LOGIC AS REGULAR BEHAVIOUR

KAMAL LODAYA

1. Expressions to logic

Fix a finite set A as an *alphabet.* Its elements are called *letters.* A finite sequence of letters $w : \{1, \dots, n\} \to A$, such as $aabab$, is called a *word* over the alphabet. A set of words is called a *language.*

The set of all words over A is written A^*. The empty word is written ε. In general, given a language L, its iteration L^* is the language formed by concatenating words from L to form another word. For instance, given the language $\{aa, ab, b\}$, also written $aa \cup ab \cup b$, the word $aabab$ is in $\{aa, ab, b\}^*$, but the word ba is not. The null word ε is always in any L^* (by taking words from L zero times).

This notation, with some ad hoc extensions, can be used to describe languages. We will not look into the details of the notation: these are called *regular expressions*, and are defined in any book on automata theory. We assume that the reader is familiar with such material. With this background, the aim of this article is to examine some subclasses of regular languages which can be characterized using logic as well as algebra. Proofs are barely sketched. The interested reader should try to follow up the material and construct more detailed proofs.

1.1. Expressions.

The *starfree expressions* are those where the iteration L^* is not allowed. The *dot depth* of a starfree expression is the maximum number of nested alternations of the boolean operations ($e_1 \cup e_2$ and $\overline{e}$) and concatenations ($e_1 e_2 \dots e_n$). Since complement is allowed, infinite languages are also describable. Thus A^* can be described as $\overline{\emptyset}$, a starfree expression of dot depth 0. The dot depth 1 expression $\overline{A^* bb A^*}$ describes the language $NoTwoBs$ of words which do *not* have consecutive occurrences of the letter b.

Exercise 1. *Give a starfree expression for the language* $Cycle = (ab)^*$.

Exercise 2. *Verify that the language* $(ab + ba)^*$ *can be described by the dot depth* 2 *starfree expression* $\overline{(\overline{A^* a}\ b(ab)^*\ \overline{a A^*}) \cup (\overline{A^* b}\ (ab)^* a\ \overline{b A^*})}$.

I was strongly encouraged to write this article by Priti Shankar, when she was working on the book referred to in the bibliography. It has taken me a long time to put it together. Pascal Weil taught me a lot of this material. I have received comments on it from graduate students over several years: Anantha Padmanabha, Simoni Shah, Srikanth Srinivasan and Vaishnavi Sundararajan. Many thanks to Gnanaraj Thomas for his patience as editor of this volume, as I kept adding one picture after another. I also thank him and his colleagues for hosting the conference on *Automata, Graphs and Logic* at Madras Christian College in August 2015, where I gave a brief talk on some of the material.

The language $(aa)^*$ is not starfree. How do we prove this?

Among the starfree expressions Marcel-Paul Schützenberger in 1976 restricted concatenation of languages $L_1L_2 \dots L_n$ to make *unambiguous expressions*, by requiring that any word belonging to such a product has a unique factorization $w = w_1w_2 \dots w_n$, with each w_i in L_i. Pascal Tesson and Denis Thérien called B^* an expression with *unambiguous dot depth* 0 over subalphabet B. Suppose e_1, e_2 are expressions over subalphabets A_1, A_2 respectively, whose difference is a single letter $a \in A_2 \setminus A_1$, and they have unambiguous dot depth i. Then e_1ae_2 and e_2ae_1 are expressions with unambiguous dot depth $i + 1$ over alphabet A_2. An *unambiguous expression* is a disjoint union of expressions with some unambiguous dot depth. For example, $A^*b(c^*d(a + c + d)^*)$ has unambiguous dot depth 2 over $A = \{a, b, c, d\}$. Its words can be unambiguously described as "after the last b there are only c's until the next d". We abbreviate this language as $AfterLastB$.

The languages $NoTwoBs$ and $Cycle$ are starfree, but cannot be defined by unambiguous expressions. How do we prove this?

1.2. **Logic.** A more systematic notation that programmers and computer scientists have come to use is logical formulas. Here is a sentence in first-order logic describing exactly the words in the language $NoTwoBs = \overline{A^*bbA^*}$:

$$\forall x \forall y (y = x{+}1 \wedge b(x) \supset \neg b(y))$$

Formally, we are working in a structure $\{1, \dots, n\}$ of *positions* in the word, with pointers indicating the positions of variables (indicated below by underlines), binary predicates for the linear order and the successor ($x < y$ and $y = x{+}1$ can be thought of as binary predicates $less(x, y)$ and $successor(x, y)$), and unary predicate symbols $\{a(.) \mid a \in A\}$. At each position, the unary predicate corresponding to the letter at that position holds, and no other. Pointer functions like $s = [x \mapsto 5, y \mapsto 6]$ below are called "assignments" and written $w, s \models \alpha$ in logic textbooks.

$$\begin{array}{l} a\ b\ a\ \underline{b}\ \underline{a}\ b \models y = x{+}1 \wedge b(x) \supset \neg b(y) \\ a\ \underline{b}\ a\ \underline{b}\ a\ b \not\models x < y \wedge b(x) \supset \neg b(y) \\ a\ b\ a\ a\ \underline{b}\ \underline{b} \not\models y = x{+}1 \wedge b(x) \supset \neg b(y) \end{array}$$

More precisely, we put the pointers into the words, expanding the alphabet to $A \times \wp(Var_1)$, where the variables Var_1 are those which appear in the first-order formula, constrained to occur exactly once in the word model.

$$\begin{pmatrix} a \\ \emptyset \end{pmatrix} \begin{pmatrix} b \\ \emptyset \end{pmatrix} \begin{pmatrix} a \\ \emptyset \end{pmatrix} \begin{pmatrix} b \\ \{x\} \end{pmatrix} \begin{pmatrix} a \\ \{y\} \end{pmatrix} \begin{pmatrix} b \\ \emptyset \end{pmatrix} \models y = x{+}1 \wedge b(x) \supset \neg b(y)$$

Position i in the word having the letter $(a, \{x, y\})$ means that in addition to its having the letter a, the variables x and y are assigned the position i. Thus a word becomes a model for a sentence (a formula with no free variables). A language, a set of words, becomes a set of *finite models* for the formula.

Here is a sentence, repeatedly reusing two variables, for $AfterLastB = A^*bc^*d(a+c+d)^*$. Since it is long, it is presented in two steps:

$$\begin{array}{lcl} AfterLastB & = & \exists x(b(x) \wedge \forall y(x < y \supset \neg b(y)) \wedge \exists y(x < y \wedge d(y) \wedge ToLastB(y)))), \\ ToLastB(y) & = & \forall x(x < y \wedge \neg c(x) \supset \exists y(x \leq y \wedge b(y))) \end{array}$$

Exercise 3. *Write a sentence for the language $Cycle = (ab)^*$. Use few variables.*

Robert McNaughton and Seymour Papert showed in their 1971 book that what we have seen from the examples generalizes all the way to starfree expressions.

Theorem 4 (McNaughton and Papert)**.** *The starfree languages are $FO[<]$-definable.*

Proof. By induction on the starfree expressions we construct a first-order logic sentence whose finite word models define the same language.

The expression $\emptyset$ maps to $false$, the expression a to the sentence $(min = max) \wedge a(min)$. The positions min and max mark the beginning and end of a word, it is easy to translate them into first-order logic using a single quantifier. Boolean operations on the expressions translate into the same operations on logical formulas. We are left with concatenation e_1e_2: for this we use the sentence $\exists x(e_1^{[min,x]} \wedge e_2^{[y\,=\,x+1,max]})$, where the successor function can also be defined in first-order logic, and the FO sentences for the expressions e_1, e_2 obtained from the induction hypothesis are *relativized* to intervals. Relativization can be defined by induction on the logical formula: $a(x)^{[i,j]}$ is $i \leq x \leq j \supset a(x)$, for boolean combinations we push the relativization down onto the subformulas. The main case is the quantifier, for which we have $(\exists x\phi(x))^{[i,j]} = \exists x(i \leq x \leq j \wedge (\phi(x)^{[i,j]}))$. □

Many first-order definable languages are regular. We can also explore subclasses by restricting the numerical predicates used, the number of variables in a formula, and so on. It is also common to describe programming behaviours using formulae of *temporal* logic, but that is something we will not venture into in this article.

Thérien and Thomas Wilke obtained a stronger result for unambiguous languages. The doctoral thesis of Simoni Shah studies this connection in detail.

Theorem 5. *Unambiguous languages are $FO^2[<]$-definable, that is, their words are models of sentences of first-order logic using two variables.*

Proof. Inductively one can maintain for expressions of unambiguous dot depth i a sentence with two variables x, y for the left and right ends of the word. In the base case they are the definitions of min and max. Tesson and Thérien illustrate the induction step for e_1be_2, where b is not in the alphabet of e_2, for the language $AfterLastB$:

$$\begin{array}{lcl} AfterLastB & = & \exists x(b(x) \wedge (\forall y > x : \neg b(y)) \wedge e_2(y) \\ e_2(y) & = & \exists x > y(d(y) \wedge \forall y < x(\neg c(y) \supset \exists x \geq y : b(y)) \end{array}$$

The language e_2 from y is constrained to be $A^*bc^*dA^*$ and the full language is constrained to be $A^*b(c^*dA^* \cap (a+c+d)^*)$. In this case e_1 is A^*, but using the variable x marking its right end we could have inductively constructed an expression for the prefix. □

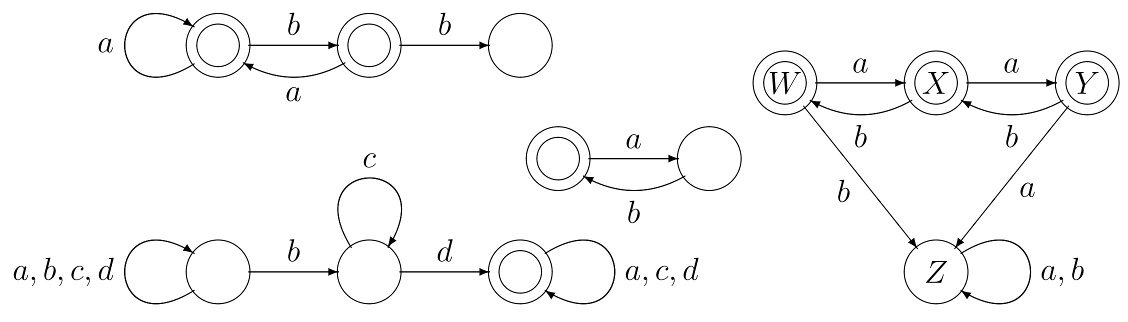

FIGURE 1. Automata for the languages *NoTwoBs*, *AfterLastB*, *Cycle* and *Bicycle*

2. Logic as automata behaviour

Cognitive scientists, like WARREN MCCULLOCH AND WALTER PITTS in the 1940s, invented a graphical notation (called *finite automata*) to represent regular languages.

Formally, a *transition system* is a directed graph (Q, δ), with the edges (called *transitions*) labelled by letters of the alphabet ($\delta \subseteq Q \times A \times Q$) which can be alternately seen as the function $\delta : A \to \wp(Q \times Q)$. The vertices are called *states*. A finite automaton is a finite transition system with a set of *initial states* and a set of *final states* $I, F \subseteq Q$. Figure 1 shows you automata for the languages we have been talking about. The leftmost state of each automaton is its only initial state, and final states are marked by double circles.

Here is how an automaton operates. It begins in an initial state. On each letter of the word, it takes the corresponding transition from the current state into a (possibly) new state. At the end of the word, if the automaton is in a final state, it *accepts* the word. The *language accepted* by the automaton is all words for which there is a sequence of moves from an initial state to a final state.

Exercise 6. *Design an automaton accepting the language Even of all even-length words.*

Does looking at languages using automata help in our attempt to describe them in logic? Yes, here is a sentence for the transition system of the automaton for *Bicycle*:

$$\begin{array}{l}
\exists W \exists X \exists Y \exists Z \\
(\ \forall w \forall x (W(w) \wedge z = w + 1 \supset ((a(w) \supset X(z)) \wedge (b(w) \supset Z(z)))) \\
\wedge \forall x \forall z (X(x) \wedge z = x + 1 \supset ((a(x) \supset Y(z)) \wedge (b(x) \supset W(z)))) \\
\wedge \forall y \forall z (Y(y) \wedge z = x + 1 \supset ((a(y) \supset Z(z)) \wedge (b(y) \supset X(z)))) \\
\wedge \forall z \forall x (Z(z) \wedge x = z + 1 \supset Z(x)) \\
)
\end{array}$$

This is a sentence of *monadic second-order logic*, with the variables $X, Y, Z \in Var_2$ ranging over sets of positions. Each set variable stands for a state of the transition system, and is interpreted as being true for the positions of the word at which the automaton, operating on the word, is in that state. A little more is required to fully describe an automaton —for instance, the initial and final states must be used.

Exercise 7. *Given an arbitrary finite automaton, make up a sentence describing the language accepted by it.*

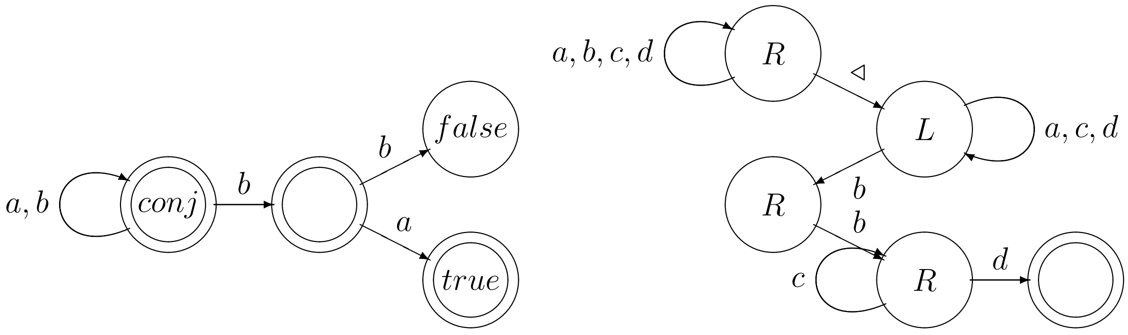

FIGURE 2. Alternating automaton for $NoTwoBs$, two-way automaton for $AfterLastB$

Formally, our word models are now also equipped with an interpretation for the second-order variables. Think of the automaton alphabet as being expanded with a set of first- and second-order variables, specifying at each position the variables which are true there.

Exercise 8. *Construct automata on the alphabet $A \times \wp(\{x, y\}) \times \wp(\{X\})$ which check if $a(x)$, $y = x + 1$ and $X(y)$ are true.*

Using these techniques, independently RICHARD BÜCHI in 1960, CALVIN ELGOT in 1960, and BORIS TRAKHTENBROT in 1962, were able to prove a fundamental result:

Theorem 9. *The languages accepted by finite automata can be defined in the monadic second-order theory of the successor relation (MSO$[+1]$) over words.*

2.1. **More automata.** The notation of MCCULLOCH AND PITTS was richer, and was formalized by MCNAUGHTON AND PAPERT as "nerve nets" with "excitatory" and "inhibitory" effects. In modern terms, as defined later by ASHOK CHANDRA, DEXTER KOZEN AND LARRY STOCKMEYER, these are *alternating* automata, which are formalized as having the transition function $\delta : A \to (Q \to \mathcal{B}^+(Q))$, where the range of a state is a positive boolean combination of states. (Positive boolean formulae include *true* and $false$.) Assume this is written in disjunctive normal form as sets (disjuncts) of conjunctions (sets) of states $\wp(\wp(Q))$.

Here is how an alternating automaton operates. Assume it begins in a single initial state. Inductively the automaton has multiple copies, say one in every state in set $X \subseteq Q$. On letter a of the word, from every one of the states $q \in X$, the automaton picks one disjunctive conjunction, say D_q, from $\delta(a)(q)$ and, for every state r in D_q, forks a copy in state r. Several copies from different states in D may arrive at r, they all join to form a single copy. Hence the automaton now has copies in every state of the set $Y = \{r \mid r \in D_q \in \delta(a)(q), q \in X\}$. At the end of the word, it is *accepted* if every copy is in a final state.

For a graphical notation, some states are marked as *conjunctive*, see Figure 2. Outgoing edges with the same letter are treated as forking copies, otherwise they are *disjunctive*, and given a letter, only one of the edges with that letter is chosen in a run. Every time the automaton in the figure sees a letter b, from its conjunctive state it forks a copy to make sure that if another b is seen immediately after, it will fail.

Exercise 10. *Given an alternating automaton, make up a sentence describing the language accepted by it.*

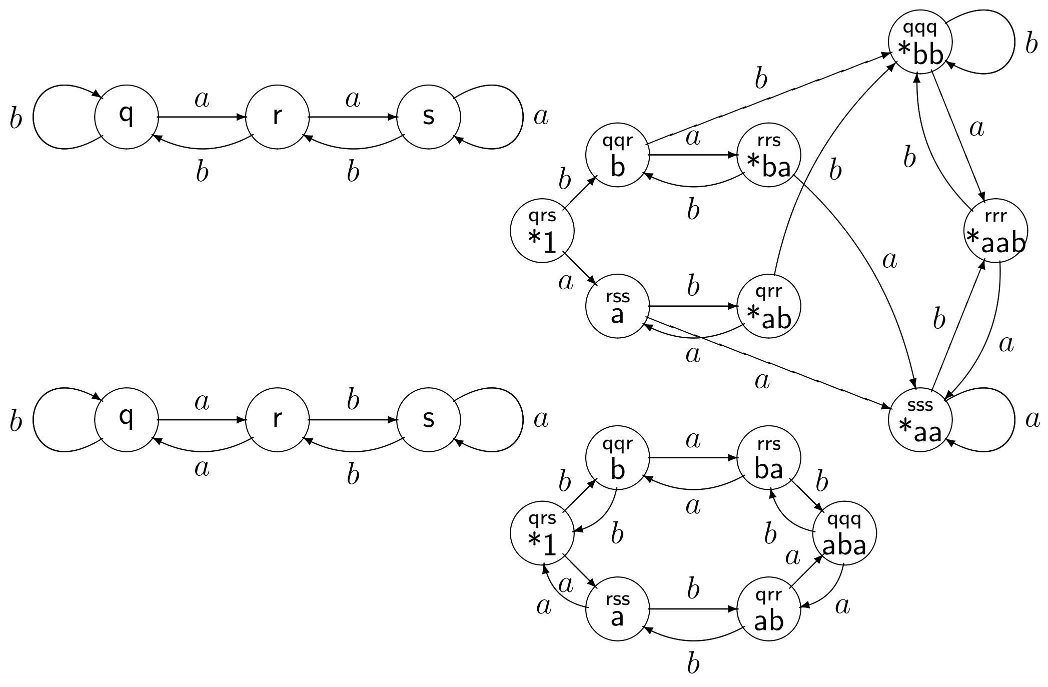

Figure 3. Two transition systems and their transition functions as monoids

Allowing many passes over the input leads to the *two-way* automata of John Shepherdson and Howard Sturgis from 1963. One definition is to divide the states into *left-moving* and *right-moving* ones, marked as L and R in Figure 2. The letter $\triangleleft$ denotes that the two-way automaton reads a right endmarker for the word (after which it has to turn left), similarly $\triangleright$ is the left endmarker.

Exercise 11. *Given a two-way automaton, make up a sentence describing the language accepted by it. Also try your hand at a sentence for a* two-way alternating *automaton.*

3. Congruences and algebra

Michael Rabin and Dana Scott in a 1965 paper showed, using a powerset construction which they derived from John Myhill, that for finite automata, it is sufficient to consider *deterministic* transition systems where there is a unique a-labelled edge leaving a state q, for every a and q.

First observe that the set of relations $\wp(Q \times Q)$ over the finite set Q is the *transition monoid* of the system, with relation composition being the monoid operation and the identity relation as the unit 1 of the monoid. Recall that a *monoid* is a set with a binary associative operation $\circ$ which has a unit, that is, $m \circ 1 = m = 1 \circ m$.

Since the set of words A^* is the free monoid generated by A, the function δ can be freely extended to a monoid homomorphism $h : A^* \to \wp(Q \times Q)$ using $h(xy) = h(x) \circ h(y)$ and $h(\varepsilon) = 1$. (As is common, we also use δ as a name for the extended morphism h.) As our description of their run shows, this extends to alternating automata, where the homomorphisms are of the form $h : A^* \to (\mathcal{B}^+(Q) \to \mathcal{B}^+(Q))$. In fact we can have a Rabin-Scott construction from an alternating automaton to a powerset nondeterministic automaton (and so to a power-powerset deterministic automaton) accepting the same language.

In this sense finite monoids can be derived from finite transition systems. Figure 3 (which appeared in McNaughton and Papert's book) shows, for a deterministic transition system with states $Q = \{\mathsf{q},\mathsf{r},\mathsf{s}\}$, its set of transition functions $Q \to Q$ (the triple is the image of $\mathsf{q},\mathsf{r},\mathsf{s}$ respectively) and how they can be seen as the multiplication table of the elements of its transition monoid. *Idempotent* elements (those e such that $e = ee$) are starred.

Exercise 12. *Construct the minimal dfa and transition monoid of the automaton for Cycle. It has eight elements, five of which are idempotent.*

3.1. **Congruences of finite index.** Two words x and y are equivalent if they define the same relation $h(x) = h(y)$ on Q. This is a finite-index congruence (called the *kernel* of h) studied by Myhill in the 1950s. Anil Nerode studied the more compact "right" congruences. The relations which map initial states I to final states F in an automaton are a subset of its transition monoid, and the language accepted is $h^{-1}(I \times F)$. The result is:

Theorem 13 (Myhill and Nerode). *The regular languages are exactly those which are inverse images of morphisms into (designated subsets of) finite monoids.*

Proof. The explanation above gives the argument in the forward direction. For the converse, the monoid itself can be used as the set of states and each multiplication $ma = m'$, $a \in A$, gives a deterministic a-labelled transition. 1 is the unique initial state and the designated subset gives the final states. □

The finite monoid is said to *recognize* the corresponding regular language. Myhill and Nerode showed that the congruences corresponding to finite automata form a lattice. Therefore, given a regular language, there is a maximal saturating finite-index congruence for it, and hence a minimal deterministic finite automaton accepting it. Its transition monoid is called the *syntactic* monoid of the language.

3.2. **Two-way automata.** Shepherdson and Sturgis showed that the Myhill-Nerode result can be used to go from a two-way automaton, making several passes over the input, to a one-way automaton for the same language. Their "crossing sequence" argument defines tables for every word:

$$T : A^* \to ((Q \cup \{1\}) \to (Q \cup \{\perp\})).$$

When the automaton crosses into the prefix x from the right (or for the first time) in left-moving state q, it crosses back out of x (or never does) from the left in right-moving state $T(x)(q)$. It is sufficient to consider finitely many prefixes x which correspond to different right congruence classes. Richard Ladner, Richard Lipton and Stockmeyer showed that *two-way alternating* automata can also be converted to one-way alternating automata. The tables now record for each word functions of the form $\mathcal{B}^+(Q) \to \mathcal{B}^+(Q)$.

Recently Christian Dax and Felix Klaedtke moved these functions into the alphabet to combine the Shepherdson-Sturgis and Rabin-Scott constructions, going from a two-way alternating automaton with states Q accepting language L over alphabet A to a two-way nondeterministic automaton with the same states Q accepting K over the expanded alphabet $A \times ((\mathcal{B}^+(Q) \to \mathcal{B}^+(Q))$ such that the projection of K to A is $\overline{L}$. Its transitions evaluate the given function on the given state:

$\delta'(q,(a,T)) = T(q)$ if $T(q) \models \delta(q,a)$ (positive boolean formula in the alternating automaton).

3.3. **Inside regular languages.** In this article we will consider a couple of subclasses of regular languages, also of automata and of monoids.

Given a finite automaton, the nonempty word $w \in A^+$ is a *counter* if the transition function $\delta(w)$ induces a nontrivial permutation on the states Q. In Figure 3, the word a is a counter in the bottom automaton on the states X, Y, Z, while the word b is a counter on X, Y. An automaton without any counter is *counter-free*, as studied by McNaughton and Papert. For example, the top automaton in Figure 3 is counter-free. It has cycles, but the cycles are not formed from counters. Recently Fabrice Chevalier, Volker Diekert, Deepak D'Souza, Paul Gastin, Raj Mohan and Pavithra Prabhakar observed that counter-freeness is preserved across automata.

Theorem 14. *Given a counter-free alternating (nondeterministic) automaton, there is a counter-free nondeterministic (deterministic, resp.) automaton accepting the same language.*

Proof. Use the Rabin-Scott construction from a counter-free automaton M to a powerset automaton N accepting the same language. Suppose N has a counter $X \overset{w^n}{\Longrightarrow} X$ for some state X, some $w \in A^+$, $n \geq 1$. From the subset construction, for each $q \in X$ from M, we find $p \overset{w^n}{\longrightarrow} q$ in the automaton M for some $p \in X$. Iterating backward and using the pigeonhole principle on the finite automaton M, we can find some $p \in X$ and positive j, k such that $p \overset{w^{jn}}{\longrightarrow} p \overset{w^{kn}}{\longrightarrow} q$ in M. Since M was counter-free, w could not be a nontrivial counter, so $p \overset{w}{\to} p$. Hence p is in $\delta(X, w^{1+kn})$, which is the same as $\delta(X,w)$. That is, $X \subseteq \delta(X,w)$. By induction $\delta(X,w) \subseteq \delta(X,w^n) = X$. So $X \overset{w}{\Longrightarrow} X$ in N and N is counter-free. $\square$

The subclass of monoids required to represent counter-free automata is easy to define. A *group* is a monoid in which every element has an inverse. The bottom automaton in Figure 3 shows that (minimal) transition systems which are groups have a very symmetric structure.

A monoid which is not a group can have submonoids which turn out to be groups. For example, any idempotent e defines a trivial subgroup $\{e\}$. This gives us the right definition.

Theorem 15 (McNaughton and Papert). *The transition monoid of a reduced counter-free automaton does not contain any nontrivial subgroup.*

Proof. Let G be a subgroup in the syntactic monoid of the language accepted by the automaton, containing powers of an element $g \in G$. One of those powers must be an idempotent $g^n = (g^n)^2 = g^{2n}$ for $n \geq 1$. Towards a contradiction assume nontriviality of the group, so $g^n \neq g^{n+1}$. By minimality of the automaton there is a word w mapping to g. So for some state q, we find states $\delta(w^n)(q) = r = \delta(w^{2n})(q) = \delta(w^n)(r)$ and by supposition, $r \neq \delta(w)(r)$. Hence w is a counter which permutes the states $\{r, \delta(w)(r), \ldots, \delta(w^{n-1})(r)\}$. $\square$

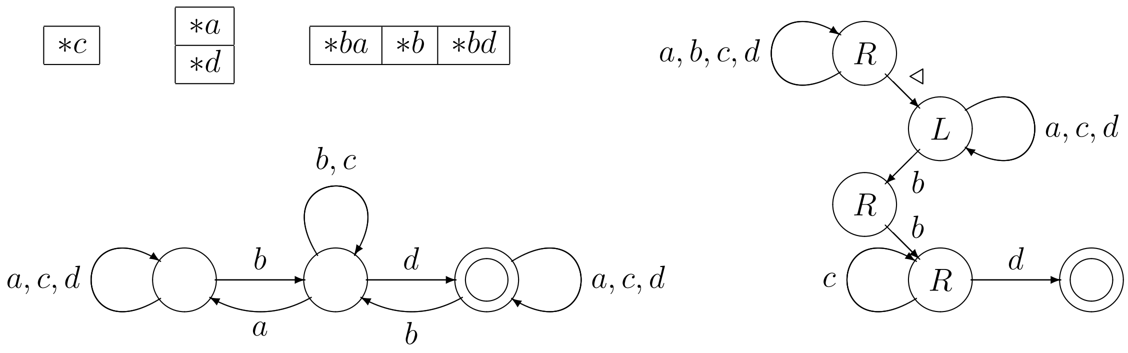

FIGURE 4. Monoid structure, minimal and po-2way-dfa for the language $AfterLastB$

4. ORDER INSIDE AUTOMATA

In a *partially ordered* transition system, defined by ALBERT MEYER AND CAROL THOMPSON in 1969, the states $(Q, \geq)$ are partially ordered, and a transition from state q can only go to states r where $q \geq r$. Hence the only cycles allowed are self-loops where $q = r$, a run taking a *falling* transition where $q > r$ can never climb back to q. Partially ordered *two-way deterministic* automata can accept more languages than the conventional ones which do one pass from left to right over a word. They are also closed under boolean operations. At the right of Figure 2 we gave a two-way dfa for $AfterLastB = A^*bc^*d(a + c + d)^*$, we observe that it is partially ordered.

The *alternating* automaton we gave in Figure 2 for the language $NoTwoBs = \overline{A^*bbA^*}$ is partially ordered. The history of these automata goes back to the *loop-free* nerve nets that MCNAUGHTON AND PAPERT used as an intermediate mechanism for counter-free automata.

It is easy to see that partially ordered automata, one-way or two-way, nondeterministic or alternating, are counter-free. The transitions divide into self-loops and falling transitions which are not part of any cycle. To use this information we have to look inside monoids.

4.1. Cycle structure.

The representation of cycles in a monoid M is through *monoid ideals.* (If you know the ring ideals of EDUARD KUMMER in number theory, their definition is used here in a generalized setting.) The *two-sided ideal* for an element $x \in M$ is $MxM = \{mxn \mid m, n \in M\}$; the *right ideal* is $xM = \{xn \mid n \in M\}$, symmetrically the *left ideal* is $Mx = \{mx \mid m \in M\}$.

JAMES GREEN in a 1951 paper defined the equivalence relations $x\mathcal{J}y$ iff $MxM = MyM$, $x\mathcal{R}y$ iff $xM = yM$, and $x\mathcal{L}y$ iff $Mx = My$. In a finite monoid, the equivalence classes are called *D-classes, $\mathcal{R}$-classes and $\mathcal{L}$-classes.* Two transitions on a cycle in an automaton will give rise to the same two-sided ideal, the words reached will be in the same D-class.

Figure 4 above has three D-classes. GREEN's results showed that each D-class can be pictured as a matrix with $\mathcal{R}$-classes as the rows and $\mathcal{L}$-classes the columns. Each entry of the matrix is called an $\mathcal{H}$-class, where $x\mathcal{H}y$ iff $x\mathcal{R}y$ and $x\mathcal{L}y$ is yet another equivalence relation. If you think about ideals, you will realize that $\mathcal{H}$-classes correspond to groups. In the figure, and in any monoid which does not have nontrivial groups, they are singletons.

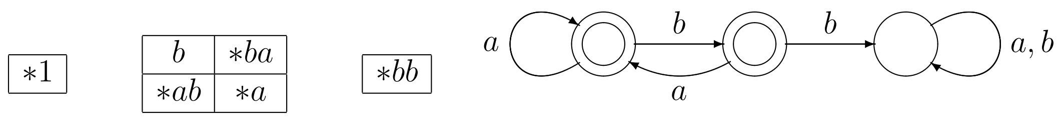

FIGURE 5. The syntactic monoid structure of $NoTwoBs$

Exercise 16. *Verify the syntactic monoid structure of $AfterLastB$ in Figure 4, and that all elements are idempotents.*

The next theorem shows that the language $AfterLastB$ cannot be accepted by a partially ordered one-way dfa. The authors partially credit IMRE SIMON's doctoral thesis.

Theorem 17 (JANUSZ BRZOZOWSKI AND FAITH ELLEN FICH)**.** *The transition monoid of a minimal partially ordered one-way deterministic automaton has trivial $\mathcal{R}$-classes. (That is, every D-class is an $\mathcal{L}$-class.)*

Proof. Let $q = \delta(w) \geq r = \delta(a)(q)$ for $a \in A$ and $r \geq q = \delta(x)(r)$ for some $x \in A^+$, so that w and wa belong to the same $\mathcal{R}$-class. In a partially ordered deterministic one-way automaton, it follows that $q = r$. Recalling that all cycles in a partially ordered automaton are self-loops, w and wa map to the same element. If w is an idempotent, so is wa. ☐

4.2. **Cycle structure of two-way automata.** We can now define the subclass DA of monoids introduced by SCHÜTZENBERGER in 1976: they satisfy the property that if there is an idempotent element in a D-class, then the entire D-class consists of idempotents. Figure 4 verifies that the syntactic monoid of the language $AfterLastB = A^*bc^*d(a+c+d)^*$ is in DA. This can also be inferred from its partially ordered two-way dfa and the next theorem.

Theorem 18 (THOMAS SCHWENTICK, DENIS THÉRIEN AND HERIBERT VOLLMER)**.** *The transition monoid of a minimal partially ordered two-way deterministic automaton is in DA.*

Proof. From the proof of the previous theorem, let $q = \delta(w) \geq r = \delta(a)(q)$ for $a \in A$ and $r \geq q = \delta(x)(r)$ for some $x \in A^+$. We get that the words w and wa or aw, depending on whether the state being considered was right- or left-moving, in the same $\mathcal{R}$-class or $\mathcal{L}$-class, map to the same element. It follows that if w maps to an idempotent, so must wa or aw. By repeating the argument, we get that all elements in the D-class must be idempotents. ☐

If you construct the syntactic monoids of $Cycle = (ab)^*$ or $NoTwoBs = \overline{A^*bbA^*}$, then you can see that they do not contain nontrivial groups, but they are not in DA. Figure 5 gives the syntactic monoid of $NoTwoBs$. The element b maps to a non-idempotent element in a D-class with idempotent elements. Hence the languages $Cycle$ and $NoTwoBs$ cannot be accepted by partially ordered two-way deterministic finite automata. They can be accepted using partially ordered alternating finite automata.

5. LOGIC TO AUTOMATA

Our starting notion of behaviour was properties expressed as sentences of first-order logic. It turned out that to describe the behaviour of an automaton we used monadic second-order logic. But one can ask, and McNaughton did ask in 1960, what can be described using first-order logic. The book of McNaughton and Papert has comprehensive answers. The connection to alternating automata was studied in the doctoral thesis of Gareth Rohde, and also in the work of Christof Löding, Kai Salomaa, Wolfgang Thomas and Sheng Yu. Towards understanding this we begin with a small subclass of first-order logic.

Theorem 19. *A language that is defined with a sentence of $FO^2[<]$ over words, with just two variables, unary predicates for the alphabet and a binary predicate for the linear order, is accepted by a partially ordered two-way deterministic automaton.*

Proof. First let us consider the quantifier-free case and then we will lift the result by induction. It is an exercise to construct such automata for the unary alphabetic predicates. We can assume that the automata start at any desired position on the word. The positive boolean operations are done sequentially. Depending on whether the automaton for the first operand accepts or rejects, the automaton for the second operand is invoked or deemed unnecessary for the result. Negation is handled by complementing the final states.

The key property of first-order logic over a linear order is that when a quantifier is introduced over a formula $\alpha(x, y)$ with at most two variables free, there are three possibilities: either we have $\exists y(y > x \wedge a(y) \wedge \alpha(x, y))$, or we have $\exists y(y < x \wedge a(y) \wedge \alpha(x, y))$, for different choices of letters a in the alphabet, or we have $\exists y(y = x \wedge \alpha(x, y))$. The last case is equivalent to the substituted formula $\alpha'(x) = \alpha(x, y)[x/y]$, which is covered by the induction hypothesis. So we have to handle one of the first two cases, the other follows by symmetry.

Schützenberger and later authors proved that the position y is unambiguous, that is, the formula $\alpha(x, y)$ and the direction from position x uniquely identify y. For example, the position of the letter d which follows the last b on the world in the language $AfterLastB$, illustrates this. For a detailed study we refer again to Shah's thesis. Hence we have a *deterministic* two-way automaton which finds the position y, marked by an unambiguously determined position of letter a, then moves down the partial order and verify $\alpha(x, y)$. $\square$

It follows from Theorem 18 that the syntactic monoid of a language defined by an $FO^2[<]$ sentence is in DA. By Theorem 5, so are the syntactic monoids of unambiguous languages, a theorem first proved by Schützenberger using a direct proof. In Section 7 we give Thérien and Wilke's proof of the converse, characterizing them logically. Also algorithmically, because the property characterizing DA can be checked on a finite monoid.

Now we move on to full first-order logic.

Theorem 20. *A language that is defined with a first-order sentence over words, with unary predicates for the alphabet and a binary predicate for the linear order, is accepted by a partially ordered two-way alternating automaton.*

Proof. First we disallow reusing of variables. Since we have no upper bound on the number of variables, this can always be done by renaming. Again let us consider the quantifier-free case and then we will lift the result by induction.

We can assume that the automata start at any desired position on the word. The atomic cases are easily done. The positive boolean operations go into the transition function of an alternating automaton. Negation is handled by taking the dual automaton, where disjunctions and conjunctions in the transition function are exchanged, and the final states complemented. During all these operations we maintain a partial order among the states.

Similar to the previous theorem, when a quantifier $\exists y$ is introduced over a formula $\alpha(y)$, but now possibly with more free variables than y, because of the linear order we need to consider two possibilities (the third being handled by the induction hypothesis as before): either y is to the left of the current position or to the right. A nondeterministic automaton can guess the position y, move down the partial order and verify $\alpha(y)$ using the induction hypothesis. An alternating two-way automaton can conjunctively check out both the directions. Hence we have a partially ordered alternating two-way automaton for $\exists y \alpha(y)$. □

From a partially ordered two-way alternating automaton accepting language L, Dax and Klaedtke construct a two-way nondeterministic automaton accepting a language K over a much larger alphabet, such that projecting K to A gives $\overline{L}$. They preserve the partial order since their construction does not change the states. The projection and complement gives a two-way nondeterministic automaton for L. A partially ordered automaton is counter-free, applying Theorem 14 so is the automaton for the projection, as also one-way nondeterministic and deterministic counter-free automata for the same language. From Theorem 15 it follows that the syntactic monoid of a language defined by a first-order sentence does not have nontrivial groups. From Theorem 4, neither do starfree languages.

We did not have to do this long proof to show all these results, but in the remaining Sections of this article we will complete the circle by proving Schützenberger's theorem that languages whose syntactic monoid do not have nontrivial groups are starfree, thereby proving the equivalence of all these different definitions. There is an algorithm checking the monoid condition and hence the questions posed in Section 1 are solved.

Finally we tackle monadic second-order logic.

Theorem 21. *A language defined by a sentence of monadic second-order logic is regular.*

Proof. Again we use induction on the structure of formulas. Finite automata can be constructed for the atomic formulas. From automata theory, we know that regular languages are closed under the boolean operations as well as under projection, which we use for the cases dealing with both the first- and second-order quantifiers. □

A little more is true. Theorem 9 gave the converse of the above theorem. Hence checking satisfiability or validity of monadic second-order logic is reduced to checking the automaton constructed above for nonemptiness or universality.

Corollary 22 (Büchi, Elgot and Trakhtenbrot)**.** *The monadic second-order theory of successor $MSO[+1]$ over finite words is decidable.*

6. Pumping lemmas

Fix a finite monoid M. Green's equivalence relations which we saw earlier are formed from pre-orders, for example, $x \geq_{\mathcal{R}} y$ iff $xM \supseteq yM$. This happens when $y \in xM$, that is, x can be extended to y. In more detail, for some z, $y = xz$. Other notions are easily defined:

$*1$

$*ab$	a
b	$*ba$

$*aabb$	aab	aa
abb	$*abba$	baa
bb	bba	$*bbaa$

$*aaa$

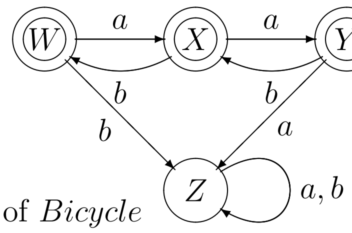

FIGURE 6. The transition monoid structure of *Bicycle*

$x \geq_{\mathcal{L}} y$ iff $Mx \supseteq My$ iff $x \in My$ iff for some z, $x = zy$ iff $Mx \supseteq My$.

$x \geq_{\mathcal{J}} y$ iff $MxM \supseteq MyM$ iff x is a factor of y iff for some z, z', $y = zxz'$ iff $MxM \supseteq MyM$.

It helps to do a quick check on a monoid. The monoid identity 1 will be a maximal element under the $\geq_{\mathcal{R}}$ order since $M = 1M$ is maximal under inclusion. If the monoid has a zero element 0 (for every m, $m0 = 0m = 0$), it will be minimal and an $\mathcal{R}$-class $\{0\}$ by itself.

Backtracking to the deterministic transition system of which M is a transition monoid, the intuition is that the $\mathcal{R}$-classes are the strongly connected components, with transitions "falling downwards" in the $\geq_{\mathcal{R}}$-order. When an automaton begins in an initial state, the prefix of the word seen is ε and the corresponding monoid element is 1. As the automaton processes an input word, it moves "downwards" in the monoid. As long as it cycles through a state, the prefix seen is mapped into the same $\mathcal{R}$-class. If the prefix seen so far is mapped by the morphism h into 0, the image of the input word is 0.

Exercise 23. *Check that the transition monoid for Bicycle in Figure 6 verifies this intuition.*

6.1. **Factorizing runs into cycles.** Given a homomorphism $h : A^* \to M$ into a finite monoid, a word w can be factorized into $\mathcal{R}$-classes

$$w = (u_0)(a_1u_1)\dots(a_tu_t),$$

where the $u_i, 0 \leq i \leq t$, are words and the $a_i \in A, 1 \leq i \leq t$, are *progress letters* from an $\mathcal{R}$-class to the next, that is, $h(u_0) = 1$, it is not the case that $h(u_0 \dots u_s)\mathcal{R}h(u_0 \dots u_s a_{s+1})$, for $s < t$, and $h(u_0 \dots u_{s-1}a_s)\mathcal{R}h(u_0 \dots u_{s-1}a_s u_s)$, for $s \leq t$. If the *content* (subalphabet) of w has ℓ letters, we call $t\ell$ the *weight* of this factorization.

Symmetrically, any word can be similarly $\mathcal{L}$-factorized from right to left.

We can also do a simultaneous right and left $\mathcal{J}$-factorization. Namely, any word w can be $\mathcal{R}$-factorized and $\mathcal{L}$-factorized:

$$w = (u_0)(a_1u_1)\dots(a_tu_t) \text{ and } w = (v_0b_1)\dots(v_{t-1}b_t)(v_t),$$

so that $h(u_0) = h(v_t) = 1$, neither $h(u_0 \dots u_s)\mathcal{J}h(u_0 \dots u_s a_{s+1})$, nor $h(v_s \dots v_t)\mathcal{J}h(b_s v_s \dots v_t)$, for $s < t$, and $h(u_0 \dots u_{s-1}a_s)\mathcal{R}h(u_0 \dots u_{s-1}a_s u_s)$, and $h(b_s v_s \dots v_t)\mathcal{L}h(v_{s-1}b_s v_s \dots v_t)$. Again we call the a_i and b_i in A, $1 \leq i \leq t$, *progress letters*. The progress of the D-classes can be depicted as follows:

$$w = \begin{pmatrix} u_0 \\ v_0b_1 \end{pmatrix} \overset{\xrightarrow{a_1}}{\underset{b_1}{\longleftarrow}} \begin{pmatrix} a_1u_1 \\ v_1b_2 \end{pmatrix} \overset{\xrightarrow{a_2}}{\underset{b_2}{\longleftarrow}} \dots \quad \dots \overset{\xrightarrow{a_{t-1}}}{\underset{b_{t-1}}{\longleftarrow}} \begin{pmatrix} a_{t-1}u_{t-1} \\ v_{t-1}b_t \end{pmatrix} \overset{\xrightarrow{a_t}}{\underset{b_t}{\longleftarrow}} \begin{pmatrix} a_tu_t \\ v_t \end{pmatrix}$$

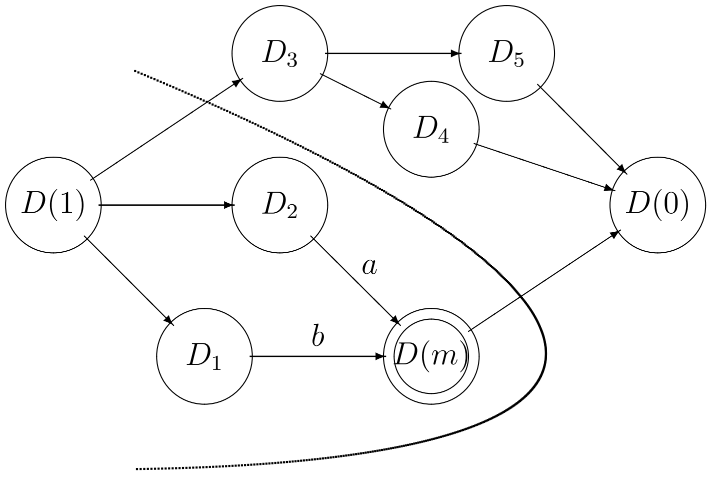

FIGURE 7. D-classes, progress letters and forbidden factors in a monoid with zero

Thus $u_0 \dots u_{t-1}$ ($v_1 \dots v_t$, respectively) is the longest prefix (suffix, resp.) of w in a different D-class from w, and the factorization proceeds recursively.

6.2. **Forbidden factors.** For SCHÜTZENBERGER's proof of 1965, we have to look at the *absence* of factorizations, which gets represented in a monoid as two-sided ideals. The ideal $F(m) = \{y \mid \neg\exists p, q : pyq = m\} = \{y \mid \neg(y \geq_{\mathcal{J}} m)\}$ consists of the *forbidden factors* of m. The exercise below shows that from a starfree expression for $h^{-1}(F(m))$, one can obtain a starfree expression for $h^{-1}(\mathcal{H}(m))$, and in a monoid without nontrivial groups, $h^{-1}(m)$.

In Figure 7, D-classes are shown as circles. The $>_{\mathcal{J}}$-maximal one is $D(1)$. $F(m)$ is the union of D-classes $D_3, D_4, D_5, D(0)$ above the curved line which encloses D-classes containing elements $\geq_{\mathcal{J}} m$. The D-classes $D(m), D_4, D_5$ just $>_{\mathcal{J}} 0$ are said to be *0-minimal.*

Exercise 24. *Work out the following.*

(1) *$F(m)$ is an ideal, a union of D-classes: if $x \in F(m)$ and x is a factor of y, then $y \in F(m)$. (Once you fall into an ideal you cannot climb out again.)*
(2) *m is not in $F(m)$. (It is forbidden.)*
(3) *If M has a zero 0, then $F(0)$ is empty. Every other $F(m)$ will contain 0.*
(4) *Using Proposition 25 below, show that $\mathcal{R}(m) = mM \setminus F(m)$; $\mathcal{L}(m) = Mm \setminus F(m)$; so $\mathcal{H}(m) = (mM \cap Mm) \setminus F(m)$.*

6.3. **Pumping cycles.** The matrix structure depicted for the D-classes comes about because GREEN showed that different $\mathcal{R}$-classes in a D-class represent different cycles, they do not interact. Figure 8 below abstracts the monoid structure of *Bicycle* from Figure 6 into $\mathcal{R}$-classes, D-classes are represented vertically. This is made precise next, using a little algebra to find and "pump" idempotents.

Proposition 25. *If x, y are in the same D-class and $x \geq_{\mathcal{R}} y$ ($x \geq_{\mathcal{L}} y$), then they are in the same $\mathcal{R}$-class ($\mathcal{L}$-class, respectively).*

Proof. Let $x \geq_{\mathcal{R}} y$, that is, $y = xz$ for some z. To show $y = xz \geq_{\mathcal{R}} x$, we should extend xz, that is, find a cycle beginning with z, which gets us back to x. The elements x, y are in the

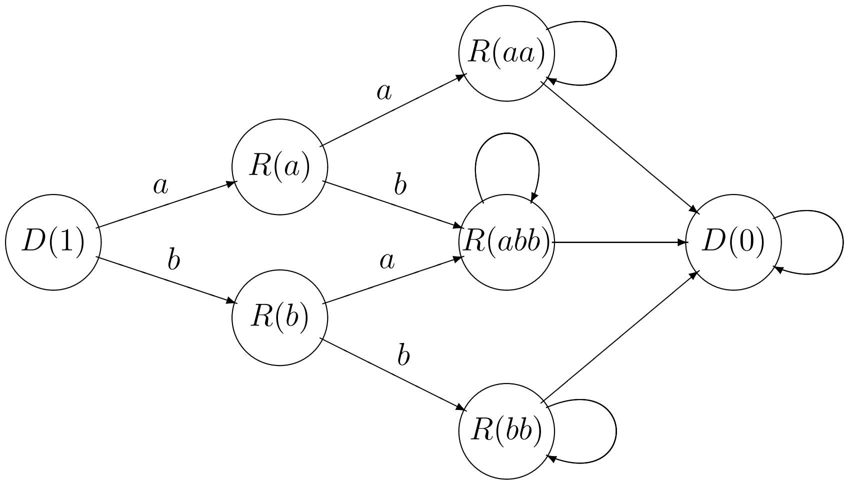

FIGURE 8. $\mathcal{R}$-classes inside the D-class structure of *Bicycle*

same D-class, so $x \in MyM$, that is, $x = pyq = pxzq$ for some p, q. We got hold of a $\mathcal{J}$-cycle for x, we pump it to get $x = p^k x(zq)^k$, for all k. Choose k so that p^k is an *idempotent* $e = ee$. This is always possible since M is finite. Then $x = ex(zq)^k = eex(zq)^k = ex$. Then we are done, because $x(zq)^k = ex(zq)^k = x$ and x extends xz as required. □

6.4. **Pumping loops in partially ordered automata.** In a partially ordered deterministic automaton, we can get a stronger result. Because a pump has to be formed from self-loops, it can be extended with substrings which form part of the pump.

Proposition 26. *Let M be a monoid in DA, $e = ee$ an idempotent in M and $x, y \in M$.*

(1) *If s is a factor of e, then $ese = e$. (This is a characterizing identity for DA.)*
(2) *If x, xy are in the same $\mathcal{R}$-class and s is a factor of y then $x\mathcal{R}xys$.*

Proof. For the first part, let $e = usv = (usv)^k = (usv)^{2k}$. In the same D-class we have $f = (svu)^k$. Since e is extended by ef, by the previous proposition they are in the same $\mathcal{R}$-class. So e, es are in the same $\mathcal{R}$-class, and $es = eses$ is an idempotent from the definition of DA. Hence $ese = esese = eseese$ is an idempotent which is in the same $\mathcal{R}$-class as well as in the same $\mathcal{L}$-class as e. Since monoids in DA have trivial groups, $ese = e$ fulfilling (1).

For part (2), since x is extended by xys, we have to extend xys in M to get back to x. Since x extends xy, let $x = xyz$. As s is a factor of y, let $y = psq$. So $xy = xyzy = xyzpsq$. Pumping up the $\mathcal{R}$-cycle, $xy = xy(zpsq)^k$, choosing k to make $(zpsq)^k$ an idempotent e. As s is a factor of e, by the DA equation from part (1) of this Proposition, $(zpsq)^k = (zpsq)^k s (zpsq)^k$. Then we are done, because $xy = xys(zpsq)^k$ extends xys as required. □

7. MONOIDS TO EXPRESSIONS

Now we can follow THÉRIEN AND WILKE's construction from a monoid in DA.

Theorem 27. *If the syntactic monoid of a language L is in DA, then it is described by an unambiguous expression.*

Proof. Let the homomorphism $h : A^* \to M$ into finite monoid M recognize L. Since the monoid is finite, with expressions for $h^{-1}(m),\ m \in M$, we can write L as their disjoint union. Since DA is a subclass of monoids which do not have nontrivial groups, it is sufficient to give an expression for $h^{-1}(\mathcal{H}(m))$ and then take unions.

Let k be the maximum of the number of $\mathcal{R}$- and $\mathcal{L}$-classes of M. Let u and v be words such that their combined content has ℓ letters. We show by induction on ℓ that if u and v have the same $\mathcal{R}$- and $\mathcal{L}$-factorizations of weight upto $k\ell$, then $h(u) = h(v)$ map to a single $\mathcal{H}(m) \in M$. Hence an unambiguous expression for these factorizations will define $h^{-1}(m)$.

In the base case $\ell = 0$, the expression is $\emptyset^* = \varepsilon$. For the induction step, consider $\ell > 0$ and u, v having the same content. Let $u = (u_0)(a_1u_1)\dots(a_tu_t),\ t \leq k$ be an $\mathcal{R}$-factorization of weight tl. Because M is in DA, by Proposition 26, a_{i+1} is a progress letter not occuring in u_i, for $i < t$, forming a factorization of $(u_i)(a_{i+1}u_{i+1})\dots(a_tu_t)$ after i initial factorizations. By hypothesis, v has a factorization of the same weight $(v_0)(a_1v_1)\dots(a_tv_t)$, where, for $i < t$ we have u_i and v_i over an alphabet of size $\ell - 1$ with weight $tl - i$. As $t(\ell-1) < tl - i$, the factors u_i, v_i have the same factorizations of weight $t(\ell - 1)$. Applying the induction hypothesis, they map to the same element $h(u_i) = h(v_i)$ which has an unambiguous expression e_i. This holds for all $i < t$. Putting all the factors together:

$$h(u)\ \mathcal{R}\ h(u_0\dots u_{t-1}a_t) = h(v_0\dots v_{t-1}a_t) \geq_{\mathcal{R}} h(v).$$

A symmetric argument gives $h(v) {\geq_{\mathcal{R}}} h(u)$ and hence $h(u)\mathcal{R}h(v)$. The unambiguous expression is $e(R) = (e_0 \cap A_0^*)a_1(e_1 \cap A_1^*)\dots a_tA_t^*$, where A_i are the subalphabets of the $\mathcal{H}$-classes of $h(u_i)$. By Proposition 26, we can write $e_0a_1e_1\dots a_tA_t^*$. Expressions of the form $A_0^*a_1\dots a_tA^*t$ are called *unambiguous monomials*. Assuming inductively that the e_i are disjoint unions of monomials, call these *unambiguous polynomials*, we can distribute the unions over the monomials and obtain an unambiguous polynomial for $e(R)$.

Working with the $\mathcal{L}$-factorization gives $h(u)\mathcal{L}h(v)$, so $h(u), h(v)$ lie in the same $\mathcal{H}$-class and have a unmabiguous polynomial $e(L) = B_0^*b_1\dots f_{t-1}b_tf_t$ for suitable progress letters b_i in the reverse direction.

The expression for $h^{-1}(m)$ is $e(L) \cap e(R)$. The intersection of unambiguous polynomials can be expanded out into a disjoint union of unambiguous monomials. Hence the induction is complete and we get at the end an unambiguous expression of some dot depth, as defined in Section 1. □

Corollary 28. *There is an algorithm to check, given a regular language, whether it is definable using a sentence of first-order logic with two variables $FO^2[<]$.*

Proof. As we said earlier, the converse of the theorem was obtained using a long chain of equivalences. The equation for DA can be checked on a monoid. □

7.1. **Starfree languages.** The proof below is close to Schützenberger's 1965 one, based on monoid ideals. A second proof of this theorem, based on the theory of implementing an automaton by a product of simpler automata (developed by Kenneth Krohn and John Rhodes in 1965), was first published by Albert Meyer in 1969. Thomas Wilke came up with a third proof, published in 1999. His proof directly produces an $FO^3[<]$ sentence.

Theorem 29. *If the syntactic monoid of L has only trivial subgroups, then it is starfree.*

Proof. Let the homomorphism $h : A^* \to M$ into finite monoid M recognize L. To simplify the proof, following PETER HIGGINS we assume M has a zero element, so that the D-class structure looks like the one in Figure 7. (This can be ensured, for example, by the alphabet having a "dead" letter.) The proof is by induction on the maximum length t of $\mathcal{J}$-factorization chains in M.

For the base case this is ≤ 2, the chain is at most $1 >_{\mathcal{J}} 0$ and $h^{-1}(1) = B^*$ for the subset B of the alphabet which maps to 1. This is starfree. $h^{-1}(0)$, its complement, is starfree.

For the induction step, we consider $>_{\mathcal{J}}$-chains of length more than 2. Because M has no nontrivial subgroups, it is sufficient to provide a starfree expression for $h^{-1}(\mathcal{H}(m)),\ m \in M$. The key idea is to do a case analysis on $F(m)$, the set of forbidden factors of m, using induction where applicable.

If $F(m)$ is of size at least two, then it must contain 0 and another element x such that $m >_{\mathcal{J}} x >_{\mathcal{J}} 0$. Quotient the monoid by the ideal to get $M/F(m)$, which will not have nontrivial groups. Further, its factorization lengths are at most $t-1$ since zero and the 0-minimal D-classes, including the one containing x, are collapsed into one new zero. So we get a starfree expression from the induction hypothesis.

If $F(m)$ is empty, then m is 0. $h^{-1}(0)$ is the union of languages A^*aA^*, where a maps to 0, together with the languages $A^*ah^{-1}(y)bA^*$, where $h(a)y$ and $yh(b)$ are nonzero but the product $h(a)yh(b) = 0$. If y were in a 0-minimal D-class, so would $h(a)y, yh(b)$ and hence $h(a)yh(b)$, contadicting its being zero. So the induction hypothesis applies to $h^{-1}(y)$.

We are left with the case that $F(m)$ is of size one, which means it has to be $\{0\}$. The set $\{x \mid |F(x)| = 1\}$ is a 0-minimal D-class. As there are $>_{\mathcal{J}}$-chains of length at least three, the $\geq_{\mathcal{J}}$-maximal element 1 is not in this D-class. So this is the hardest case.

By Exercise 24 on forbidden factors, the $\mathcal{H}$-class of m is $(mM \cap Mm) \setminus \{0\}$. We already saw that $h^{-1}(0)$ is starfree, so it is sufficient to give starfree expressions for the languages recognized by mM and Mm.

For $h^{-1}(mM)$ we use the $\mathcal{JR}$-factorization to obtain a union of languages $h^{-1}(y)aA^*$ where $y >_{\mathcal{J}} yh(a) \mathcal{R} m$, and for $h^{-1}(Mm)$ we use the $\mathcal{JL}$-factorization. These elements y are higher in the $>_{\mathcal{J}}$-chain and a marks the transition into the current D-class. So by the induction hypothesis, the $h^{-1}(y)$ languages and hence the $h^{-1}(mM)$ language are starfree. By a symmetric argument, so is $h^{-1}(Mm)$. □

Corollary 30. *There is an algorithm to check, given a regular language, whether it is definable using a sentence of first-order logic $FO[<]$.*

Proof. We saw the converse of the above theorem earlier using a long chain of implications, which are now all equivalences. Whether a monoid has nontrivial groups can be checked by an algorithm. □

Reading

We do not give a detailed bibliography of papers, but only refer to survey papers, theses and books from which the interested reader can trace back references.

Christian Dax. *From temporal logics to automata via alternation elimination*, PhD thesis, ETH Zurich (2010).

Volker Diekert, Paul Gastin and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words, *Int. J. Found. Comp. Sci.* **19**,3 (2008).

Deepak D'Souza and Priti Shankar, eds. *Modern applications of automata theory*, World Scientific (2012).

Robert McNaughton, Seymour Papert. *Counter-free automata*, MIT Press (1971).

Jean-Éric Pin. *Varieties of formal languages* (translated from the French by A. Howie), North Oxford (1986).

Gareth Rohde. *Alternating automata and the temporal logic of ordinals*, PhD thesis, U. Illinois at Urbana-Chanpaign (1997).

Simoni Shah. *Unambiguity and timed languages*, PhD thesis, TIFR (2012).

Imre Simon. *Hierarchies of events of dot-depth one*, PhD thesis, U. Waterloo (1972).

Howard Straubing. *Finite automata, formal logic, circuit complexity*, Birkhäuser (1994).

Howard Straubing and Pascal Weil. An introduction to finite automata and their connection to logic, in Deepak D'Souza and Priti Shankar, cited above.

Pascal Tesson and Denis Thérien. Logic meets algebra: the case of regular languages, *Log. Meth. Comp. Sci.* **3**,1 (2007).

Wolfgang Thomas. Languages, automata, and logic, in *Handbook of formal language theory* **III** (Grzegorz Rozenberg and Arto Salomaa, eds.), Springer (1997).

Thomas Wilke. *Classifying discrete temporal properties*, Habilitationsschrift (post-doctoral thesis), U. Kiel (1998).

The Institute of Mathematical Sciences, C.I.T. Campus, Chennai 600 113, India.