Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 159–170
www.stacs-conf.org

WEAK MSO WITH THE UNBOUNDING QUANTIFIER

MIKOŁAJ BOJAŃCZYK

University of Warsaw
E-mail address: bojan@mimuw.edu.pl
URL: www.mimuw.edu.pl/∼bojan

Abstract. A new class of languages of infinite words is introduced, called the *max-regular languages*, extending the class of ω-regular languages. The class has two equivalent descriptions: in terms of automata (a type of deterministic counter automaton), and in terms of logic (weak monadic second-order logic with a bounding quantifier). Effective translations between the logic and automata are given.

1. Introduction

This paper introduces a new class of languages of infinite words, which are called *max-regular languages*, and include all ω-regular languages. Max-regular languages can be described in terms of automata, and also in terms of a logic. A typical language in the class is the property "the distance between consecutive b's is unbounded", i.e. the language

$$L = \{a^{n_1}ba^{n_2}ba^{n_3}\ldots : \forall m\ \exists i\ n_i > m\}\ . \tag{1.1}$$

A practical motivation can be given for considering properties that speak of bounded distance; e.g. a formula of the logic in this paper could specify that a system responds to requests with bounded delay. We will begin, however, with a more fundamental motivation, which is the question: what is a regular language of infinite words?

There is little doubt as to what is a regular language of finite words. For instance, the requirement that the Myhill-Nerode equivalence relation has finitely many equivalence classes uniquely determines which languages of finite words should be regular. Other notions, such as finite semigroups, or monadic-second order logic also point to the same class.

For infinite words, however, there is more doubt. Of course, the class of ω-regular languages has much to justify calling it regular, but some doubts remain as to its uniqueness. Consider, for instance, the language L mentioned above, or the set K of ultimately periodic words, i.e. words of the form wv^ω, say over alphabet a, b. None of these languages are ω-regular. However, under the commonly accepted definition of Myhill-Nerode equivalence for infinite words, given by Arnold in [2], both languages have exactly one equivalence class.

Should these languages be called regular? If yes, what is the appropriate notion of regularity? In this paper we propose a notion of regular languages, which are called *max-regular*

Key words and phrases: automata, monadic second-order logic.
Author supported by Polish government grant no. N206 008 32/0810.

languages, that captures the language L, but not the language K. This new notion has many properties that one would wish from regular languages. The class is (effectively) closed under boolean operations, including negation. There is a finite index Myhill-Nerode relation, and equivalence classes are regular languages of finite words. There is an automaton model, there is a logical description, and translations between the two are effective. Emptiness is decidable. Membership is decidable (although since we deal with infinite words, the membership test is for certain finitely presented inputs, such as ultimately periodic words).

So, what is this new class? One definition is in terms of logic. The max-regular languages are the ones that can be defined by formulas of weak monadic second-order logic extended with the unbounding quantifier. The term "weak" means that only quantification over finite sets is allowed. The unbounding quantifier $UX.\varphi(X)$ was introduced[1] in [3], it says that the size of sets X satisfying $\varphi(X)$ is unbounded, i.e.

$$UX.\varphi(X) = \bigwedge_{n\in\mathbb{N}} \exists X \ \big(\ \varphi(X) \quad \wedge \quad n \leq |X| < \infty \ \big). \tag{1.2}$$

Monadic second-order logic with the unbounding quantifier for infinite trees was studied in [3], where an emptiness procedure was presented for formulas with restricted quantification patterns. This study was continued in [4], where the models where restricted from infinite trees to infinite words, but the quantification patterns considered were more relaxed. However, no decision procedure was given in [4] for full monadic second-order logic with the unbounding quantifier, and the expressive power of the logic seemed to be far too strong for the techniques used (no undecidability results are known, though).

The basic idea in this paper is to restrict the set quantification to finite sets (i.e. weak quantification), while keeping the unbounding quantifier. It turns out that with this restriction, lots of the problems encountered in [4] are avoided, and the resulting class is surprisingly robust. Note that for infinite words and without unbounding quantification, weak monadic second-order logic has the same expressive power as full monadic second-order logic; this is no longer true when the unbounding quantifier is allowed (we prove this using topological techniques).

The main contribution of this paper is Theorem 3.2, which shows that weak monadic second-order logic with the unbounding quantifier has the same expressive power as deterministic max-automata. A max-automaton is a finite automaton equipped with counters, which store natural numbers. The important thing is that the counters are not read during the run (and therefore do not influence the control of the automaton), which avoids the usual undecidability problems of counter machines. The counters are only used in the acceptance condition, which requires some counter values to be bounded, and some to be unbounded.

To the best of the authors knowledge, quantifiers similar to the unbounding quantifier have only been considered in [3, 4]. On the other hand, the idea to use automata with quantitative acceptance conditions, has a long history, going back to weighted automata of Schützenberger [11] (see [7] for a recent paper on weighted automata and related logics).

The max-automata used in this paper are closely related to an automaton model that has been variously called a *distance desert automaton* in [10], a *BS-automaton* in [4], or an *R-automaton* in [1]. One important application, see [10], of these automata is that they can

[1]The quantifier introduced in [3] was actually the negation of U, saying that the size is bounded.

be used to solve the famous star-height problem[2], providing simpler techniques and better complexities than in the famous result of Hashiguchi [8]. (The reduction from the star-height problem is not to emptiness of the automata, but to something called *limitedness*.) Other problems that can be tackled using this type of automata include the star-height of tree languages [5] or the Mostowski index of ω-regular languages [6].

2. The automaton

We begin our presentation with the automaton model.

A *max-automaton* has a finite set of states Q and a finite set of counters Γ. It also has a finite set of transitions. Each transition reads an input letter, changes the state, and does a finite sequence of counter operations. The counter operations are:

$c := c + 1$.	Increment counter c.
$c := 0$.	Reset counter c.
$output(c)$.	Output the value of counter c.
$c := max(c, d)$.	Store in counter c the maximal value of counters c, d.

A max-automaton is run on an infinite word $w \in \Sigma^\omega$. A run is an infinite sequence of transitions, with the usual requirement on consistency with the letters in the input word. Fix a run ρ. With each counter $c \in C$, we associate the sequence counter values $\rho_c \in \mathbb{N}^* \cup \mathbb{N}^\omega$ that have been output by the instruction $output(c)$. These outputs are used by the accepting condition, which is a boolean combination of clauses: "the sequence ρ_c is bounded".

Note that with this acceptance condition, it is only the set of values in ρ_c that matters, and not their order or multiplicity. This is unlike the parity condition (where multiplicity is important), or the S-condition of [4], where the sequence ρ_c is required to tend to infinity.

The toolkit of counter operations could be modified without affecting the expressive power of max-automata. For instance, we could have an operation $c := d$, which is equivalent to $c := 0$ followed by $c := max(c, d)$. On the other hand, the output instruction can be removed (in this case, ρ_c would contain all values of the counter during the run). The output operation can be simulated by the others as follows: for every counter c, we add a new output counter c', which is never incremented. Instead of doing $output(c)$, we do $c' := c$. This way, the counter c' gets only the values that were output on the original counter c.

Theorem 2.1. *Emptiness is decidable for max-automata.*

Proof. The difficulty in the proof is dealing with the max operation.

We will reduce the problem to a result from [4]. A direct and elementary proof can also be given. A *U-automaton* is a max-automaton that does not use the max operation, and where the acceptance condition is a *positive* boolean combination of clauses "counter c is unbounded".

Let $\mathcal{A}$ be a max-automaton that we want to test for emptiness. As is often the case, we will be searching not for an input word accepted by $\mathcal{A}$, but for an accepting run of $\mathcal{A}$ (which is also an infinite word). Fix a single clause in the accepting condition, e.g. "counter c is unbounded". Below, we will show that the set of runs which satisfy this clause can be recognized by a nondeterministic U-automaton. In particular, the set of accepting runs of

[2]This is the question of calculating the least number of nested stars in a regular expression (without negation) that defines a regular language $L \subseteq \Sigma^*$.

$\mathcal{A}$ is a boolean combination of languages accepted by U-automata. The result then follows from [4], where emptiness is shown decidable for boolean combinations of nondeterministic U-automata[3].

Before we define the U-automaton that tests if counter c is unbounded, we introduce some auxiliary definitions. Let c, d be counters of the automaton $\mathcal{A}$. Below we define what it means for a finite sequence of counter operations ρ to transfer c to d, possibly with an increment. (Formally, we are defining two ternary relations: $T(\rho, c, d)$, for transfers, and $TI(\rho, c, d)$, for transfers with an increment.) The idea is that after executing the operations ρ, the value of counter d is at least as big as the value of counter c before executing ρ. The definition of transfers is by induction on the length of ρ:

- Every counter is transferred to itself by the empty sequence of operations, as well as the operations $c := c + 1$ and $output(c)$. Furthermore, $c := c + 1$ also transfers c to itself with an increment.
- The operation $c := 0$ transfers every counter to itself, except c.
- The operation $c = \max(c, d)$ transfers every counter to itself, and also d to c.
- If a sequence of operations ρ_1 transfers c to e, and a sequence of operations ρ_2 transfers e to d, then their concatenation $\rho_1\rho_2$ transfers c to d. If either of the transfers in ρ_1 or ρ_2 does an increment, then so does the transfer in $\rho_1\rho_2$.

Note that the transfer relation is regular in the following sense: for any counters c and d, the set of words ρ that transfer counter c to d is a regular language of finite words, likewise for transfers with an increment.

Let c be a counter. A finite sequence of positions $x_1 < \cdots < x_n$ in a run of $\mathcal{A}$ is called a *c-loop* if for any $i < n$, counter c is transferred to itself with an increment by the subrun between positions x_i to x_{i+1}. For a counter d, a *d-trace* is a sequence of positions $x_1 < \cdots < x_n < y$ such that for some counter c, the positions $x_1 < \cdots < x_n$ are a c-loop, and counter c is transferred to d by the subrun between positions x_n and y.

Equipped with these definitions, we are ready to define a (nondeterministic) U-automaton that tests if counter c is unbounded in an input run. The U-automaton has only one counter, and it accepts if unbounded values are output to this counter. A run of this automaton (which inputs a run of the automaton $\mathcal{A}$) proceeds as follows. It uses nondeterminism to guess a d-trace $x_1 < \cdots < x_n < y$, and it increments its counter at each of the positions x_i. Once it sees position y, it outputs the counter value (which is n), and resets the counter. It then finds another d-trace, and again outputs its length, and so on. It is not difficult to verify the correctness of this construction. ∎

In this paper, we will be mainly interested in deterministic max-automata.

3. The logic

We consider an extension of weak monadic second-order logic, called *weak unbounding logic*. Recall that weak monadic second-order logic is an extension of first-order logic that allows quantification over finite sets (the restriction to finite sets is the reason for the name "weak"). In weak unbounding logic, we further add the *unbounding quantifier* UX, as defined in (1.2).

[3]The result in [4] is for S-automata, which are more powerful than U-automata. It is shown that a boolean combination of S-automata is equivalent to a BS-automaton, which has decidable emptiness.

Example 3.1. Consider the set L from (1.1). This language is not regular, but defined by the following formula of weak unbounding logic:

$$UX\ \forall x \leq y \leq z \quad x, z \in X\ \Rightarrow\ a(y) \wedge y \in X$$

The main result of this paper is that the logic and automata coincide, i.e.

Theorem 3.2. *Weak unbounding logic defines exactly the same languages as deterministic max-automata.*

The more difficult direction in Theorem 3.2 is presented in Section 4. The easier direction, where an automaton is simulated by the logic, can be shown by combining standard techniques with the concepts from the proof of Theorem 2.1. The key idea is that a formula of weak unbounding logic can test if a set of positions $\{x_1 < \cdots < x_n < y\}$ forms a d-trace. It is important that the automata are deterministic, which allows a formula of weak logic to uniquely decode the run that corresponds to the input word.

The formulas that are sufficient to simulate a deterministic max-automaton are of a special type, which gives a normal form for weak unbounding logic:

Proposition 3.3. *Each formula of weak unbounding logic is equivalent to a boolean combination of formulas $UX\varphi(X)$, where $\varphi(X)$ does not use the unbounding quantifier.*

Proof. By translating a formula into an automaton and then back into a formula. ∎

4. Weak bounding logic is captured by deterministic max-automata

We now turn to the more difficult part of Theorem 3.2, namely showing that for every formula of weak unbounding logic there is an equivalent deterministic max-automaton.

The proof is by induction on the size of the formula. To simplify the proof, we use the usual technique of removing first-order quantification, as in [13]. That is, first-order quantification is replaced by three new predicates, all of which can be recognized by the deterministic max-automata: "set X has one element", "set X is included in set Y" and "all elements of set X are before all elements of set Y". Together with weak second-order quantification, these new three predicates can be used to simulate first-order quantification, so the logic is the same. However, since we have removed first-order quantification, in the translation to automata we only have to deal with quantification over finite sets (weak second-order quantification) and the new quantifier.

For purposes of the induction, we generalize the statement to formulas with free variables. What is the word language corresponding to a formula $\varphi(X_1, \ldots, X_n)$? This language contains words annotated with valuations for the free set variables. We use the usual encoding, where the label of a word position $x \in \mathbb{N}$ is extended with a bit vector in $\{0,1\}^n$ that says which of the sets $X_1, \ldots, X_n$ contain position x. More formally, for sets of word positions $X_1, \ldots, X_n \subseteq \mathbb{N}$ and an infinite word $w \in \Sigma^\omega$, we define the word

$$w[X_1, \ldots, X_n]\ \in\ (\Sigma \times \{0,1\}^n)^\omega$$

as follows. On position x, the new word has a tuple $(a, b_1, \ldots, b_n)$, with a the label of the x-th position of the original word w, and the value of bit b_i being 1 if and only if position x belongs to the set X_i, for $i = 1, \ldots, n$. With this notation, we can define the set of words satisfying a formula $\varphi(X_1, \ldots, X_n)$ to be

$$L_\varphi = \{w[X_1, \ldots, X_n] : w, X_1, \ldots, X_n \models \varphi\}\ .$$

Equipped with the above definition, we can use induction to show that the logic is captured by automata, as stated in the proposition below. This result is the main ingredient in the proof of Theorem 3.2.

Proposition 4.1. *For every formula φ of weak unbounding logic, the set L_φ is recognized by a deterministic max-automaton.*

The proof is by induction on the size of the formula φ. The induction base, which corresponds to the predicates "set X has one element", "set X is included in set Y" and "all elements of set X are before all elements of set Y" is easy, since all of these are ω-regular languages, and we have:

Lemma 4.2. *Deterministic max-automata capture all ω-regular languages.*

Proof. By simulating a deterministic automaton with the Muller or parity condition. We add a new counter c_q for each state q of the automaton. Each time state q appears, counter c_q is incremented and output. The counters are never reset. In a run of this automaton, a state appears infinitely often if and only if its counter is unbounded. Therefore, the Muller acceptance condition can be encoded in the unbounding condition of a max-automaton. ∎

The induction step for boolean operations—including negation—is no more difficult, since the automata are deterministic and the accepting condition is closed under boolean operations. We are left with weak second-order quantification and the unbounding quantifier. We first deal with weak quantification, in Section 4.1, while the unbounded quantifier is treated in Section 4.2.

4.1. Weak existential quantification

This section is devoted to showing:

Proposition 4.3. *Languages recognized by deterministic max-automata are closed under weak quantification. In other words, if L is a language over $\Sigma \times \{0,1\}$ recognized by a deterministic max-automaton, then there is a deterministic max-automaton recognizing*

$$\{w \in \Sigma^\omega : w[X] \in L \text{ for some finite set } X\} .$$

A convenient way to prove this result would be to use nondeterministic automata. Unfortunately, as we will later show, adding nondeterminism to max-automata gives power beyond that of weak unbounding logic, so we cannot use this strategy. We will have to do the existential quantification directly in the deterministic automata.

The proof technique is actually very generic. It would work for any model of deterministic automata that all ω-regular languages and satisfies some relaxed assumptions, mainly that the acceptance condition is prefix-independent.

Fix a deterministic max-automaton $\mathcal{A}$ that recognizes L, with state space Q.

A *partial run* in an infinite word w is a run that begins in any position of the word (not necessarily the first position) and in any state (not necessarily the initial one). In other words, this is a word in $\bot^*\delta^\omega \cup \bot^\omega$, where δ is the set of transitions of $\mathcal{A}$, that is consistent with the word w on those positions where it is defined (i.e. where it is not $\bot$). Since the automaton is deterministic, a partial run is uniquely specified by giving the first configuration where it is defined, this is called the *seed configuration*. (There is also the undefined partial run $\bot^\omega$, which has no seed configuration.) Here, a configuration is a pair (q, x), where q is a state and x is a word position. Note that we do not include the counter

values in the seed configuration, since the acceptance condition is not sensitive to finite perturbations.

We say that two partial runs *converge* if they agree from some position on. Equivalently, they converge if they share some configuration, or both are undefined. We say a set of partial runs *spans* a word w if every partial run over w converges with some run from the set. Usually, we will be interested in finite sets of spanning runs.

Lemma 4.4. *For every word w, there is a set of at most $|Q|$ spanning runs.*

Proof. We begin with some arbitrary configuration, and take the partial run ρ_1 that begins in that configuration. If $\{\rho_1\}$ is spanning, then we are done. Otherwise, we take some partial run ρ_2 that does not converge with ρ_1, and see if the set $\{\rho_1, \rho_2\}$ is spanning. If it is not, we add a third partial run ρ_3, and so on. This process terminates after at most Q steps, because if two partial runs do not converge, then they must use different states on each position where they are both defined. So $|Q|$ partial runs that do not converge will use up all the states. ∎

To prove Proposition 4.3, we use a result stronger than Lemma 4.4. We will show that not only the spanning set of runs exists, but it can also be computed by a (deterministic, letter-to-letter) transducer. By *transducer* we mean a finite deterministic automaton where each transition is equipped with an output letter, from an output alphabet Γ. Therefore, the transducer defines a function $f : \Sigma^\omega \to \Gamma^\omega$. The transducer does not have any accepting conditions (using bounds or even parity or Muller), it just scans the word and produces its output. It is easy to see that deterministic max-automata are closed under preimages of transducers, as shown in the following lemma.

Lemma 4.5. *If f is a transducer and $\mathcal{A}$ is a deterministic max-automaton, then there is a deterministic max-automaton recognizing the set of words w such that $f(w)$ is accepted by $\mathcal{A}$.*

We now describe how the spanning partial runs will be encoded in the output of the transducer. When speaking of spanning partial runs, we mean spanning partial runs of the automaton $\mathcal{A}$ in Proposition 4.3. A single partial run can be encoded as an infinite word over the alphabet $Q \times \{0, 1\}$. The idea is that $\{0, 1\}$ is used as a marker, with 0 meaning "ignore the prefix until this position", and 1 meaning "do not ignore". Formally, an infinite word

$$(q_1, a_1)(q_2, a_2), \ldots \quad \in \quad (Q \times \{0, 1\})^\omega$$

is interpreted as the partial run which on position i has $\perp$ if $a_j = 0$ for some $j \geq i$, otherwise it has q_i. Note that if the word above has infinitely many positions j with $a_j = 0$, then the partial run is nowhere defined, i.e. it is $\perp^\infty$. If we want to encode n partial runs, we use n parallel word sequences, encoded as a single sequence over the product alphabet

$$(Q \times \{0, 1\})^n \ .$$

With the encoding of spanning runs defined, we are now ready to present the stronger version of Lemma 4.4.

Lemma 4.6. *Let $n = |Q|$. There is a transducer*

$$f : \quad \Sigma^\omega \quad \to \quad ((Q \times \{0, 1\})^n)^\omega$$

such that for any word w, the output $f(w)$ encodes n spanning partial runs.

Proof. The idea is to implement the proof of Lemma 4.4 in a transducer. The states of the transducer will be permutations of the state space, i.e. tuples from Q^n where each state appears exactly once. The initial state is any arbitrarily chosen permutation. When reading an input letter a in state $\pi = (q_1, \ldots, q_n)$, the transducer does the following operations. First, it transforms each state in π according to the letter a, giving a tuple $x = (q_1 a, \ldots, q_n a)$. This tuple is not necessarily a permutation, i.e. there are may be some coordinates $i \in \{1, \ldots, n\}$ such that the state $q_i a$ appears already in $\{q_1 a, \ldots, q_{i-1} a\}$. Let $I = \{i_1, \ldots, i_k\}$ be these coordinates, and let $\{p_1, \ldots, p_m\}$ be the states that do not appear in the new tuple x. These two sets have the same size, i.e. $k = m$. We can now correct x to be a permutation σ, by replacing its coordinate i_1 with the state p_1, the coordinate i_2 with state p_2, and so on. Note that on a the coordinates from I, the new permutation σ has a value unrelated to the one from π (i.e. σ begins a new run), while on coordinates from outside I, the new permutation σ simply continues the runs from π. This is signified in the output of the transducer, which is decorates each coordinate i of the permutation σ with a bit, which is 0 when $i \in I$ and 1 otherwise. ∎

We are now ready to prove Proposition 4.3. By properties of spanning sets of runs, a word $w \in \Sigma^\omega$ belongs to the language of the proposition if and only if there is some $i = 1, \ldots, n$ such that the following two properties hold:

(A) The i-th run encoded by $f(w)$ is defined (i.e. the encoding does not contain infinitely many cancelling 0s) and satisfies the accepting condition in the automaton $\mathcal{A}$.
(B) There is some finite set $X \subseteq \mathbb{N}$ such that the run of $\mathcal{A}$ over $w[X]$ converges with the i-th run encoded by $f(w)$.

Since deterministic max-automata are closed under union, it suffices to show that for each fixed i, both properties (A) and (B) are recognized by deterministic max-automata. For property (A), we use Lemma 4.5 on preimages. Property (B), on the other hand, is an ω-regular property, which can be recognized by a deterministic max-automaton thanks to Lemma 4.2.

4.2. Unbounding quantification

We now turn to the more difficult part of Proposition 4.1, namely that deterministic max-automata are closed under unbounding quantification.

Proposition 4.7. *Languages recognized by deterministic max-automata are closed under unbounding quantification. In other words, if L is a language over $\Sigma \times \{0,1\}$ recognized by a deterministic max-automaton, then so is*

$$UL = \{w \in \Sigma^\omega : w[X] \in L \text{ for arbitrarily large finite sets } X\} .$$

Fix a deterministic max-automaton $\mathcal{A}$ recognizing the language L in the proposition. Given a finite prefix $w \in \Sigma^*$ and a state q of $\mathcal{A}$, let $max(q, w)$ be the maximal size of a set X of positions in w such that the automaton $\mathcal{A}$ reaches state q after reading $w[X]$. We claim that the sets $max(q, w)$ can be computed in the counters of a deterministic max-automaton (not surprisingly, using the max operation).

Lemma 4.8. *There is a deterministic max-automaton with counters $\{c_q\}_{q \in Q}$ such that the value of c_q after reading a prefix $a_1 \cdots a_n$ of the input is exactly $max(q, a_1 \cdots a_n)$.*

We will use the values from the above lemma to capture the unbounding quantifier. However, some more effort is needed: it is not the case that an input word $w = a_1a_2\cdots$ belongs to UL if and only if the values $max(q, a_1\cdots a_n)$ are unbounded. In general, only the left to right implication holds. The right to left implication may fail since a value $max(q, a_1\cdots a_n)$ is relevant only if the run of $\mathcal{A}$ over w that begins in configuration (q, n) can be extended to an accepting one over the rest of the word. The correct characterization is given below:

Lemma 4.9. *A word $a_1a_2\cdots \in \Sigma^\omega$ belongs to UL if and only if for some state q, the following values are unbounded*

$\{max(q, a_1\cdots a_n) : a_{n+1}a_{n+2}\cdots[\emptyset] \in (\Sigma \times \{0,1\})^\omega$ *is accepted by $\mathcal{A}$ when starting in $q\}$*

As suggested by the above lemma, to recognize the language UL it would be convenient to have an extension of max-automata, where the automaton would have the ability to output $max(q, a_1\cdots a_n)$ only in case a certain property was satisfied by the suffix $a_{n+1}a_{n+2}\cdots$. Below, we introduce such an extension of max-automata, which we call a guarded max-automaton. We then show that this extension can be simulated by a standard max-automaton, thus completing the proof of Proposition 4.7.

An *guarded max-automaton* is like a max-automaton, except that it is also allowed to use the following counter operation:

$if\ L\ then\ output(c)$. Output the value of counter c, but only if the suffix of the input beginning at the next position belongs to $L \subseteq \Sigma^\omega$.

In the above operation, the language L—called the *guard* of the transition—must be a language recognized by a max-automaton (without guards, although allowing guards would give the same result). This new operation is all we need to recognize the language UL:

Lemma 4.10. *If a language L is recognized by a deterministic max-automaton, then UL is recognized by a deterministic guarded max-automaton.*

We will show that guarded outputs are redundant, and can be simulated by non-guarded outputs. This completes the proof Proposition 4.7. The difficulty in the proof below is that we are dealing with deterministic automata, while a guard looks to the future.

Proposition 4.11. *For every deterministic guarded max-automaton there is an equivalent deterministic max-automaton.*

Proof. Let $\mathcal{A}$ be a deterministic guarded max-automaton. To simplify notation, we assume that only one guarded operation,

$$o \quad = \quad if\ L\ then\ output(c)\ ,$$

is used. The general case is done the same way. Let $\mathcal{B}$ be a deterministic max-automaton recognizing the guard language L.

In the construction, we will use a concept of *thread*. A thread consists of a state of the automaton $\mathcal{B}$, as well as a number, which corresponds to the value of counter c output by the guarded operation o. Note that a thread does not contain information about values of the counters of automaton $\mathcal{B}$. The idea is that threads will be alive for only finitely many steps, so the counters of $\mathcal{B}$ are not relevant. We will denote threads by τ. If $a \in \Sigma$ is an input letter, then we write τa for the thread obtained from τ by updating the state according to a (and leaving the number unchanged).

The (non-guarded) max-automaton $\mathcal{C}$ that simulates $\mathcal{A}$ works as follows. At each point, the simulating automaton contains a finite set $\{\tau_1, \ldots, \tau_i\}$ of *active* threads. There will be at most one thread per state of $\mathcal{B}$, so the set of threads can be stored using finitely many counters and the finite memory of the automaton. This set of active threads is initially empty. Whenever $\mathcal{A}$ does the guarded output operation o, a new active thread is created, with the initial state of $\mathcal{B}$, and the number set to the value of counter c. Furthermore, after reading an input letter $a \in \Sigma$, the set of active threads is updated to $\{\tau_1 a, \ldots, \tau_i a\}$. If two active threads have the same state, then they are merged, and only the greater number is kept (using the max operation).

Similarly to the proof of Proposition 4.3, the automaton $\mathcal{C}$ will also read the output of a transducer f that computes spanning partial runs of the automaton $\mathcal{B}$ used for the guards. Recall that the transducer f outputs n spanning partial runs of the automaton $\mathcal{B}$, where n is the number of states in $\mathcal{B}$.

The automaton $\mathcal{C}$ accepts a word w if and only if there is some $i = 1, \ldots, n$ such that:

(A) The i-th run encoded by $f(w)$ is defined (i.e. the encoding does not contain infinitely many cancelling 0s) and satisfies the accepting condition in the automaton $\mathcal{B}$.
(B) For every m, some thread storing a number greater than m converges with i-th run encoded by $f(w)$.

Since deterministic max-automata are closed under finite union, we only need to show the construction for some fixed i. As in the previous section, property (A) is recognized by a deterministic max-automaton. For property (B), it suffices to output the number stored in a thread τ whenever its state is the same as in ρ_i. The automaton then accepts if the numbers thus produced are unbounded.

■

5. Problems with nondeterminism

In this section we show that nondeterministic max-automata are more expressive than deterministic ones.

Theorem 5.1. *Nondeterministic max-automata recognize strictly more languages than deterministic ones.*

Contrast this result with the situation for Muller or parity automata, which are equally expressive in the deterministic and nondeterministic variants. Since full monadic second-order can capture nondeterministic automata by existentially quantifying over infinite sets, the above theorem immediately implies:

Corollary 5.2. *Full monadic second-order logic with the unbounding quantifier is stronger than weak monadic second-order with the unbounding quantifier.*

The separating language in Theorem 5.1 is

$$L = \{a^{n_1} b a^{n_2} b a^{n_3} b \ldots : \text{ some number appears infinitely often in } n_1, n_2, \ldots\} . \tag{5.1}$$

This language is captured by a nondeterministic max-automaton. The automaton uses nondeterminism to output a subsequence of $n_1, n_2, \ldots$ and accepts if this subsequence is bounded. Clearly, if it is bounded, then it contains an infinite constant subsequence.

It remains to show that the language L cannot be recognized by a deterministic max-automaton. For this, we will use topological complexity. In Lemmas 5.3 and 5.4, we

will show that every language recognized by a deterministic max-automaton is a boolean combination of sets on level Σ_2 in the Borel hierarchy, while the language L is not.

Below we briefly describe the Borel hierarchy, a way of measuring the complexity of a subset of a topological space. The topology that we use on words is that of the Cantor space, as described below. A set of infinite words (over a given alphabet Σ) is called *open* if it is a union

$$\bigcup_{i \in I} w_i \Sigma^\omega \qquad w_i \in \Sigma^* \ ,$$

with the index set I being possibly infinite. In other words, membership of a word w in an open set is assured already by a finite prefix of w. For the Borel hierarchy, as far as max-automata are concerned, we will only be interested in the first two levels $\Sigma_1, \Pi_1, \Sigma_2, \Pi_2$. The open subsets are called Σ_1, the complements of these (the closed subsets) are called Π_1. Countable intersections of open subsets are called Π_2, the complements of these (countable unions of closed subsets) are called Σ_2.

Lemma 5.3. *Any language accepted by a deterministic max-automaton is a boolean combination of Σ_2 sets.*

Proof. Fix a max-automaton $\mathcal{A}$, and a counter c of this automaton. We will examine the topological complexity of the set of runs of this automaton (here, a run is an infinite sequence of transitions). For any fixed n, the following set of runs is clearly open:

A value of at least n is output at least once on counter c.

In particular, its complement

All values of counter c are at most n.

is a closed set of runs. By taking a countable union of the above over $n \in \mathbb{N}$, we deduce that the property

The values of counter c are bounded.

is a Σ_2 property. In particular, the set of accepting runs of any max-automaton is a boolean combination of Σ_2 sets. Since the automata are deterministic, the function that maps an input word to its run is continuous, i.e. preimages of open sets are also open. Since preimages of continuous functions preserve the levels of the hierarchy, we conclude that any language accepted by a deterministic max-automaton is a boolean combination of Σ_2 sets. ∎

Lemma 5.4. *The language L is not a boolean combination of Σ_2 sets.*

Proof. Consider the mapping from $\mathbb{N}^*$ to $\{a, b\}^*\omega$ defined by

$$n_1, n_2, \ldots\ldots \qquad \mapsto \qquad a^{n_1} b a^{n_2} b a^{n_3} b \ldots$$

This is a continuous mapping. The language L is the image, under this mapping, of the set X of sequences in $\mathbb{N}^\omega$ that have a bounded subsequence. The set X is known not to be a boolean combination of Σ_2 sets, see Excercise 23.2 in [9]. ∎

6. Conclusion

This paper is intended as a proof of concept. The concept is that ω-regular languages can be extended in various ways, while still preserving good closure properties and decidability. The class presented in this paper, max-regular languages, is closed under boolean operations, inverse morphisms, and quotients. It is not closed under morphic images (which corresponds to nondeterminism on the automaton side).

Some questions on max-automata are left unresolved. Is the max operation necessary in the automaton? In our construction, we use the max twice: when defining the values $max(q, a_1 \cdots a_n)$, and in Proposition 4.11. While in the first case, the max operation can be avoided by a subtle use of factorization forests [12], it is not clear how to show Proposition 4.11 without using the max operation. Another question is the exact complexity of emptiness. It would be nice to get matching upper and lower bounds, even more so if the lower bound would use acceptance conditions in DNF.

There are several other possibilities of future work. One is to investigate weak bounding logic for infinite trees (note that we will not capture all regular languages of infinite trees in this case). Another possibility would be to investigate full monadic-second order logic, or possibly other quantifiers that can be added to weak monadic second-order logics. The techniques used in this paper are fairly generic, so it seems plausible that such quantifiers can be found.

References

[1] P. A. Abdulla, P. Krcál, and W. Yi. R-automata. In *CONCUR*, pages 67–81, 2008.
[2] A. Arnold. A syntactic congruence for rational omega-language. *Theor. Comput. Sci.*, 39:333–335, 1985.
[3] M. Bojańczyk. A bounding quantifier. In *Computer Science Logic*, volume 3210 of *Lecture Notes in Computer Science*, pages 41–55, 2004.
[4] M. Bojańczyk and T. Colcombet. Omega-regular expressions with bounds. In *Logic in Computer Science*, pages 285–296, 2006.
[5] T. Colcombet and C. Löding. The nesting-depth of disjunctive mu-calculus for tree languages and the limitedness problem. In *Computer Science Logic*, volume 5213 of *Lecture Notes in Computer Science*, 2008.
[6] T. Colcombet and C. Löding. The non-deterministic mostowski hierarchy and distance-parity automata. In *International Colloquium on Automata, Languages and Programming*, volume 5126 of *Lecture Notes in Computer Science*, pages 398–409, 2008.
[7] M. Droste and P. Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.
[8] K. Hashiguchi. Algorithms for determining relative star height and star height. *Inf. Comput.*, 78(2):124–169, 1988.
[9] A. S. Kechris. *Classical Descriptive Set Theory*, volume 156 of *Graduate Texts in Mathematics*. Springer, 1995.
[10] D. Kirsten. Distance desert automata and the star height problem. *Theoretical Informatics and Applications*, 39(3):455–511, 2005.
[11] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
[12] I. Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72:65–94, 1990.
[13] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.