On Pebble Automata for Data Languages with Decidable Emptiness Problem

Tony Tan

Department of Computer Science, Technion – Israel Institute of Technology
School of Informatics, University of Edinburgh
tantony@cs.technion.ac.il

Abstract. In this paper we study a subclass of pebble automata (PA) for data languages for which the emptiness problem is decidable. Namely, we show that the emptiness problem for weak 2-pebble automata is decidable, while the same problem for weak 3-pebble automata is undecidable. We also introduce the so-called *top view* weak PA. Roughly speaking, top view weak PA are weak PA where the equality test is performed only between the data values seen by the two most recently placed pebbles. The emptiness problem for this model is still decidable.

1 Introduction

Logic and automata for words over finite alphabets are relatively well understood and recently there is broad research activity on logic and automata for words and trees over infinite alphabets. Partly, the study of infinite alphabets is motivated by the need for formal verification and synthesis of infinite-state systems and partly, by the search for automated reasoning techniques for XML. Recently, there has been a significant progress in this field, see [1, 2, 4, 5, 8, 10].

Roughly speaking, there are two approaches to studying data languages: logic and automata. Below is a brief survey on both approaches. For a more comprehensive survey, we refer the reader to [10]. The study of data languages, which can also be viewed as languages over infinite alphabets, starts with the introduction of finite-memory automata (FMA) in [5], which are also known as *register automata* (RA). The study of RA was continued and extended in [8], in which *pebble automata* (PA) were also introduced. Each of both models has its own advantages and disadvantages. Languages accepted by FMA are closed under standard language operations: intersection, union, concatenation, and Kleene star. In addition, from the computational point of view, FMA are a much easier model to handle. Their emptiness problem is decidable, whereas the same problem for PA is not. However, the PA languages possess a very nice logical property: closure under *all* boolean operations, whereas FMA languages are not closed under complementation.

Later in [2] first-order logic for data languages was considered, and, in particular, the so-called *data automata* was introduced. It was shown that data automata define the fragment of existential monadic second order logic for data languages in which the first order part is restricted to two variables only. An important feature of data automata is that their emptiness problem is decidable, even for the infinite words, but is at least as hard as reachability for Petri nets. The automata themselves always work nondeterministically and seemingly cannot be determinized, see [1]. It was also shown that the satisfiability problem for the three-variable first order logic is undecidable.

Another logical approach is via the so called *linear temporal logic with n register freeze quantifier over the labels Σ*, denoted $\mathrm{LTL}_n^{\downarrow}(\Sigma, \mathtt{X}, \mathtt{U})$, see [4]. It was shown that one way alternating n register automata accept all $\mathrm{LTL}_n^{\downarrow}(\Sigma, \mathtt{X}, \mathtt{U})$ languages and the emptiness problem for one way alternating one register automata is decidable. Hence, the satisfiability problem for $\mathrm{LTL}_1^{\downarrow}(\Sigma, \mathtt{X}, \mathtt{U})$ is decidable as well. Adding one more register or past time operators to $\mathrm{LTL}_1^{\downarrow}(\Sigma, \mathtt{X}, \mathtt{U})$ makes the satisfiability problem undecidable.

In this paper we continue the study of PA, which are finite state automata with a finite number of pebbles. The pebbles are placed on/lifted from the input word in the stack discipline – first in last out – and are intended to mark positions in the input word. One pebble can only mark one position and the most recently placed pebble serves as the head of the automaton. The automaton moves from one state to another depending on the current label and the equality tests among data values in the positions currently marked by the pebbles, as well as, the equality tests among the positions of the pebbles.

Furthermore, as defined in [8], there are two types of PA, according to the position of the new pebble placed. In the first type, the ordinary PA, also called *strong* PA, the new pebbles are placed at the beginning of the string. In the second type, called *weak* PA, the new pebbles are placed at the position of the most recent pebble. Obviously, two-way weak PA is just as expressive as two-way ordinary PA. However, it is known that one-way nondeterministic weak PA are weaker than one-way ordinary PA, see [8, Theorem 4.5.].

In this paper we show that the emptiness problem for one-way weak 2-pebble automata is decidable, while the same problem for one-way weak 3-pebble automata is undecidable. We also introduce the so-called *top view* weak PA. Roughly speaking, top view weak PA are one-way weak PA where the equality test is performed only between the data values seen by the two most recently placed pebbles. Top view weak PA are quite robust: alternating, nondeterministic and deterministic top view weak PA have the same recognition power. To the best of our knowledge, this is the first model of computation for data language with such robustness. It is also shown that top view weak PA can be simulated by one-way alternating one-register RA. Therefore, their emptiness problem is decidable. Another interesting feature is top view weak PA can simulate all $\mathrm{LTL}_1^{\downarrow}(\Sigma, \mathtt{X}, \mathtt{U})$ languages.

This paper is organized as follows. In Section 2 we review the models of computations for data languages considered in this paper. Section 3 and Section 4 deals with the decidability and the complexity issues of weak PA, respectively. In Section 5 we introduce top view weak PA. Finally, we end our paper with a brief observation in Section 6.

2 Model of computations

In Subsection 2.1 we recall the definition of weak PA from [8]. We will use the following notation. We always denote by Σ a finite alphabet of *labels* and by $\mathfrak{D}$ an infinite set of *data values.* A *Σ-data word* $w = \binom{\sigma_1}{a_1}\binom{\sigma_2}{a_2}\cdots\binom{\sigma_n}{a_n}$ is a finite sequence over $\Sigma \times \mathfrak{D}$, where $\sigma_i \in \Sigma$ and $a_i \in \mathfrak{D}$. A *Σ-data language* is a set of Σ-data words.

We assume that neither of Σ and $\mathfrak{D}$ contain the left-end marker $\triangleleft$ or the right-end marker $\triangleright$. The input word to the automaton is of the form $\triangleleft w \triangleright$, where $\triangleleft$ and $\triangleright$ mark the left-end and the right-end of the input word. Finally, the symbols $\nu, \vartheta, \sigma, \ldots$, possibly indexed, denote labels in Σ and the symbols $a, b, c, d, \ldots$, possibly indexed, denote data values in $\mathfrak{D}$.

2.1 Pebble automata

Definition 1. *(See [8, Definition 2.3]) A* one-way alternating weak k-pebble automaton *(k-PA) over Σ is a system $\mathcal{A} = \langle \Sigma, Q, q_0, F, \mu, U \rangle$ whose components are defined as follows.*

- *Q, $q_0 \in Q$ and $F \subseteq Q$ are a finite set of* states*, the* initial state*, and the set of* final states*, respectively;*
- *$U \subseteq Q - F$ is the set of* universal *states; and*
- *$\mu \subseteq \mathcal{C} \times \mathcal{D}$ is the transition relation, where*
  - *$\mathcal{C}$ is a set whose elements are of the form (i, σ, V, q) where $1 \le i \le k$, $\sigma \in \Sigma$, $V \subseteq \{i+1, \ldots, k\}$ and $q \in Q$; and*
  - *$\mathcal{D}$ is a set whose elements are of the form $(q, \texttt{act})$, where $q \in Q$ and $\texttt{act}$ is either $\texttt{stay}$, $\texttt{right}$, $\texttt{place-pebble}$ or $\texttt{lift-pebble}$.*

  Elements of μ will be written as $(i, \sigma, V, q) \to (p, \texttt{act})$.

Given a word $w = \binom{\sigma_1}{a_1}\cdots\binom{\sigma_n}{a_n} \in (\Sigma \times \mathfrak{D})^*$, a *configuration of $\mathcal{A}$ on $\triangleleft w \triangleright$* is a triple $[i, q, \theta]$, where $i \in \{1, \ldots, k\}$, $q \in Q$, and $\theta : \{i, i+1, \ldots, k\} \to \{0, 1, \ldots, n, n+1\}$, where 0 and $n+1$ are positions of the end markers $\triangleleft$ and $\triangleright$, respectively. The function θ defines the position of the pebbles and is called the *pebble assignment.* The *initial* configuration is $\gamma_0 = [k, q_0, \theta_0]$, where $\theta_0(k) = 0$ is the *initial* pebble assignment. A configuration $[i, q, \theta]$ with $q \in F$ is called an *accepting* configuration.

A transition $(i, \sigma, V, p) \to \beta$ *applies to a configuration* $[j, q, \theta]$, if

(1) $i = j$ and $p = q$,
(2) $V = \{l > i : a_{\theta(l)} = a_{\theta(i)}\}$, and
(3) $\sigma_{\theta(i)} = \sigma$.

Note that in a configuration $[i, q, \theta]$, pebble i is in control, serving as the head pebble.

Next, we define the transition relation $\vdash_{\mathcal{A}}$ as follows: $[i, q, \theta] \vdash_{\mathcal{A}} [i', q', \theta']$, if there is a transition $\alpha \to (p, \texttt{act}) \in \mu$ that applies to $[i, q, \theta]$ such that $q' = p$, $\theta'(j) = \theta(j)$, for all $j > i$, and

- if $\texttt{act} = \texttt{stay}$, then $i' = i$ and $\theta'(i) = \theta(i)$;
- if $\texttt{act} = \texttt{right}$, then $i' = i$ and $\theta'(i) = \theta(i) + 1$;
- if $\texttt{act} = \texttt{lift-pebble}$, then $i' = i + 1$;
- if $\texttt{act} = \texttt{place-pebble}$, then $i' = i - 1$, $\theta'(i-1) = \theta'(i) = \theta(i)$.

As usual, we denote the reflexive transitive closure of $\vdash_{\mathcal{A}}$ by $\vdash^*_{\mathcal{A}}$. When the automaton $\mathcal{A}$ is clear from the context, we will omit the subscript $\mathcal{A}$.

Remark 1. Note the pebble numbering that differs from that in [8]. In the above definition we adopt the pebble numbering from [3] in which the pebbles placed on the input word are numbered from k to i and not from 1 to i as in [8]. The reason for this reverse numbering is that it allows us to view the computation between placing and lifting pebble i as a computation of an $(i-1)$-pebble automaton.

Furthermore, the automaton is no longer equipped with the ability to compare positional equality, in contrast with the ordinary PA introduced in [8]. Such ability no longer makes any difference because of the "weak" manner in which the new pebbles are placed.

The acceptance criteria is based on the notion of *leads to acceptance* below. For every configuration $\gamma = [i, q, \theta]$,

– if $q \in F$, then γ leads to acceptance;
– if $q \in U$, then γ leads to acceptance if and only if for all configurations γ' such that $\gamma \vdash \gamma'$, γ' leads to acceptance;
– if $q \notin F \cup U$, then γ leads to acceptance if and only if there is at least one configuration γ' such that $\gamma \vdash \gamma'$ and γ' leads to acceptance.

A Σ-data word $w \in (\Sigma \times \mathfrak{D})^*$ is accepted by $\mathcal{A}$, if γ_0 leads to acceptance. The language $L(\mathcal{A})$ consists of all data words accepted by $\mathcal{A}$.

The automaton $\mathcal{A}$ is *nondeterministic*, if the set $U = \emptyset$, and it is *deterministic*, if there is exactly one transition that applies for each configuration. It turns out that weak PA languages are quite robust.

Theorem 1. *For all $k \geq 1$, alternating, non-deterministic and deterministic weak k-PA have the same recognition power.*

The proof is quite standard and the details will appear in the journal version of this paper. In view of this, we will always assume that our weak k-PA is deterministic.

We end this subsection with an example of language accepted by weak 2-PA. This example will be useful in the subsequent section.

Example 1. Consider a Σ-data language $L_\sim$ defined as follows. A Σ-data word $w = \binom{\sigma_1}{a_1} \cdots \binom{\sigma_n}{a_n} \in L_\sim$ if and only if for all $i, j = 1, \ldots, n$, if $a_i = a_j$, then $\sigma_i = \sigma_j$. That is, $w \in L_\sim$ if and only if whenever two positions in w carry the same data value, their labels are the same.

The language $L_\sim$ is accepted by weak 2-PA which works in the following manner. Pebbles 2 iterates through all possible positions in w. At each iteration, the automaton stores the label seen by pebble 2 in the state and places pebble 1. Then, pebble 1 scans through all the positions to the right of pebble 2, checking whether there is a position with the same data value of pebble 2. If there is such a position, then the labels seen by pebble 1 must be the same as the label seen by pebble 2, which has been previously stored in the state.

3 Decidability and undecidability of weak PA

In this section we will discuss the decidability issue of weak PA. We show that the emptiness problem for weak 3-PA is undecidable, while the same problem for weak 2-PA is decidable. The proof of the decidability of the emptiness problem for weak 2-PA will be the basis of the proof of the decidability of the same problem for top view weak PA.

Theorem 2. *The emptiness problem for weak* 3*-PA is undecidable.*

The proof of Theorem 2 is similar to the proof of the undecidability of the emptiness problem for weak 5-PA in [8]. The main technical step in the proof is to show that the following Σ-data language

$$L_{\mathrm{ord}} = \left\{ \binom{\sigma}{a_1} \cdots \binom{\sigma}{a_n} \binom{\$}{d} \binom{\sigma}{a_1} \cdots \binom{\sigma}{a_n} : a_1, \ldots, a_n \text{ are pairwise different} \right\}$$

is accepted by weak 5-PA, where $\Sigma = \{\sigma, \$\}$. We observe that weak 3-PA is sufficient. From this step, the undecidability can be easily obtained via a reduction from the Post Correspondence Problem (PCP).

Now we are going to show that the emptiness problem for weak 2-PA is decidable. The proof is by simulating weak 2-PA by one-way alternating one register automata (1-RA). See [4] for the definition of alternating RA.

In fact, the simulation can be easily generalized to arbitrary number of pebbles. That is, weak k-PA can be simulated by one-way alternating $(k-1)$-RA. (Here $(k-1)$-RA stands for $(k-1)$-register automata.) This result settles a question left open in [8]: Can weak PA be simulated by alternating RA?

Theorem 3. *For every weak k-PA $\mathcal{A}$, there exists a one-way alternating $(k-1)$-RA $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$. Moreover, the construction of $\mathcal{A}'$ from $\mathcal{A}$ is effective.*

Now, by Theorem 3, we immediately obtain the decidability of the emptiness problem for weak 2-PA because the same problem for one-way alternating 1-RA is decidable [4, Theorem 4.4].

Corollary 1. *The emptiness problem for weak* 2*-PA is decidable.*

We devote the rest of this section to the proof of Theorem 3 for $k = 2$. Its generalization to arbitrary $k \geq 3$ is pretty straightforward.

Recall that we always assume that weak PA is deterministic. Let $\mathcal{A} = \langle \Sigma, Q, q_0, \mu, F \rangle$ be a weak 2-PA. We normalize the behavior of $\mathcal{A}$ as follows.

– Pebble 1 is lifted only after it reads the right-end marker $\triangleright$.
– Only pebble 2 can enter a final state and it does so after it reads the right-end marker $\triangleright$.
– Immediately after pebble 2 moves right, pebble 1 is placed.
– Immediately after pebble 1 is lifted, pebble 2 moves right.

Such normalization can be obtained in a pretty standard manner. The details will appear in the journal version of this paper.

On input word $w = \binom{\sigma_1}{d_1} \cdots \binom{\sigma_n}{d_n}$, the run of $\mathcal{A}$ on $\triangleleft w \triangleright$ can be depicted as a tree shown in Figure 1.

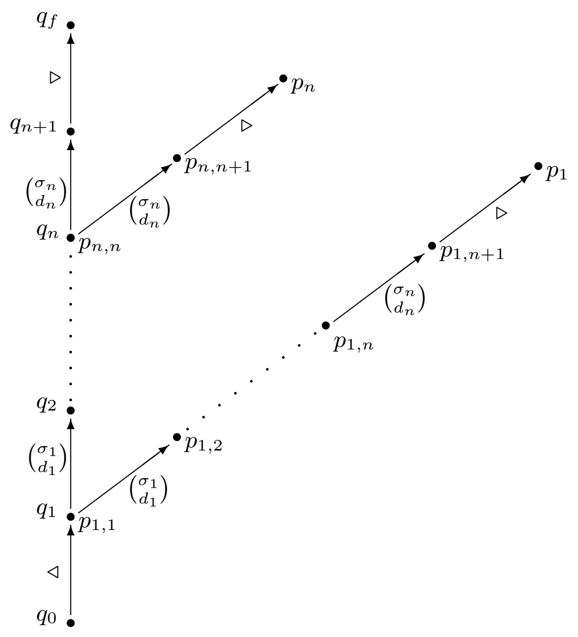

Fig. 1. The tree representation of a run of $\mathcal{A}$ on the data word $w = \binom{\sigma_1}{d_1} \cdots \binom{\sigma_n}{d_n}$.

The meaning of the tree is as follows.

– $q_0, q_1, \ldots, q_n, q_{n+1}$ are the states of $\mathcal{A}$ when pebble 2 is the head pebble reading the positions $0, 1, \ldots, n, n+1$, respectively, that is, the symbols $\triangleleft, \binom{\sigma_1}{d_1}, \ldots, \binom{\sigma_n}{d_n}, \triangleright$, respectively.
– q_f is the state of $\mathcal{A}$ after pebble 2 reads the symbol $\triangleright$.
– For $1 \leq i \leq j \leq n$, $p_{i,j}$ is the state of $\mathcal{A}$ when pebble 1 is the head pebble reading the position j, while pebble 2 is above the position i.
– For $1 \leq i \leq n$, the state p_i is the state of $\mathcal{A}$ immediately after pebble 1 is lifted and pebble 2 is above the position i.

It must be noted that there is a transition $(2, \sigma_i, \emptyset, p_i) \rightarrow (q_{i+1}, \texttt{right})$ applied by $\mathcal{A}$ that is not depicted in the figure.

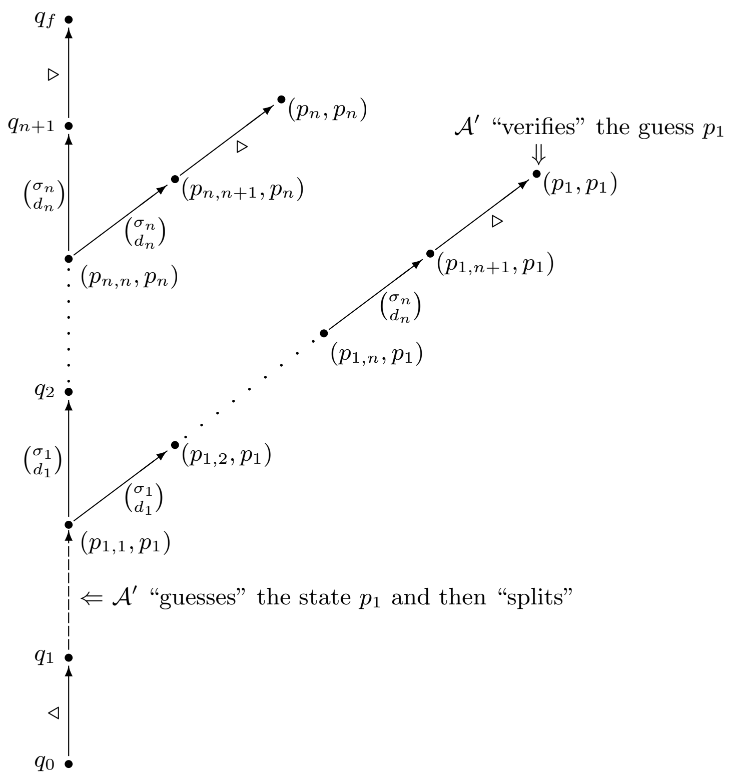

Fig. 2. The corresponding run of $\mathcal{A}'$ the data word $w = \binom{\sigma_1}{d_1} \cdots \binom{\sigma_n}{d_n}$ to the one in Figure 1.

Now the simulation of $\mathcal{A}$ by a one-way alternating 1-RA $\mathcal{A}'$ becomes straightforward if we view the tree in Figure 1 as a tree depicting the computation of $\mathcal{A}'$ on the same word w. Figure 2 shows the corresponding run of $\mathcal{A}'$ on the same word. Roughly, the automaton $\mathcal{A}'$ is defined as follows.

– The states of $\mathcal{A}'$ are elements of $Q \cup (Q \times Q)$;
– the initial state is q_0; and
– the set of final states is $F \cup \{(p, p) : p \in Q\}$.

For each placement of pebble 1 on position i, the automaton performs the following "Guess–Split–Verify" procedure which consists of the following steps.

1. From the state q_i, $\mathcal{A}'$ "guesses" (by disjunctive branching) the state in which pebble 1 is eventually lifted, i.e. the state p_i. It stores p_i in its internal state

and simulates the transition $(2, \sigma_i, \emptyset, \emptyset, q_i) \to (p_{i,i}, \texttt{place-pebble}) \in \mu$ to enter into the state $(p_{i,i}, p_i)$.

2. $\mathcal{A}'$ "splits" its computation (by conjunctive branching) into two branches.
   - In one branch, assuming that the guess p_i is correct, $\mathcal{A}'$ moves right and enters into the state q_{i+1}, simulating the transition $(2, \emptyset, p_i) \to (q_{i+1}, \texttt{right})$. After this, it recursively performs the Guess–Split–Verify procedure for the next placement of pebble 1 on position $(i+1)$.
   - In the other branch $\mathcal{A}'$ stores the data value d_i in its register and simulates the run of pebble 1 on $\binom{\sigma_i}{d_i} \cdots \binom{\sigma_n}{d_n}$, starting from the state $p_{i,i}$, to "verify" that the guess p_i is correct.
     During the simulation, to remember the guess p_i, the states of $\mathcal{A}'$ are $(p_{i,i}, p_i), \ldots, (p_{i,n+1}, p_i)$. $\mathcal{A}'$ accepts when the simulation ends in the state (p_i, p_i), that is, when the guess p_i is "correct."

4 Complexity of weak 2-PA

In this subsection we are going to study the time complexity of three specific problems related to weak 2-PA.

Emptiness problem. The emptiness problem for weak 2-PA. That is, given a weak 2-PA $\mathcal{A}$, is $L(\mathcal{A}) = \emptyset$?

Labelling problem. Given a deterministic weak 2-PA $\mathcal{A}$ over the labels Σ and a sequence of data values $d_1 \cdots d_n \in \mathfrak{D}^n$, is there a sequence of labels $\sigma_1 \cdots \sigma_n \in \Sigma^n$ such that $\binom{\sigma_1}{d_1} \cdots \binom{\sigma_n}{d_n} \in L(\mathcal{A})$?

Data value membership problem. Given a deterministic weak 2-PA $\mathcal{A}$ over the labels Σ and a sequence of finite labels $\sigma_1 \cdots \sigma_n \in \Sigma^n$, is there a sequence of data values $d_1 \cdots d_n \in \mathfrak{D}^n$ such that $\binom{\sigma_1}{d_1} \cdots \binom{\sigma_n}{d_n} \in L(\mathcal{A})$?

The emptiness problem, as we have seen in the previous section, is decidable. The labelling and data value membership problem are in NP. To solve the labelling problem, one simply guesses a sequence $\sigma_1 \cdots \sigma_n \in \Sigma^n$ and runs $\mathcal{A}$ to check whether $\binom{\sigma_1}{d_1} \cdots \binom{\sigma_n}{d_n} \in L(\mathcal{A})$. Similarly, to solve the data value membership problem, one can guess a sequence of data values $d_1 \cdots d_n$ and run $\mathcal{A}$ to check whether $\binom{\sigma_1}{d_1} \cdots \binom{\sigma_n}{d_n} \in L(\mathcal{A})$.

We will show that the emptiness problem is not primitive recursive, while both the labelling and data value membership problems are NP-complete.

Theorem 4. *The emptiness problem for weak 2-PA is not primitive recursive.*

The proof of theorem 4 is by simulation of incrementing counter automata and follows closely the proof of similar lower bound for one-way alternating 1-RA [4, Theorem 2.9]. In short, the main technical step is to show that the following Σ-data language L_{inc} which consists of the data words of the form: $\binom{\alpha}{a_1} \cdots \binom{\alpha}{a_m} \binom{\beta}{b_1} \cdots \binom{\beta}{b_n}$; where

- $\Sigma = \{\alpha, \beta\}$;
- the data values $a_1, \ldots, a_m$ are pairwise different;

- the data values $b_1, \ldots, b_n$ are pairwise different;
- each a_i appears among $b_1, \ldots, b_n$;

is accepted by weak 2-PA. The intuition of this language is to represent the inequality $m \leq n$, which is important in the simulation of incrementing counter automata, of which the emptiness problem is known to be decidable [9], but not primitive recursice [7].

Now we are going to show the NP-hardness of the labelling problem. It is by a reduction from graph 3-colorability problem which states as follows. Given an undirected graph $G = (V, E)$, is G is 3-colorable?

Let $V = \{1, \ldots, n\}$ and $E = \{(i_1, j_1), \ldots, (i_m, j_m)\}$. Assuming that $\mathfrak{D}$ contains the natural numbers, we take $i_1 j_1 \cdots i_m j_m$ as the sequence of data values. Then, we construct a weak 2-PA $\mathcal{A}$ over the alphabet $\Sigma = \{\vartheta_R, \vartheta_G, \vartheta_B\}$ that accepts data words of even length in which the following hold.

- For all odd position x, the label on position x is different from the label on position $x + 1$.
- For every two positions x and y, if they have the same data value, then they have the same label. (See Example 1.)

Thus, the graph G is 3-colorable if and only if there exists $\sigma_1 \cdots \sigma_{2m} \in \{\vartheta_R, \vartheta_G, \vartheta_B\}^*$ such that $\binom{\sigma_1}{i_1}\binom{\sigma_2}{j_1} \cdots \binom{\sigma_{2m-1}}{i_m}\binom{\sigma_{2m}}{j_m} \in L(\mathcal{A})$, and the NP-hardness, hence the NP-completeness, of the labelling problem follows.

The NP-hardness of data value membership problem can established in a similar spirit. The reduction is from the following variant of graph 3-colorability, called 3-colorability with constraint. Given a graph $G = (V, E)$ and three integers n_r, n_g, n_b, is the graph G 3-colorable with the colors R, G and B such that the numbers of vertices colored with R, G and B are n_r, n_g and n_b, respectively?

The polynomial time reduction to data value membership problem is as follows. Let $V = \{1, \ldots, n\}$ and $E = \{(i_1, j_1), \ldots, (i_m, j_m)\}$.

We define $\Sigma = \{\vartheta_R, \vartheta_G, \vartheta_B, \nu_1, \ldots, \nu_n\}$ and take

$$\nu_{i_1}\nu_{j_1} \cdots \nu_{i_m}\nu_{j_m} \underbrace{\vartheta_R \cdots \vartheta_R}_{n_r \text{ times}} \underbrace{\vartheta_G \cdots \vartheta_G}_{n_g \text{ times}} \underbrace{\vartheta_B \cdots \vartheta_B}_{n_b \text{ times}}$$

as the sequence of finite labels. Then, we construct a weak 2-PA over Σ that accepts data words of the form

$$\binom{\nu_{i_1}}{c_1}\binom{\nu_{j_1}}{d_1} \cdots \binom{\nu_{i_m}}{c_m}\binom{\nu_{j_m}}{d_m}\binom{\vartheta_R}{a_1} \cdots \binom{\vartheta_R}{a_{n_r}}\binom{\vartheta_G}{a'_1} \cdots \binom{\vartheta_G}{a'_{n_g}}\binom{\vartheta_B}{a''_1} \cdots \binom{\vartheta_B}{a''_{n_b}},$$

where

- $\nu_{i_1}, \nu_{j_1}, \ldots, \nu_{i_m}, \nu_{j_m} \in \{\nu_1, \ldots, \nu_n\}$;
- in the sub-word $\binom{\nu_{i_1}}{c_1}\binom{\nu_{j_1}}{d_1} \cdots \binom{\nu_{i_m}}{c_m}\binom{\nu_{j_m}}{d_m}$, every two positions with the same labels have the same data value (see Example 1);
- the data values $a_1, \ldots, a_{n_r}, a'_1, \ldots, a'_{n_g}, a''_1, \ldots, a''_{n_b}$ are pairwise different;

– For each $i = 1, \ldots, m$, the data values c_i, d_i appear among $a_1, \ldots, a_{n_r}$, $a'_1, \ldots, a'_{n_g}$, $a''_1, \ldots, a''_{n_b}$ such that the following holds:
  - if c_i appears among $a_1, \ldots, a_{n_r}$, then d_i appears among $a'_1, \ldots, a'_{n_g}$ or $a''_1, \ldots, a''_{n_b}$;
  - if c_i appears among $a'_1, \ldots, a'_{n_g}$, then d_i appears either among $a_1, \ldots, a_{n_r}$ or $a''_1, \ldots, a''_{n_b}$; and
  - if c_i appears among $a''_1, \ldots, a''_{n_b}$, then d_i appears among $a_1, \ldots, a_{n_r}$ or $a'_1, \ldots, a'_{n_g}$.

Note that we can store the integers r, g, b and m in the internal states of $\mathcal{A}$, thus, enable $\mathcal{A}$ to "count" up to n_r, n_g, n_b and m. We have each state for the numbers $1, \ldots, n_r$, $1, \ldots, n_g$, $1, \ldots, n_b$ and $1, \ldots, m$. The number of states in $\mathcal{A}$ is still polynomial on n.

Now the graph G is 3-colorable with the constraints n_r, n_g, n_b if and only if there exits $c_1 d_1 \cdots c_m d_m a_1 \cdots a_{n_r} a'_1 \cdots a'_{n_g} a''_1 \cdots a''_{n_b}$ such that

$$\binom{\nu_{i_1}}{c_1}\binom{\nu_{j_1}}{d_1}\cdots\binom{\nu_{i_m}}{c_m}\binom{\nu_{j_m}}{d_m}\binom{\vartheta_R}{a_1}\cdots\binom{\vartheta_R}{a_{n_r}}\binom{\vartheta_G}{a'_1}\cdots\binom{\vartheta_G}{a'_{n_g}}\binom{\vartheta_B}{a''_1}\cdots\binom{\vartheta_B}{a''_{n_b}}$$

is accepted by $\mathcal{A}$. The NP-hardness, hence the NP-completeness, of the data value membership problem then follows.

5 Top view weak k-PA

In this section we are going to define *top view* weak PA. Roughly speaking, top view weak PA are weak PA where the equality test is performed only between the data values seen by the last and the second last placed pebbles. That is, if pebble i is the head pebble, then it can only compare the data value it reads with the data value read by pebble $(i+1)$. It is not allowed to compare its data value with those read by pebble $(i+2), (i+3), \ldots, k$.

Formally, top view weak k-PA is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \mu, F\rangle$ where Q, q_0, F are as usual and μ consists of transitions of the form: $(i, \sigma, V, q) \to (q', \texttt{act})$, where V is either $\emptyset$ or $\{i+1\}$.

The criteria for the application of transitions of top view weak k-PA is defined by setting

$$V = \begin{cases} \emptyset, & \text{if } a_{\theta(i+1)} \neq a_{\theta(i)} \\ \{i+1\}, & \text{if } a_{\theta(i+1)} = a_{\theta(i)} \end{cases}$$

in the definition of transition relation in Subsection 2.1. Note that top view weak 2-PA and weak 2-PA are the same.

Remark 2. We can also define the alternating version of top view weak k-PA. However, just like in the case of weak k-PA, alternating, nondeterministic and deterministic top view weak k-PA have the same recognition power. Furthermore, by using the same proof presented in Section 4, it is straightforward to show that the emptiness problem, the labelling problem, and the data value membership problem have the same complexity lower bound for top view weak k-PA, for each $k = 2, 3, \ldots$.

The following theorem is a stronger version of Theorem 3.

Theorem 5. *For every top view weak k-PA $\mathcal{A}$, there is a one-way alternating 1-RA $\mathcal{A}'$ such that $L(\mathcal{A}') = L(\mathcal{A})$. Moreover, the construction of $\mathcal{A}'$ is effective.*

Proof. The proof is a straightforward generalization of the proof of Theorem 3. Each placement of a pebble is simulated by "Guess–Split–Verify" procedure. Since each pebble i can only compare its data value with the one seen by pebble $(i+1)$, $\mathcal{A}'$ does not need to store the data values seen by pebbles $(i+2), \dots, k$. It only needs to store the data value seen by pebble $(i+1)$, thus, one register is sufficient for the simulation.

Following Theorem 5, we immediately obtain the decidability of the emptiness problem for top view weak k-PA.

Corollary 2. *The emptiness problem for top view weak k-PA is decidable.*

Since the emptiness problem for ordinary 2-PA (See [6, Theorem 4]) and for weak 3-PA is already undecidable, it seems that top view weak PA is a tight boundary of a subclass of PA languages for which the emptiness problem is decidable.

Remark 3. In [11] it is shown that for every sentence $\psi \in \mathrm{LTL}_1^{\downarrow}(\Sigma, \mathtt{X}, \mathtt{U})$, there exists a weak k-PA $\mathcal{A}_\psi$, where $k = \mathsf{fqr}(\psi) + 1$, such that $L(\mathcal{A}_\psi) = L(\psi)$. We remark that the proof actually shows that the automaton $\mathcal{A}_\psi$ is top view weak k-PA. Thus, it shows that the class of top view weak k-PA languages contains the languages definable by $\mathrm{LTL}_1^{\downarrow}(\Sigma, \mathtt{X}, \mathtt{U})$.

6 Concluding remark

We end this paper with a quick observation on top view weak PA. We note that the finiteness of the number of pebbles for top view weak PA is not necessary. In fact, we can just define top view weak PA with unbounded number of pebbles, which we call top view weak unbounded PA.

We elaborate on it in the following paragraphs. Let $\mathcal{A} = \langle \Sigma, Q, q_0, \mu, F \rangle$ be top view weak unbounded PA. The pebbles are numbered with the numbers $1, 2, 3, \ldots$. The automaton $\mathcal{A}$ starts the computation with only pebble 1 on the input word. The transitions are of the form: $(\sigma, \chi, q) \to (p, \mathtt{act})$, where $\chi \in \{0, 1\}$ and $\sigma, q, p, \mathtt{act}$ are as in the ordinary weak PA.

Let $w = \binom{\sigma_1}{a_1} \cdots \binom{\sigma_n}{a_n}$ be an input word. A *configuration of $\mathcal{A}$ on* $\triangleleft w \triangleright$ is a triple $[i, q, \theta]$, where $i \in \mathbb{N}$, $q \in Q$, and $\theta : \mathbb{N} \to \{0, 1, \ldots, n, n+1\}$. The *initial* configuration is $[1, q_0, \theta_0]$, where $\theta_0(1) = 0$. The accepting configurations are defined similarly as in ordinary weak PA.

A transition $(\sigma, \chi, p) \to \beta$ *applies to a configuration* $[i, q, \theta]$, if

(1) $p = q$, and $\sigma_{\theta(i)} = \sigma$,
(2) $\chi = 1$ if $a_{\theta(i-1)} = a_{\theta(i)}$, and $\chi = 0$ if $a_{\theta(i-1)} \neq a_{\theta(i)}$,

The transition relation $\vdash$ and the acceptance criteria can be defined in a similar manner as in Section 2.1.

It is straightforward to show that 1-way deterministic 1-RA can be simulated by top view weak unbounded PA. Each time the register automaton change the content of the register, the top view weak unbounded PA places a new pebble.

Furthermore, top view weak unbounded PA can be simulated by 1-way alternating 1-RA. Each time a pebble is placed, the register automaton performs "Guess–Split–Verify" procedure described in Section 3. Thus, the emptiness problem for top view unbounded weak PA is still decidable.

Acknowledgment. The author would like to thank Michael Kaminski for his invaluable directions and guidance related to this paper and for pointing out the notion of unbounded pebble automata.

References

1. H. Björklund and T. Schwentick. On Notions of Regularity for Data Languages. In *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory (FCT 2007)*, Springer, pp. 88–99, *LNCS* 4639.
2. M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin and C. David. Two-Variable Logic on Words with Data. In *Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS 2006)*, pp. 7–16.
3. M. Bojańczyk, M. Samuelides, T. Schwentick and L. Segoufin. Expressive Power of Pebble Automata. In *the Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, pp. 157–168, *LNCS* 4051.
4. S. Demri and R. Lazic. LTL with the Freeze Quantifier and Register Automata. In *Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS 2006)*, pp. 17–26.
5. M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science* **134** (1994) 329–363.
6. M. Kaminski and T. Tan. A Note on Two-Pebble Automata over Infinite Alphabets. Technical Report CS-2009-02, Department of Computer Science, Technion – Israel Institute of Technology, 2009
7. R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science* **297**:1-3 (2003) 337–354.
8. F. Neven, T. Schwentick and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic* **5**:3 (2004) 403–435.
9. Ph. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters* **83**:5 (2002) 251–261.
10. L. Segoufin. Automata and Logics for Words and Trees over an Infinite Alphabet. In *Proceedings of the 20th International Workshop/15th Annual Conference of the EACSL Computer Science Logic (CSL 2006)*, pp. 41–57, *LNCS* 4207.
11. T. Tan. Graph Reachability and Pebble Automata over Infinite Alphabets. To appear in the *Proceedings of the 24th IEEE Symposium on Logic in Computer Science (LICS 2009).*