Beyond ω-regular languages: ωT-regular expressions and their automata and logic counterparts[☆]

David Barozzini[a], David de Frutos-Escrig[b], Dario Della Monica[c,b,*], Angelo Montanari[a], Pietro Sala[d]

[a] *Università di Udine, Italy*
[b] *Universidad Complutense de Madrid, Spain*
[c] *Università "Federico II" di Napoli, Italy*
[d] *Università di Verona, Italy*

Abstract

In the last years, some extensions of ω-regular languages, namely, ωB-regular (ω-regular languages extended with boundedness), ωS-regular (ω-regular languages extended with strong unboundedness), and ωBS-regular languages (the combination of ωB- and ωS-regular ones), have been proposed in the literature. While the first two classes satisfy a generalized closure property, which states that the complement of an ωB-regular (resp., ωS-regular) language is an ωS-regular (resp., ωB-regular) one, the last class is not closed under complementation. The existence of non-ωBS-regular languages that are the complements of some ωBS-regular ones and express fairly natural asymptotic behaviors motivates the search for other significant classes of extended ω-regular languages. In this paper, we present the class of ωT-regular languages, which includes meaningful languages that are not ωBS-regular. We define this new class of languages in terms of ωT-regular expressions. Then, we introduce a new class of automata (counter-check automata) and we prove that (i) their emptiness problem is decidable in PTIME, and (ii) they are expressive enough to capture ωT-regular languages. We also provide an encoding of ωT-regular expressions into S1S+U. Finally, we investigate a stronger variant of ωT-regular languages (ωT_s-regular languages). We characterize the resulting class of languages in terms of ωT_s-regular expressions, and we show how to map it into a suitable class of automata, called counter-queue automata. We conclude the paper with a comparison of the expressiveness of ωT- and ωT_s-regular languages and of the

[☆] This paper is an extended and merged version of [1] and [2]. Part of the work was done while A. Montanari was visiting the Universidad Complutense de Madrid. The work was supported by the Italian INdAM-GNCS project *Formal methods for the verification and synthesis of discrete and hybrid systems*. D. Della Monica acknowledges the financial support from a Marie Curie INdAM-COFUND-2012 Outgoing Fellowship.

[*] Corresponding author.

Email addresses: dbaro13@gmail.com (David Barozzini), defrutos@sip.ucm.es (David de Frutos-Escrig), dario.dellamonica@unina.it (Dario Della Monica), angelo.montanari@uniud.it (Angelo Montanari), pietro.sala@univr.it (Pietro Sala)

corresponding automata.

Keywords: ω-regular languages, ω-regular expressions, counter automata, monadic second-order logic of one successor

1. Introduction

Regular languages of infinite words (ω-regular languages for short) play a fundamental role in computer science, as they provide a natural setting for the specification and verification of nonterminating finite-state systems. Since the seminal work by Büchi [3], McNaughton [4], and Elgot and Rabin [5] in the sixties, a great research effort has been devoted to the theory and the applications of ω-regular languages. Equivalent characterisations of ω-regular languages have been given in terms of formal languages (ω-*regular expressions*), automata (Büchi, Rabin, and Muller automata), classical logic (weak/strong monadic second-order logic of one successor, WS1S/S1S for short), and temporal logic (Quantified Linear Temporal Logic, Extended Temporal Logic) [6].

Recently, it has been shown that ω-regular languages can be extended in some reasonable ways, preserving their decidability and some of their closure properties [7, 8, 9, 10]. As an example, ω-regular languages can be extended with the ability of constraining the distance between consecutive occurrences of a given symbol to be (un)bounded (in the limit). Boundedness comes into play in the study of *finitary fairness* as opposed to the classic notion of *fairness*, widely used in automated verification of concurrent systems. According to the latter, no individual process in a multi-process system may be ignored for ever; finitary fairness imposes the stronger constraint that every enabled transition is executed within at most b time-units, where b is an unknown, constant bound. In [11], it is shown that finitary fairness enjoys some desirable mathematical properties that are violated by the weaker notion of fairness, and yet it captures all reasonable schedulers' implementations. The same property has been investigated from a logical perspective in [12], where the logic PROMPT-LTL is introduced. Roughly speaking, PROMPT-LTL extends LTL with the *prompt-eventually* operator, which states that an event will happen within the next b time-units, b being an unknown, constant bound. An analogous extension has been proposed for the propositional interval logic of temporal neighborhood PNL in [13].

From the point of view of formal languages, the proposed extensions pair the Kleene star $(.)^*$ with bounded/unbounded variants of it. Intuitively, the bounded exponent $(.)^B$ (aka B-constructor) constrains parts of the input word to be of bounded size, while the strongly unbounded exponent $(.)^S$ (S-constructor) forces parts of the input word to be arbitrarily large. The two extensions have been studied both in isolation (ωB- and ωS-regular expressions) and in conjunction (ωBS-regular expressions). Equivalent characterisations of extended ω-regular languages are given in [7, 8] in terms of automata (ωB-, ωS-, and ωBS-automata) and classical logic (fragments of WS1S+U, i.e., the extension

of WS1S with the unbounding quantifier $\mathbb{U}$ [14], that allows one to express properties which are satisfied by finite sets of arbitrarily large size).[1] In [8], the authors also show that the complement of an ωB-regular language is an ωS-regular one and vice versa, and that ωBS-regular languages, featuring both B- and S-constructors, strictly extend ωB- and ωS-regular languages and are not closed under complementation.

In this paper, we focus on those ω-languages which are complements of ωBS-regular ones, but are not ωBS-regular. We start with an in-depth analysis of one such language [8], that allows us to identify a new meaningful extension of ω-regular languages including it (ωT-regular languages), obtained by adding a new, fairly natural constructor $(.)^T$ to the standard constructors of ω-regular expressions. Among others, an interesting feature of $(.)^T$ is that pairing $(.)^B$ and $(.)^S$ with it one can capture all possible ways of instantiating $*$-expressions (this is not the case with $(.)^B$ and $(.)^S$ only). In view of that, it can be said that $(.)^T$ "complements" $(.)^B$ and $(.)^S$ with respect to $(.)^*$.

We provide a characterization of ωT-regular languages in terms of ωT-regular expressions, which benefits from a generalization (conservative with respect to ωBS-regular languages) of the semantics of the mix/shuffle operator + given in [7, 8]. Moreover, we introduce a new class of automata, called *counter-check automata*, that are expressive enough to capture ωT-regular languages, and we show that their emptiness problem is decidable in PTIME. We also provide an encoding of ωT-regular expressions (languages) into S1S+U. Finally, we study a stronger variant of $(.)^T$, denoted by $(.)^{T_s}$, and we show that, to a large extent, the results obtained for $(.)^T$ can be replicated for it. In particular, it is possible to introduce a new class of automata, called *counter-queue automata*, that generalize counter-check ones, whose emptiness problem can be proved to be decidable in 2ETIME and which are expressive enough to capture ω-regular languages extended with $(.)^{T_s}$.

The paper is organized as follows. In Section 2, we introduce and discuss existing extensions of ω-regular languages with a special attention to ωBS-regular languages. Then, in Section 3, we define and study the new class of ωT-regular languages. Next, in Section 4, we define counter-check automata (CCA), we prove that their emptiness problem is decidable in PTIME, and we provide an encoding of ωT-regular languages into CCA. In Section 5, we show that ωT-regular languages can be defined in S1S+U. In Section 6, we define counter-queue automata (CQA), we prove that their emptiness problem is decidable in 2ETIME, and we provide an encoding of ωT_s-regular languages into CQA. Finally, in Section 7, we compare the expressiveness of ωT- and ωT_s-regular languages and of CCA and CQA. Conclusions provide an assessment of the work done and outline future research directions.

This paper is an extended and merged version of [1] and [2]. A preliminary partial account of the work on the T (resp., T_s) operator appeared in [1] (resp., [2, 16]). Besides detailed proofs of all the results and many other specific

[1] Undecidability of full S1S+U has been shown in [15].

improvements, which were missing in the conference papers [1, 2], we added several results in Section 2 (among others, the interleaving semantics of the shuffle operator + and the analysis of its relationships with the original mix one are completely new) and the whole Section 7 (comparison between T/CCA and T_s/CQA).

2. Beyond ω-regularity: ωB-, ωS-, and ωBS-regular languages

In this section, we give a short account of the extensions of ω-regular languages proposed in the literature (details can be found in [7, 8, 9]).

It is well known that ω-regular languages can be defined by means of ω-regular expressions. By exploiting such a characterization, any ω-word of an ω-regular language can be viewed as the concatenation of a finite prefix, belonging to a regular language, and an infinite sequence of finite words of another regular language (we call each of these words an *ω-iteration*). An interesting case is that of ω-iterations consisting of a finite sequence of words generated by an occurrence of the Kleene star operator $(.)^*$, aka *$*$-constructor*, in the scope of the *ω-constructor* $(.)^\omega$, e.g., the ω-regular expression $(a^*b)^\omega$ generates the language of ω-words featuring an infinite sequence of ω-iterations, each one consisting of a finite (possibly empty) sequence of a's followed by exactly one b. Given an ω-regular expression E featuring an occurrence of $(.)^*$ (sub-expression R^*) in the scope of $(.)^\omega$ and an ω-word w belonging to the language of E, we refer to the sequences of the sizes of the (maximal) blocks of consecutive iterations of R in the different ω-iterations as the *sequences of exponents of R in (the ω-iterations of) w*.[2] For instance, let $w = abaabaaab\ldots$ be an ω-word generated by $(a^*b)^\omega$. The sequence of exponents of a in w, which, in this case, is unique, is $1, 2, 3, \ldots$. Sometimes, we will denote words in a compact way by explicitly indicating the exponents of a sub-expression, e.g., we will write w as $a^1ba^2ba^3b\ldots$.

Given an expression E, let $\mathcal{L}(E)$ be the language defined by E. With a little abuse of notation, we will sometimes identify a language with the expression defining it, and vice versa, e.g., we will write "language $(a^*b)^\omega$" for "language $\mathcal{L}((a^*b)^\omega)$". Notice that $(.)^*$ allows one to impose the existence of a finite sequence of words (described by its argument expression) within each ω-iteration, but it cannot be used to express properties of sequences of exponents of its argument expression in the ω-iterations of an ω-word. To overcome such a limitation, some meaningful extensions of ω-regular expressions have been investigated in the last years, that make it possible to constrain the behavior of $(.)^*$ in the limit.

A first class of extended ω-regular languages is that of ωB-regular languages, that allow one to impose boundedness conditions. 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 bounded exponent B allows one to constrain the argument R of the expression R^B to be repeated

[2] We refer to sequences of exponents as, in general, there is more than one such sequence.

in each ω-iteration a number of times less than a certain bound fixed for the whole ω-word. As an example, the expression $(a^B b)^\omega$ denotes the language of ω-words in $(a^* b)^\omega$ for which there is an upper bound on the number of consecutive occurrences of a, that is, the sequence of exponents of a is bounded. As the bound may vary from word to word, the language is not ω-regular.

The class of ωS-regular languages extends that of ω-regular ones with strong unboundedness (also referred to as strict unboundedness in [8]). By analogy with ωB-regular expressions, ωS-regular expressions are obtained from ω-regular ones by adding a 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 natural number $k > 0$, the strongly unbounded exponent S constrains the number of ω-iterations in which the argument R is repeated at most k times to be finite. Let us consider ω-words that feature an infinite number of instantiations of the expression R^S, that is, ω-words for which there exists an infinite number of ω-iterations including a sequence of consecutive R's generated by R^S. It can be easily checked that in these words the sequence of exponents of R tends towards infinity, due to the S-constructor. As an example, the expression $(a^S b)^\omega$ denotes the language of ω-words w in $(a^* b)^\omega$ such that, for any $k > 0$, there exists a suffix of w that only features maximal sequences of consecutive a's that are longer than k.

Finally, ωBS-regular languages are generated by ωBS-regular expressions, which pair the operators of ω-regular expressions with both $(.)^B$ and $(.)^S$.

2.1. ωBS-regular expressions

In the following, we give a detailed account of ωBS-regular expressions, which are built on top of BS-regular expressions, just as ω-regular expressions are built on top of regular ones. Let Σ be a finite, nonempty alphabet. A *BS-regular expression* over Σ is defined by the grammar [8]:

$$e ::= \emptyset \mid a \mid e \cdot e \mid e + e \mid e^* \mid e^B \mid e^S, \text{ where a } \in \Sigma$$

Sometimes, we will omit the operator $\cdot$, thus writing, e.g., ee for $e \cdot e$.

BS-regular expressions differ from regular ones in that they allow constructors $(.)^B$ and $(.)^S$. Since these operators constrain the behavior of the sequence of ω-iterations in the limit, it is not possible to simply define the semantics of BS-regular expressions in terms of languages of (finite) words, and then to obtain ωBS-regular languages through infinitely many, unrelated iterations of such words. The semantics of BS-regular expressions is given in terms of languages of infinite sequences of finite words, and suitable constraints are imposed on such sequences to capture the intended meaning of $(.)^B$ and $(.)^S$.

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}_{>0} = \mathbb{N} \setminus \{0\}$. For an infinite sequence $\vec{u}$ of finite words over Σ and $i \in \mathbb{N}_{>0}$, we denote by u_i the i-th element of $\vec{u}$. The semantics of BS-regular expressions over Σ is defined as follows:

- $\mathcal{L}(\emptyset) = \emptyset$;
- for $a \in \Sigma$, $\mathcal{L}(a)$ only contains the infinite sequence of the one-letter word a, that is, $\mathcal{L}(a) = \{(a, a, a, \ldots)\}$;

- $\mathcal{L}(e_1 \cdot e_2) = \{\vec{w} \mid \forall i.w_i = u_i \cdot v_i,\ \vec{u} \in \mathcal{L}(e_1),\ \vec{v} \in \mathcal{L}(e_2)\}$;
- $\mathcal{L}(e_1 + e_2) = \{\vec{w} \mid \forall i.w_i \in \{u_i, v_i\},\ \vec{u}, \vec{v} \in \mathcal{L}(e_1) \cup \mathcal{L}(e_2)\}$;
- $\mathcal{L}(e^*) = \{(u_{f(0)}u_2 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots) \mid \vec{u} \in \mathcal{L}(e)$ and $f : \mathbb{N} \to \mathbb{N}_{>0}$ is a nondecreasing function with $f(0) = 1\}$;
- $\mathcal{L}(e^B) = \{(u_{f(0)}u_2 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots) \mid \vec{u} \in \mathcal{L}(e)$ and $f : \mathbb{N} \to \mathbb{N}_{>0}$ is a nondecreasing function, with $f(0) = 1$, such that $\exists n \in \mathbb{N}\ \forall i \in \mathbb{N}.(f(i+1) - f(i) < n)\}$;
- $\mathcal{L}(e^S) = \{(u_{f(0)}u_2 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots) \mid \vec{u} \in \mathcal{L}(e)$ and $f : \mathbb{N} \to \mathbb{N}_{>0}$ is a nondecreasing function, with $f(0) = 1$, such that $\forall n \in \mathbb{N}\ \exists k \in \mathbb{N}\ \forall i > k(f(i+1) - f(i) > n)\}$.

It is worth noticing that the constructor + must not be thought of as performing the union of two languages, but rather as a "shuffling operator" that mixes ω-iterations belonging to the two different (sub)languages. Unlike the case of word languages, indeed, when applied to languages of word sequences, it 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 $(a, b, a, b, a, b, \dots)$. In general, for all BS-regular expressions e_1, e_2, it holds that $\mathcal{L}(e_1) \cup \mathcal{L}(e_2) \subseteq \mathcal{L}(e_1 + e_2)$. Notice also that $e_1 + e_2$ allows for mixing two word sequences $\vec{u}$ and $\vec{v}$ belonging to the same expression e_1 (or e_2). This is necessary in order for the empty language to behave as the identity, i.e., $e + \emptyset = e$ for all e (by imposing $\vec{u} \in e_1$ and $\vec{v} \in e_2$, the empty language behaves as the absorbing element, i.e., $e + \emptyset = \emptyset$ for all e).

Given a sequence $\vec{v} = (u_{f(0)}u_2 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots) \in e^{op}$, with $\vec{u} \in e$ and $op \in \{*, B, S\}$, we define the *sequence of exponents of e in $\vec{v}$*, denoted by $N(\vec{v})$, as the sequence $\Big(f(i+1) - f(i)\Big)_{i \in \mathbb{N}}$. While the $*$-constructor does not impose any constraint on $N(\vec{v})$, the B-constructor forces it to be bounded (in such a case, we say that $\vec{v}$ enjoys the *B-property*) and the S-constructor forces it to be strongly unbounded, that is, its limit inferior is infinite (equivalently, the S-constructor imposes that no exponent occurs infinitely often in the sequence—we say that $\vec{v}$ enjoys the *S-property*).

The *ω-constructor* turns languages of infinite word sequences into languages of ω-words. Let e be a BS-regular expression. The semantics of the ω-constructor is defined as follows:

- $\mathcal{L}(e^\omega) = \{w \mid |w| = \infty$ and $w = u_1u_2u_3 \dots$ for some $\vec{u} \in \mathcal{L}(e)\}$.

ωBS-expressions are defined by the following grammar (we denote languages of word sequences by lowercase letters, such as e, e_1, …, and languages of words by uppercase ones, such as E, E_1, …, R, R_1, …):

$$E \ ::= \ E + E \ \mid \ R \cdot E \ \mid \ e^\omega$$

where R is a regular expression, e is a BS-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 $\mathcal{L}(E_1 \cdot E_2) = \{u \cdot v \mid u \in \mathcal{L}(E_1), v \in \mathcal{L}(E_2)\}$).[3] As we did for languages of word sequences, we will sometimes omit the operator $\cdot$ between word languages.

2.2. The shuffle *operator* +

Some remarks are in order about the *shuffle* operator +. (As a matter of fact, + is referred to as *mix* operator in [8, 9]; we find the term "shuffle" a better fit in view of the considerations below and the new conservative semantics of the operator arising from them.) We already pointed out that its semantics is quite different from the one it has in the realm of standard ω-regular expressions. In the following, we state some fundamental properties of + that are instrumental to a generalization of its semantics (see Section 3), which is conservative with respect to the one originally given in [8, 9].

To start with, we define the notion of *selection function* that allows us to provide an alternative, equivalent characterization of +.

A *selection function* is a function $g : \mathbb{N}_{>0} \to \{1, 2\}$. We say that g is:

- *1-stable*, if there is $k \in \mathbb{N}_{>0}$ such that $g(x) = 1$ for all $x > k$,
- *2-stable*, if there is $k \in \mathbb{N}_{>0}$ such that $g(x) = 2$ for all $x > k$, or
- *alternating*, if it is neither 1-stable nor 2-stable.

Definition 1 (mix/shuffle)**.** *Let g be a selection function, and $\vec{v}^1 = (v_1^1, v_2^1, \ldots)$ and $\vec{v}^2 = (v_1^2, v_2^2, \ldots)$ be two word sequences. We define g-mix$(\vec{v}^1, \vec{v}^2)$ as the word $\vec{v} = (v_1, v_2, \ldots)$, where $v_i = v_i^{g(i)}$ for all $i \in \mathbb{N}_{>0}$. We define g-shuffle$(\vec{v}^1, \vec{v}^2)$ as the word $\vec{v} = (v_1, v_2, \ldots)$, where $v_i = v^{g(i)}_{|\{j \in \mathbb{N}_{>0} | j \leq i \text{ and } g(j) = g(i)\}|}$ for all $i \in \mathbb{N}_{>0}$. Finally, we say that $\vec{v}$ is a* mix *(resp.,* shuffle*) of $\vec{v}^1$ and $\vec{v}^2$ if there is a selection function g such that g-mix$(\vec{v}^1, \vec{v}^2) = \vec{v}$ (resp., g-shuffle$(\vec{v}^1, \vec{v}^2) = \vec{v}$).*

Proposition 1. *For any BS-regular expression $e_1 + e_2$, it holds that $\vec{v} = (v_1, v_2, \ldots) \in \mathcal{L}(e_1 + e_2)$ if and only if there are two word sequences $\vec{v}^1 = (v_1^1, v_2^1, \ldots)$ and $\vec{v}^2 = (v_1^2, v_2^2, \ldots)$, with $\vec{v}^1, \vec{v}^2 \in \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$, such that $\vec{v}$ is a mix of $\vec{v}^1$ and $\vec{v}^2$.*

Intuitively, there is a selection function g that, for every $i \in \mathbb{N}_{>0}$, picks the element used to fill position i of $\vec{v} \in \mathcal{L}(e_1 + e_2)$, which is the ith element of either $\vec{v}^1$ or $\vec{v}^2$. Notice that elements that are not picked are discharged. Consider, for instance, the expression $e = a^* + b^*$ and the word sequence $\vec{v} = (a^1, b^2, a^3, b^4, \ldots)$. Clearly, $\vec{v} \in \mathcal{L}(e)$, as witnessed by sequences $\vec{v}^1 = (a^1, a^2, a^3, \ldots) \in \mathcal{L}(a^*)$ and $\vec{v}^2 = (b^1, b^2, b^3, \ldots) \in \mathcal{L}(b^*)$, and selection function g, with $g(i) = 1$ if i is odd and $g(i) = 2$ if i is even. Elements a^2, a^4, a^6, ... (resp., b^1, b^3, b^5) from sequence $\vec{v}^1$ (resp., $\vec{v}^2$) are discharged, meaning that they never appear in $\vec{v}$.

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

Roughly speaking, both sequences $\vec{v}^1$ and $\vec{v}^2$ contain *holes*, that is, finite blocks of adjacent elements that are discarded.

The following *subsequence* and *supersequence* closure properties of BS-regular expressions can be easily proved by structural induction.

Proposition 2 (Subsequence closure for BS-regular expressions [9])**.** *Let e be a BS-regular expression and $\vec{v} \in \mathcal{L}(e)$. Then, for every infinite subsequence $\vec{u}$ of $\vec{v}$, it holds that $\vec{u} \in \mathcal{L}(e)$.*

Notice that subsequence closure implies suffix closure (if a word sequence belongs to a given BS-regular language, then all of its suffixes belong to it).

Proposition 3 (Supersequence closure for BS-regular expressions)**.** *Let e be a BS-regular expression and $\vec{v} \in \mathcal{L}(e)$. Then, it is possible to extend $\vec{v}$ by inserting arbitrarily many finite words at arbitrarily chosen positions (possibly infinitely many) in such a way that the resulting sequence belongs to $\mathcal{L}(e)$.*

It is worth pointing out that while subsequence closure is universal (it states a property of all subsequences), supersequence one is existential (it guarantees the existence of supersequences).

Thanks to Propositions 2 and 3, Proposition 1 can be reformulated as follows.

Proposition 4. *For any BS-regular expression $e_1 + e_2$, it holds that $\vec{v} = (v_1, v_2, \ldots) \in \mathcal{L}(e_1 + e_2)$ if and only if there are two word sequences $\vec{v}^1 = (v^1_1, v^1_2, \ldots)$ and $\vec{v}^2 = (v^2_1, v^2_2, \ldots)$, with $\vec{v}^1, \vec{v}^2 \in \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$, such that $\vec{v}$ is a shuffle of $\vec{v}^1$ and $\vec{v}^2$.*

Proposition 4 can be read as follows: a word sequence $\vec{v}$ belongs to the BS-regular language $e_1 + e_2$ if it is witnessed by two sequences $\vec{v}^1, \vec{v}^2 \in \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$ and a selection function g that, in order to fill position i of $\vec{v}$, chooses an element from either $\vec{v}^1$ or $\vec{v}^2$ without discharging any element (except, possibly, an entire infinite suffix), that is, without creating any holes. However, the sequence $\vec{v}$ cannot be, in general, built from these two sequences $\vec{v}^1$ and $\vec{v}^2$ by means of $+$.

The existence of two sequences in $\mathcal{L}(e_1) \cup \mathcal{L}(e_2)$ whose *shuffle* generates the sequence in $\mathcal{L}(e_1 + e_2)$ is at the basis of the new semantics of $+$ we will give in the next section. We will use it in the definition of ωT-regular languages; moreover, by exploiting Proposition 4, we will prove that replacing the original semantics of $+$ by the new one does not change the class of languages captured by ωBS-regular expressions.

2.3. ωBS-regular languages and closure under complementation

One of the deepest technical result in the area is the following theorem, by Bojańczyk and Colcombet [8].

Theorem 1 ([8, Theorem 4.1])**.** *The complement of an ωS-regular language is ωB-regular. The complement of an ωB-regular language is ωS-regular.*

In [8], the authors also show that the inclusion of the classes of ωB- and ωS-regular languages in the class of ωBS-regular ones is strict, as witnessed by the ωBS-regular language $L = (a^B b + a^S b)^\omega$ consisting of those ω-words w featuring infinitely many occurrences of b and such that there are only finitely many numbers occurring infinitely often in the sequence of exponents of a in w, that is, there is a bound k such that no $h > k$ occurs infinitely often in the sequence of exponents of a in w, but infinitely many exponents may occur finitely many times. L is neither ωB- nor ωS-regular. Moreover, they prove that the class of ωBS-regular languages is not closed under complementation. A counterexample is given precisely by L, whose complement is not ωBS-regular (notice that, due to Theorem 1, ωBS-regular languages whose complement is not an ωBS-regular language are neither ωB- nor ωS-regular languages).

In this paper, we investigate those ω-languages that do not belong to the class of ωBS-regular languages, but whose complement may belong to it. Let us consider, for instance, the complement $\overline{L}$ of the language L above. Any word w in $\overline{L}$ that features infinitely many occurrences of b, that is, $w \in (a^*b)^\omega$, is such that there are infinitely many natural numbers that occur infinitely often in the sequence of exponents of a in w. By way of contradiction, suppose that there are only finitely many of them and let k be the largest one. Now, w can be viewed as an infinite sequence of ω-iterations, each of them characterised by the corresponding exponent of a. If the exponent associated with an ω-iteration is greater than k, then it does not occur infinitely often, and thus the ω-iteration is captured by the sub-expression $a^S b$. Otherwise, if the exponent is not greater than k, then the corresponding ω-iteration is captured by the sub-expression $a^B b$. As an example, the ω-word $a^1ba^2ba^1ba^3ba^1ba^4b\ldots$ does not belong to $\overline{L}$ as 1 is the only exponent occurring infinitely often, while the ω-word $a^1ba^2ba^1ba^2ba^3ba^1ba^2ba^3ba^4b\ldots$ *does* belong to it as infinitely many (actually all) natural numbers occur infinitely often in the sequence of exponents.

3. The constructor $(.)^T$ and the class of ωT-regular languages

In this section, we define and study a new extension of the class of ω-regular languages, called ωT-regular languages, which aims at capturing those ω-languages that are the complements of ωBS-regular ones while being not ωBS-regular. As already pointed out, this is the case with the complement $\overline{L}$ of the ωBS-regular language $L = (a^B b + a^S b)^\omega$. To a large extent, ω-words belonging to $\overline{L}$ are characterised by sequences of exponents where infinitely many exponents occur infinitely often. The class of *ωT-regular languages* includes those extended ω-regular languages that satisfy such a property.

3.1. ωT-regular expressions

The proposed extension of ω-regular languages has two main ingredients: a new variant of $(.)^*$, called *T-constructor* and denoted by $(.)^T$, to be used in the scope of $(.)^\omega$, and a new semantics for the shuffle operator $+$, which is based on the results stated by Proposition 4.

Intuitively, given an ωT-regular expression E and an ω-word $w \in E$, an expression R^T occurring in E forces the sequence of exponents of R in w to feature infinitely many different elements occurring infinitely often. Formally, the class of ωT-regular languages is the class of languages that can be expressed by *ωT-regular expressions*, which are in turn defined by the following grammar:

$$\begin{array}{lcl} E & ::= & E + E \mid R \cdot E \mid e^{\omega} \\ e & ::= & \emptyset \mid a \mid e \cdot e \mid e + e \mid e^{*} \mid e^{T} \end{array}$$

where R is a regular expression and $a \in \Sigma$.

The sub-grammar rooted in the non-terminal e generates the *T-regular expressions*. From a syntactic point of view, T-regular expressions differ from BS-regular ones for the replacement of $(.)^B$ and $(.)^S$ by $(.)^T$ only. However, as already pointed out, we give a different semantics for the shuffle operator $+$, and thus, to define the semantics of T-regular expressions, we need to provide the following two semantic clauses:

- $\mathcal{L}(e_1 + e_2) = \{\vec{w} \mid \vec{w} \text{ is a shuffle of } \vec{u} \text{ and } \vec{v}, \text{ for some } \vec{u}, \vec{v} \in \mathcal{L}(e_1) \cup \mathcal{L}(e_2)\}$;
- $\mathcal{L}(e^T) = \{(u_{f(0)} u_2 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots) \mid \vec{u} \in \mathcal{L}(e) \text{ and } f : \mathbb{N} \to \mathbb{N}_{>0} \text{ is a nondecreasing function, with } f(0) = 1, \text{ such that } \exists^{\omega} n \in \mathbb{N}\ \forall k \in \mathbb{N}\ \exists i > k.(f(i+1) - f(i) = n)\}$

where e, e_1, and e_2 are word sequences and $\exists^{\omega}$ is a shorthand for "there are infinitely many".

For $\vec{v} = (u_{f(0)} u_2 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots) \in e^T$, with $\vec{u} \in e$, we define the *sequence of exponents of e in $\vec{v}$*, denoted by $N(\vec{v})$, exactly as we did in the case of BS-regular expressions. Moreover, for $op \in \{*, B, S, T\}$ and $\vec{v} = (u_{f(0)} u_2 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots) \in e^{op}$, with $\vec{u} \in e$, we denote by $N_i(\vec{v})$ (resp., $N_f(\vec{v})$) the set of exponents occurring infinitely (resp., finitely) many times in $N(\vec{v})$. For every $\vec{v} \in e^T$, it holds that the cardinality of $N_i(\vec{v})$ is infinite, and thus the formal semantics of the T-constructor conforms with the intuitive one outlined at the beginning of the section.

Let us consider again the language $\overline{L}$. As we will prove shortly, $\overline{L}$ can be expressed as the union of the languages $(a^T b)^{\omega}$ and $(a^* b^*)^* a^{\omega}$, and thus it belongs to the class of ωT-regular languages. $\overline{L}$ is an example of an ωT-regular language that is not ωBS-regular; examples of ωBS-regular languages that are not ωT-regular are the languages $(a^B b)^{\omega}$ and $(a^S b)^{\omega}$.

We conclude the section by observing that, from Proposition 4, it immediately follows that for any pair of BS-regular expressions e_1 and e_2, the languages defined by $e_1 + e_2$ with the original and the new semantics of the shuffle operator $+$ coincide, thus showing that the new definition of $+$ is conservative with respect to ωBS-regular languages. This is obviously not the case with T-regular expressions. Consider the expression $a^T b + a^T b$. It trivially holds that $a^T b + a^T b = a^T b$ (with the new semantics for $+$). On the contrary, $a^T b + a^T b \neq a^T b$ holds with the old semantics of $+$. Consider a pair of sequences $\vec{v}^1$ and $\vec{v}^2$ belonging to $a^T b$, where $\vec{v}^1$ (resp., $\vec{v}^2$) is such that the word ab (resp., aab) occurs at all odd (resp., even) positions and the subsequence consisting of the elements at even

(resp., odd) positions belongs to $a^T b$. The word sequence featuring the word ab at all odd positions and aab at all even ones belongs to $a^T b + a^T b$, but it does not belong to $a^T b$. In general, it is possible to show that, with the old semantics, for any T-regular expression e, $e \subseteq e + e$.

3.2. The constructors B, S, and T and their relationships

In this section, we investigate the relationships among $(.)^B$, $(.)^S$, and $(.)^T$. Let us consider the expressions generated by the basic constructors of regular expressions paired with $(.)^B$, $(.)^S$, and $(.)^T$ (let us call them *BST-regular expressions*). Given a BST-regular expression e over an alphabet Σ, we denote by $reg(e)$ the set $\{u \in \Sigma^* \mid u \text{ occurs in some sequence } \vec{u} \in \mathcal{L}(e)\}$. It can be easily checked that $reg(e) = \mathcal{L}(e')$, where e' is the regular expression obtained from e by replacing every occurrence of $(.)^B$, $(.)^S$, and $(.)^T$ with the Kleene star. A sequence $\vec{v} = (v_1, v_2, \ldots)$ is said to be *e-compatible* if $v_i \in reg(e)$ for all i.

The next proposition shows that $(.)^T$ satisfies a supersequence closure property stronger than the one for BS-languages stated in Proposition 3.

Proposition 5 (Supersequence closure for T-regular expressions)**.** *Let e be a T-regular expression and $\vec{v} \in \mathcal{L}(e)$. Then, for every e-compatible sequence $\vec{u}$ of which $\vec{v}$ is a subsequence, it holds that $\vec{u} \in \mathcal{L}(e)$.*

BST-regular expressions satisfy a property of prefix independence, as formally stated by the following proposition.

Proposition 6 (Prefix independence for BST-regular expressions)**.** *Let e be a BST-regular expression. For every e-compatible sequence $\vec{v} = (v_1, v_2, \ldots)$ and every finite prefix $\vec{u} = (u_1, \ldots, u_h)$ of an e-compatible sequence, it holds that*

$$\vec{v} \in \mathcal{L}(e) \text{ if and only if } \vec{u} \cdot \vec{v} \in \mathcal{L}(e).$$

Since prefix independence implies suffix closure, the above proposition extends suffix closure to BST-regular languages.

As already pointed out in the introduction, one of the motivations for the proposal of the T-constructor stems from the fact that it somehow complements the other two with respect to the Kleene star: while the B-constructor forces the existence of finitely many exponents and the S-constructor constrains the exponents to occur finitely many times, the T-constructor forces the existence of infinitely many times exponents occurring infinitely many times. We can make such a claim more precise as follows. Let e be a BST-regular expression and $\vec{u} \in \mathcal{L}(e^{op})$, with $op \in \{B, S, T\}$. If $\vec{u} \in \mathcal{L}(e^B)$, then $N(\vec{u})$ is bounded, while if either $\vec{u} \in \mathcal{L}(e^S)$ or $\vec{u} \in \mathcal{L}(e^T)$ it is unbounded; moreover, if $\vec{u} \in \mathcal{L}(e^S)$, then $N_i(\vec{u}) = \emptyset$, while if $\vec{u} \in \mathcal{L}(e^T)$, then $N_i(\vec{u})$ is infinite. The next proposition shows that, when paired with $(.)^B$ and $(.)^S$, $(.)^T$ makes it possible to define the Kleene star.[4]

[4]It is worth pointing out that the expression e in the statement is a standard regular expression, that is, e does not feature any occurrence of the constructors B, S, and T, as the formulation of the statement given in [1] (Proposition 1) was incorrect.

Proposition 7. *Let e be a (standard) regular expression. Then, it holds that $e^* = e^B + e^S + e^T$.*

Proof. As for inclusion $\mathcal{L}(e^B + e^S + e^T) \subseteq \mathcal{L}(e^*)$, we observe that $\mathcal{L}(e^B) \subseteq \mathcal{L}(e^*)$, $\mathcal{L}(e^S) \subseteq \mathcal{L}(e^*)$, and $\mathcal{L}(e^T) \subseteq \mathcal{L}(e^*)$. Then, it trivially holds that $\mathcal{L}(e^B + e^S + e^T) \subseteq \mathcal{L}(e^* + e^* + e^*) = \mathcal{L}(e^*)$.

In order to prove the converse inclusion, we assume that $\vec{v} \in \mathcal{L}(e^*)$ and show that $\vec{v} \in \mathcal{L}(e^B + e^S + e^T)$. By definition of e^*, $\vec{v} = (u_{f(0)}u_2 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots)$, for a word sequence $\vec{u} \in \mathcal{L}(e)$ and a nondecreasing function $f : \mathbb{N} \to \mathbb{N}_{>0}$, with $f(0) = 1$.

Let $N(\vec{v})$ be the sequence of exponents $(n_1, n_2, \dots)$ and let $N_f(\vec{v})$ and $N_i(\vec{v})$ be the sets of exponents that respectively occur finitely and infinitely many times in $N(\vec{v})$. We distinguish the following three cases.

If $N_i(\vec{v})$ is infinite, then $\vec{v} \in \mathcal{L}(e^T) \subseteq \mathcal{L}(e^B + e^S + e^T)$.

If both $N_i(\vec{v})$ and $N_f(\vec{v})$ are finite, then $N(\vec{v})$ is bounded, and thus $\vec{v} \in \mathcal{L}(e^B) \subseteq \mathcal{L}(e^B + e^S + e^T)$.

If $N_i(\vec{v})$ is finite and $N_f(\vec{v})$ is infinite, we extract from $\vec{v}$ the (infinite) subsequence of words whose exponents belong to $N_f(\vec{v})$, say it $\vec{u}$, and the (infinite) subsequence of words whose exponents belong to $N_i(\vec{v})$, say it $\vec{w}$. Each word in $\vec{v}$ clearly belongs either to $\vec{u}$ or to $\vec{w}$. It is immediate to see that $\vec{u}$ belongs to $\mathcal{L}(e^S)$ and $\vec{w}$ belongs to $\mathcal{L}(e^B)$, and thus $\vec{v} \in \mathcal{L}(e^B + e^S) \subseteq \mathcal{L}(e^B + e^S + e^T)$. □

The next proposition shows that, unlike $(.)^B$ and $(.)^S$, $(.)^T$ is not idempotent.

Proposition 8 ((Non-)Idempotency)**.** *The following statements hold:*

1. *$(e^B)^B = e^B$, for all BST-regular expressions e;*
2. *$(e^S)^S = e^S$, for all BST-regular expressions e;*
3. *$(e^T)^T \neq e^T$, for some BST-regular expression e.*

Proof. The truth of items 1 and 2 can be easily checked. To prove item 3, it suffices to provide a counterexample: we show that $(a^T)^T \neq a^T$, for $a \in \Sigma$. To this end, let us consider the word sequence $\vec{v} = (a, a^3, a^3, a^3, a^6, a^4, a^{10}, a^5, a^6, a^9, a^6, a^{10}, a^{11}, a^7, a^{15}, a^{13}, a^8, \dots)$ depicted in Figure 1. Such a sequence belongs to $(a^T)^T$, but not to a^T, as no exponent occurs infinitely often. A detailed account of the counter-example, in particular an explanation of the number sequences that come into play, is in order.

Consider the word sequence $\vec{u} = (a, a, a, \dots) \in \mathcal{L}(a)$. By grouping the elements of $\vec{u}$ according to the sequence of exponents $1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, \dots$, we obtain the word sequence $\vec{w} \in \mathcal{L}(a^T)$ (see Figure 1(a) - top). By grouping the elements of $\vec{w}$ according to the sequence of exponents $1, 2, 2, 1, 3, 1, 4, 1, 3, 2, 1, 4, 2, 1, 5, 2, 1, \dots$, we obtain the word sequence $\vec{v}$ (see Figure 1(a) - bottom). The word sequence $\vec{w}$ can be viewed as the concatenation of subsequences $(a^1, \dots, a^j)$, for all $j \geq 1$. To generate $\vec{v}$ from $\vec{w}$, we group the elements of any subsequence $a^1, \dots, a^j$ as follows (the case for $j = 20$ is depicted in Figure 1(b)). We start from the last position j and

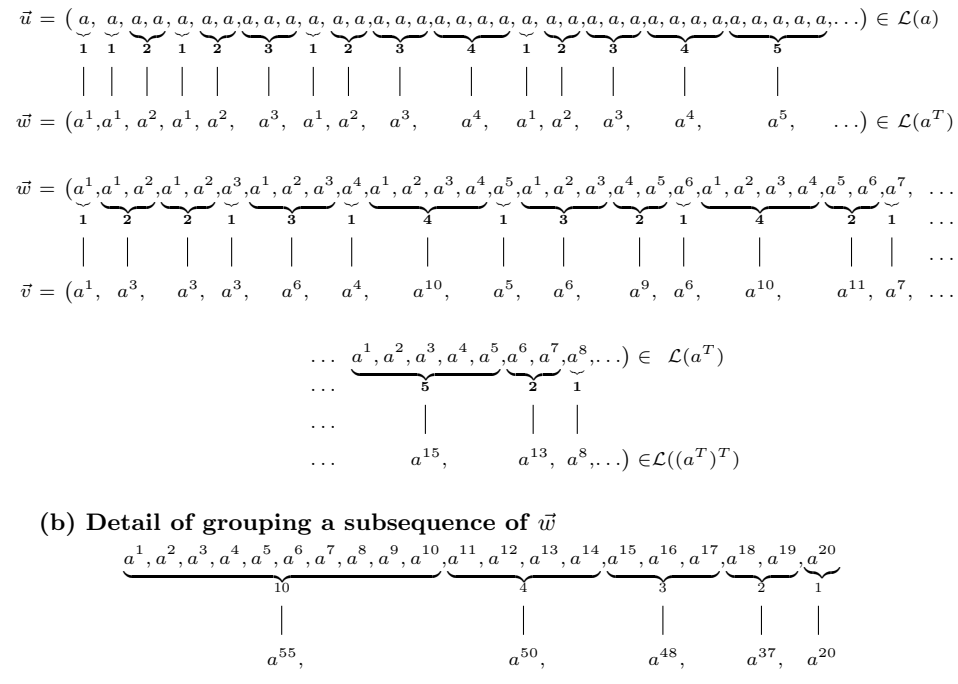


Figure 1: A counter-example to the equality $(a^T)^T = a^T$.

proceed backwardly. The last element a^j is a singleton, unless $\sum_{i=1}^{j-1} i < j$ (in which case the whole sequence is grouped altogether). Then, the two elements $a^{j-2}a^{j-1}$ are grouped together, unless $\sum_{i=1}^{j-3} i < j$ (in which case the sequence $a^1, \ldots, a^{j-1}$ is grouped together). Next, the three elements $a^{j-5}a^{j-4}a^{j-3}$ are grouped together, unless $\sum_{i=1}^{j-6} i < j$ (in which case the sequence $a^1, \ldots, a^{j-3}$ is grouped together), and so on.

Such a construction can be formalized as follows. For all $j \geq 1$, let k_j (we omit the subscript when clear from the context) be the largest number such that $\sum_{i=1}^{j-\sum_{h=1}^{k} h} i \geq j$. Note that this implies $j > \sum_{h=1}^{k} h$ (otherwise $\sum_{i=1}^{j-\sum_{h=1}^{k} h} i = 0 < j$). The grouping procedure executes the following two steps: (i) for all $h \in \{1, \ldots, k\}$, $a^{j-(\sum_{i=1}^{h} i)+1}, \ldots, a^{j-\sum_{i=1}^{h-1} i}$ are grouped together, and (ii) $a^1, \ldots, a^{j-\sum_{i=1}^{k} i}$ are grouped together. Since $\vec{w} \in \mathcal{L}(a^T)$ and the above-described sequence contains infinitely many exponents occurring infinitely often, it holds that $\vec{v} \in \mathcal{L}((a^T)^T)$.

To prove that $\vec{v} \in a^S$, and thus $\vec{v} \notin a^T$, it suffices to show that:

$$\left(j - \left(\sum_{i=1}^{h} i\right) + 1\right) + \ldots + \left(j - \sum_{i=1}^{h-1} i\right) \geq j$$

for all $h \in \{1, \ldots, k\}$. Since j is strictly increasing, indeed, it immediately follows that the sequence of a in $\vec{v}$ is strongly unbounded.

Let us show that, for all $h \in \{1, \ldots, k\}$, the stronger inequality $\left(j - \left(\sum_{i=1}^{h} i\right) +\right.$

$1) + \ldots + \left(j - \sum_{i=1}^{h-1} i\right) > j$ holds. First, we observe that $\left(j - (\sum_{i=1}^{h} i) + 1\right) + \ldots + \left(j - \sum_{i=1}^{h-1} i\right) = \sum_{s=-(\sum_{i=1}^{h} i)+1}^{-\sum_{i=1}^{h-1} i} (j+s) = \sum_{s=-(\sum_{i=1}^{h} i)+1}^{-(\sum_{i=1}^{h-1} i)-1} (j+s) + (j - \sum_{i=1}^{h-1} i)$. As $\sum_{i=1}^{h} i < j$ for all $h \in \{1, \ldots, k\}$, it holds that $j - (\sum_{i=1}^{h} i) + 1$ (i.e., the value of $j+s$ when s assume the lowest possible value in the range of the summation $\sum_{s=-(\sum_{i=1}^{h} i)+1}^{-(\sum_{i=1}^{h-1} i)-1} (j+s)$) is greater than 1. Consequently, as s increases, $j+s$ becomes greater than 2, 3, and so on, that is, $j-(\sum_{i=1}^{h} i)+2 > 2$, $j - (\sum_{i=1}^{h} i) + 3 > 3$, and so on. Since s ranges over $h-1$ values ($(-(\sum_{i=1}^{h-1} i) - 1) - (-(\sum_{i=1}^{h} i) + 1) + 1 = h - 1$), it holds that $\sum_{s=-(\sum_{i=1}^{h} i)+1}^{-(\sum_{i=1}^{h-1} i)-1} (j+s) > \sum_{i=1}^{h-1} i$, and thus $\sum_{s=-(\sum_{i=1}^{h} i)+1}^{-(\sum_{i=1}^{h-1} i)-1} (j+s) + (j - \sum_{i=1}^{h-1} i) > \sum_{i=1}^{h-1} i + (j - \sum_{i=1}^{h-1} i) = j$.

Therefore, we have that $\vec{v} \in (a^T)^T \setminus a^T$, hence the thesis. □

We conclude the section by formally proving that $(a^T b)^\omega + (a^* b)^* a^\omega (= \overline{L})$ is the complement of $(a^B b + a^S b)^\omega (= L)$, that is, $\overline{L} = \{a, b\}^\omega \setminus L$.

First, we observe that $(a^* b)^* a^\omega$ is the language of those ω-words over $\{a, b\}$ that feature only finitely many occurrences of b. Since L only contains words featuring infinitely many occurrences of b, it holds that $(a^* b)^* a^\omega \subseteq \{a, b\}^\omega \setminus L$.

We now introduce a *fair* version of the shuffle operator, denoted by $\oplus$, which, intuitively, switches infinitely often between the two argument word sequences.

Definition 2 (fair shuffle)**.** *Let e_1 and e_2 be BST-regular expressions. The* fair shuffle *operator, denoted by $\oplus$, is defined as follows:*

$$\mathcal{L}(e_1 \oplus e_2) = \{\vec{w} \mid \vec{w} = g\text{-}shuffle(\vec{u}, \vec{v}), \text{ for some } \vec{u}, \vec{v} \in \mathcal{L}(e_1) \cup \mathcal{L}(e_2) \text{ and some alternating selection function } g\}.$$

Let $\uplus$ be the disjoint union. We have that:

$$L = (a^B b)^\omega \uplus (a^S b)^\omega \uplus (a^B b \oplus a^S b)^\omega. \tag{1}$$

The set $(a^B b)^\omega$ contains exactly those ω-words in $(a^B b + a^S b)^\omega$ that from a given position on consist of $a^B b$ blocks only. By Proposition 6 (prefix independence), this amounts to say that the whole ω-word is made up of $a^B b$ blocks only. The set $(a^S b)^\omega$ contains exactly those ω-words in $(a^B b + a^S b)^\omega$ that from a given position on consist of $a^S b$ blocks only. Again, thanks to prefix independence, this amounts to say that the whole ω-word is made up of $a^S b$ blocks only. Finally, the set $(a^B b \oplus a^S b)^\omega$ contains exactly those ω-words in $(a^B b + a^S b)^\omega$ that feature infinitely many occurrences of both $a^B b$ and $a^S b$ blocks.

Proposition 9. *Let $t \in \{a, b\}^\omega \setminus L$ be a word featuring an infinite number of occurrences of b. Then, $t \in (a^T b)^\omega$.*

Proof. Since $t \in \{a, b\}^\omega$ and t contains an infinite number of b's, we know that $t \in (a^* b)^\omega$. Assume, towards a contradiction, that $t \notin (a^T b)^\omega$, that is, there are only finitely many exponents of a that occur infinitely often (possibly no one). We distinguish three cases. If no exponent occurs infinitely often, then $t \in (a^S b)^\omega \subseteq L$ (contradiction). If some exponent occurs infinitely often and

finitely many exponents occur finitely often, then $t \in (a^B b)^\omega \subseteq L$ (contradiction). Finally, if some exponent occurs infinitely often and infinitely many exponents occur finitely often, then the (sub-)sequence of a^*b blocks corresponding to exponents occurring infinitely often, say t_1, belongs to $a^B b$ and the (sub-) sequence of blocks discarded by the above sequence, say t_2, belongs to $a^S b$. It immediately follows that $t \in ((a^B b) \oplus (a^S b))^\omega \subseteq L$ (contradiction). □

3.3. A stronger variant of the T-constructor

A stronger variant of $(.)^T$, called $(.)^{T_s}$, that forces ω-words to feature infinitely many exponents, *all of them* occurring infinitely often, has also been proposed in the literature [2].

T_s-regular expressions differ from T-regular ones for the replacement of the semantic clause for $(.)^T$ by the following one:

- $\mathcal{L}(e^{T_s}) = \{(u_{f(0)}u_2 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots) \mid \vec{u} \in \mathcal{L}(e)$ and $f : \mathbb{N} \to \mathbb{N}_{>0}$ is a nondecreasing function, with $f(0) = 1$, such that $\exists^\omega n \in \mathbb{N}\, \forall k \in \mathbb{N}\, \exists i > k.(f(i+1) - f(i) = n)$ and $\forall n \in \mathbb{N}\, \forall k \in \mathbb{N}.(f(k+1) - f(k) = n \to \exists i > k.f(i+1) - f(i) = n)\}$

where e is a word sequence.

ωT_s-regular expressions are defined exactly the same way as ωT-regular ones.

Some results obtained for $(.)^T$ can be transferred to $(.)^{T_s}$: non-idempotency holds for $(.)^{T_s}$ as well, and a suitable expression making use of $(.)^{T_s}$, instead of $(.)^T$, can be given that captures the complement of $L = (a^B b + a^S b)^\omega$.

There are, however, some significant differences between $(.)^T$ and $(.)^{T_s}$.

First, $(.)^{T_s}$ does not satisfy the property of prefix independence. Let $\vec{u} = (u_1, u_2, \dots)$ and $\vec{v} = (u_h, u_{h+1}, \dots)$ be two word sequences such that $\vec{v}$ is the infinite suffix of $\vec{u}$ starting at position h. For any T_s-regular expression, it holds that if $\vec{u}$ belongs to the language, then $\vec{v}$ belongs to it as well, but not (necessarily) vice versa. Moreover, it can be easily checked that the equality $e^* = e^B + e^S + e^T$, stated by Proposition 7, does not hold anymore if we replace e^T by e^{T_s}. Last but not least, there is no immediate way to generalize the embedding (given in Section 5) of ωT-regular languages into S1S+U to ωT_s-regular ones (as a matter of fact, we strongly believe such a variant not to be definable in S1S+U).

4. ωT-regular languages and counter-check automata (CCA)

In this section, we introduce a new class of automata, which we name counter-check automata (CCA), we prove that their emptiness problem is decidable, and we provide a translation of ωT-regular expressions into CCA. In the next section, we provide an encoding of ωT-regular expressions into S1S+U.

4.1. Counter-check automata (CCA)

To start with, we define CCA and we show that their emptiness problem is decidable in PTIME. Later, we will show that they are expressive enough to encode ωT-regular expressions. It is worth pointing out that CCA closely resemble B- and S-automata [8, 9], the main difference being that they allow for *ϵ-transitions*, that is, transitions that do not consume word symbols. It is not clear whether or not such transitions can be avoided, as it happens with B- and S-automata (as a matter of fact, this is the main obstacle towards an expressive completeness result for CCA and ωT-regular languages). Moreover, unlike B- and S-automata, which allow for one operation per counter at each transition, CCA allow for at most one counter operation per transition. Thanks to the presence of ϵ-transitions, this does not cause any loss in expressive power.

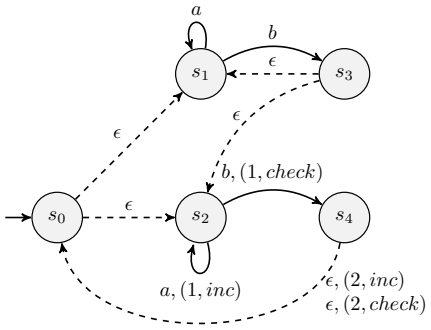


Figure 2: A CCA with 2 counters for the language $((a^*b)^*a^Tb)^\omega$.

A *CCA* is an automaton equipped with a fixed number of counters. A transition can possibly *increment* or *reset* one of them (or do nothing). We refer to reset operations as *check* operations to emphasize the fact that computations keep track of the evolution of counter values. CCA acceptance condition depends on the sequences of *checked values*, that is, the values when a check operation is performed, for all counters. An example of CCA is given in Figure 2.

Definition 3 (CCA)**.** *A* counter-check automaton *(*CCA*) 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,* $N \in \mathbb{N}_{>0}$ *is the number of counters, and* $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S \times (\{1, \ldots, N\} \times \{no\_op, inc, check\})$ *is a transition relation, subject to the constraint: if* $(s, \sigma, s', (k, op)) \in \Delta$ *and* $op = no\_op$*, then* $k = 1$*.*

A *configuration* of a CCA $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ is a pair $(s, \mathbf{v})$, where $s \in S$ and $\mathbf{v} \in \mathbb{N}^N$ ($\mathbf{v}$ is called *counter vector*). For $\mathbf{v} \in \mathbb{N}^N$ and $i \in \{1, \ldots, N\}$, let $\mathbf{v}[i]$ be the i-th component of $\mathbf{v}$, that is, the value of the i-th counter.

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

- if $op = no\_op$, then $\mathbf{v}'[k] = \mathbf{v}[k]$;
- if $op = inc$, then $\mathbf{v}'[k] = \mathbf{v}[k] + 1$;
- if $op = check$, then $\mathbf{v}'[k] = 0$.

In such a case, we say that $(s, \mathbf{v}) \rightarrow_{\mathcal{A}}^{\sigma} (s', \mathbf{v}')$ via δ. Let $\rightarrow_{\mathcal{A}}^{*}$ be the reflexive and transitive closure of $\rightarrow_{\mathcal{A}}^{\sigma}$ (where we abstract away symbols in $\Sigma \cup \{\epsilon\}$).

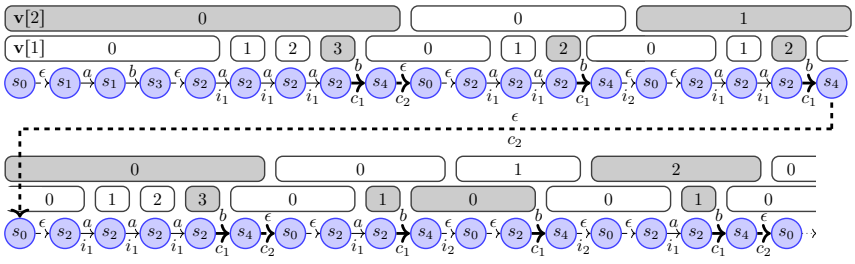


Figure 3: A prefix of a computation of the automaton in Figure 2. A configuration is characterised by a circle (state) and the rounded-corner rectangles above it (counter vector). $\mathbf{v}[i]$ is a counter vector component. Checked values for counters are highlighted in gray, with the corresponding transitions being written in boldface.

The *initial configuration* of $\mathcal{A}$ is the pair $(s_0, \mathbf{v}_0)$, where $\mathbf{v}_0[k] = 0$ for each $k \in \{1, \ldots, N\}$. A *computation* of $\mathcal{A}$ is an infinite sequence of configurations $\mathcal{C} = (s_0, \mathbf{v}_0)(s_1, \mathbf{v}_1)\ldots$, where, for all $i \in \mathbb{N}$, $(s_i, \mathbf{v}_i) \rightarrow_{\mathcal{A}}^{\sigma_i} (s_{i+1}, \mathbf{v}_{i+1})$ for some $\sigma_i \in \Sigma \cup \{\epsilon\}$. An example is given in Figure 3. For a computation $\mathcal{C} = (s_0, \mathbf{v}_0)(s_1, \mathbf{v}_1)\ldots$ of $\mathcal{A}$, we let $check^{\infty}_{\mathcal{C},k}$ $(k \in \{1, \ldots, N\})$ denote the set $\{n \in \mathbb{N} \mid \forall h \exists i > h$ such that $\mathbf{v}_i[k] = n$ and $\mathbf{v}_{i+1}[k] = 0\}$, that is, $check^{\infty}_{\mathcal{C},k}$ is the set of values of the k-th counter that are checked infinitely often along $\mathcal{C}$. A *good computation* of $\mathcal{A}$ is a computation $\mathcal{C}$ of $\mathcal{A}$ such that $|check^{\infty}_{\mathcal{C},k}| = +\infty$ for all $k \in \{1, \ldots, N\}$. Given two configurations $(s_i, \mathbf{v}_i)$ and $(s_j, \mathbf{v}_j)$ in $\mathcal{C}$, with $i \leq j$, we say that $(s_j, \mathbf{v}_j)$ is *ϵ-reachable* from $(s_i, \mathbf{v}_i)$, written $(s_i, \mathbf{v}_i) \rightarrow_{\mathcal{A}}^{*\epsilon} (s_j, \mathbf{v}_j)$, if $(s_{j'-1}, \mathbf{v}_{j'-1}) \rightarrow_{\mathcal{A}}^{\epsilon} (s_{j'}, \mathbf{v}_{j'})$ for all $j' \in \{i+1, \ldots, j\}$. Moreover, we use $(s_i, \mathbf{v}_i) \rightarrow_{\mathcal{A}}^{*\epsilon} (s_\infty, \mathbf{v}_\infty)$ as an abbreviation for $(s_i, \mathbf{v}_i) \rightarrow_{\mathcal{A}}^{*\epsilon} (s_j, \mathbf{v}_j)$ for all $j \geq i$.

A *run* π of $\mathcal{A}$ on w is a good computation $\pi = (s_0, \mathbf{v}_0)(s_1, \mathbf{v}_1)\ldots$ for which there is an increasing function $f : \mathbb{N}_{>0} \rightarrow \mathbb{N} \cup \{\infty\}$ (i.e., $f(i) < f(i+1)$ unless $f(i+1) = \infty$), called *trace of w in π with respect to $\mathcal{A}$*, such that:

- $(s_0, \mathbf{v}_0) \rightarrow_{\mathcal{A}}^{*\epsilon} (s_{f(1)}, \mathbf{v}_{f(1)})$, and
- for all $i \geq 1$, if $f(i) \neq \infty$ then $(s_{f(i)}, \mathbf{v}_{f(i)}) \rightarrow_{\mathcal{A}}^{w[i]} (s_{f(i)+1}, \mathbf{v}_{f(i)+1})$ and $(s_{f(i)+1}, \mathbf{v}_{f(i)+1}) \rightarrow_{\mathcal{A}}^{*\epsilon} (s_{f(i+1)}, \mathbf{v}_{f(i+1)})$.

A run π of $\mathcal{A}$ on w is *accepting* if and only if $f(i) \neq \infty$ for all i, where f is the trace of w in π with respect to $\mathcal{A}$. Notice that, while the notion of run is defined also for finite words, accepting runs concern ω-words only. An ω-word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if and only if there is an accepting run of $\mathcal{A}$ on w. We denote by $\mathcal{L}(\mathcal{A})$ the set of all ω-words in Σ^ω that are accepted by $\mathcal{A}$, and we say that $\mathcal{A}$ *accepts* the language $\mathcal{L}(\mathcal{A})$. Figure 2 depicts a CCA with two counters $(N = 2)$ accepting the language $((a^*b)^*a^Tb)^\omega$. (Note that an automaton for the same language with one counter only can be devised as well.)

4.2. Decidability of the emptiness problem for CCA

We now prove that the emptiness problem for CCA is decidable in PTIME. The proof consists of 3 steps: (i) we replace general CCA by simple ones; (ii)

we prove that their emptiness can be decided by checking the existence of finite witnesses of accepting runs; (iii) we show that the latter can be verified by checking for emptiness a suitable NFA.

Simple CCA. To begin with, we define the notion of *simple CCA*.

Definition 4 (simple CCA)**.** *A CCA* $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ *is* simple *if and only if for each* $s \in S$*, either* $|\{(s, \sigma, s', (k, op)) \mid (s, \sigma, s', (k, op)) \in \Delta \text{ for some } \sigma, s', k, op\}| = 1$ *or* $op = no\_op$*,* $k = 1$*, and* $\sigma = \epsilon$ *for all* $(s, \sigma, s', (k, op)) \in \Delta$*.*

Basically, the states of a simple CCA can be partitioned in two classes: those in which it can fire exactly one action and those in which it makes a nondeterministic choice. Moreover, for all pairs of configurations $(s, \mathbf{v}), (s', \mathbf{v}')$ with $(s, \mathbf{v}) \rightarrow_{\mathcal{A}}^{\sigma} (s', \mathbf{v}')$, the transition $\delta \in \Delta$ that has been fired in $(s, \mathbf{v})$ is uniquely determined by s and s'. By exploiting *ϵ-transitions*, i.e., transitions of the form $(s, \epsilon, s', (k, op))$, and by adding a suitable number of states, it can be easily shown that every CCA $\mathcal{A}$ may be turned into a simple one $\mathcal{A}'$, whose size is polynomial in the size of $\mathcal{A}$, such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. W.l.o.g., in the rest of the section we restrict our attention to simple CCA.

The set of states of a CCA 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 CCA. A *prefix computation* of $\mathcal{A}$ is a finite prefix of a computation of $\mathcal{A}$. Formally, it is a finite sequence $\mathcal{P} = (s_0, \mathbf{v}_0) \dots (s_n, \mathbf{v}_n)$ such that, for all $i \in \{0, \dots, n-1\}$, $(s_i, \mathbf{v}_i) \rightarrow_{\mathcal{A}}^{\sigma_i} (s_{i+1}, \mathbf{v}_{i+1})$, for some $\sigma_i \in \Sigma \cup \{\epsilon\}$. We denote by $Prefixes_{\mathcal{A}}$ the sets of all prefix computations of $\mathcal{A}$. For every $\mathcal{P} = (s_0, \mathbf{v}_0) \dots (s_n, \mathbf{v}_n) \in Prefixes_{\mathcal{A}}$, it holds that if $(s_n, \mathbf{v}_n) \rightarrow_{\mathcal{A}}^{\sigma} (s, \mathbf{v})$, for some $s \in S$, some counter vector $\mathbf{v}$, and some $\sigma \in \Sigma \cup \{\epsilon\}$, then $\mathbf{v}$ is uniquely determined by s_n and $\mathbf{v}_n$, that is, there is no $\mathbf{v}' \neq \mathbf{v}$ such that $(s_n, \mathbf{v}_n) \rightarrow_{\mathcal{A}}^{\sigma'} (s, \mathbf{v}')$, for any s and σ'.

Finite witnesses of accepting runs. We show now how to decide CCA emptiness by making use of the notion of accepting witness for a CCA.

Definition 5 (Accepting witness)**.** *Let* $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ *be a CCA. A prefix computation* $\mathcal{P} = (s_0, \mathbf{v}_0) \dots (s_n, \mathbf{v}_n) \in Prefixes_{\mathcal{A}}$ *is an* accepting witness *for* $\mathcal{A}$ *if and only if there are* $2N + 2$ *indexes* $begin < b_1 < e_1 < \dots < b_N < e_N < end$ *such that* $0 \leq begin, end \leq n$*, and the following conditions hold:*

1. *a non-ϵ-transition can be fired from* s_{begin}*;*
2. $s_{begin} = s_{end}$ *and, for each* $k \in \{1, \dots, N\}$*,* $s_{b_k} = s_{e_k}$*,* s_{b_k} *is an* inc_k *state, and* s_j *is not a* $check_k$ *state for any* j *with* $b_k \leq j \leq e_k$*;*

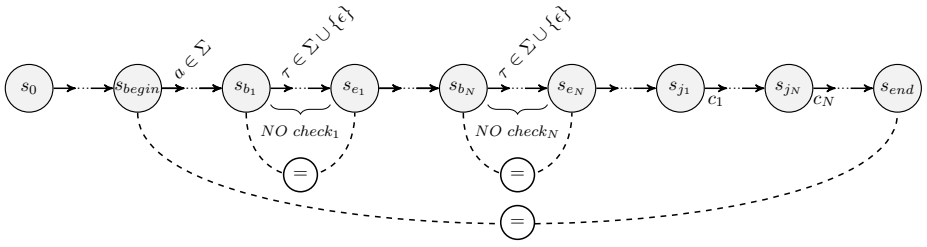


Figure 4: An accepting witness for a CCA.

3. *for each* $k \in \{1, \ldots, N\}$*, there is* j_k*, with* $e_N < j_k < end$*, such that* s_{j_k} *is a* $check_k$ *state.*

As usual, an accepting witness for $\mathcal{A}$ can be seen as a finite representation of an accepting run of some ω-word on $\mathcal{A}$ (see Figure 4). Thus, deciding whether a CCA $\mathcal{A}$ accepts a nonempty language amounts to searching $Prefixes_{\mathcal{A}}$ for accepting witnesses as formally stated by the next lemma, whose proof is straightforward and thus omitted.

Lemma 1. *Let* $\mathcal{A}$ *be a CCA. Then,* $\mathcal{L}(\mathcal{A}) \neq \emptyset$ *if and only if* $Prefixes_{\mathcal{A}}$ *contains an accepting witness for* $\mathcal{A}$*.*

From CCA to NFA. Thanks to Lemma 1, deciding the (non)emptiness problem for a CCA $\mathcal{A}$ amounts to searching $Prefixes_{\mathcal{A}}$ for an accepting witness. Since we restricted ourselves to simple CCA, we can safely identify elements of $Prefixes_{\mathcal{A}}$ with their sequence of states and thus, by slightly abusing the notation, we write, e.g., $s_0 s_1 \ldots s_n \in Prefixes_{\mathcal{A}}$ for $(s_0, \mathbf{v}_0) \ldots (s_n, \mathbf{v}_n) \in Prefixes_{\mathcal{A}}$. Given a CCA $\mathcal{A}$, let $\mathcal{L}_{\mathsf{w}}(\mathcal{A})$ be the language of finite words over the alphabet S (the set of states of $\mathcal{A}$) that are accepting witnesses for $\mathcal{A}$. It is easy to see that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ if and only if $\mathcal{L}_{\mathsf{w}}(\mathcal{A}) \neq \emptyset$. In what follows, for a CCA $\mathcal{A}$, we build a nondeterministic finite automata (NFA) whose language is exactly $\mathcal{L}_{\mathsf{w}}(\mathcal{A})$. Since the (non)emptiness problem for NFA is decidable, so is the one for CCA.

In what follows, w.l.o.g., we restrict our attention to accepting witnesses for which the set of indexes required by item 3 of Definition 5 is ordered. More precisely (we borrow the notation from Definition 5), we assume that there are N indexes $c_1 < \ldots < c_N$, with $e_N < c_1$ and $c_N < end$, 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 5). Given a CCA $\mathcal{A}$, it is easy to check that $Prefixes_{\mathcal{A}}$ contains an accepting witness, as specified by Definition 5, if and only if it contains one satisfying the additional ordering property above. Thus, Lemma 1 holds with respect to the new definition of accepting witness as well.

Given a CCA $\mathcal{A}$, we apply the following steps to build an NFA $\mathcal{N}$ such that $\mathcal{L}(\mathcal{N}) = \mathcal{L}_{\mathsf{w}}(\mathcal{A})$: *(i)* we build an NFA $\mathcal{N}'$ accepting finite words over the set of states of $\mathcal{A}$ that are potential accepting witnesses, i.e., they satisfy conditions 1- 3 of Definition 5 but they might not be prefix computations; in

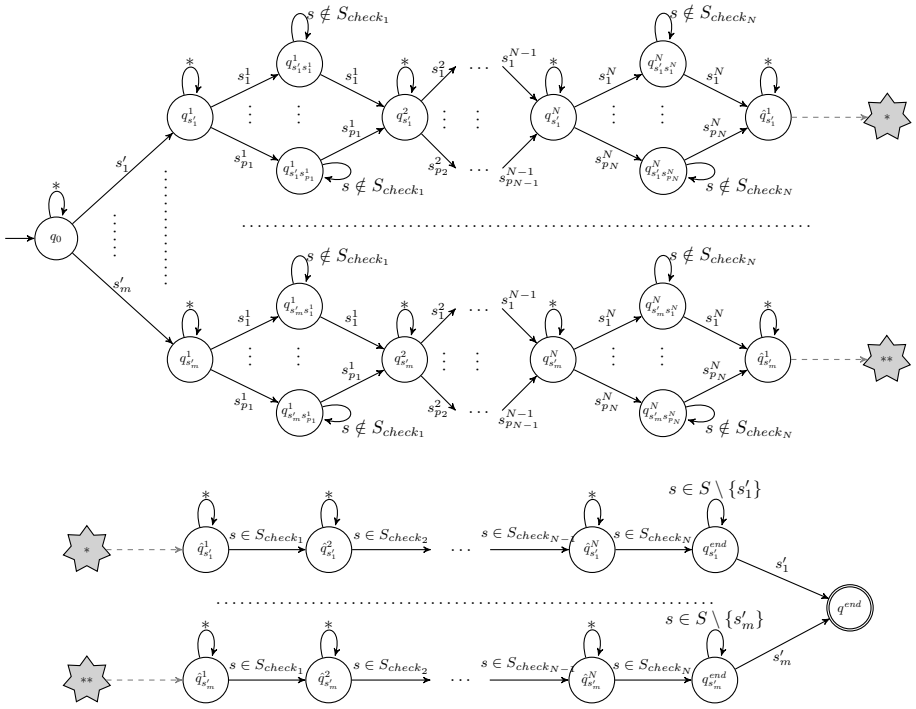


Figure 5: A graphical account of the automaton $\mathcal{N}'$, where $S_{non\text{-}\epsilon} = \{s'_1, s'_2, \ldots, s'_m\}$ and $S_{inc_k} = \{s^k_1, s^k_2, \ldots, s^k_{p_k}\}$, with $k \in \{1, \ldots, N\}$.

other words, such an automaton might as well accept words not belonging to $\mathit{Prefixes}_{\mathcal{A}}$; *(ii)* since $\mathit{Prefixes}_{\mathcal{A}}$ is a regular language, thanks to closure properties of NFA, there exists an NFA $\mathcal{N}$ whose language is $\mathcal{L}(\mathcal{N}') \cap \mathit{Prefixes}_{\mathcal{A}} = \mathcal{L}_{\mathsf{w}}(\mathcal{A})$.

Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ be a CCA. We define $\mathcal{N}' = \langle Q, \Sigma_{\mathcal{N}'}, \delta, q_0, F \rangle$ as follows: $\Sigma_{\mathcal{N}'} = S$, $F = \{q^{end}\}$, and $Q = \{q_0, q^{end}\} \cup \{q^{end}_{s'} \mid s' \in S_{non\text{-}\epsilon}\} \cup \bigcup_{k=1}^{N} \{q^k_{s'}, \hat{q}^k_{s'} \mid s' \in S_{non\text{-}\epsilon}\} \cup \bigcup_{k=1}^{N} \{q^k_{s's''} \mid s' \in S_{non\text{-}\epsilon}, s'' \in S_{inc_k}\}$, where $S_{non\text{-}\epsilon}$ is the set of states of S from which a non-ϵ-transition can be fired, and, for $k \in \{1, \ldots, N\}$, S_{inc_k} is the set of inc_k states in S. A graphical account of the behavior of the automaton $\mathcal{N}'$, in particular of its transition relation δ, is given in Figure 5 ($*$ stands for any symbol in S). It can be summarized as follows:

1. it nondeterministically guesses index *begin* when a symbol $s' \in S_{non\text{-}\epsilon}$ is read; the next state $q^1_{s'}$ that it reaches stores information about the state s' of $\mathcal{A}$ being read to check, at a later stage (when index *end* is guessed), that $s_{begin} = s' = s_{end}$;

2. similarly, for each $k \in \{1, \ldots, N\}$, it nondeterministically guesses indexes b_k and e_k, when a symbol s^k corresponding to an inc_k state of $\mathcal{A}$ is read; once again, information about the state s^k of $\mathcal{A}$ being read is stored in the next state $q^k_{s's^k}$ that it reaches, in order to check that the same state s^k is

read when e_k is guessed ($s_{b_k} = s^k = s_{e_k}$); moreover, it forces the absence of a $check_k$ state in between indexes b_k and e_k;

3. it verifies the existence, after e_N, of $check_k$ states, for $k \in \{1, \ldots, N\}$, in the desired order;

4. it waits for the input symbol s', that is, the same symbol read when *begin* was guessed; when such a symbol is read, it enters the final state q^{end}.

Let S_{check_k} be the set of $check_k$ states in S, and let $S_{non\text{-}\epsilon}$ and S_{inc_k}, with $k \in \{1, \ldots, N\}$, be the above-defined sets. We formally define δ as follows:

$$
\begin{aligned}
\delta = {} & \{(q_0, s, q_0) \mid s \in S\} \cup \{(q_0, s', q^1_{s'}) \mid s' \in S_{non\text{-}\epsilon}\} \\
& \cup \textstyle\bigcup_{k=1}^{N} \{(q^k_{s'}, s, q^k_{s'}) \mid s' \in S_{non\text{-}\epsilon}, s \in S\} \\
& \cup \textstyle\bigcup_{k=1}^{N} \{(q^k_{s'}, s^k, q^k_{s's^k}) \mid s' \in S_{non\text{-}\epsilon}, s^k \in S_{inc_k}\} \\
& \cup \textstyle\bigcup_{k=1}^{N} \{(q^k_{s's^k}, s, q^k_{s's^k}) \mid s' \in S_{non\text{-}\epsilon}, s^k \in S_{inc_k}, s \in S \setminus S_{check_k}\} \\
& \cup \textstyle\bigcup_{k=1}^{N-1} \{(q^k_{s's^k}, s^k, q^{k+1}_{s'}) \mid s' \in S_{non\text{-}\epsilon}, s^k \in S_{inc_k}\} \\
& \cup \{(q^N_{s's^N}, s^N, \hat{q}^1_{s'}) \mid s' \in S_{non\text{-}\epsilon}, s^N \in S_{inc_N}\} \\
& \cup \textstyle\bigcup_{k=1}^{N} \{(\hat{q}^k_{s'}, s, \hat{q}^k_{s'}) \mid s' \in S_{non\text{-}\epsilon}, s \in S\} \\
& \cup \textstyle\bigcup_{k=1}^{N-1} \{(\hat{q}^k_{s'}, s^k, \hat{q}^{k+1}_{s'}) \mid s' \in S_{non\text{-}\epsilon}, s^k \in S_{check_k}\} \\
& \cup \{(\hat{q}^N_{s'}, s^N, q^{end}_{s'}) \mid s' \in S_{non\text{-}\epsilon}, s^N \in S_{check_N}\} \\
& \cup \{(q^{end}_{s'}, s, q^{end}_{s'}) \mid s' \in S_{non\text{-}\epsilon}, s \in S \setminus \{s'\}\} \\
& \cup \{(q^{end}_{s'}, s', q^{end}) \mid s' \in S_{non\text{-}\epsilon}\}.
\end{aligned}
$$

Since the size of $\mathcal{N}'$ is polynomial in the size of $\mathcal{A}$ ($|Q| \leq 2 + 2 \cdot N \cdot |S| + N \cdot |S|^2 + |S|$), we have a polynomial reduction from the emptiness problem for CCA to the one for NFA.

Theorem 2. *The emptiness problem for CCA is decidable in PTIME.*

4.3. A translation of ωT-regular expressions into CCA

We now show how to translate ωT-regular expressions into CCA. To this end, we apply some preliminary rewriting to the input expression. In particular, we get rid of $+$ in favor of $\oplus$ (see Definition 2), and we replace the Kleene star $(.)^*$ (0 or more iterations of the the argument expression) by the combination of the operator for the empty string and the usual constructor $(.)^+$ (1 or more iterations). While we only need these results to hold for ωT-regular languages, we state and prove them within the more general setting of ωBST-regular languages.

The next proposition generalizes the notion of prefix independence to ωBST-regular expressions (recall that, for a BST-regular expression e, $reg(e)$ denotes the regular expression obtained from e by replacing occurrences of B-, S-, and T-constructors with the Kleene star).

Proposition 10 (Prefix independence for ωBST-regular expressions)**.** *Let e^ω be an ωBST-regular expression and $u \in \mathcal{L}((reg(e))^*)$. Then, for all v:*

(a) if $v \in \mathcal{L}(e^\omega)$, then $u \cdot v \in \mathcal{L}(e^\omega)$;

(b) if $u \cdot v \in \mathcal{L}(e^\omega)$, then there are infinitely many suffixes of $u \cdot v$ belonging to $\mathcal{L}(e^\omega)$.

Proof. Item (a) follows from prefix independence of BST-regular expressions (Proposition 6). Item (b) follows from suffix closure of BST-regular expressions, which in turn follows from prefix independence (again Proposition 6). □

In order to remove the + operator from the input expression, we introduce two auxiliary operators:

- $\mathcal{L}(e_1 \oplus\!\!\!\!- e_2) = \{\vec{w} \mid \vec{w} = g\text{-}\mathit{shuffle}(\vec{u}, \vec{v})$, for some $\vec{u}, \vec{v} \in \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$ and some 1-stable selection function $g\}$;
- $\mathcal{L}(e_1 \ominus e_2) = \{\vec{w} \mid \vec{w} = g\text{-}\mathit{shuffle}(\vec{u}, \vec{v})$, for some $\vec{u}, \vec{v} \in \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$ and some 2-stable selection function $g\}$;

Let e be a BST-regular expression. We define $\mathit{elim}_+(e)$ as the set of all those expressions obtained from e by replacing each occurrence of + by one among $\oplus$, $\oplus\!\!\!\!-$, and $\ominus$ e.g., $\mathit{elim}_+(a+b+c) = \{a \ominus b \ominus c, a \ominus b \oplus c, a \ominus b \oplus\!\!\!\!- c, a \oplus b \ominus c, a \oplus b \oplus c, a \oplus b \oplus\!\!\!\!- c, a \oplus\!\!\!\!- b \ominus c, a \oplus\!\!\!\!- b \oplus c, a \oplus\!\!\!\!- b \oplus\!\!\!\!- c\}$. Clearly, $|\mathit{elim}_+(e)| = 3^n$, where n is the number of occurrences of + in e. Moreover, for any $e' \in \mathit{elim}_+(e)$, let $\mathit{elim}_{\oplus\!\!\!\!-,\ominus}(e')$ be the expression obtained from e' by replacing each sub-expression of the form $e_1 \oplus\!\!\!\!- e_2$ (resp., $e_1 \ominus e_2$) by e_1 (resp., e_2).

Lemma 2. *Let e be a BST-regular expression. It holds that:*

$$\mathcal{L}(e) = \bigcup_{e' \in \mathit{elim}_+(e)} \mathcal{L}(e').$$

Proof. By structural induction on e. □

Corollary 1. *Let e be a BST-regular expression. It holds that:*[5]

$$\mathcal{L}(e^\omega) = \mathcal{L}\left(\sum_{e' \in \mathit{elim}_+(e)} (e')^\omega\right).$$

Lemma 3. *Let e be a BST-regular expression. It holds that:*

$$\mathcal{L}\left(\sum_{e' \in \mathit{elim}_+(e)} (e')^\omega\right) = \mathcal{L}\left((\mathit{reg}(e))^* \cdot \sum_{e' \in \mathit{elim}_+(e)} (\mathit{elim}_{\oplus\!\!\!\!-,\ominus}(e'))^\omega\right).$$

Proof. We first prove the right-to-left inclusion, and then the opposite one.

Right-to-left inclusion. Let $v \in \mathcal{L}\left((\mathit{reg}(e))^* \cdot \sum_{e' \in \mathit{elim}_+(e)} (\mathit{elim}_{\oplus\!\!\!\!-,\ominus}(e'))^\omega\right)$, i.e., $v = u \cdot w$ for $u \in \mathcal{L}((\mathit{reg}(e))^*)$ and $w \in \mathcal{L}((\mathit{elim}_{\oplus\!\!\!\!-,\ominus}(e'))^\omega)$ for some $e' \in \mathit{elim}_+(e)$. Since $\mathcal{L}(\mathit{elim}_{\oplus\!\!\!\!-,\ominus}(e')) \subseteq \mathcal{L}(e')$ for all $e' \in \mathit{elim}_+(e)$, we have that $w \in \mathcal{L}\left(\sum_{e' \in \mathit{elim}_+(e)} (e')^\omega\right) =_{\text{(by Corollary 1)}} \mathcal{L}(e^\omega)$. Then, by Proposition 10(a), it holds that $v = u \cdot w \in \mathcal{L}(e^\omega) = \mathcal{L}\left(\sum_{e' \in \mathit{elim}_+(e)} (e')^\omega\right)$.

Left-to-right inclusion. Let $v \in \mathcal{L}\left(\sum_{e' \in \mathit{elim}_+(e)} (e')^\omega\right)$, i.e., $v = v_1 v_2 \ldots$, with $\vec{v} = (v_1, v_2, \ldots) \in \mathcal{L}(e')$, for some $e' \in \mathit{elim}_+(e)$. Let *selection-functions*

[5] In what follows, we overload the symbol Σ (already used to denote the alphabet) by using it as summation symbol, as is customary.

be the set of selection functions, associated with the occurrences of $\oplus$ and $\ominus$ in e', used to produce $\vec{v}$, and let

$$max\text{-}index = \max\{i \in \mathbb{N}_{>0} \mid g(i) \neq g(i+1), g \in selection\text{-}functions\}.$$

It is easy to see that the word sequence $\vec{s} = (v_{max\text{-}index+1}, v_{max\text{-}index+2}, \ldots) \in \mathcal{L}(elim_{\oplus,\ominus}(e'))$, which implies that $v_{max\text{-}index+1}v_{max\text{-}index+2}\cdots \in \mathcal{L}\left(\sum_{e' \in elim_+(e)}(elim_{\oplus,\ominus}(e'))^\omega\right)$. Moreover, since, for all i, $v_i \in \mathcal{L}(reg(e))$, it holds that $v_1 v_2 \ldots v_{max\text{-}index} \in \mathcal{L}((reg(e))^*)$. Therefore, $v \in \mathcal{L}\left((reg(e))^* \cdot \sum_{e' \in elim_+(e)}(elim_{\oplus,\ominus}(e'))^\omega\right)$, hence the thesis. □

Corollary 2. *Let e be a BST-regular expression. It holds that:*

$$\mathcal{L}(e^\omega) = \mathcal{L}\left((reg(e))^* \cdot \sum_{e' \in elim_+(e)}(elim_{\oplus,\ominus}(e'))^\omega\right).$$

Let us now introduce the operator for the empty string ϵ and the constructor $(.)^+$, whose semantics are as follows:

- $\mathcal{L}(\epsilon) = \{(\epsilon, \epsilon, \epsilon, \ldots)\}$;
- $\mathcal{L}(e^+) = \{(u_{f(0)}u_2 \ldots u_{f(1)-1}, u_{f(1)} \ldots u_{f(2)-1}, \ldots) \mid \vec{u} \in \mathcal{L}(e)$ and $f : \mathbb{N} \to \mathbb{N}_{>0}$ is an increasing function with $f(0) = 1\}$.

The constant ϵ defines the language that contains only the word sequence whose elements are the empty string ϵ (notice that the symbol ϵ is used to denote both the empty word and an expression operator); the constructor $(.)^+$ is similar to the Kleene star, the only difference being that $(.)^+$ does not allow for empty strings (0-iterations). It is immediate to see that the following equivalence holds:

$$\mathcal{L}(e^*) = \mathcal{L}(\epsilon + e^+).$$

We now suitably rewrite the input expression by making use of ϵ and e^+.

Lemma 4. *Every ωBST-regular expression E can be rewritten into an ωBST-regular expression E', where all occurrences of the operators $+$ and $(.)^*$ have been replaced by (occurrences of) the operators $\oplus$ and $(.)^+$ and the constant ϵ.*

Proof. Let E be an ωBST-regular expression. First, we get rid of the Kleene star, by replacing each sub-expression of the form e^* with the expression $\epsilon + e^+$. The resulting ωBST-regular expression E'' has no occurrence of the $(.)^*$ operator, and, thanks to the above remark, it holds that $\mathcal{L}(E) = \mathcal{L}(E'')$.

To remove all the occurrences of the $+$ operator from E'', we exploit Corollary 2, which allows us to replace any sub-expression of E'' of the form e^ω with a new equivalent one with no occurrence of the shuffle operator. □

Thanks to Lemma 4, w.l.o.g., we can restrict ourselves to T- and ωT-regular expressions that have no occurrence of the operators $+$ and $(.)^*$.

We are now ready to show how to map an *ωT-regular expression E* into a corresponding CCA $\mathcal{A}_E$ in such a way that $\mathcal{L}(E) = \mathcal{L}(\mathcal{A}_E)$. We proceed by structural induction on ωT-regular expressions, i.e., when building the CCA $\mathcal{A}_{E'}$ for a sub-expression E' of E, we assume the automata for the sub-expressions of

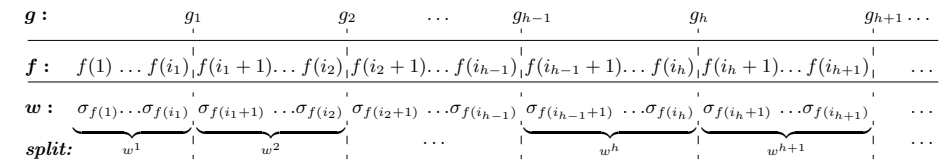


Figure 6: An infinite word $w = w[1]w[2]\dots w[i]\dots = \sigma_{f(1)}\sigma_{f(2)}\dots\sigma_{f(i)}\dots$ is split using sequence $g_1 < g_2 < \dots < g_h < \dots$ into infinitely many finite words w^1, w^2, …, w^h, ….

E' to be available. By construction, all the automata we generate during the algorithm are forced to feature a distinguished *final state* s_f. Slightly abusing the notation, we denote an automaton with such a state as $\mathcal{A} = (S, \Sigma, s_0, s_f, N, \Delta)$, where s_f is the final state of $\mathcal{A}$. W.l.o.g., we assume the sets of states of any pair of distinct automata generated by the construction to be disjoint.

Encoding of T-regular expressions. We first deal with T-regular expressions (sub-grammar rooted in e at page 10). Since T-regular expressions produce languages of word sequences, while automata accept ω-words, we must find a way to extract sequences from ω-words. Intuitively, we do that by splitting an infinite word into infinitely many finite sub-words, each of them corresponding to the sequence of symbols in between two consecutive *check*'s of the 1st counter along the corresponding accepting run.

Formally, let $\pi = (s_0, \mathbf{v}_0)(s_1, \mathbf{v}_1)\dots$ be a run of some CCA $\mathcal{A}$ on some (possibly finite) word w such that $(s_i, \mathbf{v}_i) \rightarrow_{\mathcal{A}}^{\sigma_i} (s_{i+1}, \mathbf{v}_{i+1})$ via some transition δ_i, for each $i \geq 0$, and let f be the trace of w in π with respect to $\mathcal{A}$ (see definitions of trace and run at page 17). Recall that f is such that $\sigma_{f(i)} = w[i]$ for all $i \geq 1$ with $f(i) \neq \infty$ (roughly speaking, f enumerates symbols different from ϵ within sequence $\sigma_0\sigma_1\dots$). Moreover, let $g_1 < g_2 < \dots < g_h < \dots$, with $g_h \in \mathbb{N}$ for every h, be the sequence of indexes corresponding to transitions in π where the 1st counter is checked, that is, for every $i \in \mathbb{N}$, we have that δ_i has the form $(s_i, \sigma_i, s_{i+1}, (1, check))$ if and only if $i = g_h$ for some h. As shown in Figure 6, the sequence $\langle g_h \rangle_{h \in \mathbb{N}_{>0}}$ defines a unique partition of the word $w = \sigma_{f(1)}\sigma_{f(2)}\sigma_{f(3)}\dots$ into infinitely many finite sub-words (some of them are possibly empty words): $w^1 = \sigma_{f(1)}\sigma_{f(2)}\dots\sigma_{f(i_1)}$, $w^2 = \sigma_{f(i_1+1)}\sigma_{f(i_1+2)}\dots\sigma_{f(i_2)}$, $w^3 = \sigma_{f(i_2+1)}\dots\sigma_{f(i_3)}$, …, $w^h = \sigma_{f(i_{h-1}+1)}\dots\sigma_{f(i_h)}$, and so on, with $f(i_h) < g_h \leq f(i_h + 1)$ for every h. We define the language of word sequences accepted by $\mathcal{A}$, denoted by $\mathcal{L}_s(\mathcal{A})$, as $\mathcal{L}_s(\mathcal{A}) = \{(w^1, w^2, \dots, w^h, \dots) \mid$ there exists a run of $\mathcal{A}$ on $w\}$.

For every $\mathcal{A} = (S, \Sigma, s_0, s_f, N, \Delta)$, let $\widehat{\mathcal{A}} = (S, \Sigma, s_0, s_f, N, \Delta \cup \{(s_f, \epsilon, s_0, (1, check))\})$. For each expression e, we build a CCA $\mathcal{A}_e$ such that

$$\mathcal{L}(e) = \mathcal{L}_s(\widehat{\mathcal{A}_e}).$$

Base cases. If $e = \emptyset$, then $\mathcal{A}_\emptyset = (\{s_0, s_f\}, \Sigma, s_0, s_f, 1, \emptyset)$.
If $e{=}\epsilon$, then $\mathcal{A}_\epsilon{=}(\{s_0, s_f\}, \Sigma, s_0, s_f, 1, \{(s_0, \epsilon, s_f, (1, no\_op)), (s_f, \epsilon, s_f, (1, inc))\})$.
If $e{=}a$, then $\mathcal{A}_a{=}(\{s_0, s_f\}, \Sigma, s_0, s_f, 1, \{(s_0, a, s_f, (1, no\_op)), (s_f, \epsilon, s_f, (1, inc))\})$.
See Figure 7, items (a), (b), and (c), for a graphical account of the three cases.

Inductive step. For every CCA $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ and natural number $N' \geq 1$, let us define the *N'-shifted version* of $\mathcal{A}$ as the automaton $\mathcal{A}' = (S, \Sigma, s_0, N + N', \{(s, \sigma, s, (k{+}N', op)) \mid (s, \sigma, s, (k, op)) \in \Delta\})$. Four cases must be considered.

- Let $e = e_1 \cdot e_2$, $\mathcal{A}_{e_1} = (S, \Sigma, s_0, s_f, N, \Delta)$, and $\mathcal{A}_{e_2} = (S', \Sigma, s'_0, s'_f, N', \Delta')$. Moreover, let $\mathcal{A}'_{e_1} = (S, \Sigma, s_0, s_f, N{+}1, \Delta'')$ and $\mathcal{A}'_{e_2} = (S', \Sigma, s'_0, s'_f, N' + N{+}1, \Delta''')$ be the 1-shifted version of $\mathcal{A}_{e_1}$ and the $N{+}1$-shifted version of $\mathcal{A}_{e_2}$, respectively. We define $\mathcal{A}_e = (S \cup S' \cup \{s''_f\}, \Sigma, s_0, s''_f, N{+}N'{+}1, \Delta'' \cup \Delta''' \cup \{(s_f, \epsilon, s'_0, (2, check)), (s'_f, \epsilon, s''_f, (N{+}2, check)), (s''_f, \epsilon, s''_f, (1, inc))\})$. See Figure 7, item (d), for a graphical account.

- Let $e = e_1 \oplus e_2$, $\mathcal{A}_{e_1} = (S, \Sigma, s_0, s_f, N, \Delta)$, $\mathcal{A}_{e_2} = (S', \Sigma, s'_0, s'_f, N', \Delta')$, $\mathcal{A}'_{e_1} = (S, \Sigma, s_0, s_f, N + 1, \Delta'')$, and $\mathcal{A}'_{e_2} = (S', \Sigma, s'_0, s'_f, N' + N + 1, \Delta''')$. We define $\mathcal{A}_e = (S \cup S' \cup \{s''_0, s''_f\}, \Sigma, s''_0, s''_f, N' + N + 1, \Delta'' \cup \Delta''' \cup \{(s''_0, \epsilon, s_0, (1, no\_op)), (s''_0, \epsilon, s'_0, (1, no\_op)), (s_f, \epsilon, s''_f, (2, check)), (s'_f, \epsilon, s''_f, (N{+}2, check)), (s''_f, \epsilon, s''_f, (1, inc))\}$. See Figure 7, item (e), for a graphical account.

- Let $e = e_1^+$, $\mathcal{A}_{e_1} = (S, \Sigma, s_0, s_f, N, \Delta)$, and $\mathcal{A}'_{e_1} = (S, \Sigma, s_0, s_f, N{+}1, \Delta'')$. We let $\mathcal{A}_e = (S \cup \{s''_f\}, \Sigma, s_0, s''_f, N + 1, \Delta'' \cup \{(s_f, \epsilon, s_0, (1, no\_op)), (s_f, \epsilon, s''_f, (2, check)), (s''_f, \epsilon, s''_f, (1, inc))\})$. See Figure 7, item (f), for a graphical account.

- Let $e = e_1^T$ and $\mathcal{A}_{e_1} = (S, \Sigma, s_0, s_f, N, \Delta)$. Moreover, let $\mathcal{A}'_{e_1} = (S, \Sigma, s_0, s_f, N + 2, \Delta'')$ be the 2-shifted version of $\mathcal{A}_{e_1}$. We let $\mathcal{A}_e = (S \cup \{s''_f, s''_1\}, s_0, s''_f, N + 2, \Delta'' \cup \{(s_f, \epsilon, s_0, (2, inc)), (s''_1, \epsilon, s''_f, (2, check)), (s_0, \epsilon, s''_1, (3, check)), (s''_f, \epsilon, s''_f, (1, inc))\})$. See Figure 7, item (g), for a graphical account.

It is worth pointing out the differences between the automata for $(.)^+$ and $(.)^T$. First of all, the automaton for $(.)^T$ makes use of counter 2 to check the condition imposed by $(.)^T$; moreover, it has a transition labeled by $(\epsilon, (3, check))$, exiting from s_0, to allow for the empty string; finally, before entering s''_f, it has to check both counter 2 and counter 3, and since CCA can only execute one check at a time, an intermediate state s''_1 is necessary.

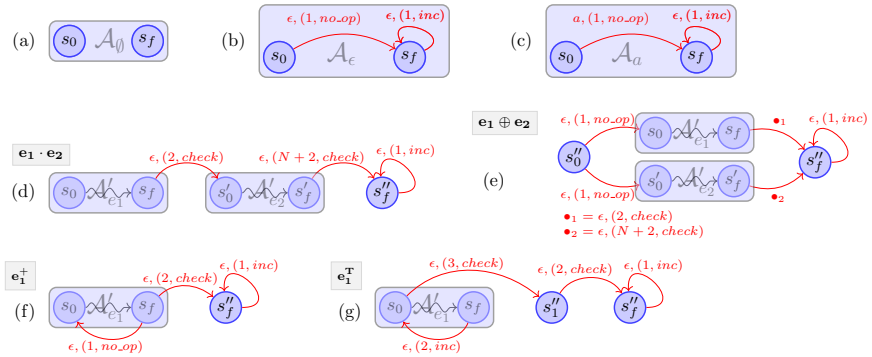


Figure 7: The automata for the translation of a T-regular expression e.

The number of counters used in the translation of a T-regular expression e into a CCA is equal to the length of e plus the number of T-constructors occurring in it. We do not know if a translation exists that uses a number of counters linear in the number of T-constructors occurring in e (if such a translation existed, it would probably be much more technically involved).

The next lemma states the correctness of the proposed encoding. Its proof is straightforward and thus omitted.

Lemma 5. *Let e be a T-regular expression and $\mathcal{A}_e$ be the corresponding automaton. It holds that $\mathcal{L}(e) = \mathcal{L}_s(\widehat{\mathcal{A}_e})$.*

Encoding of ωT-regular expressions. Let us now consider ωT-regular expressions (sub-grammar rooted in E at page 10). We must distinguish three cases.

- Let $E = E_1 + E_2$, $\mathcal{A}_{E_1} = (S, \Sigma, s_0, s_f, N, \Delta)$, and $\mathcal{A}_{E_2} = (S', \Sigma, s'_0, s'_f, N', \Delta')$. Moreover, let $\mathcal{A}'_{E_2} = (S', \Sigma, s'_0, s'_f, N + N', \Delta'')$ be the N-shifted version of $\mathcal{A}_{E_2}$, and let s''_0 be a fresh state. We define $\mathcal{A}_E = (S \cup S' \cup \{s''_0\}, \Sigma, s''_0, s_f, N + N', \Delta \cup \Delta'' \cup \{(s''_0, \epsilon, s_0, (1, no\_op)), (s''_0, \epsilon, s'_0, (1, no\_op))\} \cup \{(s_f, \epsilon, s_f, (k, *)) : * \in \{inc, check\}, N < k \leq N' + N\} \cup \{(s'_f, \epsilon, s'_f, (k, *)) : * \in \{inc, check\}, 0 < k \leq N\})$.

- Let $E = R \cdot E'$, $A_R = (S_R, \Sigma, s^R_0, \Delta_R, F_R)$ be the NFA that recognises the regular language $\mathcal{L}(R)$, and $\mathcal{A}_{E'} = (S, \Sigma, s_0, s_f, N, \Delta)$. We define $\mathcal{A}_E = (S \cup S_R, \Sigma, s^R_0, s_f, N, \Delta \cup \{(s, \sigma, s', (1, no\_op)) : (s, \sigma, s') \in \Delta_R\} \cup \{(s, \epsilon, s_0, (1, no\_op)) : s \in F_R\})$.

- Let $E = e^\omega$. We let $\mathcal{A}_E = \widehat{\mathcal{A}_e}$.

As in the case of T-regular expressions, it is easy to check that this is a valid encoding, as stated by the following theorem.

Theorem 3. *Let E be an ωT-regular expression and $\mathcal{A}_E$ be the corresponding CCA. It holds that $\mathcal{L}(E) = \mathcal{L}(\mathcal{A}_E)$. Therefore, for every ωT-regular expression E there exists a CQA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(E)$.*

5. From ωT-regular languages to S1S+U

In this section, we provide an encoding of ωT-regular expressions into S1S+U [7], which extends S1S with the unbounding quantifier $\mathbb{U}$.

5.1. The logic S1S+U

The logic S1S is the monadic second-order logic of one successor (MSO) interpreted over ω-words. Its formulas are built over a finite, non-empty alphabet Σ and sets V_1 and V_2 of first- and second-order variables, respectively:

$$\varphi ::= \tau \in P_\sigma \mid \tau \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x \varphi \mid \exists X \varphi$$
$$\tau ::= x \mid s(\tau)$$

where $\sigma \in \Sigma$, $x \in V_1$, $X \in V_2$, and $s(.)$ is the successor function. Strings generated by the sub-grammar rooted in τ are called *terms*. We denote by V_Σ

the set $\{P_\sigma \mid \sigma \in \Sigma\}$ (we will write $P(\tau)$ for $\tau \in P$, with $P \in V_2 \cup V_\Sigma$). The elements of V_Σ are second-order variables (i.e., they range over sets of positive natural numbers), but with the following intended semantics: they partition $\mathbb{N}_{>0}$ and an interpretation for them $\mathcal{I} : V_\Sigma \to 2^{\mathbb{N}_{>0}}$ identifies an ω-word $w^\mathcal{I}$ over Σ as follows: $w^\mathcal{I}[i] = \sigma$ if and only if $i \in \mathcal{I}(P_\sigma)$, for all $i \in \mathbb{N}_{>0}$ and $\sigma \in \Sigma$. Notice also that variables in V_Σ always occur free (i.e., not bound by any quantifier). We say that a formula is *closed* if the only free variables are the ones in V_Σ; otherwise, it is *open*. The semantics of a closed formula φ, denoted by $[\![\varphi]\!]$, is the set of all ω-words that *satisfy* it, i.e, $[\![\varphi]\!] = \{w^\mathcal{I} \mid \mathcal{I} \models \varphi\}$. Notice that the constant 1 (the initial position) as well as atomic formulas $t_1 = t_2$ and $t_1 < t_2$, where t_1 and t_2 are terms, are all definable in $\mathsf{S1S}$.

The *unbounding quantifier* $\mathbb{U}$ is defined as follows [7]:

$$\mathbb{U}X\varphi(X) := \bigwedge_{n\in\mathbb{N}} \exists_{\mathsf{fin}} X(\varphi(X) \wedge |X| \geq n).$$

where $\exists_{\mathsf{fin}}$ allows for existential quantification over finite sets, i.e., $\exists_{\mathsf{fin}} X\varphi \equiv \exists X(\varphi \wedge \exists y X \subseteq \{1, \ldots, y\})$ for every second-order variable X and $\mathsf{S1S}{+}\mathsf{U}$-formula φ. The universal quantifier $\forall_{\mathsf{fin}}$ is the dual of $\exists_{\mathsf{fin}}$. Intuitively, $\mathbb{U}$ makes it possible to state that a formula $\varphi(X)$ (containing at least one second-order free variable X) is satisfied by infinitely many finite sets and there is no bound on their sizes. The *bounding quantifier* $\mathbb{B}$ is defined as the negation of $\mathbb{U}$: $\mathbb{B}X\varphi(X) := \neg\mathbb{U}X\varphi(X) \equiv \bigvee_{n\in\mathbb{N}} \forall_{\mathsf{fin}} X(\varphi(X) \to |X| < n)$. Its intended meaning is that there is a bound on the sizes of finite sets that satisfy $\varphi(X)$.

It is worth observing that other extensions of $\mathsf{S1S}$ can be found in the literature (e.g., $\mathsf{S1S}$ with the recurrence quantifier [10]), aiming at capturing prefix-independent extensions of ω-regular languages.

5.2. Encoding

Let E be an ωT-regular expression. We show how to build a formula Φ_E such that $\mathcal{L}(E) = [\![\Phi_E]\!]$.

Using the closure of ωT-regular languages under projection, we can assume all the letters that occur in E are different, thus all sub-expressions occurring in E are distinct. Hereafter, we will always make such an assumption.

The intuitive idea behind the encoding is that a word belongs to the language defined by a certain expression if there is a parsing of the word, compatible with the expression, that witnesses that the word is indeed in the language. For instance, let us consider the concatenation e_1e_2. Every word in $\mathcal{L}(e_1e_2)$ can be decomposed in two parts, with the first one belonging to $\mathcal{L}(e_1)$ and the second one to $\mathcal{L}(e_2)$. In the following, for any ωT-regular expression E, we will provide a formula Φ_E such that an ω-word w satisfies Φ_E if and only if there exists a parsing of w that matches E.

For every sub-expression e of E, we introduce three second-order variables X_e, Y_e, and Z_e. Intuitively, every element of X_e represents the beginning of a sub-string that is matched to e, the elements of Y_e represent the successors of the ending points of the non-empty sub-strings begun by elements of X_e, and Z_e is a subset of X_e, whose elements identify the occurrences of empty sub-strings.

Formally, every triple X_e, Y_e, and Z_e satisfy the following properties:

- if $x \in X_e$, then (i) there exists $y \in Y_e$ such that $x < y$ or (ii) $x \in Z_e$ (possibly both);
- if $y \in Y_e$, then there exists $x \in X_e$ such that $x < y$;
- if $x, x' \in X_e$ and $x < x'$, then (i) there exists $y \in Y_e$ such that $x < y \leq x'$ or (ii) $x \in Z_e$;
- if $y, y' \in Y_e$ and $y < y'$, then there exists $x \in X_e$ such that $y \leq x < y'$;
- $Z_e \subseteq X_e$.

Note that all these conditions are first-order definable. For every sub-expression e, let Ψ_e be the formula with free variables X_e, Y_e, and Z_e that expresses the conjunction of these properties.

For the sake of notation, Let $cons(x, X, y, Y)$ be a formula stating that $x \in X$, $y \in Y$, $x < y$, and between x and y there are no other positions in X or Y.

Let us define now the formulas capturing the possible parsings of sub-expressions. Since the focal point is the translation of the T operator, we restrict our attention to expressions of the form $E = e^\omega$. For each T-regular sub-expression e, let φ_e be defined as follows.

- If $e = a \in \Sigma$, then φ_e states that Z_e is empty and that $x \in X_e$ if and only if $x + 1 \in Y_e$ if and only if $x \in P_a$.
- If $e = e_1 e_2$, then φ_e states that $X_{e_1} = X_e$, $Y_{e_2} \cup Z_{e_2} = Y_e$, and for every $x \in X_e$ we have:
  - if $x \in Z_e$ (both sub-strings are empty), then $x \in Z_{e_1} \cap Z_{e_2}$;
  - if y is the (unique) position for which $cons(x, X_e, y, Y_e)$ holds, then one of the following holds as well:
    - $x \in Z_{e_1}$, $x \in X_{e_2}$, and $y \in Y_{e_2}$ (only the first sub-string is empty);
    - $y \in Z_{e_2}$ and $y \in Y_{e_1}$ (only the second sub-string is empty);
    - there is z such that $x < z < y$, $z \in Y_{e_1}$, $z \in X_{e_2}$, and $y \in Y_{e_2}$ (both sub-strings are non-empty).
- If $e = e_1 + e_2$, then φ_e states that X_{e_1} and X_{e_2} partition X_e, Y_{e_1} and Y_{e_2} partition Y_e, Z_{e_1} and Z_{e_2} partition Z_e. Moreover, whenever $cons(x, X_e, y, Y_e)$, we have that $x \in X_{e_1}$ if and only if $y \in Y_{e_1}$.
- If $e = e_1^*$, then φ_e states that X_e is a subset of $X_{e_1} \cup Z_e$, Y_e is a subset of Y_{e_1}, and if $cons(x, X_e, y, Y_e)$, then $x \in X_{e_1}$ and, for all $z \in X_{e_1}$ such that $x < z \leq y$, it holds $z \in Y_{e_1}$, that is, we can decompose the sub-word matched to e in a bunch of sub-words, each one matched to e_1.
- Let now e be e_1^T. Note that locally this is equivalent to the case above, so all the properties specified by $\varphi_{e_1^*}$ should be specified by $\varphi_{e_1^T}$.

Moreover we need to state that the T property holds. To this end, we use the notion of e-block. Let $e = e_1^T$. An *e-block* is a maximal (finite) set of positions, all of them belonging to the same ω-iteration and starting substrings parsed as e_1; roughly speaking, an e-block represents an iteration of e. The *size* of an e-block X is its cardinality; an e-block is *extendable* if $X \cap Z_{e_1} \neq \emptyset$ and the *potential sizes* of an extendable e-block are all the natural numbers greater than or equal to its size (a non-extendable e-block has a unique potential size that coincides with its size). An *e-block set* is the union of infinitely many e-blocks of bounded size. It is important to observe that an e-block set contains infinitely many e-blocks of the same size. Formula $\psi_e(X)$ below states that X is an e-block set; as a matter of fact, for purely technical reasons, $\psi_e(X)$ also forces the additional constraint $\mathit{infinite}(Z_{e_1} \setminus X)$, that is, if e_1 is instantiated infinitely many times with the empty string, then infinitely many of them (and the e-blocks containing them) must be left out of X.

We formally define $\psi_e(X)$ as the formula

$$\begin{aligned}
&X \subseteq X_{e_1} \ \wedge\ \mathit{infinite}(X) \ \wedge\ (\mathit{infinite}(Z_{e_1}) \rightarrow \mathit{infinite}(Z_{e_1} \setminus X)) \wedge \\
&\forall z \in X \exists x \exists y \big(\ \mathit{cons}(x, X_e, y, Y_e) \ \wedge\ x \leq z \lessdot y \ \wedge \\
&\qquad\qquad \forall t \in X_{e_1}(x \leq t \lessdot y \ \rightarrow\ t \in X)\big) \wedge \\
&\mathbb{B}Y \subseteq X \exists x \exists y \big(\mathit{cons}(x, X_e, y, Y_e) \ \wedge\ x \leq Y \lessdot y\big),
\end{aligned}$$

where $\mathit{infinite}(X)$ is a shorthand for $\forall x \in X \exists y \in X(y > x)$, $x \leq z \lessdot y$ is a shorthand for $x \leq z \lessdot y \vee (z = y \wedge z \in Z_{e_1})$, and $x \leq Y \lessdot y$ is a shorthand for $\forall z \in Y(x \leq z \lessdot y)$.

Now, the T property can be expressed by saying that every e-block set can be extended into another one that contains infinitely many new e-blocks, captured by the following formula φ_e:

$$\begin{aligned}
&\varphi_{e_1^*} \wedge (\mathit{infinite}(X_e) \rightarrow \exists X \psi_e(X)) \wedge \\
&\qquad \forall X(\psi_e(X) \ \rightarrow\ \exists Y(\psi_e(Y) \wedge \mathit{infinite}(Y \setminus X))).
\end{aligned}$$

For an intuitive account of the correctness proof for the encoding of expressions e_1^T (a formal proof is given in Lemma 6), consider the e-block set containing all e-blocks of size at most k. It necessarily contains infinitely many e-blocks of size $k_1 \leq k$. By the above property, it can be extended into another that contains infinitely many new e-blocks whose size is greater than k, yet bounded. Thus, such an extended e-block sets contains infinitely many e-blocks of size $k_2 > k$ and infinitely many e-blocks of size $k_1 \leq k$. By iterating such an argument, it is possible to convince oneself that the T-property is fulfilled.

Let $E = e^\omega$. The last thing we need is a formula φ_E defining a parsing for the ω operator. We can simply define φ_E just saying that X_e is infinite, and except for the first one, all positions in X_e are also in Y_e (that is, we force each e-word to be followed by another e-word).

The formula Φ_E is now defined as the existential closure of the conjunction of the formulas $\varphi_{e'} \wedge \Psi_{e'}$ for every e' that is a sub-expression of e, and the formula φ_E.

As said above, we only focused on expressions of the form e^ω. However this encoding can be easily extended to all ωT-regular expressions just by defining a parsing for the two remaining constructors. Thus, we can conclude the main result of this section.

Lemma 6. *Let E be an ωT-regular expression containing the sub-expression $e = e_1^T$ and let $w \in E' = E[e_1^T \mapsto e_1^*]$ (i.e., w satisfies $\Phi_{E'}$, where E' is obtained from E by replacing e_1^T with e_1^*). Then, w satisfies Φ_E iff either there are in w only finitely many occurrences of sub-strings parsed as e, that is, X_e is finite, or there are infinitely many natural numbers k such that w features infinitely many e-blocks with potential size k.*

Proof. Assume w to satisfy Φ_E and assume, towards a contradiction, that *(i)* X_e is infinite and *(ii)* there are only finitely many natural numbers k such that infinitely many e-blocks with potential size k occur in w. Let K be the finite set containing such natural numbers. Moreover, notice that if w satisfies Φ_E, then w satisfies φ_e as well.

By *(i)* and the conjunct $(\mathit{infinite}(X_e) \rightarrow \exists X \psi_e(X))$ of φ_e, we are guaranteed of the existence of an e-block set and, since an e-block set contains infinitely many e-blocks of the same size, we know that K is not empty.

Let $k_{\max}$ be the largest element of K and $\bar{X}$ be an e-block set containing all non-extendable e-blocks of size not larger than $k_{\max}$ ($\bar{X}$ may or may not contain extendable e-block with size not larger than $k_{\max}$). Clearly, w satisfies $\psi_e[X \mapsto \bar{X}]$ and thus, by φ_e, there exists an e-block set $\bar{Y}$ that contains infinitely many e-blocks (of bounded size) that do not belong to $\bar{X}$. By the definition of $\bar{X}$, such e-blocks are extendable or of size larger than $k_{\max}$ (or both). In either case, it follows that there exists a number $k' > k_{\max}$ such that infinitely many e-blocks of potential size k' occur in w. This is in contradiction with our initial hypothesis that $k_{\max}$ is the largest number such that infinitely many e-blocks of potential size $k_{\max}$ occur in w, hence the thesis follows.

In order to prove the converse direction, we observe that, since we assume that w satisfies $\Phi_{E'}$, showing that w satisfies Φ_E amounts to showing that w satisfies φ_e. More precisely, since w satisfies $\Phi_{E'}$, we have that w satisfies $\varphi_{e_1^*}$ as well. Consequently, to show that w satisfies Φ_E it is enough to prove that w satisfies the second and the third conjunct of φ_e only.

Let us first assume that X_e is finite. Then, the second conjunct of φ_e is vacuously satisfied. Moreover, X_e being finite implies that $\psi(X)$ does not hold for any X, and thus the third conjunct of φ_e is vacuously satisfied as well.

Now, let us assume that X_e is infinite. If there are infinitely many natural numbers k such that infinitely many e-blocks with potential size k occur in w, then we are guaranteed that an e-block set exists. Thus, the second conjunct of φ_e is satisfied. Let $\bar{X}$ be an e-block set (i.e., w satisfies $\psi_e[X \mapsto \bar{X}]$). By the definition of $\psi_e(X)$ (specifically, the last conjunct), there is a bound on

the size of all e-blocks in $\bar{X}$. Let $k_{\max}$ be such a bound. By our assumption, there is a number $k' > k_{\max}$ such that infinitely many e-blocks with potential size k' occur in w. Let Y' be the set containing all e-blocks in $\bar{X}$ and, in addition, all non-extendable e-blocks of size k' occurring in w. If Y' contains infinitely many e-blocks of (actual, not potential) size k', then we let $\bar{Y} = Y'$. Otherwise, there must be infinitely many extendable e-blocks with potential size k' not belonging to $\bar{X}$ (due to the third conjunct of $\psi_e(X)$). Let Y'' be an infinite collection of such extendable e-blocks such that infinitely many of them do not belong to Y'' and let $\bar{Y} = Y' \cup Y''$. Clearly, $\bar{Y}$ is an e-block-set that features infinitely many elements not belonging to $\bar{X}$ (i.e., w satisfies the formula $(\psi_e(X) \wedge \mathit{infinite}(Y \setminus X))[X \mapsto \bar{X}, Y \mapsto \bar{Y}]$), and thus w satisfies Φ_E. □

Theorem 4. *For every ωT-regular expression E, we have that $[\![\Phi_E]\!] = \mathcal{L}(E)$.*

As a conclusive remark, notice that existentially quantified second-order variables X_e, Y_e, and Z_e range over infinite sets, suggesting that φ_t does not belong to the language of WS1S+U, where second-order variables are only allowed to range over finite sets.

6. ωT_s-regular languages and counter-queue automata

In this section, we focus on ωT_s-regular languages. We introduce the class of counter-queue automata (CQA), we prove that their emptiness problem is decidable, and, finally, we show that every ωT_s-regular expression may be turned into a CQA that recognizes exactly the same language.

6.1. Counter-queue automata (CQA)

Let us now introduce counter-queue automata. 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 $\emptyset$. Moreover, we denote the set of the elements in q by $\mathit{Set}(q)$, the maximum among them by $\max(q)$, and the i-th element in q by $q[i]$. Formally, $\mathit{Set}(q) = \{n \in \mathbb{N} : \exists i.q[i] = n\}$ and $\max(q) = \max(\mathit{Set}(q))$ if $\mathit{Set}(q) \neq \emptyset$, $\max(q) = -1$ otherwise. The first and the last element of q can be selected by means of the usual front and back operations: $\mathit{front}(q) = q[1]$ and $\mathit{back}(q) = q[|q|]$ if $\mathit{Set}(q) \neq \emptyset$, $\mathit{front}(q) = \mathit{back}(q) = -1$ otherwise. The *enqueue* operation has to satisfy the uniqueness constraint on the elements of q: for every $n \in \mathbb{N}$, $\mathit{enqueue}(q, n) = q \cdot n$ if $n \notin \mathit{Set}(q)$, $\mathit{enqueue}(q, n) = q$ otherwise. The *dequeue* operation is defined as usual: $\mathit{dequeue}(q) = q[2] \dots q[|q|]$. We denote by $\mathcal{Q}$ the set of all queues.

Counter-queue automata (*CQA*) are defined exactly as CCA (see Definition 3): a CQA 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, $N \in \mathbb{N}_{>0}$ is the number of counters, and $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S \times (\{1, \dots, N\} \times \{\mathit{no\_op}, \mathit{inc}, \mathit{check}\})$

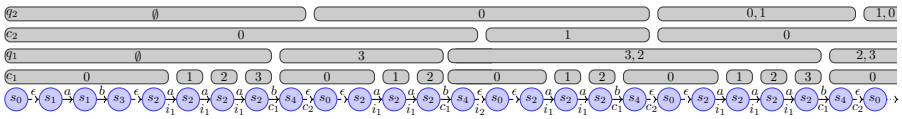


Figure 8: A prefix of a computation of the CQA in Figure 2 (the automaton can be viewed both as a CCA and a CQA). A configuration is characterised by a circle (state) and the rounded-corner rectangles above it (counter-queue configuration). For each i, with $1 \leq i \leq N = 2$, c_i (resp., q_i) is one of its counter (resp., queue) components.

is a transition relation that satisfies the constraint: if $(s, \sigma, s', (k, op)) \in \Delta$ and $op = no\_op$, then $k = 1$. Unlike CCA, however, CQA are equipped with queues, which take action in the definitions of configuration and acceptance conditions.

Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ be a CQA. 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 C, where n_i and q_i are its *counter* and *queue* components, respectively. Hereafter, 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 $\sigma \in \Sigma \cup \{\epsilon\}$, $(s, C) \rightarrow_{\mathcal{A}}^{\sigma} (s', C')$ 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 $(s, C) \rightarrow_{\mathcal{A}}^{\sigma} (s', C')$ via δ. Let $\rightarrow_{\mathcal{A}}^{*}$ be the reflexive and transitive closure of $\rightarrow_{\mathcal{A}}^{\sigma}$ (where we abstract away symbols in $\Sigma \cup \{\epsilon\}$).

The intuition behind the last case of the previous itemization (case $op = check$) is as follows. The automaton has to check infinitely many times all values in the queue (i.e., all values checked at least once). Thus, the head of the queue can be seen as the *current goal* (i.e., the next value to be checked). When a checked value fulfills the current goal (it is equal to the head of the queue), then said goal goes to the back of the line; instead, if the checked value fulfills a non-current goal (it is equal to some element of the queue different from the head), nothing is changed; finally, if it is a new goal (it is a new value), then it is appended to the queue.

The *initial configuration* of $\mathcal{A}$ is the pair (s_0, C_0), where for each $k \in \{1, \ldots, N\}$ we have $C_0[k] = (0, \emptyset)$. 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 8). A *good computation* of $\mathcal{A}$ is a computation $\mathcal{C}$ of $\mathcal{A}$ such that:

(**ac1**) for all $k \in \{1, \ldots, N\}$, $\lim_{i \to +\infty} |queue(C_i[k])| = +\infty$;

(**ac2**) for all $k \in \{1, \ldots, N\}$, $i \in \mathbb{N}$, and $n \in Set(queue(C_i[k]))$, it holds that $|\{i' \in \mathbb{N} \mid back(queue(C_{i'}[k])) = n\}| = +\infty$.

Condition (**ac1**) forces the insertion of infinitely many (distinct) numbers in each queue and condition (**ac2**), together with condition (**ac1**), guarantees that each of them occurs, that is, is removed and added back, infinitely often.

Given two configurations (s_i, C_i) and (s_j, C_j) in $\mathcal{C}$, with $i \leq j$, we say that (s_j, C_j) is *ϵ-reachable* from (s_i, C_i), written $(s_i, C_i) \to^{*\epsilon}_{\mathcal{A}} (s_j, C_j)$, if $(s_{j'-1}, C_{j'-1}) \to^{\epsilon}_{\mathcal{A}} (s_{j'}, C_{j'})$ for all $j' \in \{i+1, \ldots, j\}$, and, as we did for CCA, we write $(s_i, C_i) \to^{*\epsilon}_{\mathcal{A}} (s_\infty, C_\infty)$ for $(s_i, C_i) \to^{*\epsilon}_{\mathcal{A}} (s_j, C_j)$ for all $j \geq i$.

The notions of *run* and *accepting run* of a CQA $\mathcal{A}$ on a word w, as well as the one of *trace* of w in a run π with respect to $\mathcal{A}$, are defined analogously to the corresponding ones for CCA. We denote by $\mathcal{L}(\mathcal{A})$ the set of all ω-words $w \in \Sigma^\omega$ for which an accepting run of $\mathcal{A}$ on w exists, and we say that $\mathcal{A}$ *accepts* the language $\mathcal{L}(\mathcal{A})$.

6.2. Decidability of the emptiness problem for CQA

We now prove that the emptiness problem for CQA is decidable in 2ETIME. The proof is similar to the one for the decidability of emptiness for CCA (see Section 4.2). It consists of 3 steps: (i) we replace general CQA by simple ones; (ii) we prove that their emptiness can be decided by checking the existence of finite witnesses of accepting runs; (iii) we show that the latter can be verified by checking for emptiness a suitable multi-pushdown automaton (MPDA).

Notice that the procedure given here is also a decision procedure for CCA, which translate into CQA. However, the *ad hoc* decision procedure for CCA given in Section 4.2 allows us to establish a better upper bound for the complexity of the emptiness problem for CCA.

The first step is exactly the same as in the proof for CCA; the notion witness of accepting runs for CQA is slightly more complex than then one for CCA; finally, a different class of automata (multi-pushdown rather than NFA) is used to search for witnesses.

Simple CQA. We recall that a CQA is *simple* if, and only if, for each $s \in S$ either $|\{(s, \sigma, s', (k, op)) \mid (s, \sigma, s', (k, op)) \in \Delta \text{ for some } \sigma, s', k, op\}| = 1$ or $op = no\_op$, $k = 1$, and $\sigma = \epsilon$ for all $(s, \sigma, s', (k, op)) \in \Delta$.

As in the case of CCA, it can be easily shown that every CQA $\mathcal{A}$ may be turned into a simple one $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. From now on, we focus on the latter. Moreover, for all pairs of configurations $(s, C), (s', C')$ such that $(s, C) \to^{\sigma}_{\mathcal{A}} (s', C')$, the transition $\delta \in \Delta$ that has been fired in (s, C) is uniquely determined by s and s'. The set of states of a simple CQA can be partitioned in four subsets: $check_k$ states, inc_k states, *sym* states, and *choice* states.

The notion of *prefix computation* of a CQA is analogous to the one for CCA and, with an abuse of notation, we denote by $Prefixes_{\mathcal{A}}$ the sets of all prefix computations of a CQA $\mathcal{A}$. For every prefix computation $\mathcal{P} = (s_0, C_0) \ldots (s_n, C_n) \in$

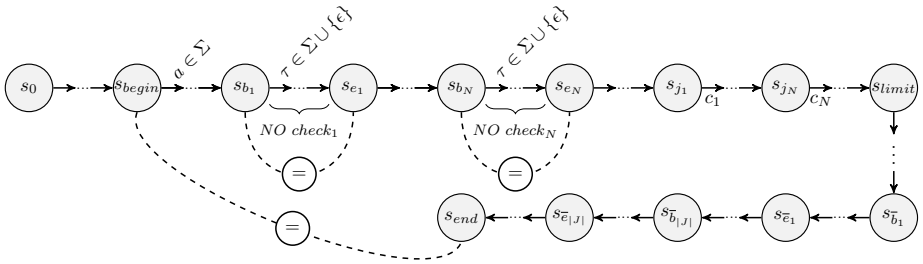


Figure 9: An accepting witness for a CQA.

Prefixes$_{\mathcal{A}}$, it holds that if $(s_n, C_n) \rightarrow_{\mathcal{A}}^{\sigma} (s, C)$, for some $s \in S$, some counter-queue configuration C, and some $\sigma \in \Sigma \cup \{\epsilon\}$, then C is uniquely determined by s_n 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 s and σ'.

Finite witnesses of accepting runs. We show now how to decide CQA emptiness by making use of the notion of accepting witness for a CQA.

Definition 6 (Accepting witness for CQA)**.** *Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ be a CQA. A prefix computation $\mathcal{P} = (s_0, C_0) \dots (s_n, C_n) \in Prefixes_{\mathcal{A}}$ is an* accepting witness *(for $\mathcal{A}$) iff there are $2N + 3$ indexes $begin < b_1 < e_1 < \ldots < b_N < e_N < limit < end$ such that $0 \leq begin, end \leq n$, 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 $k \in \{1, \ldots, N\}$, 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 $b_k \leq j \leq e_k$;*
3. *for each $k \in \{1, \ldots, N\}$, there is j_k, with $e_N < j_k < limit$, such that s_{j_k} is a $check_k$ state;*
4. *let $J = \{j \mid 0 \leq j \leq limit$ 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 $counter(C_j[k])$ many inc_k states between $s_{\overline{b}_j}$ and $s_{\overline{e}_j}$.*

As for the case of CCA, an accepting witness for a CQA $\mathcal{A}$ can be seen as a finite representation of an accepting run of some ω-word on $\mathcal{A}$ (see Figure 9). Thus, deciding whether a CQA $\mathcal{A}$ accepts the empty language amounts to searching *Prefixes*$_{\mathcal{A}}$ for accepting witnesses (the proof easily follows from results in [16]).

Lemma 7. *Let $\mathcal{A}$ be a CQA. Then, $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff Prefixes$_{\mathcal{A}}$ contains an accepting witness for $\mathcal{A}$.*

From CQA to MPDA. By Lemma 7, deciding the emptiness problem for a CQA $\mathcal{A}$ amounts to searching $\mathit{Prefixes}_{\mathcal{A}}$ for an accepting witness. Since we restricted ourselves to simple CQA, we can safely identify elements of $\mathit{Prefixes}_{\mathcal{A}}$ with their sequence of states and thus, by slightly abusing the notation, we write, e.g., $s_0 s_1 \dots s_n \in \mathit{Prefixes}_{\mathcal{A}}$ for $(s_0, C_0) \dots (s_n, C_n) \in \mathit{Prefixes}_{\mathcal{A}}$. Given a CQA $\mathcal{A}$, $\mathcal{L}_{\mathsf{w}}(\mathcal{A})$ is defined, as for CCA, as the language of finite words over the alphabet S (the set of states of $\mathcal{A}$) that are accepting witnesses for $\mathcal{A}$. It is easy to see that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ if and only if $\mathcal{L}_{\mathsf{w}}(\mathcal{A}) \neq \emptyset$. In what follows, for a CQA $\mathcal{A}$ we build a multi-pushdown automaton whose language is exactly $\mathcal{L}_{\mathsf{w}}(\mathcal{A})$. Since the emptiness problem for multi-pushdown automata is decidable, so is the one for CQA.

Similarly to what we have done for CCA and without loss of generality, in what follows we restrict our attention to accepting witnesses for which the sets of indexes required by items 3 and 4 of Definition 6 are ordered. More precisely (we borrow the notation from Definition 6), we assume that

1. there are N indexes $c_1 < \ldots < c_N$, with $e_N < c_1$ and $c_N < \mathit{limit}$, such that s_{c_k} is a check_k state, for each $k \in \{1, \dots, N\}$ (this requirement strengthens the one imposed by item 3 of Definition 6), and
2. for all $j', j'' \in J$, with $j' < j''$, such that $s_{\overline{b}_{j'}}$ and $s_{\overline{b}_{j''}}$ are, respectively, $\mathit{check}_{k'}$ and $\mathit{check}_{k''}$ 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 6).

Notice that if $k' = k''$, then $\overline{b}_{j''} < \overline{b}_{j'}$. It is easy to check that, given a CQA $\mathcal{A}$, $\mathit{Prefixes}_{\mathcal{A}}$ contains an accepting witness, as specified by Definition 6, if and only if it contains one satisfying the additional ordering properties above. Thus, Lemma 7 holds with the new definition of accepting witness as well.

Multi-pushdown automata generalize pushdown ones by featuring more than one stack [17]. At each transition, the automaton can write on any stack, possibly more than one, but it reads only from the first non-empty one.

Definition 7 (multi-pushdown automata)**.** *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* $n \geq 1$ *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,* $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times (\Gamma^*)^n$ *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 $\langle q, w, \gamma_1, \dots, \gamma_n \rangle$, where $q \in Q$, $w \in \Sigma^*$, and $\gamma_i \in \Gamma^*$, for $i \in \{1, \dots, n\}$. We define a binary relation $\vdash_{\mathcal{M}}$ over pairs of configurations as follows: $\langle q, aw, \epsilon, \dots, \epsilon, A\gamma_i, \dots, \gamma_n \rangle \vdash_{\mathcal{M}} \langle q', w, \alpha_1, \dots, \alpha_{i-1}, \alpha_i\gamma_i, \dots, \alpha_n\gamma_n \rangle$ if and only if $(q, a, A, q', (\alpha_1, \dots, \alpha_n)) \in \delta$. 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 $\alpha_1, \dots, \alpha_n$ in the stacks. We denote by $\vdash^*_{\mathcal{M}}$ the reflexive and transitive closure of $\vdash_{\mathcal{M}}$.

A word $w \in \Sigma^*$ is *accepted* by $\mathcal{M}$ if, and only if, $\langle q_0, w, \epsilon, \ldots, \epsilon, Z_0\rangle \vdash^*_{\mathcal{M}} \langle q, \epsilon, \gamma_1, \ldots, \gamma_n\rangle$ 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 5 ([18, 17])**.** *The emptiness problem for MPDA is 2ETIME-complete.*

Lemma 8 ([17])**.** *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 $\mathcal{L}(\mathcal{M}') = \mathcal{L} \cap \mathcal{L}'$.*

Given a CQA $\mathcal{A}$, we build an MPDA $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}_{\mathsf{w}}(\mathcal{A})$ as follows. We first build an MPDA $\hat{\mathcal{M}}$, whose input alphabet is the set of states of $\mathcal{A}$, which accepts accepting witnesses. Unfortunately, such an automaton might also accept accepting witnesses not belonging to $\mathit{Prefixes}_{\mathcal{A}}$. However, since $\mathit{Prefixes}_{\mathcal{A}}$ is a regular language, thanks to Lemma 8, there exists an MPDA $\mathcal{M}$ such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\hat{\mathcal{M}}) \cap \mathit{Prefixes}_{\mathcal{A}} = \mathcal{L}_{\mathsf{w}}(\mathcal{A})$.

Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ be a CQA. $\hat{\mathcal{M}} = \langle n, Q, \Sigma_{\hat{\mathcal{M}}}, \Gamma, \delta, q_0, F, Z_0\rangle$ 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 10 and 11. In particular, the transition relation δ forces the automaton to behave as described in the following 5 steps:

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 $k \in \{1, \ldots, N\}$, 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 $k = 1, \ldots, N$, in the desired order;

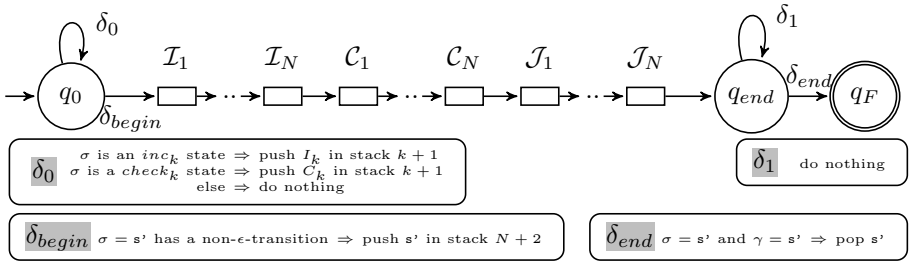


Figure 10: A graphical account of $\hat{\mathcal{M}}$ (part 1).

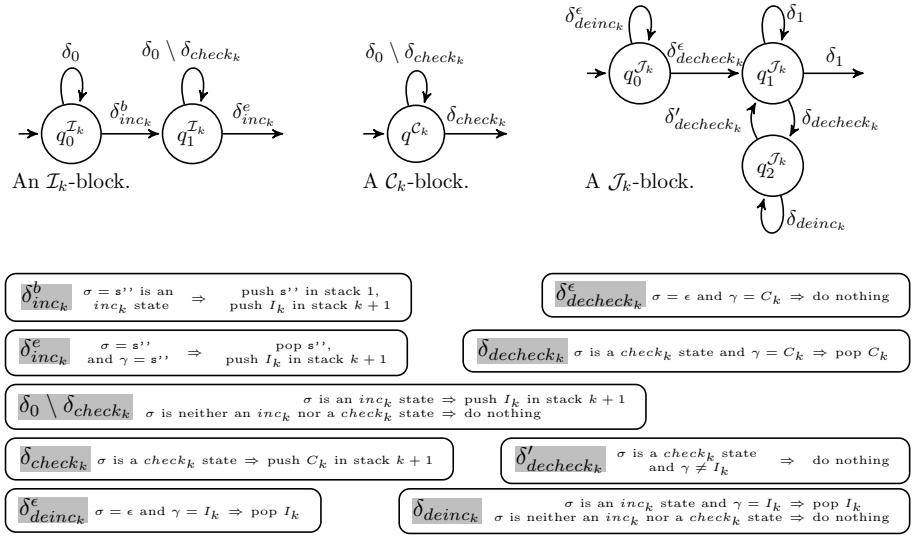


Figure 11: A graphical account of $\hat{\mathcal{M}}$ (part 2): $\mathcal{I}_k$-, $\mathcal{C}_k$-, and $\mathcal{J}_k$-blocks, for $k \in \{1, \ldots, N\}$.

4. until *limit* is reached, whenever it reads an inc_k (resp., $check_k$) state for some $k \in \{1, \ldots, N\}$, 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.

Figure 10 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 11). 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 10) force the automaton to behave exactly as described in step 1 above, while δ_0 in both Figure 10 and Figure 11 captures the behavior described in step 4. Let us consider, for instance, the loop edge labeled with δ_0 in Figure 10. 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 11. 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 6: 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 6. *The emptiness problem for CQA is decidable in 2ETIME.*

6.3. A translation of ωT_s-regular expressions into CQA

In this section, we show how to map an ωT_s-regular expression E into a corresponding CQA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(E)$ (the semantics of the constructor $(.)^{T_s}$ is given in Section 3.3). $\mathcal{A}$ is built in a compositional way: for each sub-expression E' of E, starting from the atomic ones, we introduce a set $\mathcal{S}_{E'}$ of CQA; 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. $\mathcal{A}$ is obtained from a suitable merge of the automata in the set of automata for E.

W.l.o.g., we assume the sets of states of all automata generated during the construction to be pairwise disjoint, i.e., if $\mathcal{A}' \in \mathcal{S}_{E'}$ and $\mathcal{A}'' \in \mathcal{S}_{E''}$ are distinct automata, where E' and E'' are two (not necessarily distinct) sub-expressions of E, then the sets of states of $\mathcal{A}'$ and $\mathcal{A}''$ are disjoint.

We proceed by structural induction on ωT_s-regular expressions, by assuming, when building the set $\mathcal{S}_{E'}$ of CQA for a sub-expression E' of E, the sets of CQA for the sub-expressions of E' to be available. In addition, by the construction, we force all the generated CQA $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ to feature a distinguished *final state* s_f. As we have done for CCA, slightly abusing the notation, we denote an automaton with such a state as $\mathcal{A} = (S, \Sigma, s_0, s_f, N, \Delta)$.

Encoding of T_s-regular expressions. We first deal with T_s-regular expressions (see Section 3.3 for their formal definition). As in the case of T-regular expressions, we must convert ω-words (accepted by CQA) into word sequences (captured by T_s-regular expressions). To this purpose, we define the language of word sequences accepted by a CQA $\mathcal{A}$, denoted by $\mathcal{L}_s(\mathcal{A})$, exactly as we have done for CCA. Intuitively, $\mathcal{L}_s(\mathcal{A})$ is the language of word sequences obtained by splitting the ω-words accepted by $\mathcal{A}$ at the positions where the first counter of $\mathcal{A}$ is checked (see paragraph "Encoding of T-regular expressions" on page 24). Furthermore, as in the case of T-regular expressions, we replace the Kleene star $(.)^*$ (0 or more iterations of the argument expression) by the combination of the operator for the empty string and the usual constructor $(.)^+$ (1 or more iterations).

Despite the fact that the construction of an automaton for a T_s-regular expressions does not introduce particular new insights with respect to the one proposed for T-regular expressions, we have to deal with some technical issues that make the construction more complex. The main one is a direct consequence of the fact that, unlike T-regular expressions, T_s-regular ones are not prefix independent. Such a drawback leads to severe consequences in the complexity of the construction in order to deal correctly with the shuffle operator. One of the consequences of the inter-play of the variability of the shuffle operator (see below) and the lack of prefix-independence is the production of a set of automata (rather than a single one) for each sub-expression E' of E. For better explaining how such an issue is addressed in the construction, we pair the formal definition of the construction with the illustration of a simple running example based on the ωT_s-regular expression $E = ((a^{T_s} + b^{T_s})c)^\omega$.

Let e be the T_s-regular expression $(a^{T_s} + b^{T_s})c$ and let $\vec{v} = (\sigma_1^{n_1}c, \sigma_2^{n_2}c, \ldots)$, with $\sigma_i \in \{a, b\}$ for all $i \in \mathbb{N}_{>0}$, be a word of $\mathcal{L}(e)$. We must distinguish three possible cases.

(1) The shuffle operator behaves fairly. For every $i \in \mathbb{N}$, there is $i' > i$ such that $\sigma_{i'} = a$ and there is $i'' > i$ such that $\sigma_{i''} = b$. In this case, $\vec{v}$ must satisfy the two following conditions: (i) for all $\sigma_i = a$, $|\{i' \mid \sigma_{i'} = a, n_{i'} = n_i\}| = +\infty$ and (ii) for all $\sigma_i = b$, $|\{i' \mid \sigma_{i'} = b, n_{i'} = n_i\}| = +\infty$.

(2) The shuffle operator privileges the left operand. There exists $j \in \mathbb{N}_{>0}$ such that $\sigma_i = a$ for every $i \geq j$. In this case, $\vec{v}$ must satisfy the condition: for all $\sigma_i = a$, $|\{i' \mid \sigma_{i'} = a, n_{i'} = n_i\}| = +\infty$.

(3) The shuffle operator privileges the right operand. There exists $j \in \mathbb{N}_{>0}$ such that $\sigma_i = b$ for every $i \geq j$. In this case, $\vec{v}$ must satisfy the condition: for all $\sigma_i = b$, $|\{i' \mid \sigma_{i'} = b, n_{i'} = n_i\}| = +\infty$.

Generally speaking, a CQA that recognizes $\mathcal{L}(e)$ reads a finite prefix, and then it non-deterministically moves into a region where *(1)* it recognizes infinitely many blocks of (consecutive) a's and of (consecutive) b's, *(2)* it recognizes infinitely many blocks of a's only, or *(3)* it recognizes infinitely many blocks of b's only. Due to the universal nature of the T_s operator, the automaton must keep track of the sizes of the blocks of a's and b's that occur in the finite prefix; however, in case (2) (resp., (3)), the sizes of the blocks of b's (resp., a's) are later discarded. This is the major difference between the T- and the T_s- con-

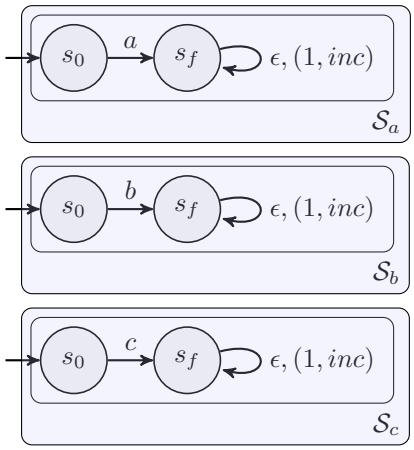


Figure 12: The sets of automata $\mathcal{S}_a, \mathcal{S}_b$ and $\mathcal{S}_c$.

structor: when dealing with the T-constructor, the automaton can ignore any finite number of exponents, and thus it can safely ignore the whole finite prefix.

Let $\mathcal{A}$ be the CQA $(S, \Sigma, s_0, N, \Delta)$. We define the set of *effective counters* of $\mathcal{A}$ as the set $N_{\mathit{eff}} = \{1 \leq i \leq N \mid \exists (s, \sigma, s', (i, op)) \in \Delta \text{ for some } op \in \{check, inc\}\} \subseteq \{1, \ldots, N\}$. $N_{\mathit{eff}} \subseteq \{1 \leq i \leq N\}$ thus identifies the subset of counters on which $\mathcal{A}$ actually operates.

Moreover, as in the case of CCA, for any CQA $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ and natural number $N' \geq 1$, we define the *N'-shifted version* of $\mathcal{A}$ as the automaton $\mathcal{A}' = (S, \Sigma, s_0, N + N', \{(s, \sigma, s, (k + N', op)) \mid (s, \sigma, s, (k, op)) \in \Delta\})$.

Automata for T_s-regular expressions are built recursively as follows.

Base cases.

- If $e = \emptyset$, then $\mathcal{S}_e = \{(\{s_0, s_f\}, \Sigma, s_0, s_f, 1, \emptyset)\}$;
- if $e = a$, then $\mathcal{S}_e = \{(\{s_0, s_f\}, \Sigma, s_0, s_f, 1, \{(s_0, a, s_f, (1, no\_op)), (s_f, \epsilon, s_f, (1, inc))\})\}$;
- if $e = \epsilon$ then $\mathcal{S}_e = \{(\{s_0, s_f\}, \Sigma, s_0, s_f, 1, \{(s_0, \epsilon, s_f, (1, no\_op)), (s_f, \epsilon, s_f, (1, inc))\})\}$.

The sets of automata $\mathcal{S}_a, \mathcal{S}_b$, and $\mathcal{S}_c$ for the T_s-expression e of the running example are depicted in Figure 12

Inductive step.

- Let $e = e_1 \cdot e_2$, $\mathcal{S}_{e_1} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, with $\mathcal{A}_i = (S_i, \Sigma, s_0^i, s_f^i, N_i, \Delta_i)$, and $\mathcal{S}_{e_2} = \{\mathcal{A}'_1, \ldots, \mathcal{A}'_m\}$, with $\mathcal{A}'_j = (S'_j, \Sigma, {s'_0}^j, {s'_f}^j, N'_j, \Delta'_j)$.

  The set of automata for e consists of the concatenation of each automaton in $\mathcal{S}_{e_1}$ with each automaton in $\mathcal{S}_{e_2}$. Since automata are merged in the last step of the construction, we keep all the counters of the automata in $\mathcal{S}_e$ distinct. To this end, we introduce a shifting function $c_{t_1,t_2} : \{1, \ldots, n\} \times \{1, \ldots, m\} \to \mathbb{N}_{>0}$, which is recursively defined as follows. For all $1 \leq i \leq n$ and $1 \leq j \leq m$,

  - $c_{e_1,e_2}(1, 1) = 1$;
  - $c_{e_1,e_2}(i, j) = c_{e_1,e_2}(i, j-1) + N_i + N'_{j-1}$ (for $j > 1$);
  - $c_{e_1,e_2}(i, 1) = c_{e_1,e_2}(i-1, m) + N_{i-1} + N'_m$.

  Let $\hat{\mathcal{A}}_{(i,j)} = (S_i, \Sigma, s_0^i, s_f^i, \hat{N}_{(i,j)}, \hat{\Delta}_{(i,j)})$ be the $c_{e_1,e_2}(i,j)$-shifted version of $\mathcal{A}_i$ and $\hat{\mathcal{A}}'_{(i,j)} = (S'_j, \Sigma, {s'_0}^j, {s'_f}^j, \hat{N}'_{(i,j)}, \hat{\Delta}'_{(i,j)})$ be the $(c_{e_1,e_2}(i,j) + N_i)$-shifted version of $\mathcal{A}'_j$. We define $\mathcal{A}_{(i,j)}$ as follows:

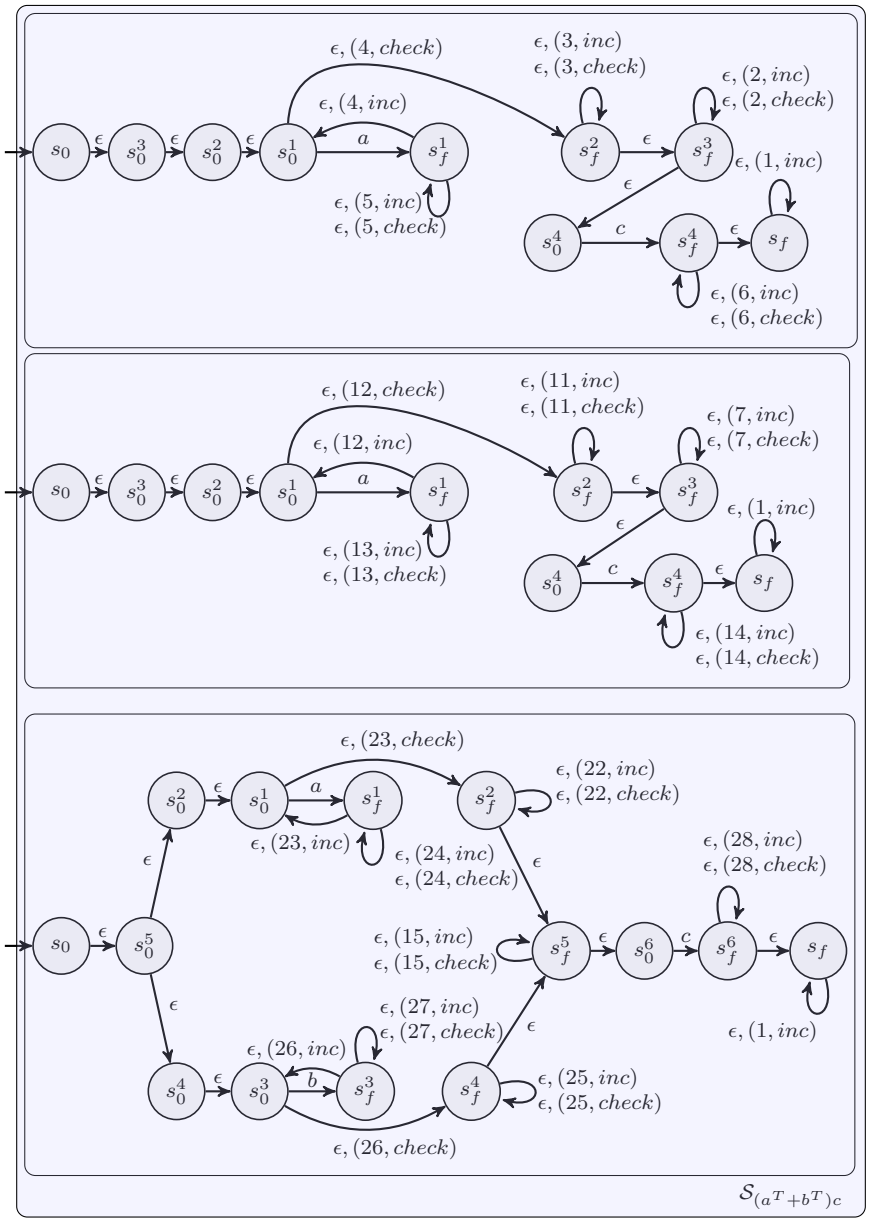


Figure 13: The concatenation operator applied to the sets $\mathcal{S}_{a^{T_s}+b^{T_s}}$ and $\mathcal{S}_c$.

$$\mathcal{A}_{(i,j)} = \left(\begin{array}{l} \{s_0, s_f\} \cup S_i \cup S'_j, \Sigma, s_0, s_f, \\ \hat{N}'_{(i,j)}, \hat{\Delta}_{(i,j)} \cup \hat{\Delta}'_{(i,j)} \cup \\ \left\{ \begin{array}{l} (s_0, \epsilon, s_0^i, (1, no\_op)), \\ (s_f^i, \epsilon, s_f^i, (c_{e_1,e_2}(i,j)+1, check)), \\ (s_f^i, \epsilon, {s'_0}^j, (1, no\_op)), \\ ({s'_f}^j, \epsilon, {s'_f}^j, (c_{e_1,e_2}(i,j)+N_i+1, check)), \\ ({s'_f}^j, \epsilon, s_f, (1, no\_op)), \\ (s_f, \epsilon, s_f, (1, inc)) \end{array} \right\} \end{array} \right).$$

The resulting set of automata is $\mathcal{S}_{e_1 \cdot e_2} = \{\mathcal{A}_{(i,j)} \mid 1 \leq i \leq n, 1 \leq j \leq m\}$. The result of the application of such a construction to the concatenation of $\mathcal{S}_{a^{T_s}+b^{T_s}}$ and $\mathcal{S}_c$ is shown in Figure 13.

- Let $e = e_1 + e_2$, and let $\mathcal{S}_{e_1}$ and $\mathcal{S}_{e_2}$ be defined as for concatenation. In such a case, the shifting function $s_{e_1,e_2} : \{1,\ldots,n\} \times \{1,\ldots,m\} \times \{1,2,3\} \to \mathbb{N}_{>0}$ is recursively defined as follows. For all $i, j \in \mathbb{N}$, with $1 \leq i \leq n$ and $1 \leq j \leq m$,
  - $s_{e_1,e_2}(1,1,1) = 1$;
  - $s_{e_1,e_2}(i,j,2) = s_{e_1,e_2}(i,j,1) + N_i$;
  - $s_{e_1,e_2}(i,j,3) = s_{e_1,e_2}(i,j,2) + N'_j$;
  - $s_{e_1,e_2}(i,j,1) = s_{e_1,e_2}(i,j-1,3) + N_i + N'_{j-1}$ (for $j > 1$);
  - $s_{e_1,e_2}(i,1,1) = s_{e_1,e_2}(i-1,m,3) + N_{i-1} + N'_m$.

  For all $i, j \in \mathbb{N}$, with $1 \leq i \leq n$ and $1 \leq j \leq m$, and each $p \in \{1,2,3\}$, we define $\mathcal{A}_{(i,j,p)}$ as follows.
  - Let $p = 1$ and $\hat{\mathcal{A}}_{(i,j,p)} = (S_i, \Sigma, s_0^i, s_f^i, \hat{N}_{(i,j,p)}, \hat{\Delta}_{(i,j,p)})$ be the $s_{e_1,e_2}(i,j,p)$-shifted version of $\mathcal{A}_i$. Then,

$$\mathcal{A}_{(i,j,p)} = \left(\begin{array}{l} \{s_0, s_f\} \cup S_i, \Sigma, s_0, s_f, \hat{N}_{(i,j,p)}, \hat{\Delta}_{(i,j,p)} \cup \\ \left\{ \begin{array}{l} (s_0, \epsilon, s_0^i, (1, no\_op)), \\ (s_f^i, \epsilon, s_f^i, (s_{e_1,e_2}(i,j,p)+1, check)), \\ (s_f^i, \epsilon, s_f, (1, no\_op)), \\ (s_f, \epsilon, s_f, (1, inc)) \end{array} \right\} \end{array} \right).$$

  - Let $p = 2$ and $\hat{\mathcal{A}}_{(i,j,p)} = (S'_j, \Sigma, {s'_0}^j, {s'_f}^j, \hat{N}'_{(i,j,p)}, \hat{\Delta}'_{(i,j,p)})$ be the $s_{e_1,e_2}(i,j,p)$-shifted version of $\mathcal{A}'_j$. Then,

$$\mathcal{A}_{(i,j,p)} = \left(\begin{array}{l} \{s_0, s_f\} \cup S'_j, \Sigma, s_0, s_f, \hat{N}'_{(i,j,p)}, \hat{\Delta}'_{(i,j,p)} \cup \\ \left\{ \begin{array}{l} (s_0, \epsilon, {s'_0}^j, (1, no\_op)), \\ ({s'_f}^j, \epsilon, {s'_f}^j, (s_{e_1,e_2}(i,j,p)+1, check)), \\ ({s'_f}^j, \epsilon, s_f, (1, no\_op)), \\ (s_f, \epsilon, s_f, (1, inc)) \end{array} \right\} \end{array} \right).$$

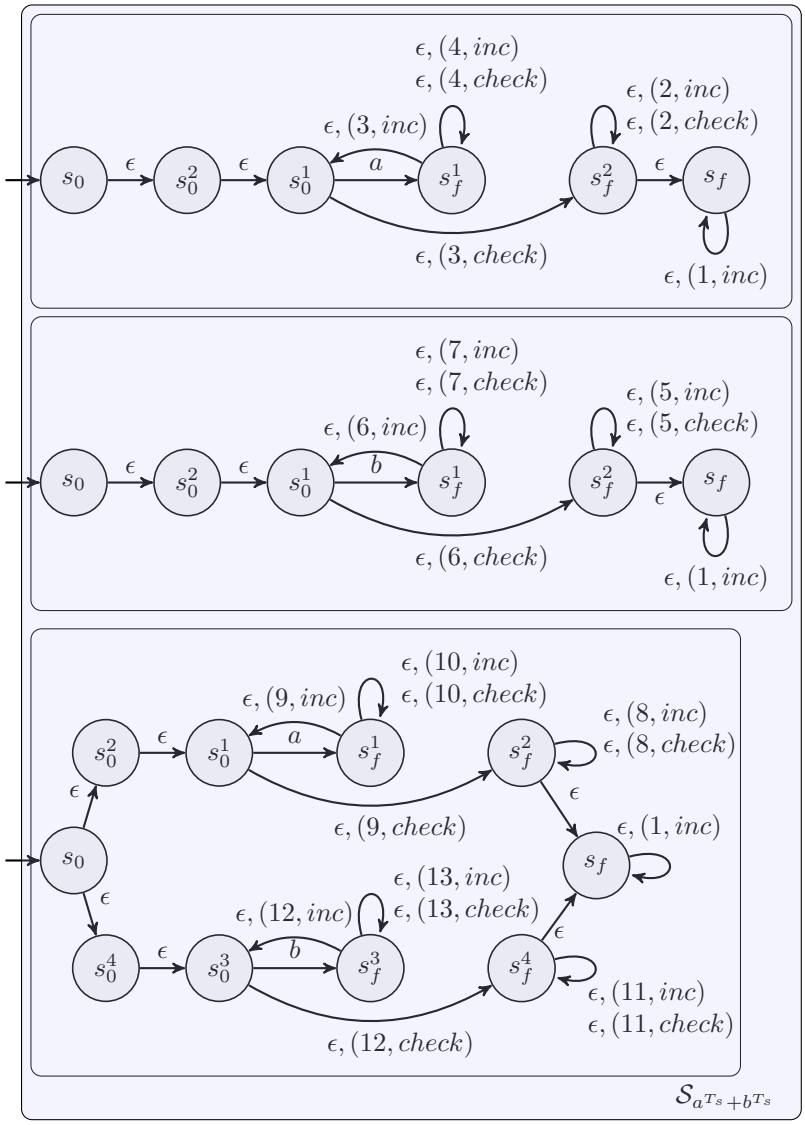


Figure 14: The shuffle operator applied to the sets $\mathcal{S}_{a^T}$ and $\mathcal{S}_{b^T}$.

– Let $p = 3$, $\hat{\mathcal{A}}_{(i,j,p)} = (S_i, \Sigma, s_0^i, s_f^i, \hat{N}_{(i,j,p)}, \hat{\Delta}_{(i,j,p)})$ be the $s_{e_1,e_2}(i,j,p)$-shifted version of $\mathcal{A}_i$, and $\hat{\mathcal{A}}'_{(i,j,p)} = (S'_j, \Sigma, {s'_0}^j, {s'_f}^j, \hat{N}'_{(i,j,p)}, \hat{\Delta}'_{(i,j,p)})$ be the $(s_{e_1,e_2}(i,j,p) + N_i)$-shifted version of $\mathcal{A}'_j$. Then,

$$\mathcal{A}_{(i,j,p)} = \left(\begin{array}{l} \{s_0, s_f\} \cup S_i \cup S'_J, \Sigma, s_0, s_f, \\ \hat{N}'_{(i,j,p)}, \hat{\Delta}_{(i,j,p)} \cup \hat{\Delta}'_{(i,j,p)} \cup \\ \left\{ \begin{array}{l} (s_0, \epsilon, s_0^i, (1, no\_op)), \\ (s_0, \epsilon, {s'_0}^j, (1, no\_op)), \\ (s_f^i, \epsilon, s_f^i, (s_{e_1,e_2}(i,j,p) + 1, check)), \\ ({s'_f}^j, \epsilon, {s'_f}^j, (s_{e_1,e_2}(i,j,p) + N_i + 1, check)), \\ (s_f^i, \epsilon, s_f, (1, no\_op)), \\ ({s'_f}^j, \epsilon, s_f, (1, no\_op)), \\ (s_f, \epsilon, s_f, (1, inc)) \end{array} \right\} \end{array} \right).$$

The resulting set of automata is $\mathcal{S}_{e_1+e_2} = \{\mathcal{A}_{(i,j,p)} \mid 1 \leq i \leq n, 1 \leq j \leq m, 1 \leq p \leq 3\}$. The result of the application of such a construction to $\mathcal{S}_{a^{T_s}}$ and $\mathcal{S}_{b^{T_s}}$ is shown in Figure 14.

- Let $e = e_1^+$, $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, with $\mathcal{A}_i = (S_i, \Sigma, s_0^i, s_f^i, N_i, \Delta_i)$, and $\hat{\mathcal{A}}_i = (S_i, \Sigma, s_0^i, s_f^i, \hat{N}_i, \hat{\Delta}_i)$ be the 1-shifted version of $\mathcal{A}_i$. We define $\mathcal{A}_i^+$ as the automaton:

$$\mathcal{A}_i^+ = \left(\{s_0, s_f\} \cup S_i, \Sigma, s_0, s_f, \hat{N}_i, \hat{\Delta}_i \cup \left\{ \begin{array}{l} (s_0, \epsilon, s_0^i, (1, no\_op)), \\ (s_f^i, \epsilon, s_0^i, (1, no\_op)), \\ (s_f^i, \epsilon, s_f^i, (2, check)), \\ (s_f^i, \epsilon, s_f, (1, no\_op)), \\ (s_f, \epsilon, s_f, (1, inc)) \end{array} \right\} \right).$$

The resulting set of automata is $\mathcal{S}_{e_1^+} = \{\mathcal{A}_i^+ : 1 \leq i \leq n\}$.

- Let $e = e_1^{T_s}$, $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, with $\mathcal{A}_i = (S_i, \Sigma, s_0^i, s_f^i, N_i, \Delta_i)$, and $\hat{\mathcal{A}}_i = (S_i, \Sigma, s_0^i, s_f^i, \hat{N}_i, \hat{\Delta}_i)$ be the 2-shifted version of $\mathcal{A}_i$. We define $\mathcal{A}_i^{T_s}$ as the automaton:

$$\mathcal{A}_i^{T_s} = \left(\{s_0, s_f\} \cup S_i, \Sigma, s_0, s_f, \hat{N}_i, \hat{\Delta}_i \cup \left\{ \begin{array}{l} (s_0, \epsilon, s_0^i, (1, no\_op)), \\ (s_0^i, \epsilon, s_f, (2, check)), \\ (s_f^i, \epsilon, s_0^i, (2, inc)), \\ (s_f^i, \epsilon, s_f^i, (3, check)), \\ (s_f, \epsilon, s_f, (1, inc)) \end{array} \right\} \right).$$

The resulting set of automata is $\mathcal{S}_{e_1^{T_s}} = \{\mathcal{A}_1^{T_s}, \ldots, \mathcal{A}_n^{T_s}\}$. The result of the application of such a construction to the (singleton) sets $\mathcal{S}_a$ and $\mathcal{S}_b$ is depicted in Figure 15.

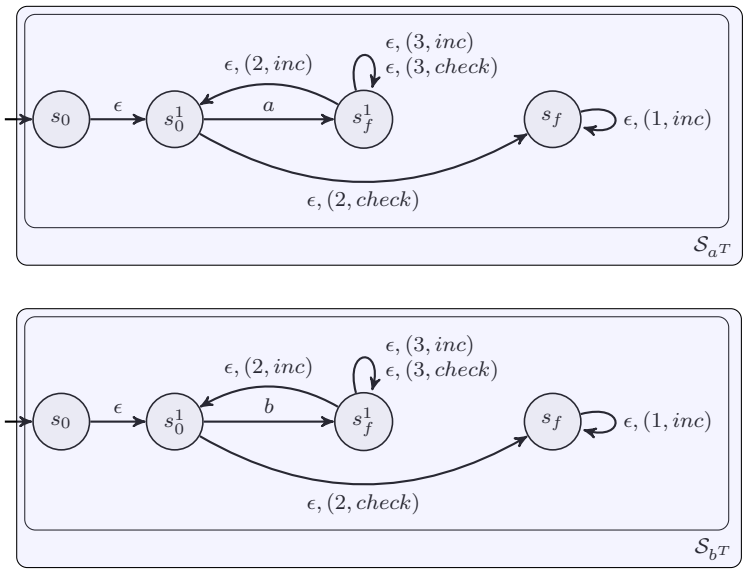


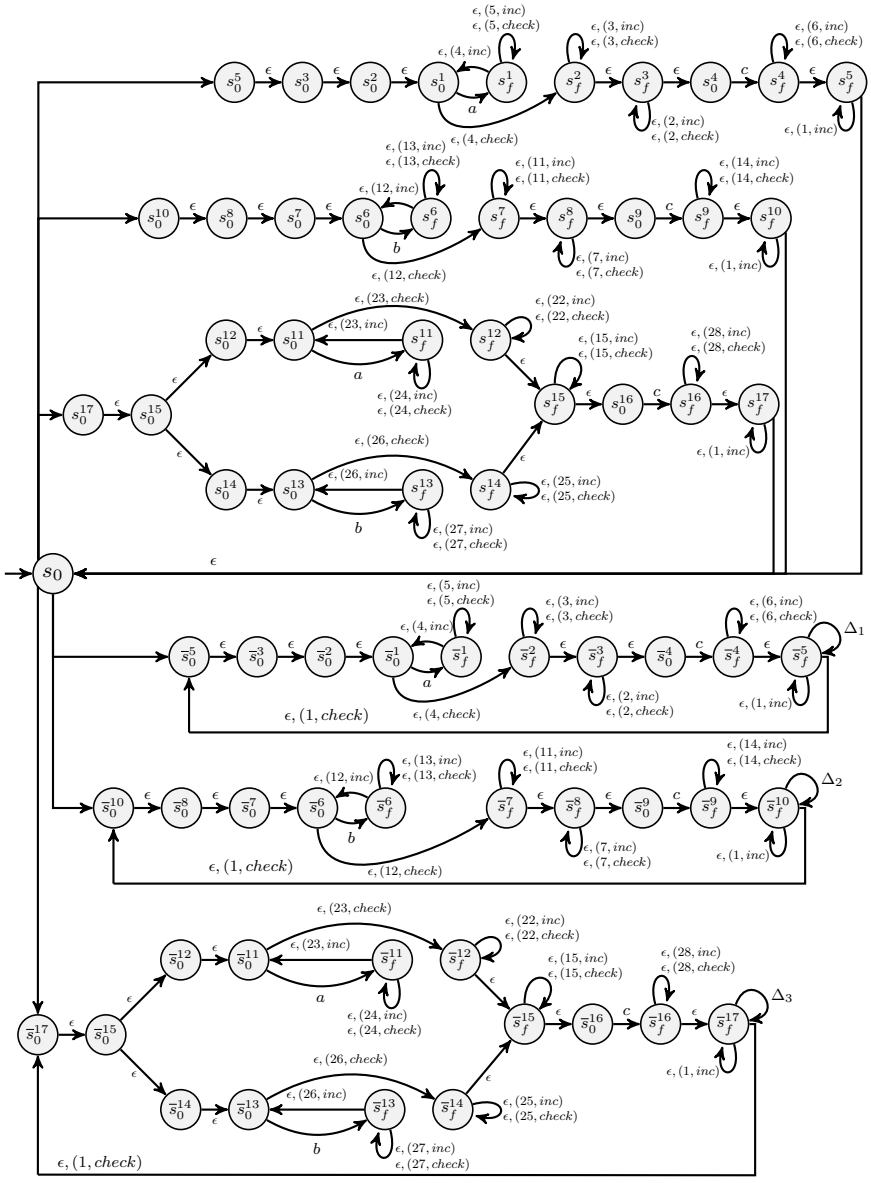
Figure 15: The automata set resulting from the application of $(.)^{T_s}$ to set $\mathcal{S}_a$ (resp., $\mathcal{S}_b$).

Let $\mathcal{S}_e = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, for a T-regular expression e, with $\mathcal{A}_i = (S_i, \Sigma, s_0^i, s_f^i, N_i, \Delta_i)$. It is worth noticing that, by the construction, we have $N_{\mathit{eff}}(\mathcal{A}_i) \cap N_{\mathit{eff}}(\mathcal{A}_j) = \{1\}$ for all $1 \leq i < j \leq n$, that is, distinct automata work on different counters, apart from the special counter 1.

We have already mentioned that the variability of the shuffle operator (together with the lack of prefix-independence) results in having a set of automata (rather than a single one) associated with each sub-expression of an ωT_s-regular one: each automaton of the set captures one of the possible behavior induces by the shuffle operator. An accepting run of a CQA simulating an ωT_s-regular expression must be allowed to switch arbitrarily among the different behaviors (different automata in the set) for a finite prefix of computation, before eventually committing to one of them. Thus, we introduce a set $\overline{\mathcal{S}}_e$ of copies of the automata in $\mathcal{S}_e$: the computation can switch arbitrarily among automata in $\mathcal{S}_e$ before committing to one of the automata in $\overline{\mathcal{S}}_e$.

Formally, let $\overline{\mathcal{S}}_e = \{\overline{\mathcal{A}}_1, \ldots, \overline{\mathcal{A}}_n\}$, with $\overline{\mathcal{A}}_i = (\overline{S}_i, \Sigma, \overline{s}_0^i, \overline{s}_f^i, N_i, \overline{\Delta}_i)$, be the set of copies of the automata in $\mathcal{S}_e$ where each state $s \in S_i$ is renamed $\overline{s}$ in $\overline{S}_i$. Let s_0 be a fresh state and $N_{\max} = \max_{1 \leq i \leq n} N_i$. We define the *closure of* $\mathcal{S}_e$, written $\mathit{closure}(\mathcal{S}_e)$, as:

$$\mathit{closure}(\mathcal{S}_e) = \left(\begin{array}{l} \{s_0\} \cup \bigcup_{i=1}^n (S_i \cup \overline{S}_i), \Sigma, s_0, N_{\max}, \bigcup_{i=1}^n (\Delta_i \cup \overline{\Delta}_i) \cup \\ \bigcup_{i=1}^n \left\{ \begin{array}{l} (s_0, \epsilon, s_0^i, (1, \mathit{no\_op})), (s_f^i, \epsilon, s_0, (1, \mathit{no\_op})), \\ (s_0, \epsilon, \overline{s}_0^i, (1, \mathit{no\_op})), (\overline{s}_f^i, \epsilon, \overline{s}_0^i, (1, \mathit{check})) \end{array} \right\} \cup \\ \bigcup_{i=1}^n \left\{ \begin{array}{l} (\overline{s}_f^i, \epsilon, \overline{s}_f^i, (k, \mathit{inc})), (\overline{s}_f^i, \epsilon, \overline{s}_f^i, (k, \mathit{check})) \mid \\ k \in \{2, \ldots, N_{\max}\} \setminus N_{\mathit{eff}}(\overline{\mathcal{A}}_i) \end{array} \right\} \end{array} \right).$$



$\Delta_1 = \{(i, inc), (i, check) : 7 \leq i \leq 28\}$	$\Delta_2 =$ $\{(i, inc), (i, check) : 2 \leq i \leq 6\} \cup$ $\{(i, inc), (i, check) : 8 \leq i \leq 10\} \cup$ $\{(i, inc), (i, check) : 15 \leq i \leq 28\}$	$\Delta_3 =$ $\{(i, inc), (i, check) : 2 \leq i \leq 14\} \cup$ $\{(i, inc), (i, check) : 16 \leq i \leq 21\}$

Figure 16: The automata representing the closure of $\mathcal{S}_{(a^{T_s}+b^{T_s})c}$.

The closure of $\mathcal{S}_{(a^{T_s}+b^{T_s})^c}$ is shown in Figure 16.

Let $E = r \cdot (e_1 + \ldots + e_n)^\omega$ be an ωT_s-regular expression. For all $1 \leq i \leq n$, let $\mathcal{A}_i = (S_i, \Sigma, s_0^i, N_i, \Delta_i)$ be the closure of $\mathcal{S}_{e_i}$ and $A_r = (S_r, \Sigma, s_0^r, 1, \Delta_r)$ be an ϵ-NFA that recognizes $\mathcal{L}(r)$ with a distinguished final state s_f^r (for the sake of consistency, we assume all the transitions of the ϵ-NFA to be of the form $(s, \sigma, s', (1, no\_op))$). It is easy to prove that the CQA

$$\mathcal{A}_E = (S_r \cup \bigcup_{i=1}^{n} S_i, \Sigma, s_0^r, N_{\max}, \Delta_r \cup \bigcup_{i=1}^{n} \Delta_i \cup \{(s_f^r, \epsilon, s_0^i, (1, no\_op)) \mid 1 \leq i \leq n\})$$

is such that $\mathcal{L}(\mathcal{A}_E) = \mathcal{L}(E)$.

Theorem 7. *Let E be an ωT_s-regular expression. Then, there exists a CQA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(E)$.*

7. An expressiveness comparison

In Section 4, we proved that CCA are at least as expressive as ωT-regular expressions; then, in Section 6 we showed that CQA are at least as expressive as ωT_s-regular expressions. In this section, we demonstrate that CQA are strictly more expressive than CCA. We first prove that CQA are at least as expressive as CCA (Theorem 8), and then we show that there exists an ωT_s-regular language which is not captured by any CCA (Theorem 9).

Theorem 8. *Let $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ be a CCA. Then, there exists a CQA $\mathcal{A}' = (S', \Sigma, s_0', N, \Delta')$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Proof. (sketch) Given a CCA $\mathcal{A}$, we build a CQA $\mathcal{A}'$ that accepts the same language. The termination condition of CQA $\mathcal{A}'$ forces a value that is checked once for a counter to be checked infinitely many times (for that counter). On the other hand, $\mathcal{A}$ is allowed to check a value for a counter only finitely many times. Thus, in order for $\mathcal{A}'$ to accept exactly when $\mathcal{A}$ does, $\mathcal{A}'$ must avoid checking values that are checked by $\mathcal{A}$ only finitely many times.

To implement such a policy, we build $\mathcal{A}'$ in a way that it *behaves* as $\mathcal{A}$ only on a subset of counters (the set of *tracked* counters), while ignoring the others. The set of tracked counters is not fixed: it evolves along the computation; intuitively, a counter is tracked if its next checked value (in $\mathcal{A}$) is checked infinitely often (in $\mathcal{A}$). At the beginning of the computation, $\mathcal{A}'$ guesses which counters are initially tracked (i.e., counters whose first checked value is checked infinitely often); then, every time a counter is checked (and thus reset), $\mathcal{A}'$ guesses if it is to be tracked until it is checked next (i.e., whether or not its next checked value is checked infinitely often).

Technically, this is done as follows.

- States of $\mathcal{A}'$ are pairs $(s, \mathcal{N})$, where s is a state of $\mathcal{A}$ and $\mathcal{N}$ is the set of counters that are being tracked currently.

- Transitions from $(s_0, \emptyset)$ to $(s_0, \mathcal{N})$ are added to $\mathcal{A}'$, for every subset $\mathcal{N}$ of counters.

- For each inc_k transition in $\mathcal{A}$ from s to s' and each state $(s, \mathcal{N})$ in $\mathcal{A}'$, an inc_k transition from $(s, \mathcal{N})$ to $(s', \mathcal{N})$ is added to $\mathcal{A}'$ if k is being tracked ($k \in \mathcal{N}$); otherwise, a transition from $(s, \mathcal{N})$ to $(s', \mathcal{N})$ that does not act on the counter is added to $\mathcal{A}'$.

- Similarly, for a $check_k$ transition in $\mathcal{A}$ there must be a $check_k$ transition in $\mathcal{A}'$ if and only if k is being tracked; in addition, when a $check_k$ transition is executed in $\mathcal{A}$, $\mathcal{A}'$ must guess whether or not counter k is to be tracked until its next check; thus, for each inc_k transition in $\mathcal{A}$ from s to s' and each state $(s, \mathcal{N})$ in $\mathcal{A}'$, two inc_k transitions, one from $(s, \mathcal{N})$ to $(s', \mathcal{N} \cup \{k\})$ and another from $(s, \mathcal{N})$ to $(s', \mathcal{N} \setminus \{k\})$, are added to $\mathcal{A}'$ if k is being tracked ($k \in \mathcal{N}$); otherwise, transitions from $(s, \mathcal{N})$ to $(s', \mathcal{N} \cup \{k\})$ and from $(s, \mathcal{N})$ to $(s', \mathcal{N} \setminus \{k\})$ not acting on the counter are added to $\mathcal{A}'$. □

Proof. W.l.o.g., let us assume that $\mathcal{A}$ is simple, that is, if an $inc_k/check_k/sym$ transition, for some $k \in \{1, \ldots, N\}$, may be fired at a state $s \in S$, then such a transition is the only outgoing transition for s. As already pointed out in Section 4, this allows us to distinguish among inc_k, $check_k$, and sym states. The only remaining states s are those with one or more outgoing transitions of the form $(s, \epsilon, s', (1, no\_op))$ for some $s' \in S$. Let us define $\mathcal{A}'$ as follows. $S' = (S \times 2^{\{1,\ldots,N\}}) \cup \{s'_0\}$ (i.e., a state is either a fresh new initial state s'_0 or a pair consisting of an original state of the automaton and a subset of counters). For every $(s, \mathcal{N}), (s', \mathcal{N}') \in (S \times 2^{\{1,\ldots,N\}})$, $\sigma \in \Sigma \cup \{\epsilon\}$, $k \in \{1, \ldots, N\}$, and $op \in \{no\_op, inc, check\}$, we have $((s, \mathcal{N}), \sigma, (s', \mathcal{N}'), (k, op)) \in \Delta'$ if and only if one of the following conditions holds:

- $(s, \sigma, s', (k, op)) \in \Delta$, $\mathcal{N} = \mathcal{N}'$, and $op = no\_op$;
- $(s, \sigma, s', (k, op)) \in \Delta$, $k \in \mathcal{N}$, $\mathcal{N} = \mathcal{N}'$, and $op = inc$;
- $(s, \sigma, s', (k, op')) \in \Delta$, $k \notin \mathcal{N}$, $\mathcal{N} = \mathcal{N}'$, $op = no\_op$ and $op' = inc$;
- $(s, \sigma, s', (k, op)) \in \Delta$, $k \in \mathcal{N}$, $\mathcal{N} = \mathcal{N}'$, and $op = check$;
- $(s, \sigma, s', (k, op)) \in \Delta$, $k \in \mathcal{N}$, $\mathcal{N}' = \mathcal{N} \setminus \{k\}$, and $op = check$;
- $(s, \sigma, s', (k, op')) \in \Delta$, $k \notin \mathcal{N}$, $\mathcal{N} = \mathcal{N}'$, and $op = no\_op$ and $op' = check$;
- $(s, \sigma, s', (k, op')) \in \Delta$, $k \notin \mathcal{N}$, $\mathcal{N}' = \mathcal{N} \cup \{k\}$, $op = no\_op$ and $op' = check$.

Moreover, we have $(s'_0, \epsilon, (s_0, \mathcal{N}), (1, no\_op)) \in \Delta'$ for every $\mathcal{N} \subseteq \{1, \ldots, N\}$.

Now we prove that for every word $w \in \Sigma^\omega$ we have $w \in \mathcal{L}(\mathcal{A})$ if and only if $w \in \mathcal{L}(\mathcal{A}')$.

Left-to-right direction. Let $w \in \mathcal{L}(\mathcal{A})$. Let $\pi = (s_0, \mathbf{v}_0)(s_1, \mathbf{v}_1)\ldots$ be an accepting run of $\mathcal{A}$ on w, where $(s_i, \mathbf{v}_i) \rightarrow_{\mathcal{A}}^{\sigma_i} (s_{i+1}, \mathbf{v}_{i+1})$ via δ_i (recall that $\sigma_i \in \Sigma \cup \{\epsilon\}$) for all i. For every $i \in \mathbb{N}$ and $k \in \{1, \ldots, N\}$, let us define $value(i, k)$ as the value of the counter k at the first $check_k$ transition fired at a

position $j \geq i$. Formally, the function $value : \mathbb{N} \times \{1, \ldots, N\} \to \mathbb{N}$ is recursively defined as follows: $value(i,k) = \mathbf{v}_i[k]$ if δ_i is a $check_k$ transition, $value(i,k) = value(i+1,k)$ otherwise. Since π is accepting, for every $k \in \{1, \ldots, N\}$ there are infinitely many $check_k$ transitions and thus the function $value$ is total. For every $k \in \{1, \ldots, N\}$, let $\infty_k = \{n \in \mathbb{N} \mid |\{i \in \mathbb{N} \mid value(i,k) = n\}| = \infty\}$. The run $\pi' = (s'_0, C_0)(s'_1, C_1) \ldots$ of $\mathcal{A}'$, where $(s'_i, C_i) \to_{\mathcal{A}'}^{\sigma'_i} (s'_{i+1}, C_{i+1})$ via δ'_i for all i, is iteratively defined in such a way that $\delta'_0 = (s'_0, \epsilon, (s_0, \mathcal{N}), (1, no\_op))$, where $\mathcal{N} = \{k \in \{1, \ldots, N\} \mid value(0,k) \in \infty_k\}$, (s'_0, C_0) is the initial configuration of $\mathcal{A}'$, that is, we have $C_0[k] = (0, \emptyset)$ for each $k \in \{1, \ldots, N\}$, and that for every $i \in \mathbb{N}_{>0}$ it holds:

- $\sigma'_i = \sigma_{i-1}$;
- $s'_i = (s_{i-1}, \mathcal{N}_{i-1})$, where $\mathcal{N}_{i-1} = \{k \in \{1, \ldots, N\} \mid value(i-1,k) \in \infty_k\}$;
- $\mathbf{v}_i[k] = \begin{cases} \mathbf{v}_{i-1}[k] & \text{if } value(i-1,k) \in \infty_k, \\ 0 & \text{otherwise.} \end{cases}$

Since $\mathcal{A}$ is simple, C_i is univocally determined for all $i > 0$. More precisely, we have the following correspondence between π and π'. For $i > 0$:

- if $\delta_{i-1} = (s, \sigma, s', (k, no\_op))$, then $\delta'_i = ((s, \mathcal{N}_i), \sigma, (s', \mathcal{N}_i), (k, no\_op))$;
- if $\delta_{i-1} = (s, \sigma, s', (k, inc))$ and $k \in \mathcal{N}_i$, then $\delta'_i = ((s, \mathcal{N}_i), \sigma, (s', \mathcal{N}_i), (k, inc))$;
- if $\delta_{i-1} = (s, \sigma, s', (k, inc))$ and $k \notin \mathcal{N}_i$, then $\delta'_i = ((s, \mathcal{N}_i), \sigma, (s', \mathcal{N}_i), (k, no\_op))$;
- if $\delta_{i-1} = (s, \sigma, s', (k, check))$, $k \in \mathcal{N}_i$ and $value(i,k) \in \infty_k$, then $\delta'_i = ((s, \mathcal{N}_i), \sigma, (s', \mathcal{N}_i), (k, check))$;
- if $\delta_{i-1} = (s, \sigma, s', (k, check))$, $k \in \mathcal{N}_i$ and $value(i,k) \notin \infty_k$, then $\delta'_i = ((s, \mathcal{N}_i), \sigma, (s', \mathcal{N}_i \setminus \{k\}), (k, check))$;
- if $\delta_{i-1} = (s, \sigma, s', (k, check))$, $k \notin \mathcal{N}_i$ and $value(i,k) \notin \infty_k$, then $\delta'_i = ((s, \mathcal{N}_i), \sigma, (s', \mathcal{N}_i), (k, no\_op))$;
- if $\delta_{i-1} = (s, \sigma, s', (k, check))$, $k \notin \mathcal{N}_i$ and $value(i,k) \in \infty_k$, then $\delta'_i = ((s, \mathcal{N}_i), \sigma, (s', \mathcal{N}_i \cup \{k\}), (k, no\_op))$.

It is easy to check that the above construction guarantees that if f is the trace of w in π with respect to $\mathcal{A}$, then f', where $f'(n) = f(n) + 1$ for every $n \in \mathbb{N}$, is the trace of w in π' with respect to $\mathcal{A}'$. Moreover, a number n is inserted in the queue of a counter k ($k \in \{1, \ldots, N\}$) if and only if $n \in \infty_k$. Since π is accepting, it holds that $|\infty_k| = \infty$ for each k, and thus π' is an accepting run of $\mathcal{A}'$ on w. Hence, $w \in \mathcal{L}(\mathcal{A}')$.

Right-to-left direction. Let $w \in \Sigma^\omega$ be a word accepted by $\mathcal{A}'$ and let $\pi' = (s'_0, C_0)((s_1, \mathcal{N}_1), C_1)((s_2, \mathcal{N}_2), C_2) \ldots$ be an accepting run of $\mathcal{A}'$ on w,

where $((s_i, \mathcal{N}_i), C_i) \rightarrow^{\sigma'_i}_{\mathcal{A}'} ((s_{i+1}, \mathcal{N}_{i+1}), C_{i+1})$ via δ'_i for all $i > 0$. Let $value : \{1, \ldots, N\} \times \mathbb{N} \rightarrow \mathbb{N}$ be the function such that $value(k, i) = |\{j \in \mathbb{N} \mid 0 < j < i, s_j \text{ is an } inc_k \text{ state, and for every } j' \in \{j, \ldots, i-1\} \text{ the state } s_{j'} \text{ is not a } check_k \text{ state } \}|$.

It suffices to prove that $\pi = (s_1, \mathbf{v}_1)(s_2, \mathbf{v}_2)\ldots$, where for every $i \geq 1$ we have $\mathbf{v}_i[k] = value(k, i)$, is an accepting run of $\mathcal{A}$ on w. First, by the definition of $\mathcal{A}'$, $(s_1, \mathbf{v}_1)$ is an initial configuration of $\mathcal{A}$. Next, we show that $(s_i, \mathbf{v}_i) \rightarrow^{\sigma_i}_{\mathcal{A}} (s_{i+1}, \mathbf{v}_{i+1})$ for every $i > 0$. Let $\delta'_i = ((s_i, \mathcal{N}_i), \sigma_i, (s_{i+1}, \mathcal{N}_{i+1}), (k, op)) \in \Delta'$ be the transition executed at position i of run π'. By the definition of Δ' and by the fact that $\mathcal{A}$ is simple, we have that there exists a unique transition $\delta_i = (s_i, \sigma_i, s_{i+1}, (k, op')) \in \Delta$ for some $op' \in \{inc, check, no\_op\}$. In order to verify that $(s_i, \mathbf{v}_i) \rightarrow^{\sigma_i}_{\mathcal{A}} (s_{i+1}, \mathbf{v}_{i+1})$ via δ_i, for every $k' \in \{1, \ldots, N\}$ we need to consider the following cases.

- If $k' \neq k$ or $op = no\_op$, then δ_i does not modify counter k'. On the other hand, s_i is neither an $inc_{k'}$ nor a $check_{k'}$ state, and thus we have $\mathbf{v}_i[k'] = value(k', i) = value(k', i+1) = \mathbf{v}_{i+1}[k']$, which is coherent with the semantics of δ_i.
- If $k' = k$ and $op = inc$, then s^i is an $inc_{k'}$ state, and thus we have $\mathbf{v}_i[k'] = value(k', i) = value(k', i+1) - 1 = \mathbf{v}_{i+1}[k'] - 1$, which amounts to $\mathbf{v}_{i+1}[k'] = \mathbf{v}_i[k'] + 1$, coherently with the semantics of δ_i.
- If $k' = k$ and $op = check$ then s_i is a $check_{k'}$ state, and thus we have $\mathbf{v}_{i+1}[k'] = 0$, which is, once again, coherent with the semantics of δ_i.

This shows that π is a computation of $\mathcal{A}$. In order to conclude that it is an accepting run of $\mathcal{A}$ on w, we notice that if f' is the trace of w in π' with respect to $\mathcal{A}'$, then f, where $f(n) = f(n) - 1$ for every $n \in \mathbb{N}$, is the trace of w in π with respect to $\mathcal{A}$. Moreover, it is immediate to see that if a counter is incremented (resp., checked) in π', then it is incremented (resp., checked) in π as well. Since π' is accepting, for every $k \in \{1, \ldots, N\}$ it holds that $|\{i \in \mathbb{N} \mid value(k, j) = i \text{ and } s_j \text{ is a } check_{k'} \text{ state}\}| = \infty$ (i.e., infinitely many values are checked for each counters) and if there is a position j such that $value(k, j) = i$ and s_j is a $check_{k'}$ state, then there are infinitely many such positions (i.e., if a value is checked for a counter, then it is checked infinitely many times for that counter). Hence, for every counter there are infinitely many values that are checked infinitely many times, meaning that π is an accepting run of $\mathcal{A}$ on w, and thus $w \in \mathcal{L}(\mathcal{A})$. □

Theorem 9. *The ωT_s-regular language $L = (a^{T_s}b)^\omega$ is not recognized by any CCA, that is, $\mathcal{L}(\mathcal{A}) \neq L$ for every CCA $\mathcal{A}$.*

Proof. The idea of the proof is to show that, whenever a CCA accepts a particular word $w \in L$, it necessarily accepts a word $\overline{w}$, obtained by w by pumping some large block of consecutive a's, that does not belong to L. Intuitively, in order to accept large blocks of consecutive a's, a run must go through a loop;

an accepting run of a CCA is allowed to pump this loop an arbitrary number of times, thus producing some counter values that are checked only once, and, consequently, accepting a word not belonging to L.

Towards a more formal proof, let us suppose by contradiction that there exists a CCA $\mathcal{A} = (S, \Sigma, s_0, N, \Delta)$ that recognizes L. In particular, $\mathcal{A}$ recognizes the word $w = \Pi_{i=0}^{\omega} w_i$ where $w_0 = b$, $w_{i+1} = w_i \cdot a^{i+1}b$, and Π represents the repeated concatenation operator. Given two $i, i' \in \mathbb{N}$, with $i \leq i'$, we denote by $w_{i,i'}$ the sub-word of w defined as:

$$w_{i,i'} = \begin{cases} a^i b & \text{if } i = i' \\ a^i b \cdot w_{i+1,i'} & \text{otherwise.} \end{cases}$$

Let $\pi = (s_0, \mathbf{v}_0)(s_1, \mathbf{v}_1)\ldots$ be an accepting run of $\mathcal{A}$ on w, where $(s_i, \mathbf{v}_i) \rightarrow_{\mathcal{A}}^{\sigma_i} (s_{i+1}, \mathbf{v}_{i+1})$ via δ_i for all i. We define the function $map : \mathbb{N} \rightarrow \mathbb{N}$ such that, for every i, $\sigma_{map(i)} \ldots \sigma_{map(i+1)-1} = w_i$, with $\sigma_{map(i)} \neq \epsilon$. Intuitively, $map(i)$ allows one to access (the beginning of) the portion of the run that processes the word w_i in w. Moreover, in order to have a finer access to sub-words of w, we define the function *block*, which allows one to access the beginning of the portion of the run that processes the i-th block of consecutive a's of word $w_{i'}$ in w. Formally, for every $i, i' \in \mathbb{N}_{>0}$, with $i \leq i'$, $block : \{(i, i') \mid i \leq i'\} \rightarrow \mathbb{N}$ satisfies the following conditions:

1. $map(i') \leq block(i, i') < map(i' + 1)$;
2. $\sigma_0 \ldots \sigma_{block(i,i')-1} \in \Sigma^* \cdot b \cup \{\epsilon\}$;
3. there exists $j > block(i, i')$ such that $\sigma_{block(i,i')} \ldots \sigma_j = a^i b$.

Let us now show that from an accepting run of $\mathcal{A}$ on w we can obtain an accepting run of $\mathcal{A}$ on an ω-word $\overline{w} \notin L$. Let $\bar{i} = |S|+1$. We define two auxiliary functions $begin : \mathbb{N}_{>0} \rightarrow \mathbb{N}$ and $end : \mathbb{N}_{>0} \rightarrow \mathbb{N}$, such that, for every $i \in \mathbb{N}_{>0}$,

1. $block(\bar{i}, \bar{i} + i) \leq begin(i) < end(i) < block(\bar{i} + 1, \bar{i} + i)$;
2. $\sigma_{begin(i)} = \sigma_{end(i)} = a$;
3. $s_{begin(i)} = s_{end(i)}$.

The word $\overline{w}$ is equal to $\Pi_{i=0}^{\omega} \overline{w}_i$, where the word $\overline{w}_i$ is defined as follows:

$$\overline{w}_i = \begin{cases} w_i & \text{if } i \leq \bar{i}, \\ w_{\bar{i}-1} \cdot a^{\bar{i}+|\{j \in \mathbb{N} \mid begin(i) \leq j < end(i), \sigma_j = a\}|} b \cdot w_{\bar{i}+1,i} & \text{otherwise.} \end{cases}$$

Notice that, by the definition of *begin* and *end*, $|\{j \in \mathbb{N} \mid begin(i) \leq j < end(i), \sigma_j = a\}| \geq 1$. Clearly, $\overline{w}$ does not belong to $(a^{T_s} b)^{\omega}$ since it features only one occurrence of $a^{\bar{i}}b$, precisely that in $\overline{w}_{\bar{i}}$ (by construction, in every word $\overline{w}_{\bar{i}+i}$, with $i > 0$, the $\bar{i}$-th block of consecutive a's contains at least $\bar{i} + 1$ occurrences of a).

To complete the proof, it suffices to prove that $\overline{w}$ is accepted by $\mathcal{A}$. Let π be the accepting run of $\mathcal{A}$ on w. To this end, we introduce a set $\mathbf{\Omega} \subseteq \{\Omega_{n,z} \mid 1 \leq n \leq N, z \in \mathbb{N}\}$ that satisfies the following conditions:

(a) $\Omega_{n,z} \subseteq \{(x,y) \in \mathbb{N}^2 \mid x \leq y\}$, for all $\Omega_{n,z} \in \mathbf{\Omega}$;

(b) for each $n \in \{1,\ldots,N\}$, there are infinitely many $z \in \mathbb{N}$ such that $\Omega_{n,z} \in \mathbf{\Omega}$;

(c) $|\Omega_{n,z}| = +\infty$, for all $\Omega_{n,z} \in \mathbf{\Omega}$;

(d) for all $\Omega_{n,z}, \Omega_{n',z'} \in \mathbf{\Omega}$, with $(n,z) \neq (n',z')$, $\Omega_{n,z} \cap \Omega_{n',z'} = \emptyset$, and for all $(x,y),(x',y') \in \bigcup_{\Omega_{n,z} \in \mathbf{\Omega}} \Omega_{n,z}$, with $(x,y) \neq (x',y')$, either $y' < x$ or $y < x'$;

(e) for every $\Omega_{n,z} \in \mathbf{\Omega}$ and $(x,y) \in \Omega_{n,z}$, $\mathbf{v}_y[n] = z$, δ_x and δ_y are $check_n$ transitions, and for all $x' \in \mathbb{N}$, with $x < x' < y$, $\delta_{x'}$ is not a $check_n$ transition.

It is not difficult to see that for every computation $\pi = (s_0,\mathbf{v}_0)(s_1,\mathbf{v}_1)\ldots$ of $\mathcal{A}$ such that $(s_i,\mathbf{v}_i) \rightarrow_{\mathcal{A}}^{\sigma_i} (s_{i+1},\mathbf{v}_{i+1})$ for all i and $\sigma_0\sigma_1\sigma_2\ldots$ is an ω-word, π is an accepting run of $\mathcal{A}$ on $\sigma_0\sigma_1\sigma_2\ldots$ if and only if there exists a set $\mathbf{\Omega}$ that satisfies, with respect to π, conditions *(a)*-*(e)* above.

Let us now show how to turn π into an accepting run $\overline{\pi} = (\overline{s}_0,\overline{\mathbf{v}}_0)(\overline{s}_1,\overline{\mathbf{v}}_1)\ldots$ of $\mathcal{A}$ on $\overline{w}$. Let us recursively define an auxiliary function $f : \mathbb{N} \rightarrow \mathbb{N}$ as follows: $f(0) = 0$, and

$$f(i{+}1) = \begin{cases} f(i) - (end(j) - begin(j)) + 1 & \text{if } f(i)+1 = end(j) \text{ for some } j \text{ and for} \\ & \text{every } i' < i \text{ we have } f(i')+1 \neq end(j); \\ f(i) + 1 & \text{otherwise.} \end{cases}$$

The run $\overline{\pi}$ is inductively defined as follows. The initial configuration is the same as π (i.e., $(\overline{s}_0,\overline{\mathbf{v}}_0) = (s_0,\mathbf{v}_0)$). For every $i \geq 0$, configuration $(\overline{s}_{i+1},\overline{v}_{i+1})$ is obtained by applying transition $\delta_{f(i)}$ to configuration $(\overline{s}_i,\overline{v}_i)$ (and thus we have $\overline{\sigma}_i = \sigma_{f(i)}$). By the definition of f we have that $\overline{s}_i = s_{f(i)}$ for every $i \in \mathbb{N}$. Moreover, it is easy to prove that $\overline{\sigma}_0\overline{\sigma}_1\ldots = \overline{w}$. Thus, in order to prove that $\overline{\pi}$ is an accepting run of $\mathcal{A}$ on $\overline{w}$, it remains to show the existence of a set $\overline{\mathbf{\Omega}}$ that satisfies, with respect to $\overline{\pi}$, conditions *(a)*-*(e)* above. For each $n \in \{1,\ldots,N\}$, let $\overline{\Omega}_{n,0},\overline{\Omega}_{n,1},\ldots$ be the sets defined by the following iterative procedure starting at $z = 0$ (let $processed_n$ be set initially to the empty set):

1. if $z \in processed_n$ (i.e., $\overline{\Omega}_{n,z}$ has been already defined in some previous iteration—this can happen in step 4 below), then do nothing and restart from step 1 with z updated to $z+1$;
2. if $\Omega_{n,z} \notin \mathbf{\Omega}$, then we jump to step 1 with z updated to $z+1$;
3. if there exists an infinite sequence of pairs $(x_1,y_1),(x_2,y_2)\ldots$ in $\Omega_{n,z}$ for which it holds, for every $j \in \mathbb{N}_{>0}$:
   - $y_j < x_{j+1}$ and
   - let $i_j^b = \max_{i \in \mathbb{N}_{>0}, begin(i) \leq x_j} i$ and $i_j^e = \min_{i \in \mathbb{N}_{>0}, end(i) > y_j} i$; we have that for every $i_j^b < i < i_j^e$ all the transitions $\delta_{begin(i)}\ldots\delta_{end(i)-1}$ are not inc_n transitions,

then let ∇ be one of them. We put $\overline{\Omega}_{n,z} = \{(\overline{x}_j, \overline{y}_j) \mid (x_j, y_j) \in \nabla, \overline{x}_j = \max_{f(i)=x_j} f(i), \overline{y}_j = \min_{f(i)=y_j} f(i)\}$, we set $\mathit{processed}_n = \mathit{processed}_n \cup \{z\}$, and we jump to step 1 with z updated to $z+1$;

4. if none of the above conditions applies, then there exists an infinite sequence of pairs $(x_1, y_1), (x_2, y_2) \ldots$ in $\Omega_{n,z}$ for which it holds, for every $j \in \mathbb{N}_{>0}$:

   (i) $y_j < x_{j+1}$ and

   (ii) let $i^b_j = \max_{i \in \mathbb{N}_{>0}, \mathit{begin}(i) \leq x_j} i$ and $i^e_j = \min_{i \in \mathbb{N}_{>0}, \mathit{end}(i) > y_j} i$; we have that there exists $i^b_j < i < i^e_j$ and an index k such that $\mathit{begin}(i) \leq k < \mathit{end}(i)$ and the transition δ_k is an inc_n transition (notice that $x_j < \mathit{begin}(i) < \mathit{end}(i) \leq y_j$ holds).

   Since for every $j \in \mathbb{N}_{>0}$ we have exactly z inc_n transitions between x_j and y_j we may find a sequence $\nabla = (x_1, y_1), (x_2, y_2) \ldots$ in $\Omega_{n,z}$ that also satisfies the following condition (besides conditions (i)-(ii) above):

   (iii) there exist z' many values $k_1, \ldots, k_{z'}$, with $1 \leq z' \leq z$, such that
   - $k_{j'} > 0$ for every j',
   - $\sum_{1 \leq j' \leq z'} k_{j'} \leq z$,
   - for every $j \in \mathbb{N}_{>0}$ there exist exactly z' many indexes $i_1, \ldots, i_{z'}$, for which, for every $j' \in \{1, \ldots, z'\}$, *(a)* $i^b_j < i_{j'} < i^e_j$, *(b)* there is an index $h \in \{\mathit{begin}(i_{j'}), \ldots, \mathit{end}(i_{j'}) - 1\}$ for which the transition δ_h is an inc_n transition, and *(c)* $|\{h \in \mathbb{N} \mid \mathit{begin}(i_{j'}) \leq h < \mathit{end}(i_{j'}),\ \delta_h \text{ is an } \mathit{inc}_n \text{ transition}\}| = k_{j'}$.

   Now, it is important to observe that in the portions of $\overline{\pi}$ delimited by the pairs $(x_j, y_j) \in \nabla$, the value $\overline{z} = z + \sum_{1 \leq j' \leq z'} k_{j'}$ is checked for counter n (and thus the "new" value $\overline{z}$ for counter n is still checked infinitely many times). Then, we put $\overline{\Omega}_{n,\overline{z}} = \{(\overline{x}_j, \overline{y}_j) \mid (x_j, y_j) \in \nabla, \overline{x}_j = \max_{f(i)=x_j} f(i), \overline{y}_j = \min_{f(i)=y_j} f(i)\}$, we set $\mathit{processed}_n = \mathit{processed}_n \cup \{\overline{z}\}$ (notice that, in this case, we are processing $\overline{z}$ while inside iteration $z < \overline{z}$), and we jump to step 1 with z updated to $z+1$.

We can now collect these sets into $\overline{\mathbf{\Omega}}$, which is defined as:

$$\overline{\mathbf{\Omega}} = \{\overline{\Omega}_{n,z} \mid 1 \leq n \leq N, z \in \mathbb{N}, z \in \mathit{processed}_n\}.$$

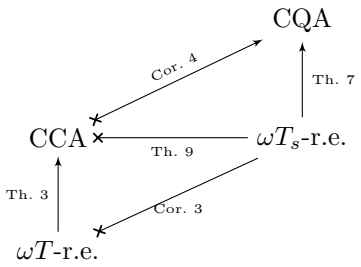
Since $\overline{\mathbf{\Omega}}$ satisfies, with respect to $\overline{\pi}$, conditions *(a)-(e)* above, $\overline{\pi}$ is an accepting run of $\mathcal{A}$ on $\overline{w}$, which implies $\overline{w} \in \mathcal{L}(\mathcal{A})$, thus yielding a contradiction. □

Corollary 3. *The ωT_s-regular language $L = (a^{T_s} b)^\omega$ is not recognized by any ωT-regular language.*

Proof. The claim immediately follows from Theorems 3 and 9. □

Corollary 4. *CQA are strictly more expressive than CCA.*

Proof. The claim immediately follows from Theorems 7, 8, and 9. □



Legend:

A → B denotes "B is at least as expressive as B" ($A \preceq B$)

A ↛ B denotes "it is not the case that $A \preceq B$" ($A \not\preceq B$)

A ↛→ B denotes "B is strictly more expressive than A", i.e., $A \preceq B$ and $B \not\preceq A$ ($A \prec B$)

Expressiveness results:
ωT-r.e. $\preceq$ CCA (Th. 3)
ωT_s-r.e. $\preceq$ CQA (Th. 7)
CCA $\preceq$ CQA (Th. 8)
ωT_s-r.e. $\not\preceq$ CCA (Th. 9)
ωT_s-r.e. $\not\preceq$ ωT-r.e. (Cor. 3)
CCA $\prec$ CQA (Cor. 4)

Figure 17: Summary of expressiveness results.

A summary of known expressiveness results is given in Figure 17. It remains open the question whether ωT-regular languages (resp., CCA) can be encoded by ωT_s-regular languages.

8. Conclusions

In this paper, we introduced a new class of extended ω-regular languages (ωT-regular languages), that captures meaningful languages not belonging to the class of ωBS-regular ones. We first gave a characterization of them in terms of ωT-regular expressions. Then, we defined a new class of automata, called counter-check automata (CCA), whose emptiness problem can be decided in PTIME, and we proved that CCA are expressive enough to capture ωT-regular languages (whether or not ωT-regular languages are expressively complete with respect to CCA is still an open problem). Finally, we provided an embedding of ωT-regular languages in S1S+U.

In the exploration of the space of possible extensions of ω-regular languages, we studied also a stronger variant of $(.)^T$, denoted by $(.)^{T_s}$, that forces ω-words to feature infinitely many exponents, *all of them* occurring infinitely often. To a large extent, the results obtained for $(.)^T$ can be replicated for $(.)^{T_s}$. In particular, it is possible to introduce a new class of automata, called counter-queue automata (CQA), that generalize CCA, whose emptiness problem can be proved to be decidable in 2ETIME and which are expressive enough to capture ω-regular languages extended with $(.)^{T_s}$. As in the case of ωT-regular languages, the problem of establishing whether or not the new languages are expressively complete with respect to CQA is open. There are, however, at least two significant differences between$(.)^T$ and $(.)^{T_s}$. First, $(.)^T$ satisfies the property of *prefix independence* (both $(.)^B$ and $(.)^S$ satisfy it), while this is not the case with $(.)^{T_s}$. The second difference is that there seems to be no way to generalize the embedding of ωT-regular languages into S1S+U to ωT_s-regular ones.

As for future work, we would like to investigate different combinations of $(.)^B$ and $(.)^S$ with $(.)^T$ (resp., $(.)^{T_s}$). A first and natural issue is the one about

closure of ωBST-regular languages under complementation. To give a negative answer to this question (in analogy to the case of ωBS-regular languages), it suffices to show that the ωBST-regular languages live at the first level of the projective hierarchy (analytic sets). Indeed, if they were closed under complementation, they would be expressively complete for S1S+U (in fact, this last statement already holds for ωBS-regular languages as a consequence of [9, Definition 4.1, Fact 4.2]), and it is known from [19] that S1S+U makes it possible to define languages that are complete for arbitrary levels of the projective hierarchy. Another particularly interesting issue is the one about the intersections of ωB-, ωS-, and ωT- (resp. ωT_s-) regular languages. In [20], it has been shown that a language which is both ωB- and ωS-regular is also ω-regular. We aim at providing a characterization of languages which are both ωB- (resp., ωS-) and ωT-/ωT_s-regular. Finally, we are interested in (modal) temporal logic counterparts of extended ω-regular languages. To the best of our knowledge, none was provided in the literature. We started to fill such a gap in [21, 22].

Acknowledgements. We would like to thank Gabriele Puppis for many useful discussions and Massimo Benerecetti for some helpful comments on the relationships among *-, B-, S-, and T-constructors.

References

[1] D. Della Monica, A. Montanari, P. Sala, Beyond ωBS-regular languages: ωT-regular expressions and counter-check automata, in: P. Bouyer, A. Orlandini, P. San Pietro (Eds.), Proc. of the 8th International Symposium on Games, Automata, Logics and Formal Verification (GandALF), Vol. 256 of EPTCS, Roma, Italy, 2017, pp. 223–237. doi:10.4204/EPTCS.256.16.

[2] D. Barozzini, D. Della Monica, A. Montanari, P. Sala, Counter-queue automata with an application to a meaningful extension of omega-regular languages, in: Joint Proceedings of the 18th Italian Conference on Theoretical Computer Science (ICTCS) and the 32nd Italian Conference on Computational Logic (CILC), Vol. 1949 of CEUR Workshop Proceedings, CEUR-WS.org, 2017, pp. 27–38.

[3] J. R. Büchi, On a decision method in restricted second order arithmetic, in: Proc. of the 1960 Int. Congress on Logic, Methodology and Philosophy of Science, 1962, pp. 1–11.

[4] R. McNaughton, Testing and generating infinite sequences by a finite automaton, Information and Control 9 (5) (1966) 521–530. doi:10.1016/S0019-9958(66)80013-X.

[5] C. C. Elgot, M. O. Rabin, Decidability and undecidability of extensions of second (first) order theory of (generalized) successor, J. Symb. Log. 31 (2) (1966) 169–181. doi:10.1002/malq.19600060105.

[6] W. Thomas, Automata on infinite objects, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), MIT Press, 1990, pp. 133–192.

[7] M. Bojańczyk, Weak MSO with the unbounding quantifier, Theory of Computing Systems 48 (3) (2011) 554–576. doi:10.1007/s00224-010-9279-2.

[8] M. Bojańczyk, T. Colcombet, Bounds in ω-regularity, in: LICS, 2006, pp. 285–296. doi:10.1109/LICS.2006.17.

[9] M. Bojańczyk, T. Colcombet, Boundedness in languages of infinite words, Logical Methods in Computer Science 13 (4:3) (2017) 1–54.

[10] M. Bojańczyk, S. Toruńczyk, Deterministic automata and extensions of weak MSO, in: R. Kannan, K. N. Kumar (Eds.), Proc. of the 29th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Vol. 4 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009, pp. 73–84. doi:10.4230/LIPIcs.FSTTCS.2009.2308.

[11] R. Alur, T. A. Henzinger, Finitary fairness, ACM Trans. Program. Lang. Syst. 20 (6) (1998) 1171–1194. doi:10.1145/295656.295659.

[12] O. Kupferman, N. Piterman, M. Y. Vardi, From liveness to promptness, Formal Methods in System Design 34 (2) (2009) 83–103. doi:10.1007/s10703-009-0067-z.

[13] D. Della Monica, A. Montanari, A. Murano, P. Sala, Prompt interval temporal logic, in: JELIA, Vol. 10021 of LNCS, Springer, 2016, pp. 207–222. doi:10.1007/978-3-319-48758-8_14.

[14] M. Bojańczyk, A bounding quantifier, in: CSL, Vol. 3210 of LNCS, Springer, 2004, pp. 41–55. doi:10.1007/978-3-540-30124-0_7.

[15] M. Bojańczyk, P. Parys, S. Toruńczyk, The MSO+U theory of (N, $<$) is undecidable, in: STACS, Vol. 47 of LIPIcs, 2016, pp. 21:1–21:8. doi:10.4230/LIPIcs.STACS.2016.21.

[16] D. Della Monica, A. Montanari, P. Sala, Beyond ωBS-regular languages: The class of ωT-regular languages, research Report 2017/01, Dept. of Mathematics, Computer Science, and Physics, University of Udine, Italy (2017).
URL https://users.dimi.uniud.it/~angelo.montanari/res_rep2017_1.pdf

[17] L. Breveglieri, A. Cherubini, C. Citrini, S. Crespi-Reghizzi, Multi-pushdown Languages and Grammars, Int. J. of Foundations of Computer Science 7 (03) (1996) 253–291.

[18] M. F. Atig, B. Bollig, P. Habermehl, Emptiness of multi-pushdown automata is 2ETIME-complete, in: Developments in Language Theory, Springer, 2008, pp. 121–133.

[19] S. Hummel, M. Skrzypczak, The topological complexity of MSO+U and related automata models, Fundam. Inform. 119 (1) (2012) 87–111. doi:10.3233/FI-2012-728.

[20] M. Skrzypczak, Separation property for ωB- and ωS-regular languages, Logical Methods in Computer Science 10 (1). doi:10.2168/LMCS-10(1:8)2014.

[21] A. Montanari, P. Sala, Adding an equivalence relation to the interval logic $AB\overline{B}$: complexity and expressiveness, in: LICS, IEEE Computer Society, 2013, pp. 193–202. doi:10.1109/LICS.2013.25.

[22] A. Montanari, P. Sala, Interval logics and ωB-regular languages, in: LATA, Vol. 7810 of LNCS, Springer, 2013, pp. 431–443.