On Pebble Automata for Data Languages with Decidable Emptiness Problem

Tony Tan
Department of Computer Science
Technion – Israel Institute of Technology
Haifa 32000, Israel
Email: tantony@cs.technion.ac.il

Abstract

In this paper we determine a subclass of pebble automata (PA) for data languages for which the emptiness problem is decidable. Namely, we 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 decidable. We also show that it is robust: alternating, nondeterministic and deterministic top view weak PA have the same recognition power. Moreover, this model is strong enough to accept all data languages expressible in Linear Temporal Logic with the future-time operators, augmented with one register freeze quantifier.

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, 6, 9, 11] and this paper aims to contribute to the progress.
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 [11]. The study of data languages, or languages over infinite alphabets, starts with the introduction of finite-memory automata (FMA) in [6], which are also known as register automata (RA). The study of RA was continued and extended in [9], 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,\(^1\) 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 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, 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 tree-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 \(\Sigma\), denoted LTL\(^\downarrow\)\(_n\)(\(\Sigma, X, U\)), see [4]. It is shown that one way alternating \(n\) register automata accept all LTL\(^\downarrow\)\(_n\)(\(\Sigma, X, U\)) languages and the emptiness problem for one way alternating one register automata is decidable. Hence, the satisfiability problem for LTL\(^\downarrow\)\(_1\)(\(\Sigma, X, U\)) is decidable as well. Adding one more register or past time operators to LTL\(^\downarrow\)\(_1\)(\(\Sigma, X, U\)) 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 last pebble placed serves as the head of the automaton. The automaton moves from one state to another depending on the current label and the

\(^1\)Though it is unknown whether they are closed under Kleene star.
equality tests among data values in the positions and the positions themselves currently marked by the pebbles.

Furthermore, as defined in [9], 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 [9, Theorem 4.5].

The main contributions of this paper is the introduction of a weaker version of one way weak PA, which we call 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. Top view weak PA are quite robust: for a fixed number of pebbles, 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. For practical purposes, the most interesting feature of top view weak PA is, perhaps, their containment of all LTL$_1^\downarrow$($\Sigma$, X, U) languages. In fact, the number of pebbles of top view weak PA needed to simulate an LTL$_1^\downarrow$($\Sigma$, X, U) sentence linearly depends on its so called freeze quantifier rank, an analog of the standard quantifier rank of a first-order sentence.

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.

## 2 Models of computations

In Subsections 2.1 and 2.2 we recall the definition of weak PA from [9], and review the strict hierarchy of weak PA languages established in [12]. In Subsection 2.3 we recall the temporal logical framework for data languages.

We will use the following notation. We always denote by $\Sigma$ a finite alphabet of labels and by $\mathcal{D}$ an infinite set of data values. A $\Sigma$-data word $w = (\sigma_1, \mathcal{a}_1)(\sigma_2, \mathcal{a}_2) \cdots (\sigma_n, \mathcal{a}_n)$ is a finite sequence over $\Sigma \times \mathcal{D}$, where $\sigma_i \in \Sigma$ and
A \( \Sigma \)-data language is a set of \( \Sigma \)-data words. The idea is that the alphabet \( \Sigma \) is accessed directly, while data values can only be tested for equality.

We assume that neither of \( \Sigma \) and \( D \) contain the left-end marker \(<\) or the right-end marker \(>\). The input word to the automaton is of the form \(<w>\), where \(<\) and \(>\) mark the left-end and the right-end of the input word.

We will also use the following notations. For \( w = (\sigma_1 a_1) \cdots (\sigma_n a_n) \),

\[
\begin{align*}
\text{Proj}_{\Sigma}(w) &= \sigma_1 \cdots \sigma_n, \\
\text{Proj}_{D}(w) &= a_1 \cdots a_n, \\
\text{Cont}_{\Sigma}(w) &= \{\sigma_1, \ldots, \sigma_n\}, \\
\text{Cont}_{D}(w) &= \{a_1, \ldots, a_n\}.
\end{align*}
\]

Finally, the symbols \( \nu, \vartheta, \sigma, \ldots \), possibly indexed, denote labels in \( \Sigma \) and the symbols \( a, b, c, d, \ldots \), possibly indexed, denote data values in \( D \).

### 2.1 Pebble automata

**Definition 1** (See [9, Definition 2.3]) A one-way alternating weak \( k \)-pebble automaton or, in short, \( k \)-PA, over \( \Sigma \) is a system \( A = (\Sigma, Q, q_0, F, \mu, U) \) 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 \((i, \sigma, V, q)\) where \( 1 \leq i \leq k \), \( \sigma \in \Sigma \), \( V \subseteq \{i + 1, \ldots, k\} \) and \( q \in Q \); and
  
  - \( \mathcal{D} \) is a set whose elements are of the form \((q, \text{act})\), where \( q \in Q \) and \( \text{act} \in \{\text{stay}, \text{right}, \text{place-pebble}, \text{lift-pebble}\} \).

Elements of \( \mu \) will be written as \((i, \sigma, V, q) \rightarrow (p, \text{act})\).

**Remark 2** Note that the pebble numbering that differs from that in [9]. 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 [9]. 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 [9]. Such ability no longer makes any difference because the new pebbles are placed in the “weak” manner.

Given a word \( w = (a_1) \cdots (a_n) \in (\Sigma \times \mathcal{D})^* \), a configuration of \( A \) on \( w \) 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 \( \langle \) and \( \rangle \), respectively. The function \( \theta \) 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 \((i, \sigma, V, p) \rightarrow \beta\) 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. \( \sigma_{\theta(i)} = \sigma \).

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

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

As usual, we denote the reflexive transitive closure of \( \vdash_A \) by \( \vdash^*_A \). When the automaton \( A \) is clear from the context, we shall omit the subscript \( A \).

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 \( \gamma \) leads to acceptance;
• if \( q \in U \), then \( \gamma \) leads to acceptance if and only if for all configurations \( \gamma' \) such that \( \gamma \vdash \gamma' \), \( \gamma' \) leads to acceptance;

• if \( q \notin F \cup U \), then \( \gamma \) leads to acceptance if and only if there is at least one configuration \( \gamma' \) such that \( \gamma \vdash \gamma' \), and \( \gamma' \) leads to acceptance.

A \( \Sigma \)-data word \( w \in (\Sigma \times \mathcal{D})^* \) is accepted by \( A \), if \( \gamma_0 \) leads to acceptance. The language \( L(A) \) consists of all data words accepted by \( A \).

The automaton \( A \) 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 3** For all \( k \geq 1 \), alternating, non-deterministic and deterministic weak \( k \)-PA have the same recognition power.

The proof is quite standard. For the details of the proof, we refer the reader to Appendix D.

Next, we define the hierarchy of languages accepted by PA. For \( k \geq 1 \), we define the following classes of languages.

\[
\begin{align*}
wPA_k &= \{ L : L \text{ is accepted by a weak } k\text{-PA} \}; \\
wPA &= \bigcup_{k \geq 1} wPA_k
\end{align*}
\]

**Example 4** Consider a \( \Sigma \)-data language \( L_\sim \) defined as follows. A \( \Sigma \)-data word \( w = (\sigma_1 a_1) \cdots (\sigma_n a_n) \in L_\sim \) 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, then they are labeled with the same label. The language \( L_\sim \) is accepted by weak 2-PA which works in the following manner. Both pebbles iterate through all possible positions in \( w \). At each cycle, if both pebbles see the same data values, then the automaton checks whether their labels are the same.

### 2.2 Strict hierarchy of weak PA languages

In this section we review an example of data language introduced in [12]. It will be useful in establishing our definability results for \( \text{LTL}_1(\Sigma, x, U) \) languages.
Let $\Sigma = \{\sigma\}$ be a singleton alphabet. For an integer $m \geq 1$, the language $R^+_m$ consists of $\Sigma$-data words of the form
\[
\left(\sigma\right)_{a_0}\left(\sigma\right)_{a_1} \cdots \cdots \left(\sigma\right)_{a_{m-2}}\left(\sigma\right)_{a_{m-1}}\left(\sigma\right)_{a_m}
\]
where
- for each $i = 0, 1, \ldots, m - 1$, $a_i \neq a_{i+1}$;
- for each $i = 1, \ldots, m - 1$, $a_i \notin \text{Cont}_D(w_i)$.

The language $R^+$ is defined as
\[
R^+ = \bigcup_{m=1,2,\ldots} R^+_m.
\]

**Theorem 5** (See [12, Lemma 18].) For each $k = 1, 2, \ldots$,
1. $R_k \in wPA_k$ and $R_{k+1} \notin wPA_k$;
2. $wPA_k \subset \subset wPA_{k+1}$.

### 2.3 Linear temporal logic with one register freeze quantifier

In this section we recall the definition of Linear Temporal Logic (LTL) with one register freeze quantifier [4]. We consider only one-way temporal operators “next” $X$ and “until” $U$, and do not consider their past time counterparts.

Let $\Sigma$ be a finite alphabet of labels. Roughly, the logic $\text{LTL}_1^1(\Sigma, X, U)$ is standard LTL augmented with a register to store a data value. Formally, the formulas are defined as follows.
- Both $\text{True}$ and $\text{False}$ belong to $\text{LTL}_1^1(\Sigma, X, U)$.
- The empty formula $\epsilon$ belongs to $\text{LTL}_1^1(\Sigma, X, U)$.
- For each $\sigma \in \Sigma$, $\sigma$ is in $\text{LTL}_1^1(\Sigma, X, U)$.
- If $\varphi, \psi$ are in $\text{LTL}_1^1(\Sigma, X, U)$, then so are $\neg \varphi$, $\varphi \lor \psi$ and $\varphi \land \psi$.
- $\uparrow$ is in $\text{LTL}_1^1(\Sigma, X, U)$. 

• If $\varphi$ is in $\text{LTL}_1(\Sigma, X, U)$, then so is $X\varphi$.

• If $\varphi$ is in $\text{LTL}_1(\Sigma, X, U)$, then so is $\downarrow \varphi$.

• If $\varphi, \psi$ are in $\text{LTL}_1(\Sigma, X, U)$, then so is $\varphi \lor \psi$.

Intuitively, the predicate $\uparrow$ is intended to mean that the current data value is the same as the data value in the register, while $\downarrow \varphi$ is intended to mean that the formula $\varphi$ holds when the register contains the current data value. This will be made precise in the definition of the semantics of $\text{LTL}_1(\Sigma, X, U)$ below.

An occurrence of $\uparrow$ within the scope of some freeze quantification $\downarrow$ is bounded by it; otherwise, it is free. A sentence is a formula with no free occurrence of $\uparrow$.

Next we define the freeze quantifier rank of a sentence $\varphi$, denoted by $\text{fqr}(\varphi)$.

• For each $\sigma \in \Sigma$, $\text{fqr}(\sigma) = 0$.

• $\text{fqr}(\text{True}) = \text{fqr}(\text{False}) = \text{fqr}(\uparrow) = 0$.

• $\text{fqr}(X\varphi) = \text{fqr}(\neg \varphi) = \text{fqr}(\varphi)$, for every $\varphi$ in $\text{LTL}_1(\Sigma, X, U)$.

• $\text{fqr}(\varphi \lor \psi) = \text{fqr}(\varphi \land \psi) = \text{fqr}(\varphi \lor \psi) = \max(\text{fqr}(\varphi), \text{fqr}(\psi))$, for every $\varphi$ and $\psi$ in $\text{LTL}_1(\Sigma, X, U)$.

• $\text{fqr}(\downarrow \varphi) = \text{fqr}(\varphi) + 1$, for every $\varphi$ in $\text{LTL}_1(\Sigma, X, U)$.

Finally, we define the semantics of $\text{LTL}_1(\Sigma, X, U)$. Let $w = (\sigma_1^a) \cdots (\sigma_n^a)$ be a $\Sigma$-data word. For a position $i = 1, \ldots, n$, a data value $a$ and a formula $\varphi$ in $\text{LTL}_1(\Sigma, X, U)$, $w, i \models_a \varphi$ means that $\varphi$ is satisfied by $w$ at position $i$ when the content of the register is $a$. As usual, $w, i \not\models_a \varphi$ means $\varphi$ is not satisfied by $w$ at position $i$ when the content of the register is $a$. The satisfaction relation is defined inductively as follows.

• $w, i \models_a \epsilon$ for all $i = 1, 2, \ldots, n$ and $a \in \mathcal{D}$.

• $w, i \models_a \text{True}$ and $w, i \not\models_a \text{False}$, for all $i = 1, 2, 3, \ldots$ and $a \in \mathcal{D}$.

• $w, i \models_a \sigma$ if and only if $\sigma_i = \sigma$.

• $w, i \models_a \varphi \lor \psi$ if and only if $w, i \models_a \varphi$ or $w, i \models_a \psi$. 

8
\begin{itemize}
  \item $w, i \models_a \varphi \land \psi$ if and only if $w, i \models_a \varphi$ and $w, i \models_a \psi$.
  \item $w, i \models_a \neg\varphi$ if and only if $w, i \not\models_a \varphi$.
  \item $w, i \models_a \varphi \land \psi$ if and only if $1 \leq i < |w|$ and $w, i + 1 \models_a \varphi$.
  \item $w, i \models_a \varphi \lor \psi$ if and only if there exists $j \geq i$ such that
    \begin{itemize}
      \item $w, j \models_a \psi$ and
      \item $w, j' \models_a \varphi$, for all $j' = i, \ldots, j - 1$
    \end{itemize}
  \item $w, i \models_a \downarrow \varphi$ if and only if $w, i \models_a \varphi$
  \item $w, i \models_a \uparrow$ if and only if $a = a_i$.
\end{itemize}

For a sentence $\varphi$ in $\text{LTL}_1^\Sigma(X,U)$, we define the $\Sigma$-data language $L(\varphi)$ by

$$L(\varphi) = \{ w \mid w, 1 \models_a \varphi \text{ for some } a \in \mathcal{D} \}.$$ 

Note that since $\varphi$ is a sentence, all occurrences of $\uparrow$ in $\varphi$ are bounded. Thus, it makes no difference which data value $a$ is used in the statement $w, 1 \models_a \varphi$ of the definition of $L(\varphi)$.

## 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 6** The emptiness problem for weak 3-PA is undecidable.

**Proof.** The proof is very similar to the proof of the undecidability of the emptiness problem for weak 5-PA in [9]. We observe that the same proof can be easily adopted to weak 3-PA. The details are provided below. It uses a reduction from the Post Correspondence Problem (PCP), which is well known to be undecidable [5]. An instance of PCP is a sequence of pairs $(x_1, y_1), \ldots, (x_n, y_n)$, where each $x_1, y_1, \ldots, x_n, y_n \in \{\alpha, \beta\}^*$. 

9
This instance has a solution if there exist indexes \( i_1, \ldots, i_m \in \{1, \ldots, n\} \)
such that \( x_{i_1} \cdots x_{i_m} = y_{i_1} \cdots y_{i_m} \). The PCP asks whether a given instance of the problem has a solution.

In the following we show how to encode a solution of an instance of PCP into a data word which possesses properties that can be checked by a weak 3-PA. Let \( \Sigma = \{1, \ldots, n, \alpha, \beta, \gamma\} \). We denote by \( x_i = \nu_{i,1} \cdots \nu_{i,l_i} \) for each \( i = 1, \ldots, n \). Each string \( x_i \) is encoded as \( \text{Enc}(x_i) = (\nu_{i,1})_{a_{i,1}} \cdots (\nu_{i,l_i})_{a_{i,l_i}} \) where \( a_{i,1}, \ldots, a_{i,l_i} \) are pairwise different.

The string \( x_{i_1} x_{i_2} \cdots x_{i_m} \) can be encoded as

\[
\text{Enc}(x_{i_1}, x_{i_2}, \ldots, x_{i_m}) = \left( \begin{array}{c} i_1 \\ b_1 \end{array} \right) \text{Enc}(x_{i_1}) \left( \begin{array}{c} i_2 \\ b_2 \end{array} \right) \text{Enc}(x_{i_2}) \cdots \left( \begin{array}{c} i_m \\ b_m \end{array} \right) \text{Enc}(x_{i_m})
\]

where all the data values that appear in it are pairwise different. Note that even if \( i_j = i_{j'} \) for some \( j, j' \), the data values that appear in \( \text{Enc}(x_{i_j}) \) do not appear in \( \text{Enc}(x_{i_{j'}}) \) and vice versa. The idea is each data value is used to mark a place in the string.

Similarly, the string \( y_{j_1} y_{j_2} \cdots y_{j_l} \) can be encoded as

\[
\text{Enc}(y_{j_1}, y_{j_2}, \ldots, y_{j_l}) = \left( \begin{array}{c} j_1 \\ c_1 \end{array} \right) \text{Enc}(y_{j_1}) \left( \begin{array}{c} j_2 \\ c_2 \end{array} \right) \text{Enc}(y_{j_2}) \cdots \left( \begin{array}{c} j_l \\ c_l \end{array} \right) \text{Enc}(y_{j_l})
\]

where the data values that appear in it are pairwise different.

Now the data word

\[
\left( \begin{array}{c} i_1 \\ b_1 \end{array} \right) \text{Enc}(x_{i_1}) \cdots \left( \begin{array}{c} i_m \\ b_m \end{array} \right) \text{Enc}(x_{i_m}) \left( \begin{array}{c} j_1 \\ c_1 \end{array} \right) \text{Enc}(y_{j_1}) \cdots \left( \begin{array}{c} j_l \\ c_l \end{array} \right) \text{Enc}(y_{j_l})
\]

constitutes a solution to the instance of PCP if and only if

\[
i_1 i_2 \cdots i_m = j_1 j_2 \cdots j_l \quad (1)
\]

\[
\text{Proj}_\Sigma(\text{Enc}(x_{i_1}) \cdots \text{Enc}(x_{i_m})) = \text{Proj}_\Sigma(\text{Enc}(y_{j_1}) \cdots \text{Enc}(y_{j_l})) \quad (2)
\]

Now, in order to able to check such property with weak 3-PA, we demand the following additional criteria.

1. \( b_1 \cdots b_m = c_1 \cdots c_l \);

2. \( \text{Proj}_D(\text{Enc}(x_{i_1}) \cdots \text{Enc}(x_{i_m})) = \text{Proj}_D(\text{Enc}(y_{j_1}) \cdots \text{Enc}(y_{j_l})) \)
3. For any two positions $h_1$ and $h_2$ where $h_1$ is to the left of the delimiter ($\langle$) and $h_2$ is to the right of the delimiter ($\rangle$), if both of them have the same data value, then both of them are labelled with the same label.

All the Criterias (1)--(3) imply Equations 1 and 2.

Because the data values that appears in $\text{Proj}_2(\text{Enc}(x_{i_1}), \ldots, \text{Enc}(x_{i_m}))$ are pairwise different, all of them are checkable by three pebbles in the “weak” manner. For example, to check Criteria (1), the automaton does the following.

- Check that $b_1 = c_1$.
- Check that for each $i = 1, \ldots, m - 1$, there exists $j$ such that $a_i a_{i+1} = b_j b_{j+1}$. It can be done by placing pebble 3 to read $a_i$ and pebble 2 to read $a_{i+1}$, then using pebble 3 to search on the other side of $\$$ for the index $j$.
- Finally, check that $b_m = c_l$.

Criteria (2) can be checked similarly and Criteria (3) is straightforward. The reduction is now complete and we prove that the emptiness problem for weak 3-PA is undecidable.

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). 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. This result settles a question left open in [9]: Can weak PA be simulated by alternating RA? We refer the reader to Appendix C for the details of the proof.

**Theorem 7** For every weak 2-PA $\mathcal{A}$, there exists a one-way alternating 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 7, we immediately obtain the decidability of weak 2-PA because the emptiness problem for one-way alternating 1-RA is decidable [4, Theorem 4.4].

**Corollary 8** The emptiness problem for weak 2-PA is decidable.
We devote the rest of this section to the proof of Theorem 7. Let $\mathcal{A} = (Q, q_0, \mu, F)$ be a weak 2-PA. We assume that $\mathcal{A}$ is deterministic. Furthermore, we normalize the behavior of $\mathcal{A}$ as follows.

- Pebble 1 is lifted only after it reads the right-end marker symbol $\triangleright$.
- 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.

On input word $w = (\sigma_1^{d_1}) \cdots (\sigma_n^{d_n})$, the run of $\mathcal{A}$ on $\langle w \triangleright \rangle$ can be depicted as a tree shown in Figure 3.

Figure 1: The tree representation of a run of $\mathcal{A}$ on $w = (\sigma_1^{d_1}) \cdots (\sigma_n^{d_n})$.

The meaning of the tree is as follows.
\(q_0, q_1, \ldots, q_n, q_{n+1}\) are the states of \(A\) when pebble 2 is the head pebble reading the positions 0, 1, \ldots, \(n, n+1\), respectively, that is, the symbols \(\prec, (\sigma^1_d), \ldots, (\sigma^n_d), \succ\), respectively.

- \(q_f\) is the state of \(A\) after pebble 2 reads the symbol \(\succ\).

- For \(1 \leq i \leq j \leq n\), \(p_{i,j}\) is the state of \(A\) when pebble 1 is the head pebble above 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 \(A\) immediately after pebble 1 is lifted and pebble 2 is above the position \(i\).

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

\[A'\] “verifies” the guess \(p_1\)

\[A'\] “guesses” the state \(p_1\) and then “splits”

Figure 2: The corresponding run of \(A'\) to the one in Figure 3.
Now the simulation of $A$ by a one-way alternating 1-RA $A'$ becomes straightforward by transforming the tree in Figure 3 into a tree depicting the computation of $A'$ on the same word $w$.

Roughly, the automaton $A'$ is defined as follows.

- The states of $A'$ are elements of $Q \cup (Q \times Q)^2$;
- 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 $q_i$, $A'$ “guesses” the state in which pebble 1 is eventually lifted, i.e. the state $p_i$, and stores it in its internal state. That is, $A'$ enters into the state $(q_i, p_i)$.

2. $A'$ “splits” its computation (conjunctively) into two branches.
   - In one branch, assuming that the guess $p_i$ is correct, $A'$ moves right and enters into the state $q_{i+1}$, simulating the transition $(2, \emptyset, p_i) \rightarrow (q_{i+1}, \text{right})$. 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 $A'$ stores the data value $d_i$ in its register and simulates the run of pebble 1 on $(\sigma_d^i) \cdots (\sigma_d^n)$ to “verify” that the guess $p_i$ is correct.
     That is, $A'$ accepts only if it ends in the state $(p_i, p_i)$.

Figure 3 shows the corresponding run of $A'$ on the same word.

4 Complexity of weak 2-PA

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

\[\text{Actually } A' \text{ needs some other auxiliary states. However, for the intuitive explanation here the set } Q \cup (Q \times Q) \text{ suffices. We refer the reader to Appendix C for the details.}\]
Emptiness problem. The emptiness problem for weak 2-PA. That is, given a weak 2-PA \( A \), is \( L(A) = \emptyset \)?

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

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

The emptiness problem, as we have seen in the previous section, is decidable. The labelling and data value membership problem are definitely decidable. To solve the labelling problem, one simply iterates all possible sequence \( \sigma_1 \cdots \sigma_n \in \Sigma^n \) and runs \( A \) to check whether \( (\sigma_1^d_1) \cdots (\sigma_n^d_n) \in L(A) \). Such straightforward algorithm requires \( O(|\Sigma| \cdot n^2) \) computational steps.

Similarly, to solve the data value membership problem, one can iterate all possible sequence of data values \( d_1 \cdots d_n \) and run \( A \) to check whether \( (\sigma_1^d_1) \cdots (\sigma_n^d_n) \in L(A) \). Since the word is of length \( n \), one simply needs to consider up to \( n \) different data values. Such algorithm takes \( O(n^2 \cdot n^2) \) computational steps.

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

We start the proof with a few simple examples of languages accepted by weak 2-PA. Though simple, they are very crucial in determining the complexity of the emptiness problem for weak 2-PA.

Example 9 Let \( \Sigma = \{ \alpha, \beta \} \). We define the \( \Sigma \)-data language \( L_{inc} \) which consists of the data words of the following form:

\[
\left( \alpha_{a_1} \cdots \alpha_{a_m} \right)_{u_1} \left( \beta_{b_1} \cdots \beta_{b_n} \right)_{u_2},
\]

where
- the data values \( a_1, \ldots, a_m \) are pairwise different;
- the data values \( b_1, \ldots, b_n \) are pairwise different;
• \( \text{Proj}_\Sigma(w_1) = \alpha^m; \)
• \( \text{Proj}_\Sigma(w_2) = \beta^n; \)
• \( \text{Cont}_\Delta(w_1) \subseteq \text{Cont}_\Delta(w_2). \)

All these conditions can be checked by weak 2-PA. The intention of data words in \( L_{\text{inc}} \) is to represent the inequality \( m \leq n. \)

**Example 10** Let \( \Sigma = \{\alpha, \beta\} \). For a fixed \( l \geq 0 \), we define the language \( L_{\text{inc},+1} \) which consists of the data words of the following form:

\[
\begin{array}{c}
(\alpha)_{a_1} \cdots (\alpha)_{a_m} (\beta)_{b_1} \cdots (\beta)_{b_n} \\
\text{w}_1 \quad \text{w}_2
\end{array}
\]

where

• the data values \( a_1, \ldots, a_m \) are pairwise different;
• the data values \( b_1, \ldots, b_n \) are pairwise different;
• \( \text{Proj}_\Sigma(w_1) = \alpha^m; \)
• \( \text{Proj}_\Sigma(w_2) = \beta^n; \)
• For each \( a_i \in \text{Cont}_\Delta(w_1), a_i \neq b_1. \)
• \( \{a_1, \ldots, a_m\} \subseteq \{b_2, \ldots, b_n\}. \)

Again, all these conditions can be checked by weak 2-PA. The intention of data words in \( L_{\text{inc},+1} \) is to represent the inequality \( m + 1 \leq n. \)

**Example 11** Let \( \Sigma = \{\alpha, \beta\} \). For a fixed \( l \geq 0 \), we define the language \( L_{\text{inc},-1} \) which consists of the data words of the following form:

\[
\begin{array}{c}
(\alpha)_{a_1} \cdots (\alpha)_{a_m} (\beta)_{b_1} \cdots (\beta)_{b_n} \\
\text{w}_1 \quad \text{w}_2
\end{array}
\]

where
• the data values $a_1, \ldots, a_m$ are pairwise different;
• the data values $b_1, \ldots, b_n$ are pairwise different;
• $\text{Proj}_\Sigma(w_1) = \alpha^m$;
• $\text{Proj}_\Sigma(w_2) = \beta^n$;
• The symbol $a_1 \notin \{b_1, \ldots, b_n\}$;
• For each $i = 2, \ldots, m$, $a_i \in \{b_1, \ldots, b_n\}$.

Again, all these conditions can be checked by weak 2-PA. The intention of data words in $L_{\text{inc} - 1}$ is to represent the inequality $m - 1 \leq n$.

Theorem 12 The emptiness problem for weak 2-PA is not primitive recursive.

Proof. The proof is by simulation of incrementing counter automata. It follows closely the proof of similar lower bound for one-way alternating 1-RA [4, Theorem 2.9]. It is known that the emptiness problem for incrementing counter automata is decidable [8, Theorem 6], but not primitive recursive [10]. We refer the reader to Appendix A for the formal definition of incrementing counter automata.

In short, an incrementing $l$-counter automaton over $\Sigma$ is an automaton with $l$ counters, operates on words over $\Sigma$, and the value in each counter is allowed to erroneously increase, hence, the name incrementing.

A configuration is a tuple $(q, \sigma, \nu)$ where $q$ is a state, $\sigma$ is the current symbol read and $\nu : \{1, \ldots, l\} \rightarrow \mathbb{N}$, where $\nu(i)$ denotes the value stored in counter $i$.

Now a configuration $(q, \nu)$ can be encoded as a $(Q \cup \Sigma \cup \{c_1, \ldots, c_l\})$-data word as follows.

$$
\left( q \right) \left( \sigma \right) \left( c_1 \right) \left( a_{1,1} \right) \cdots \left( c_1 \right) \left( a_{1,\nu(1)} \right) \cdots \left( c_l \right) \left( a_{l,1} \right) \cdots \left( c_l \right) \left( a_{l,\nu(l)} \right).
$$

where the symbols $a_{1,1}, \ldots, a_{l,\nu(l)}$ are pairwise different. The labels $c_1, \ldots, c_l$ are used as pointers that the current data value is part of the encoding of the counters $\nu(1), \ldots, \nu(l)$, respectively.
Since the automaton allows for erroneous increment of values in each counter, we can check the validity of the application of each transition, like in Examples 9, 10 and 11.

Now we are going to show the NP-completeness of the labelling problem. It is by a reduction from graph 3-colorability problem.

Given an undirected graph $G = (V, E)$, let $V = \{1, \ldots, n\}$ and $E = \{(i_1, j_1), \ldots, (i_m, j_m)\}$. We can take $i_1j_1 \cdots i_mj_m$ as the sequence of data values. Then, we construct a weak 2-PA $A$ 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.

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

$$
\left( \sigma_1 \right)_{i_1} \left( \sigma_2 \right)_{j_1} \cdots \left( \sigma_{2m} \right)_{j_m} \in L(A),
$$

and the NP-completeness of the labeling problem follows.

The NP-completeness 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 $n_r, n_g, n_b$ in unary form, can the graph $G$ be colored 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 $\Sigma = \{\vartheta_R, \vartheta_G, \vartheta_B, \nu_1, \ldots, \nu_n\}$. We take

$$
\nu_{i_1}\nu_{j_1}\cdots\nu_{i_m}\nu_{j_m}\vartheta_R^1\cdots\vartheta_R^n\vartheta_G^1\cdots\vartheta_G^n\vartheta_B^1\cdots\vartheta_B^n
$$

as the sequence of finite labels.

Then, we construct a weak 2-PA over $\Sigma$ that accepts data words of the form

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

- $\nu_{i_1}, \nu_{j_1}, \ldots, \nu_{i_m}, \nu_{j_m} \in \{\nu_1, \ldots, \nu_n\}$;
- in the sub-word $(\nu_{i_1})^{c_1}(\nu_{j_1})^{d_1} \cdots (\nu_{i_m})^{c_m}(\nu_{j_m})^{d_m}$, every two positions with the same labels have the same data value, see Example 4;
- 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 $A$, thus, enable $A$ 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 $1, \ldots, m$. Furthermore, the unary form of $n_r, n_g$ and $n_b$ is crucial here to ensure that the number of the states of $A$ is still polynomial in the length of the input.

Now the graph $G$ is 3-colorable with constraint if and only if there exits $c_1d_1 \cdots c_md_m a_1 \cdots a_n a'_1 \cdots a'_{n_g} a''_1 \cdots a''_{n_b}$ such that

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

is accepted by $A$, and the NP-completeness of data value membership problem follows.

## 5 Top view weak $k$-PA

In this section we are going to restrict the definition of weak $k$-PA so that its emptiness problem becomes decidable. 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 \(i + 2, \ldots, k\).

Formally, the transitions of top view weak \(k\)-PA \(\mathcal{A} = \langle Q, q_0, \mu, F \rangle\) are of the form

\[
(i, \sigma, V, q) \rightarrow (q', \text{act})
\]

where \(V\) is either \(\emptyset\) or \(\{i + 1\}\).

The definition 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 are just the same as weak 2-PA. 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.

**Theorem 13** 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 7. 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 pebble \(i + 2, \ldots, k\). It only need to store the data value seen by pebble \(i + 1\), thus, one register suffices.

Following Theorem 13, we immediately obtain the decidability of the emptiness problem for top view weak \(k\)-PA.

**Corollary 14** The emptiness problem for top view weak \(k\)-PA is decidable.

**Remark 15** Since the emptiness problem for ordinary 2-PA and for weak 3-PA is already undecidable (See Theorem 6 and [7, Theorem 4]), it seems that top view weak PA is a tight boundary of a subclass of PA languages for which the emptiness problem is decidable.
Theorem 16 For every sentence $\psi \in LTL^1(\Sigma, X, U)$, there exists a top view weak $k$-PA $A_\psi$, where $k = fqr(\psi) + 1$, such that $L(A_\psi) = L(\psi)$.

Proof. Let $\psi$ be an LTL$^1(\Sigma, X, U)$ sentence. We construct an alternating top view weak $k$-PA $A_\psi$, where $k = fqr(\psi) + 1$ such that given a data word $w$, the automaton $A_\psi$ checks whether $w, 1 \models \psi$. $A_\psi$ accepts if it is so. Otherwise, it rejects.

Intuitively, the computation of $w, 1 \models \psi$ is done recursively as follows. The automaton $A_\psi$ "consists of" the automata $A_\varphi$ for all sub-formula $\varphi$ of $\psi$, including $A_\epsilon$ to represent the empty formula $\epsilon$.

- The automaton $A_\epsilon$ accepts every data words.
- If $\psi = \sigma \varphi$, then check whether the current label is $\sigma$. If it is not, then $A$ rejects immediately. Otherwise, $A_\psi$ proceeds to run $A_\varphi$.
- If $\psi = \varphi \lor \varphi'$, then $A_\psi$ nondeterministically chooses one of $A_\varphi$ or $A_{\varphi'}$ and proceeds to run one of them.
- If $\psi = \varphi \land \varphi'$, then $A_\psi$ splits its computation (by conjunctive branching) into two and proceed to run both of $A_\varphi$ and $A_{\varphi'}$.
- If $\psi = X \varphi$, then $A_\psi$ moves to the right one step. If it reads the right-end marker, then the automaton rejects immediately. Otherwise, it proceeds to run $A_\varphi$.
- If $\psi = \uparrow \varphi$, then $A_\psi$ checks whether the data value seen by its head pebble is the same as the one seen by the second last placed pebble. If it is not the same, then it rejects immediately. Otherwise, it proceeds to run $A_\varphi$.
- If $\psi = \downarrow \varphi$, then $A_\psi$ places a new pebble and proceeds to run $A_\varphi$.
- If $\psi = \varphi U \varphi'$, then $A_\psi$ it runs $A_{\varphi'} \lor (\varphi \land X (\varphi \land \varphi'))$.
- If $\psi = \neg \varphi$, then $A_\psi$ runs $A_\varphi$. If $A_\varphi$ accepts, then $A_\psi$ rejects. Otherwise, $A_\psi$ accepts.

Note that since $fqr(\varphi) = k$, on each computation path then the automaton $A_\psi$ only needs to place the pebble $k$ times, thus, $A_\psi$ requires only $k + 1$. It is a straight forward induction to show that $L(A_\psi) = L(\psi)$. \qed
Our next results deals with the expressive power of \(\text{LTL}^1(\Sigma, X, U)\) based on the freeze quantifier rank. It is an analog of the classical hierarchy of first order logic based on the ordinary quantifier rank. We start by defining an \(\text{LTL}^1(\Sigma, X, U)\) sentence for the language \(\mathcal{R}^+_m\) defined in Subsection 2.2.

**Lemma 17** For each \(k = 1, 2, 3, \ldots\), there exists a sentence \(\psi_k\) in \(\text{LTL}^1(\Sigma, X, U)\) such that \(L(\psi_k) = \mathcal{R}^+_k\) and

- \(\text{fqr}(\psi_1) = 1\); and
- \(\text{fqr}(\psi_k) = k - 1\), when \(k \geq 2\).

**Proof.** First, we define a formula \(\varphi_k\) such that \(\text{fqr}(\varphi_k) = k - 1\) and for every data word \(w = (\sigma_{d_1}) \cdots (\sigma_{d_n})\), for every \(i = 1, \ldots, n\),

\[
  w, i \models_{d_i} \varphi_k \quad \text{if and only if} \quad (\sigma_{d_i}) \cdots (\sigma_{d_n}) \in \mathcal{R}^+_k.
\]

We construct \(\varphi_k\) inductively as follows.

- \(\varphi_1 := X(\neg \uparrow) \land \neg (X\text{True})\).
- For each \(k = 1, 2, 3, \ldots\),

\[
  \varphi_{k+1} := X(\neg \uparrow) \land X(\downarrow X((\neg \uparrow)U(\uparrow \land \varphi_k))).
\]

Note that since \(\text{fqr}(\varphi_1) = 0\), then for each \(k = 1, 2, \ldots\), \(\text{fqr}(\varphi_k) = k - 1\).

Assuming first that \(\varphi_k\) satisfies the property in Equation 3, the desired sentence \(\psi_k\) is defined as follows.

- \(\psi_1 := \downarrow (X(\neg \uparrow) \land \neg (X\text{True}))\).
- For each \(k = 2, 3, \ldots\),

\[
  \psi_k := \downarrow (X(\neg \uparrow)) \land X(\downarrow X((\neg \uparrow)U(\uparrow \land \varphi_{k-1}))).
\]

Since \(\text{fqr}(\varphi_{k-1}) = k - 2\), then \(\text{fqr}(\psi_k) = k - 1\).

Now we want to show that the formula \(\varphi_k\) satisfies Equation 3. The proof is by induction on \(k\). The base case, \(k = 1\), is trivial. Suppose, for the induction hypothesis, the formula \(\varphi_k\) satisfies Equation 3.
The induction step is as follows. Let \( w = (d_1) \cdots (d_n) \). We have the following chain of application of the semantics of LTL.

\[
\begin{align*}
    w, i & \models_{d_i} \varphi_{k+1} \\
    & \Downarrow \\
    w, i & \models_{d_i} X(\neg \uparrow) \land X(\downarrow X((\neg \uparrow)U(\uparrow \land \varphi_k))) \\
    & \Downarrow \\
    w, i & \models_{d_i} X(\neg \uparrow) \quad \text{and} \quad w, i \models_{d_i} X(\downarrow X((\neg \uparrow)U(\uparrow \land \varphi_k)))
\end{align*}
\]

For the first part, we have

\[
    w, i \models_{d_i} X(\neg \uparrow) \quad \text{if and only if} \quad d_i \neq d_{i+1} \tag{4}
\]

Now we evaluate the second part.

\[
\begin{align*}
    w, i & \models_{d_i} X(\downarrow X((\neg \uparrow)U(\uparrow \land \varphi_k))) \\
    & \Downarrow \\
    w, i + 1 & \models_{d_i} (- \downarrow X((\neg \uparrow)U(\uparrow \land \varphi_k))) \\
    & \Downarrow \\
    w, i + 1 & \models_{d_{i+1}} X((\neg \uparrow)U(\uparrow \land \varphi_k)) \\
    & \Downarrow \\
    w, i + 2 & \models_{d_{i+1}} (\neg \uparrow)U(\uparrow \land \varphi_k) \tag{5}
\end{align*}
\]

Equation 5 holds if and only if there exists \( j \) such that \( i + 2 \leq j \) and

1. \( w, j \models_{d_{i+1}} \uparrow \land \varphi_k \),

2. \( w, j' \models_{d_{i+1}} \neg \uparrow \), for each \( j' = i + 1, \ldots, j - 1 \).

By the semantics of LTL and the induction hypothesis, Clause 1 holds if and only if \( d_j = d_{i+1} \) and \((d_j) \cdots (d_n) \in \mathcal{R}_k^+ \). The meaning of Clause 2 is \( d_j \neq d_{i+1} \), for each \( j' = i + 1, \ldots, j - 1 \). Both clauses, together with Equation 4, means that \((d_j) \cdots (d_n) \in \mathcal{R}_{k+1}^+ \). This completes the induction hypothesis. \( \square \)

**Lemma 18** For each \( k = 1, 2, \ldots \), the language \( \mathcal{R}_{k+1}^+ \) is not expressible by a sentence in \( LTL_1(\Sigma, X, U) \) of freeze quantifier rank \( k - 1 \).
Proof. By Theorem 5, $R_{k+1}^+$ is not accepted by weak $k$-PA, thus, it is also not accepted by top-view $k$-PA. Then, by Theorem 16, $R_{k+1}^+$ is not expressible by $\text{LTL}^\downarrow(\Sigma, X, U)$ sentence of freeze quantifier rank $k - 1$. □

Combining both Lemmas 17 and 18, we obtain the following strict hierarchy of $\text{LTL}^\downarrow(\Sigma, X, U)$ based on its freeze quantifier rank.

Theorem 19 For each $k = 1, 2, \ldots$, the class of sentences in $\text{LTL}^\downarrow(\Sigma, X, U)$ of freeze quantifier rank $k + 1$ is strictly more expressive than those of freeze quantifier rank $k$.

References

[1] Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. In Proceedings of the 16th International Symposium on Fundamentals of Computation Theory, FCT 2007, Budapest, Hungary, August 27-30, 2007, volume 4639 of Lecture Notes in Computer Science, pages 88–99. Springer, 2007.

[2] Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, pages 7–16. IEEE Computer Society, 2006.

[3] Mikolaj Bojanczyk, Mathias Samuelides, Thomas Schwentick, and Luc Segoufin. Expressive power of pebble automata. In Part I of the Proceedings of Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, volume 4051 of Lecture Notes in Computer Science, pages 157–168. Springer, 2006.

[4] Stéphane Demri and Ranko Lazic. Ltl with the freeze quantifier and register automata. In Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, pages 17–26. IEEE Computer Society, 2006.

[5] John E. Hopcroft and Jeffrey D. Ullman. Introduction to Automata Theory, Languages and Computation. Addison-Wesley, 1979.
A Counter Automata

A *Minsky k-counter automata* (CA), with $\epsilon$-transitions and zero testing, is a tuple $A = \langle \Sigma, Q, q_0, \delta, F \rangle$, where

- $\Sigma$ is a finite alphabet;
- $Q$ is a finite set of states;
- $q_0$ is the initial state;
Given a word \( w = \sigma_1 \cdots \sigma_n \in \Sigma^* \), a configuration of \( \mathcal{A} \) is a triple \([i, q, v]\) where \( 0 \leq i \leq n \), \( q \in Q \) and a counter valuation \( v : \{1, \ldots, k\} \rightarrow \mathbb{N} \), where \( \mathbb{N} \) is the set of natural number \( \{1, 2, 3, \ldots\} \).

The initial configuration is \([0, q_0, v_0]\) where \( v_0(j) = 0 \) for each \( j = 1, \ldots, k \).

The run of \( \mathcal{A} \) on \( w \) is a sequence \([0, q_0, v_0], [1, q_1, v_1], \ldots, [n, q_n, v_n]\) where for each \( i = 0, \ldots, n - 1 \), there exists a transition \((q_i, \sigma_{i+1}, l, q_{i+1}) \in \delta \) and

- if \( l = (\text{inc}, j) \) for some \( j = 1, \ldots, k \), then \( v_{i+1}(j) = v_i(j)+1 \) and for all other \( j' \neq j \), \( v_{i+1}(j') = v_i(j') \).
- if \( l = (\text{dec}, j) \) for some \( j = 1, \ldots, k \), then \( v_i(j) > 0 \) and \( v_{i+1}(j) = v_i(j) - 1 \) and for all other \( j' \neq j \), \( v_{i+1}(j') = v_i(j') \).
- if \( l = (\text{ifz}, j) \) for some \( j = 1, \ldots, k \), then \( v_i(j) = 0 \) and \( v_{i+1} = v_i \).

The word \( w \) is accepted by \( \mathcal{A} \) if \( q_n \in F \). As usual, we denote by \( L(\mathcal{A}) \) the set of all words over \( \Sigma \) accepted by \( \mathcal{A} \).

We say that the automaton \( \mathcal{A} \) is incrementing if its counters may erroneously increase at any time. More precisely, The run of an incrementing \( \mathcal{A} \) on \( w \) is a sequence of configurations \([0, q_0, v_0], [1, q_1, v_1], \ldots, [n, q_n, v_n]\) where for each \( i = 0, \ldots, n - 1 \), there exists a transition \((q_i, \sigma_{i+1}, l, q_{i+1}) \in \delta \) and

- if \( l = (\text{inc}, j) \) for some \( j = 1, \ldots, k \), then \( v_{i+1}(j) \geq v_i(j)+1 \) and for all other \( j' \neq j \), \( v_{i+1}(j') \geq v_i(j') \).
- if \( l = (\text{dec}, j) \) for some \( j = 1, \ldots, k \), then \( v_i(j) > 0 \) and \( v_{i+1}(j) \geq v_i(j) - 1 \) and for all other \( j' \neq j \), \( v_{i+1}(j') \geq v_i(j') \).
- if \( l = (\text{ifz}, j) \) for some \( j = 1, \ldots, k \), then \( v_i(j) = 0 \) and for all \( j' = 1, \ldots, k \), \( v_{i+1}(j') \geq v_i(j') \).

**Theorem 20** [4, Theorem 2.9] (See also [8, Theorem 6] and [10]) The nonemptiness problem for incrementing counter automata is decidable, but not primitive recursive.

26
B Register automata

We are only going to sketch roughly the definition of register automata. Readers interested in its more formal treatment can consult [4, 6]. In essence, a register automaton, or, shortly $k$-RA, is a finite state automaton equipped with a header to scan the input and $k$ registers, numbered from 1 to $k$. Each register can store exactly one data value from $D$. The automaton is two-way if the header can move to the left or to the right. It is alternating if it is allowed to branch into a finite number of parallel computations.

More formally, a two-way alternating $k$-RA over the label $\Sigma$ is a tuple $A = \langle \Sigma, Q, q_0, u_0, \mu, F \rangle$ where

- $Q$, $q_0 \in Q$ and $F \subseteq Q$ are the finite state of states, the initial state and the set of final states, respectively.
- $u_0 = a_1 \cdots a_k$ is the initial content of the registers.
- $\mu$ is a set of transitions of the following form.

  i) $(q, \sigma) \rightarrow q'$ where $a \in \{\leftarrow, \rightarrow\}$ and $q, q' \in Q$.
  That is, if the automaton $A$ is in state $q$ and the header is currently reading either of the symbols $\leftarrow, \rightarrow$, then the automaton can enter the state $q'$.

  ii) $(q, \sigma, V) \rightarrow q'$ where $\sigma \in \Sigma$, $V \subseteq \{1, \ldots, k\}$ and $q, q' \in Q$.
  That is, if the automaton $A$ is in state $q$ and the header is currently reading a position labeled with $\sigma$ and $V$ is the set of all registers containing the current data value, then the automaton can enter the state $q'$.

  iii) $q \rightarrow (q', I)$ where $I \subseteq \{1, \ldots, k\}$ and $q, q' \in Q$.
  That is, if the automaton $A$ is in state $q$, then the automaton can enter the state $q'$ and store the current data value into the registers whose indices belong to $I$.

  iv) $q \rightarrow (q_1 \land \cdots \land q_i)$ and $q \rightarrow (q_1 \lor \cdots \lor q_i)$ where $i \geq 1$ and $q, q' \in Q$.
  That is, if the automaton $A$ is in state $q$, then it can decide to perform conjunctive or disjunctive branching into the states $q_1, \ldots, q_i$.

  v) $q \rightarrow (q', \text{act})$ where $\text{act} \in \{\text{left, right}\}$ and $q, q' \in Q$.
  That is, if the automaton $A$ is in state $q$, then it can enter the state $q'$ and move to the next or the previous word position.
A register automaton is called non deterministic if the branchings of state (in item (iv)) are all disjunctive. It is called one-way if the header is not allowed to move to the previous word position.

A configuration \( \gamma = [j, q, b_1 \cdots b_k] \) of the automaton \( \mathcal{A} \) consists of the current position of the header in the input word \( j \), the state of the automaton \( q \) and the content of the registers \( b_1 \cdots b_k \). The configuration \( \gamma \) is called accepting if the state is a final state in \( F \).

From each configuration \( \gamma \), the automaton performs legitimate computation according to the transition relation and enters another configuration \( \gamma' \). If the transition is branching, then it can split into several configurations \( \gamma'_1, \ldots, \gamma'_m \).

Similarly, we can define the notion of leads to acceptance for a configuration \( \gamma \) as follows.

- Every accepting configuration leads to acceptance.
- If \( \gamma' \) is the configuration obtained from \( \gamma \) by applying a non-branching transition, then \( \gamma \) leads to acceptance if and only if \( \gamma' \) leads to acceptance.
- If \( \gamma'_1, \ldots, \gamma'_m \) are the configurations obtained from \( \gamma \) by applying a disjunctive branching transition, then \( \gamma \) leads to acceptance if and only if at least one of \( \gamma'_1, \ldots, \gamma'_m \) leads to acceptance.
- If \( \gamma'_1, \ldots, \gamma'_m \) are the configurations obtained from \( \gamma \) by applying a conjunctive branching transition, then \( \gamma \) leads to acceptance if and only if all of \( \gamma'_1, \ldots, \gamma'_m \) lead to acceptance.

An input word \( w \) is accepted by \( \mathcal{A} \) if the initial configuration leads to acceptance. As usual, \( L(\mathcal{A}) \) denotes the language accepted by \( \mathcal{A} \).

C Generalization of Theorem 7

Let \( \mathcal{A} = (Q, q_0, \mu, F) \) be a weak \( k \)-PA. We will show how to construct one-way alternating \((k - 1)\)-RA \( \mathcal{A}' \). For our convenience, we assume that \( \mathcal{A} \) is deterministic. We also assume that \( \mathcal{A} \) behaves as follows.

- For every configuration \( \gamma \) of \( \mathcal{A} \), there exists a transition in \( \mu \) that applies to it.
- Only pebble $k$ can enter a final state and it does so only after it reads the right-end marker $\triangleright$.

- For every $i = 2, \ldots, k$, immediately after pebble $i$ moves right, pebble $i - 1$ is placed.

- For every $i = 1, \ldots, k - 1$, pebble $i$ is lifted only when it reaches the right-end marker $\triangleright$.

- For every $i = 1, \ldots, k - 1$, immediately after pebble $i$ is lifted, pebble $(i + 1)$ moves right.

See Subsection D.2 on how this normalization can be done.

We also assume that the set of states $Q$ is partitioned into $Q_1 \cup \cdots \cup Q_k$ where $Q_i \cap Q_j$ whenever $i \neq j$ and $Q_i$ is the set of states when pebble $i$ is in control.

The automaton $A' = \langle Q', q'_0, u_0, \mu', F' \rangle$ is defined as follows.

- The set of states is $Q' = Q \cup Q^2 \cup Q^3 \cup \tilde{Q} \cup \tilde{Q} \times Q$, where $\tilde{Q} = \{ \tilde{q} \mid q \in Q \}$ and $\tilde{Q} \times Q = \{ (\tilde{q}, p) \mid p, q \in Q \}$.

- The initial state is $q'_0 = q_0 \in Q_k$.

- The initial assignment is $\#^{k-1}$.

- The set of final states is $F' = F \cup \{(q, q) : q \in Q\}$.

For our convenience, we number the registers of $A'$ from 2 to $k$, not from 1 to $(k - 1)$. The set of transitions $\mu'$ consists of the following.

- For $i = k - 1, \ldots, 1$, we have the following transitions.

  1. For each transition $(i, \sigma, V, q) \rightarrow (q', \text{right}) \in \mu$, there are transitions $((q, p), \sigma, V) \rightarrow (q', p) \in \mu'$ for all $p \in Q_{i+1}$.

  2. For each transition $(i, P, V, q) \rightarrow (q', \text{place-pebble}) \in \mu$, there are the following transitions in $\mu'$. For every $p \in Q_{i+1}$,

\[
\begin{align*}
(q, p) &\rightarrow (\tilde{q}, p), \{i\} \\
(\tilde{q}, p) &\rightarrow \bigvee_{p_j \in Q_i} (q, p_j, p) \\
(q, p_j, p) &\rightarrow (p_j, p) \land (q', p_j) \quad \text{for every } p_j \in Q_i
\end{align*}
\]
For $i = k$, there are the following transitions in $\mu'$.

1. For each transition $(k, \sigma, \emptyset, q) \rightarrow (q', \text{right}) \in \mu$, there is a transition $(q, \sigma, \emptyset) \rightarrow (q') \in \mu'$.

2. For each transition $(k, \sigma, \emptyset, q) \rightarrow (q', \text{place-pebble}) \in \mu$, there are the following transitions in $\mu'$.

\[
\begin{align*}
q & \rightarrow \tilde{q}, \{k\} \\
\tilde{q} & \rightarrow \bigvee_{p_j \in Q_k} (q, p_j) \\
(q, p_j) & \rightarrow p_j \land (q', p_j) \text{ for every } p_j \in Q_k
\end{align*}
\]

We can show the following proposition by straightforward induction on $i$.

**Proposition 21** Let $w = (\sigma_1) \cdots (\sigma_n)$ be a $\Sigma$-data word. For $i = 1, \ldots, k-1$, there exists an $i$-run $[i, q_1, \theta_1] \vdash_* [i, q_2, \theta_2]$ of $A$ on $w$ and $[i, q_2, \theta_2] \vdash [i + 1, q_3, \theta_3]$ if and only if the configuration $[\theta(i), (q_1, q_3), u_2 \cdots u_k]$ of $A'$ on $w$ leads to acceptance, where $u_j = a_{\theta(j)}$, for $j = i + 1, \ldots, k$.

Then, by the definition of $\mu'$, we can easily deduce the following. For each $\ell = 1, \ldots, n$,

\[
[k, q_1, \theta_1] \vdash_A [k - 1, q_2, \theta_2] \vdash^* \vdash_A [k - 1, q_3, \theta_3] \vdash_A [k, q_4, \theta_4] \vdash_A [k, q_5, \theta_5]
\]

is a $k$-run of $A$ on $w$, where $\theta_1(k) = \theta_2(k) = \theta_3(k) = \theta_4(k) = \ell$ and $\theta_5(k) = \ell + 1$ and $\theta_2(k - 1) = \ell$, $\theta_3(k - 1) = n + 1$ if and only if

\[
\begin{align*}
[k, q_1, \#^{k-2}a_{\ell - 1}] & \vdash [\ell, q_1, \#^{k-2}a_\ell] \\
[k, q_1, \#^{k-2}a_\ell] & \vdash [\ell, q_1, q_4, \#^{k-2}a_\ell] \\
[k, (q_1, q_4), \#^{k-2}a_\ell] & \vdash [\ell, (q_1, q_4), \#^{k-2}a_\ell] \\
[k, (q_1, q_4), \#^{k-2}a_\ell] & \vdash [\ell, (q_2, q_4), \#^{k-2}a_\ell] \\
[k, (q_4, \#^{k-2}a_\ell)] & \vdash [\ell + 1, q_5, \#^{k-2}a_\ell]
\end{align*}
\]

and the configuration $[\ell, (q_2, q_4), \#^{k-2}a_\ell]$ leads to acceptance.

Now, the equivalence between $L(A)$ and $L(A')$ follows immediately.
D Equivalence between alternating and deterministic one-way weak \( k \)-PA

For every one-way alternating weak \( k \)-PA, we will construct its equivalent one-way deterministic weak \( k \)-PA. This is done in two steps.

1. First, we transform the one-way alternating weak \( k \)-PA into its equivalent one-way nondeterministic weak \( k \)-PA.

2. Then, we transform the one-way nondeterministic weak \( k \)-PA into its equivalent one-way deterministic weak \( k \)-PA.

D.1 From alternating to nondeterministic

Let \( A = \langle \Sigma, Q, q_0, \mu, F, U \rangle \) be one-way alternating weak \( k \)-PA. Adding some extra states, we can normalize \( A \) as follows.

- For every \( p \in U \), if \((i, \sigma, V, p) \rightarrow (q, \text{act}) \in \mu\), then \( \text{act} = \text{stay} \).
- Every pebble can be lifted only after it reads the right-end marker \( \triangleright \).
- Only pebble \( k \) can enter a final state and it does so only after it reads the right-end marker \( \triangleright \).

We assume that \( Q \) is partitioned into \( Q_1 \cup \cdots \cup Q_k \) where \( Q_i \) is the set of states, where pebble \( i \) is the head pebble, for each \( i = 1, \ldots, k \). We can further partition each \( Q_i \) into four sets of states: \( Q_i, \text{stay} \), \( Q_i, \text{right} \), \( Q_i, \text{place} \), \( Q_i, \text{lift} \) such that for every \( i \), \( \sigma \), \( V \), \( q \) and \( p \),

- if \( q \in Q_i, \text{stay} \) and \((i, \sigma, V, q) \rightarrow (p, \text{act}) \in \mu\), then \( \text{act} = \text{stay} \);
- if \( q \in Q_i, \text{right} \) and \((i, \sigma, V, q) \rightarrow (p, \text{act}) \in \mu\), then \( \text{act} = \text{right} \);
- if \( q \in Q_i, \text{place} \) and \((i, \sigma, V, q) \rightarrow (p, \text{act}) \in \mu\), then \( \text{act} = \text{place-pebble} \);
- if \( q \in Q_i, \text{lift} \) and \((i, \sigma, V, q) \rightarrow (p, \text{act}) \in \mu\), then \( \text{act} = \text{lift-pebble} \).

Now we define a nondeterministic weak \( k \)-PA \( A' = \langle \Sigma, Q', q_0', \mu', F' \rangle \) where \( Q' = 2^Q \), \( q_0' = \{ q_0 \} \) and \( F' = 2^F \).

The set \( \mu' \) contains the following transitions. For every \( i = 1, 2, \ldots, k \), for every \( V \subseteq \{ i+1, \ldots, k \} \), for every \( S \in Q' \), for every \( \sigma \in \Sigma \),
• if $S$ contains a state $q \in U$, then
  
  $\left((i, \sigma, V, S) \rightarrow \left(((S - \{q\}) \cup S_q), \text{stay}\right) \in \mu \right)^*$

  where $S_q = \{ p \mid (i, \sigma, V, q) \rightarrow (p, \text{stay}) \in \mu \}$;

• if $S$ contains a state $q \in Q_i, \text{stay}$ and $S \cap U = \emptyset$, then
  
  $\left((i, \sigma, V, S) \rightarrow \left(((S - \{q\}) \cup \{p\}), \text{stay}\right) \in \mu \right)^*$

  where $(i, \sigma, V, q) \rightarrow (p, \text{stay}) \in \mu$;

• if $S$ contains a state $q \in Q_i, \text{place}$ and $S \cap Q_i, \text{stay} = \emptyset$, then
  
  $\left((i, \sigma, V, S) \rightarrow \left(((S - \{q\}) \cup \{p\}), \text{place-pebble}\right) \in \mu \right)^*$

  where $(i, \sigma, V, q) \rightarrow (p, \text{place-pebble}) \in \mu$; and

• if $S \cap Q_i, \text{place} = \emptyset$ and $S \cap Q_i \subseteq Q_i, \text{right}$, then
  
  $\left((i, \sigma, V, S) \rightarrow \left(((S - \{q\}) \cup S'), \text{right}\right) \in \mu \right)^*$

  where $S' = \{ p \mid (i, \sigma, V, q) \rightarrow (p, \text{right}) \in \mu \}$ and $q \in S \cap Q_i$.

The following proposition immediately implies that $L(A) = L(A')$.

**Proposition 22** For every $w \in (\Sigma \times D)^*$ of length $n$, for every $S \subseteq Q$, for every $i = 1, \ldots, k$, for every pebble assignment $\theta$, the following Statements 1 and 2 are equivalent.

1. The initial configuration $[0, q_0, \theta_0]$ leads to acceptance and $S$ is the set of states such that

   • $[0, q_0, \theta_0] \vdash [i, p, \theta]$, for all $p \in S$;
   
   • $[0, q_0, \theta_0]$ leads to acceptance if and only if $[i, p, \theta]$ leads to acceptance, for all $p \in S$.

2. There exists an accepting run of $A'$ on $w$

   $[0, \{q_0\}, \theta_0] \vdash [i, S, \theta] \vdash [k, R, \theta_f]$ for some $R \in F'$ and pebble assignment $\theta_f$, where $\theta_f(k) = n + 1$.

The proof is by routine induction on the run of $A$ on $w$.
D.2 From nondeterministic to deterministic

We start with the simple case. We will show how to determinize nondeterministic weak 2-PA. The idea can be generalized to arbitrary number of pebbles.

Let $A = (Q, q_0, F, \mu)$ be a nondeterministic weak 2-PA. We start by normalizing the behavior of $A$ as follows.

N1. For every configuration $\gamma$ of $A$, there exists a transition in $\mu$ that applies to it.

N2. Only pebble 2 can enter a final state and it does so only after it reads the right-end marker $\triangleright$.

N3. Immediately after pebble 2 moves right, pebble 1 is placed.

N4. Pebble 1 is lifted only when it reaches the right-end marker $\triangleright$.

Such normalization can be done by adding some extra states to $A$. This normalization N4 is especially important, as it implies that nondeterminism on pebble 1 is now limited only to deciding which state to enter. There is no nondeterminism in choosing which action to take, i.e. either to lift pebble 1 or to keep on moving right.

Next, we note that immediately after pebble 1 is lifted, there can be two choices of actions for pebble 2:

- to place pebble 1 again; or
- moves pebble 2 to the right.

The following fifth normalization is supposed to handle this situation:

N5. Immediately after pebble 1 is lifted, pebble 2 moves right.

In other words, while pebble 2 is reading a specific position, pebble 1 makes exactly one pass, from the position of pebble 2 to the right end of the input, instead of making several rounds of passes by placing pebble 1 again immediately after it is lifted. Since there are only finitely many states, there can only be finitely many passes. So, the normalization N4 can be achieved by simultaneously simulating all possible passes in one pass.

With the normalization N1–N5, there is no nondeterminism in choosing which action to take for pebble 2. The same as for pebble 1, the nondeterminism for pebble 2 is now limited only in deciding which states to take. This is summed up in the following remark.
Remark 23 For each \( i = 1, 2 \), if \((i, P, V, p) \rightarrow (q_1, \text{act}_1)\) and \((i, P, V, p) \rightarrow (q_2, \text{act}_2)\), then \(\text{act}_1 = \text{act}_2\).

Now that the nondeterminism is reduced to deciding which state to enter, the determinization of \( \mathcal{A} \) becomes straightforward. Similar to the classical proof of the equivalence between nondeterministic and deterministic finite state automata, we can take the power set of the states of \( \mathcal{A} \) to deterministically simulate \( \mathcal{A} \).

Now the normalization steps N1–N5 can be performed similarly for weak \( k \)-PA \( \mathcal{A} \).

N1’. For every configuration \( \gamma \) of \( \mathcal{A} \), there exists a transition in \( \mu \) that applies to it.

N2’. Only pebble \( k \) can enter a final state and it does so only after it reads the right-end marker \( \triangleright \).

N3’. For each \( i = 2, \ldots, k \), immediately after pebble \( i \) moves right, pebble \( (i - 1) \) is placed.

N4’. For each \( i = 1, \ldots, k - 1 \), pebble \( i \) is lifted only when it reaches the right-end marker \( \triangleright \).

N5’. For each \( i = 1, \ldots, k - 1 \), Immediately after pebble \( i \) is lifted, pebble \( i + 1 \) moves right.

Such normalization results in reducing the nondeterminism to deciding which state to enter and the determinization of \( \mathcal{A} \) follows immediately.