RAIRO-Theor. Inf. Appl. 46 (2012) 461–478
DOI: 10.1051/ita/2012021

Available online at:
www.rairo-ita.org

k-COUNTING AUTOMATA

Joël Allred[1] and Ulrich Ultes-Nitsche[1]

Abstract. In this paper, we define k-counting automata as recognizers for ω-languages, *i.e.* languages of infinite words. We prove that the class of ω-languages they recognize is a proper extension of the ω-regular languages. In addition we prove that languages recognized by k-counting automata are closed under Boolean operations. It remains an open problem whether or not emptiness is decidable for k-counting automata. However, we conjecture strongly that it is decidable and give formal reasons why we believe so.

Mathematics Subject Classification. 68Q45, 20F10.

1. Introduction

In this paper, we study ω-languages [18], *i.e.* languages of infinite word-length. These languages play an important role when modelling reactive (non-terminating) systems. In practice, the most important class of ω-languages is the class of regular ω-languages (*cf.* [18]). They can be defined by an ω-regular expression [5, 18], accepted by a Büchi-automaton [5], a Muller-automaton [15], or any other equivalent formalism. A practically important property of regular ω-languages is that they are closed under Boolean operations: given regular ω-languages, their intersection, union, and complement is also ω-regular. In addition, regular ω-languages can be tested effectively for emptiness, *i.e.* whether or not the ω-language described by one of the formalisms that represent regular ω-languages is the empty language (*i.e.* the empty set). As a consequence of the Boolean closure and the computable emptiness test, testing whether a regular ω-language is a subset of another regular

Keywords and phrases. ω-automata, extensions to regular ω-languages, closure under Boolean operations, emptiness problem, infinite hierarchy of ω-languages.

[1] Department of Computer Science, University of Fribourg, Boulevard de Pérolles 90, 1700 Fribourg, Switzerland, {joel.allred,uun}@unifr.ch.

Article published by EDP Sciences

ω-language is decidable, a result that is very important in linear-time temporal verification algorithms [1, 11, 20–22].

In this paper, we try to extend the class of regular ω-languages, still keeping the nice properties that regular ω-languages possess (Boolean closure, computable emptiness test). We define a new machine model for representing ω-languages that we call a k-counting automaton. k-counting automata are automata equipped with k-many counters. Their acceptance condition makes statements about the boundedness or unboundedness of the counters in infinite runs of the automaton. We use the name "k-*counting* automaton" rather than "k-*counter* automaton" to avoid confusing them with multi-counter machines [14] (cited after [7], cited after [9]). In multi-counter machines, the transition function depends on whether or not the counters are zero. Being able to perform a zero test makes them Turing-complete, and therefore any interesting property of languages recognized by multi-counter machines is undecidable (Rice's theorem [17], cited after [9]). In contrast, in k-counting automata, the transition function, even though it changes counter values, is independent of the counter values (k-counting automata "do not read" counters). This, in essence, makes them less powerful than Turing machines, and thus interesting properties can remain decidable.

We prove that ω-languages recognized by k-counting-automata properly extend the class of regular ω-languages: we show that all regular ω-languages can be recognized by k-counting automata, but k-counting automata can recognize ω-languages that are not regular. We also show the closure of ω-languages accepted by k-counting automata under Boolean operation. We conjecture that the emptiness test for k-counting automata is decidable. Even though we are not able to prove this fact, we at least give formal arguments that make us believe in the existence of a computable emptiness check.

To our knowledge, there exist very few works in the literature related to ω-automata extended by counters. We shall consider results by Bojańczyk and Colcombet [2, 3]. They also use a (un-)boundedness condition on counter values. Their approach, however, differs from k-counting automata introduced in this paper. Also the work by Fernau and Stiebe on blind counter automata on ω-words [6] is related to our approach. Finally, the problem we study is related to the star height problem for regular languages [8].

2. Preliminaries

We review here the notions of ω-languages and ω-automata up to the level needed for presenting the main results of this paper. Let Σ^* be the set of all (finitely long) words over an alphabet (finite set of symbols) Σ, and let Σ^ω be the set of all infinite words (aka. ω-words) over Σ. Each subset L of Σ^* is a language over Σ, and each subset L_ω of Σ^ω is an ω-language over Σ.

For an ω-word $x \in \Sigma^\omega$, let $pre(x) = \{w \in \Sigma^* \mid \exists y \in \Sigma^\omega : wy = x\}$ be the set of all finitely long prefixes of x.

An ω-language is called regular if and only if [13] it can be recognized, for instance, by a Muller automaton [15]. Such a Muller automaton $A = (Q, \Sigma, \delta, q_{in}, F)$ is a deterministic automaton and consists of:

- a finite set Q of states;
- a finite set Σ of symbols;
- a transition function $\delta : Q \times \Sigma \to Q$;
- an initial state $q_{in} \in Q$;
- a set F of sets of states (*i.e.* $F \subseteq 2^Q$).

A run of automaton A on ω-word $x = x_0x_1x_2 \ldots \in \Sigma^\omega$ is an infinite state sequence $r = r_0r_1r_2 \ldots \in Q^\omega$ such that $r_0 = q_{in}$ and $\forall i \geq 0 : \delta(r_i, x_i) = r_{i+1}$. For run r, let $\omega(r)$ designate its infinitely recurring states:

$$\omega(r) = \{q \in Q \mid \exists^\infty i \geq 0 : r_i = q\}.^2$$

Because Muller-automata are always deterministic[3] and can always be made complete[4], we can talk about *the* run of A on x and denote it by $r(A, x)$. Run $r(A, x)$ is called Muller-accepting, if and only if $\omega(r(A, x)) \in F$. The ω-language recognized by Muller automaton A is

$$L_\omega(A) = \{x \in \Sigma^\omega \mid \omega(r(A, x)) \in F\}.$$

When subsequently defining k-counting automata, we will refer to the set of all Boolean (*i.e.* propositional) formulas that can be constructed over a set P of atomic propositions using the notation $\mathbb{B}(P)$. By $[0, k]$ we designate the set $\{0, 1, \ldots, k\}$ of the first $k + 1$ non-negative integers.

3. k-Counting automata

k-counting automata are deterministic finite-state automata equipped with k-many counters. The counters do not control whether or not a transition can be taken (k-counting automata "do not read" the counters), but are used solely to define when an infinite run is accepting or not. If a transition $\delta(p, a) = (q, C_+, C_-)$ is taken (p, q being states, a being a symbol, and C_+, C_- being disjoint sets of counters), all counters in C_+ are increased by 1, all counters in C_- are decreased by 1, and all other not in C_+ or C_- are left unchanged. Runs of the automata are not sequences of states anymore, but sequences of state/counter-valuation pairs.

The acceptance condition of a k-counting automaton is a Boolean formula. It is constructed from atomic propositions c_+ (or c_-) that indicate whether in a run, for each integer m, counter c assumes eventually a value that is greater (or smaller) than m.

[2] Read "$\exists^\infty \ldots$" as "there exist infinitely many different ..."

[3] In contrast to Büchi automata, where the degree of non-determinism can only be reduced to 2 [19]. Therefore a run of a Muller-automaton on an ω-word is always unique, if it exists.

[4] Completeness together with determinism guarantees the existence of exactly one run on each ω-word.

Definition 3.1 (k-counting automaton)**.** We define a k-counting automaton $A = (Q, C, \Sigma, \delta, q_{in}, \Phi)$ to consist of

- a finite set Q of states;
- a finite set C of k-many counters (*i.e.* $\mid C \mid = k$);
- a finite set Σ of symbols;
- a transition function $\delta : Q \times \Sigma \to Q \times 2^C \times 2^C$;
- an initial state $q_{in} \in Q$;
- an acceptance condition $\Phi \in \mathbb{B}(\{c_+ \mid c \in C\} \cup \{c_- \mid c \in C\})$,

such that, for all $p \in Q$ and $a \in \Sigma$, if $\delta(p, a) = (q, C_+, C_-)$, then $C_+ \cap C_- = \emptyset$.

Definition 3.2 (run)**.** Let V be the set of all counter valuations (assignments of integers to counters). For $v \in V$, let $v(c) \in \mathbb{Z}$ denote the value of counter c under counter valuation v.

A run $r \in (Q \times V)^\omega$ of A on ω-word $x = x_0 x_1 x_2 \ldots \in \Sigma^\omega$ is a sequence of state/counter-valuation pairs

$$(q_0, v_0)(q_1, v_1)(q_2, v_2) \ldots,$$

such that

$$q_0 = q_{in},$$

$$\forall i, 0 \leq i \leq k-1 : v_0(c_i) = 0,$$

and, for all $j \geq 0$,

$$\delta(q_j, x_j) = (q_{j+1}, C_+, C_-)$$

and

$$\forall i \in C_+ : v_{j+1}(c_i) = v_j(c_i) + 1,$$

$$\forall i \in C_- : v_{j+1}(c_i) = v_j(c_i) - 1,$$

$$\forall i \in C \setminus (C_+ \cup C_-) : v_{j+1}(c_i) = v_j(c_i).$$

For a run to be accepting, the counter values in that run may or may not grow (positively or negatively) beyond any fixed number (the atomic proposition considered in well-formed satisfaction conditions are making such statements about counter values). Boolean formulas on the (un-)boundedness of counters define whether or not a run is accepting:

Definition 3.3 (accepting run)**.** For set $\{c_+ \mid c \in C\} \cup \{c_- \mid c \in C\}$ of atomic propositions, we define the satisfaction of atomic propositions by run

$$r = (q_0, v_0)(q_1, v_1)(q_2, v_2) \ldots$$

as follows. For all $c \in C$,

$$r \models c_+ \iff \forall m \in \mathbb{Z} : \exists j > 0 : v_j(c) > m$$

and

$$r \models c_- \iff \forall m \in \mathbb{Z} : \exists j > 0 : v_j(c) < m.$$

Based on the satisfaction relation for the atomic propositions, and using the standard semantics of Boolean connectives, run r is called accepting if and only if

$$r \models \Phi.$$

Note that the satisfaction condition for atomic propositions given in the above definition implies

$$r \models \neg c_+ \iff \exists m \in \mathbb{Z} : \forall j > 0 : v_j(c) \leq m$$

and

$$r \models \neg c_- \iff \exists m \in \mathbb{Z} : \forall j > 0 : v_j(c) \geq m.$$

So the satisfaction of an atomic proposition states that a counter is positively unbounded (subscript "+") or negatively unbounded (subscript "−"), and the satisfaction of a negated atomic proposition states that a counter is positively bounded (subscript "+") or negatively bounded (subscript "−").

Definition 3.4 (accepted ω-language)**.** An ω-word x is accepted by A if and only if there exists an accepting run of A on x. The ω-language accepted by A is the set containing all accepted ω-words.

4. Beyond ω-regularity

We show in this section that the class of regular ω-languages can be recognized by k-counting automata where the contrary is not true: there exist non-regular ω-languages that are recognizable by k-counting automata. Therefore the class of ω-languages recognizable by k-counting automata properly extends the regular ω-languages.

Lemma 4.1. *A regular ω-language L_ω can be recognized by a k-counting automaton, for some $k > 0$.*

Proof. Let $A = (Q, \Sigma, \delta, q_{in}, F)$ be a deterministic Muller automaton recognizing L_ω. We number the states in Q from 0 to $|Q| - 1$. Let $n(q)$ be state q's number.

Let $A' = (Q, \{c_0, c_1, \ldots, c_{|Q|-1}\}, \Sigma, \delta', q_{in}, \Phi)$ be a $|Q|$-counting automaton with

$$\delta'(p, a) = (q, \{c_{n(q)}\}, \emptyset) \text{ if and only if } \delta(p, a) = q, \text{ for all } p \in Q, a \in \Sigma,$$

and

$$\Phi = \bigvee_{\phi \in F} \Big(\bigwedge_{q \in \phi} c_{n(q)_+} \Big) \wedge \Big(\bigwedge_{q \in Q \setminus \phi} \neg c_{n(q)_+} \Big).$$

Then A' recognizes L_ω. □

The next lemma shows that even a single counter is sufficient to recognize ω-languages that are not regular:

Lemma 4.2. *There exist non-regular ω-languages that can be recognized by a* 1*-counting automaton.*

Proof. Let $A = (\{q_{in}\}, \{c\}, \{a, b\}, \delta, q_{in}, \neg c_+ \wedge \neg c_-)$ be a 1-counting automaton with

$$\delta(q_{in}, a) = (q_{in}, \{c\}, \emptyset),$$
$$\delta(q_{in}, b) = (q_{in}, \emptyset, \{c\}).$$

We show that the ω-language L_ω recognized by A is not regular. A is defined in such a way that, for all ω-words $x \in L_\omega$, there exists $m > 0$ such that in all prefixes w of x, the number of occurrences of a and the number of occurrences of b in w cannot differ by more than m. Readers who see this fact immediately can skip the next two paragraphs in which we prove formally that ω-words in L_ω have the mentioned structure.

For $w \in \{a, b\}^*$, let $[w]_a$ be the number of occurrences of symbol a in w, and let $[w]_b$ be the number of occurrences of symbol b in w. Then

$$L_\omega = \{x \in \{a, b\}^\omega \mid \exists m > 0 : \forall w \in pre(x) : |[w]_a - [w]_b| < m\}.$$

This observation is true, because an accepting run $(q_0, v_0)(q_1, v_1)(q_2, v_2)\ldots$ must satisfy $\neg c_+ \wedge \neg c_-$, indicating that there exist $m_+ > 0$ and $m_- < 0$ such that $\forall i > 0 : v_i(c) < m_+ \wedge v_i(c) > m_-$. Taking $m = \max\{|m_+|, |m_-|\}$, we get that $|v_i(c)| < m$, for all $i > 0$. Because c is increased when reading a and decreased when reading b, $v_i(c) = [w_i]_a - [w_i]_b$, where w_i in the prefix of x of length $i + 1$.

We assume now that L_ω is regular and show a contradiction. If L_ω were regular, then it could be recognized by a deterministic Muller automaton M. Let m be the number of states of M, and consider ω-word $(a^m b^m)^\omega$ that is in L_ω. We will show that if M accepts $(a^m b^m)^\omega$, then it will also accept ω-words that are not in L_ω, contradicting the assumption that M accepts L_ω.

The proof idea is pretty simple, using the pigeon-hole principle: whenever M reads the sub-words a^m, it will visit $m + 1$-many states (including the state M starts in when reading a^m). Because M has only m-many states, it must visit a state twice in such a run. Therefore, the recurring state could be revisited twice in a run of M, not changing the acceptance of an ω-word containing as many consecutive symbols a as can be read by repeating the run from the recurring state to itself twice. Hence M also accepts $a^{n_0} b^m a^{n_1} b^m \ldots$, where all n_i are greater than m. Such an ω-word is, however, not accepted by A. In the remainder of this proof, we formalize this argument.

Assume M accepts $(a^m b^m)^\omega$. Because M is deterministic there exists a unique accepting run $r = q_0 q_1 q_2 \ldots$ of M on $(a^m b^m)^\omega$. We subdivide r into infinitely many finitely long runs. For $i \geq 0$, let

$$r_i = q_{im} q_{im+1} \ldots q_{im+m}.$$

Note that for $i = 2j$ (*i.e.* for an even i), r_i is the sub-run of r that M executes while reading the $(j+1)$st occurrence of sub-word a^m in $(a^m b^m)^\omega$, and that for $i = 2j+1$ (*i.e.* for an odd i), r_i is the sub-run of r that M executes while reading the $(j+1)$st occurrence of sub-word b^m in $(a^m b^m)^\omega$, for all $j \geq 0$.

For consecutive sub-runs r_i and r_{i+1}, let $r_i \times r_{i+1}$ be their concatenation where the last state of r_i is omitted (because it is already the first state of r_{i+1}), *i.e.*

$$r_i \times r_{i+1} = q_{im} q_{im+1} \cdots q_{im+m-1} q_{(i+1)m} q_{(i+1)m+1} \cdots q_{(i+2)m}.$$

Then, by extending "$\times$" to already concatenated sub-runs, we get

$$r = r_0 \times r_1 \times r_2 \times \ldots$$

We are considering now the sub-words a^m, and the sub-runs r_i, for even i, that M executes while reading the sub-words a^m. Because M has m-many states, but the r_i contain $(m+1)$-many states, at least one state occurs twice in r_i (pigeon-hole principle). Let g_i and h_i, $m \geq h_i > g_i \geq 0$, be chosen such that $q_{im+g_i} = q_{im+h_i}$, and let $n_i = h_i - g_i$.

We define

$$r'_i = q_{im} \cdots q_{im+g_i} q_{im+g_i+1} \cdots q_{im+h_i} q_{im+g_i+1} \cdots q_{im+h_i} q_{im+h_i+1} \cdots q_{im+m},$$

i.e. r'_i is identical to r_i, except for the fact that the cycle from q_{im+g_i} to q_{im+h_i} is duplicated in r'_i. Because while following r_i, M can read a^m, while following r'_i, M can read a^{m+n_i} with $n_i > 0$. Then

$$r' = r'_0 \times r_1 \times r'_2 \times r_3 \times r'_4 \times r_5 \times \ldots$$

is a run of M on

$$x' = a^{m+n_0} b^m a^{m+n_2} b^m a^{m+n_4} b^m \ldots$$

with $n_i > 0$ for all $i \geq 0$. x' is therefore not in L_ω. However, in r' when compared to r, the number of occurrences of each state is at most doubled, having no influence on whether a state occurs finitely or infinitely often. Therefore

$$\omega(r) = \omega(r'),$$

and because r is an accepting run of M on $(a^m b^m)^\omega$, r' is an accepting run of M on x'. Therefore M accepts x', contradicting our assumption that M recognizes L_ω. □

5. Completeness

To prove the closure of ω-languages recognizable by k-counting-automata under Boolean operations, it will be convenient that the automaton is complete, *i.e.* for all states p and all symbols a, $\delta(p, a)$ is defined. We show in this section that each incomplete k-counting automaton can be made complete by adding one state and one counter to it.

Lemma 5.1. *Let $A = (Q, C, \Sigma, \delta, q_{in}, \Phi)$ be an incomplete k-counting automaton recognizing L_ω. There exists a complete $(k+1)$-counting automaton recognizing L_ω.*

The proof is a standard construction using an added trap state t.

Proof. Let $t \notin Q$ and $c \notin C$. Then let $A' = (Q \cup \{t\}, C \cup \{c\}, \Sigma, \delta', q_{in}, \Phi \wedge \neg c_+)$ such that, for all $p \in Q \cup \{t\}$ and $a \in \Sigma$,

$$\delta'(p,a) = \begin{cases} \delta(p,a) & \text{if } \delta(p,a) \text{ is defined,} \\ (t, \{c\}, \emptyset) & \text{if } p = t \text{ or } \delta(p,a) \text{ is undefined.} \end{cases}$$

We show that A' recognizes L_ω. Let $x \in L_\omega$. Each accepting run r of A on x is also a run of A' on x. Note that $r \models \Phi$. Because in r the new counter c in A' is not changed, we get $r \models \neg c_+$, implying $r \models \Phi \wedge \neg c_+$, which itself implies that r is an accepting run of A' on x.

On the contrary, let $x \in \Sigma^\omega \setminus L_\omega$. Then there exists no accepting run of A on x. Therefore, either there exists a run r of A on x such that $r \not\models \Phi$ or no run of A on x exists.

In the first case, r is also a run of A' on x, and because $r \not\models \Phi$, also $r \not\models \Phi \wedge c_+$. Hence A' does not accept x.

In the second case, there will be a run r' of A' on x, because A' is complete. However, since there is no run of A on x, r' will visit state t and will remain trapped there. Because revisiting t will each time increase the new counter c, $r' \models c_+$. Therefore $r' \not\models \neg c_+$ and thus $r' \not\models \Phi \wedge \neg c_+$. So r' is not an accepting run of A' on x. Hence also in this case A' does not accept x.

Therefore A' accepts exactly those ω-words that A accepts and consequently recognizes L_ω. □

6. Boolean closure of k-counting automata

In this section, we show that the class of ω-languages recognized by k-counting automata is closed under Boolean operations. From the previous section it follows that we can assume that the automata are complete.

Lemma 6.1. *Let $A_1 = (Q_1, C_1, \Sigma_1, \delta_1, q_{in_1}, \Phi_1)$ and $A_2 = (Q_2, C_2, \Sigma_2, \delta_2, q_{in_2}, \Phi_2)$ be a complete k_1-counting and a complete k_2-counting automaton, respectively, recognizing L_{ω_1} and L_{ω_2}, respectively. There exists a complete $(k_1 + k_2)$-counting automaton recognizing $L_{\omega_1} \cap L_{\omega_2}$.*

The proof is a standard product construction of automata compatible with the acceptance relation for k-counting automata:

Proof. We assume $C_1 \cap C_2 = \emptyset$ (this can always be achieved by, for instance, renaming the counters in A_2). We set

$$A_{A_1 \cap A_2} = (Q_1 \times Q_2, C_1 \cup C_2, \Sigma_1 \cap \Sigma_2, \delta, (q_{in_1}, q_{in_2}), \Phi_1 \wedge \Phi_2)$$

with, for all $p_1 \in Q_1$, $p_2 \in Q_2$, and $a \in \Sigma_1 \cap \Sigma_2$,

$$\delta((p_1, p_2), a) = ((q_1, q_2), C_{1_+} \cup C_{2_+}, C_{1_-} \cup C_{2_-})$$

if and only if

$$\delta_1(p_1, a) = (q_1, C_{1_+}, C_{1_-})$$

and

$$\delta_2(p_2, a) = (q_2, C_{2_+}, C_{2_-}).$$

Let L_ω be the ω-language accepted by $A_{A_1 \cap A_2}$. We show $L_\omega = L_{\omega_1} \cap L_{\omega_2}$.

"$L_\omega \subseteq L_{\omega_1} \cap L_{\omega_2}$": let $x \in L_\omega$. Then there exists an accepting run

$$r = ((p_0, q_0), v_0)((p_1, q_1), v_1)((p_2, q_2), v_2) \dots$$

of $A_{A_1 \cap A_2}$ on x. Because r is an accepting run, we have

$$r \models \Phi_1 \wedge \Phi_2. \tag{6.1}$$

For each of the v_i, $i \geq 0$, we set $v_{i_1} = v_i \cap C_1$ and $v_{i_2} = v_i \cap C_2$. If we restrict r to counters in C_1, we get run

$$r_1 = (p_0, v_{0_1})(p_1, v_{1_1})(p_2, v_{2_1}) \dots$$

for which, because of (6.1) and the fact that the v_{i_1} extract the counters of C_1 from r, we get $r_1 \models \Phi_1$. Therefore r_1 is an accepting run of A_1 on x. With a similar argument, we get that

$$r_2 = (q_0, v_{0_2})(q_1, v_{1_2})(q_2, v_{2_2}) \dots$$

is an accepting run of A_2 on x. Therefore $x \in L_{\omega_1} \cap L_{\omega_2}$.

"$L_{\omega_1} \cap L_{\omega_2} \subseteq L_\omega$": let $x \in L_{\omega_1}$ and let $x \in L_{\omega_2}$. Then there exists an accepting run

$$r_1 = (p_0, v_{0_1})(p_1, v_{1_1})(p_2, v_{2_1}) \dots$$

of A_1 on x and an accepting run

$$r_2 = (q_0, v_{0_2})(q_1, v_{1_2})(q_2, v_{2_2}) \dots$$

of A_2 on x. Then, by construction,

$$r = ((p_0, q_0), v_{0_1} \cup v_{0_2})((p_1, q_1), v_{1_1} \cup v_{1_2})((p_2, q_2), v_{2_1} \cup v_{2_2}) \dots$$

is a run of $A_{A_1 \cap A_2}$ on x. Because $r_1 \models \Phi_1$, we get $r \models \Phi_1$, and because $r_2 \models \Phi_2$, we get $r \models \Phi_2$. So $r \models \Phi_1 \wedge \Phi_2$ and is therefore an accepting run of $A_{A_1 \cap A_2}$ on x. Thus $x \in L_\omega$. □

In the next step, we prove the closure of the class of ω-languages accepted by k-counting automata under union. We assume that the involved automata use the same alphabet (it is always possible to augment an alphabet, resulting in an incomplete automaton that then can be made complete by adding one state and one counter as discussed in the previous section).

Lemma 6.2. *Let $A_1 = (Q_1, C_1, \Sigma, \delta_1, q_{in_1}, \Phi_1)$ and $A_2 = (Q_2, C_2, \Sigma, \delta_2, q_{in_2}, \Phi_2)$ be a complete k_1-counting and a complete k_2-counting automaton, respectively, recognizing L_{ω_1} and L_{ω_2}, respectively. There exists a complete $(k_1 + k_2)$-counting automaton recognizing $L_{\omega_1} \cup L_{\omega_2}$.*

Proof. We assume, as in the previous proof, that $C_1 \cap C_2 = \emptyset$. Let $A_{A_1 \cup A_2} = (Q_1 \times Q_2, C_1 \cup C_2, \Sigma, \delta, (q_{in_1}, q_{in_2}), \Phi_1 \vee \Phi_2)$ with, for all $p_1 \in Q_1$, $p_2 \in Q_2$, and $a \in \Sigma$,

$$\delta((p_1, p_2), a) = ((q_1, q_2), C_{1_+} \cup C_{2_+}, C_{1_-} \cup C_{2_-})$$

if and only if

$$\delta_1(p_1, a) = (q_1, C_{1_+}, C_{1_-})$$

and

$$\delta_2(p_2, a) = (q_2, C_{2_+}, C_{2_-}).$$

Let L_ω be the ω-language recognized by $A_{A_1 \cup A_2}$. From the construction it follows immediately that, if $x \in L_{\omega_1}$ or $x \in L_{\omega_2}$, then $x \in L_\omega$. Therefore, we must only show that $L_\omega \subseteq (L_{\omega_1} \cup L_{\omega_2})$.

Let $x \in L_\omega$. Then there exists an accepting run

$$r = ((p_0, q_0), v_0)((p_1, q_1), v_1)((p_2, q_2), v_2) \dots$$

of $A_{A_1 \cup A_2}$ on x. Because r is an accepting run, we have

$$r \models \Phi_1 \vee \Phi_2.$$

For each of the v_i, $i \geq 0$, we set $v_{i_1} = v_i \cap C_1$ and $v_{i_2} = v_i \cap C_2$. If we restrict r to counters in C_1 and C_2, respectively, we get runs

$$r_1 = (p_0, v_{0_1})(p_1, v_{1_1})(p_2, v_{2_1}) \dots$$
$$r_2 = (q_0, v_{0_2})(q_1, v_{1_2})(q_2, v_{2_2}) \dots$$

of A_1 and A_2, for which we get $r_1 \models \Phi_1$ or $r_2 \models \Phi_2$, implying that r_1 or r_2 is an accepting run of A_1 or A_2 on x, respectively. Therefore $x \in L_{\omega_1} \cup L_{\omega_2}$. □

Finally, we prove that negating the acceptance condition suffices to complement complete k-counting automata.

Lemma 6.3. *Let $A = (Q, C, \Sigma, \delta, q_{in}, \Phi)$ be a complete k-counting automaton recognizing L_ω. There exists a k-counting automaton recognizing $\Sigma^\omega \setminus L_\omega$.*

Proof. Let $A_{\neg A} = (Q, C, \Sigma, \delta, q_{in}, \neg\Phi)$. We show that $A_{\neg A}$ recognizes $\Sigma^\omega \setminus L_\omega$. Let $x \in \Sigma^\omega$. Because A and thus $A_{\neg A}$ are complete and, except for the acceptance condition, identical, there exists a run r of A and $A_{\neg A}$ on x. If r is an accepting run of A, then $r \models \Phi$ and therefore $r \not\models \neg\Phi$, and thus r is not an accepting run of $A_{\neg A}$. If r is an accepting run of $A_{\neg A}$, then $r \models \neg\Phi$ and therefore $r \not\models \Phi$, and thus r is not an accepting run of A. So whenever A accepts x, $A_{\neg A}$ does not, and *vice versa*. □

The following corollary subsumes the lemmas given above (together with the lemma on complete automata):

Corollary 6.4. *The class of ω-languages recognized by k-counting-automata is closed under Boolean operations.*

7. An open problem: deciding emptiness

Language containment is a property that one frequently wants to decide for a given language class. In linear-time temporal verification, for instance, the satisfaction relation is based on language containment. If a language class is closed under Boolean operations, then deciding emptiness of a given language is sufficient for the containment test, using the following equivalence:

$$L_{\omega_1} \subseteq L_{\omega_2} \iff L_{\omega_1} \cap (\Sigma^\omega \setminus L_{\omega_2}) = \emptyset$$

where Σ is the set of symbols of L_{ω_1} and L_{ω_2}. We believe that testing emptiness is decidable for k-counting automata, but have not yet been able to prove it:

Conjecture 7.1. *Let $A = (Q, C, \Sigma, \delta, q_{in}, \Phi)$ be a k-counting automaton recognizing some L_ω. We believe that it is decidable whether or not $L_\omega = \emptyset$.*

We are going to analyse the problem of testing k-counting automata for emptiness in this section, presenting two approaches that are both candidates for an emptiness decision procedure.

When aiming to decide emptiness, it is probably a good attempt to turn an arbitrary acceptance condition Φ into an equivalent formula Φ' in disjunctive normal form, *i.e.* the disjunction of conjunctive clauses. Then a run r satisfies Φ' if and only if it satisfies at least one of the conjunctive clauses in Φ'. Hence, finding a decision procedure for acceptance conditions that are single conjunctive clauses is sufficient to get a general decision procedure for emptiness.

A conjunctive clause contains literals c_+, c_-, $\neg c_+$, and $\neg c_-$, for the different counters c. A sufficient condition for proving non-emptiness is the following: for

FIGURE 1. Example 1-counting automaton.

conjunctive clause ϕ, search for a reachable cyclic path[5] P in A such that the following conditions are all satisfied:

- for each c_+ in ϕ, there are more transitions in P that increase counter c than there are transitions decreasing it;
- for each c_- in ϕ, there are more transitions in P that decrease counter c than there are transitions increasing it;
- for each $\neg c_+$ in ϕ, there are not more transitions in P that increase counter c than there are transitions decreasing it;
- for each $\neg c_-$ in ϕ, there are not more transitions in P that decrease counter c than there are transitions increasing it.

If we find such a cyclic path P, then each run of A on an ω-word x that makes A start in the initial state, go to P, and then cycle through P forever, is an accepting run, and thus A is not empty.

This condition is, however, not a necessary condition. We can see this by the following example: let $A = (\{q\}, \{c\}, \{a, b\}, \delta, q, c_+ \wedge c_-)$ with $\delta(q, a) = (q, \{c\}, \emptyset)$ and $\delta(q, b) = (q, \emptyset, \{c\})$. The structure of A is depicted in Figure 1.

Whenever this automaton reads symbol a, the counter is increased, and whenever it reads symbol b, the counter is decreased. In addition, an ω-word x is accepted whenever, while reading x, the counter can assume arbitrarily large and arbitrarily small values (acceptance condition: $c_+ \wedge c_-$). Therefore in each ω-word that is accepted by A, there must be prefixes where, for any $m > 0$, there are at least m-more occurrences of a than there are occurrences of b (and vice versa). An ω-word accepted by this automaton is, for instance:

$$abbaaabbbbaaaaabbbbbbaaaaaaabbbbbbb\ldots,$$

which can be represented as

$$a^1b^2a^3b^4a^5b^6a^7b^8\ldots$$

Clearly the non-emptiness of A cannot be checked by finding a *single* cycle in which the same counter is strictly increased and strictly decreased while going once through that cycle as these are contradicting statements. However, there are

[5]By "path", we denote a sequence of consecutive transitions in A.

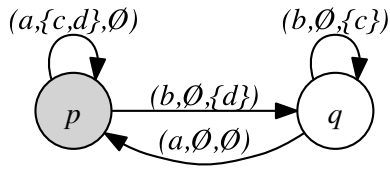

FIGURE 2. Example 2-counting automaton (p being the initial state).

two cycles in A, $q \xrightarrow{a} q$ in which c is strictly increased, and $q \xrightarrow{b} q$ in which c is strictly decreased, and these cycles are reachable from one another.

So it is possible to let c grow arbitrarily big – by cycling through the $q \xrightarrow{a} q$ cycle – and then make c become arbitrarily small – by cycling through the $q \xrightarrow{b} q$ cycle. So the acceptance condition can be satisfied and A is not empty.

7.1. SHORTEST-CYCLE INEQUALITIES

We believe that a generalization of the observation made in the last paragraph of the previous section suffices to handle all unbounded counters (*i.e.* those that are referred to in the acceptance condition by an unnegated atomic proposition): find a reachable strongly-connected component of A in which for each positively (negatively) unbounded counter a cycle increasing (decreasing) this counter exists. What is missing, however, is handling bounded counters (those that are referenced in the acceptance condition by a negated atomic proposition). We believe that it is possible to integrate bounded counters into the emptiness test by building a system of inequalities based on shortest cycles (*i.e.* cycles in which, by going from one state to itself, none of the other states appears twice)[6].

We explain this idea by using an example. Consider the 2-counting automaton in Figure 2 together with acceptance condition $c_+ \wedge c_- \wedge \neg d_+$. This automaton has three shortest cycles with the following effects on the counters:

- $p \xrightarrow{a} p$; c and d are both increased by one;
- $q \xrightarrow{b} q$; c is decreased by one, d is not changed;
- $p \xrightarrow{b} q \xrightarrow{a} p$; d is decreased by one, c is not changed[7].

All cycles are then compositions of *interconnected* shortest cycles:

- $p \xrightarrow{a} p$ on its own;
- $q \xrightarrow{b} q$ on its own;
- $p \xrightarrow{b} q \xrightarrow{a} p$ on its own;

[6]This idea has been refined during various discussions with Carine Poffet who has helped us clarifying several aspects of "cycle inequalities" and who has started working on the problem of effectively testing k-counting automata for emptiness.

[7]Note that we have not listed $q \xrightarrow{a} p \xrightarrow{b} q$ here, as it is a different representation of cycle $p \xrightarrow{b} q \xrightarrow{a} p$ (to write it down, we just simply start with state q rather than with state p).

- $p \xrightarrow{a} p$ combined with $p \xrightarrow{b} q \xrightarrow{a} p$;
- $q \xrightarrow{b} q$ combined with $p \xrightarrow{b} q \xrightarrow{a} p$;
- $p \xrightarrow{a} p$ combined with $q \xrightarrow{b} q$ combined with $p \xrightarrow{b} q \xrightarrow{a} p$.

Note that $p \xrightarrow{a} p$ cannot be combined with $q \xrightarrow{b} q$ because they are not interconnected ($p \xrightarrow{b} q \xrightarrow{a} p$ is needed to connect these shortest cycles). In the procedure that we envisage, all possible combinations of interconnected shortest cycles must be explored. We restrict ourselves in this example to the last case of combining all three shortest cycles as it will serve to illustrate our reasoning.

If we cycle through $p \xrightarrow{a} p$ n_1-many times, through $p \xrightarrow{b} q \xrightarrow{a} p$ n_2-many times, and through $q \xrightarrow{b} q$ n_3-many times, taking into account the effect that each shortest cycle has on the values of the counters, the values of c and d will change as follows (confer to Fig. 2 and the previous list of shortest cycles for the effects that the cycles have on the counters):

$$\Delta c = n_1 - n_3,$$
$$\Delta d = n_1 - n_2.$$

From the acceptance condition $c_+ \wedge c_- \wedge \neg d_+$, we get that, in order to satisfy it, it must be possible to increase c without increasing d, and that it must be possible to decrease c without increasing d. If we could not increase c, then repeatedly going through the combined cycle cannot make c grow arbitrarily large, and therefore c_+ could not be satisfied. If we could not decrease c, then repeatedly going through the combined cycle cannot make c grow arbitrarily small, and therefore c_- could not be satisfied. If those changes of c would not be possible without increasing d, d would grow arbitrarily large when making c arbitrarily large or arbitrarily small, respectively. Therefore $\neg d_+$ could not be satisfied.

So we must check whether or not the system of inequalities

$$\Delta c > 0 \tag{7.1}$$
$$\Delta d \leq 0$$

and the system of inequalities

$$\Delta c < 0 \tag{7.2}$$
$$\Delta d \leq 0$$

each has a positive integer solution. Inequalities (7.1) are

$$n_1 - n_3 > 0$$
$$n_1 - n_2 \leq 0.$$

Combining the two inequalities, we get

$$n_3 < n_1 \leq n_2,$$

which has infinitely many solutions. Also inequalities (7.2), which are

$$n_1 - n_3 < 0$$

$$n_1 - n_2 \leq 0$$

and can be turned into

$$n_1 < n_3$$

$$n_1 \leq n_2,$$

have infinitely many solutions. Because both systems of inequalities have positive integer solutions, these integers tell us how many times which shortest cycle must be visited to have the desired effect on the counter values. Hence, if the inequalities have positive integer solutions, the k-counting automaton contains a cycle that allows creating an accepting run through the automaton, and thus the k-counting automaton cannot be empty.

We believe that by turning all possible interconnected shortest cycles of a k-counting automaton into systems of inequalities as done in the example above, if no such system has positive integer solutions, the automaton is empty, otherwise, if at least one combination of interconnected shortest cycles leads to systems of inequalities that all have a positive integer solution, the automaton accepts a non-empty ω-language.

7.2. Vector-addition systems to represent runs

In the theory of Petri nets [16] and their equivalent counterpart [10], vector addition systems, questions similar to the emptiness problem for k-counting automata can be answered by using the reachability graph [12] of the vector addition system[8]. As this is frequently infinite, a suitable abstraction is the coverability graph [10] of the system. We discuss in this section, how k-counting automata can be interpreted as vector addition systems, hinting also towards the decidability of the emptiness problem.

Let $A = (Q, C, \Sigma, \delta, q_{in}, \Phi)$ be a k-counting automaton. Let v be a valuation of the counters in A. We order the counters from c_0 to c_{k-1} and represent v by an integer-valued k-dimensional vector

$$\nu_v = (v(c_0), v(c_1), \ldots, v(c_{k-1})).$$

For a transition $\delta(p, a) = (q, C_+, C_-)$, we represent the change in the counter values imposed by the sets C_+ and C_- by a $\{-1, 0, 1\}$-valued k-dimensional vector

$$\nu_{(C_+, C_-)} = (\gamma(c_0), \gamma(c_1), \ldots, \gamma(c_{k-1}))$$

such that, for counters c,

[8]The idea of considering vector-addition systems for representing runs of a k-counting automaton is due to Alain Finkel who, in a personal conversation with the authors, pointed out that the emptiness problem for k-counting automata may be reducible to questions about the coverability graph of a corresponding vector-addition system.

- $\gamma(c) = 1$ if $c \in C_+$;
- $\gamma(c) = -1$ if $c \in C_-$; and
- $\gamma(c) = 0$ otherwise.

Then $r = (q_0, v_0)(q_1, v_1)(q_2, v_2)\ldots$ is a run of A on $x = x_0x_1x_2\ldots$ if and only if

- $q_0 = q_{in}$;
- $\delta(q_i, x_i) = (q_{i+1}, C_+, C_-)$ and $\nu_{v_{i+1}} = \nu_{v_i} + \nu_{(C_+,C_-)}$, for all $i \geq 0$,

where "+" is the component-wise vector addition.

Apparently, since there are only finitely many states in a k-counting automaton, also the states can be represented by additional integer (0/1) valued components in the vector. Therefore the automaton itself can be represented fully by a vector addition system. However, an additional acceptance condition is needed that makes statements about the boundedness or unboundedness of some vector components in infinite runs of the vector-addition system. Therefore the emptiness problem cannot be answered simply by investigating the standard coverability graph of the vector-addition system.

8. Conclusions

We have introduced k-counting automata in this paper as recognizers for ω-languages. We could show that the class of ω-languages recognized by k-counting automata contains all regular ω-languages, but goes beyond ω-regularity. Still the language class is closed under all Boolean operations (union, intersection, and complement), and we believe the emptiness problem to be decidable, which would imply that ω-language containment were decidable.

We are not aware of many papers on related subjects in the literature. Bojańczyk and Colcombet (BC) [2, 3] have extended ω-automata by counters and use a (un)boundedness condition on counter values. It will be up to future research to study the relation of k-counting automata to BC-automata, but important differences exist. In particular, BC-automata do not allow decreasing counters, except for resets of counters, and therefore do not have a notion of "compensation" of actions. But the most important difference, is that BC-automata are nondeterministic, whereas k-counting automata are constructed in a deterministic way. In addition, Fernau and Stiebel (FS) [6] explore ω-automata with *blind* counters, looking at their Boolean closure and the ω-language hierarchies they create. FS automata are non-deterministic and equipped with accepting states at which certain properties of the counters must be satisfied. Also the relation between blind counter ω-automata and k-counting automata will be subject to further study.

We defined k-counting automata when aiming to express properties that are not expressible by regular ω-languages: we would have liked expressing properties such as "there are never more b than a in a prefix of an ω-word". This is motivated by practical problems arising in temporal verification. Such a mechanism would allow, for example, to express that in an operating system, we cannot kill more processes than we started before, and similar properties. This would require,

however, transitions that are only enabled when a counter is not zero (similar to the firing condition in Petri nets that requires places not to be empty). As we want to have the closure under complementation for the language class that we define, such a mechanism would immediately lead to the enablement of a transition also depending on a successful *zero test* of a counter. Therefore we would get a multi-counter machine which is Turing-complete [14], and all the properties we would like to decide would become undecidable [17]. Therefore, such a mechanism, even though practically desirable, falls through because of its formal properties. Nevertheless, there exist decidable models, which properly extend our construction, but remain closed under Boolean operations [4].

Future work comprises the emptiness problem for k-counting automata, and the question of whether or not non-deterministic k-counting automata recognize the same class of ω-languages as the deterministic version introduced in this paper. Another open problem is how k-counting-automaton-recognized ω-languages could be represented in an extension to monadic second-order logic [5,18]. Furthermore, we would like to explore whether or not k-counting automata induce an infinite ω-language hierarchy.

Acknowledgements. We are thankful to the reviewers of the NCMA 2011 workshop who have provided us with very helpful constructive comments on the workshop version of this paper. Also the reviewer comments on the draft of this journal version helped us immensely in improving the article. Regarding the "systems of inequalities" approach for possibly deciding emptiness of k-counting automata, we thank Carine Poffet very much for her invaluable comments during several discussions on this topic. For identifying the similarities between testing k-counting automata for emptiness and coverability problems in vector-addition systems, we thank Alain Finkel very much. It was him who came up with this idea in a personal conversation with the authors of this paper. We also would like to thank Ludwig Staiger very much for making us aware of the work by Fernau and Stiebe.

References

[1] B. Alpern and F.B. Schneider, Defining liveness. *Inf. Process. Lett.* **21** (1985) 181–185.
[2] M. Bojanczyk, Beyond ω-regular languages, in *Proc. STACS*, *LIPIcs*, edited by J.-Y. Marion and T. Schwentick. Schloss Dagstuhl – Leibniz-Zentrum für Informatik **5** (2010) 11–16.
[3] M. Bojanczyk and T. Colcombet, Boundedness in languages of infinite words. Unpublished manuscript. Extended version of M. Bojanczyk and T. Colcombet, Bounds in ω-Regularity, in *LICS* (2006) 285–296.
[4] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick and L. Segoufin, Two-variable logic on data words. *ACM Trans. Comput. Logic* **12** (2011) 27:1–27:26.
[5] J.R. Büchi, On a decision method in restricted second order arithmetic, in *Proc. of the International Congress on Logic, Methodology and Philosophy of Science 1960*, edited by E. Nagel *et al.* Stanford University Press (1962) 1–11.
[6] H. Fernau and R. Stiebe, Blind counter automata on ω-words. *Fundam. Inform.* **83** (2008) 51–64.
[7] P.C. Fischer, Turing machines with restricted memory access. *Inf. Control* **9** (1966) 364–379.

[8] K. Hashiguchi, Algorithms for determining relative star height and star height. *Inf. Comput.* **78** (1988) 124–169.
[9] J.E. Hopcroft, R. Motwani and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, Pearson Education (2006).
[10] R.M. Karp and R.E. Miller, Parallel program schemata. *J. Comput. Syst. Sci.* **3** (1969) 147–195.
[11] R.P. Kurshan, *Computer-Aided Verification of Coordinating Processes*, 1st edition. Princeton University Press, Princeton, New Jersey (1994).
[12] E.W. Mayr, An algorithm for the general petri net reachability problem, in *Proc of the 13th Annual ACM Symposium on Theory of Computing, STOC '81*. New York, USA, ACM (1981) 238–246.
[13] R. McNaughton, Testing and generating infinite sequences by a finite automaton. *Inf. Control* **9** (1966) 521–530.
[14] M.L. Minsky, Recursive unsolvability of post's problem of "tag" and other topics in theory of turing machines. *Ann. Math.* **74** (1961) 437–455.
[15] D.E. Muller, Infinite sequences and infinite machines, in *AIEE Proc. of the 4th Annual Symposium on Switching Theory and Logical Design* (1963) 3–16.
[16] C.A. Petri, *Kommunikation mit Automaten*. Ph.D. thesis, Rheinisch-Westfälisches Institut für instrumentelle Mathematik an der Universität Bonn (1962).
[17] H.G. Rice, Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.* **74** (1953) 358–366.
[18] W. Thomas, Automata on infinite objects, in *Formal Models and Semantics*, edited by J. van Leeuwen. *Handbook of Theoret. Comput. Sci.* **B** (1990) 133–191.
[19] U. Ultes-Nitsche, A power-set construction for reducing Büchi automata to non-determinism degree two. *Inform. Process. Lett.* **101** (2007) 107–111.
[20] U. Ultes-Nitsche and S.St. James, Improved verification of linear-time properties within fairness – weakly continuation-closed behaviour abstractions computed from trace reductions. *Software Testing, Verification and Reliability* **13** (2003) 241–255.
[21] M.Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification, in *Proc. of the 1st Symposium on Logic in Computer Science*. Cambridge (1986).
[22] M.Y. Vardi and P. Wolper, Reasoning about infinite computations. *Inform. Comput.* **115** (1994) 1–37.

Communicated by R. Freund.
Received November 9, 2011. Accepted June 13, 2012.