scope of $(.)^\omega$. For every ωS-regular expression containing the sub-expression R^S and every $k > 0$, the S-constructor constrains the number of ω-iterations in which the argument R is repeated exactly k times to be finite. It follows that the sequence of exponents of R in ω-words that feature an infinite number of ω-iterations including a sequence of consecutive R's generated by R^S tends towards infinity (strong unboundedness). ωBS-regular expressions are built by using the operators of ω-regular expressions and both $(.)^B$ and $(.)^S$. The class of ωBS-regular languages strictly includes the classes of ωB- and ωS-regular ones, as witnessed by the ωBS-regular language $L = (a^B \cdot b + a^S \cdot b)^\omega$, which is neither ωB- nor ωS-regular (the constructor $+$ must not be thought of as performing the union of the two different (sub-)languages, but rather as a "shuffling operator" that mixes ω-iterations belonging to them), and it is not closed under complementation. A counter-example is given precisely by L, whose complement is not ωBS-regular.

Here, we focus on a new variant of $(.)^*$, that we call (strong) *T-constructor* and denote by $(.)^T$, to be used in the scope of $(.)^\omega$. An expression R^T occurring (in the scope of $(.)^\omega$) in some ω-expression E forces two conditions on the ω-words w belonging to E: (i) the sequence of exponents of R in w features an infinite number of distinct exponents, and (ii) every exponent occurring in the sequence occurs infinitely often. Hence, the distinctive features of these ω-words is that they feature infinitely many exponents occurring infinitely often. It can be easily checked that the complement of L above can be defined as $((a^* \cdot b)^* \cdot a^T \cdot b)^\omega + (a^* \cdot b^*)^* \cdot a^\omega$, and thus it belongs to the class of ωT-regular languages. As for the relationship between ωBS- and ωT-regular languages, it is easy to devise an ωT-regular language whose complement is not ωBS-regular and, vice versa, an ωBS-regular language whose complement is not ωT-regular. Despite of that, we can say that the T-constructor somehow complements the B- and the S-constructor as, when paired with them, it makes it possible to define $(.)^*$: for every BST-regular expression e, it indeed holds that $e^* = e^B + e^S + e^T$ (*BST-regular expressions* are obtained from BS-regular ones by enriching them with the T-constructor). The proof can be found in [3].

The class of *ωT-regular languages* is the class of languages defined by ωT-regular expressions, which are built on top of T-regular expressions, just as

ω-regular expressions are built on top of regular ones. Let Σ be a finite, non-empty alphabet. A *T-regular expression* over Σ is defined by the grammar: $e ::= \varnothing \mid a \mid e \cdot e \mid e + e \mid e^* \mid e^T$, with $a \in \Sigma$. T-regular expressions differ from standard regular ones for the presence of $(.)^T$. Since $(.)^T$ constrains the behavior of the sequence of ω-iterations to the limit, it is not possible to simply define the semantics of T-regular expressions in terms of languages of (finite) words, and then to obtain ωT-regular languages through infinitely many, unrelated iterations of such words. We give their semantics in terms of languages of infinite sequences of finite words; suitable constraints are then imposed to such sequences to capture the intended meaning of $(.)^T$.

Let $\mathcal{L}(e)$ be the language defined by an expression e. Moreover, let $\mathbb{N}$ be the set of natural numbers, $\mathbb{N}_{>0} = \mathbb{N}\backslash\{0\}$, and, given an infinite sequence $\boldsymbol{u}$ of finite words over Σ, let u_i, with $i \in \mathbb{N}^+$, be the i-th element of $\boldsymbol{u}$. The semantics of T-regular expressions over Σ is defined as follows (hereafter we assume $f(0) = 1$):

- $\mathcal{L}(\varnothing) = \varnothing$;
- for $a \in \Sigma$, $\mathcal{L}(a)$ is the infinite sequence of the one-letter word a $\{(a, a, a, \ldots)\}$;
- $\mathcal{L}(e_1 \cdot e_2) = \{\boldsymbol{w} \mid \forall i.w_i = u_i \cdot v_i,\ \boldsymbol{u} \in \mathcal{L}(e_1),\ \boldsymbol{v} \in \mathcal{L}(e_2)\}$;
- $\mathcal{L}(e_1 + e_2) = \{\boldsymbol{w} \mid \forall i.w_i \in \{u_i, v_i\},\ \boldsymbol{u}, \boldsymbol{v} \in \mathcal{L}(e_1) \cup \mathcal{L}(e_2)\}$;[5]
- $\mathcal{L}(e^*) = \{(u_{f(0)}u_2 \ldots u_{f(1)-1}, u_{f(1)} \ldots u_{f(2)-1}, \ldots) \mid \boldsymbol{u} \in \mathcal{L}(e)$ and $f : \mathbb{N} \to \mathbb{N}_{>0}$ is an unbounded and nondecreasing function$\}$;
- $\mathcal{L}(e^T) = \{(u_{f(0)}u_2 \ldots u_{f(1)-1}, u_{f(1)} \ldots u_{f(2)-1}, \ldots) \mid \boldsymbol{u} \in \mathcal{L}(e)$ and $f : \mathbb{N} \to \mathbb{N}_{>0}$ is an unbounded and nondecreasing function such that
  (i) $\forall n \exists i.f(i+1) - f(i) > n$, and
  (ii) $\forall n.[\text{if } \exists i.f(i+1) - f(i) = n, \text{then } \forall k \exists j > k.f(j+1) - f(j) = n]\}$.

Given a sequence $\boldsymbol{v} = (u_{f(0)}u_2 \ldots u_{f(1)-1}, u_{f(1)} \ldots u_{f(2)-1}, \ldots) \in t^{op}$, with $op \in \{*, T\}$, we define the *sequence of exponents of t in $\boldsymbol{v}$*, denoted by $N(\boldsymbol{v})$, as the sequence $\Big(f(i) - f(i-1)\Big)_{i \in \mathbb{N}_{>0}}$. While the semantics of $(.)^*$ does not impose any constraint on $N(\boldsymbol{v})$, the semantics of $(.)^T$ guarantees the existence of infinitely many distinct exponents in $N(\boldsymbol{v})$ (item (i)) and forces each exponent occurring at least once in $N(\boldsymbol{v})$ to occur infinitely often (item (ii)).

The *ω-constructor* turns languages of infinite sequences of words into languages of infinite words. Let e be a T-regular expression. The semantics of $(.)^\omega$ is defined as: $\mathcal{L}(e^\omega) = \{w \mid w = u_1u_2u_3 \ldots$ for some $\boldsymbol{u} \in \mathcal{L}(e)\}$. *$\omega T$-expressions* are defined by the grammar: $E ::= E + E \mid R \cdot E \mid e^\omega$, where R is a regular expression, e is a T-regular expression, and $+$ and $\cdot$ respectively denote union and concatenation of word languages (formally, $\mathcal{L}(E_1 + E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$ and $\mathcal{L}(E_1 \cdot E_2) = \{u \cdot v \mid u \in \mathcal{L}(E_1), v \in \mathcal{L}(E_2)\}$).[6]

The mapping of ωT-regular languages into CQ automata. We now show how to map an ωT-regular expression E into a corresponding CQ automaton $\mathcal{A}$ such

[5] Unlike the case of word languages, when applied to languages of word sequences, the operator $+$ does not return the union of the two argument languages. As an example, $\mathcal{L}(a) \cup \mathcal{L}(b) \subsetneq \mathcal{L}(a + b)$, as witnessed by the word sequence $(a, b, a, b, a, b, \ldots)$. In general, for all t_1, t_2, it holds that $\mathcal{L}(t_1) \cup \mathcal{L}(t_2) \subseteq \mathcal{L}(t_1 + t_2)$.

[6] Notice the abuse of notation with the previous definition of the operators $+$ and $\cdot$ over languages of infinite word sequences.

that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(E)$. We build $\mathcal{A}$ in a compositional way: for each sub-expression E' of E, starting from the atomic ones, we introduce a set $\mathcal{S}_{E'}$ of CQ automata; then, we show how to produce the set of automata for complex sub-expressions by suitably combining automata in the sets associated with their sub-expressions. Eventually, we obtain a set of automata for the ωT-regular expression E. $\mathcal{A}$ results from a suitable merge of the automata in such a set (due to lack of space, the proof of the following result is omitted; it can be found in [3]).

Theorem 3. *For every ωT-regular expression E, there exists a CQ automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(E)$.*

5 Conclusions

In this paper, we introduced and studied a new class of counter automata, called counter-queue automata (CQ automata), and we proved the decidability of their emptiness problem in 2ETIME. Then, we applied them to the analysis of extended ω-regular languages. We defined a new extension of ω-regular languages, called ωT-regular languages, that captures meaningful languages not belonging to the well-known class of ωBS-regular ones, and we provided an encoding of ωT-regular expressions into CQ automata. Whether or not ωT-regular languages are expressively complete with respect to CQ automata is an open problem.

References

1. Alur, R., Henzinger, T.A.: Finitary fairness. ACM Trans. Program. Lang. Syst. 20(6), 1171–1194 (1998)
2. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of multi-pushdown automata is 2ETIME-complete. In: Developments in Language Theory. pp. 121–133 (2008)
3. Barozzini, D., Della Monica, D., Montanari, A., Sala, P.: Extending ω-regular languages with a strong T-constructor: ωT-regular languages and counter-queue automata, Research Report 2017/01, Dept. of Mathematics, Computer Science, and Physics, University of Udine, Italy
4. Bojańczyk, M.: A bounding quantifier. In: CSL. LNCS, vol. 3210, pp. 41–55 (2004)
5. Bojańczyk, M.: Weak MSO with the unbounding quantifier. Theory of Computing Systems 48(3), 554–576 (2011)
6. Bojańczyk, M., Colcombet, T.: Bounds in ω-regularity. In: LICS. pp. 285–296 (2006)
7. Breveglieri, L., Cherubini, A., Citrini, C., Crespi-Reghizzi, S.: Multi-push-down Languages and Grammars. Int. J. of Foundations of Computer Science 7(03), 253–291 (1996)
8. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proc. of the 1960 Int. Congress on Logic, Methodology and Philosophy of Science. pp. 1–11 (1962)
9. Elgot, C.C., Rabin, M.O.: Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. J. Symb. Log. 31(2), 169–181 (1966)
10. Kupferman, O., Piterman, N., Vardi, M.Y.: From liveness to promptness. Formal Methods in System Design 34(2), 83–103 (2009)
11. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Information and Control 9(5), 521–530 (1966)