Towards Regular Languages over Infinite Alphabets

Frank Neven[1⋆], Thomas Schwentick[2], and Victor Vianu[3⋆⋆]

[1] Limburgs Universitair Centrum
[2] Friedrich-Schiller-Universität Jena
[3] U.C. San Diego

Abstract. 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, non-deterministic, and alternating control. We investigate the expressiveness and complexity of the automata, their connection to the logics, as well as standard decision problems.

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 [1]. Since XML documents have a 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, automata theory has naturally emerged as a central tool in formal work on XML [13,14,15,16,17,18,19]. 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 [3]). 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. Understanding such automata on strings is also interesting for the case of trees, as most formalisms allow reasoning along paths in the tree. In the case of XML, it would be more

⋆ Post-doctoral researcher of the Fund for Scientific Research, Flanders.
⋆⋆ This author supported in part by the National Science Foundation under grant number IIS-9802288.

J. Sgall, A. Pultr, and P. Kolman (Eds.): MFCS 2001, LNCS 2136, pp. 560–572, 2001.

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; (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. In the second approach, we follow Kaminski and Francez [11,10] and extend finite automata with a finite number of *registers* 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 [6].

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, non-deterministic, and alternating, and PA and RA for pebble and register 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 [11]. 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. 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.

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). 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 some open questions from Kaminsky and Francez We show that universality and containment of 1N-RAs and non-emptiness of 2D-RA are undecidable and that non-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 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.

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.

When the input is restricted to a finite alphabet, PAs recognize the regular languages, even in the presence of alternation [14]. We point out that the pebbling mechanism we employ is based on the one of Milo, Suciu, and Vianu [14] and is more liberal than the one used by Globerman and Harel [7]: 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.

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. In Section 5, we compare the register and pebble models. In Section 6, we discuss decision problems. We conclude with a discussion in Section 7. Due to space limitations proofs are only sketched.

2 Definitions

We consider strings over an infinite alphabet $\mathbf{D}$. Formally, a $\mathbf{D}$-*string* w is a finite sequence $d_1 \cdots d_n \in \mathbf{D}^*$. As we are often dealing with 2-way automata we delimit input strings by two special symbols, $\triangleright, \triangleleft$ for the left and the right end

of the string, neither of which is in $\mathbf{D}$. I.e., automata always work on strings of the form $w = \triangleright v \triangleleft$, where $v \in \mathbf{D}^*$. By $\mathrm{dom}(w)$ we denote the set $\{1, \ldots, |w|\}$ with $|w|$ the length of w. For $i \in \mathrm{dom}(w)$, we also write $\mathrm{val}_w(i)$ for d_i.

2.1 Register Automata

Definition 1 ([11,10]). A *k-register automaton* $\mathcal{B}$ 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; $\tau_0 : \{1, \ldots, k\} \to \mathbf{D} \cup \{\triangleright, \triangleleft\}$ 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 $\tau : \{1, \ldots, k\} \to \mathbf{D} \cup \{\triangleright, \triangleleft\}$. 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 $\gamma = [j, q, \tau]$ 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 $\mathrm{val}_w(j)$. We denote the transitive closure of $\vdash$ by $\vdash^*$. Intuitively, transitions $(i, q) \to (q', d)$ can only be applied when the value of the current position is in register i. Transitions $q \to (q', i, d)$ 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.

We require that the initial register assignment contains the symbols $\triangleright$ and $\triangleleft$, so automata can recognize the boundaries of the input. Furthermore, from a $\triangleright$ only right-transitions and from a $\triangleleft$ only left-transitions are allowed.

As usual, a string w is accepted by $\mathcal{B}$, if $\gamma_0 \vdash^* \gamma$, for some accepting configuration γ. The language $L(\mathcal{B})$ accepted by $\mathcal{B}$, is defined as $\{v \mid \triangleright v \triangleleft \text{ is accepted by } \mathcal{B}\}$.

The automata we defined so far are in general *non-deterministic*. 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 are defined in the usual way. 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.

2.2 Pebble Automata

We borrow some notation from Milo, Suciu, and Vianu [14].

Definition 2. A *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 $i \in \{1, \ldots, k\}$, $s \in \mathbf{D} \cup \{\triangleright, \triangleleft\}$, $P, V \subseteq \{1, \ldots, i-1\}$, and β is of the form (q, d) with $q \in Q$ and $d \in \{$stay, left, right, place-new-pebble, lift-current-pebble$\}$.

Given a string w, a *configuration of* $\mathcal{A}$ *on* w is of the form $\gamma = [i, q, \theta]$ where $i \in \{1, \ldots, k\}$, $q \in Q$ and $\theta : \{1, \ldots, i\} \to \mathrm{dom}(w)$. 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 $\mathrm{depth}(\gamma)$ ($\mathrm{depth}(\theta)$). The *initial* configuration is $\gamma_0 := [1, q_0, \theta_0]$ where $\theta_0(1) = 1$. 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. $P = \{l < i \mid \mathrm{val}_w(\theta(l)) = \mathrm{val}_w(\theta(i))\}$,
3. $V = \{l < i \mid \theta(l) = \theta(i)\}$, and
4. $\mathrm{val}_w(\theta(i)) = s$.

A transition $(i, P, V, q) \to \beta$ *applies* to γ if (1)-(3) hold and no transition $(i', s', P', V', q') \to \beta$ applies to γ. Intuitively, $(i, s, P, V, p) \to \beta$ applies to a configuration, if i is the current number of placed pebbles, p is the current state, P is the set of pebbles that see the same symbol as the top pebble, V is the set of pebbles that are at the same position as the top pebble, and the current symbol seen by the top pebble is s. 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*, *alternating* and *one-way* are analogous to the case of register automata. We refer to these automata as dC-RA 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 $\theta'(i+1) = 1$ (and keeping $\theta'(i) = \theta(i)$) in the *place-new-pebble* case of the definition of the transition relation.

2.3 Logic

We consider first-order and monadic second-order logic over **D**-strings. The representation as well as the logics are special instances of the meta-finite structures and their logics as defined by Grädel and Gurevich [6]. 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 $d \in \mathbf{D}$, 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 **D** is allowed. The logic MSO* is obtained by adding quantification over sets over $\mathrm{dom}(w)$; again, no quantification over **D** is allowed.

2.4 Complexity Classes over Infinite Alphabets

Some of our separating results are relative to complexity-theoretic assumptions. To this end, we assume the straightforward generalization of standard complexity classes (like LOGSPACE, NLOGSPACE, and PTIME) to the case of infinite alphabets that can be defined, e.g., by using multi-tape Turing machines which are able to compare and move symbols between the current head positions. It should be clear that the collapse of two of these generalized classes immediately implies the collapse of the respective finite alphabet classes.

3 Register Automata

We start by investigating RAs. In particular, we compare them with FO* and MSO*. Our main conclusion 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

Theorem 3. *MSO* cannot define all 2D-RA.*

Proof. (sketch) 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.

A 2D-RA can recognize this property by checking that the number of leftmost occurences of symbols is the same in u and v. To this end it moves back and forth between leftmost occurences of symbols in u and v. □

We next show that RAs cannot capture FO*, even with alternation. The proof is based on communication complexity[9].

Theorem 4. *2A-RA cannot express FO*.*

Proof. 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. For clarity, we will often denote i-hypersets with a superscript i, as in $S^{(i)}$. Let us assume that $\mathbf{D}$ contains all natural numbers and let, for $j > 0$, $\mathbf{D}_j$ be $\mathbf{D} - \{1, \ldots, j\}$. Let $j > 0$ be fixed. We inductively define *encodings* of i-hypersets over $\mathbf{D}_j$. A string $w = 1d_1d_2 \cdots d_n1$ over $\mathbf{D}_j$ 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, $iw_1iw_2 \cdots iw_ni$ is an encoding of the i-hyperset $\{H(w_i) \mid i \leq n\}$. Define $L^m_=$ as the language $\{u\#v \mid u$ and v are encodings of m-hypersets over $\mathbf{D}_m - \{\#\}$ and $H(u) = H(v)\}$.

Lemma 5. *For each m, $L^m_=$ is definable in FO*.*

Next, we show that no 2A-RA can recognize $L^m_=$ for $m > 4$. The 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 $H(v)$. Our proof is inspired by a proof of Abiteboul, Herr, and Van den Bussche [2]. To simulate 2A-RA, however, we need a more powerful protocol where the number of messages depends on the number of different data values in u and v. This protocol is defined next. Let $\exp_0(n) := n$ and $\exp_i(n) := 2^{\exp_{i-1}(n)}$, for $i > 0$.

Definition 6. Let P be a binary predicate on i-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 all i-hypersets $X^{(i)}$ and $Y^{(i)}$ over a finite set D there is a finite alphabet Δ of size at most $p(|D|)$ such that $P(X^{(i)}, Y^{(i)})$ can be computed as follows: `I` gets $X^{(i)}$ and `II` gets $Y^{(i)}$; both know D and Δ; they send k-hypersets over Δ back and forth; and, after $\exp_l(p(|D|))$ rounds of message exchanges, both `I` and `II` have enough information to decide whether $P(X^{(i)}, Y^{(i)})$ holds. We refer to strings of the form $u\#v$, where u and v do not contain $\#$, as *split strings*. A communication protocol computes on such strings by giving u to `I` and v to `II`.

Lemma 7. *For $m > 4$, $L^m_=$ cannot be computed by a (2,2)-communication protocol.*

Proof. Suppose there is a protocol computing $L^m_=$. 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 $a_1b_1a_2b_2 \ldots$ a dialogue. Every dialogue has at most $\exp_2(p(d))$ rounds. Hence, there are at most $\exp_2(2d \cdot \exp_2(p(d)))$ different dialogues. However, the number of different m-hypersets over D is $\exp_m(d)$. Hence, for $m > 4$ and D large enough there are m-hypersets $X^{(m)} \neq Y^{(m)}$ such that the protocol gives the same dialogue for $P(X^{(m)}, X^{(m)})$ and $P(Y^{(m)}, Y^{(m)})$, and therefore also on $P(X^{(m)}, Y^{(m)})$ and $P(Y^{(m)}, X^{(m)})$. This leads to the desired contradiction. □

Lemma 8. *On split strings, the language defined by a 2A-RA can be recognized by a (2,2)-communication protocol.*

Proof. (sketch) Let $\mathcal{B}$ be a 2A-RA working on split strings over $\mathbf{D}$. On an input string $w := u\#v$, $[e, q, \tau]$ is a #-configuration when e is the position of # in w. Define $p(n) := |Q|n^k$, where Q and k are the set of states and number of registers of $\mathcal{B}$, respectively. Then the number of #-configurations is $|Q|m^k$ where m is the number of different symbols in w. We assume w.l.o.g. that there are no transitions possible from final configurations. Further, we assume that $\mathcal{B}$ never changes direction or accepts at the symbol #. Hence, on w, when $\mathcal{B}$ leaves u to the right it enters v and vice versa. 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. We omit further details. □

Theorem 4 now follows from Lemmas 5, 7, and 8. □

When restricted to one-way computations, RAs can only express "regular" properties. The next proposition is easily shown using standard techniques.

Proposition 9. *MSO^* can simulate every 1N-RA.*

3.2 Control

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 an a or 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. As deterministic, nondeterministic, and alternating multi-head automata recognize precisely LOGSPACE, NLOGSPACE, and PTIME languages, respectively, membership for 2D-RA, 2N-RA, and 2A-RA is hard for these classes, respectively [20,12]. Furthermore, it is easy to see that the respective membership problems also belong to the infinite alphabet variants of these classes. Thus, we can show the following proposition in which all complexity classes are over infinite alphabets.

Proposition 10. *1. Membership of 2D-RA is complete for* LOGSPACE*;*
2. Membership of 2N-RA is complete for NLOGSPACE*; and*
3. Membership of 2A-RA is complete for PTIME.

The indexing technique above cannot be used for 1N-RAs, because they cannot check whether the odd positions form a unique index. However, we can extend (2) to 1N-RAs using a direct reduction from an NLOGSPACE-complete problem: ordered reachability.

Proposition 11. *Membership of 1N-RA is complete for* NLOGSPACE.

From Theorem 3 and Proposition 9 we get immediately that the class accepted by 1N-RA is different from those accepted by 2D-RA and 2N-RA. It follows from Proposition 11 that all four classes defined by the mentioned automata models are different unless the corresponding complexity classes collapse.

4 Pebble Automata

In this section we show that PAs are better behaved than RAs with regard to the connection to logic. In a sense, PAs are more "regular" than RAs. Indeed, we show that strong 1D-PAs can simlate FO* and that even the most liberal pebble model, 2A-PA, can be defined in MSO*. 1D-PAs are clearly more expressive than FO*; furthermore, we can separate 2A-RAs 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, concatenation and Kleene star, these results suggest that strong PAs define a robust class of languages.

4.1 Expressiveness

Proposition 12. *FO* is strictly included in strong 1D-PA.*

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

Theorem 13. *MSO* can simulate 2A-PA.*

It is open whether the above inclusion is strict. However, we can show the following.

Proposition 14. *For every $i \in \mathbb{N}$, there are MSO* formulas φ_i and ψ_i such that the model checking problem for φ_i and ψ_i is hard for Σ_i^P and Π_i^P, respectively. In contrast, membership for 2A-PAs is PTIME-complete.*

Here, the model checking problem for a logical formula consists in determining, given a string w, whether $w \models \varphi$. Since the first part of the proposition is already known for graphs, it suffices to observe that graphs can readily be encoded as strings. We end this section by considering weak PAs. Recall that this notion only makes a difference for one-way PAs. Unlike their strong counterparts, we show that they cannot simulate FO*, which justifies their name.

Theorem 15. *Weak 1N-PA cannot simulate FO*.*

Proof.(sketch) The proof is similar to the proof of Theorem 4. We show by a communication complexity argument that the FO*-expressible language $L^2_=$ defined in that proof cannot be recognized by a weak 1N-PA. In the current proof, however, we use a different kind of communication protocol which better reflects the behaviour of a weak 1N-PA. We define this protocol next. Again we use strings of the form $u\#v$ where u and v encode 2-hypersets. Let k be fixed and let S_1, S_2 be finite sets. The 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. There is a fixed function $f : \mathbf{D}^* \times \mathbf{D}^k \times S_1 \to S_2$ and the agent can evaluate f on all arguments $(v, \boldsymbol{d}, s)$, where $\boldsymbol{d}$ is a tuple of length k of symbols from u and $s \in S_1$. Based on this information and on u the agent decides whether $u\#v$ is accepted. □

4.2 Control

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

Theorem 16. *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 $\mathcal{B}$ which accepts the same language. Actually, $\mathcal{B}$ will use the same number k of pebbles as $\mathcal{A}$. Very briefly, we show by an induction on i that, for all states q and pebble placements θ, $\mathcal{B}$ can compute all states q' such that $[i, q, \theta] \vdash^*_{>i} [i, q', \theta]$, where the subscript $> i$ indicates that only subcomputations are considered in which, at every step, more than i pebbles are present. □

5 Registers versus Pebbles

The known inclusions between the considered classes are depicted in Figure 1. The pebble and register models are rather incomparable. 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 two-way register model. Some open problems about the relationships between register and pebble automata are given in Section 7.

6 Decision Problems

We briefly discuss the standard decision problems for RAs and PAs. Kaminski and Francez already showed that non-emptiness of 1N-RAs is decidable and that it is decidable whether for a 1N-RA $\mathcal{A}$ and a 1N-RA $\mathcal{B}$ with 2-registers $L(\mathcal{A}) \subseteq L(\mathcal{B})$. We next show that *universality* (does an automaton accept every string) of 1N-RAs is undecidable, which implies that containment of *arbitry* 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, when PAs lift 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.

Theorem 17. *1. It is undecidable whether a 1N-RA is universal.*
2. Containment of 1N-RAs is undecidable.
3. It is undecidable whether a 2D-RA is non-empty.
4. It is undecidable whether a weak 1D-PA is non-empty.

7 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, the extended 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 non-deterministic) have the same expressive power.

Some of the results in the paper are quite intricate. The proofs bring into play a variety of techniques at the confluence of communication complexity, language theory, and logic. Along the way, we answer 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? (*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 2A-PAs.) (*iii*) are weak 1N-PAs strictly more powerful than weak 1D-PAs? (*iv*) are 2A-PAs strictly more powerful than 2N-PAs?

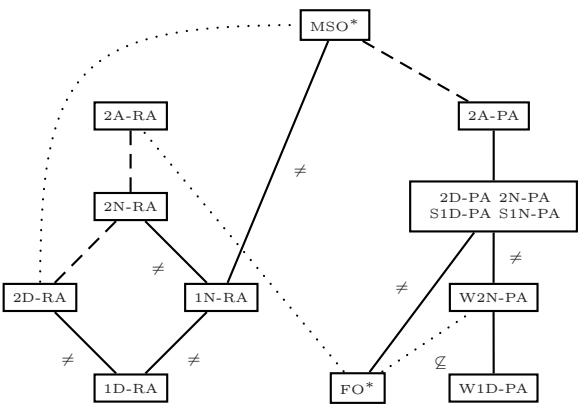

Fig. 1. Inclusions between the classes under consideration. Solid lines indicate inclusion (strictness shown as $\neq$), dotted lines indicate that the classes are incomparable. Dashed lines indicate strict inclusion subject to complexity-theoretic assumptions.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999. 560
2. S. Abiteboul, L. Herr, and J. Van den Bussche. Temporal connectives versus explicit timestamps to query temporal databases. *Journal of Computer and System Sciences*, 58(1):54–68, 1999. 566
3. N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with Data Values: Typechecking Revisited. To be presented at LICS 2001. 560
4. J. Clark XSL Transformations (XSLT) specification, 1999. `http://www.w3.org/TR/WD-xslt.`
5. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 3. Springer, 1997. 572
6. E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 1998. 561, 565
7. N. Globerman and D. Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science*, 169(2):161–184, 1996. 562
8. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
9. J. Hromkovič. *Communication Complexity and Parallel Computing*. Springer-Verlag, 2000. 565
10. M. Kaminski and N. Francez. Finite-memory automata. In *Proceedings of 31th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 683–688, 1990. 561, 563
11. M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. 561, 563
12. A. K. Chandra, L. J. Stockmeyer. Alternation. In *Proceedings of 17th IEEE Symposium on Foundations of Computer Science (FOCS 1976)*, 98–108, 1976. 567
13. S. Maneth and F. Neven. Structured document transformations based on XSL. In R. Connor and A. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming (DBPL'99)*, volume 1949 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2000. 560
14. T. Milo, D. Suciu, and V. Vianu. Type checking for XML transformers. In *Proceedings of the Nineteenth ACM Symposium on Principles of Database Systems*, pages 11–22. ACM Press, 2000. 560, 562, 563
15. F. Neven. Extensions of attribute grammars for structured document queries. In R. Connor and A. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming (DBPL'99)*, volume 1949 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2000. 560
16. F. Neven and T. Schwentick. Query automata. In *Proceedings of the 18th ACM Symposium on Principles of Database Systems (PODS 1999)*, pages 205–214. ACM Press, 1999. 560
17. F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *Proceedings of 19th ACM Symposium on Principles of Database Systems (PODS 2000), Dallas*, pages 145–156, 2000. 560
18. F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. In *Proceedings of 17th ACM Symposium on Principles of Database Systems (PODS 1998)*, pages 11–17. ACM Press, 1998. 560

19. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proceedings of 19th ACM Symposium on Principles of Database Systems (PODS 2000)*, pages 35–46. ACM, Press 2000. 560
20. I. H. Sudborough. On tape-bounded complexity classes and multihead finite automata. *Journal of Computer and System Sciences*, 10(1):62–76, 1975. 567
21. W. Thomas. Languages, automata, and logic. In Rozenberg and Salomaa [5], chapter 7.