Graph Reachability and Pebble Automata over Infinite Alphabets

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

School of Informatics
University of Edinburgh
Edinburgh, Scotland

***Abstract*—We study the graph reachability problem as a language over an infinite alphabet. Namely, we view a word of even length $a_0b_0 \cdots a_nb_n$ over an infinite alphabet as a directed graph with the symbols that appear in $a_0b_0 \cdots a_nb_n$ as the vertices and $(a_0, b_0), \ldots, (a_n, b_n)$ as the edges. We prove that for any positive integer k, k pebbles are sufficient for recognizing the existence of a path of length $2^k - 1$ from the vertex a_0 to the vertex b_n, but are not sufficient for recognizing the existence of a path of length $2^{k+1} - 2$ from the vertex a_0 to the vertex b_n. Based on this result, we establish a number of relations among some classes of languages over infinite alphabets.**

***Keywords*-Graph reachability; pebble automata; infinite alphabets**

I. 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 [2], [3], [4], [7], [8], [11] and this paper aims to contribute to the progress.

Roughly speaking, there are two approaches to studying languages over infinite alphabets: 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 languages infinite alphabets, which can also be viewed as data languages, starts with the introduction of finite-memory automata (FMA) in [7], 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,[1] whereas FMA languages are not closed under complementation.

Later in [3] 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 [2]. 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*, denoted $\text{LTL}_n^{\downarrow}(\mathtt{X}, \mathtt{U})$, see [4], in which only the future time operators are allowed: the "next" operator $\mathtt{X}$ and the "until" operator $\mathtt{U}$. It was shown that one-way alternating n register automata accept all $\text{LTL}_n^{\downarrow}(\mathtt{X}, \mathtt{U})$ languages and the emptiness problem for one way alternating one register automata is decidable. Hence, the satisfiability problem for $\text{LTL}_1^{\downarrow}(\mathtt{X}, \mathtt{U})$ is decidable as well. Adding one more register or past time operators to $\text{LTL}_1^{\downarrow}(\mathtt{X}, \mathtt{U})$ makes the satisfiability problem undecidable.

In this paper we continue the study of PA, which are finite state automata equipped with a finite number of pebbles, introduced in [8]. 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 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.

[1] Though it is unknown whether they are closed under Kleene star.

As mentioned earlier, PA languages possess a very nice logical property: closure under *all* boolean operations. Another desirable property of PA languages is, as shown in [8], that nondeterminism and two-way-ness do not add expressive power to one-way deterministic PA [8, Theorem 4.6]. In addition, the class of PA languages lies strictly in between the first-order logic and the monadic second-order logic [8, Theorems 4.1 and 4.2].

Moreover, looking at the stack discipline imposed on the placement of the pebbles, one can rightly view PA as a natural extension of first-order logic for word structure. To simulate a first-order sentence of quantifier rank k, a pebble automaton with k pebbles suffices: one pebble for each quantifier depth. (See Proposition 5.)

In this paper we study PA as a model of computation for graph reachability problem. This is done by viewing a word of even length $w = a_0b_0 \cdots a_nb_n$ over an infinite alphabet as a directed graph $G_w = (V_w, E_w)$ with the symbols that appear in $a_0b_0 \cdots a_nb_n$ as the vertices in V_w and $(a_0, b_0), \ldots, (a_n, b_n)$ as the edges in E_w. We say that w induces the graph G_w.

With regards to graph reachability, we prove that PA behave similarly to the first-order logic: For any positive integer k, k pebbles are sufficient for recognizing the existence of a path of length $2^k - 1$ from the vertex a_0 to the vertex b_n, but are not sufficient for recognizing the existence of a path of length $2^{k+1} - 2$ from the vertex a_0 to the vertex b_n.

Based on this result, we establish the following relationships among the classes of languages over infinite alphabets.

1) The strict hierarchy of the PA languages based on the number of pebbles.
2) The separation of monadic second order logic from the PA languages.
3) The separation of the one-way deterministic RA languages from the PA languages.

Some of these results settle questions left open in [8], [11].

Although, in general, the emptiness problem for PA is undecidable, we believe that our study may contribute to the technical aspect of reasoning on classes of languages with decidable property. For example, in Section V a similar technique is used to obtain definability result for Linear Temporal Logic augmented with one register freeze quantifier, a class of languages over infinite alphabets with decidable satisfaction problem.

Related work: There is an analogy between our result with the classical first-order quantifier lower bounds for directed graph (s, t)-reachability which states as follows. (s, t)-reachability of distance $\leq m$ is expressible in a first order sentence of quantifier rank k if and only if $m \leq 2^k$. See, for example, [13].

As far as we can see, this classical result does not imply our result here, as explained in the following paragraphs. In [12], we introduce a logic for directed graphs, where the domain consists of the edges only, and excludes the vertices. It is established in [12] that every first-order sentence φ in such logic can be appropriately translated into a first-order sentence ψ in logic over word structure such that $w \models \psi$ if and only if $G_w \models \varphi$. Since PA is already stronger than first-order logic (Proposition 5), the result in this paper can be seen as a "stronger" version of the classical result. Indeed, our result in this paper can be used to reestablished the classical quantifier bound for (s, t)-reachability for the logic introduced in [12]. That is, (s, t)-reachability in directed graphs is expressible in a first order sentence of quantifier rank k, where the quantification is over the edges, if and only if the distance from s to t is less than or equal to 2^k. See [12, Proposition 4, Theorem 5].

Other related results are those established in [1], [6], [10]. To the best of our knowledge, those results have no connection with the result in this paper. In [1] it is established that (s, t)-reachability in directed graph is not in monadic NP, while in [6], [10] it is established that undirected graph connectivity is not in monadic NP. All these results hold even in the presence of built-in relations. However, no lower bound on first-order quantifier rank is established. Furthermore, the logic introduced in [12], which can be seen as a first-order logic for PA as a model of computation for graphs, behaves differently from the standard logic for graphs. In the logic, where the edges are the domain, (s, t)-reachability in directed graph can be expressed in monadic NP. See [12, Proposition 3].

This paper is organized as follows. In Section II we review two models of computations for words over infinite alphabet: monadic second-order logic MSO* and pebble automata (PA). Section III can be viewed as a preliminary to Section IV, where we give an example of a language not accepted by 2-PA. Section IV is the core of the paper in which we present all our results. Finally, in Section V we discuss how to adjust our results/proofs presented in Section IV to a weaker version of PA: the so called weak PA and its connection with the logic $\mathrm{LTL}_1^{\downarrow}(\mathtt{X}, \mathtt{U})$. Most of the details omitted in this paper will be presented in the journal version.

II. Models of computations

In Section II-A we recall the definition of pebble automata from [8], and in Section II-B we recall the logical framework for languages over infinite alphabets.

We will use the following notation: $\mathfrak{D}$ is a fixed infinite alphabet not containing 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 $w \in \mathfrak{D}^*$. Symbols of $\mathfrak{D}$ are denoted by low case letters a, b, c, etc., possibly indexed, and words over $\mathfrak{D}$ are denoted by low case letters u, v, w, etc., possibly indexed. The set of symbols occurring in w is denoted by $[w]$.

A. Pebble automata

Definition 1: *(See [8, Definition 2.3]) A* two-way nondeterministic k-pebble automaton*, (in short k-PA) is a system $\mathcal{A} = \langle Q, q_0, F, \mu \rangle$ 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*
- *μ is a finite set of transitions of the form $\alpha \to \beta$ such that*
  - *α is of the form (i, a, P, V, q) or (i, P, V, q), where $i \in \{1, \ldots, k\}$, $a \in \mathfrak{D} \cup \{\triangleleft, \triangleright\}$, $P, V \subseteq \{i+1, \ldots, k\}$, and*
  - *β is of the form (q, act), where $q \in Q$ and*

$$\mathsf{act} \in \{\mathsf{left}, \mathsf{right}, \mathsf{place\text{-}pebble}, \mathsf{lift\text{-}pebble}\}.$$

A symbol $a \in \mathfrak{D}$ which occurs in a transition $(i, a, P, V, p) \to \beta \in \mu$ is called a *constant* symbol. We denote by $\Theta_{\mathcal{A}}$ the set of all constant symbols of $\mathcal{A}$. Throughout this paper, we will concern only with automata without constant symbols.

Remark 2: *Here we define PA as a model of computation for languages over infinite alphabet. Another option is to define PA as a model of computation for data words. A data word is a finite sequence of $\Sigma \times \mathfrak{D}$, where Σ is a finite alphabet of labels. There is only a slight technical difference between the two models. Every data word can be viewed as a word over infinite alphabet in which every odd position contains a constant symbol. In the context of our paper, we ignore the finite labels, thus, Definition 1 is more convenient.*

Given a word $w = a_1 \cdots a_n \in \mathfrak{D}^*$, 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 $\theta : \{i, i+1, \ldots, k\} \to \{0, 1, \ldots, n, n+1\}$. The function θ defines the position of the pebbles and is called the *pebble assignment*. The symbols in the positions 0 and $n+1$ are $\triangleleft$ and $\triangleright$, respectively.

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

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

A transition $(i, P, V, q) \to \beta$ *applies to a configuration* $[j, q, \theta]$, if conditions (1)–(3) above hold and no transition of μ of the form $(i, a, P, V, q) \to \beta$ applies to $[j, q, \theta]$.

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

- if $\mathsf{act} = \mathsf{left}$, then $i' = i$ and $\theta'(i) = \theta(i) - 1$,
- if $\mathsf{act} = \mathsf{right}$, then $i' = i$ and $\theta'(i) = \theta(i) + 1$,
- if $\mathsf{act} = \mathsf{lift\text{-}pebble}$, then $i' = i + 1$,
- if $\mathsf{act} = \mathsf{place\text{-}pebble}$, then $i' = i - 1$, $\theta'(i-1) = 0$ and $\theta'(i) = \theta(i)$.

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 $\mathcal{A}$. For $1 \leq i \leq k$, an i-configuration is a configuration of the form $[i, q, \theta]$, i.e., a configuration in which the head pebble is pebble i.

Remark 3: *Note the pebble numbering that differs from that in [8]. In the above definition we adopt the pebble numbering from [5] 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. This approach will be convenient in proofs by induction on the number of pebbles already placed on the input word.*

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 word $w \in \mathfrak{D}^*$ 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. If $\mathsf{act} \in \{\mathsf{right}, \mathsf{lift\text{-}pebble}, \mathsf{place\text{-}pebble}\}$ for all transitions, then the automaton is *one-way*. It turns out that PA languages are quite robust. Namely, alternation and two-way-ness do not increase the expressive power, see Theorem 4 below.

Theorem 4: For each $k \geq 1$, two-way alternating PA and one-way deterministic PA have the same recognition power.

The proof of this theorem 4 will be published in some other venue. It is actually a stronger version of [8, Theorem 4.6] which states that two-way nondeterministic PA and one-way deterministic PA have the same recognition power. In view of this equivalence, we will always assume that pebble automata under consideration are deterministic and one-way.

Next, we define the hierarchy of languages accepted by PA. For $k \geq 1$, PA_k is the set of all languages accepted by

k-PA, and PA is the set of all PA languages. That is,

$$\text{PA} = \bigcup_{k \geq 1} \text{PA}_k.$$

B. Logic

Formally, a word $w = a_1 \cdots a_n$ is represented by the logical structure with domain $\{1, \ldots, n\}$; the natural ordering $<$ on the domain with its induced successor $+1$; and the equivalence relation $\sim$ on the domain $\{1, \ldots, n\}$, where $i \sim j$ whenever $a_i = a_j$. Two words u and v are isomorphic if they are isomorphic as word structures.

The atomic formulas in this logic is of the form $x < y$, $y = x + 1$, $x \sim y$. The first-order logic FO* is obtained by closing the atomic formulas under the propositional connectives and first-order quantification over $\{1, \ldots, n\}$. The second-order formulas MSO* are obtained by adding quantification over unary predicates on $\{1, \ldots, n\}$. A sentence φ defines the set of words

$$L(\varphi) = \{w : w \models \varphi\}.$$

If $L = L(\varphi)$ for some sentence φ, then we say that the sentence φ *expresses* the language L.

We use the same notation FO* and MSO* to denote the languages expressible by sentences in FO* and MSO*, respectively. That is,

$$\text{FO}^* = \{L(\varphi) : \varphi \text{ is an FO}^* \text{ sentence}\}$$

and

$$\text{MSO}^* = \{L(\varphi) : \varphi \text{ is an MSO}^* \text{ sentence}\}.$$

Proposition 5: (See [8, Theorem 4.1]) FO$^* \subsetneq$ PA. More specifically, let $\varphi \in$ FO* with quantifier rank k. Then, $L(\varphi) \in \text{PA}_k$.

Proof: First, we note that PA$_k$ is closed under boolean operations. Let $\varphi = Qx_k\psi(x_k)$ where $Q \in \{\forall, \exists\}$ and $\psi(x_k)$ is of quantifier rank $k-1$.

The proof is by straightforward induction on k. A k-PA $\mathcal{A}$ iterates pebble k through all possible positions in the input word w. On each iteration, the automaton $\mathcal{A}$ recursively calls a $(k-1)$-PA $\mathcal{A}'$ that accepts the language $L(\psi(x_k))$, treating the position of pebble k as the assignment value for x_k.

- If $Q = \forall$, then $\mathcal{A}$ accepts w if and only if $\mathcal{A}'$ accepts on all iterations.
- If $Q = \exists$, then $\mathcal{A}$ accepts w if and only if $\mathcal{A}'$ accepts on at least one iteration.

That the inclusion is strict is straightforward and has been proved in [8, Theorem 4.1]. ∎

We end this section with Theorem 6 below which states that all languages accepted by pebble automaton can be expressed by an MSO* sentence.

Theorem 6: *([8, Theorem 4.2])* PA $\subseteq$ MSO*.

III. A language not accepted by 2-PA

This section can be seen as a preliminary for the next section. We consider a language which is not accepted by 2-PA. Though seemingly unrelated, the idea in the proof in this section will be the core idea of the proof of our main result in the next section.

Let $L_{\text{diff}} = \{a_1 \cdots a_n \mid a_i \neq a_j \text{ whenever } i \neq j\}$. Consider the language L_3 which consists of the words of the form:

$$w = \underbrace{\cdots}_{u_1} a \underbrace{\cdots}_{u_2} a \underbrace{\cdots}_{u_3} b \underbrace{\cdots}_{u_4} b \underbrace{\cdots}_{u_5} c \underbrace{\cdots}_{u_6} c \underbrace{\cdots}_{u_7}$$

where

- a, b, c are not constants;
- $a \neq b$, $b \neq c$, $a \neq c$;
- all symbols in w, except a, b, c, appears only once;
- a appears exactly twice; and
- b and c appears exactly three times.

Or, more intuitively, $w \in L_3$ if

- there exists exactly three different symbols in w that appear more than once, that is, the symbols a, b and c;
- the first symbol that appears more than once in w, the symbol a, appears exactly twice;
- the other symbols that appear more than once in w, the symbols b and c, each appears exactly three times; and
- there exist two appearances of b and two appearances of c such that both appearances of a precede the two appearances of b, which in turn, precede the two appearances of c.

We will prove that L_3 is not accepted by 2-PA. Suppose L_3 is accepted by 2-PA $\mathcal{A} = \langle Q, q_0, F, \mu \rangle$. By Theorem 4, we may assume that $\mathcal{A}$ is one-way and deterministic. We may also assume that $\mathcal{A}$ is normalized as follows:

- the states Q is partitioned into disjoint sets Q_1 and Q_2 such that Q_1 is the set of states when pebble 1 is the head pebble and Q_2 is the set of states when pebble 2 is the head pebble;
- after each move of pebble 2, $\mathcal{A}$ places pebble 1;
- pebble 1 is lifted only when it reaches the right-end marker $\triangleright$; and
- immediately after pebble 1 is lifted, pebble 2 moves right.

Such normalization is pretty straightforward and we omit it here.

Consider the following two words:

$$v = \underbrace{\cdots}_{u_1} a \underbrace{\cdots}_{u_2} c \underbrace{\cdots}_{u_3} a \underbrace{\cdots}_{u_4} b \underbrace{\cdots}_{u_5} b \underbrace{\cdots}_{u_6} c \underbrace{\cdots}_{u_7} c \underbrace{\cdots}_{u_8} b \underbrace{\cdots}_{u_9}$$

$$w = \underbrace{\cdots}_{u_1} a \underbrace{\cdots}_{u_2} c \underbrace{\cdots}_{u_3} a \underbrace{\cdots}_{u_4} b \underbrace{\cdots}_{u_5} c \underbrace{\cdots}_{u_6} b \underbrace{\cdots}_{u_7} c \underbrace{\cdots}_{u_8} b \underbrace{\cdots}_{u_9}$$

where for each $i = 1, \ldots, 9$, the string $u_i \in L_{\text{diff}}$ and the symbols a, b, c do not appear in u_i. Recall that L_{diff} is a language consists of words in which each symbol appears at most once. Moreover, $[u_i] \cap [u_j] = \emptyset$ whenever $i \neq j$. Thus, $v \in L_3$ but $w \notin L_3$.

Our claim that L_3 is not accepted by the automaton $\mathcal{A}$ follows directly from the following claim.

Claim: There exist appropriate lengths for each $u_1, \ldots, u_9 \in L_{\text{diff}}$ such that $\mathcal{A}$ finishes the computation on both v and w in the same state.

Proof: For convenience, we write the words v and w as

$$v = \underbrace{\cdots}_{u_1} a \underbrace{\cdots}_{u_2} c \underbrace{\cdots}_{u_3} a' \underbrace{\cdots}_{u_4} b \underbrace{\cdots}_{u_5} b' \underbrace{\cdots}_{u_6} c' \underbrace{\cdots}_{u_7} c'' \underbrace{\cdots}_{u_8} b'' \underbrace{\cdots}_{u_9}$$

$$w = \underbrace{\cdots}_{u_1} a \underbrace{\cdots}_{u_2} c \underbrace{\cdots}_{u_3} a' \underbrace{\cdots}_{u_4} b \underbrace{\cdots}_{u_5} c' \underbrace{\cdots}_{u_6} b' \underbrace{\cdots}_{u_7} c'' \underbrace{\cdots}_{u_8} b'' \underbrace{\cdots}_{u_9}$$

where $a = a'$, $b = b' = b''$ and $c = c' = c''$. We use the primes just to differentiate the first, the second and the third appearances of the symbols a, b, c.

To prove our lemma, we will fix each $u_1, \ldots, u_9$ long enough so that

1) Pebble 2 arrives and leaves the symbol c in the same state in both v and w.
2) Pebble 2 arrives and leaves the symbol b in the same state in both v and w.
3) Pebble 2 arrives and leaves the symbol b' in v in the same state as it arrives and leaves the symbol c' in w.
4) Pebble 2 arrives and leaves the symbol c' in v in the same state as it arrives and leaves the symbol b' in w.
5) Pebble 2 arrives and leaves the symbol c'' in the same state in both v and w.
6) Pebble 2 arrives and leaves the symbol b'' in the same state in both v and w.

Those lengths can be obtained by exploiting the "regular" behavior of pebble 1 with respect to the symbol read by pebble 2. That is, pebble 1 will enter into a "periodic" of states if each of the words $u_1, \ldots, u_9$ are long enough, and thus, is not able to discern that b' and c' have swapped places.

Let $G = (V, E)$ be a (directed) subgraph representation of the subautomaton of $\mathcal{A}$ where

- $V = Q_1$,
- $E = \{(q, q') \mid (1, \emptyset, \emptyset, q) \to (q', \mathsf{right}) \in \mu\}$.

For a cycle ϕ in the graph G, we denote by $\ell(\phi)$ to be the length of ϕ. We fix

$$N = \prod_{\phi \text{ is a cycle in } G} \ell(\phi)$$

and $|u_1| = \cdots = |u_9| = |Q| \cdot N - 1$. (The purpose of the factor $|Q|$ is only to ensure that the length is greater than the number of states.) Recall that u_i and u_j do not share the same symbol whenever $i \neq j$.

To proceed we need the following notation. Let w' be a non-empty prefix of w. We denote by $R(w', w)$ the state in the run of $\mathcal{A}$ on w in which pebble 2 leaves the rightmost symbol of w'. We will prove the following

$$\begin{aligned}
R(u_1 a u_2 c, v) &= R(u_1 a u_2 c, w) && (1)\\
R(u_1 \cdots u_4 b, v) &= R(u_1 \cdots u_4 b, w) && (2)\\
R(u_1 \cdots u_5 b', v) &= R(u_1 \cdots u_5 c', w) && (3)\\
R(u_1 \cdots u_6 c', v) &= R(u_1 \cdots u_6 b', w) && (4)\\
R(u_1 \cdots u_7 c'', v) &= R(u_1 \cdots u_7 c'', w) && (5)\\
R(u_1 \cdots u_8 b'', v) &= R(u_1 \cdots u_8 b'', w) && (6)
\end{aligned}$$

Proof of (1): Since the symbols in u_1 do not appear anywhere else, pebble 2 arrives in a in the same state in both v and w. Since the positions of a and a' are the same in both v and w, pebble 2 leaves a in the same state in both v and w. Again, since the symbols in u_2 do not appear anywhere else pebble 2 arrives in c in the same state in both v and w.

As the length of each u_i's is longer than $|Q|$, pebble 1 will loop in a cycle ϕ before it arrives in c'. Pebble 1 arrives in c' in the same state in both v and w if and only if

$$4 + \sum_{i=2}^{6} |u_i| \equiv 3 + \sum_{i=2}^{5} |u_i| \pmod{\ell(\phi)}$$

if and only if $|u_6| + 1 \equiv 0 \pmod{\ell(\phi)}$. Since $|u_6| = |Q| \cdot N - 1$, it follows that pebble 1 arrives in c' in the same state in both v and w.

Similarly, pebble 1 arrives in c'' in both v and w in the same state if and only if

$$|u_7| \equiv |u_6| + |u_7| + 1 \pmod{\ell(\phi)}$$

for some cycle ϕ in G. It immediately follows that pebble 1 arrives at the end of both v and w in the same state, thus, pebble 2 leaves the symbol c in both v and w in the same state.

Proof of (2): It is very similar to the proof of (1). The symbols in $u_2 a u_3 a u_4$ do not appear anywhere else. So, pebble 2 arrives in b in the same state in both v and w. Since $|u_6| + 1 \equiv 0 \pmod{\ell(\phi)}$ for all cycle ϕ in G, it immediately follows that $R(u_1 \cdots u_4 b, v) = R(u_1 \cdots u_4 b, w)$.

Proof of (3): The proof is very similar to the proof of (2). The symbols in u_5 do not appear anywhere else. So, pebble 2 arrives in b' in v in the same state as pebble 2 arrives in c' in w. As

$$\begin{aligned}
|u_2| + |u_3| + |u_4| + 3 &\equiv 0 \pmod{\ell(\phi)}\\
|u_8| + 1 &\equiv 0 \pmod{\ell(\phi)}
\end{aligned}$$

for all cycle ϕ in G, it follows that $R(u_1 \cdots u_5 b', v) = R(u_1 \cdots u_5 c', w)$.

Proof of (4): The proof is very similar to the proof of (3). The symbols in u_6 do not appear anywhere else. So, pebble 2 arrives in c' in v in the same state as pebble 2 arrives in b' in w. As

$$\begin{aligned} |u_2| + |u_3| + |u_4| + 3 &\equiv 0 \pmod{\ell(\phi)} \\ |u_8| + 1 &\equiv 0 \pmod{\ell(\phi)} \end{aligned}$$

for all cycle ϕ in G, it follows that $R(u_1 \cdots u_6 c', v) = R(u_1 \cdots u_6 b', w)$.

Proof of (5): The proof is very similar to the proof of (4). The symbols in u_7 do not appear anywhere else. So, pebble 2 arrives in c'' in the same state in both v and w. As

$$|u_6| + 1 \equiv 0 \pmod{\ell(\phi)}$$

for all cycle ϕ in G, it follows that $R(u_1 \cdots u_7 c'', v) = R(u_1 \cdots u_7 c'', w)$.

Proof of (6): The proof is similar to that of (5). The symbols in u_8 do not appear anywhere else. So, pebble 2 arrives in b'' in the same state in both v and w. As

$$|u_6| + 1 \equiv 0 \pmod{\ell(\phi)}$$

for all cycle ϕ in G, it follows that $R(u_1 \cdots u_8 b'', v) = R(u_1 \cdots u_8 b'', w)$. ■

Remark 7: *The reader can easily identify the cycles in the graph G as the "regular" behavior of 2-PA. Indeed, such regularity is the crucial feature of PA that we are going to exploit in the proof of our main result in the next section.*

IV. Words of $\mathfrak{D}^*$ as Graphs

Let $w = a_0 b_0 \cdots a_n b_n \in \mathfrak{D}^*$ be a word of even length. The word w *induces* a directed graph $G_w = (V_w, E_w)$, whose set of vertices is $V_w = \{a_0, b_0, \ldots, a_n, b_n\}$, that is, the set of symbols that appear in w, and the set of edges is $E_w = \{(a_0, b_0), \ldots, (a_n, b_n)\}$. With such view, we will use the term *graph* when referring to an even length word. We also write $s_w = a_0$ and $t_w = b_n$ to denote the first and the last symbol in w, respectively. For convenience, we consider only the words w in which s_w and t_w occur only once.

We need the following basic graph terminology. Let a and b be vertices in a graph G. A *path* of length m from a to b is a sequence of m edges in G: $(a_{i_1}, b_{i_1}), \ldots, (a_{i_m}, b_{i_m})$ such that $a_{i_1} = a$, $b_{i_m} = b$ and for each $j = 1, \ldots, m-1$, $b_{i_j} = a_{i_{j+1}}$. The *distance* from a to b is the length of a minimal path from a to b in G. If there is no path from a to b in G, then we set the distance be ∞.

We define the following reachability languages. For $m \geq 1$,

$$\mathcal{R}_m = \{w : \text{the distance from } s_w \text{ to } t_w \text{ in } G_w \text{ is } \leq m\}$$

and

$$\mathcal{R} = \bigcup_{m=1,2,\ldots} \mathcal{R}_m$$

Remark 8: *When processing an input word w, an automaton $\mathcal{A}$ can remember by its state whether its head pebble is currently at an odd-number or even-number position in w. (We count that the leftmost symbol of w is in position 1.) Therefore, we will always assume that the input word w is of even length, which also naturally fits the subject of this appear. In addition, unless indicated otherwise, we always denote an input word w by $a_0 b_0 \cdots a_n b_n$. That is, the odd-position symbols are denoted by a_i's and the even-position symbols by b_i's.*

Again, we remind the reader that in k-PA the pebbles placed on the input word are numbered from k to i, see Remark 3.

Proposition 9: For each $k = 1, 2, \ldots$, $\mathcal{R}_{2^k-1} \in \text{PA}_k$.

Proof: The proof of this proposition is the standard implementation of Savitch's algorithm for (s-t)-reachability [9]. It is by induction on k. For the basis, we prove that $\mathcal{R}_3 \in \text{PA}_2$. On input word $w = a_0 b_0 \cdots a_n b_n$, a 2-PA $\mathcal{A}_2$ performs the following.

- Pebble 2 iterates through all a_i.
- On each iteration, the automaton uses pebbles 1 to check whether $b_0 = a_i$.
- If $b_0 = a_i$, the automaton moves pebble 2 to the right to read b_i, and check whether $b_i = a_n$.

It is obvious that this automaton accepts $\mathcal{R}_3$.

Now, we prove the induction step. On input word $w = a_0 b_0 \cdots a_n b_n$, a k-PA $\mathcal{A}_k$ performs the following.

- Pebble k iterates through all a_i.
- On each iteration, the automaton recursively uses pebbles $(k-1), \ldots, 1$ to check whether there is a path from a_0 to a_i. That is, $\mathcal{A}_k$ makes a recursive call of $\mathcal{A}_{k-1}$.
- If such path exists, the automaton lifts all pebbles $1, \ldots, (k-1)$; moves pebble k to the right to read b_i, and recursively uses pebbles $(k-1), \ldots, 1$ to check whether there is a path from b_i to b_n. That is, $\mathcal{A}_k$ makes a recursive call of $\mathcal{A}_{k-1}$ one more time.

$\mathcal{A}_k$ accepts w if there is an index i such that there is a path from a_0 to a_i and a path from b_i to b_n. By straightforward induction, $\mathcal{A}_k$ accepts $\mathcal{R}_{2^k-1}$, and thus, $\mathcal{R}_{2^k-1} \in \text{PA}_k$. ■

Lemma 10 below is the backbone of most of the results presented in this paper. Its proof is quite long. Therefore, we only give an intuitive sketch. The complete proof will be presented in the journal version of this paper. Let $n_k = 2^{k+1} - 2$, for each $k = 1, 2, \ldots$.

Lemma 10: Let $\mathcal{A}$ be a k-pebble automaton such that $L(\mathcal{A}) \subseteq \mathcal{R}$. Then, $\mathcal{R}_{n_k} \not\subseteq L(\mathcal{A})$.

Intuitively, the proof is as follows. We will show that for every k-PA $\mathcal{A}$, there exists $w \in \mathcal{R}_{n_k} - \mathcal{R}_{n_k-1}$ such that

$\mathcal{A}$ cannot recognize a path from s_w to t_w. The proof is by induction on k. The basis, $k = 1$, is relatively trivial to demonstrate as 1 pebble is not even capable of comparing data values.

Let $\mathcal{A}$ be a k-PA and let $\mathcal{A}_{k-1}$ be $(k-1)$-PA subautomaton of $\mathcal{A}$. By the induction hypothesis, there exists $w \in \mathcal{R}_{n_{(k-1)}} - \mathcal{R}_{n_{(k-1)}-1}$ such that $\mathcal{A}_{k-1}$ cannot recognize a path from s_w to t_w.

The induction step is as follows. Let w' be a word isomorphic to w such that $[w] \cap [w'] = \emptyset$. Recall that $[w]$ and $[w']$ denote the set of symbols occurring in w and w', respectively.

We define the word u such that

$$u = w \underbrace{\cdots}_{v_1} a_1 a_2 \underbrace{\cdots}_{v_2} a_2 a_3 \underbrace{\cdots}_{v_3} w';$$

where

- $a_1 = t_w$;
- $a_3 = s_{w'}$;
- $a_2, a_3 \notin [w]$;
- $a_1, a_2 \notin [w']$;
- v_1, v_2, v_3 are some strings depending on the number of states of $\mathcal{A}$.

It is obvious that $u \in \mathcal{R}_{n_k} - \mathcal{R}_{n_k-1}$. We claim that $\mathcal{A}$ cannot recognize a path from s_u to t_u. The proof is roughly as follows.

- When pebble k is above $wv_1a_1a_2$, by the induction hypothesis, the subautomaton $\mathcal{A}_{k-1}$ cannot recognize the path from $s_{w'}$ to $t_{w'}$, thus, the path from s_u to t_u.
- When pebble k is above $v_2a_2a_3v_3w'$, by the induction hypothesis, the subautomaton $\mathcal{A}_{k-1}$ cannot recognize the path from s_w to t_w, thus, the path from s_u to t_u.

Therefore, the automaton cannot recognize the path from s_u to t_u. We want to notify the reader here that the stack discipline imposed on the placement of the pebbles is crucial in the application of the induction hypothesis. Pebble k is fixed on its current position once we start using the other $(k-1)$ pebbles, and cannot be moved unless all the other $(k-1)$ pebbles are lifted.

Furthermore, the construction of the strings v_1, v_2, v_3 follows essentially the same idea of the construction of the strings $u_1, \ldots, u_9$ in Section III. That is, for long enough v_1, v_2, v_3, pebble k enters into a periodic of states while scanning v_1, v_2, v_3. Such periodic of states intuitively means that pebble k is "confused," not being able to differentiate whether it is above the symbols a_1, a_2, a_3, or the strings v_1, v_2, v_3.

Corollary 11: *$\mathcal{R}_{n_k} \notin PA_k$.*

Proof: The corollary immediately follows from the lemma, because $\mathcal{R}_k \subseteq \mathcal{R}$. ∎

Corollary 12: *$\mathcal{R} \notin PA$.*

Proof: Assume to the contrary that $\mathcal{R} = L(\mathcal{A})$ for a k-PA $\mathcal{A}$. Then, $\mathcal{R}_{n_k} \subseteq L(\mathcal{A})$, which contradicts Lemma 10. ∎

The following theorem establishes the proper hierarchy of the PA languages.

Theorem 13: For each $k = 1, 2, \ldots$, $\mathrm{PA}_k \subsetneq \mathrm{PA}_{k+1}$.

Proof: By Proposition 9, $\mathcal{R}_{2^{k+1}-1} \in \mathrm{PA}_{k+1}$. Suppose $\mathcal{R}_{2^{k+1}-1} \in \mathrm{PA}_k$. Let $\mathcal{A}$ be a k-PA that accepts $\mathcal{R}_{2^{k+1}-1}$. Then, $L(\mathcal{A}) \subseteq \mathcal{R}$ and by Lemma 10, $\mathcal{R}_{2^{k+1}-2} \not\subseteq L(\mathcal{A})$. But this is a contradiction since $L(\mathcal{A}) = \mathcal{R}_{2^{k+1}-1} \supseteq \mathcal{R}_{2^{k+1}-2}$. ∎

Another consequence of Corollary 12 is that the inclusion of PA in MSO* provided by Theorem 6 is proper.

Theorem 14: $\mathrm{PA} \subsetneq \mathrm{MSO}^*$.

Proof: Without loss of generality, we may assume that MSO* contains two constant symbols, min and max, which denote minimum and the maximum elements of the domain, respectively. For a word $w = a_1 \cdots a_n$, the minimum and the maximum elements are 1 and n, respectively, and not 0 and $n+1$ which are reserved for the end-markers $\triangleleft$ and $\triangleright$.

Then the language $\mathcal{R}$ can be expressed in MSO* as follows. There exist unary predicates S_{odd} and P such that

- S_{odd} is the set of all odd elements in the domain where $\mathsf{min} \in S_{odd}$ and $\mathsf{max} \notin S_{odd}$.
- The predicate P satisfies the conjunction of the following FO* sentences:
  - $P \subseteq S_{odd}$ and $\mathsf{min} \in P$ and $\mathsf{max} - 1 \in P$,
  - for all $x \in P - \{\mathsf{max} - 1\}$, there exists exactly one $y \in P$ such that $x + 1 \sim y$, and
  - for all $x \in P - \{\mathsf{min}\}$, there exists exactly one $y \in P$ such that $y + 1 \sim x$.

Now, the theorem follows from Corollary 12. ∎

Remark 15: *Combining Theorems 4 and 14, we obtain that MSO* is stronger than two-way alternating PA. This settles a question left open in [8] whether MSO* is strictly stronger than two-way alternating PA.*

Next, we define a restricted version of the reachability languages. Recall that $[w]$ denotes the set of symbols occurring in w.

For a positive integer $m \geq 1$, the language $\mathcal{R}^+_m$ consists of all words of the form

$$c_0 c_1 \underbrace{\cdots}_{u_1} c_1 c_2 \underbrace{\cdots}_{u_2} c_2 c_3 \cdots\cdots\cdots\cdots \hookleftarrow$$
$$\hookrightarrow \cdots\cdots\cdots c_{m-3} c_{m-2} \underbrace{\cdots}_{u_{m-2}} c_{m-2} c_{m-1} \underbrace{\cdots}_{u_{m-1}} c_{m-1} c_m$$

where

- $c_0, \ldots, c_m \in \mathfrak{D}$, and $c_i \neq c_{i+1}$, for all $i = 0, \ldots, m-1$;
- $u_1, \ldots, u_{m-1} \in \mathfrak{D}^*$;
- for each $i = 1, \ldots, m-1$, $c_i \notin [u_i]$.

The language $\mathcal{R}^+$ is defined as

$$\mathcal{R}^+ = \bigcup_{m=1,2,\ldots} \mathcal{R}^+_m.$$

Remark 16: *Actually, in the proof of Lemma 10 we show that for every k-PA $\mathcal{A}$, there exist graphs $G \in \mathcal{R}^+_{n_k}$ and $G' \notin \mathcal{R}^+$ such that either $\mathcal{A}$ accepts both G and G', or rejects both of them are not. Therefore, $\mathcal{R}^+ \notin$ PA.*

The following theorem answers a question left open in [8], [11]: Can one-way deterministic FMA be simulated by pebble automata? (We refer the reader to [7, Definition 1] for the formal definition of FMA.)

Theorem 17: The language $\mathcal{R}^+$ is accepted by one-way deterministic FMA, but is not accepted by pebble automata.

Proof: Note that $\mathcal{R}^+$ is accepted by a one-way two register deterministic FMA. On input word $w = a_0 b_0 \cdots a_n b_n$, the automaton stores b_0 in the first register and then moves right (using the second register to scan the input symbols) until it finds a vertex $a_{i_1} = b_0$. If it finds one, then it stores b_{i_1} in register 1 and moves right again until it finds another vertex $a_{i_2} = b_{i_1}$. It repeats the process until either of the following holds.

- a_n is the same as the content of the register 1, or,
- it cannot find a vertex currently stored in the first register.

In the former case, the automaton accepts the input graph, and in the latter case does not.

However, by Remark 16, $\mathcal{R}^+$ is not a PA language. ■

V. Weak PA and $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$

There is an analogue of our results to another, but weaker, version of pebble automata. In the model defined in Section II, the new pebble is placed in the beginning of the input word. An alternative would be to place the new pebble at the position of the most recent one. The model defined this way is usually referred as *weak* PA. Formally, it is defined by setting $\theta'(i-1) = \theta(i)$ (and keeping $\theta'(i) = \theta(i)$) in the case of act = place-pebble of the definition of the transition relation in Definition 1.

Obviously, two-way weak PA is just as expressive as two-way strong PA. However, one-way deterministic weak k-PA is weaker than strong k-PA. For example, $\mathcal{R}_{2^k-1}$ is not a weak k-PA language, see Lemma 19 below. Furthermore, weak k-PA are also robust as stated in the theorem below.

Theorem 18: For each $k \geq 1$, one-way alternating, nondeterministic and deterministic weak k-PA have the same recognition power.

In view of this theorem, we will always assume that weak k-PA are one-way and deterministic. Its proof will be published in another venue.

Let

$$\text{wPA}_k = \{L : L \text{ is accepted by a weak } k\text{-PA}\}$$

and

$$\text{wPA} = \bigcup_{k \geq 1} \text{wPA}_k$$

Lemma 19: For each $k = 1, 2, \ldots$, $\mathcal{R}_k \in \text{wPA}_k$, but $\mathcal{R}_{k+1} \notin \text{wPA}_k$.

The full proof of Lemma 19 will appear in the journal version of this paper. It immediately implies the strict hierarchy for wPA languages.

Theorem 20: For each $k = 1, 2, \ldots$, $\text{wPA}_k \subsetneq \text{wPA}_{k+1}$.

A. Linear temporal logic with one register freeze quantifier

In this section we recall the definition of Linear Temporal Logic (LTL) augmented 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. Moreover, in [4] the LTL model is defined over data words. Since in this paper we essentially ignore the finite labels, the LTL model presented here also ignores the finite labels. However, the result here also holds for the data word model.

Roughly, the logic $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$ is the standard LTL augmented with a register to store a symbol from the infinite alphabet. Formally, the formulas are defined as follows.

- Both True and False belong to $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$.
- The empty formula ϵ belongs to $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$.
- If φ, ψ are in $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$, then so are $\neg\varphi$, $\varphi \vee \psi$ and $\varphi \wedge \psi$.
- $\uparrow$ is in $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$.
- If φ is in $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$, then so is $\mathtt{X}\varphi$.
- If φ is in $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$, then so is $\downarrow \varphi$.
- If φ, ψ are in $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$, then so is $\varphi \mathtt{U} \psi$.

Intuitively, the predicate $\uparrow$ is intended to mean that the current symbol is the same as the symbol in the register, while $\downarrow \varphi$ is intended to mean that the formula φ holds when the register contains the current symbol. This will be made precise in the definition of the semantics of $\text{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$ 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 φ, denoted by $\mathsf{fqr}(\varphi)$.

- $\mathsf{fqr}(\mathsf{True}) = \mathsf{fqr}(\mathsf{False}) = \mathsf{fqr}(\epsilon) = \mathsf{fqr}(\uparrow) = 0$.
- $\mathsf{fqr}(\mathtt{X}\varphi) = \mathsf{fqr}(\neg\varphi) = \mathsf{fqr}(\varphi)$, for every φ in $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$.
- $\mathsf{fqr}(\varphi \vee \psi) = \mathsf{fqr}(\varphi \wedge \psi) = \mathsf{fqr}(\varphi \mathtt{U} \psi) = \max(\mathsf{fqr}(\varphi), \mathsf{fqr}(\psi))$, for every φ and ψ in $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$.
- $\mathsf{fqr}(\downarrow \varphi) = \mathsf{fqr}(\varphi) + 1$, for every φ in $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$.

Finally, we define the semantics of $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$. Let w be a word. For a position $i = 1,\ldots,n$, a symbol a and a formula φ in $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$, $w,i \models_a \varphi$ means that φ is satisfied by w at position i when the content of the register is a. As usual, $w,i \not\models_a \varphi$ means the opposite. The satisfaction relation is defined inductively as follows.

- $w,i \models_a \epsilon$ for all $i = 1,2,\ldots,n$ and $a \in \mathfrak{D}$.
- $w,i \models_a \mathsf{True}$ and $w,i \not\models_a \mathsf{False}$, for all $i = 1,2,3,\ldots$ and $a \in \mathfrak{D}$.
- $w,i \models_a \varphi \vee \psi$ if and only if $w,i \models_a \varphi$ or $w,i \models_a \psi$.
- $w,i \models_a \varphi \wedge \psi$ if and only if $w,i \models_a \varphi$ and $w,i \models_a \psi$.
- $w,i \models_a \neg\varphi$ if and only if $w,i \not\models_a \varphi$.
- $w,i \models_a \mathtt{X}\varphi$ if and only if $1 \le i < n$ and $w,i+1 \models_a \varphi$.
- $w,i \models_a \varphi \mathtt{U} \psi$ if and only if there exists $j \ge i$ such that
  - $w,j \models_a \psi$ and
  - $w,j' \models_a \varphi$, for all $j' = i,\ldots,j-1$.
- $w,i \models_a \downarrow\varphi$ if and only if $w,i \models_{a_i} \varphi$
- $w,i \models_a \uparrow$ if and only if $a = a_i$.

For a sentence φ in $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$, we define the language $L(\varphi)$ by

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

Note that since φ is a sentence, all occurrences of $\uparrow$ in φ 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 $L(\varphi)$.

Theorem 21: For every sentence $\psi \in \mathrm{LTL}^{\downarrow}(\mathtt{X},\mathtt{U})$, there exists a weak k-PA $\mathcal{A}_\psi$, where $k = \mathsf{fqr}(\psi) + 1$, such that $L(\mathcal{A}_\psi) = L(\psi)$.

Proof: Let ψ be an $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$ sentence. We construct an alternating weak k-PA $\mathcal{A}_\psi$, where $k = \mathsf{fqr}(\psi) + 1$ such that given a word w, the automaton $\mathcal{A}_\psi$ "checks" whether $w,1 \models \psi$. $\mathcal{A}_\psi$ accepts w if it is so. Otherwise, it rejects.

Intuitively, the computation of $w,1 \models \psi$ is done recursively as follows. The automaton $\mathcal{A}_\psi$ "consists of" the automata $\mathcal{A}_\varphi$ for all sub-formula φ of ψ, including $\mathcal{A}_\epsilon$ to represent the empty formula ϵ.

- The automaton $\mathcal{A}_\epsilon$ accepts every input.
- If $\psi = \varphi \vee \varphi'$, then $\mathcal{A}_\psi$ nondeterministically chooses one of $\mathcal{A}_\varphi$ or $\mathcal{A}_{\varphi'}$ and proceeds to run one of them.
- If $\psi = \varphi \wedge \varphi'$, then $\mathcal{A}_\psi$ splits its computation (by conjunctive branching) into two and proceeds to run both $\mathcal{A}_\varphi$ and $\mathcal{A}_{\varphi'}$.
- If $\psi = \mathtt{X}\varphi$, $\mathcal{A}_\psi$ moves to the right one step. If it reads the right-end marker $\triangleright$, then it rejects immediately. Otherwise, it proceeds to run $\mathcal{A}_\varphi$.
- If $\psi = \uparrow \varphi$, then $\mathcal{A}_\psi$ checks whether the symbol 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 $\mathcal{A}_\varphi$.
- If $\psi = \downarrow \varphi$, then $\mathcal{A}_\psi$ places a new pebble and proceeds to run $\mathcal{A}_\varphi$.
- If $\psi = \varphi \mathtt{U} \varphi'$, then $\mathcal{A}_\psi$ runs $\mathcal{A}_{\varphi' \vee (\varphi \wedge \mathtt{X}(\varphi \mathtt{U} \varphi'))}$.
- If $\psi = \neg\varphi$, then $\mathcal{A}_\psi$ runs the complement of $\mathcal{A}_\varphi$. The complement of $\mathcal{A}_\varphi$ can be constructed by switching the accepting states into non-accepting states and the non-accepting states into accepting states, as well as, switching the universal states into non-universal states and the non-universal states into universal states.

Note that since $\mathsf{fqr}(\varphi) = k$, on each computation path the automaton $\mathcal{A}_\psi$ only needs to place the pebble k times, thus, $\mathcal{A}_\psi$ requires only $(k+1)$ pebbles. It is a straightforward induction to show that $L(\mathcal{A}_\psi) = L(\psi)$. ■

Our next results deals with the expressive power of $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$ 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 $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$ sentence for the language $\mathcal{R}_m^+$ defined in Section IV.

Lemma 22: For each $k = 1,2,3,\ldots$, there exists a sentence ψ_k in $\mathrm{LTL}_1^{\downarrow}(\mathtt{X},\mathtt{U})$ such that $L(\psi_k) = \mathcal{R}_k^+$ and

- $\mathsf{fqr}(\psi_1) = 1$; and
- $\mathsf{fqr}(\psi_k) = k-1$, when $k \ge 2$.

Proof: First, we define a formula φ_k such that $\mathsf{fqr}(\varphi_k) = k-1$ and for every word $w = d_1 \cdots d_n$, for every $i = 1,\ldots,n$,

$$w,i \models_{d_i} \varphi_k \quad \text{if and only if} \quad \binom{\sigma}{d_i} \cdots \binom{\sigma}{d_n} \in \mathcal{R}_k^+. \quad (7)$$

We construct φ_k inductively as follows.

- $\varphi_1 := \mathtt{X}(\neg\uparrow) \wedge \neg(\mathtt{XX})$.
- For each $k = 1,2,3,\ldots$,

$$\varphi_{k+1} \quad := \quad \mathtt{X}(\neg\uparrow) \wedge \mathtt{X}\Big(\downarrow \mathtt{X}\Big((\neg\uparrow)\mathtt{U}(\uparrow \wedge \varphi_k)\Big)\Big)$$

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

It is straightforward to show that φ_k satisfies (7). The desired sentence ψ_k is defined as follows.

- $\psi_1 := \downarrow \big(\mathtt{X}(\neg\uparrow) \wedge \neg(\mathtt{XX})\big)$.
- For each $k = 2,3,\ldots$,

$$\psi_k \quad := \quad \downarrow(\mathtt{X}(\neg\uparrow)) \wedge \mathtt{X}\Big(\downarrow \mathtt{X}\Big((\neg\uparrow)\mathtt{U}(\uparrow \wedge \varphi_{k-1})\Big)\Big)$$

Obviously, $\mathsf{fqr}(\psi_1) = 1$. For $k \geq 2$, $\mathsf{fqr}(\varphi_{k-1}) = k-2$, thus, $\mathsf{fqr}(\psi_k) = k-1$. ■

Lemma 23: For each $k = 1, 2, \ldots$, the language $\mathcal{R}^+_{k+1}$ is not expressible by a sentence in $\mathrm{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$ of freeze quantifier rank $(k-1)$.

Proof: By Lemma 19, $\mathcal{R}^+_{k+1}$ is not accepted by weak k-PA, thus, it is also not accepted by top-view k-PA. Then, by Theorem 21, $\mathcal{R}^+_{k+1}$ is not expressible by $\mathrm{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$ sentence of freeze quantifier rank $(k-1)$. ■

Combining both Lemmas 22 and 23, we obtain that for each $k = 1, 2, \ldots$, the language $\mathcal{R}_{k+1}$ separates the class of $\mathrm{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$ sentences of freeze quantifier rank k from the class of $\mathrm{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$ sentences of freeze quantifier rank $(k-1)$. Formally, we state it as follows.

Theorem 24: For each $k = 1, 2, \ldots$, the class of sentences in $\mathrm{LTL}^{\downarrow}_1(\mathtt{X}, \mathtt{U})$ of freeze quantifier rank k is strictly more expressive than those of freeze quantifier rank $(k-1)$.

ACKNOWLEDGMENT

The author would like to thank Michael Kaminski for all helpful discussions without which this paper would not exist, for pointing out that the original example of the author is, actually, the reachability problem in graphs, and for his comments on the first version of this paper.

REFERENCES

[1] M. Ajtai and R. Fagin, "Reachability Is Harder for Directed than for Undirected Finite Graphs," Journal of Symbolic Logic, vol. 55, no. 1, pp. 113–150, 1990.

[2] H. Björklund and T. Schwentick, "On Notions of Regularity for Data Languages," Proc. Symp. Fundamentals of Computation Theory (FCT 07), pp. 88–99.

[3] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin and C. David, "Two-Variable Logic on Words with Data," Proc. IEEE Symp. Logic in Computer Science (LICS 06), IEEE Computer Society, pp. 7–16.

[4] S. Demri and R. Lazic, "LTL with the Freeze Quantifier and Register Automata," Proc. IEEE Symp. Logic in Computer Science (LICS 06), IEEE Computer Society, pp. 17–26.

[5] M. Bojanczyk, M. Samuelides, T. Schwentick and L. Segoufin, "Expressive Power of Pebble Automata," Proc. Colloq. Automata, Languages and Programming (ICALP 06), LNCS 4051, pp. 157–168.

[6] R. Fagin, L. Stockmeyer and M. Vardi, "On Monadic NP vs. Monadic co-NP," Information and Computation, vol. 120, no. 1, pp. 78–92, 1995

[7] M. Kaminski and N. Francez, "Finite-Memory Automata," Theoretical Computer Science, vol. 134, no. 2, pp. 329–363, 1994.

[8] F. Neven, T. Schwentick and V. Vianu, "Finite state machines for strings over infinite alphabets," ACM Transactions on Computational Logic, vol. 5, no. 3, pp. 403–435, 2004.

[9] W. Savitch, "Relationships Between Nondeterministic and Deterministic Tape Complexities," Journal of Computer and System Sciences, vol. 4, no. 2, pp. 177–192, 1970.

[10] T. Schwentick, "On Winning Ehrenfeucht Games and Monadic NP," Annals of Pure and Applied Logic, vol. 79, no. 1, pp. 61–92, 1996.

[11] L. Segoufin, "Automata and Logics for Words and Trees over an Infinite Alphabet," Proc. EACSL Conf. Computer Science Logic (CSL 06), LNCS 4207, pp. 41–57.

[12] T. Tan, "A Logic for Graph with the Edges as the Domain," Technical Report, Department of Computer Science, Technion – Israel Institute of Technology, no. CS-2009-04, 2009.

[13] G. Turán, "On the definability of properties of finite graphs," Discrete Mathematics, vol. 49, no. 3, pp. 291–302, 1984.