Counting Multiplicity over Infinite Alphabets

Amaldev Manuel and R. Ramanujam

Institute of Mathematical Sciences
Chennai, India - 600113
{amal,jam}@imsc.res.in

Abstract. In the theory of automata over infinite alphabets, a central difficulty is that of finding a suitable compromise between expressiveness and algorithmic complexity. We propose an automaton model where we count the multiplicity of data values on an input word. This is particularly useful when such languages represent behaviour of systems with unboundedly many processes, where system states carry such counts as summaries. A typical recognizable language is: "every process does at most k actions labelled a". We show that emptiness is elementarily decidable, by reduction to the covering problem on Petri nets.

1 Summary

Consider a system of concurrently running sequential processes. When there is no a priori bound on the number of processes, though at any point of time only finitely many are active, the necessity of the system to distinguish one process from another involves potentially unbounded data. Typically, system states carry summary information about processes that are known to be active, and hence the set of system configurations is infinite. Such systems arise in the study of web services, communication protocols and software systems with recursive concurrent threads of execution.

Infinite state systems are not unfamiliar in theory of computation; a rich body of results exists on counter systems, pushdown systems and Petri nets. Most reachability properties of such infinite state systems are either undecidable or have such high complexity that algorithmic verification is impractical. On the other hand, if we restrict ourselves to only finite state systems, we can reason only about systems where the set of processes is fixed and known a priori, and we do not (as yet) have clear abstractions that allow us to transfer the results of such reasoning to systems of unbounded processes. Hence there is a clear need for formal models that work with unbounded systems but yet restrict expressiveness to allow decidable verification.

Notice that interesting properties of such systems do not involve process names (or identifiers) explicitly. A specification that restricts attention only to processes P_1, P_2 and P_3 can be implemented by a finite state system. On the other hand, consider a specification such as: "at least k processes get to perform an a action": this necessitates remembering potentially unboundedly many values, thus leading to infinite state systems.

O. Bournez and I. Potapov (Eds.): RP 2009, LNCS 5797, pp. 141–153, 2009.
© Springer-Verlag Berlin Heidelberg 2009

This paper is situated in such a context and while we have no definitive answers, we consider "state summaries" that allow elementary decidability. The model we use is that of finite automata over infinite alphabets and we use counters to record the intended "summaries". The main result is that emptiness if Expspace-complete for such a class of automata. Unfortunately, the automata are not closed under complementation, and even the word problem is intractable, suggesting that we have more work to be done to further restrict expressiveness.

The study of automaton mechanisms over infinite alphabets has gained interest in recent years, especially from the viewpoint of database theory. In this approach, data values are modelled using a countably infinite domain, and structures are finite words labelled by this infinite alphabet. Typically the alphabet is presented as a product $(\Sigma \times \mathbb{D})$, where Σ is finite and $\mathbb{D}$ is countable. For our purposes, we can think of $\mathbb{D}$ as process names and Σ as the finite set of events they participate in, or conditions that hold.

The study of languages over infinite alphabets were initiated in [ABB80] and [Ott85], where the approach was to define the notion of regularity for languages over infinite alphabets in terms of morphisms to languages over finite alphabets. There are many automaton mechanisms for studying word languages over infinite alphabets: register automata ([KF94]), pebble automata ([NSV01], [NSV04]), data automata ([BMS+06]), nested words ([AM06]), class memory automata ([BS07]) and automata on Gauss words ([LPS09]), with different expressive power and complexity. Logic based approaches include monadic second order logics ([Bou02], [Bac03]), two variable first order logics ([BMS+06]) and temporal logics with special "freeze" quantifiers ([DL06]) or predicate abstraction ([LP05], [LP09]). Algebraic approaches involve quasi-regular expressions ([KT06]), or register monoid mechanisms ([BPT01]). All these involve interesting tradeoffs between expressiveness and complexity of decision procedures. A unifying framework placing all these models in perspective is as yet awaited (see [Seg06] for an excellent survey).

While register automata have polynomial complexity, they are effectively finite state; data automata are more expressive, but emptiness is not known to be elementary. What we present here is a restriction of class memory automata: these automata that can not only test for existence of data values, but can also count the multiplicity of occurrences of data values, subject to constraints on such counts. However, these counters are *monotone*, and hence the constraints are limited in expressive power: we can compare counts against constants, but not much more. We show that such a model of **Class counting automata (CCA)** is interesting, for several reasons; specifically, we get elementary decidability. We see this as "populating the landscape" of classes of data languages, in the sense of [BS07].

From the viewpoint of reasoning about unbounded systems of processes, it is unclear what exactly is the expressiveness needed. For instance, consider the specification: "No two successive positions carry the same data value"; this is naturally implemented using a register mechanism. But this is a "hard" global scheduling constraint: after any process event is scheduled, the succeeding event

must necessarily be from a different process; it is hardly clear that such a constraint is important for loosely coupled systems of processes. This indicates that while we do want to specify combinations of global and local properties, we need to nonetheless allow for sufficient flexibility.

2 Class Counting Automata

Let $k > 0$; we use $[k]$ to denote the set $\{1, 2, \ldots k\}$. When we say $[k]_0$, we mean the set $\{0\} \cup [k]$. By $\mathbb{N}$ we mean the set of natural numbers $\{0, 1, \ldots\}$. When $f : A \to B$, $(a, b) \in (A \times B)$, by $f \oplus (a, b)$, we mean the function $f' : A \to B$, where $f'(a') = f(a')$ for all $a' \in A, a' \neq a$, and $f'(a) = b$.

Customarily, the infinite alphabet is split into two parts: it is of the form $\Sigma \times D$, where Σ is a finite set, and D is a countably infinite set. Usually, Σ is called the *letter alphabet* and D is called the *data alphabet.* Elements of D are referred to as *data values.* We use letters a, b etc to denote elements of Σ and use d, d' to denote elements of D.

A **data word** w is an element of $(\Sigma \times D)^*$. A collection of data words $L \subseteq (\Sigma \times D)^*$ is called a *data language.* In this article, by default, we refer to data words simply as words and data languages as languages. As usual, by $|w|$ we denote the length of w.

Let $w = (a_1, d_1)(a_2, d_2) \ldots (a_n, d_n)$ be a data word. The *string projection* of w, denoted as $str(w) = a_1 a_2 \ldots a_n$, the projection of w to its Σ components. Let $i \in [n] = |w|$. The data class of d_i in w is the set $\{j \in [n] \mid d_i = d_j\}$. A subset of $[n]$ is called a data class of w if it is the data class of some d_i, $i \in [n]$. Note that the set of data classes of w form a partition of $[|w|]$.

The automaton we present below includes a bag of infinitely many monotone counters, one for each possible data value. When it encounters a letter - data pair, say (a, d), the multiplicity of d is checked against a given constraint, and accordingly updated, the transition causing a change of state, as well as possible updates for other data as well. We can think of the bag as a hash table, with elements of D as keys, and counters as hash values. Transitions depend only on hash values (subject to constraints) and not keys.

A **constraint** is a pair $c = (op, e)$, where $op \in \{<, =, \neq, >\}$ and $e \in \mathbb{N}$. When $v \in \mathbb{N}$, we say $v \models c$ if $v \,\mathsf{op}\, e$ holds. Let $\mathcal{C}$ denote the set of all constraints. Define a *bag* to be a finite set $h \subseteq (\mathbb{D} \times \mathbb{N})$ such that whenever $(d, n_1) \in h$ and $(d, n_2) \in h$, we have: $n_1 = n_2$. Thus h defines a partial function from $\mathbb{D}$ to $\mathbb{N}$ which is defined on a finite subset of $\mathbb{D}$. By convention, we implicitly extend it to a total function on D by considering h to represent the set $h' = h \cup \{(d, 0) \mid$ there is no $n \in \mathbb{N}$ such that $(d, n) \in h\}$. Hence we (ab)use the notation $h(d) = n$ for a bag h. Let $\mathbb{B}$ denote the set of bags. Note that the notation $h \oplus (d, n)$ now stands for the bag $h' = (h - (\{d\} \times \mathbb{N})) \cup \{(d, n)\}$.

Below, let $Inst = \{\uparrow^+, \downarrow\}$ stand for the set of *instructions*: $\uparrow^+$ tells the automaton to increment the counter, whereas $\downarrow$ asks for a reset. Note that the instruction $(\uparrow^+, 0)$ says that we do not wish to make any update, and $(\uparrow^+, 1)$ causes a unit increment; we use the notation $[0]$ and $[+1]$ for these instructions below.

Definition 1. *A* **class counting automaton***, abbreviated as CCA, is a tuple $CCA = (Q, \Delta, I, F)$, where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states. The transition relation is given by: $\Delta \subseteq (Q \times \Sigma \times C \times Inst \times U \times Q)$, where C is a finite subset of $\mathcal{C}$ and U is a finite subset of $\mathbb{N}$.*

Let A be a CCA. A configuration of A is a pair (q, h), where $q \in Q$ and $h \in \mathbb{B}$. The initial configuration of A is given by (q_0, h_0), where h_0 is the empty bag; that is, $\forall d \in \mathbb{D}, h_0(d) = 0$ and $q_0 \in I$.

Given a data word $w = (a_1, d_1), \ldots (a_n, d_n)$, a run of A on w is a sequence $\gamma = (q_0, h_0)(q_1, h_1) \ldots (q_n, h_n)$ such that $q_0 \in I$ and for all $i, 0 \leq i < n$, there exists a transition $t_i = (q, a, c, \pi, m, q') \in \Delta$ such that $q = q_i$, $q' = q_{i+1}$, $a = a_{i+1}$ and:

- $h_i(d_{i+1}) \models c$.
- h_{i+1} is given by:

$$h_{i+1} = \begin{cases} h_i \oplus (d_{i+1}, m') \text{ if } \pi = \uparrow^+, m' = h_i(d_{i+1}) + m \\ h_i \oplus (d_{i+1}, m) \ \text{ if } \pi = \downarrow \end{cases}$$

γ is an accepting run above if $q_n \in F$. The language accepted by A is given by $\mathcal{L}(A) = \{w \in (\Sigma \times \mathbb{D})^* \mid A \text{ has an accepting run on } w\}$. $\mathcal{L} \subseteq ((\Sigma \times \mathbb{D}))^*$ is said to be recognizable if there exists a CCA A such that $\mathcal{L} = \mathcal{L}(A)$. Note that the counters are either incremented or reset to fixed values.

We first observe that CCA runs have some useful properties. To see this, consider a bag h and $d_1, d_2 \in D$, $d_1 \neq d_2$ such that at a confguration (q, h), we have two transitions enabled on inputs (a_1, d_1) and (a_2, d_2) leading to configurations (q_1, h_1) and (q_2, h_2) respectively. Notice that for any condition c, if $h(d_2) \models c$ then so also $h_1(d_2) \models c$. Similarly, for any condition c', if $h(d_1) \models c'$ then so also $h_2(d_1) \models c'$. Thus when we have distinct data values, tests on them do not "interfere" with each other. We can extend this observation further: given data words u and v such that the data values in u are pairwise disjoint from those in v, if we have a run from (q, h) on u to (q, h_1) and on v from (q, h_1) to (q', h_2), then there is a configuration (q', h') and a run from (q, h) on v to (q', h'). This will be useful in the following.

Example 1. The language $L_{fd(a)}$ = *"Data values under a are all distinct"* is accepted by a CCA. The CCA accepting this language is the automaton $A = (Q, \Delta, q_0, F)$ where $Q = \{q_0, q_1\}$, q_0 is the only initial state and $F = \{q_0\}$. Δ consists of:

- $(q_0, a, (=, 0), q_0, [+1])$; $(q_0, a, (=, 1), q_1, [0])$;
- $(q_0, b, (\geq, 0), q_0, [0])$; $(q_1, \Sigma, (\geq, 0), q_1, [0])$.

Since the automaton above is deterministic, by complementing it, that is, setting $F = \{q_1\}$, we can accept the language $\overline{L_{fd(a)}}$ = *"There exists a data value appearing at least twice under a"*. On the other hand, since every data word can mention only finitely many data values, trivially every word has a value

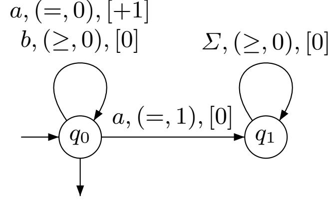

Fig. 1. Automaton in the Example 1

that appears less than twice under a (namely zero times). Hence the statement above can be strengthened to saying that the language $L_{\exists a, \neq\ n}$ = *" There exists a data value whose multiplicity under a is not 2"* is recognizable. But as we show below, its complement language, $L_{\forall, =\ n}$ = *"All data values under a occur exactly twice"* is not recognizable. Thus, CCA- recognizable data languages are not closed under complementation.

Proposition 1. *The language* $L_{forall, =\ n}$ = *"All data values under a occur exactly twice" is not recognizable.*

Proof. Suppose there is a CCA A with m states accepting this language. Consider the data word

$$w = (a, d_1)(a, d_2)!..(a, d_{m+1})(a, d_1)(a, d_2)..(a, d_{m+1})$$

Clearly, $winL_{\forall, =\ n}$. Therefore, there is a successful run of A on w. Then there is a state q repeating in the suffix of length $m + 1$. Let us say this splits w as $u{\cdot}v{\cdot}v'$, where the configurations at the repeating state after u with configuration (q, h) to (q, h_1) on v and to (q', h_2) on v'. Then by the remarks we made earlier, we can find a run from (q, h) to a configuration (q', h') on v' as well. Thus we have "chopped" off a part of the run so that we have an accepting run on a word $u \cdot v'$. But then $u \cdot v'$ is not in $L_{forall, =\ n}$. □

The following statement is easily proved:

Proposition 2. *CCA-recognizable data languages are closed under union and intersection but not under complementation.*

The following observation will be useful for decision questions that follow. Given a CCA $A = (Q, \Delta, q_0, F)$ let m be the maximum constant used in Δ. We define the following equivalence relation on $\mathbb{N}$, $c \simeq_{m+1} c'$ iff $c < (m{+}1) \vee c' < (m{+}1) \Rightarrow c = c'$. Note that if $c \simeq_{m+1} c'$ then a transition is enabled at c if and only if it is enabled at c'. We can extend this equivalence to configurations of the CCA as follows. Let $(q_1, h_1) \simeq_{m+1} (q_2, h_2)$ iff $q_1 = q_2$ and $\forall d \in \mathbb{D}, h_1(d) \simeq_{m+1} h_2(d)$.

Lemma 1. *If* C_1, C_2 *are two configurations of the* CCA *such that* $C_1 \simeq_{m+1} C_2$, *then* $\forall w \in ((\Sigma \times \mathbb{D}))^*$, $C_1 \vdash_w^* C_1' \implies \exists C_2', C_2 \vdash_w^* C_2'$ *and* $C_1' \simeq_{m+1} C_2'$.

Proof. Proof by induction on the length of w. For the base case observe that any transition enabled at C_1 is enabled at C_2 and the counter updates respects the equivalence. For the inductive case consider the word $w.a$. By induction hypothesis $C_1 \vdash_w^* C_1' \implies \exists C_2', C_2 \vdash_w^* C_2'$ and $C_1' \simeq_{m+1} C_2'$. If $C_1' \vdash_a C_1''$ then using the above argument there exists C_2'' such that $C_2' \vdash_a C_2''$ and $C_1'' \simeq_{m+1} C_2''$.

In fact the lemma holds for any $N \geq m+1$, where m is the maximum constant used in Δ. This observation paves the way for proving the decidability of the emptiness problem (in the next section).

3 Decision Problems

Since the space of configurations of a CCA is infinite, reachability is in general non-trivial to decide. We now show that the emptiness problem is elementarily decidable.

Theorem 1. *The non-emptiness problem for CCA is* EXPSPACE*-complete.*

3.1 Upper Bound

We reduce the emptiness problem of CCA to the covering problem on Petri nets. For checking emptiness, we can omit the $\Sigma \times D$ labels from the configuration graph; we are then left with counter behaviour. However since we have unboundedly many counters, we are led to the realm of vector addition systems.

Definition 2. *An ω-counter machine B is a tuple (Q, Δ, q_0) where Q is a finite set of states, $q_0 \in Q$ is the initial state and $\Delta \subseteq (Q \times C \times Inst \times U \times Q)$, where C is a finite subset of $\mathcal{C}$ and U is a finite subset of $\mathbb{N}$.*

A configuration of B is a pair (q, h), where $q \in Q$ and $h : \mathbb{N} \to \mathbb{N}$. The initial configuration of B is (q_0, h_0) where $h_0(i) = 0$ for all i in $\mathbb{N}$. A run of B is a sequence $\gamma = (q_0, h_0)(q_1, h_1) \ldots (q_n, h_n)$ such that for all i such that $0 \leq i < n$, there exists a transition $t_i = (p, c, \pi, m, q) \in \Delta$ such that $p = q_i$, $q = q_{i+1}$ and there exists j such that $h(j) \models c$, and the counters are updated in a similar fashion to that of CCA.

The reachability problem for B asks, given $q \in Q$, whether there exists a run of B from (q_0, h_0) ending in (q, h) for some h ("Can B reach q?").

Lemma 2. *Checking emptiness for CCA can be reduced to checking reachability for ω-counter machines.*

Proof. It suffices to show, given a CCA, $A = (Q, \Delta, q_0, F)$, where $F = \{q\}$, that there exists a counter machine $B_A = (Q, \Delta', q_0)$ such that A has an accepting run on some data word exactly when B_A can reach q. (When F is not a singleton, we simply repeat the construction.) Δ' is obtained from Δ by converting every transition (p, a, c, π, m, q) to (p, c, π, m, q). Now, let $L(A) \neq \emptyset$. Then there exists a data word w and an accepting run $\gamma = (q_0, h_0)(q_1, h_1) \ldots (q_n, h_n)$ of A on w,

with $q_n = q$. Let $g : \mathbb{N} \to D$ be an enumeration of data values. It is easy to see that $\gamma' = (q_1, h_0 \circ g)(q_1, h_1 \circ g) \dots (q_n, h_n \circ g)$ is a run of B_A reaching q.

($\Leftarrow$) Suppose that B_A has a run $\eta = (q_0, h_0)(q_1, h_1) \dots (q_n, h_n)$, $q_n = q$. It can be seen that $\eta' = (q_0, h_0 \circ g^{-1})(q_1, h_1 \circ g^{-1}) \dots (q_n, h_n \circ g^{-1})$ is an accepting run of A on $w = (a_1, d_1) \dots (a_n, d_n)$ where w satisfies the following. Let (p, c, π, m, q) be the transition of B_A taken in the configuration (q_i, h_i), and d_k such that $h_i(d_k) \models c$. Then by the definition of B_A there exists a transition (p, a, c, π, m, q) in Δ. Then it should be the case that $a_{i+1} = a$ and $d_{i+1} = g(d_k)$.

Proposition 3. *Checking non-emptiness of ω-counter machines is decidable.*

Let $s \subseteq \mathbb{N}$, and c a constraint. We say $s \models c$, if for all $n \in s$, $n \models c$.

We define the following partial function *Bnd* on all finite and cofinite subsets of $\mathbb{N}$. Given $s \subseteq_{fin} \mathbb{N}$, $Bnd(s)$ is defined to be the least number greater than all the elements in s. Given $s \subseteq_{cofinite} \mathbb{N}$, $Bnd(s)$ is defined to be $Bnd(\mathbb{N} \backslash s)$. Given an ω-counter machine $B = (Q, \Delta, q_0)$ let $m_B = max\{Bnd(s) \mid s \models c, c \text{ is used in } \Delta\}$.

We construct a Petri net $N_B = (S, T, F, M_0)$ where,

- $S = Q \cup \{i \mid i \in \mathbb{N}, 1 \leq i \leq m_B\}$.
- T is defined according to Δ as follows. Let $(p, c, \pi, n, q) \in \Delta$ and let i be such that $0 \leq i \leq m_B$ and $i \models c$. Then we add a transition t such that ${}^{\bullet}t = \{p, i\}$ and $t^{\bullet} = \{q, i'\}$, where (i) if π is $\uparrow^+$ then $i' = min\{m_B, i + n\}$, and (ii) if π is $\downarrow$ then $i' = n$.
- The flow relation F is defined according to ${}^{\bullet}t$ and $t^{\bullet}$ for each $t \in T$.
- The initial marking is defined as follows. $M_0(q_0) = 1$ and for all p in S, if $p \neq q_0$ then $M_0(p) = 0$.

The construction above glosses over some detail: Note that elements of these sets can be zero, in which case we add edges only for the places in $[m_B]$ and ignore the elements which are zero.

Let M be any marking of N_B. We say that M is a *state marking* if there exists $q \in Q$ such that $M(q) = 1$ and $\forall p \in Q$ such that $p \neq q$, $M(p) = 0$. When M is a state marking, and $M(q) = 1$, we speak of q as the state marked by M. For $q \in Q$, define $M_f(q)$ to be set of state markings that mark q. It can be shown, from the construction of N_B, that in any reachable marking M of N_B, if there exists $q \in Q$ such that $M(q) > 0$, then M is a state marking, and q is the state marked by M.

We now show that the counter machine B can reach a state q iff N_B has a reachable marking which covers a marking in $M_f(q)$. We define the following equivalence relation on $\mathbb{N}$, $m \simeq_{m_B} n$ iff $(m < m_B) \vee (n < m_B) \Rightarrow m = n$. We can lift this to the hash functions (in ω-counters) in the natural way: $h \simeq_{m_B} h'$ iff $\forall i\, (h(i) < m_B) \vee (h'(i) < m_B) \Rightarrow h(i) = h'(i)$. It can be easily shown that if $h \simeq_{m_B} h'$ then a transition is enabled at h if and only if it is enabled at h'.

Let μ be a mapping B-configurations to N_B-configurations as follows: given $\chi = (q, h)$, define $\mu(\chi) = M_\chi$, where

$$M_\chi(p) = \begin{cases} 1 & \text{iff } p = q \\ 0 & \text{iff } p \in Q \backslash \{q\} \\ |[p]| & \text{iff } p \in P \backslash Q, p \neq 0 \end{cases}$$

Above $[p]$ denotes the equivalence class of p under $\simeq_{m_B}$ on $\mathbb{N}$ in h. Now suppose that B reaches q. Let the resulting configuration be $\chi = (q, h)$. We claim that the marking $\mu(\chi)$ of N_B is reachable (from M_0) and covers $M_f(q)$. Conversely if a reachable marking M of N_B covers $M_f(q)$, for some $q \in Q$, then there exists a reachable configuration $\chi = (q, h)$ of B such that $\mu(\chi) = M$. This is proved by a simple induction on the length of the run.

Since the covering problem for Petri nets is decidable, so is reachability for ω-counter machines and hence emptiness checking for CCA is decidable.

3.2 Lower Bound

The decision procedure above runs in EXPSPACE, and thus we have elementary decidability. We now show that the emptiness problem is also EXPSPACE-hard. Effectively this is a reduction of the covering problem again, but for technical convenience, we use multi-counter automata.

A k multi-counter automaton with weak acceptance is a tuple $A = (Q, \Sigma, \Delta, q_0, F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is a set of final states. The transition relation is of the form $\Delta \subseteq_{fin} (Q \times \Sigma \times \mathbb{N}^k \times \mathbb{N}^k \times Q)$. The two vectors in the transition specify decrements and increments of the counters.

The automaton works as follows: it has k-counters, denoted by $\bar{v} = (v_1, \ldots v_k)$ which hold non-negative counter values. A configuration of the machine is of the form $(q, \bar{v})$ where $q \in Q$ and $\bar{v} \in \mathbb{N}^k$. The initial configuration is $(q_0, \bar{0})$. Given a configuration $(q, \bar{v})$ the automaton can go to a configuration $(q', \bar{v}')$ on letter a if there is a transition $(q, a, v\bar{_{dec}}, v\bar{_{inc}}, q')$ such that $\bar{v} - v\bar{_{dec}} \geq \bar{0}$ (pointwise) and $\bar{v}' = \bar{v} - v\bar{_{dec}} + v\bar{_{inc}}$. A final configuration is one in which the state is final.

The problem of checking non-emptiness of a multicounter automaton with weak acceptance is known to be (at least) EXPSPACE-hard ([Lip76]).

Any multicounter automaton $M = (Q, \Sigma, \Delta, q_0, F)$ can be converted to another (in a "normal form"): $M' = (Q', \Sigma, \Delta', q_0, F)$ such that $L(M)$ is non-empty if and only if $L(M')$ is non-empty and M' uses only unit vectors or zero vectors in its transitions. A unit vector is of the form $(b_1, b_2, \ldots, b_k)$ where there is a unique $i \in [k]$ such that $b_i = 1$ and for $j \neq i$k, $b_j = 0$. That is M' decrements or increments at most one counter in each transition.

Δ' is obtained as follows. Let $t = (q, a, v\bar{_{dec}}, v\bar{_{inc}}, q')$. Let $\bar{u_1}, \bar{u_2}, \ldots, \bar{u_n}$ be a sequence of unit vectors such that $v\bar{_{dec}} = \Sigma_i \bar{u_i}$ and $\bar{u_1}', \bar{u_2}', \ldots, \bar{u_m}'$ be a sequence of unit vectors such that $v\bar{_{inc}} = \Sigma_i \bar{u_i}'$. We add intermediate states to rewrite t by the following sequence of transitions,

$$(q, a, \bar{u_1}, \bar{0}, q_{(t,\bar{u_1})}), (q_{(t,\bar{u_1})}, a, \bar{u_2}, \bar{0}, q_{(t,\bar{u_2})}), \ldots,$$

$$(q_{(t,\bar{u_n})}, a, \bar{0}, \bar{u_1}', q_{(t,\bar{u_1}')}), (q_{(t,\bar{u_1}')}, a, \bar{0}, \bar{u_2}', q_{(t,\bar{u_2}')}), \ldots,$$

$$(q_{(t,\bar{u_{m-1}}')}, a, \bar{0}, \bar{u_m}', q')$$

Lemma 3. *$L(M)$ is non-empty if and only if $L(M')$ is non-empty.*

Proof. By an easy induction on the length of the run. It is easy to see that for every accepting run ρ of M we have an accepting run ρ' of M', this is achieved by replacing every transition t in the run ρ by the corresponding sequence of transitions. For the reverse direction, we need to show that every run accepting run ρ' of M' can be translated to an accepting run ρ of M. This is possible since the intermediate states added to obtain the transitions in M' are unique for each transition t in M. Hence for every sequence of transitions taking M' from q_1 to q_2 where $q_1, q_2 \in Q$ there is a unique transition t which takes M from q_1 to q_2. By doing an induction on the number of states occuring in ρ' which are from Q we can show that there is a valid run ρ which is accepting.

Next we convert M' to a CCA thus establishing a lowerbound of Expspace for the emptiness problem. Let $M' = (Q, \Sigma, \Delta, q_0, F)$ be a k-multicounter automaton in normal form. We construct the automaton $A = (Q, \Sigma, \Delta_A, q_0, F)$. Let $t = (q, a, \bar{u}, \bar{u}', q')$ where $\bar{u}, \bar{u}'$ are either unit or zero vectors. If $\bar{u}$ is a i-th unit vector and $\bar{u}'$ is a zero vector, we add a transition $t_A = (q, a, (x = i), (\downarrow, 0), q')$ to Δ_A. If $\bar{u}$ is a i-th unit vector and $\bar{u}'$ is j-th unit vector, we add a transition $t_A = (q, a, (x = i), (\downarrow, j), q')$ to Δ_A. If $\bar{u}$ is a zero vector and $\bar{u}'$ is a j-th unit vector, we add a transition $t_A = (q, a, (x = 0), (\downarrow, j), q')$ to Δ_A.

Lemma 4. *$L(M')$ is non-empty if and only if $L(A)$ is non-empty.*

Proof. The proof is by induction on the length of the run. First we define a mapping from configurations of A to configurations of M' in the following manner, $\mu((q, \bar{h})) = (q, \bar{v})$ where $v_i = |\{j \mid \bar{h}(j) = i\}|$. We show, by induction on the length of the run, that for every configuration χ reachable by A there is a configuration ψ of M' such that $\mu(\chi) = \psi$ and conversely for every configuration ψ reachable by M' there is a configuration χ reachable by A such that $\mu(\chi) = \psi$.

For the base case, it is evident that $\mu((q_0, \bar{h_0})) = (q_0, \bar{0})$.

Suppose that $\chi = (q, \bar{h})$ is a configuration reachable in l steps, and that the transition $t = (q, a, x = j, (\downarrow, i), q')$ is enabled at χ. Therefore there is a counter holding the value j. By induction hypothesis there exists a configuration ψ such that $\mu(\chi) = \psi = (q, \bar{v})$ such that $v_j > 0$. After the transition t, the number of counters holding the value j decreases by one and the number of counters holding the value i increases by one(if $i \neq 0$). This is achieved by the transition $(q, a, \bar{u_j}, \bar{u_i}, q')$ in Δ', preserving the map μ.

Conversely, suppose a configuration $\psi = (q, \bar{v})$ is reachable by M' in l steps. Then by induction hypothesis we have a configuration χ reachable by the automaton A such that $\mu(\chi) = \psi$. Suppose a transition $t' = (q, a, \bar{u_i}, \bar{u_j}, q')$ is enabled in ψ resulting in ψ'.

Consider the case where $\bar{u_i} \neq \bar{0}$ and $\bar{u_j} \neq \bar{0}$. By construction t' is obtained from a transition $t = (q, a, (x = i), \downarrow, j, q')$. We choose the smallest counter holding the value zero and apply the transition t, resulting in ξ' such that $\mu(\xi') = \psi'$. The remaining cases are similar.

3.3 Inclusion and Word Problem

The next interesting algorithmic question is that of checking inclusion among accepted languages. It turns out that this problem is undecidable, which can be shown by reduction from the Post Correspondence Problem. We postpone the discussion on this until we discuss alternation later.

Since emptiness checking is of such high complexity, one may wonder whether the model is complex enough to render even the word problem to be hard: the simplest algorithmic question of how one can check whether a given word is accepted or not. The important thing to note is that during a run, the size of the configuration is bounded by the length of the input data word. Therefore a nondeterministic Turing machine can easily guess a path in polynomial time and check for acceptance. Hence the word problem is easily seen to be in NP. Interestingly, it turns out to be NP-hard as well.

Theorem 2. *The word problem for CCA is NP-complete.*

The proof is by reduction of the satisfiability problem for 3-CNF formulas to the word problem for CCAs. Given the 3-CNF formula, we code it up as a data word, where data values are used to remember the identity of literals in clauses. We use a two letter alphabet with $+, -$ indicating whether a propositional variable occurs positively or negatively. Data values stand for the propositional variables themselves. Thus a pair $((+, d_1)$ asserts that the first boolean variable occurs positively.

We show the coding by an example, let $\varphi \equiv (p_1 \vee \neg p_3 \vee p_4) \wedge (\neg p_2 \vee p_5 \vee p_1) \wedge (\neg p_3 \vee \neg p_4 \vee p_5)$, we construct the corresponding word $w = (+, d_1)(-, d_3)(+, d_4)$ $(\#, d)$ $(-, d_2)(+, d_5)(+, d_1)$ $(\#, d)$ $(-, d_3)(-, d_4)(+, d_5)(\#, d) \in (\{+, -, \#\} \times \mathbb{D})^*$.

The nondeterministic automaton checks satisfiability in the following way. Every time the automaton encounters a new data value (representing a propositional variable), the automaton nondeterministically assigns a boolean value and stores it in the counter (1 for $\bot$ and 2 for $\top$) corresponding to the data value, in the future whenever the same data value occurs the counter is consulted to obtain the assigned value to the propositional variable. The automaton evaluates each clause and carries the partial evaluation in its state. Finally the automaton accepts the word if the formula evaluates to $\top$.

4 Discussion

We first observe that the model admits many extensions, without substantially affecting the main decidability result.

4.1 Extensions

1. Instead of working with one bag of counters, the automaton can use several bags of counters, much as multiple registers are used in the register automaton. It is easy to formally define CCA with k-bags, using k-tuples of constraints on guards.

2. Another strengthening involves checking for the presence of *any* counter satisfying a given constraint and updating it.
3. The language of constraints can be strengthened: any syntax that can specify a finite or co-finite subset of $\mathbb{N}$ will do. Indeed, we can work with constraints specifying *semilinear sets* without affecting the technical results, and the syntax can be any formula in Presburger arithmetic.

On the other hand, some natural extensions of the model do affect the decidability of non-emptiness problem. One such is *alternation.* However, we then find that *the non-emptiness problem for the class of alternating class counting automata is undecidable.* The proof of this proceeds by reduction of the Post Correspondence Problem to this one in a manner similar to the one in [BMS^{+}06]. From this, we can show that the inclusion problem for CCAs is undecidable as well.

Other interesting extensions relate to the kind of updates allowed and to acceptance conditions. While adding decrements to counters in CCA leads to undecidability of the emptiness problem, we can add resets to counters preserving decidability. A reset operation sets the corresponding counter value to zero. The acceptance condition we have in CCA is *global* in the sense that it relates only to the global control state rather than multiplicities encountered. We can strengthen the acceptance condition as follows: $A = (Q, \Delta, q_0, F, C)$ where (Q, q_0, Δ, F are as before, and $C \subset_{fin} N$. We say a final configuration (q, h) is accepting if $q \in F$ and $\forall d \in \mathbb{D}, h(d) \in C$ or $h(d) = 0$.

We then find that the non-emptiness problem (for CCAs with reset and counter conditions) continues to be decidable but becomes as hard as Petri net reachability, which is not even known to be elementarily decidable.

4.2 Other Automata Models

CCA are situated among a family of automata models that have been proposed for data languages. The simplest form of memory is a finite random access read-write storage device, traditionally called *register.* In *finite memory* automata [KF94], the machine is equipped with finitely many registers, each of which can be used to store one data value. Every automaton transition includes access to the registers, reading them before the transition and writing to them after the transition. The new state after the transition depends on the current state, the input letter and whether or not the input data value is already stored in any of the registers. If the data value is not stored in any of the registers, the automaton can choose to write it in a register. The transition may also depend on which register contains the encountered data value. Because of finiteness of the number of registers, in a sufficiently long word the automaton cannot distinguish between all data values. On the other hand, register automata have the capability of keeping the "latest information", a capability that deterministic CCA do not have.

In class memory automata (CMA, [BS07]), a function assigns to every data value d the state of the automaton that was assumed after reading the previous

position with value d. We can think of this as using hash tables, with values coming from a finite set. On reading a (a, d), the automaton reads the table entry corresponding to d and makes a transition dependent on the table entry, the input letter a and the current state. The transition causes a change of state as well as updating of the table entry.

We can show that the class of CCA-recognizable languages is strictly contained in the class of CMA-recognizable languages, but when we add resets and counter acceptance conditions as above, the class becomes exactly as expressive as CMAs. Indeed, we see CCA as a natural restriction of CMAs yielding elementary decidability of the non-emptiness problem.

Another simple computational model, based on transducers is the data automaton model introduced in [BMS⁺06], and [BS07] shows that this model is exactly as expressive as CMA.

4.3 Restrictions

With an NP-hard word problem, Expspace-hard non-emptiness question and undecidable language inclusion, working with data languages does seem daunting. However, given the need for verifying properties of systems with unboundedly many processes, the abstraction of infinite alphabets is yet worth preserving. What we need to look at are restrictions that are meaningful for systems of unbounded processes, and we are studying some proposals in this regard.

References

[ABB80] Autebert, J.-M., Beauquier, J., Boasson, L.: Langages sur des alphabets infinis. Discrete Applied Mathematics 2, 1–20 (1980)

[AM06] Alur, R., Madhusudan, P.: Adding nesting structure to words. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 1–13. Springer, Heidelberg (2006)

[Bac03] Baclet, M.: Logical characterization of aperiodic data languages. Research Report LSV-03-12, Laboratoire Spécification et Vérification, ENS Cachan, France, 16 p. (September 2003)

[BMS+06] Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: LICS, pp. 7–16. IEEE Computer Society, Los Alamitos (2006)

[Bou02] Bouyer, P.: A logical characterization of data languages. Inf. Process. Lett. 84(2), 75–85 (2002)

[BPT01] Bouyer, P., Petit, A., Thérien, D.: An algebraic characterization of data and timed languages. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 248–261. Springer, Heidelberg (2001)

[BS07] Björklund, H., Schwentick, T.: On notions of regularity for data languages. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 88–99. Springer, Heidelberg (2007)

[DL06] Demri, S., Lazic, R.: Ltl with the freeze quantifier and register automata. In: LICS 2006: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science, pp. 17–26. IEEE Computer Society, Los Alamitos (2006)

[KF94] Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. 134(2), 329–363 (1994)

[KT06] Kaminski, M., Tan, T.: Regular expressions for languages over infinite alphabets. Fundam. Inform. 69(3), 301–318 (2006)

[Lip76] Lipton, R.: The reachability problem requires exponential space. Research Report 62, Yale University (1976)

[LP05] Lisitsa, A., Potapov, I.: Temporal logic with predicate lambda-abstraction. In: TIME, pp. 147–155 (2005)

[LP09] Lisitsa, A., Potapov, I.: On the computational power of querying the history. Fundam. Inform. 91(2), 395–409 (2009)

[LPS09] Lisitsa, A., Potapov, I., Saleh, R.: Automata on gauss words. In: LATA, pp. 505–517 (2009)

[NSV01] Neven, F., Schwentick, T., Vianu, V.: Towards regular languages over infinite alphabets. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 560–572. Springer, Heidelberg (2001)

[NSV04] Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Log. 5(3), 403–435 (2004)

[Ott85] Otto, F.: Classes of regular and context-free languages over countably infinite alphabets. Discrete Applied Mathematics 12, 41–56 (1985)

[Seg06] Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 41–57. Springer, Heidelberg (2006)