4.1 Introduction

In this chapter we introduce Class Counting Automata, an extension of finite state automata with counters. We show that the non-emptiness problem for these automata is decidable in elementary time. We also study several extensions of these automata and the complexity of their decision problems. The contents of this chapter appeared in [MRT11].

4.2 Class counting automata

A constraint $\varphi(x)$ is a univariate inequality of the form $x \leq e$ or $x \geq e$, where $e \in \mathbb{N}$. When $v \in \mathbb{N}$, we say $v \models \varphi(x)$ if $\varphi(v)$ holds. For convenience, often we denote the constraints as $c, c_1, \ldots$. Let $C$ denote the set of all constraints. Define a bag to be a finite set $h \subseteq (\Gamma \times \mathbb{N})$ 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 $\Gamma$ to $\mathbb{N}$ which is defined on a finite subset of $\Gamma$. By convention, we implicitly extend it to a total function on $\Gamma$ by considering $h$ to represent the set $h' = h \cup \{(d, 0) \mid d \notin \text{Domain}(h)\}$. Hence we (ab)use the notation $h(d) = n$ for a bag $h$. Let $\mathcal{B}$ denote the set of bags. Note that the notation $h \oplus (d, n)$ now stands for the bag $h' = (h - (\{d\} \times \mathbb{N})) \cup \{(d, n)\}$.

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 \(\Gamma\) as keys, and counters as hash values. Transitions depend only on hash values (subject to constraints) and not keys.

Below, let \(\text{Inst} = \{\text{inc}, \text{reset}\}\) stand for the set of instructions. We use variables \(\pi, \pi_1, \ldots\) to represent the instructions. Each instruction takes a natural number as an argument. The \text{inc} instruction with argument \(k\) tells the automaton to increment the counter by \(k\), whereas \text{reset} with argument \(k\) asks for a reset to the value \(k\). Note that the instruction \((\text{inc}, 0)\) says that we do not wish to make any update, and \((\text{inc}, 1)\) causes a unit increment; we use the notation \([0]\) and \([+1]\) for these instructions below.

**Definition 4.2.1.** A class counting automaton, abbreviated as CCA, is a tuple \(\text{CCA} = (Q, \Sigma, \Delta, I, F)\), 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: \(\Delta \subseteq \text{fin}(Q \times \Sigma \times C \times \text{Inst} \times \mathbb{N} \times Q)\).

**Representation of constants:** We note here that the constants in the definition of the automata are represented in unary. The mode of representation of numbers turns out to be crucial for the upper bound of the emptiness problem.

Let \(A\) be a CCA. A configuration of \(A\) is a pair \((q, h)\), where \(q \in Q\) and \(h \in \mathcal{B}\). An initial configuration of \(A\) is given by \((q_0, h_0)\), where \(q_0 \in I\) and \(h_0\) is the empty bag; that is, \(\forall d \in \Gamma, h_0(d) = 0\) 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 \((q_0, h_0)\) is an initial configuration and for each \(1 \leq i \leq n\) 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:

\[
\begin{align*}
    h_{i+1} &= \begin{cases} 
        h_i \oplus (d_{i+1}, m') & \text{if } \pi = \text{inc}, m' = h_i(d_{i+1}) + m \\
        h_i \oplus (d_{i+1}, m) & \text{if } \pi = \text{reset}
    \end{cases}
\end{align*}
\]
\( \gamma \) is an **accepting** run above if \( q_n \in F \). The language accepted by \( A \) is given by \( L(A) = \{ w \in (\Sigma \times \Gamma)^* \mid A \text{ has an accepting run on } w \} \). \( L \subseteq (\Sigma \times \Gamma)^* \) is said to be recognizable if there exists a CCA \( A \) such that \( L = L(A) \). Note that the counters are either incremented or reset to fixed values.

If the configuration \( c_2 = (q_2, h_2) \) is reachable from \( c_1 = (q_1, h_1) \) on \((a, d)\) we denote it by \( c_1 \vdash_{(a, d)} c_2 \). Extending this notion further if \( c_2 \) is reachable from \( c_1 \) on the data word \( w \) we denote it by \( c_1 \vdash_w c_2 \). We first observe that CCA runs have some useful properties. To see this, consider a bag \( h \) and \( d_1, d_2 \in \Gamma \), \( d_1 \neq d_2 \) such that at a configuration \((q, h)\), 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, that is \((q, h) \vdash_{(a_1, d_1)} (q_1, h_1)\) and \((q, h) \vdash_{(a_2, d_2)} (q_2, h_2)\). 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 \((q', h')\), that is:

\[
(q, h) \vdash_u (q, h_1) \vdash_v (q', h_2) \implies \exists h' \in B, (q, h) \vdash_v (q', h')
\]

This observation will be useful in the following.

**Example 4.2.2.** The language \( L_a \) is accepted by the CCA shown in Figure 4.1. The CCA accepting this language is the automaton \( A = (Q, \Sigma, \Delta, \{q_0\}, F) \) where \( Q = \{q_0, q_1\}, q_0 \) is the only initial state and \( F = \{q_0\} \). \( \Delta \) consists of:

\[
\Delta = \left\{ (q_0, a, x = 0, [+1], q_0), (q_0, a, x = 1, [0], q_1), (q_0, b, x \geq 0, [0], q_0), (q_1, \Sigma, x \geq 0, [0], q_1) \right\}
\]

The automaton works as follows. Whenever the automaton sees an ‘\( a \)’ it increases the counter corresponding to the data value. On ‘\( b \)’ it does nothing. The automaton moves to a non-final state if it sees an ‘\( a \)’ with the data value whose corresponding counter value is 1.

Since the automaton above is deterministic, by complementing it, that is, set-
Chapter 4. Class counting automata

\[ a, x = 0, [+1] \quad a, x \geq 0, [0] \]
\[ b, x \geq 0, [0] \quad b, x \geq 0, [0] \]

Figure 4.1: CCA accepting the language \( L_a \)

\[ \Sigma, x = 0, [0] \quad \Sigma, x \geq 0, [0] \]
\[ \Sigma, x = 0, [+1] \quad \Sigma, x = 1, [0] \]

Figure 4.2: CCA accepting the language \( L_{dd} \).

In Example 4.2.3, we can accept the language \( \overline{L_a} = “\text{There exists a data value appearing at least twice under } a” \).

**Example 4.2.3.** Since a finite state automaton can be viewed as a CCA which does not increase its counters, the language \( L_{a^*b^*} \) is recognizable by CCA.

**Example 4.2.4.** The language \( L_{dd} \) is accepted by a CCA in the following way (shown in Figure 4.2). The automaton starts in the initial state \( q_0 \) with all its counters carrying value 0. Initially the automaton leaves the counters untouched. At some point during the run the automaton nondeterministically increases the counter value to 1 and moves to the state \( q_1 \). In the next step the automaton verifies that the counter corresponding to the current data value is 1 and if so the automaton moves to the final state \( q_f \) and stays there for the rest of the word.

**Example 4.2.5.** The family of languages \( L_{\exists n} \) is accepted by CCA with \( (n + 1) \)-states \( q_0, \ldots, q_n \) in the following way (depicted in Figure 4.3). Each fresh data value is marked by increasing the counter corresponding to them and the number of distinct data values is seen is kept in the state. Finally the word is accepted if the number reaches \( n \).
Chapter 4. Class counting automata

Example 4.2.6. The language $L_{<n}$ is accepted by a CCA in the following fashion (shown in Figure 4.4). The automaton starts in the initial state $q_0$ which is also a final state. During the run the multiplicity of each data value is kept in the counters. If for some data value the multiplicity exceeds $n$ the automaton moves to a non-initial state $q_1$.

Example 4.2.7. Fix $\Sigma$ to be $\{a\}$. Let the language $L_2$ be: “There exists a data value whose multiplicity is not two.” The CCA accepting this language is the automaton $A = (Q, \Sigma, \Delta, q_0, F)$ where $Q = \{q_0, q_1, q_2, q_3\}$, $q_0$ is the only initial state and $F = \{q_1, q_3\}$. $\Delta$ consists of:

$$\Delta = \left\{ (q_0, a, x = 0, [+1], q_1), (q_0, a, x = 0, [0], q_0), (q_1, a, x = 1, [+1], q_2), \\
(q_1, a, x = 0, [0], q_1), (q_2, a, x = 2, [+1], q_3), (q_2, a, x = 0, [0], q_2), \\
(q_3, a, x \geq 0, [0], q_3) \right\}$$

The automaton is shown in the Figure 4.5. The idea is that the automaton chooses non-deterministically a data value and faithfully counts its multiplicity, while keeping the counters of other data values zero. Finally the automaton accepts the word, if the current count is not two.
But as we show below, its complement language, $L_2^c = \text{“All data values occur exactly twice”}$ is not recognizable. Thus, CCA-recognizable data languages are not closed under complementation.

**Proposition 4.2.8.** The language $L_2^c = \text{“All data values occur exactly twice”}$ is not recognizable.

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

$$w = (a, d_1)(a, d_2)\ldots(a, d_{m+1})(a, d_1)(a, d_2)\ldots(a, d_{m+1})$$

Clearly, $w \in L_2^c$. 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 $u \cdot v \cdot v'$, such that the configuration after $u$ is $(q, h)$ and after $v$ it is $(q, h_1)$. Then by the remarks we made earlier, we can find an accepting run for $u \cdot v'$ as well. But then $u \cdot v'$ is not in $L_2^c$.

**Proposition 4.2.9.** CCA-recognizable data languages are closed under union and intersection but not under complementation.

**Proof.** Closure under union is easily obtained by non-determinism. Closure under intersection requires the use of more than one bag which we will discuss later.

The following observation will be useful for decision questions that follow. Given a CCA $A = (Q, \Sigma, \Delta, I, F)$ let $m$ be the maximum constant used in $\Delta$. We define the following equivalence relation on $\mathbb{N}$, $e \simeq_{m+1} e'$, $e, e' \in \mathbb{N}$ iff $e < (m + 1) \lor e' < (m + 1) \Rightarrow e = e'$. Note that if $e \simeq_{m+1} e'$ then a transition is enabled at $e$ if and only if it is enabled at $e'$. 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 \(\forall d \in \Gamma, h_1(d) \simeq_{m+1} h_2(d)\).

**Lemma 4.2.10.** If \(c_1, c_2\) are two configurations of the CCA such that \(c_1 \simeq_{m+1} c_2\), then \(\forall w \in (\Sigma \times \Gamma)^*, c_1 \vdash_w c_1' \implies \exists c_2', c_2 \vdash_w c_2'\) 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 \(w \cdot (a, d)\). 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 \(c_1' \vdash_{(a, d)} c_1''\) then using the above argument there exists \(c_2''\) such that \(c_2' \vdash_{(a, d)} c_2''\) and \(c_1'' \simeq_{m+1} c_2''\). \(\square\)

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

### 4.3 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 4.3.1.** The non-emptiness problem for CCA is EXPSPACE-complete.

#### 4.3.1 Upper bound

We reduce the emptiness problem of CCA to the covering problem on Petri nets ([Esp96]). For checking emptiness, we can omit the \(\Sigma\) labels from the configuration graph; we are then left only with counter behavior. However since we have unboundedly many counters, we are led to the realm of multi-counter automata, or vector addition systems.

**Definition 4.3.2.** An \(\omega\)-counter machine \(B\) is a tuple \((Q, \Delta, I)\) where \(Q\) is a finite set of states, \(I \subseteq Q\) is the set of initial states and \(\Delta \subseteq_{fin} (Q \times C \times \text{Inst} \times \mathbb{N} \times Q)\).
Proposition 4.3.4. A configuration of $B$ is a pair $(q, h)$, where $q \in Q$ and $h : \mathbb{N} \to \mathbb{N}$. The initial configurations of $B$ are of the form $(q_0, h_0)$ where $q_0 \in I$ and $h_0(i) = 0$ 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 4.3.3.** Checking emptiness for CCA can be reduced to checking reachability for $\omega$-counter machines.

**Proof.** It suffices to show, given a CCA, $A = (Q, \Sigma, \Delta, I, F)$, where $F = \{q\}$, that there exists a counter machine $B_A = (Q', \Delta', I)$ such that $A$ has an accepting run on some data word exactly when $B_A$ can reach $q$. (When $F$ is not singleton, we simply repeat the construction.) $\Delta'$ is obtained from $\Delta$ by converting every transition $(p, a, \pi, m, q)$ to $(p, c, \pi, m, q)$. 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 $g : \mathbb{N} \to \Gamma$ be an enumeration of data values. It is easy to see that $\gamma' = (q_1, h_0 \circ g)(q_1, h_1 \circ g)\ldots(q_n, h_n \circ g)$ is a run of $B_A$ reaching $q$.

$(\Leftarrow)$ Suppose that $B_A$ has a run $\eta = (q_0, h_0)(q_1, h_1)\ldots(q_n, h_n)$, $q_n = q$. It can be seen that $\eta' = (q_0, h_0 \circ g^{-1})(q_1, h_1 \circ g^{-1})\ldots(q_n, h_n \circ g^{-1})$ is an accepting run of $A$ on $w = (a_1, d_1)\ldots(a_n, d_n)$ where $w$ satisfies the following. Let $(p, c, \pi, m, q)$ 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 $(p, a, \pi, m, q)$ in $\Delta$. Then it should be the case that $a_{i+1} = a$ and $d_{i+1} = g(d_k)$.

**Proposition 4.3.4.** Checking non-emptiness of $\omega$-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 co-finite subsets of $\mathbb{N}$. Given $s \subseteq \mathbb{N}$, $Bnd(s)$ is defined to be the least number greater than all the elements in $s$. If $s$ is a co-finite subset of $\mathbb{N}$, $Bnd(s)$ is defined to be $Bnd(\mathbb{N}\setminus s)$.
Chapter 4. Class counting automata

Proof.

Given an \( \omega \)-counter machine \( B = (Q, \Delta, q_0) \) let

\[
m_B = \max\{ \text{Bnd}(s) \mid s \models c, c \text{ is used in } \Delta \}.
\]

It is worth noting that \( m_B \) is of size \( \mathcal{O}(|A|) \).

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

- \( S = \{ P_q \mid q \in Q \} \cup \{ P_i \mid i \in \mathbb{N}, 1 \leq i \leq m_B \} \).

- \( T \) is defined according to \( \Delta \) 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

  - \( t = \{ P_p, P_i \} \) and \( t^* = \{ P_q, P_i' \} \), where (i) if \( \pi \) is inc then \( i' = \min\{m_B, i+n\} \), and (ii) if \( \pi \) is reset then \( i' = \min\{m_B, n\} \).

Formally we define \( T \) as follows. Given a transition \( \delta = (p, c, \pi, n, q) \in \Delta \), let \( I(\delta) \subseteq (\{0, 1, \ldots, m_B\} \times \{0, 1, \ldots, m_B\}) \) be the pairs \( (i, i') \) such that,

\[
I(\delta) = \left\{ (i, i') \mid i \models c, \pi = \text{inc}, i' = \min\{m_B, i+n\} \right\} \\
\left\{ (i, i') \mid i \models c, \pi = \text{reset}, i' = \min\{m_B, n\} \right\}
\]
Finally, $T$ is defined as,

$$T = \bigcup_{\delta=(p,c,\pi,n,q) \in \Delta} \left\{ \begin{array}{l} \{P_p, P_i\}, \{P_q, P_{i'}\} \mid i \neq 0, i' \neq 0, (i, i') \in I(\delta) \\ \{P_p\}, \{P_q, P_{i'}\} \mid i' \neq 0, (0, i') \in I(\delta) \\ \{P_p\}, \{P_q\} \mid i \neq 0, (i, 0) \in I(\delta) \\ \{P_p\}, \{P_q\} \mid (0, 0) \in I(\delta) \end{array} \right\}$$

- The flow relation $F$ is defined according to $t^* t$ and $t^*$ for each $t \in T$.

- The initial marking is defined as follows. $M_0(P_{q_0}) = 1$ and for all $p$ in $S$, if $p \neq q_0$ then $M_0(P_p) = 0$.

Let $M$ be any marking of $N_B$. We say that $M$ is a state marking if there exists $q \in Q$ such that $M(P_q) = 1$ and $\forall p \in Q$ such that $p \neq q$, $M(P_p) = 0$. When $M$ is a state marking, and $M(P_q) = 1$, we speak of $q$ as the state marked by $M$. For $q \in Q$, define $M_f(P_q)$ 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 $M(P_q) > 0$, 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 $M_f(P_q)$. We define the following equivalence relation on $\mathbb{N}$, $m \simeq_{m_B} n$ iff $(m < m_B) \lor (n < m_B) \Rightarrow m = n$. We can lift this to the bags (in $\omega$-counters) in the natural way: $h \simeq_{m_B} h'$ iff $\forall i (h(i) < m_B) \lor (h'(i) < m_B) \Rightarrow h(i) = h'(i)$. 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 $\mu$ be a mapping of $B$-configurations to $N_B$-configurations as follows: given $\chi = (q, h)$, define $\mu(\chi) = M_\chi$, where

$$M_\chi(P_p) = \begin{cases} 1 & \text{iff } p = q \\ 0 & \text{iff } p \in Q \setminus \{q\} \\ |[i]| & \text{iff } P_p = P_{i} \end{cases}$$

Above $[i]$ denotes the equivalence class of $i$ 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 $M_f(P_q)$. Conversely if a reachable marking $M$ of $N_B$ covers $M_f(P_q)$, for some $q \in Q$, then there exists a reachable configuration $\chi = (q, h)$ of $B$ such that $\mu(\chi) = M$.

From the claim it follows that checking reachability of $q$ in $B$ reduces to checking reachability of a marking which covers $M$ such that $M(P_q) = 1$ and for all other places $p$, $M(p) = 0$.

$(\Rightarrow)$ The proof is by induction on the length of the $B$-run. For the base case, observe that $\mu(\chi_0) = M_0$, which is a state marking that marks $q_0$. Assume that for every run of length $n$ the claim is true.

Suppose that $\chi = (q, f)$ is a configuration reachable in $n$ steps, and that the transition $t = (q, c, \pi, m, q')$ can be taken at $\chi$ on counter $i$ such that $f(i) = c$, resulting in the configuration $\chi' = (q', f')$. By induction hypothesis there exists a marking $M$ such that $\mu(\chi) = M$. By definition of $\mu$ it is the case that $M(P_q) = 1$.

If $f(i) = 0$ then the transition $t_0 \in T$ with $\bullet t_0 = \{P_q\}$ is enabled (since its only input place, namely $P_q$ contains a token) and is fired. In the resulting marking $M'$, if $q \neq q'$ then $M'(P_q) = 0$ and $M'(P_q') = 1$, else $M'(P_q) = M(P_q)$ since $P_q \in t_0$. If $f(i)$ is updated to $f'(i) = 0$ then $t_0' = \{P_q'\}$, which means the transition $t$ did not increment the counter $i$ or reset it to zero. In which case for all $u \in [m_B]$ it is the case that $M'(u) = M(u)$. Hence $\mu(\chi') = M'$. If $f(i)$ is updated to $f'(i) > 0$ then $t_0' = \{P_q', P_{v'}\}$ where $v' \simeq_{m_B} f'(i)$, in which case, $M'(P_{v'}) = M(P_{v'}) + 1$ and for all $u \in [m_B]\{v'\}$ is the case that $M'(u) = M(u)$. Hence again $\mu(\chi') = M'$.

If $f(i) > 0$ then there exists $v \in [m_B]$ such that $M(P_v) > 0$ and $v \simeq_{m_B} f(i)$. Then $t_v \in T$ with $\bullet t_v = \{P_q, P_v\}$ is enabled and is fired. Again, in the resulting marking $M'$, if $q \neq q'$ then $M'(P_q) = 0$ and $M'(P_q') = 1$, else $M'(P_q) = M(P_q)$, since $P_q \in t_v$. Then $f'(i)$ is updated to $f'(i) = 0$ then $M'(P_v) = M(P_v) - 1$ and for all $u \in [m_B]\{v\}$ it is the case that $M'(u) = M(u)$. Hence $\mu(\chi') = M'$. If $f(i)$ is updated to $f'(i) > 0$ then $t_v' = \{P_{v'}, P_{v'}\}$ where $v' \simeq_{m_B} f'(i)$, in which case, if $v \neq v'$ then $M'(P_v) = M(P_v) - 1$, $M'(P_{v'}) = M(P_{v'}) + 1$ and for all $u \in [m_B]\{v, v'\}$ it is the case that $M'(u) = M(u)$. If $v = v'$ then for all $u \in [m_B]$ it is the case that $M'(u) = M(u)$. Hence again, $\mu(\chi') = M'$.

Thus $\mu(\chi')$ is reachable from $M$ in one step by firing $t'$.

$(\Leftarrow)$ The proof in the other direction is similar. We do induction on the length
of the $N_B$-marking sequence. For the base case, as in the previous case $\mu(\chi_0) = M_0$. Assume that for every marking sequence of length $n$ the claim is true.

We are considering only one case below; other cases follow similarly. Suppose that $M$ is a marking reachable in $n$ steps, and that the transition $t_v = (\{P_q, P_v\}, \{P_{q'}, P_{v'}\})$, $q, q' \in Q, v, v' \in [m_B]$ is enabled at $M$ and is fired resulting in the marking $M'$. By induction hypothesis there exists a $B$-configuration $\chi = (q, f)$ such that $\mu(\chi) = M$. There exists an $i \in \mathbb{N}$ such that $f(i) \simeq_{m_B} v$ since $M(P_v) > 0$. By construction, the transition $t_v$ was formed from a transition $t = (q, c, \pi, m, q'), t \in \Delta$ in $B$ such that $v \models c$ and therefore $f(i) \models c$. Therefore the transition can be taken in $B$ resulting in configuration $\chi' = (q', f')$ such that updating $f(i)$ with respect to $\pi$ and $m$ will result in a value $f'(i)$ which is $m_B$-equivalent to $v'$. This is by virtue of the construction of $t_v$. Hence, $\mu(\chi') = M'$.

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

**Complexity of Emptiness checking:** The decision procedure discussed above runs in EXPSPACE [Esp96], and thus we have elementary decidability. Note that the representation of constants in unary is a crucial assumption about the EXPSPACE upper bound. When the constants are represented in binary, we do not know whether the upper bound still holds.

### 4.3.2 Lower bound

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 multicounter automata.

A $k$-multicounter 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 $\Delta \subseteq_{fin} (Q \times \Sigma \times N^k \times N^k \times Q)$. The two vectors in the transition specify decrements and increments of the counters.

The automaton works as follows: it has $k$-counters, denoted by $\vec{v} = (v_1, \ldots v_k)$ 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 \((q, a, v_{\text{dec}}, v_{\text{inc}}, q')\) such that \(\bar{v} - v_{\text{dec}} \geq \bar{0}\) (pointwise) and \(\bar{v}' = \bar{v} - v_{\text{dec}} + v_{\text{inc}}\). 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 \text{EXPSPACE-hard} \cite{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 \(j \neq i, b_j = 0\). That is \(M'\) decrements or increments at most one counter in each transition.

\(\Delta'\) is obtained as follows. Let \(t = (q, a, v_{\text{dec}}, v_{\text{inc}}, q')\). Let \(\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n\) be a sequence of unit vectors such that \(v_{\text{dec}} = \sum_i \bar{u}_i\) and \(\bar{u}'_1, \bar{u}'_2, \ldots, \bar{u}'_m\) be a sequence of unit vectors such that \(v_{\text{inc}} = \sum_i \bar{u}'_i\). We add intermediate states to rewrite \(t\) by the following sequence of transitions,

\begin{align*}
(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'), a, \bar{0}, \bar{u}'_m, q')
\end{align*}

**Lemma 4.3.5.** \(L(M)\) 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 \(\rho\) of \(M\) we have an accepting run \(\rho'\) of \(M'\), this is achieved by replacing every transition \(t\) in the run \(\rho\) by the corresponding sequence of transitions. For the reverse direction, we need to show that every run accepting run \(\rho'\) of \(M'\) can be translated to an accepting run \(\rho\) 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 occurring in \(\rho'\) which are from \(Q\) we can show that there is a valid run \(\rho\) which is accepting. \(

42
Next we convert $M'$ to a CCA thus establishing a lower bound 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 the $i$-th unit vector and \( \bar{u}' \) is a zero vector, we add a transition $t_A = (q, a, (x = i), (\text{reset}, 0), q')$ to $\Delta_A$. If \( \bar{u} \) is the $i$-th unit vector and \( \bar{u}' \) is the $j$-th unit vector, we add a transition $t_A = (q, a, (x = i), (\text{reset}, j), q')$ to $\Delta_A$. If \( \bar{u} \) is a zero vector and \( \bar{u}' \) is the $j$-th unit vector, we add a transition $t_A = (q, a, (x = 0), (\text{reset}, j), q')$ to $\Delta_A$.

**Lemma 4.3.6.** $L(M')$ 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 $\chi$ reachable by $A$ there is a configuration $\psi$ of $M'$ such that $\mu(\chi) = \psi$ and conversely for every configuration $\psi$ reachable by $M'$ there is a configuration $\chi$ 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 $t = (q, a, x = j, (\text{reset}, i), q')$ is enabled at $\chi$. Therefore there is a counter holding the value $j$. By induction hypothesis there exists a configuration $\psi$ 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 $\Delta'$, preserving the map $\mu$.

Conversely, suppose a configuration $\psi = (q, \bar{v})$ is reachable by $M'$ in $l$ steps. Then by induction hypothesis we have a configuration $\chi$ 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 $\psi$ resulting in $\psi'$. 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 $t = (q, a, (x = i), \text{reset}, j, q')$. We choose the smallest counter holding the value zero and apply the transition $t$, resulting in $\xi'$ such that $\mu(\xi') = \psi'$. The remaining cases are similar. \(\square\)
The reduction from $M$ to $M'$ is not in polynomial time when the constants in the transitions of the Multicounter automata are encoded in binary. However, we observe that the EXPSPACE-hardness for covering problem from [Esp96, Lip76] can be obtained with updates restricted to the values $-1, 0$ and $1$. Hence, the lower bound extends to the scenario where the constants are represented in binary.

4.3.3 Word problem

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 non-deterministic 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 4.3.7.** The word problem for CCA is NP-complete.

**Proof.** 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 $\varphi \equiv (p_1 \lor \neg p_3 \lor p_4) \land (\neg p_2 \lor p_5 \lor p_1) \land (\neg p_3 \lor \neg p_4 \lor p_5)$, we construct the corresponding word over the alphabet $\{+, -, \#\} \times \Gamma$, $w = (+, d_1)(-, d_3)(+, d_4)(\#, d)(-, d_2)(+, d_5)(+, d_1)(\#, d)(-, d_3)(-, d_4)(+, d_5)(\#, d)$

The non-deterministic automaton checks satisfiability in the following way. Every time the automaton encounters a new data value (representing a propositional
variable), the automaton non-deterministically 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 $\top$.

4.4 Extensions and subclasses

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

4.4.1 Deterministic CCA

To define the deterministic subclass of CCA, we need a way of ensuring that nondeterminism is only on $Q$. Towards this, we say that two constraints $c_1$ and $c_2$ are non-intersecting if there does not exist $v \in \mathbb{N}$ such that $v \models c_1$ and $v \models c_2$. Observe that any automaton can be converted to an automaton in which the transitions are such that:

- If $(q, a, c_1, \pi_1, m_1, q_1) \in \Delta$, $(q, a, c_2, \pi_2, m_2, q_2) \in \Delta$ and $c_1 \neq c_2$, then $c_1$ and $c_2$ are non-intersecting.

An automaton $A$ is a deterministic class counting automaton (DCCA) if it is a CCA with the property mentioned above and whenever $(q, a, c, \pi_1, m_1, q_1) \in \Delta$ and $(q, a, c, \pi_2, m_2, q_2) \in \Delta$, we have $\pi_1 = \pi_2$, $m_1 = m_2$ and $q_1 = q_2$. Since the size of the configuration is bounded by the size of the data word, the word problem of DCCA is in $P$. Also by an easy reduction from Monotone-CVP we can show that the problem is $P$-hard.

**Proposition 4.4.1.** The word problem for DCCA is $P$-complete.

**Proof.** It is easy to see that the size of the configuration of an automaton on a word is bounded by the length of the word. Hence checking membership is polynomial
time in the length of the word, hence in $P$. For completeness we reduce the circuit valuation problem (CVP) to the membership problem of a CCA. Circuit valuation problem asks the following question: Given a circuit $C$ and a valuation $V$, does the circuit evaluate to $\top$? We assume that the circuit is presented in a topologically sorted order. For example, let the circuit be

$$c_0 = p_0 \lor \neg p_1, c_1 = \neg p_0 \land p_2, C = c_0 \lor c_1$$

and the valuation be $(p_0, 0), (p_1, 1), (p_2, 1)$. We construct a word $w$ coding both the circuit and the evaluation in the following way,

$$w = (\bot, d_0)(\top, d_1)(\top, d_2); (d_1, d_2)(+, d_0)(-, d_1)(\lor, c_0)(-, d_0)(+, d_2)(\land, c_1)(+, c_0)(+, c_1)(\lor, C)$$

Here the data values $d_0, d_1, \ldots$ stand for the input variables and $c_0, c_1, \ldots$ represent the gates. The automaton works in two phases. In the first phase, before encountering the letter ‘;’, the automaton consults the letters from $\{\top, \bot\}$ to initialize the counter corresponding to the data values to either 1 (for $\bot$) or 2 (for $\top$). Once the automaton reaches the letter ‘;’, it moves on to the evaluation phase where it evaluates each gate and stores the output value of the gate in the counter corresponding to the data value denoting the gate. Computing the output value of a gate depends on the value of the input values (appropriated with their signs, + or −) and the type of gate ($\lor$ or $\land$). Finally the automaton accepts if the last gate has value $\top$.

The restriction of determinism makes DCCA strictly weaker than CCA as shown by the following proposition.

**Proposition 4.4.2.** The language $L_{dd}$ is not accepted by any DCCA.

**Proof.** The proof is by contradiction. Assume $L_{dd}$ is accepted by a DCCA with $m$ states. Consider the data word $w = (a, d_1)(a, d_2)\ldots(a, d_n)$ such that all data values are distinct and $n = 2 \cdot m + 1$. Let $C_0, C_1, C_2 \ldots C_n$ be the unique run of the automaton on $w$, where $C_i = (q_i, h_i)$. By pigeonhole principle there are two configurations $C_i$ and $C_j$, $1 \leq i < j \leq n$, such that $q_i = q_j$ and $h_i(d_i) = h_j(d_j)$. Let $w \mid_i = (a, d_1)(a, d_2)\ldots(a, d_i)$ be the prefix of $w$ of length $i$. Since $w \mid_j \cdot (a, d_j) \in L_{dd}$, there is a transition $t$ enabled at $C_j$ on $(a, d_j)$ such that $C_j \xrightarrow{t} C_f$, where $C_f$ is
Chapter 4. Class counting automata

a final configuration. Since $\bar{h}_i(d_i) = \bar{h}_j(d_j)$ and all data values are distinct, $t$ is enabled at $C_j$ on $(a,d_i)$ also. Therefore the automaton accepts $w | j \cdot \left( a, d_i \right)$ as well, though it is not in the language.

Recall that $L_{dd}$ on the other hand is accepted by a register automaton. This along with the fact that $L_a$ is accepted by a DCCA (which is not accepted by register automata) shows that;

**Theorem 4.4.3.** DCCA and Register automata are incomparable in terms of expressive power.

### 4.4.2 Many bags

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. An interesting fact is that a CCA with $k$-bags can be converted to a CCA with one bag. This can be achieved because of the following:

- Any CCA, no matter how many bags it has, can be converted to a CCA whose counter values are bounded (We take the maximum constant used in $\Delta$ and rewrite the transitions in such a way that we never increment a counter once it reaches that value). This is a direct consequence of Lemma 4.2.10.

- A $k$-bag CCA whose counters are bounded can be simulated by a CCA with one bag, by using a bit representation. Since the counters are bounded, we know a priori how many bits are needed to represent each bag.

Now we are ready to show that CCA are closed under intersection.

**Proposition 4.4.4.** CCA are closed under intersection.

**Proof.** Given two CCA $A_1$ and $A_2$ with state spaces $Q_1$ and $Q_2$ respectively, we construct a CCA $A$ with two bags and state space $Q_1 \times Q_2$ such that $A$ simulates $A_1$ and $A_2$. The automaton utilizes its first bag for simulating $A_1$’s counters and
Chapter 4. Class counting automata

second bag for $A_2$’s counters. Now above discussion shows that $A$ can be converted to a CCA with only one bag and hence the proposition.

\[\square\]

4.4.3 Checking any counter

Another strengthening involves checking for the presence of any counter satisfying a given constraint and updating it. The idea is to extend the transitions to the following form, $t = (q, a, \tau_0, \tau_1, \ldots, \tau_n, q')$ where each $\tau_i \in C \times \text{Inst} \times \mathbb{N}$ is of the form $(c_i, \pi_i, m_i)$. The intended semantics of the transition is as follows. Suppose that the current letter is $a$ and data value is $d_0$. The transition $t$ is enabled if there exist distinct data values $d_1, \ldots, d_n$ such that, for every $i \in [n]_0$, $d_i$ satisfies $\tau_i$. On the occurrence of $t$ each $d_i$ is updated with respect to $\tau_i$. Note that in this way we can modify the counter of a data value which is not the current data value.

Formally a CCA with context check, denoted CCAC, is a tuple $(Q, n, \Delta, I, F)$, where the transition relation is modified to be $\Delta \subseteq \text{fin}(Q \times \Sigma \times (C \times \text{Inst} \times \mathbb{N})^n \times Q)$ where $n \in \mathbb{N}$.

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

Given a data word $w = (a_1, d_1), \ldots, (a_m, d_m)$, a run of $A$ on $w$ is a sequence $\gamma = (q_0, h_0)(q_1, h_1) \ldots (q_m, h_m)$ such that $q_0 \in I$ and for all $i, 0 \leq i < m$, there exists a transition $t_i = (q_i, a_i, \tau_i) \ldots, \tau_n, q') \in \Delta$ where $\tau_j = (c_j, \pi_j, m_j)$ such that $q = q_i$, $q' = q_{i+1}$, $a = a_{i+1}$ and:

- $h_i(d_{i+1}) \models c_0$ and there exist distinct $e_1, \ldots, e_n$ in $\Gamma$ such that for all $j \in \{1, \ldots, n\}$, $e_j \neq d_{i+1}$ and $h_i(e_j) \models c_j$.
- $h_{i+1}$ is given by:

$$h_{i+1} = \begin{cases} h_i \oplus (d_{i+1}, m') & \text{if } \pi_0 = \text{inc}, m' = h_i(d_{i+1}) + m_0 \\ h_i \oplus (d_{i+1}, m_0) & \text{if } \pi_0 = \text{reset} \\ h_i \oplus (e_j, m') & \text{if } \pi_j = \text{inc}, m' = h_i(e_j) + m_j \\ h_i \oplus (e_j, m_j) & \text{if } \pi_j = \text{reset} \end{cases}$$
We define $\omega$-counter machines with context in a similar way: such a machine is a tuple $(Q, \Delta, q_0)$ where $Q$ is a finite set of states, $q_0$ is the initial state and $\Delta \subseteq \text{fin}(Q \times (C \times \text{Inst} \times \mathbb{N})^n \times Q)$. A run of an $\omega$-counter machine with context is defined analogously to that of CCA with context. We can then easily show that checking emptiness for CCA with context can be reduced to checking reachability for $\omega$-counter machines with context.

Finally, the following proposition shows that checking emptiness of CCA with context is decidable in ExpSpace.

**Proposition 4.4.5.** Checking non-emptiness of $\omega$-counter machines with context is decidable in EXPSPACE.

**Proof.** Given an $\omega$-counter machine $B = (Q, \Delta, q_0)$, we define $m_B$ as in the proof of Proposition 4.3.4.

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 $\Delta$ as follows. Let $t = (q, a, \tau_0, \tau_1, \ldots, \tau_n, q')$ be a transition in $\Delta$ where $\tau_j = (c_j, \pi_j, m_j)$ and let $i_0, i_1, \ldots, i_n$ be such that $0 \leq i_j \leq m_B$ and $i_j \models c_j$. Then we add a transition $t$ such that $\cdot t = \{p, i_0, i_1, \ldots, i_n\}$ and $t^\cdot = \{q, i'_0, i'_1, \ldots, i'_n\}$ (take note of the fact that $\cdot t$ and $t^\cdot$ are multisets), where (i) if $\pi_j$ is inc then $i'_j = \min\{m_B, i_j + n_j\}$, and (ii) if $\pi_j$ is reset then $i'_j = \min\{m_B, n_j\}$. Note that $i_j$ can be zero, in which case we add edges only for the places in $[m_B]$.

- The flow relation $F$ is defined according to $\cdot t$ and $t^\cdot$ 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 rest of the proof is similar to the proof of Proposition 4.3.4 with obvious modifications.

Given a $k$-register automaton $A = (Q, \Sigma, \Delta, I, F)$ we can construct a CCA with context which accepts the language $L(A)$.
The way the CCA $A' = (Q', \Sigma', \Delta', q_0', F')$ simulates the register automaton $A$ is as follows. The states of $A'$, namely the set $Q' = Q \times \{0, 1\}^k$ stores two kinds of information, the current state of the automaton $A$ and the registers which store a data value (0 indicates the register is holding $\perp$ and 1 indicates the register is holding a data value). When a register write takes place, if the bit corresponding to the written register is 0 it is updated to 1. The information that which data value is in which register is stored in the counter corresponding to the data value. This is done in the following manner. If the counter corresponding to a data value $d$ has value $i$, $1 \leq i \leq k$, it means that the register $i$ contains the data value $d$. We also make sure that exactly one counter holds the value $i$ at any time. Suppose $\Delta$ contains a read transition $(p, a, i, q)$, we add the set of transitions\{(p, \bar{v}), a, x = i, [0], (q, \bar{v})\} | \bar{v} \in \{0, 1\}^{i-1} \times \{1\} \times \{0, 1\}^{k-i-1}\}. Suppose $\Delta$ contains a write transition $(p, a, q, i)$, we add the set of transitions\{(p, \bar{v}), a, (x \geq 0), (y = i), (\text{reset}, i), (\text{reset}, 0), (q, \bar{v}') | \bar{v} \in \{0, 1\}^k, \bar{v}' = \bar{v} + u_i\} to $\Delta'$ ($u_i$ is the $i$-th unit vector). The initial state $q_0' = (q_0, 0^k)$, and final states are $F' = \{(q, \bar{v}) | q \in F, \bar{v} \in \{0, 1\}^k\}$. We omit the proof here since it is straightforward. It follows that;

**Proposition 4.4.6.** Register automata are strictly weaker than CCA with context in terms of expressiveness.

### 4.4.4 The language of constraints

The language of constraints can be strengthened. Previously, the constraints where of the form $x \leq e$ or $x \geq e$. Consider the following language, the language of Presburger arithmetic. The terms in this language are given by the grammar,

$$t ::= 0 | 1 | t_1 + t_2 | x, \ x \in V$$

where $V$ is a countably infinite set of variables. The formulas of this language are given by:

$$\varphi ::= t_1 \leq t_2 | \lnot \varphi | \varphi_1 \lor \varphi_2 | \exists x. \varphi.$$

The semantics is given as follows. The variables take natural numbers as their values and $+$ is interpreted as addition. We call a formula $\varphi(x)$ with one free variable, a Presburger constraint. We say that $k \in \mathbb{N}$ satisfies $\varphi(x)$ if $k \models \varphi(x)$. 

50
Note that the set of numbers satisfying a constraint may be neither finite nor co-finite. For example, the formula $\exists y. y + y = x$ defines the set of even numbers.

Let $C_p$ be the set of all Presburger constraints. We define CCA with Presburger constraints, abbreviated as CCA + Presburger, as a tuple $\text{CCA} = (Q, \Sigma, \Delta, I, F)$, where the transition relation is modified to be $\Delta \subseteq_{\text{fin}} (Q \times \Sigma \times C_p \times \text{Inst} \times \mathbb{N} \times Q)$. The definitions of run and acceptance condition is defined in the obvious way.

A set of natural numbers $D$ is eventually periodic iff there exists positive numbers $m$ and $p$ such that for all $n$ greater than $m$, $n \in D$ iff $n + p \in D$. From End72, we know that the set of numbers satisfying a Presburger constraint is eventually periodic.

Using this, the decision procedure in Section 3 can be modified to check the emptiness of CCA with Presburger constraints. As above, we define $\omega$-counter machines with Presburger constraints: such a machine is a tuple $(Q, \Delta, q_0)$ where $Q$ is a finite set of states, $q_0 \in Q$ is the initial state and $\Delta \subseteq_{\text{fin}} (Q \times C_p \times \text{Inst} \times \mathbb{N} \times Q)$. Runs are defined in the natural way.

We can then easily show that checking emptiness for CCA with Presburger constraints can be reduced to checking reachability for $\omega$-counter machines with Presburger constraints. Then the following proposition shows that checking emptiness of CCA with Presburger constraints is decidable in $\text{ExpSpace}$.

**Proposition 4.4.7.** Checking non-emptiness of $\omega$-counter machines with Presburger constraints is in $\text{ExpSpace}$.

**Proof.** Given an $\omega$-counter machine $B = (Q, \Delta, q_0)$, let $c_1, \ldots, c_n$ be the constraints used in $\Delta$. From End72, we know that $c_1, \ldots, c_n$ are eventually periodic with the pairs $(m_1, p_1), \ldots, (m_n, p_n)$. We take $m = m_1 + \ldots + m_n$ and $p$ as the least common multiple of $p_1, \ldots, p_n$.

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

- $S = Q \cup \{i \mid i \in \mathbb{N}, 1 \leq i \leq m + p\}$.
- $T$ is defined according to $\Delta$ as follows. Let $(p, c, \pi, n, q) \in \Delta$ and let $i$ be such that $0 \leq i \leq m + p$ and $i \models c$. Then we add a transition $t$ such that $t^* = \{p, i\}$ and $t^* = \{q, i'\}$, where (i) if $\pi$ is inc then $i' = \min\{i+n, m+(i+n-m) \mod p\}$,
and (ii) if \( \pi \) is reset then \( i' = \min\{n, m + (n - m) \mod p\} \). Note that \( i \) can be zero, in which case we add edges only for the places in \([m_B]\).

- The flow relation \( F \) is defined according to \( \bullet t \) and \( t^* \) 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 rest of the proof is similar to the proof of Proposition 4.3.4 with obvious modifications.

### 4.4.5 Two-way CCA

A two-way CCA is system \((Q, \Sigma, \Delta, I, F)\), where \(Q, I, F\) are as usual, the transition relation is \( \Delta \subseteq_{fn} (Q \times \Sigma \times C \times \text{Inst} \times \mathbb{N} \times Q \times \{L, R, S\}) \). A configuration of \( A \) is a triple \((q, i, h)\), where \( q \in Q \), \( i \in \mathbb{N} \) and \( h \in B \), where the variable \( i \) denotes the position of the head. The initial configuration of \( A \) is given by \((q_0, 1, h_0)\), where \( h_0 \) is the empty bag; that is, \( \forall d \in \Gamma, h_0(d) = 0 \) 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 \( \gamma = (q_0, i_0, h_0)(q_1, i_1, h_1) \ldots (q_l, i_l, h_l) \) such that \( q_0 \in I \) and for all \( j, 0 \leq j < l \), there exists a transition \( t_j = (q, a, c, \pi, m, q', \mu) \in \Delta \) such that \( q = q_j, q' = q_{j+1}, a = a_{i_j} \) and \( h_j(d_{i_j}) \models c \). The resulting counter configuration \( h_{j+1} \) is defined as in the case of CCA. Finally, the updated position of the head is determined in the following way:

\[
i_{j+1} = \begin{cases} 
  i_j - 1 & \text{if } \mu = L \\
  i_j + 1 & \text{if } \mu = R \\
  i_j & \text{if } \mu = S 
\end{cases}
\]

We assume that the input word is wrapped with end markers so that if the machine tries to go off the boundary of the word it halts erroneously. We say a run is accepting if the machine halts in a final state.

As we will see below, the emptiness problem is undecidable for the two-way extension of CCAs.
4.4.6 Alternating CCA

An alternating CCA is system \((Q = Q_\forall \cup Q_\exists, \Delta, I)\), where \(Q, I, \Delta\) are as usual. Note that there is no designated set of final states; instead, the state set is partitioned into a set of universal states \(Q_\forall\) and a set of existential states \(Q_\exists\). A configuration of \(A\) is a tuple \((q, h)\), where \(q \in Q\) and \(h \in B\). The initial configuration of \(A\) is given by \((q_0, h_0)\), \(q_0 \in I\) and \(h_0\) is the empty bag; that is, \(\forall d \in \Gamma, h_0(d) = 0\) and \(q_0 \in I\).

Given a data word \(w = (a_1, d_1), \ldots (a_n, d_n)\), assume that the automaton is at position \(i\) with configuration \((q_i, h_i)\). We say that \((q_{i+1}, h_{i+1})\) is a valid successor configuration if there exists a transition \(t = (q, a, \pi, m, q', \mu) \in \Delta\) such that \(q = q_i\), \(q' = q_{i+1}\), \(a = a_{i+1}\) and \(h_i(d_{i+1}) \models c\). The resulting counter configuration \(h_{j+1}\) is defined as in the case of CCA.

We say that a configuration \((q, h)\) is accepting if

1. \(q \in Q_\forall\) and all of its valid successor configurations are accepting. (Note that a configuration with no valid successor configurations is accepting.)

2. \(q \in Q_\exists\) and there is a valid successor configuration \((q', h')\) which is accepting.

Finally we say that the word is accepted if the initial configuration \((q_0, h_0)\) is accepting.

**Theorem 4.4.8.** The emptiness problem is undecidable for Two-way CCAs and for Alternating CCAs.

**Proof.** We do the proofs simultaneously by reducing the Post’s Correspondence Problem to the emptiness of two-way CCA and of alternating CCA. Without loss of generality, assume that we are given a PCP instance \(I\) which is a set of ordered pairs of non-empty strings over the alphabet \(\Sigma = \{l_1, l_2, \ldots l_k\}\), that is \(I = \{(u_i, v_i) \mid i \in [n], u_i, v_i \in \Sigma^+\}\). A solution for \(I\) is a finite sequence of integers \(i_0, i_1, \ldots i_m\), all of which are from the set \(\{1, \ldots n\}\) such that \(u_{i_0} u_{i_1} \ldots u_{i_m} = v_{i_0} v_{i_1} \ldots v_{i_m}\). We define a two-way CCA which accepts precisely all solutions of \(I\).
Chapter 4. Class counting automata

For this purpose, we code the PCP solution as a data word, in the following way. Let $\bar{\Sigma} = \{\bar{l}_1, \bar{l}_2, \ldots, \bar{l}_k\}$ and $\check{\Sigma} = \Sigma \cup \bar{\Sigma}$. Given a word $w = a_1a_2\ldots a_n \in \Sigma^*$, we denote by $\bar{w}$ the word $\bar{a}_1\bar{a}_2\ldots\bar{a}_n \in \bar{\Sigma}^*$.

A solution of $I$ is a data word $w$ over $\check{\Sigma}$ such that,

(I) The string projection of the word is in $(u_1\bar{v}_1 + u_2\bar{v}_2 + \ldots + u_n\bar{v}_n)^+$.

(II) Every data value $d$ occurring in $w$ appears precisely twice, once labelled by a letter from $\Sigma$ and once by a letter from $\bar{\Sigma}$. Moreover if $d$ is labelled by $l_i \in \Sigma$ in $w$ if and only if it is labelled by $\bar{l}_i \in \bar{\Sigma}$ in $v$ (the second occurrence).

(III) The ordering of data values in the positions labelled by $\Sigma$ is exactly the same as the ordering of data values in positions labelled by $\bar{\Sigma}$. Formally, let $d$ and $e$ are data values occurring in $w$. Let $d_\Sigma$ and $e_\Sigma$ be the positions where $d$ and $e$ are labelled by letters from $\Sigma$. Similarly, let $d_{\bar{\Sigma}}$ and $e_{\bar{\Sigma}}$ be the positions where $d$ and $e$ are labelled by letters from $\bar{\Sigma}$. The condition says that $d_\Sigma < e_\Sigma$ if and only if $d_{\bar{\Sigma}} < e_{\bar{\Sigma}}$.

It is easy to see that there is a data word $w$ satisfying the above three conditions iff $I$ has a solution. We show that two-way CCA and alternating CCA can check these three conditions.

1. The first condition is a regular property and can be checked by any finite state automaton. Hence it is easily checked by a CCA.

2. The conjunction of the following four conditions is equivalent to condition (II).

   (a) Data values occurring in $\Sigma$-labelled positions are all distinct.
   (b) Data values occurring in $\bar{\Sigma}$-labelled positions are all distinct.
   (c) All data values occurring under $\Sigma$-labels occur under $\Sigma$-labels as well.
   (d) All data values occurring under $\bar{\Sigma}$-labels occur under $\bar{\Sigma}$-labels as well.

Note that each of these conditions can be checked by a CCA. Since CCAs are closed under intersection, a CCA can verify condition (II).
3. Condition (III) is checked by a two-way CCA in the following way. We assume that conditions (I) and (II) are verified independently. Given a position $i$ labelled by a letter from $\Sigma$ we say that the position $j > i$ is the $\Sigma$-successor of $i$ iff $j$ is a position labelled by a letter from $\Sigma$ and all positions $k$, $i < k < j$ are labelled by letters from $\bar{\Sigma}$. Similarly we can define $\bar{\Sigma}$-successor of a $\Sigma$-labelled position. Let $i$ and $j$ be $\Sigma$-successors and let $d_i$ and $d_j$ be the corresponding data values. We know that $d_i$ and $d_j$ occur under $\bar{\Sigma}$ as well. Let those positions be $\bar{i}$ and $\bar{j}$. For each $\Sigma$-successors $i, j$ the automaton verifies that $\bar{i}$ and $\bar{j}$ are $\bar{\Sigma}$ successors.

To achieve this, assume that the automaton starts in a $\Sigma$ position $i$, it resets the counter of $d_i$ to 1 and goes to next $\Sigma$-labelled position $j$. It increments the counter of $d_j$ to 2. Now, the automaton moves to left end marker and makes a left to right sweep ignoring all $\Sigma$ positions. During this sweep the automaton stops when it sees the data value $d_j$ under a $\bar{\Sigma}$ label. It resets counter of $d_i$ to zero and then verifies that the next $\bar{\Sigma}$ position has the data value $d_j$ with the help of the counter. After this step the automaton goes to the left end of the word and again makes a right sweep. This time it stops when it sees the data value $d_j$ under a $\Sigma$ label. Then the procedure is repeated for position $j$. Finally the machine halts and accepts when it reaches the last $\Sigma$ position in the data word.

4. Condition (III) is checked by an alternating CCA in the following way. The automaton starts in state $q_0$. In this state automaton records all the data values it has seen till the current position. Whenever it sees a fresh data value, it makes a universal branching, one branch continues in state $q_0$ and one branch goes to state $q_1$. In the state $q_1$ the automaton verifies the following. Assume the fresh data value $d$ occurs under a $\Sigma$ label and let the data value on its $\Sigma$ successor position is $e$. The automaton verifies that the positions where $d$ and $e$ are occurring under $\bar{\Sigma}$ labels are $\bar{\Sigma}$ successors. This can easily be done by incrementing the counters corresponding to $d$ and $e$ to specially designated values. The $q_1$ branching halts successfully after each verification. The $q_0$ branching accepts at the end of the word.
Chapter 4. Class counting automata

In the previous proof, conditions (I), (II) and (III) are in fact verified by a universal CCA. This implies that the emptiness problem for universal CCA is undecidable. Since emptiness problem for universal CCA and universality problem for CCA are equivalent it follows that the universality problem for CCA is undecidable, and hence the language inclusion problem for CCA is undecidable.

4.4.7 Counter acceptance conditions

We compare the expressiveness of CCA and CMA.

Proposition 4.4.9. The class of CCA-recognizable languages are strictly contained in the class of CMA-recognizable languages.

Proof. Let $A = (Q, \Sigma, \Delta, I, F)$ be a CCA with $m$ being the maximum constant used in $\Delta$. Let $V = \{0, \ldots, m + 1\}$. We construct a CMA $A_{cma} = (Q', \Sigma, \Delta', I', F'_l, F'_g)$ where $Q' = Q \times V$, $I' = I \times \{0\}$, $F'_l = Q'$, $F'_g = \{(q, v) \in Q' \mid q \in F\}$. $\Delta'$ is defined in the following way,

$$\Delta' = \bigcup_{(q, a, c, \pi, s, q') \in \Delta, (p, v) \in Q'} \left\{ (q, w), a, (p, v), (q', v') \mid v \models c, v' \in V, v' \simeq_{m+1} \pi(v, s) \right\} \left\{ (q, w), a, 0, (q', v') \mid 0 \models c, v' \in V, v' \simeq_{m+1} \pi(0, s) \right\}$$

where $\pi(v, s)$ denotes the result of the operation $\pi$ (one of inc or reset) with argument $s$ on value $v$ and the equivalence is defined as $c \simeq_{m+1} d$ if $\forall i c < m + 1 \vee d < m + 1 \Rightarrow c = d$. From Lemma 4.2.10 it follows that $L(A) = L(A_{cma})$.

The strict containment follows from the fact that CCA do not accept the language $L_2$ (4.2.8) while this language is accepted by a CMA as saw in the last chapter.

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: CCA with counter acceptance conditions $A$ is given by $A = (Q, \Sigma, \Delta, I, F, G)$ where $Q, \Sigma, I, \Delta, F$ are as before, and $G \subset_{fin} N$. We say a final configuration $(q, h)$ is accepting if $q \in F$ and $\forall d \in \Gamma, h(d) \in G$ or $h(d) = 0$.  

56
We then find that the non-emptiness problem continues to be decidable but becomes as hard as Petri net reachability, which is not even known to be elementarily decidable. This is proved by relating this class to that of class memory automata discussed below.

**Proposition 4.4.10.** CCA with counter acceptance conditions are expressively equivalent to CMA.

**Proof.** The proof of Proposition 4.4.9 can be extended to show that the class of languages recognized by CCA with counter acceptance conditions is contained in the class of CMA-recognizable languages. Let \( A = (Q, \Sigma, \Delta, I, F, G) \) be a CCA with counter acceptance condition. Considering \( A \) as a CCA construct \( A'_{cma} = (Q', \Sigma, \Delta', I', F') \) with \( m \) being the maximum constant used in \( \Delta \) and \( G \) as above. Replace the local accepting states \( F_l = Q \times G \) to \( A'_{cma} \) to get \( A_{cma} \). It is easy to see that \( L(A) = L(A_{cma}) \).

For the other direction, let \( A = (Q, \Sigma, \Delta, I, F_i, F_g) \) be a CMA. Let \( Q = \{q_1, q_2, \ldots q_n\} \). We construct a CCA with counter acceptance \( A' = (Q', \Sigma, \Delta', I', F, G) \) as follows. We define \( Q' = Q, I' = I, F = F_g \). The accepting counter configurations are defined as \( G = \{i \mid q_i \in F_i\} \). The transitions \( \Delta' \) is given by,

\[
\Delta' = \bigcup_{(q, a, x, \tau, q_k) \in \Delta} \left\{ (q_i, a, x = j, \text{reset}, k, q_k) \mid \tau = q_j \right\} \\
\bigcup_{(q, a, x, \tau, q_k) \in \Delta} \left\{ (q_i, a, x = 0, \text{inc}, k, q_k) \mid \tau = \bot \right\}
\]

It is easy to see that \( L(A) = L(A') \)

### 4.5 Discussion

In this chapter we introduced the automaton model CCA. This class of automata is strictly weaker than CMA but at the same time has an elementarily decidable emptiness problem. It is also possible to extend this model to match the expressiveness of CMA.

CCA can accept certain languages, for instance \( L_a \) which are not accepted by register automata. The question whether CCA contains register automata is still open. The language \( \overline{L_{dd}} \) is accepted by a register automaton, however it is open
whether $L_{dd}$ is accepted by a CCA. It is possible to extend CCA with context information to include register automata. The language $L_{dd}$ is not accepted by the deterministic subclass of CCA. Since deterministic CCA can accept the language $L_a$ while register automata can not, deterministic CCA and register automata are incomparable in terms of expressiveness.

Regarding the complexity of emptiness checking CCA falls strictly in between register automata and CMA. But with respect to the word problem all these automata have the same complexity.