Finite State Machines for Strings over Infinite Alphabets

FRANK NEVEN
Limburgs Universitair Centrum
and
THOMAS SCHWENTICK
Philipps-Universität Marburg
and
VICTOR VIANU
U.C. San Diego

Motivated by formal models recently proposed in the context of XML, we study automata and logics on strings over infinite alphabets. These are conservative extensions of classical automata and logics defining the regular languages on finite alphabets. Specifically, we consider register and pebble automata, and extensions of first-order logic and monadic second-order logic. For each type of automaton we consider one-way and two-way variants, as well as deterministic, nondeterministic, and alternating control. We investigate the expressiveness and complexity of the automata, their connection to the logics, as well as standard decision problems. Some of our results answer open questions of Kaminski and Francez on register automata.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*automata, relations between models*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*model theory*; F.4.1 [**Mathematical Logic and Formal Languages**]: Formal languages—*classes defined by grammars or automata*

General Terms: Languages, Theory, Algorithms

Additional Key Words and Phrases: automata, expressiveness, first-order logic, infinite alphabets, monadic second-order logic, pebbles, registers, XML

1. INTRODUCTION

One of the significant recent developments related to the World Wide Web (WWW) is the emergence of the Extensible Markup Language (XML) as the standard for data exchange on the Web [Abiteboul et al. 1999]. Since XML documents have a

A preliminary version with title *Towards regular languages over infinite alphabets* appeared in the proceedings of the 26th International Symposium on *Mathematical Foundations of Computer Science (MFCS 2001)*, Czech Republic, LNCS 2136, pages 560–572, 2001. Victor Vianu is supported in part by the National Science Foundation under grant number IIS-9802288. E-mail: frank.neven@luc.ac.be, tick@informatik.uni-marburg.de, vianu@cs.ucsd.edu.

tree structure (usually defined by DTDs), XML queries can be modeled as mappings from trees to trees (tree transductions) and schema languages are closely related to tree automata. Therefore automata theory has naturally emerged as a central tool in formal work on XML [Bex et al. 2002; Milo et al. 2000; Neven 2002a; Neven and Schwentick 2000; 2002; Papakonstantinou and Vianu 2001; Vianu 2001]. The connection to logic and automata proved very fruitful in understanding such languages and in the development of optimization algorithms and static analysis techniques. However, these abstractions ignore an important aspect of XML, namely the presence of *data values* attached to leaves of trees, and comparison tests performed on them by XML queries. These data values make a big difference – indeed, in some cases the difference between decidability and undecidability (e.g., see [Alon et al. 2001]). It is therefore important to extend the automata and logic formalisms to trees with data values. In this initial investigation we model data values by infinite alphabets, and consider the simpler case of strings rather than trees. Strings are also relevant in the tree case, as most formalisms allow reasoning along paths in the tree. In the case of XML, it would be more accurate to consider strings labeled by a finite alphabet and attach data values to positions in the string. However, this would render the formalism more complicated and has no bearing on the results. Although limited to strings, we believe that our results provide a useful starting point in investigating the more general problem. In particular, our lower-bound results will easily be extended to trees.

We only consider models which accept precisely the regular languages when restricted to finite alphabets. It is useful to observe that for infinite alphabets it is no longer sufficient to equip automata with states alone. Indeed, automata should at least be able to check equality of symbols. There are two main ways to do this:

(1) store a finite set of positions and allow equality tests between the symbols on these positions; and
(2) store a finite set of symbols only and allow equality tests with these symbols.

The first approach, however, leads to multi-head automata, immediately going beyond regular languages. Therefore, we instead equip automata with a finite set of *pebbles* whose use is restricted by a stack discipline. The automaton can test equality by comparing the pebbled symbols. We follow in [Milo et al. 2000] modeling the reading head of the automaton by the top pebble. In the second approach, we follow Kaminski and Francez [Kaminski and Francez 1994; 1990] and extend finite automata with a finite number of *registers*[1] that can store alphabet symbols. When processing a string, an automaton compares the symbol on the current position with values in the registers; based on this comparison it can decide to store the current symbol in some register.

In addition to automata, we consider another well-known formalism: monadic second-order logic (MSO). To be precise, we associate to strings first-order structures in the standard way, and consider the extensions of MSO and FO denoted by MSO* and FO*, as done by Grädel and Gurevich in the context of meta-finite models [Grädel and Gurevich 1998]. MSO has proven to be a good yardstick when

[1]Kaminski and Francez called these automata *finite-memory automata*. We use the term *register automata* here to emphasize the difference to pebble automata.

other generalizations of regular languages were investigated, e.g., for trees, infinite strings [Thomas 1997] and graphs [Courcelle 1990].

Our results concern the expressive power of the various models, provide lower and upper complexity bounds, and consider standard decision problems. For the above mentioned automata models we consider deterministic (D), nondeterministic (N), and alternating (A) control, as well as one-way and two-way variants. We denote these automata models by dC-X where $d \in \{1, 2\}$, $C = \{\mathrm{D}, \mathrm{N}, \mathrm{A}\}$, and $X \in \{\mathrm{RA}, \mathrm{PA}\}$. Here, 1 and 2 stand for one- and two-way, respectively, D, N, and A stand for deterministic, nondeterministic, and alternating, and PA and RA for pebble and register automata. As our main focus is on deterministic and nondeterministic automata we do not consider one-way alternating automata. Our main results are the following (the expressiveness results are also represented in Figure 1).

Registers. We pursue the investigation of register automata initiated by Kaminski and Francez [Kaminski and Francez 1994]. In particular, we investigate the connection between RAs and logic and show that they are essentially incomparable. Indeed, we show that MSO* cannot define 2D-RA. Furthermore, there are even properties in FO* that cannot be expressed by 2A-RAs. The proof of the latter is based on communication complexity [Hromkovic 2000]. Next, we consider the relationship between the various RA models. We separate 1N-RAs, 2D-RAs, 2N-RAs, and 2A-RAs, subject to standard complexity-theoretic assumptions. The separation between 1N-RAs and 2N-RAs is already proved in [Kaminski and Francez 1994].

Pebbles. We consider two kinds of PAs: one where every new pebble is placed on the first position of the string and one where every new pebble is placed on the position of the current pebble. We refer to them as strong and weak PAs, respectively. Clearly, this pebble placement only makes a difference in the case of one-way PAs. In the one-way case, strong 1D-PA can simulate FO* while weak 1N-PA cannot (whence the names). The proof of the latter separation is again based on communication complexity. Furthermore, we show that all pebble automata variants can be defined in MSO*. Finally, we provide more evidence that strong PAs are a robust notion by showing that the power of strong 1D-PA, strong 1N-PA, 2D-PA, and 2N-PA coincide.

Decision Problems. Finally, we consider decision problems for RAs and PAs, and answer several open questions from Kaminski and Francez. First, we show that universality of 1N-RAs and emptiness of 2D-RA are undecidable. Next, we obtain that emptiness is undecidable even for weak 1D-PAs.

As RAs are orthogonal to logically defined classes one might argue that PAs are better suited to define the notion of regular languages over infinite alphabets. Indeed, they are reasonably expressive as they lie between FO* and MSO*. Furthermore, strong PAs form a robust notion. Adding two-wayness and nondeterminism does not increase expressiveness and the class of definable languages is closed under Boolean operations, concatenation and Kleene star. Capturing exactly MSO* most likely requires significant extensions of PAs, as in MSO* one can express complete problems for every level of the polynomial hierarchy, while computations of 2A-PAs are in P.

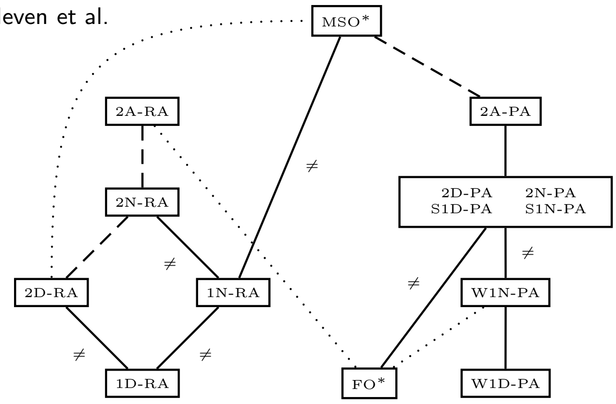

Fig. 1. Inclusions between the classes under consideration. Solid lines indicate inclusion (strictness shown as $\neq$), dotted lines indicate that the classes are incomparable. Dashed lines indicate inclusions where strictness is subject to complexity-theoretic assumptions. Additionally, 2D-RA $\not\subseteq$ 1N-RA and they are incomparable if LOGSPACE and NL are different. The three inequalities on the left regarding register automata as well as the fact that 2D-RA is not included in 1N-RA are already proved in[Kaminski and Francez 1994].

1.1 Related work

Kaminski and Francez were the first to consider RAs (which they called *finite-memory automata*) to handle strings over infinite alphabets. They showed that 1N-RAs are closed under union, intersection, concatenation, and Kleene star. They further showed that non-emptiness is decidable for 1N-RAs and that containment is decidable for 1N-RAs when the automaton in which containment has to be tested has only two registers. Sakamoto and Ikeda considered the complexity of the membership and the non-emptiness problem of 1D-RAs and 1N-RAs [Sakamoto and Ikeda 2000].

When the input is restricted to a finite alphabet, PAs recognize the regular languages, even in the presence of alternation [Milo et al. 2000]. We point out that the pebbling mechanism we employ is based on the one of Milo, Suciu, and Vianu [Milo et al. 2000] and is more liberal than the one used by Globerman and Harel [Globerman and Harel 1996]: indeed, in our case, after a pebble is placed the automaton can still walk over the whole string and sense the presence of the other pebbles. Globerman and Harel prove certain lower bounds in the gap of succinctness of the expressibility of their automata.

In [Otto 1985] regular and context-free languages and finite automata over countably infinite alphabets are considered. In the context of that article, the alphabet is ordered and automata can make use of the relative position of symbols in this order. As far we can see, the results are incomparable to ours.

1.2 Overview

This paper is organized as follows. In Section 2, we provide the formal framework. In Section 3, we study register automata. In Section 4, we examine pebble automata and conclude the section by comparing the register and pebble models. In Section 5, we consider decision problems. We end with a discussion in Section 6.

2. DEFINITIONS

We consider strings over an infinite alphabet $\mathbf{D}$. Intuitively, $\mathbf{D}$ represents a set of data values. Formally, a $\mathbf{D}$-*string* w is a finite sequence $d_1 \cdots d_n$, where each d_i is from $\mathbf{D}$ and $n \geq 0$. The *length* $|w|$ of w is n and we write $\mathrm{dom}(w)$ for the set $\{1, \ldots, |w|\}$ of positions in w. For $i \in \mathrm{dom}(w)$, we also write $\mathrm{val}_w(i)$ for d_i. A *language* is a set of $\mathbf{D}$-strings.

As we are often dealing with 2-way automata we delimit input strings by two special symbols, $\rhd, \lhd \notin \mathbf{D}$ for the left and the right end of the string, neither of which is in $\mathbf{D}$. Hence, automata always work on extended strings of the form $w = \rhd v \lhd$, where $v \in \mathbf{D}^*$. The positions of $\rhd$ and $\lhd$ are 0 and $|w|+1$, respectively. We write $\mathrm{dom}^+(w)$ for the extended set $\{0, \ldots, |w|+1\}$ of positions. We likewise extend the function val_w by defining $\mathrm{val}_w(0) = \rhd$ and $\mathrm{val}_w(|w|+1) = \lhd$.

2.1 Register automata

As explained in the introduction, register automata are finite state machines equipped with a finite number of registers. These registers hold values from $\mathbf{D}$. When processing a string, an automaton compares the symbol on the current position with values in the registers; based on this comparison it can decide to store the current symbol in some register. We stress that the only allowed operation on registers (apart from assignment) is a comparison with the symbol currently being processed. A transition depends on the current state and whether the current symbol is already present in some register. The transition relation specifies change of state, movement of the head, and possibly whether the current symbol should be stored in a register. We follow the definition of Kaminski and Francez [Kaminski and Francez 1994; 1990].

Definition 2.1. A *nondeterministic two-way k-register automaton $\mathcal{B}$ over* $\mathbf{D}$ is a tuple (Q, q_0, F, τ_0, P) where

—Q is a finite set of *states*;

—$q_0 \in Q$ is the *initial state*;

—$F \subseteq Q$ is the set of *final states*;

—$\tau_0 : \{1, \ldots, k\} \to \mathbf{D} \cup \{\rhd, \lhd\}$ is the *initial register assignment*; and,

—P is a finite set of *transitions* of the forms $(i, q) \to (q', d)$ or $q \to (q', i, d)$. Here, $i \in \{1, \ldots, k\}$, $q, q' \in Q$ and $d \in \{\mathrm{stay}, \mathrm{left}, \mathrm{right}\}$.

Given a string w, a *configuration of $\mathcal{B}$ on* w is a tuple $[j, q, \tau]$ where $j \in \mathrm{dom}^+(w)$, $q \in Q$, and $\tau : \{1, \ldots, k\} \to \mathbf{D} \cup \{\rhd, \lhd\}$. The *initial configuration* is $\gamma_0 := [1, q_0, \tau_0]$. A configuration $[j, q, \tau]$ with $q \in F$ is *accepting*. Given $\gamma = [j, q, \tau]$, the transition $(i, p) \to \beta$ (respectively, $p \to \beta$) *applies* to γ iff $p = q$ and $\mathrm{val}_w(j) = \tau(i)$ (respectively, $\mathrm{val}_w(j) \neq \tau(i)$ for all $i \in \{1, \ldots, k\}$).

Given configurations $\gamma = [j, q, \tau]$ and $\gamma' = [j', q', \tau']$, we define the one step transition relation $\vdash$ on configurations as follows: $\gamma \vdash \gamma'$ iff there is a transition $(i, q) \to (q', d)$ that applies to γ, $\tau' = \tau$, and $j' = j$, $j' = j - 1$, $j' = j + 1$ whenever $d = \mathrm{stay}$, $d = \mathrm{left}$, or $d = \mathrm{right}$, respectively; or there is a transition $q \to (q', i, d)$ that applies to γ, j' is as defined in the previous case, and τ' is obtained from τ by setting $\tau'(i)$ to $\mathrm{val}_w(j)$.

Note that this definition implies that the automaton never moves from position 0 to the left or from position $|w|+1$ to the right.

We denote the transitive closure of $\vdash$ by $\vdash^*$. Intuitively, transitions $(i,q) \rightarrow (q',d)$ can only be applied when the value of the current position is in register i. Transitions $q \rightarrow (q',i,d)$ can only be applied when the value of the current position differs from all the values in the registers. In this case, the current value is copied into register i.

To allow automata to recognize the endmarkers, we usually assume that the initial register assignment contains the symbols $\triangleright$ and $\triangleleft$.

As usual, a string w is accepted by $\mathcal{B}$, if $\gamma_0 \vdash^* \gamma$, for some accepting configuration γ. The language $L(\mathcal{B})$ consists of all strings w that are accepted by $\mathcal{B}$.

An automaton is *deterministic*, if in each configuration at most one transition applies. If there are no left-transitions, then the automaton is *one-way*. Alternating automata will be defined below in Subsection 2.3. As explained in the introduction, we refer to these automata as dC-RA where $d \in \{1,2\}$ and $C = \{\mathrm{D}, \mathrm{N}, \mathrm{A}\}$. Clearly, when the input is restricted to a finite alphabet, RAs accept only regular languages.[2]

Example 2.2. To illustrate the definition of an RA, we recall Example 1 of [Kaminski and Francez 1994]. We construct an automaton $\mathcal{B}$ accepting the language $L = \{d_1 \cdots d_n \mid n \geq 0, \exists i,j : i \neq j \wedge d_i = d_j\}$. The automaton is nondeterministic and one-way. Intuitively, when processing the input string, $\mathcal{B}$ nondeterministically chooses an input symbol and stores it in register 2. The automaton then compares the remainder of the input symbols with the symbol in register 2; $\mathcal{B}$ accepts when one of them equals the value in the second register. Formally, $\mathcal{B} = (Q, q_\triangleright, F, \tau_0, P)$ is a 2-register automaton and is defined as follows:

—$Q = \{q_\triangleright, q_1, q_2, q_{\mathrm{acc}}\}$;
—$F = \{q_{\mathrm{acc}}\}$;
—$\tau_0(1) = \triangleright$ and $\tau_0(2) = \triangleleft$; and,
—P consists of the following transitions:
 (1) $(1, q_\triangleright) \rightarrow (q_1, \mathrm{right})$;
 (2) $q_1 \rightarrow (q_1, 1, \mathrm{right})$;
 (3) $q_1 \rightarrow (q_2, 2, \mathrm{right})$;
 (4) $(2, q_2) \rightarrow (q_{\mathrm{acc}}, \mathrm{stay})$;
 (5) $(1, q_2) \rightarrow (q_2, \mathrm{right})$;
 (6) $q_2 \rightarrow (q_2, 1, \mathrm{right})$;

We briefly explain the above construction: (1) $\mathcal{B}$ reads the left endmarker and moves to state q_1; (2–3) $\mathcal{B}$ chooses whether the current symbol should be disregarded or whether it is one that will reappear later on. In the first case, $\mathcal{B}$ remains in state q_1 and stores the current symbol in register 1 (this register acts as a trash can and will be disregarded in the remainder of the computation); in the latter case, M moves to state q_2, stores the current symbol in register 2, and moves right. (4–6) if $\mathcal{B}$ encounters the symbol that is stored in register 2, it moves to the final state and accepts; otherwise $\mathcal{B}$ stays in state q_2. □

[2]Recall that two-way alternating finite automata over finite alphabet strings accept only the regular languages [Ladner et al. 1984].

2.2 Pebble automata

Pebble automata are finite state machines equipped with a finite number of pebbles. To ensure a "regular" behavior, the use of pebbles is restricted by a stack discipline. To this end, pebbles are numbered from 1 to k and pebble $i+1$ can only be placed when pebble i is present on the string. Likewise, pebble i can only be lifted when pebble $i+1$ is not present on the string. The highest-numbered pebble present on the string acts as the head of the automaton. A transition depends on the current state, the pebbles placed on the current position of the head, and the equality type of the data values under the placed pebbles. The transition relation specifies change of state, movement of the head, and possibly whether the head pebble is removed or a new pebble is placed. If pebble i is removed, pebble $i-1$ becomes the current head.

In our formal definition, we borrow some notation from Milo, Suciu, and Vianu [Milo et al. 2000].

Definition 2.3. A *nondeterministic two-way k-pebble automaton $\mathcal{A}$ over* $\mathbf{D}$ is a tuple (Q, q_0, F, T) where

—Q is a finite set of *states*;

—$q_0 \in Q$ is the *initial state*;

—$F \subseteq Q$ is the set of *final states*; and,

—T is a finite set of *transitions* of the form $\alpha \to \beta$, where

 —α is of the form (i, s, P, V, q) or (i, P, V, q), where $i \in \{1, \dots, k\}$, $s \in \mathbf{D} \cup \{\triangleright, \triangleleft\}$, $P, V \subseteq \{1, \dots, i-1\}$, and

 —β is of the form (q, d) with $q \in Q$ and

$$d \in \{\text{stay}, \text{left}, \text{right}, \text{place-new-pebble}, \text{lift-current-pebble}\}.$$

Given a string w, a *configuration of $\mathcal{A}$ on w* is of the form $\gamma = [i, q, \theta]$ where $i \in \{1, \dots, k\}$, $q \in Q$ and $\theta : \{1, \dots, i\} \to \text{dom}^+(w)$. We call θ a *pebble assignment* and i the *depth* of the configuration (and of the pebble assignment). Sometimes we denote the depth of a configuration γ (pebble assignment θ) by depth(γ) (depth(θ)). The *initial configuration* is $\gamma_0 := [0, q_0, \theta_0]$ where $\theta_0(1) = 0$. A configuration $[i, q, \theta]$ with $q \in F$ is *accepting*.

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

(1) $i = j$, $p = q$,
(2) $V = \{l < i \mid \text{val}_w(\theta(l)) = \text{val}_w(\theta(i))\}$
(3) $P = \{l < i \mid \theta(l) = \theta(i)\}$, and
(4) $\text{val}_w(\theta(i)) = s$.

A transition $(i, P, V, q) \to \beta$ *applies* to γ if (1)–(3) hold and no transition

$$(i, s, P, V, q) \to \beta$$

of T applies to γ. Intuitively, $(i, s, P, V, p) \to \beta$ applies to a configuration if pebble i is the current head, p is the current state, V is the set of pebbles that see the same symbol as the top pebble, P is the set of pebbles that sit at the same position as the top pebble, and the current symbol seen by the top pebble is s. As T is a finite set of transitions, the automaton can only identify a finite number of distinguished symbols s. Hence, every s can be regarded as a constant. In the register model, constants are determined by the initial register assignment.

We define the transition relation $\vdash$ as follows: $[i, q, \theta] \vdash [i', q', \theta']$ iff there is a transition $\alpha \to (p, d)$ that applies to γ such that $q' = p$ and $\theta'(j) = \theta(j)$, for all $j < i$, and

—if $d =$ stay, then $i' = i$ and $\theta'(i) = \theta(i)$,

—if $d =$ left, then $i' = i$ and $\theta'(i) = \theta(i) - 1$,

—if $d =$ right, then $i' = i$ and $\theta'(i) = \theta(i) + 1$,

—if $d =$ place-new-pebble, then $i' = i + 1$, $\theta'(i + 1) = \theta'(i) = \theta(i)$,

—if $d =$ lift-current-pebble then $i' = i - 1$.

The definitions of the *accepted language*, *deterministic* and *one-way* are analogous to the case of register automata. We refer to these automata as dC-PA where d and C are as before.

In the above definition, new pebbles are placed at the position of the most recent pebble. An alternative would be to place new pebbles at the beginning of the string. While the choice makes no difference in the two-way case, it is significant in the one-way case. We refer to the model as defined above as *weak* pebble automata and to the latter as *strong* pebble automata. Strong pebble automata are formally defined by setting $\theta'(i + 1) = 0$ (and keeping $\theta'(i) = \theta(i)$) in the *place-new-pebble* case of the definition of the transition relation.

It should be noted that when a PA lifts pebble i, the control is transferred to pebble $i - 1$. Therefore, even weak 1D-PAs can make several left-to-right sweeps of the input string.

EXAMPLE 2.4. To illustrate the definition of a PA, we exhibit a PA $\mathcal{A}$ accepting the language L defined in Example 2.2. The PA $\mathcal{A}$ operates like the RA $\mathcal{B}$ of Example 2.2 with the only difference that $\mathcal{A}$ lays down a pebble when $\mathcal{B}$ would store the current value in the second register. Formally, $\mathcal{A} = (Q, q_1, F, T)$ is a 2-pebble automaton and is defined as follows:

—$Q = \{q_1, q_2, q_\to, q_{\mathrm{acc}}\}$;

—$F = \{q_{\mathrm{acc}}\}$;

—T consists of the following transitions:

(1) $(1, \emptyset, \emptyset, q_1) \to (q_1, \mathrm{right})$;
(2) $(1, \emptyset, \emptyset, q_1) \to (q_\to, \text{place-new-pebble})$;
(3) $(2, \{1\}, \{1\}, q_\to) \to (q_2, \mathrm{right})$;
(4) $(2, \emptyset, \emptyset, q_2) \to (q_2, \mathrm{right})$;
(5) $(2, \emptyset, \{1\}, q_2) \to (q_3, \mathrm{stay})$;

Recall that the highest-numbered pebble on the string acts as the head of the automaton. So, although we only place one pebble, $\mathcal{A}$ is a 2-pebble PA. We briefly explain the above construction: (1–2) $\mathcal{A}$ stays in state q_1 while moving to the right or places a pebble; (3) after the placement of the pebble $\mathcal{A}$ moves one position to the right; (4–5) $\mathcal{A}$ continues moving to the right and when $\mathcal{A}$ reads a symbol equal to the symbol under the first pebble, it moves to the final state. □

EXAMPLE 2.5. We define a 2N-PA with three pebbles that accepts all words w where w is of length at least two and there exists some position i in dom(w) such that the set of symbols occurring at positions smaller than i is disjoint from the set of symbols occurring at positions larger than or equal to i. For example, the automaton accepts $\triangleright\, abb \,\triangleleft$ and rejects $\triangleright\, abab \,\triangleleft$. Informally, the automaton works as follows: the first pebble is used to nondeterministically guess i. The second pebble is used to step through the positions to the left of i, one at a time, from $i-1$ down to $\triangleright$. For each such position of the second pebble, the third pebble is used to check that all symbols at positions $\geq i$ are distinct from the symbol under pebble 2. The input is accepted if pebble 2 can reach the leftmost symbol $\triangleright$. We define the 2N-PA automaton $A = (Q, q_0, F, T)$ where $Q = \{q_0, \ldots, q_7\}$, $F = \{q_5\}$ and T is the set of transitions:

$$
\begin{array}{lll}
(1, \triangleright, \emptyset, \emptyset, q_0) & \rightarrow (q_0, \text{right}) & \\
(1, \emptyset, \emptyset, q_0) & \rightarrow (q_1, \text{right}) & \\
(1, \triangleleft, \emptyset, \emptyset, q_1) & \rightarrow (q_2, \text{stay}) & \texttt{\% } q_2 \texttt{ is a "blocking" state} \\
(1, \emptyset, \emptyset, q_1) & \rightarrow (q_1, \text{right}) & \\
(1, \emptyset, \emptyset, q_1) & \rightarrow (q_3, \text{place-new-pebble}) & \\
 & & \\
(2, \{1\}, \{1\}, q_3) & \rightarrow (q_4, \text{left}) & \\
(2, \emptyset, \emptyset, q_3) & \rightarrow (q_4, \text{left}) & \\
(2, \triangleright, \emptyset, \emptyset, q_4) & \rightarrow (q_5, \text{stay}) & \texttt{\% accept} \\
(2, \emptyset, \emptyset, q_4) & \rightarrow (q_6, \text{place-new-pebble}) & \\
 & & \\
(3, \{2\}, \{2\}, q_6) & \rightarrow (q_6, \text{right}) & \\
(3, \emptyset, \emptyset, q_6) & \rightarrow (q_6, \text{right}) & \\
(3, \emptyset, \{2\}, q_6) & \rightarrow (q_6, \text{right}) & \\
(3, \{1\}, \{1\}, q_6) & \rightarrow (q_7, \text{right}) & \\
(3, \emptyset, \{1\}, q_7) & \rightarrow (q_7, \text{right}) & \\
(3, \emptyset, \emptyset, q_7) & \rightarrow (q_7, \text{right}) & \\
(3, \triangleleft, \emptyset, \emptyset, q_7) & \rightarrow (q_3, \text{lift-pebble}) &
\end{array}
$$

□

2.3 Alternating Automata

For both automata models we also define an *alternating version*. Alternating automata $\mathcal{A}$ additionally have a set $U \subseteq Q$ of *universal states*. The sets from $Q - U$ are called *existential*. If $U = \emptyset$, then the automaton is *nondeterministic*.

Let γ be a configuration. A *γ-run* of $\mathcal{A}$ on w is a tree where nodes are labeled with configurations as follows:

(1) the root is labeled with γ;

(2) every inner node labeled with an existential configuration δ has exactly one child δ' and $\delta \vdash \delta'$; and,

(3) every inner node labeled with a universal configuration δ has children labeled with $\gamma_1, \ldots, \gamma_n$ where $\{\gamma_1, \ldots, \gamma_n\} = \{\gamma' \mid \delta \vdash \gamma'\}$.

We call a γ_0-run, where γ_0 is the initial configuration simply a *run*. An *accepting* run is a run where every leaf node is labeled with an accepting configuration. The language accepted by $\mathcal{A}$, is defined as

$$L(\mathcal{A}) := \{w \in \mathbf{D}^* \mid \text{there is an accepting run of } \mathcal{A} \text{ on } w\}.$$

2.4 Logic

We consider first-order and monadic second-order logic over $\mathbf{D}$-strings. The representation as well as the logics are special instances of meta-finite structures and their logics as defined by Grädel and Gurevich [Grädel and Gurevich 1998]. To simplify the correspondence between logic and automata we let formulas being evaluated over strings of the form $\triangleright\ w \triangleleft$. More formally, a string w is represented by the logical structure with domain $\mathrm{dom}^+(w)$, the natural ordering $<$ on the domain, and a function $\mathrm{val} : \mathrm{dom}(w) \to \mathbf{D}$ instantiated by val_w. An atomic formula is of the form $x < y$, $\mathrm{val}(x) = \mathrm{val}(y)$, or $\mathrm{val}(x) = d$ for $d \in \mathbf{D} \cup \{\triangleright, \triangleleft\}$, and has the obvious semantics. The logic FO* is obtained by closing the atomic formulas under the boolean connectives and first-order quantification over $\mathrm{dom}^+(w)$. Hence, no quantification over $\mathbf{D}$ is allowed. The logic MSO* is obtained by adding quantification over unary predicates on $\mathrm{dom}^+(w)$; again, no quantification over $\mathbf{D}$ is allowed. A sentence φ defines a set of strings via $L(\varphi) := \{w \in \mathbf{D}^* \mid \triangleright\ w \triangleleft\ \models \varphi\}$.

As an example, consider the FO* formula

$$\forall x \forall y (x \neq y \to \mathrm{val}(x) \neq \mathrm{val}(y)),$$

defining the set of strings where every position carries a different symbol.

For each of the logics we consider the following holds. For each formula φ there is a formula φ' such that for all strings $w \in \mathbf{D}^*$ it holds $\triangleright\ w \triangleleft\ \models \varphi$ iff $w \models \varphi'$ and vice versa. Therefore, all our results translate into the setting where a string w is represented in the standard way by a finite structure over $\mathrm{dom}(w)$.

2.5 Terminology

In the sequel we present several expressiveness and complexity results. We first introduce some notation. First of all, we use the name $\mathcal{F}$ of a formalism also for the class of defined languages. As an example, we write 1N-PA=2N-PA to express the fact that the classes of languages accepted by 1N-PAs and 2N-PAs, respectively, coincide.

In addition to the expressive power of our formalisms, we are interested in the computational complexity of the following problems. Let $\mathcal{F}$ be a formalism defining languages of $\mathbf{D}$-strings. Then *membership of* $\mathcal{F}$ is the problem of determining, for given $M \in \mathcal{F}$ and $w \in \mathbf{D}^*$, whether $w \in L(M)$. We only consider data complexity [Abiteboul et al. 1995], that is, we consider M as fixed. Hence, the complexity is only measured in the size of w.

Two other problems we are interested in are *universality* and *containment* of $\mathcal{F}$. The first is the problem of deciding whether $L(M) = \mathbf{D}^*$ for a given $M \in \mathcal{F}$. The second is determining whether $L(M_1) \subseteq L(M_2)$ for given $M_1, M_2 \in \mathcal{F}$. Here, the complexity is measured with respect to M, and M_1 and M_2, respectively.

3. REGISTER AUTOMATA

We start by investigating RAs. In particular, we compare them with FO* and MSO*. Our main result is that RAs are orthogonal to these logics as they cannot even express all FO* properties but can express properties not definable in MSO*. Further, we separate the variants of RAs subject to standard complexity-theoretic assumptions.

3.1 Expressiveness

We show that there is a property of $\mathbf{D}$-strings that is not definable in MSO* but is easily expressible with a 2D-RA. In the rest of the paper we make extensive use of the symbol $\#$ as a delimiter. We, therefore, assume that $\# \in \mathbf{D}$.

THEOREM 3.1. *2D-RA* $\not\subseteq$ *MSO**.

PROOF. Consider strings of the form $u\#v$ where $u, v \in (\mathbf{D} - \{\#\})^*$. Define N_u and N_v as the set of symbols occurring in u and v, respectively. Denote by n_u and n_v their cardinalities. We show that there is a 2D-RA $\mathcal{A}$ that accepts $u\#v$ iff $n_u = n_v$ while there is no such MSO* sentence.

We first introduce some notation. For a string w, denote by $\mathrm{lmo}_w(d)$ the position in w of the leftmost occurrence of $d \in \mathbf{D}$. Suppose $N_u = \{a_1, \ldots, a_n\}$ and $N_v = \{b_1, \ldots, b_m\}$ where $n = n_u$, $m = n_v$ and, for every $i < j$, $\mathrm{lmo}_u(a_i) < \mathrm{lmo}_u(a_j)$ and $\mathrm{lmo}_v(b_i) < \mathrm{lmo}_v(b_j)$. Then $\mathcal{A}$ compares n_u with n_v by visiting $\mathrm{lmo}_u(a_1)$, $\mathrm{lmo}_v(b_1)$, $\mathrm{lmo}_u(a_2)$, $\mathrm{lmo}_v(b_2)$, $\ldots$ in sequence. Hence, if a_n and b_m are not reached simultaneously, the automaton rejects the input string. Otherwise, it accepts.

It remains to explain how the 2D-RA can visit $\mathrm{lmo}_u(a_1)$, $\mathrm{lmo}_v(b_1)$, $\mathrm{lmo}_u(a_2)$, $\mathrm{lmo}_v(b_2)$, $\ldots$ in sequence. Clearly, $\mathrm{lmo}_u(a_1)$ and $\mathrm{lmo}_v(b_1)$ are the first positions of u and v, respectively. If $\mathcal{A}$ has the values of a_i and b_i stored in its registers it can compute a_{i+1} and b_{i+1}. Indeed, to compute a_{i+1} it proceeds as follows. It first moves its head to position $\mathrm{lmo}_w(a_i)$ by going to the left boundary and afterwards walking to the right until it encounters a_i. Now it tests, for all positions $\mathrm{lmo}_w(a_i) + j$, $j > 0$, starting with $j = 1$, whether they carry a leftmost occurrence of a symbol d. This is done as follows: it stores d in a register, and goes from position $\mathrm{lmo}_w(a_i) + j$ to the left until it either sees a d or reaches the left end of the string. In the former case, it goes back to $\mathrm{lmo}_w(a_i) + j$ (identified by the first d) and proceeds with $\mathrm{lmo}_w(a_i) + j + 1$. In the latter case $\mathrm{lmo}_w(a_i) + j$ carries the leftmost d, therefore a_{i+1} is identified. The computation of b_{i+1} can be done in a similar way. We note that the used construction is similar to Example 11 of [Kaminski and Francez 1994].

Assume towards a contradiction that φ^* is an MSO* sentence such that $u\#v \models \varphi^*$ iff $n_u = n_v$. Let C be the set of $\mathbf{D}$-symbols occurring in φ^*. We call a string $u\#v$ admissible iff $N_u \cap N_v = \emptyset$; each $\mathbf{D}$-symbol occurs at most once in u or v; and, no symbol in C occurs in u or v. Let φ be obtained from φ^* by replacing each occurrence of $\mathrm{val}(x) = \mathrm{val}(y)$ by $x = y$, and every occurrence of $\mathrm{val}(x) = d$ by *false*, if $d \neq \#$. Then for every admissible string $d_1 \cdots d_n \# e_1 \cdots e_m$, $a^n \# a^m \models \varphi$ iff $d_1 \cdots d_n \# e_1 \cdots e_m \models \varphi$ iff $d_1 \cdots d_n \# e_1 \cdots e_m \models \varphi^*$ iff $n = m$. Hence, $\{a^n \# a^n \mid n \in \mathbb{N}\}$ would be MSO definable and therefore regular. This leads to the desired contradiction. $\square$

As MSO has proven to be a good yardstick for generalizations of regular string languages to trees and graphs [Courcelle 1990; Thomas 1997], the above theorem suggests that 2D-RAs can behave in a "non-regular" manner.

The following theorem shows that this is essentially due to the ability to move the head in both directions.

THEOREM 3.2. *1N-RA* $\subseteq$ *MSO**.

PROOF. Let $\mathcal{B} = (Q, q_0, F, \tau_0, T)$ be a 1N-RA. Assume for the moment that no transition of $\mathcal{B}$ encodes a stay-move. That is, no transition is of the form $(i, q) \to (q', \text{stay})$ or $q \to (q', i, \text{stay})$. We describe the construction of an MSO* formula φ which holds for an input $w = w_1 \cdots w_n$ iff $\mathcal{B}$ has an accepting computation on w. First of all, φ guesses, for each position i of w, the state that $\mathcal{B}$ is in after reading w_i. This can be done by existentially quantifying over (disjoint) sets $(S_q)_{q \in Q}$, where $i \in S_q$ means that the state of $\mathcal{B}$ after reading w_i is q. Next, φ guesses, again for each position i, which transition $\mathcal{B}$ applies when it reads w_i. This is done by quantifying over sets $(T_t)_{t \in T}$. Correspondingly, $i \in T_t$ means that transition t is applied when $\mathcal{B}$ reads w_i. Now assume that there is an accepting computation of $\mathcal{B}$ on w and that $(S_q)_{q \in Q}$ and $(T_t)_{t \in T}$ are chosen accordingly. Then, for each register j, the register content of $\mathcal{B}$ before reading position i can be determined as follows: it is either determined from the initial register assignment τ_0, if the register has not been changed yet, otherwise it is the symbol w_l, where $l = \max\{m < i \mid m \in T_t, \text{ where } t \text{ is of the form } q \to (q', j, d)\}$. It is straightforward to express this in MSO* (even in FO*). With the ability to determine register contents it is now easy to check (again in FO*) that $(S_q)_{q \in Q}$ and $(T_t)_{t \in T}$ are consistent with the transition relation of $\mathcal{B}$.

It remains to describe how we can deal with stay-transitions. To this end, we remark that when a 1N-RA accepts a string, there is a run that makes at every position only a bounded number of stay transitions. Indeed, let r be the number of transitions of the 1N-RA. Consider an input string and an accepting sequence of configurations. Suppose the automaton makes a sequence of stay-transitions at some position in the string. Call stay-transitions of the form $(i, q) \to (q', \text{stay})$ and $p \to (p', i, \text{stay})$, type-1 and type-2 transitions, respectively. Note that a type-2 transition can only be followed by type-1 transitions. Further, a consecutive sequence of more than r type-1 transitions always contains a cycle, and as the automaton accepts the input string, this sequence can be reduced to one containing less than $r + 1$ type-1 transitions. So, the construction sketched above can take stay-transitions into account by quantifying over sets $T_{\bar{t}}$, as opposed to sets T_t, where $\bar{t}$ is a sequence of at most r transitions. The latter information can then be used to check the register content. Consequently, consistency with the transition relation can be enforced. $\square$

So far, one might think that 2-way register automata are simply too strong to be compared with MSO logic. The next result provides more evidence that the computation model based in registers is simply not comparable with logics. More precisely, we show that RAs cannot capture FO*, even with alternation. The proof is based on a communication complexity argument.It should be noted that our communication complexity proofs could as well be expressed in terms of crossing sequences [Hennie 1965], as one of the referees pointed out. However, when the formalism of RAs is strengthened, the idea of crossing sequences no longer applies while the communication complexity method does [Neven 2002b].

In communication complexity (see, e.g., [Hromkovic 2000]) the input string is divided in a pre-determined manner between two parties (generally referred to as `I` and `II`) that can send messages to each other according to a given protocol. A language is accepted by a protocol if for each string both parties can decide after execution of the protocol whether the string belongs to the language. Both parties have unlimited computation power on their part of the string. The protocol only restricts the way in which the parties communicate, typically by restricting the form and number of messages.

We next sketch the main idea for separation of 2D-RAs and FO*. This basic idea is then extended to deal with alternation in the proof of Theorem 3.7. We consider strings of the form $u\#v$ where u and v encode sets of sets of $\mathbf{D}$-symbols in a suitable way. As will be seen, the language

$$L := \{u\#v \mid u \text{ and } v \text{ represent the same set of sets}\}$$

is definable in FO*. To show that the language is not accepted by a 2D-RA, we note that each 2D-RA $\mathcal{A}$ working on strings of the form $u\#v$ can be simulated by a protocol in the following way: `I` is given u while `II` is given v. The first party simulates $\mathcal{A}$ until this computation tries to cross the delimiter # to the right. At this point, it sends the present state q and the data values $d_1, \ldots, d_k$ currently in its registers. Hence, `II` gets full information about the configuration of $\mathcal{A}$ (as the position of the symbol # is fixed). Then `II`, similarly, simulates the behavior of $\mathcal{A}$ until it tries to cross # to the left and then `II` sends in turn the current configuration to `I`. This process continues until one of the parties detects a final state. What kind of protocol can simulate such behavior? First, we need a message for every configuration. Suppose we restrict to at most N different data values in the strings $u\#v$. Then $M := |Q| \cdot N^k$ different messages are needed. Here, k is the number of registers and Q is the set of states of the 2D-RA. Call a sequence of messages a dialogue. We only need to consider dialogues up to length M (as every message can only be sent once in every direction). Hence, there are only M^{2M} different dialogues on input strings consisting of N different $\mathbf{D}$-symbols. The latter value is exponential in N. However, there are 2^{2^N} sets of sets of N different $\mathbf{D}$-symbols. So for large enough N there must be different strings $u\#u$ and $v\#v$ accepted by the protocol via the same dialogue but such that u and v represent different sets of sets. But, this means that $u\#v$ is also accepted. Hence, no such protocol can define L, which implies that no 2D-RA accepts L. In the proof of Theorem 3.7 we extend this idea to alternation. This requires exchanging partial computation trees of the 2A-RA. To apply the same counting trick we then have to consider more deeply nested hypersets.

Our proof technique is inspired by a proof of Abiteboul, Herr, and Van den Bussche [Abiteboul et al. 1999] separating the temporal query languages ETL from TS-FO. To this end, they show that every query in ETL on a special sort of databases can be evaluated by a communication protocol with a constant number of messages, whereas this is not the case for TS-FO. In our case, to simulate 2D-RA (and 2A-RA) we need a more powerful protocol where the number of messages depends on the number of different data values in u and v.

The remaining part of this subsection is devoted to the proof that FO* is not included in 2A-RA. We start with some terminology. Let D be a finite or infinite set. A *1-hyperset over* D is a finite subset of D. For $i > 1$, an *i-hyperset over* D is a finite set of $(i-1)$-hypersets over D. We often denote i-hypersets with a superscript i, as in $S^{(i)}$.

For ease of presentation, we assume that $\mathbf{D}$ contains all natural numbers. For each $j > 0$, let $\mathbf{D}_j$ be $\mathbf{D} - \{1, \ldots, j, \#\}$. Next, let $j > 0$ be fixed. We inductively define *encodings* of i-hypersets over $\mathbf{D}_j$. First, a string $w = 1d_1d_2 \cdots d_n1$ with $d_1, \ldots, d_n \in \mathbf{D}_j$ is an encoding of the 1-hyperset $H(w) = \{d_1, \ldots, d_n\}$ over $\mathbf{D}_j$. For each $i \leq j$, and encodings $w_1, \ldots, w_n$ of $(i-1)$-hypersets, $w = iw_1iw_2 \cdots iw_ni$ is an encoding of the i-hyperset $H(w) = \{H(w_i) \mid i \leq n\}$. We define $L^m_=$ as the language

$$\{u\#v \mid u \text{ and } v \text{ are encodings of } m\text{-hypersets over } \mathbf{D}_m \text{ and } H(u) = H(v)\}.$$

LEMMA 3.3. *For each m, $L^m_=$ is definable in FO*.*

PROOF. Let $m \geq 1$ be fixed. For each $i \leq m$, we define an FO* formula $\varphi_i(x^\ell, x^r, y^\ell, y^r)$ expressing on input w that the intervals $[x^\ell, x^r]$ and $[y^\ell, y^r]$ encode the same i-hyperset over $\mathbf{D}_m$.

By $x \in [y, z]$ we abbreviate $y \leq x \wedge x \leq z$. By $\mathrm{clean}_i(x^\ell, x^r)$ we abbreviate the formula $\neg\exists z(x^\ell < z \wedge z < x^r \wedge \mathrm{val}(z) \in \{i, \ldots, m, \#\})$. For each i, the formula $\mathrm{hyp}_i(x^\ell, x^r)$ shall express that $[x^\ell, x^r]$ encodes an i-hyperset. Therefore, let

$$\mathrm{hyp}_1(x^\ell, x^r) := x^\ell < x^r \wedge \mathrm{val}(x^\ell) = 1 \wedge \mathrm{val}(x^r) = 1 \wedge \mathrm{clean}_1(x^\ell, x^r)$$

and, for $i > 1$,

$$\begin{aligned}\mathrm{hyp}_i(x^\ell, x^r) := x^\ell < x^r \wedge \mathrm{val}(x^\ell) = i \wedge \mathrm{val}(x^r) = i \\ \wedge\, \mathrm{clean}_i(x^\ell, x^r) \wedge val(x^\ell + 1) = i - 1 \wedge \mathrm{val}(x^r - 1) = i - 1 \\ \wedge\, \forall y^\ell, y^r(\mathrm{val}(y^\ell) = i - 1 \wedge \mathrm{val}(y^r) = i - 1 \wedge \mathrm{clean}_{i-1}(y^\ell, y^r) \to \mathrm{hyp}_{i-1}(y^\ell, y^r)).\end{aligned}$$

The formula φ_i is inductively defined as follows:

$$\begin{aligned}\varphi_1(x^\ell, x^r, y^\ell, y^r) := \mathrm{hyp}_1(x^\ell, x^r) \wedge \mathrm{hyp}_1(y^\ell, y^r) \\ \wedge\, \forall u \in [x^\ell, x^r]\exists v \in [y^\ell, y^r](\mathrm{val}(u) = \mathrm{val}(v)) \\ \wedge\, \forall v \in [y^\ell, y^r]\exists u \in [x^\ell, x^r](\mathrm{val}(u) = \mathrm{val}(v))\end{aligned}$$

and, for $i > 1$,

$$\begin{aligned}\varphi_i(x^\ell, x^r, y^\ell, y^r) := \mathrm{hyp}_i(x^\ell, x^r) \wedge \mathrm{hyp}_i(y^\ell, y^r) \\ \wedge\, \forall u^\ell, u^r \in [x^\ell, x^r](\mathrm{hyp}_{i-1}(u^\ell, u^r) \to \exists v^\ell, v^r \in [y^\ell, y^r]\varphi_{i-1}(u^\ell, u^r, v^\ell, v^r)) \\ \wedge\, \forall v^\ell, v^r \in [y^\ell, y^r](\mathrm{hyp}_{i-1}(v^\ell, v^r) \to \exists u^\ell, u^r \in [x^\ell, x^r]\varphi_{i-1}(u^\ell, u^r, v^\ell, v^r)).\end{aligned}$$

The language $L^m_=$ is then expressed by the formula

$$\begin{aligned}\exists x^\ell, x^r, z, y^\ell, y^r(x^\ell = 1 \wedge x^r + 1 = z \wedge \mathrm{val}(z) = \# \\ \wedge\, z + 1 = y^\ell \wedge y^r = \max \wedge \varphi_m(x^\ell, x^r, y^\ell, y^r)).\end{aligned}$$

Here, 1 and max refer to the first and last element of the string, respectively, and $+1$ is the successor function. □

Next, we show that no 2A-RA can recognize $L^m_=$ for $m > 3$. As described above, the underlying idea is that for large enough m, a 2A-RA simply cannot communicate enough information between the two sides of the input string to check whether $H(u)$ equals $H(v)$. Define $\exp_0(n) := n$ and $\exp_i(n) := 2^{\exp_{i-1}(n)}$, for $i > 0$.

Definition 3.4. Let P be a binary predicate on m-hypersets over $\mathbf{D}$ and let $k, l \geq 0$. We say that P can be *computed by a* (k,l)*-communication protocol* between two parties (denoted by `I` and `II`) if there is a polynomial p such that for each finite set $D \subseteq \mathbf{D}$ there is a finite alphabet Δ of size at most $p(|D|)$ such that, for all m-hypersets X, Y over D, $P(X,Y)$ can be computed as follows:

(1) `I` gets X and `II` gets Y;
(2) `I` sends a message $a_1(X)$ to `II` and `II` replies with a message $b_1 = b_1(Y, a_1)$ to `I`. Each message is a k-hyperset over Δ.
(3) `I` sends a message $a_2 = a_2(X, b_1)$ to `II` and `II` replies with a message $b_2 = b_2(Y, a_2)$ to `I` and so on.
(4) After $r \leq \exp_l(p(|D|))$ rounds of message exchanges, both `I` and `II` have enough information to decide whether $P(X,Y)$ holds. More precisely, they apply a Boolean function $a_{r+1}(X, b_r)$ (for `I`) or $b_{r+1}(Y, a_r)$ (for `II`) that evaluates to true iff $P(X,Y)$ holds.

So, formally, a protocol consists of the functions $a_1, \ldots, a_{r+1}, b_1, \ldots, b_{r+1}$. Note that the computing power of `I` and `II` can be completely arbitrary.

Lemma 3.5. *For $m > 3$, the equality-predicate of m-hypersets over $\mathbf{D}_m$ cannot be computed by a (2,2)-communication protocol.*

Proof. Towards a contradiction, suppose there is such a protocol. For every finite set D with d elements, the number of different possible messages is the number of 2-hypersets which is at most $\exp_2(p(d))$. Call a complete sequence of exchanged messages $a_1 b_1 a_2 b_2 \ldots, a_{r+1}, b_{r+1}$ a dialogue. Every dialogue has at most $\exp_2(p(d))$ rounds. Hence, there are at most $2^{2 \cdot 2^{p(d)} \exp_2(p(d))}$ different dialogues. However, the number of different m-hypersets over D is $\exp_m(d)$. Hence, for $m > 3$ and D large enough there are m-hypersets $X^{(m)} \neq Y^{(m)}$ such that the protocol gives the same dialogue for $(X^{(m)}, X^{(m)})$ and $(Y^{(m)}, Y^{(m)})$. But that means it also gives the same dialogue on $(X^{(m)}, Y^{(m)})$ and $(Y^{(m)}, X^{(m)})$. This leads to the desired contradiction. □

If, for some m, L is a set of strings of the form $u\#v$, such that u and v encode m-hypersets over $\mathbf{D}_m$ then we say that L encodes the binary predicate on m-hypersets defined by $H(L) := \{(H(u), H(v)) \mid u\#v \in L\}$.

Lemma 3.6. *If, for some m, L encodes a binary predicate on m-hypersets over $\mathbf{D}_m$ and L is recognized by a 2A-RA then $H(L)$ is computed by a (2,2)-communication protocol.*

Proof. Let $\mathcal{B} = (Q, q_0, U, F, \tau, P)$ be a k-register 2A-RA working on strings $u\#v$ of the required form. We assume w.l.o.g. that there are no transitions possible from accepting configurations. Further we assume that $\mathcal{B}$ never changes direction at the symbol $\#$. Hence, on a string $u\#v$, when it leaves u to the right it enters

v and vice versa. Define p as the polynomial $p(n) := |Q|n^k$, that is, the number of configurations that can be assumed at the position labeled with # on strings with at most n different data values. Let $w = u\#v$ be an input string; let D be the set of data values occurring in w. Let the position in the string of the symbol # be e. The configurations that are assumed at the position of # are therefore of the form $[e, q, \tau]$. We refer to them as #-configurations. Each #-configuration $\gamma = [e, q, \tau]$ is uniquely determined by its *representant* (q, τ) which we also denote by $\overline{\gamma}$. We set $\Delta := \{[q, \tau] \mid q \in Q, \text{and } \tau : \{1, \ldots, k\} \to D \cup \{\triangleright, \triangleleft\}\}$. So, Δ is the set of representants of #-configurations. Note that $|\Delta| = p(|D|)$.

In essence, both parties compute partial runs where they send the #-configurations in which $\mathcal{B}$ walks off their part of the string to the other party. To be concrete, I computes all the runs of $\mathcal{B}$ on u. The leaves of these runs consist of #-configurations or accepting configurations, and no inner vertex is labeled with a #-configuration. It then sends to II the set of all sets of representants of #-configurations appearing at leaves of such runs. Party II in turn, computes the same information for runs starting from the sets of #-configurations it received and sends it to I. This process is repeated. If after $\exp_2(|\Delta|)$ messages there is a message containing the empty set then the input is accepted (as accepting configurations are not transmitted, the presence of an empty set indicates a run where all leaves are accepting configurations). We next describe this formally.

A configuration $[i, q, \tau]$ is called a u-configuration if $i < e$ and a v-configuration if $i > e$. A #-configuration is called a $u\#$-configuration if it is assumed from the left and a $\#v$-configuration if it is assumed from the right. Note that, by our assumption that $\mathcal{B}$ never changes direction at #, we can distinguish these two sets of configurations.

We introduce the following notions. For an arbitrary configuration γ, a $(\gamma, \#)$-run is a run where the root is labeled with γ, all the leaves are labeled with accepting configurations or #-configurations and no inner vertex, besides possibly the root, is labeled by a #-configuration. For such a run t, we define Leaf-labels(t) as the set of representants of #-configurations occurring at the leaves of t. For a set $C^{(1)} = \{\overline{\gamma_1}, \ldots, \overline{\gamma_n}\}$ of representants of configurations, define $\ell(C^{(1)})$ as the set of sets of representants of configurations

$$\{\bigcup_{i=1}^{n} \text{Leaf-labels}(t_i) \mid \text{for each } i,\ t_i \text{ is a } (\gamma_i, \#)\text{-run}\}.$$

Finally, for a set $S^{(2)}$ of sets of representants of configurations, define

$$\ell(S^{(2)}) := \bigcup_{C^{(1)} \in S^{(2)}} \ell(C^{(1)}).$$

Let S_0 be the singleton 2-hyperset containing the singleton set $\{\overline{\gamma_0}\}$, that is, $S_0 := \{\{\overline{\gamma_0}\}\}$. Recall that γ_0 is the initial configuration. Define a sequence of 2-hypersets of representants of configurations by $S_i := \ell(S_{i-1})$, for all $i \geq 1$. Note that, for $i > 0$, if i is even then S_i is a 2-hyperset of representants of $\#v$-configurations. If i is odd then S_i is a 2-hyperset of representants of $u\#$-configurations.

Let us call a run t an i-pass run if all its leaf configurations are either accepting or #-configurations that are reached by computations that have visited # exactly

i times. Clearly, if $C^{(1)} \in S_i$, then there is an i-pass run t of $\mathcal{B}$ on w such that Leaf-labels$(t) = C^{(1)}$ and vice versa. Hence, $\mathcal{B}$ accepts w iff there is an i and a $C^{(1)} \in S_i$ containing the empty set.

The protocol works as follows. Party I starts by sending S_1. For $i > 0$, when party II receives S_{2i-1} it responds with S_{2i}; when party I receives S_{2i} it responds with S_{2i+1}. The parties accept whenever a 2-hyperset with the empty set is transmitted. As there are only $\exp_2(|\Delta|)$ different 2-hypersets over Δ the parties can reject if the empty set was never obtained after $\exp_2(|\Delta|)$ rounds of messages. $\square$

THEOREM 3.7. *FO* $\not\subseteq$ 2A-RA.*

PROOF. By Lemmas 3.5 and 3.6, $L^m_=$ is not computable by a 2A-RA. Hence, Theorem 3.7 follows. $\square$

3.2 Control

In the previous section we related RAs with logic. In this section we compare RAs with various kinds of control. Kaminski and Francez already showed that 1D-RAs are weaker than 1N-RAs and that there is a property (are all data values different?) that is in 2D-RA but not in 1N-RA. We can only separate the other classes by relying on complexity theoretic assumptions as explained next.

The main observation is that on strings of a special shape, RAs can simulate multi-head automata. These strings are of even length where the odd positions contain pairwise distinct elements and the even positions carry symbols from a finite alphabet, say $\{a, b\}$. By storing the unique ids preceding the a's and b's, the RA can remember the positions of the heads of a multi-head automaton. Note that 2D-RAs can check whether the input string is of the desired form. Deterministic, nondeterministic, and alternating multi-head automata recognize precisely LOGSPACE, NLOGSPACE, and PTIME languages, respectively [Chandra et al. 1981; Sudborough 1975]. We get a similar correspondence between automata on infinite alphabets and complexity classes.

Let $\mathbf{D}$ be a fixed infinite alphabet. For a language L over some finite alphabet $\Sigma \subseteq \mathbf{D}$ we define $f_{\mathbf{D}}(L)$ as the set of strings of the form $d_1 a_1 \cdots d_n a_n$ where the d_i are pairwise distinct symbols from $\mathbf{D}$ and $a_1 \cdots a_n \in L$.

THEOREM 3.8. *For each set $L \subseteq \Sigma^*$ the following conditions hold.*

(1) $L \in$ LOGSPACE *if and only if* $f_{\mathbf{D}}(L) \in$ *2D-RA.*
(2) $L \in$ NLOGSPACE *if and only if* $f_{\mathbf{D}}(L) \in$ *2N-RA.*
(3) $L \in$ PTIME *if and only if* $f_{\mathbf{D}}(L) \in$ *2A-RA.*

As a consequence of Theorem 3.8 and the above results of Francez and Kaminski, all four classes defined by the mentioned automata models are different unless the corresponding complexity classes collapse. We refer the reader to Figure 1 for a visual representation of these relationships.

4. PEBBLE AUTOMATA

We next focus on pebble automata. We show that PAs are better behaved than RAs with respect to the connection to logic. In a sense, PAs are more "regular" than RAs. Indeed, we show that strong 1D-PAs can simulate FO* and that even the most liberal pebble model, 2A-PA, can be defined in MSO*. Furthermore, we can separate 2A-PAs from MSO* under usual complexity-theoretic assumptions. Next, we show that weak one-way PAs do not suffice to capture FO*. Again, the proof is based on communication complexity. Finally, we prove that for strong PAs, the one-way, two-way, deterministic and nondeterministic variants collapse. Together with the straightforward closure under Boolean operations,[3] concatenation and Kleene star, these results suggest that strong PAs define a robust class of languages.

4.1 Expressiveness

THEOREM 4.1. *FO* $\subsetneq$ strong 1D-PA.*

PROOF. Clearly, FO* cannot define the **D**-strings of even length [Ebbinghaus and Flum 1999] while 1D-PAs (strong or weak) can easily do so.

For the inclusion, note that an FO* sentence φ in prenex normal form can be evaluated straightforwardly by a strong 1D-PA. We use one pebble for each quantifier. Pebble 1 is used for the outermost quantifier, and the pebble with the largest number is used for the innermost quantifier. The automaton cycles through all possible assignments of positions to the pebbles hence to the variables. In order to evaluate the quantifier-free body of φ, it records in its state all information about equalities and inequalities among the symbols at the pebbled positions as well as about their relative order. $\square$

We next show that PAs are subsumed by MSO*. Thus, they behave in a "regular" manner.

THEOREM 4.2. *2A-PA $\subseteq$ MSO*.*

PROOF. The proof is an extension to *infinite* alphabets of a proof in [Milo et al. 2000] where it is shown that alternating tree-walking pebble automata over *finite* alphabets can be simulated in MSO. Essentially, we need to express that the initial configuration of the automaton can reach some accepting configuration in the graph of consecutive configurations. However, because of the alternating control, we have to consider a directed *and/or* graph of configurations. Since this lies at the core of our construction, we therefore begin by showing how to express in MSO the *Alternating Graph Accessibility Problem*, AGAP [Greenlaw et al. 1995]. An alternating graph (or *and/or graph*) is a directed graph $G = (V, E)$ whose nodes V are partitioned into *and*-nodes and *or*-nodes: $V = V_\wedge \cup V_\vee$. AGAP consists of all pairs (G, x), where G is an alternating graph and x is an accessible node of G, with accessibility defined recursively as follows: an *and*-node is accessible if all its successors are accessible; an *or*-node is accessible if at least one of its successors is accessible. Note that *and*-nodes with no successor are by definition accessible. It can be shown that the set of accessible nodes is definable in MSO. Indeed, consider the formula

$$\varphi(x) \;:=\; \forall S(\textit{reverse-closed}(S) \Rightarrow S(x)), \tag{1}$$

[3]We remark that by a technique of Sipser [Sipser 1980] any 2D-PA can be transformed into an equivalent 2D-PA that always halt. This gives us closure under complement.

where *reverse-closed*(S) is:

$$\begin{aligned} &\forall y(D_\vee(y) \wedge \exists z(E(y,z) \wedge S(z)) \Rightarrow S(y)) \\ \wedge\ &\forall y(D_\wedge(y) \wedge \forall z(E(y,z) \Rightarrow S(z)) \Rightarrow S(y)). \end{aligned} \tag{2}$$

Here, $D_\vee$ and $D_\wedge$ are unary relations containing the *or*- and *and*-nodes, respectively.

In the following we require, w.l.o.g., that there are no transitions from a final state of an alternating k-pebble automaton. Given an alternating k-pebble automaton $\mathcal{A} = (Q, q_0, U, F, T)$ over $\mathbf{D}$ and a string w, we construct the following directed *and/or* graph $G_{\mathcal{A},w} = (V, E)$. Its *or*-nodes are all configurations of $\mathcal{A}$ on w whose states are existential; its *and*-nodes are all configurations whose states are universal, together with an additional distinguished *and*-node ϵ. The set of edges E is $\{(\gamma, \gamma') \mid \gamma \vdash \gamma'\} \cup \{(\gamma, \epsilon) \mid \gamma \text{ is accepting}\}$. It follows directly from the definitions of $L(\mathcal{A})$ and of AGAP that a string w is in $L(\mathcal{A})$ iff the initial configuration γ_0 is accessible in the graph $G_{\mathcal{A},w}$. Hence, it only remains to show that we can express the AGAP problem on $G_{\mathcal{A},w}$ in MSO*. Here, the difficulty lies in the fact that the nodes in $G_{\mathcal{A},w}$ are tuples of positions from $\mathrm{dom}^+(w)$ (a configuration $\gamma = [i, q, \theta]$ of $\mathcal{A}$ on w is represented by the i-tuple $(\theta(1), \ldots, \theta(i))$; q does not depend on w and will be encoded separately.) Hence, the predicate S in (1) is no longer unary. To circumvent that, we rely on a special property of $G_{\mathcal{A},w}$. Namely, if two nodes described by an i-tuple $(\theta(1), \ldots, \theta(i))$ and a j-tuple $(\theta'(1), \ldots, \theta'(j))$ are connected by an edge, then either $i = j$, or $i = j + 1$, or $i = j - 1$, and the tuples agree on all but the last position. This follows from the stack discipline on pebbles in $\mathcal{A}$ (only the last pebble can be moved) and allows us to quantify independently, on different portions of the graph. The construction of the MSO* formula relies on this observation. We outline this construction next.

For simplicity, we can assume w.l.o.g. that for each universal state q and (i, s, P, V), either $\{\beta \mid (i, s, P, V, q) \to \beta\} = \emptyset$ or there are two states q_1, q_2 such that $\{\beta \mid (i, s, P, V, q) \to \beta\} = \{(q_1, \mathit{stay}), (q_2, \mathit{stay})\}$. It is also convenient to assume that Q is partitioned into disjoint $Q_1 \cup \ldots \cup Q_k$ such that states in Q_i "control" pebble i, i.e., whenever i is the current pebble the current state is from Q_i. Furthermore, we enumerate the states in Q such that $Q = \{q_0, q_1, \ldots, q_n\}$, and $Q_1 = \{q_0, \ldots, q_{n_1}\}$, $Q_2 = \{q_{n_1+1}, \ldots, q_{n_2}\}$, …, $Q_k = \{q_{n_{k-1}+1}, \ldots, q_{n_k}\}$ and $n_k = n$.

We consider first the case when $\mathcal{A}$ uses a single pebble, to illustrate how one encodes the state in a configuration and how to encode the transitions. For simplicity, we only consider transitions involving constants, i.e., with left-hand side of the form (i, s, P, V, q); dealing with transitions with left-hand side of the form (i, P, V, q) is an easy generalization (see, e.g., Example 2.5). In the case of only one pebble, configurations can be identified with pairs (q, x), and transitions can be simplified to $(s, q) \to (p, d)$ where $d \in \{\mathit{stay,left,right}\}$. The MSO* formula $\varphi_{\mathcal{A}}$ defining acceptance by $\mathcal{A}$ uses a different unary predicate S_j for each state $q_j \in Q$. A set $\mathcal{C}$ of configurations corresponds to a choice of predicates S_j via $(q_j, x) \in \mathcal{C}$ iff $S_j(x)$. Namely $\varphi_{\mathcal{A}}$ is:

$$\varphi_{\mathcal{A}} := \forall S_0 \forall S_1 \ldots \forall S_{n_1}(\mathit{reverse\text{-}closed} \to S_0(0)), \tag{3}$$

where *reverse-closed* is a formula with free variables $S_0, S_1, \ldots, S_{n_1}$, stating that the set of configurations represented by $S_0, S_1, \ldots, S_{n_1}$ is closed under reverse transitions of $\mathcal{A}$ according to the *and/or* semantics. Thus, $\varphi_{\mathcal{A}}$ states that the initial configuration of $\mathcal{A}$ is accessible in the *and/or* graph $G_{\mathcal{A},w}$. It follows that $\varphi_{\mathcal{A}}$ holds iff $\mathcal{A}$ accepts w.

The formula *reverse-closed* is a direct representation of the transitions in $\mathcal{A}$ (and edges in $G_{\mathcal{A},w}$) in MSO*. For each transition $(a, q_u) \rightarrow \beta$ of $\mathcal{A}$ where q_u is existential, *reverse-closed* includes one conjunct. For example, if $\beta = (q_v, \mathit{right})$, the corresponding conjunct is

$$\forall x \forall y((\mathit{val}(x) = a \wedge \mathit{succ}(x,y) \wedge S_v(y)) \rightarrow S_u(x)),$$

where $\mathit{succ}(x,y)$ is the FO* formula defining the successor relation on $\mathit{dom}(w)$. If q_u is universal, $a \in \mathbf{D}$, and q_u's transitions under a are $(a, q_u) \rightarrow (q_v, \mathit{stay})$ and $(a, q_u) \rightarrow (q_s, \mathit{stay})$, *reverse-closed* includes the conjunct

$$\forall x((\mathit{val}(x) = a \wedge S_v(x) \wedge S_s(x)) \rightarrow S_u(x)).$$

Finally, for every final state q, *reverse-closed* contains the conjunct $\forall x S_q(x)$. This captures the fact that all accepting configurations (q, x) are accessible in the graph $G_{\mathcal{A},w}$. In general, we denote the conjunct corresponding to a transition τ by ψ_τ.

To see that $\varphi_{\mathcal{A}}$ holds on w iff $\mathcal{A}$ accepts w,[4] it suffices to notice the similarity between (1) and (3). To this end, it is sufficient to observe how the formula for *reverse-closed* in a general graph (2) becomes the formula above when instantiated to $G_{\mathcal{A},w}$. For example, notice that each *and*-node in $G_{\mathcal{A},w}$ has zero or two successors (hence the universal quantifier in (2) becomes a conjunction).

We now extend (3) to the case when k is arbitrary. Recall that, at each moment in the computation of $\mathcal{A}$, some pebble i is active, and the transitions depend on the location of the first $i-1$ pebbles and the equality type of the corresponding values. The stack discipline on pebbles ensures that once pebble i is placed, the first $i-1$ pebbles are fixed until pebble i is lifted. This is the key technical point in the construction. We define, for each $i = 1, \ldots, k$, a predicate *reverse-closed* $^{(i)}(x_1, \ldots, x_{i-1})$ stating that $S_{n_{i-1}+1}, \ldots, S_{n_i}$ are closed under reverse transitions of $\mathcal{A}$ that involve only pebbles i or higher (excluding placing and lifting pebble i), assuming that the first $i-1$ pebbles are placed on the nodes represented by the free variables $x_1, \ldots, x_{i-1}$. Apart from the free first-order variables $x_1, \ldots, x_{i-1}$, *reverse-closed* $^{(i)}(x_1, \ldots, x_{i-1})$ also has $S_{n_{i-1}+1}, \ldots, S_{n_i}$ as free second-order variables. Note that, for $i = k$, the formula describes only transitions involving pebble k; for $i < k$, the formula *reverse-closed* $^{(i)}(x_1, \ldots, x_{i-1})$ must take into account sub-computations where pebble $i+1$ is placed and lifted. Thus, the formula makes use of *reverse-closed* $^{(i+1)}(x_1, \ldots, x_i)$. We therefore define the formulas *reverse-closed* $^{(i)}(x_1, \ldots, x_{i-1})$ by backward induction, starting with $i = k$ and ending with $i = 1$. Then, the MSO* formula equivalent to $\mathcal{A}$ will be φ_A in (3), with *reverse-closed* replaced with *reverse-closed* $^{(1)}$.

We next give the inductive definition of *reverse-closed* $^{(i)}(x_1, \ldots, x_{i-1})$. For $i = k$, *reverse-closed* $^{(k)}(x_1, \ldots, x_{k-1})$ is very similar to the formula *reverse-closed* for the one-pebble case described earlier, except that now it also inspects the presence/absence of the previous $k-1$ pebbles and tests equality/inequality of the corresponding values with the current value. For example, for $k = 3$ and

[4]We ignore end-markers here, since it is clear that a language L is definable in MSO* iff $\{\triangleright\, v \,\triangleleft \mid v \in L\}$ is definable in MSO*.

$\tau = ((3, a, \{1\}, \{2\}, q_u) \rightarrow (q_v, \textit{stay}))$ the corresponding conjunct in

$$\textit{reverse-closed}^{(k)}(x_1, \ldots, x_{k-1})$$

is:

$$\forall x_k[(\textit{val}(x_k) = a \wedge x_k = x_1 \wedge x_k \neq x_2 \wedge \textit{val}(x_k) \neq \textit{val}(x_1) \wedge \textit{val}(x_k) = \textit{val}(x_2) \wedge S_v(x_k)) \rightarrow S_u(x_k)]$$

In general, for a transition $(k, s, P, V, q) \rightarrow \beta$ we use the formulas

$$\xi_P(x_1, \ldots, x_k) := \bigwedge_{j \in P} (x_k = x_j) \wedge \bigwedge_{j \notin P} (x_k \neq x_j),$$

and

$$\psi_V(x_1, \ldots, x_k) := \bigwedge_{j \in V} (\textit{val}(x_k) = \textit{val}(x_j)) \wedge \bigwedge_{j \notin V} (\textit{val}(x_k) \neq \textit{val}(x_j)).$$

Finally, for every final state q_j where $n_{k-1} + 1 \leq j \leq n_k$, we add the conjunct $\forall x S_j(x)$.

Now suppose $i < k$ and the formulas $\textit{reverse-closed}^{(j)}(x_1, \ldots, x_{j-1})$ have been defined for $i < j \leq k$. Let T_i be the set of transitions from states in Q_i, that do not involve lifting pebble i. As before, to each transition τ corresponds a conjunct ψ_τ. Then,

$$\textit{reverse-closed}^{(i)} := \bigwedge_{\tau \in T_i} \psi_\tau.$$

For transitions τ not placing pebble $i+1$, ψ_τ is the same as for $\textit{reverse-closed}^{(k)}$, with $k = i$, including the respective conjunct for every final state. The new transitions are the *place-new-pebble* transitions. These determine the following conjuncts in $\textit{reverse-closed}^{(i)}$. For each transition $\tau = (i, s, P, V, q_u) \rightarrow (q_v, \textit{place-new-pebble})$ (where $q_u \in Q_i$ and $q_v \in Q_{i+1}$),

$$\psi_\tau = \forall x_i((\textit{val}(x_i) = s \;\wedge\; \xi_P(x_1, \ldots, x_i) \;\wedge\; \psi_V(x_1, \ldots, x_i) \;\wedge\; \varphi_v^{(i+1)}) \Longrightarrow S_u(x_i)),$$

where the formula $\varphi_v^{(i+1)}$ is explained next. Intuitively, $\varphi_v^{(i+1)}$ states that $S_v(x_i)$ is reachable, in the *and/or* graph of reverse transitions among configurations involving at least $i+1$ pebbles, from final configurations and from the configurations of depth i specified by $S_{n_{i-1}+1}, \ldots S_{n_i}$ reached by lifting pebble $i+1$. Specifically,

$$\varphi_v^{(i+1)} = \forall S_{n_i+1} \ldots \forall S_{n_{i+1}}[(\textit{lift}^{(i+1)} \wedge \textit{reverse-closed}^{(i+1)}) \Longrightarrow S_v(x_i)],$$

where $\textit{lift}^{(i+1)}$ states that $S_{n_i+1} \ldots S_{n_{i+1}}$ are closed under reverse transitions in which pebble $i+1$ is lifted, from configurations of depth i specified by $S_{n_{i-1}+1}, \ldots S_{n_i}$. More precisely, for each transition

$$\tau = (i+1, s, P, V, q_w) \rightarrow (q_z, \textit{lift-current-pebble})$$

where $q_w \in Q_{i+1}$ and $q_z \in Q_i$, $\textit{lift}^{(i+1)}$ contains the conjunct ψ_τ:

$$\forall x_{i+1}((\textit{val}(x_{i+1}) = s \wedge \xi_P(x_1, \ldots, x_{i+1}) \wedge \psi_V(x_1, \ldots, x_{i+1}) \wedge S_z(x_i)) \rightarrow S_w(x_{i+1})).$$

Note that here q_w acts as a terminal state for pebble $i+1$. Once pebble $i+1$ is lifted, pebble i becomes active starting from state q_z.

Note the resemblance of $\varphi_v^{(i+1)}$ to (3): here $q_v \in Q_{i+1}$ acts as an initial state for pebble $i+1$ (which is first placed on x_i according to the definition of *place-new-pebble* moves). This completes the inductive definition of the formulas *reverse-closed* $^{(i)}(x_1, \ldots, x_{i-1})$, and the translation of $\mathcal{A}$ into MSO*.

To prove that the constructed formula defines acceptance by $\mathcal{A}$, it is enough to verify that *reverse-closed* $^{(i)}$ works as promised. More precisely, fix i, let θ be an assignment $x_1, \ldots, x_{i-1}$ for the first $i-1$ pebbles, and let $G^\theta_{\mathcal{A},w}$ be the subgraph of $G_{\mathcal{A},w}$ whose nodes are the configurations of $\mathcal{A}$ with at least i pebbles and where θ is the assignment for the first $i-1$ pebbles. It can be shown by induction on the definition of *reverse-closed* $^{(i)}$ (from k down to 1) that $S_{n_{i-1}+1}, \ldots, S_{n_i}$ satisfy *reverse-closed* $^{(i)}(\theta)$ iff they are closed under reverse transitions involving pebbles i or higher (excluding lifting pebble i), and θ is the assignment of the first $i-1$ pebbles. More precisely, consider the set of configurations

$$\mathcal{C}_\theta \;=\; \{[i, q_j, (\theta, x_i)] \mid S_j(x_i), n_{i-1}+1 \leq j \leq n_i\}.$$

Every configuration of depth i that is accessible in $G^\theta_{\mathcal{A},w}$ under *and/or* semantics from $\mathcal{C}_\theta$ together with the set of accepting configurations, belongs already to $\mathcal{C}_\theta$. It immediately follows that the formula

$$\forall S_0 \forall S_1 \ldots \forall S_{n_1} (\textit{reverse-closed}^{\,(1)} \rightarrow S_0(0)) \tag{4}$$

states that the initial configuration is accessible in $G_{\mathcal{A},w}$ (from the set of accepting configurations). This completes the proof of the theorem. Note that the stack discipline imposed on the use of pebbles is essential to the construction. □

Example 4.3. We illustrate the construction in the proof of Theorem 4.2 using the 2N-PA of Example 2.5. Note that in that example, $Q_1 = \{q_0, q_1, q_2\}$, $Q_2 = \{q_3, q_4, q_5\}$, $Q_3 = \{q_6, q_7\}$. We next exhibit the formulas

$$\textit{reverse-closed}^{\,(i)}(x_1, \ldots, x_{i-1})$$

constructed in the proof of Theorem 4.2, for $i = 3, 2, 1$. The formula

$$\textit{reverse-closed}^{\,(3)}(x_1, x_2)$$

is the following (one conjunct for each of the six transitions involving pebble 3 that do not lift the pebble):

$$\begin{aligned}
&\forall x_3 \forall x'_3[(succ(x_3, x'_3) \wedge S_6(x'_3) \wedge x_3 = x_2 \wedge x_3 \neq x_1 \wedge \\
&\qquad val(x_3) = val(x_2) \wedge val(x_3) \neq val(x_1)) \rightarrow S_6(x_3)] \wedge \\
&\forall x_3 \forall x'_3[(succ(x_3, x'_3) \wedge S_6(x'_3) \wedge x_3 \neq x_2 \wedge x_3 \neq x_1 \wedge \\
&\qquad val(x_3) \neq val(x_2) \wedge val(x_3) \neq val(x_1)) \rightarrow S_6(x_3)] \wedge \\
&\forall x_3 \forall x'_3[(succ(x_3, x'_3) \wedge S_6(x'_3) \wedge x_3 \neq x_2 \wedge x_3 \neq x_1 \wedge \\
&\qquad val(x_3) = val(x_2) \wedge val(x_3) \neq val(x_1)) \rightarrow S_6(x_3)] \wedge \\
&\forall x_3 \forall x'_3[(succ(x_3, x'_3) \wedge S_7(x'_3) \wedge x_3 \neq x_2 \wedge x_3 = x_1 \wedge \\
&\qquad val(x_3) \neq val(x_2) \wedge val(x_3) = val(x_1)) \rightarrow S_6(x_3)] \wedge \\
&\forall x_3 \forall x'_3[(succ(x_3, x'_3) \wedge S_7(x'_3) \wedge x_3 \neq x_2 \wedge x_3 \neq x_1 \wedge \\
&\qquad val(x_3) \neq val(x_2) \wedge val(x_3) = val(x_1)) \rightarrow S_7(x_3)] \wedge \\
&\forall x_3 \forall x'_3[(succ(x_3, x'_3) \wedge S_7(x'_3) \wedge x_3 \neq x_2 \wedge x_3 \neq x_1 \wedge val(x_3) \neq \lhd \wedge \\
&\qquad val(x_3) \neq val(x_2) \wedge val(x_3) \neq val(x_1)) \rightarrow S_7(x_3)]
\end{aligned}$$

The formula *reverse-closed* $^{(2)}(x_1)$ uses the above. Again, *reverse-closed* $^{(2)}(x_1)$ contains one conjunct for each transition involving pebble 2, and in addition the conjunct $\forall x_2 S_5(x_2)$ because q_5 is a final state:

$$\begin{aligned}
&\forall x_2 S_5(x_2) \wedge \\
&\forall x_2 \forall x'_2[(succ(x'_2, x_2) \wedge S_4(x'_2) \wedge x_2 = x_1 \wedge val(x_2) = val(x_1)) \rightarrow S_3(x_2)] \wedge \\
&\forall x_2 \forall x'_2[(succ(x'_2, x_2) \wedge S_4(x'_2) \wedge x_2 \neq x_1 \wedge val(x_2) \neq val(x_1)) \rightarrow S_3(x_2)] \wedge \\
&\forall x_2[(S_5(x_2) \wedge x_2 \neq x_1 \wedge val(x_2) \neq val(x_1) \wedge val(x_2) = \rhd) \rightarrow S_4(x_2)] \wedge \\
&\forall x_2[(x_2 \neq x_1 \wedge val(x_2) \neq val(x_1) \wedge val(x_2) \neq \rhd \wedge \\
&\forall S_6 \forall S_7((\mathit{lift}^{(3)}(x_1, x_2) \wedge \textit{reverse-closed}\ ^{(3)}(x_1, x_2)) \rightarrow S_6(x_2))) \rightarrow S_4(x_2)]
\end{aligned}$$

where $\mathit{lift}^{(3)}(x_1, x_2)$ is the formula:

$$\begin{aligned}
&\forall x_3[(S_3(x_2) \wedge x_3 \neq x_2 \wedge x_3 \neq x_1 \wedge val(x_3) = \lhd \wedge \\
&\qquad\qquad val(x_3) \neq val(x_2) \wedge val(x_3) \neq val(x_1)) \rightarrow S_7(x_3)]
\end{aligned}$$

Finally, *reverse-closed* $^{(1)}$ is the formula:

$$\begin{aligned}
&\forall x_1 \forall x'_1[(succ(x_1, x'_1) \wedge S_0(x'_1) \wedge val(x_1) = \rhd) \rightarrow S_0(x_1)] \wedge \\
&\forall x_1 \forall x'_1[(succ(x_1, x'_1) \wedge S_1(x'_1) \wedge val(x_1) \neq \rhd) \rightarrow S_0(x_1)] \wedge \\
&\forall x_1[(S_2(x_1) \wedge val(x_1) = \lhd) \rightarrow S_1(x_1)] \wedge \\
&\forall x_1 \forall x'_1[(succ(x_1, x'_1) \wedge S_1(x'_1) \wedge val(x_1) \neq \lhd) \rightarrow S_1(x_1)] \wedge \\
&\forall x_1[(val(x_1) \neq \lhd \ \wedge\ \forall S_3 \forall S_4 \forall S_5(\textit{reverse-closed}\ ^{(2)}(x_1) \rightarrow S_3(x_1))) \rightarrow S_1(x_1)].
\end{aligned}$$

The formula corresponding to the pebble automaton is now

$$\forall S_0 \forall S_1 \forall S_2[\textit{reverse-closed}\ ^{(1)} \rightarrow S_0(0)].$$

This completes the construction. □

It is open whether the inclusion of pebble automata in MSO* is strict. However, there is strong evidence that it is. Let $G = (\{1, \ldots, n\}, E)$ be a graph and let $\overline{d} = d_1, \ldots, d_n$ be a sequence of distinct elements of G. Denote by α_i the string $\# d_i\, d_{i_1} \ldots d_{i_k}$ where $i \in \{1, \ldots, n\}$ and $\{i_1, \ldots, i_k\} = \{j \mid G \models E(i, j)\}$. Finally, let $w(G, \overline{d})$ be the string $\alpha_1 \alpha_2 \cdots \alpha_n$. For a set L of graphs and an infinite alphabet $\mathbf{D}$, define $f_{\mathbf{D}}(L) := \{w(G, \overline{d}) \mid G \in L, \overline{d} \in \mathbf{D}\}$.

We show the following.

THEOREM 4.4. *For every $i \in \mathbb{N}$, there are sets L_i and L'_i of graphs that are hard for Σ_i^P and Π_i^P, respectively, such that $f_{\mathbf{D}}(L_i)$ and $f_{\mathbf{D}}(L')$ can be expressed by MSO* formulas.*

In contrast, for a set of graphs L, $f_{\mathbf{D}}(L)$ is in 2A-PA only if L is in PTIME.

PROOF. Ajtai, Fagin, and Stockmeyer [Ajtai et al. 2000] showed that for every level of the polynomial hierarchy (PH) there is an MSO formula over graphs such that model checking is hard for that level. We describe a translation from MSO formulas φ to MSO* formulas φ^* such that $G \models \varphi$ if and only if $w(G, \overline{d}) \models \varphi^*$.

To this end, let vertex(x) be the formula $(\exists z')(\text{val}(z') = \# \wedge \text{succ}(z') = x)$. Then φ^* is obtained from φ by replacing every occurrence of $E(x, y)$ by

$$\exists y'(\text{val}(y') = \text{val}(y) \wedge x < y' \wedge \neg\exists z(x < z \wedge z < y \wedge \text{val}(z) = \#).$$

Then inductively replace from the inside to the outside every occurrence of a subformula $\exists x\alpha$, $\forall x\alpha$, $\exists X\alpha$, and $\forall X\alpha$, by $\exists x(\text{vertex}(x) \wedge \alpha)$, $\forall x(\text{vertex}(x) \rightarrow \alpha)$, $\exists X(\forall x'(X(x') \rightarrow \text{vertex}(x')) \wedge \alpha)$, and $\forall X(\forall x'(X(x') \rightarrow \text{vertex}(x')) \rightarrow \alpha)$. This completes the description of the translation.

For the latter statement of the theorem, it is easy to see that for each 2A-PA $\mathcal{A}$ there is an alternating Turing machine M such that, for each G and $\overline{d}$, the computation of $\mathcal{A}$ on input $w(G, \overline{d})$ is simulated by M if it gets as input an encoding of $w(G, \overline{d})$, where each d_i is represented by [binary representation of i]. Furthermore, M works in logarithmic space. Therefore $f_{\mathbf{D}}(L) \in$ 2A-PA would imply $L \in$ ALOGSPACE. Hence, the statement follows as ALOGSPACE equals PTIME. □

Theorem 4.1 and Theorem 4.2 show that PAs fall nicely in between FO* and MSO*. In the next subsection we show that, determinism, nondeterminism, one- and two-way coincide for strong PAs. It is open whether alternating control yields additional power.

We end this subsection by considering weak PAs. Recall that weak PAs place new pebbles at the location of the current head rather than the beginning of the string. Clearly, this only makes a difference for one-way models. Unlike their strong counterparts, we show that weak PAs cannot simulate FO*, which justifies their name. The proof is once again based on communication complexity. We show that the language $L^2_=$, defined in the proof of Theorem 3.7, cannot be computed by weak 1N-PAs. However, we use a different kind of communication protocol which better reflects the behavior of a weak 1N-PA.

THEOREM 4.5. *FO* $\not\subseteq$ weak 1N-PA.*

PROOF. The proof is similar to that of Theorem 3.7. We show by a communication complexity argument that the FO*-definable language $L^2_=$ defined in the proof of Theorem 3.7 cannot be recognized by a weak 1N-PA. Recall that $L^2_=$ consists of strings of the form $u\#v$ where u and v encode the same 2-hyperset. Let k be fixed and let S_1, S_2 be fixed finite sets. The new protocol has only one agent which has arbitrary access to the string u but only limited access to the string v. On u, its computational power is unlimited. The access to v is restricted as follows. Let $\perp \notin \mathbf{D}$. There is a function $f : (\mathbf{D} \cup \{\perp\})^k \times S_1 \rightarrow S_2$ depending on u called the advice function for v. The agent is allowed to evaluate f on all arguments $(\bar{d}, s)$, where $\bar{d}$ is a tuple of length k of symbols from u or $\perp$, and $s \in S_1$. Based on this information and on u the agent decides whether $u\#v$ is accepted.

First, we show that for no function f there is an agent which recognizes $L^2_=$. Towards a contradiction assume otherwise. Let S_1 and S_2 be the corresponding finite sets. Let D be a fixed finite set and let $u\#v$ be a string with $u \in D^*$ and $v \in \mathbf{D}^*$. Further let f be the advice function for v. We set $d := |D|+1$, $m_1 := |S_1|$, and $m_2 := |S_2|$. We can assume w.l.o.g. that the agent always evaluates f for all possible arguments. As there are at most $d^k m_1$ such arguments there are at most $m_2^{d^k m_1}$ different "interactions" between the agent and function f. It is important here that the protocol is non-adaptive, i.e., the order of the questions does not matter. Let $h = 2^{2^{d-2}}$. If d is large enough with respect to k, m_1, and m_2 then there are more 2-hypersets on $D - \{1, 2\}$ than different interactions. Hence, there must be encodings u, u' of 2-hypersets with $H(u) \neq H(u')$ such that the interactions on $u\#u$ and $u'\#u'$ are the same. Hence, the agent accepts $u\#u$ if and only if he accepts $u\#u'$, which is a contradiction.

It remains to show that on strings $u\#v$ that encode pairs of 2-hypersets a weak 1N-PA $\mathcal{A} = (Q, q_0, F, T)$ with k pebbles can be simulated by a protocol. Intuitively, this works as follows. We take $S_1 = \{(i, q) \mid i \leq k, q \in Q\}$ and $S_2 = 2^Q$. On input $u\#v$, as we consider *one*-way *weak* PAs, whenever the current pebble enters v, the computation remains in v until that pebble is lifted. Further, at this time there are no other pebbles placed on v. Therefore, the set of states which can be obtained when lifting the pebble only depends on v and the symbols below the pebbles placed in u. Hence, we define $f(\bar{d}, (i, q))$ as the set of states that can be reached when pebble i enters v in state q and the pebbles in u are placed on symbols $\bar{d}$ (with $\perp$ indicating that a pebble is not present). This function provides enough information for the agent to simulate the 1N-PA. □

4.2 Control

Our next result shows that all variants of strong pebble automata without alternation collapse. This suggests that strong PAs provide a robust automaton model.

THEOREM 4.6. *The following have the same expressive power: 2N-PA, 2D-PA, strong 1N-PA and strong 1D-PA.*

PROOF. We show that, for each 2N-PA $\mathcal{A}$, there is a strong 1D-PA $\mathcal{B}$ that accepts the same language. Actually, in our construction $\mathcal{B}$ uses the same number of pebbles as $\mathcal{A}$. Therefore, let $\mathcal{A} = (Q, q_0, F, T)$ be a 2N-PA with k pebbles.

For technical simplicity, we assume w.l.o.g. that $\mathcal{A}$ lifts pebbles only at the right delimiter (instead of lifting a pebble at an arbitrary position it can remember the target state q, go to the right delimiter and lift the pebble there, moving into state q).

First, we informally describe the idea of the construction. Recall the classical powerset construction which translates a nondeterministic 1-way automaton M (over a finite alphabet and with one pebble) into a deterministic one, M'. Intuitively, M' computes, for each prefix u of the input string $w = w_1 \cdots w_n$, the set of states that M might reach by reading u. M' performs an on-line simulation of

M in the sense that each step in the computation of M corresponds to exactly one step of M'.

One cannot expect such an on-line simulation to work for 2-way automata (even for finite alphabets), as the nondeterministic behavior of a 2-way automaton might involve moving in different directions. Instead (in the finite case with one pebble) the deterministic automaton can compute, for each position i in the input string $w = w_1 \cdots w_n$, a binary relation R_i and a set Q_i, which describes the aggregate behavior of M on $w_1 \cdots w_i$. More precisely, R_i contains all pairs (p, q) such that M might leave $w_1 \cdots w_i$ (to the right) in state q when it enters it (from th right) in state p. Further, Q_i contains all states in which M might leave $w_1 \cdots w_i$ (to the right) when started at w_1 in the initial state. Note that R_i and Q_i can be computed inductively from left to right (forgetting R_{i-1} and Q_{i-1} once R_i and Q_i are computed). In the end, R_n and Q_n provide all necessary information to decide whether w is accepted. This construction was first presented by Shepherdson [Shepherdson 1959].

It is maybe a bit surprising that this approach can, by and large, be adapted to the case where pebbles are present and the alphabet is infinite. We proceed as follows. First, we assume that $\mathcal{A}$ is further normalized in that it accepts its input only in configurations $[1, q, \theta]$, i.e., with only one pebble. By virtually adding two steps we view an accepting computation as consisting of (1) a first step in which the first pebble is placed at the left delimiter, i.e., position 0, (2) a computation in which always at least one pebble is present, and (3) a final step in which the only remaining pebble is removed. Writing $[0, p, \theta_\emptyset]$ for a (virtual) configuration without pebble, to determine whether $\mathcal{A}$ accepts, one has to find out whether $[0, q_0, \theta_\emptyset] \vdash^* [0, q, \theta_\emptyset]$, for some final state q.

The latter can be done by recursively solving subproblems of the form $[i, q, \theta] \vdash^*_{>i} [i, q', \theta]$, where the subscript $> i$ indicates that only subcomputations are considered in which, at every step, more than i pebbles are present.

More formally, we show the following claim by induction on i (starting from $i = k$).

Claim 4.7. For each $i \in \{0, \ldots, k\}$ and each finite set R, there is a strong 1D-PA $\mathcal{B}_i$ (with k pebbles) such that, whenever $\mathcal{B}_i$ starts from a configuration $[i, p, \theta]$, where $p \in R$, the next configuration of depth i of $\mathcal{B}_i$ is $[i, (p, S), \theta]$, where $S = \{(q, q') \in Q \times Q \mid [i, q, \theta] \vdash^*_{>i} [i, q', \theta]\}$.

In particular, the set of states of $\mathcal{B}_i$ contains R and $R \times 2^{Q \times Q}$.

Before the claim is proved, it should be noted that the theorem indeed follows from the claim. To this end, we set $i = 0$ and let $R = \{p_0\}$ (the intended initial state of $\mathcal{B}_0$) and obtain an automaton which ends up in a state (p_0, S), where S is the set $\{(q, q') \in Q \times Q \mid [0, q, \theta_\emptyset] \vdash^*_{>0} [0, q', \theta_\emptyset]\}$. The set of final states of $\mathcal{B}_0$ simply consists of all states (p_0, S), where S contains a pair (q_0, q) with $q \in F$.

For $i = k$ the proof of the claim is trivial, as there are no configurations of depth $> k$. Hence, $\mathcal{B}_k$ can compute (p, S) by a stay-transition. Therefore, let $i < k$ and suppose the claim holds for all $j > i$.

Intuitively, the set S can be computed by one left-to-right pass of the $(i + 1)$st pebble, i.e., $\mathcal{B}_i$ places pebble $i + 1$ moves it from left to right through the string and lifts it at the right delimiter. During this pass, $\mathcal{B}_i$ computes, for each position l in the string the sets of pairs (q, q'), such that there is a sub-computation which starts from state q at position l and ends in state q' at position l, without moving pebble $i+1$ to positions $l' \geq l$. Note that such subcomputations might move pebbles $j > i+1$ to positions $\geq l$. To compute this information, the automaton $\mathcal{B}_{i+1}$ is used repeatedly.

We first introduce some more notation. Let the input string w of length n be fixed. Hence $\mathcal{B}_i$ works on the string $\rhd\, w \lhd$ with positions from $\mathrm{dom}^+(w)$. For $l \leq n+1$, let θ^l denote the $(i+1)$-pebble assignment that coincides with θ in the first i pebbles and for which $\theta^l(i+1) = l$. We write $S_{\leftrightharpoons}(\theta^l)$ for the set of pairs (q, q') of states such that there is a computation starting at $[i+1, q, \theta^l]$ and reaching $[i+1, q', \theta^l]$ which only includes configurations $[j, q'', \theta']$ that fulfill $j > i+1$ or ($j = i+1$ and $\theta'(i+1) \leq l$). Intuitively, this says that pebble $i+1$ is not allowed to move to the right of position l. We write $S_{\rightharpoonup}(\theta^l)$ for the set of pairs (q, q') of states for which $[i+1, q', \theta^l]$ can be reached from $[i+1, q, \theta^1]$ by a subcomputation satisfying the same property, i.e., including only configurations $[j, q'', \theta']$ that fulfill $j > i+1$ or ($j = i+1$ and $\theta'(i+1) \leq l$).

We are now ready to complete the proof of the claim. The set $S_{\leftrightharpoons}(\theta^l)$ can be computed as follows. Let $R_{\leftrightharpoons}(\theta^l)$ be the set of pairs (q, q') for which one of the following holds:

(a) There exist p_1, p_2 such that $[i+1, q, \theta^l] \vdash [i+1, p_1, \theta^{l-1}]$, $(p_1, p_2) \in S_{\leftrightharpoons}(\theta^{l-1})$, and $[i+1, p_2, \theta^{l-1}] \vdash [i+1, q', \theta^l]$;

(b) $[i+1, q, \theta^l] \vdash^*_{>i+1} [i+1, q', \theta^l]$;

(c) $[i+1, q, \theta^l] \vdash [i+1, q', \theta^l]$.

It is straightforward to see that $S_{\leftrightharpoons}(\theta^l)$ is simply the transitive closure of $R_{\leftrightharpoons}(\theta^l)$. They are computed simultaneously. The information needed for (a) can be computed in one left-to-right pass of pebble $i+1$. By induction we can assume a subautomaton $\mathcal{B}_{i+1}$ that computes, for each position l, the part of $R_{\leftrightharpoons}(\theta^l)$ contributed by condition (b). Note that (c) and the computation of the transitive closure do not require any pebble movements. During the same pass, the automaton can compute, for each position l, the set $S_{\rightharpoonup}(\theta^l)$. The computation of $S_{\rightharpoonup}(\theta^l)$ makes use of the sets $S_{\leftrightharpoons}(\theta^m), m \leq l$.

From $S_{\leftrightharpoons}(\theta^{n+1})$, $S_{\rightharpoonup}(\theta^{n+1})$ and the transition relation of $\mathcal{A}$ one can deduce, during a lift-pebble step, the set S as in the claim. Note that $n+1$ is the position of the right delimiter and recall that $\mathcal{A}$ lifts its pebbles only at that position.

This completes the proof of the claim and of the theorem. □

4.3 Registers versus Pebbles

The known inclusions between the classes that we considered are depicted in Figure 1. The pebble and register models are rather incomparable. Indeed, from the connection with logic we can deduce the following. As 2D-RA can already express non-MSO* definable properties, no two-way register model is subsumed by a pebble model. Conversely, as strong 1D-PAs can already express FO*, no strong pebble model is subsumed by any register model. Some open problems about the relationships between register and pebble automata are given in Section 6.

5. DECISION PROBLEMS

We discuss the standard decision problems for RAs and PAs. Kaminski and Francez already showed that emptiness of 1N-RAs is decidable and that it is decidable whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ for a 1N-RA $\mathcal{A}$ and a 1N-RA $\mathcal{B}$ with 2 registers. We next show that *universality* (does an automaton accept every string) of 1N-RAs is undecidable, which implies that equivalence (and hence containment) of *arbitrary* 1N-RAs is undecidable. Kaminski and Francez further asked whether the decidability of non-emptiness can be extended to 2D-RAs: we show it cannot. Regarding PAs, we show that non-emptiness is already undecidable for weak 1D-PAs. This is due to the fact that, as already noted in Section 2, even weak 1D-PAs can make several left-to-right sweeps of the input string.

In our proofs, we use a reduction from the Post Correspondence Problem (PCP) which is well known to be undecidable [Hopcroft and Ullman 1979]. An *instance* of PCP is a sequence of pairs $(x_1, y_1), \ldots, (x_n, y_n)$, where $x_i, y_i \in \{a, b\}^*$ for $i = 1, \ldots, n$. This instance has a *solution* if there are $m \in \mathbb{N}$ and $\alpha_1, \ldots, \alpha_m \in \{1, \ldots, n\}$ such that $x_{\alpha_1} \cdots x_{\alpha_m} = y_{\alpha_1} \cdots y_{\alpha_m}$. The PCP asks whether a given instance of the problem has a solution.

Suppose w.l.o.g. that the integer numbers $\{1, \ldots, n\}$ and the values $a, b, \&, \#$ are in $\mathbf{D}$. Denote the latter set of symbols by Sym. We consider input strings of the form $w = u\#v$, where $\#$ is a delimiter and u and v are strings representing a candidate solution $(x_{\alpha_1}, \ldots, x_{\alpha_m}; y_{\beta_1}, \ldots, y_{\beta_l})$ for the PCP instance in a suitable way, u representing the x's and v the y's.

To check whether such a candidate is indeed a solution, we have to check whether

(1) $l = m$ and, for each i, $\alpha_i = \beta_i$, that is, corresponding pairs are taken; and

(2) both strings are the same, that is, corresponding positions in $x_{\alpha_1} \cdots x_{\alpha_m}$ and $y_{\alpha_1} \cdots y_{\alpha_m}$ carry the same symbol.

To check (1) and (2), we use a double indexing system based on unique data values.

We describe the encoding in more detail. Each item x_{α_j} is encoded as a string of the form $\&\gamma\, \alpha_j\ \delta_1 a_1 \cdots \delta_k a_k$. Here, $\&$ is a separator, $\gamma \in \mathbf{D} - \mathrm{Sym}$ represents j by a unique data value, the a_i are from $\{a, b\}$ such that $x_{\alpha_j} = a_1 \cdots a_k$ and the δ_i represent the position of a_i in x by a unique data value. To achieve uniqueness, all γ- and δ-symbols are allowed to occur only once in u. Correspondingly, y_{β_j} is encoded by a string of the form $\&\gamma\, \beta_j\ \delta_1 a_1 \cdots \delta_k a_k$ such that $y_{\beta_j} = a_1 \cdots a_k$ and the corresponding conditions hold. A string $u\#v$ is *syntactically correct* if it has the properties described so far and fulfills the following two conditions:

—the γ-projection of u (i.e., the string consisting of the γ-entries of u) equals the γ-projection of v; and

—the δ-projection of u equals the δ-projection of v.

A syntactically correct string $u\#v$ *represents a solution of the PCP instance*, if,

—for each γ, the number to the right of γ is the same in u and in v, i.e., $\gamma\alpha_j = \gamma\beta_j$, and

—for each δ, the symbol from $\{a, b\}$ at the right of δ is the same in u and in v.

We next show that universality of 1N-RAs and emptiness of weak 1D-PAs are undecidable. In the former case, we construct an 1N-RA $\mathcal{A}$ that accepts an input string w if and only if it is *not* syntactically correct or does *not* represent a solution. Hence, $\mathcal{A}$ accepts *all* inputs if and only if the PCP instance has *no* solution. In the construction nondeterminism comes into play. The automaton simply tries to guess an error in the encoding represented by the input string.

When showing undecidability of emptiness of weak 1D-PAs, we construct an automaton that checks whether no error occurs in the encoding represented by the input string. This happens by doing several sweeps over the input string.

5.1 Register Automata

THEOREM 5.1. *It is undecidable whether a 1N-RA is universal.*

PROOF. The initial register assignment assigns the values in Sym to the first $n+5$ registers.

We construct an 1N-RA $\mathcal{A}$ that accepts an input string w if and only if it is *not* syntactically correct or does *not* represent a solution. Hence, $\mathcal{A}$ accepts *all* inputs if and only if the PCP instance has *no* solution. Hence, $\mathcal{A}$ checks that one of the following conditions holds for its input string w. Note that each single condition is easy to check and the class of languages definable by 1N-RAs is closed under union.

(1) w is of the wrong form.
   (a) w is not of the form $u\#v$ or u or v is not of the form $(\&\gamma i\ \delta_1 a_1 \cdots \delta_k a_k)^*$, $(i \in \{1,\dots,n\})$, $a_j \in \{a,b\}$, for all $j \leq k$.
   (b) $x_i \neq a_1 \cdots a_k$ in some entry $\&\gamma i\ \delta_1 a_1 \cdots \delta_k a_k$ in u or $y_i \neq a_1 \cdots a_k$ in some entry in v.

(2) The γ-projections are wrong.
   (a) the first γ in u differs from the first γ in v;
   (b) the last γ in u differs from the last γ in v;
   (c) two γ's in u are the same;
   (d) two γ's in v are the same; or
   (e) γ_1 and γ_2 are successors in u but not in v.

   The latter three conditions involve nondeterministic guesses of the positions where the failure takes place.

(3) The δ-projections are wrong. This can be done in a completely analogous fashion.

(4) w does not represent a solution:
   (a) The α-value for some γ in u is different from the corresponding β-value in v.
   (b) The a/b-value for some δ in u is different from the corresponding a/b-value in v.

Clearly, w is not a solution iff one of these conditions holds. □

COROLLARY 5.2. *Equivalence of 1N-RAs is undecidable.*

The next question was also raised by Kaminski and Francez. In Section 3.2, we observed that two-way RAs can simulate multi-head automata on strings of a special shape. As emptiness of multi-head automata is undecidable the next theorem easily follows.

THEOREM 5.3. *It is undecidable for a 2D-RA $\mathcal{A}$ whether $L(\mathcal{A}) = \emptyset$.*

5.2 Pebble Automata

The next result implies that standard decision problems like equivalence and containment are undecidable for all classes of pebble automata.

THEOREM 5.4. *It is undecidable for a weak 1D-PA $\mathcal{A}$ whether $L(\mathcal{A}) = \emptyset$.*

PROOF. The weak 1D-PA $\mathcal{A}$ first checks whether the input is of the desired form and then accepts if the input encodes a solution of the PCP instance. As pebbles can only be moved to the right, we keep the first pebble on the first position and invoke subroutines at that position which are then performed by the other pebbles. $\mathcal{A}$ operates as follows.

(1) $\mathcal{A}$ checks whether u and v are of the form $(\&\gamma\, i\, \delta_1 a_1 \cdots \delta_k a_k)^*$ and that $x_i = a_1 \cdots a_k$ and $y_i = a_1 \cdots a_k$, respectively, for all entries $\&\gamma\, i\, \delta_1 a_1 \cdots \delta_k a_k$ in u and v. This can be achieved by one left to right scan of the second pebble. When reaching the end of the string the pebble is simply lifted.

(2) To check that w is syntactically correct, $\mathcal{A}$ further verifies the following.
   - (a) All γ's in u are different: $\mathcal{A}$ places the second pebble on the first γ and scans the other γ's in u with the third pebble. If all are different from the first one, the second pebble is moved to the next γ and the process is repeated.
   - (b) Checking that all γ's in v are different is similar.
   - (c) The first γ in u equals the first γ in w: $\mathcal{A}$ puts the second pebble on the first γ and uses the third pebble to run to the first γ in w.
   - (d) The last γ in u equals the last γ in w: $\mathcal{A}$ moves the second pebble to the end of u. To recognize the last position of u it uses pebble 3 to check whether the symbol to the right of pebble 2 is #. In a similar fashion it moves pebble 3 to the end of w.
   - (e) If γ_1 and γ_2 are successors in u then they also are successors in v: this involves four pebbles (numbered 2 to 5). The second pebble cycles through all γ in u. For each such value d, the automaton proceeds as follows. The third pebble is placed on the γ right after the second pebble. The fourth pebble then cycles through the γ-symbols in v until it finds d. If d is found, the fifth pebble is placed on the γ right after d and consistency can be checked. If this check fails or d is not found in v then the input is rejected. Otherwise the three most recent pebbles are removed.
   - (f) In an analogous way, it can also be verified that the δ's form an index.

(3) To check that w represents a solution of the PCP instance $\mathcal{A}$ proceeds as follows.
   - (a) $\mathcal{A}$ checks that when x_i is picked in u the corresponding choice in v is y_i. Thereto, the second pebble cycles through all γ values of u. $\mathcal{A}$ keeps the corresponding α-value in the finite memory, uses the third pebble to run to the same γ in v and checks whether the β-entry of the latter conforms to the α-entry of the former.

(b) In an analogous way, $\mathcal{A}$ can also check that the a-values at corresponding δ-entry are the same.

This completes the description of the construction of $\mathcal{A}$. It is straightforward to check that $\mathcal{A}$ accepts an input if and only if it represents a solution of the PCP instance. Hence the PCP instance has a solution if and only if $L(\mathcal{A})$ is non-empty. □

6. DISCUSSION

We investigated several models of computations for strings over an infinite alphabet. One main goal was to identify a natural notion of regular language and corresponding automata models. In particular, such a notion should agree in the finite alphabet case with the classical notion of regular language. We considered two plausible automata models: RAs and PAs. Our results tend to favor PAs as the more natural of the two. Indeed, the expressiveness of PAs lies between FO* and MSO*. The inclusion of FO* provides a reasonable expressiveness lower bound, while the MSO* upper bound indicates that the languages defined by PAs remain regular in a natural sense. Moreover, strong PAs are quite robust: all variants without alternation (one or two-way, deterministic or nondeterministic) have the same expressive power.

On the other hand, as one of the referees pointed out, the undecidability results might also be interpreted as an indication that MSO is too powerful in the context of infinite alphabets and that neither of the two models is an appropriate generalization of regular languages to infinite alphabets.

Some of the proofs in this paper bring into play a variety of techniques at the confluence of communication complexity, language theory, and logic. Along the way, we answered several questions on RAs left open by Kaminski and Francez.

Several problems remain open:

(i) can weak 1D-PA or weak 1N-PA be simulated by 2D-RAs or by 2A-RAs?

(ii) are 1D-RA or 1N-RA subsumed by any pebble model? (We know that they can be defined in MSO*. As 1N-RAs are hard for NLOGSPACE they likely cannot be simulated by 2D-PAs.)

(iii) are weak 1N-PAs strictly more powerful than weak 1D-PAs?

(iv) are 2A-PAs strictly more powerful than 2N-PAs?

(v) Are there other, weaker notions of regular languages for infinite alphabets that behave more moderately with respect to the standard decision problems?

ACKNOWLEDGMENTS

We thank the referees for a lot of helpful suggestions.

REFERENCES

Abiteboul, S., Buneman, P., and Suciu, D. 1999. *Data on the Web : From Relations to Semistructured Data and XML.* Morgan Kaufmann.

Abiteboul, S., Herr, L., and Van den Bussche, J. 1999. Temporal connectives versus explicit timestamps to query temporal databases. *Journal of Computer and System Sciences 58,* 1, 54–68.

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases.* Addison-Wesley.

AJTAI, M., FAGIN, R., AND STOCKMEYER, L. J. 2000. The closure of monadic NP. *Journal of Computer and System Sciences 60,* 3, 660–716.

ALON, N., MILO, T., NEVEN, F., SUCIU, D., AND VIANU, V. 2001. XML with data values: Typechecking revisited. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001).* ACM Press, 560–572.

BEX, G. J., MANETH, S., AND NEVEN, F. 2002. A formal model for an expressive fragment of XSLT. *Information Systems 27,* 1, 21–39.

CHANDRA, A. K., KOZEN, D., AND STOCKMEYER, L. 1981. Alternation. *Journal of the ACM 28,* 1, 114–133.

COURCELLE, B. 1990. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B. Elsevier, Chapter 5.

EBBINGHAUS, H.-D. AND FLUM, J. 1999. *Finite Model Theory*, 2 ed. Springer.

GLOBERMAN, N. AND HAREL, D. 1996. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science 169,* 2, 161–184.

GRÄDEL, E. AND GUREVICH, Y. 1998. Metafinite model theory. *Information and Computation 140,* 1, 26–81.

GREENLAW, R., HOOVER, H., AND RUZZO, W. L. 1995. *Limits to Parallel Computation. P-Completeness Theory.* Oxford University Press.

HENNIE, F. C. 1965. One-tape, off-line turing machine computations. *Information and Control 8,* 6, 553–578.

HOPCROFT, J. AND ULLMAN, J. 1979. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

HROMKOVIC, J. 2000. *Communication Complexity and Parallel Computing.* Texts in Theoretical Computer Science - An EATCS Series. Springer-Verlag.

KAMINSKI, M. AND FRANCEZ, N. 1990. Finite-memory automata. In *Proceedings of 31th IEEE Symposium on Foundations of Computer Science (FOCS).* 683–688.

KAMINSKI, M. AND FRANCEZ, N. 1994. Finite-memory automata. *Theoretical Computer Science 134,* 2, 329–363.

LADNER, R. E., LIPTON, R. J., AND STOCKMEYER, L. J. 1984. Alternating pushdown and stack automata. *SIAM J. Comput. 13,* 1, 135–155.

MILO, T., SUCIU, D., AND VIANU, V. 2000. Type checking for XML transformers. In *Proceedings of the Nineteenth ACM Symposium on Principles of Database Systems.* ACM Press, 11–22.

NEVEN, F. 2002a. Automata, logic, and XML. In *CSL*, J. C. Bradfield, Ed. Lecture Notes in Computer Science, vol. 2471. Springer, 2–26.

NEVEN, F. 2002b. On the power of walking for querying tree-structured data. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002).* ACM Press, 77–84.

NEVEN, F. AND SCHWENTICK, T. 2000. Expressive and efficient pattern languages for tree-structured data. In *Proc. 19th Symposium on Principles of Database Systems (PODS 2000).* 145–156.

NEVEN, F. AND SCHWENTICK, T. 2002. Query automata on finite trees. *Theoretical Computer Science 275*, 633–674.

OTTO, F. 1985. Classes of regular and context-free languages over countably infinite alphabets. *Discrete Applied Mathematics 12*, 41–56.

PAPAKONSTANTINOU, Y. AND VIANU, V. 2001. DTD inference for views of XML data. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001).* ACM Press, 35–46.

SAKAMOTO, H. AND IKEDA, D. 2000. Intractability of decision problems for finite-memory automata. *Theoretical Computer Science 231,* 2, 297–308.

SHEPHERDSON, J. C. 1959. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development 3*, 198–200.

SIPSER, M. 1980. Halting space-bounded computations. *Theoretical Computer Science 10*, 335–338.

SUDBOROUGH, I. H. 1975. On tape-bounded complexity classes and multihead finite automata. *Journal of Computer and System Sciences 10,* 1, 62–76.

THOMAS, W. 1997. Languages, automata, and logic. In *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds. Vol. 3. Springer, Chapter 7, 389–456.

VIANU, V. 2001. A web odyssey: From Codd to XML. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*. 1–15.

Received March 2002; revised January 2003; accepted January 2003.