Algebraic language theory

Mikołaj Bojańczyk

March 17, 2020
The latest version can be downloaded from:
https://www.mimuw.edu.pl/ bojan/2019-2020/algebraic-language-theory-2020

Contents

Preface

These are lecture notes on the algebraic approach to regular languages. The classical algebraic approach is for finite words; it uses semigroups instead of automata. However, the algebraic approach can be extended to structures beyond words, e.g. infinite words, or trees or graphs.

PART ONE

WORDS

1

Semigroups, monoids and their structure

In this chapter, we define semigroups and monoids, and show how they can be used to recognise languages of finite words.

Definition 1.1 (Semigroup)**.** A *semigroup* consists of an underlying set S together with a binary product operation

$$(a, b) \mapsto ab,$$

that is associative in the sense that

$$a(bc) = (ab)c \qquad \text{for all } a, b, c \in S.$$

The definition says that the order of evaluation in a semigroup is not important, i.e. that different ways of parenthesising a sequence of elements in the monoid will yield the same result as far as far as the semigroup product is concerned. For example,

$$((ab)c)(d(ef)) = ((((ab)c)d)e)f.$$

Therefore, it makes sense to omit the parentheses and write simply

$$abcdef.$$

This means that the product operation in the semigroup can be seen as defined not just on pairs of semigroups elements, but also on all finite words consisting of semigroup elements.

A *semigroup homomorphism* if a function between semigroups that preserves the structure of semigroups, i.e. a function

$$h : \underbrace{S}_{\text{semigroup}} \to \underbrace{T}_{\text{semigroup}}$$

which is consistent with the product operation in the sense that

$$h(a \cdot b) = h(a) \cdot h(b),$$

where the semigroup product on the left is in S, and the semigroup product on the right is in T.

A *monoid* is the special case of a semigroup where there is an identity element, denoted by $1 \in S$, which satisfies

$$1a = a1 \qquad \text{for all } a \in S.$$

The identity element, if it exists, must be unique. This is because if there are two candidates for the identity, then taking their product reveals the true identity. A *monoid homomorphism* is a semigroup homomorphism that preserves the identity element.

Example 1.2. Here are some examples of monoids and semigroups.

(1) If Σ is a set, then the set Σ^+ of nonempty words over Σ, equipped with concatenation, is a semigroup, called the *free*[1] *semigroup over generators* Σ. The *free monoid* is the set Σ^* of possibly empty words.

(2) Every group is a monoid.

(3) For every set Q, the set of all functions $Q \to Q$, equipped with function composition, is a monoid. The monoid identity is the identity function.

(4) For every set Q, the set of all binary relations on Q is a monoid, when equipped with relational composition

$$a \circ b = \{(p, q) : \text{there is some } r \in Q \text{ such that } (p, r) \in a \text{ and } (r, q) \in b\}.$$

The monoid identity is the identity function. The monoid from the previous item is a submonoid of this one, i.e. the inclusion map is a monoid homomorphism.

(5) Here are all semigroups of size two, up to semigroup isomorphism:

$$\underbrace{(\{0,1\},+)}_{\text{addition mod 2}} \qquad (\{0,1\},\times) \qquad \underbrace{(\{0,1\},\pi_1)}_{\text{product } ab \text{ is } a} \qquad \underbrace{(\{0,1\},\pi_2)}_{\text{product } ab \text{ is } b}$$

The first two are monoids.

[1] The reason for this name is the following universality property. The free semigroup is generated by Σ, and it is the biggest semigroup generated by Σ in the following sense. For every semigroup S that is generated by Σ, there exists a (unique) surjective semigroup homomorphism $h : \Sigma^+ \to S$ which is the identity on the Σ generators.

Semigroup homomorphisms are closely related with functions that are compositional in the sense defined below. Let S be a semigroup, and let X be a set (without a semigroup structure). A function

$$h : S \to X$$

is called *compositional* if for every $a, b \in S$, the value $h(a \cdot b)$ is uniquely determined by the values $h(a)$ and $h(b)$. If X has a semigroup structure, then every semigroup homomorphism $S \to X$ is a compositional function. The following lemma shows that the converse is also true for surjective functions.

Lemma 1.3. *Let S be a semigroup, let X be a set, and let $h : S \to X$ be a surjective compositional function. Then there exists (a unique) semigroup structure on X which makes h into a semigroup homomorphism.*

Proof Saying that $h(a \cdot b)$ is uniquely determined by $h(a)$ and $h(b)$, as in the definition of compositionality, means that there is a binary operation $\circ$ on X, which is not yet known to be associative, that satisfies

$$h(a \cdot b) = h(a) \circ h(b) \qquad \text{for all } a, b \in S. \tag{1.1}$$

The semigroup structure on X uses $\circ$ as the semigroup operation. It remains to prove associativity of $\circ$. Consider three elements of X, which can be written as $h(a), h(b), h(c)$ thanks to the assumption on surjectivity of h. We have

$$(h(a) \circ h(b)) \circ h(c) \overset{(1.1)}{=} (h(ab)) \circ h(c) \overset{(1.1)}{=} h(abc).$$

The same reasoning shows that $h(a) \circ (h(b) \circ h(c))$ is equal to $h(abc)$, thus establishing associativity. □

Exercise 1. Show a function between two monoids that is a semigroup homomoprhism, but not a monoid homomorphism.

Exercise 2. Show that there are exponentially many semigroups of size n.

Exercise 3. Show that for every semigroup homomorphism $h : \Sigma^+ \to S$, with S finite, there exists some $N \in \{1, 2, \ldots\}$ such that every word of length at least N can be factorised as $w = w_1 w_2 w_3$ where $h(w_2)$ is an idempotent[2].

Exercise 4. Show that if S is a semigroup, then the same is true for the *powerset semigroup*, whose elements are possibly empty subsets of S, and where

[2] This exercise can be seen as the semigroup version of the pumping lemma.

the product is defined coordinate-wise:

$$A \cdot B = \{a \cdot b : a \in A, b \in B\} \qquad \text{for } A, B \subseteq S.$$

Exercise 5. Let us view semigroups as a category, where the objects are semigroups and the morphisms are semigroup homomorphisms. What are the product and co-products of this category?

Exercise 6. Let Σ be an alphabet, and let

$$X \subseteq \Sigma^+ \times \Sigma^+$$

be a set of words pairs. Define $\sim_X$ to be least congruence on Σ^+ which contains all pairs from X. This is the same as the symmetric transitive closure of

$$\{(wxv, wyv) : w, v \in \Sigma^*, (x, y) \in X\}.$$

Show that the following problem – which is called the *word problem for semigroups* – is undecidable: given finite Σ, X and $w, v \in \Sigma^+$, decide if $w \sim_X v$.

1.1 Recognising languages

In this book, we are interested in monoids and semigroups as an alternative to finite automata for the purpose of recognising languages. Since languages are usually defined for possibly empty words, we use monoids and not semigroups when recognising languages.

Definition 1.4. Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is *recognised* by a monoid homomorphism

$$h : \Sigma^* \to M$$

if membership in $w \in L$ is determined uniquely by $h(w)$. In other words, there is a subset $F \subseteq M$ such that

$$w \in L \quad \text{iff} \quad h(w) \in F \qquad \text{for every } w \in \Sigma^*.$$

We say that a language is recognised by a monoid if it is recognised by some monoid homomorphism into that monoid. The following theorem shows that, for the purpose of recognising languages, finite monoids and finite automata are equivalent.

Theorem 1.5. *The following conditions are equivalent for every $L \subseteq \Sigma^*$:*

(1) *L is recognised by a finite nondeterministic automaton;*
(2) *L is recognised by a finite monoid.*

Proof

2 ⇒ 1 From a monoid homomorphism one creates a deterministic automaton, whose states are elements of the monoid, the initial state is the identity, and the transition function is

$$(m, a) \mapsto m \cdot (\text{homomorphic image of } a).$$

After reading an input word, the state of the automaton is its homomorphic image, and therefore the accepting state from the monoid homomorphisms can be used. This automaton computes the monoid product according to the choice of parentheses illustrated in this example:

$$(((((ab)c)d)e)f)g.$$

1 ⇒ 2 Let Q be the states of the nondeterministic automaton recognising L. Define a function[3]

$$\delta : \Sigma^* \to \text{monoid of binary relations on } Q$$

which sends a word w to the binary relation

$$\{(p, q) \in Q^2 : \text{some run over } w \text{ goes from } p \text{ to } q\}.$$

This is a monoid homomorphism. It recognises the language: a word is in the language if and only if its image under the homomorphism contains at least one (initial, accepting) pair.

□

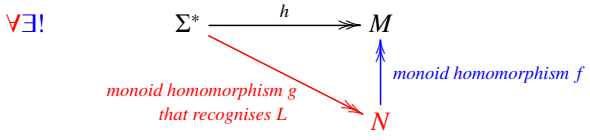
The syntactic monoid of a language. Deterministic finite automata have minimisation, i.e. for every language there is a minimal deterministic automaton, which can be found inside every other deterministic automaton that recognises the language. The same is true for monoids, as proved in the following theorem.

Theorem 1.6. *For every language*[4] $L \subseteq \Sigma^*$ *there is a surjective monoid homomorphism*

$$h : \Sigma^* \to M,$$

[3] This transformation from a nondeterministic (or deterministic) finite automaton to a monoid incurs an exponential blow-up, which is unavoidable in the worst case.
[4] The language need not be regular, and the alphabet need not be finite.

called the syntactic homomorphism of L, which recognises it and is minimal in the sense explained in the following quantified diagram[5]



Proof The proof is the same as for the Myhill-Nerode theorem about minimal automata, except that the corresponding congruence is two-sided. Define the *syntactic congruence* of L to be the equivalence relation $\sim$ on Σ^* which identifies two words $w, w' \in \Sigma^*$ if

$$uwv \in L \quad \text{iff} \quad uw'v \in L \qquad \text{for all } u, v \in \Sigma^*.$$

Define h to be the function that maps a word to its equivalence class under syntactic congruence. It is not hard to see that h is compositional, and therefore by (the monoid version of) Lemma 1.3, one can equip the set of equivalence classes of syntactic congruences with a monoid structure – call M the resulting monoid – which turns h into a monoid homomorphism.

It remains to show minimality of h, as expressed by the diagram in the lemma. Let then g be as in the diagram. Because g recognises the language L, we have

$$g(w) = g(w') \quad \text{implies} \quad w \sim w',$$

which, thanks to surjectivity of g, yields some function f from N to M, which makes the diagram commute, i.e. $h = f \circ g$. Furthermore, f must be a monoid homomorphism, because

$$\begin{aligned}
f(a_1 \cdot a_2) &= && \text{(by surjectivity of } g\text{, each } a_i \text{ can be presented as } g(w_i) \text{ for some } w_i) \\
f(g(w_1) \cdot g(w_2)) &= && (g \text{ is a monoid homomorphism}) \\
f(g(w_1 w_2)) &= && \text{(the diagram commutes)} \\
h(w_1 w_2) &= && (h \text{ is a monoid homomorphism}) \\
h(w_1) \cdot h(w_2) &= && \text{(the diagram commutes)} \\
f(g(w_1)) \cdot f(g(w_2)) &= \\
f(a_1) \cdot f(a_2).
\end{aligned}$$

[5] Here is how to read the diagram. For every red extension of the black diagram there exists a unique blue extension which makes the diagram commute. Double headed arrows denote surjective homomorphisms, which means that ∀ quantifies over surjective homomorphisms, and the same is true for ∃!.

□

Exercise 7. Show that the translation from deterministic finite automata to monoids is exponential in the worst case.

Exercise 8. Show that the translation from (left-to-right) deterministic finite automata to monoids is exponential in the worst case, even if there is a right-to-left deterministic automaton of same size.

Exercise 9. Which languages are recognised by finite commutative monoids?

Exercise 10. Prove that surjectivity of g is important in Theorem 1.6.

Exercise 11. Show that for every language, not necessarily regular, its syntactic homomorphism is the function

$$w \in \Sigma^* \qquad \mapsto \qquad \underbrace{(q \mapsto qw)}_{\substack{\text{state transformation} \\ \text{in the sytactic automaton}}},$$

where the syntactic automaton is the deterministic finite automaton from the Myhill-Nerode theorem.

1.2 Green's relations and the structure of finite semigroups

In this section, we describe some of the structural theory of finite semigroups. This theory is based on Green's relations, which are pre-orders in a semigroup that are based on prefixes, suffixes and infixes.

We begin with idempotents, which are ubiquitous in the analysis of finite semigroups. A semigroup element e is called *idempotent* if it satisfies

$$ee = e.$$

Example 1.7. In a group, there is a unique idempotent element, namely the group identity. There can be several idempotent elements, for example all elements are idempotent in the semigroup

$$(\{1, \ldots, n\}, \max).$$

One can think of idempotents as being a relaxed version of identity elements.

Lemma 1.8 (Idempotent Power Lemma)**.** *Let S be a finite semigroup. For every $a \in S$, there is exactly one idempotent in the set*

$$\{a^1, a^2, a^3, \ldots\} \subseteq S.$$

Proof Because the semigroup is finite, the sequence $a^1, a^2, \ldots$ must contain a repetition, i.e. there must exist $n, k \in \{1, 2, \ldots\}$ such that

$$a^n = a^{n+k} = a^{n+2k} = \cdots .$$

After multiplying both sides of the above equation by a^{nk-n} we get

$$a^{nk} = a^{nk+k} = a^{nk+2k} = \cdots ,$$

and therefore $a^{nk} = a^{nk+nk}$ is an idempotent. To prove uniqueness of the idempotent, suppose $n_1, n_2 \in \{1, 2, \ldots\}$ are powers such that that a^{n_1} and a^{n_2} are idempotent. The we have

$$\underbrace{a^{n_1} = (a^{n_1})^{n_2}}_{\substack{\text{because } a^{n_1} \\ \text{is idempotent}}} = a^{n_1 n_2} = \underbrace{(a^{n_1})^{n_2} = a^{n_2}}_{\substack{\text{because } a^{n_2} \\ \text{is idempotent}}}$$

□

We use the name *idempotent power* for the unique idempotent in the above lemma. Note that it is the resulting semigroup element a^n that is unique, and not the exponent n. Finiteness is crucial for the above lemma, for example the infinite semigroup

$$(\{1, 2, \ldots\}, +)$$

contains no idempotents. The analysis presented in the rest of this chapter will hold in any semigroup which satisfies the conclusion of the Idempotent Power Lemma.

Green's relations

We now give the main definition of this chapter.

Definition 1.9 (Green's relations)**.** Let a, b be elements of a semigroup S. We say that a is a *prefix* of b if there exists a solution x of

$$ax = b.$$

The solution x can be an element of the semigroup, or empty (i.e. $a = b$). Likewise we define the suffix and infix relations, but with the equations

$$\underbrace{xa = b}_{\text{suffix}} \qquad \underbrace{xay = b}_{\text{infix}}.$$

In the case of the infix relation, one or both of x and y can be empty.

Figure 1.1 shows a monoid along with the accompanying Green's relations. The prefix, suffix and infix relations are pre-orders, i.e. they are transitive and reflexive[6]. They need not be anti-symmetric, for example in a group every element is an prefix (suffix, infix) of every other element. We say that two elements of a semigroup are in the same *prefix class* if they are prefixes of each other. Likewise we define *suffix classes* and *infix classes*.

Clearly every prefix class is contained in some infix class, because prefixes are special cases of infixes. Therefore, every infix class is partitioned into prefix classes. For the same reasons, every infix class is partitioned into suffix classes. The following lemma describes the structure of these partitions.

Lemma 1.10 (Eggbox lemma)**.** *The following hold in every finite semigroup.*

(1) *all distinct prefix classes in a given infix class are incomparable:*

$$a, b \text{ are infix equivalent, and } a \text{ is a prefix of } b \implies a, b \text{ are prefix equivalent}$$

(2) *if a prefix class and a suffix class are contained in the same infix class, then they have nonempty intersection;*
(3) *all prefix classes in the same infix class have the same size.*

Of course, by symmetry, the lemma remains true after swapping infixes with suffixes.

Proof

[6] Another description of the prefix pre-order is that a is a prefix of b if

$$aS^1 \supseteq bS^1. \tag{1.2}$$

In the above, S^1 is the monoid obtained from S by adding an identity element, unless it was already there. The sets aS^1, bS^1 are called *right ideals*. Because of the description in terms of inclusion of right ideals, the semigroup literature uses the notation

$$a \geq_{\mathcal{R}} b \stackrel{\text{def}}{=} aS^1 \supseteq bS^1$$

for the prefix relation. Likewise, $a \geq_{\mathcal{L}} b$ is used for the prefix relation, which is defined in terms of left ideals. Also, for some mysterious reason, $a \geq_{\mathcal{J}} b$ is used for the infix relation. We avoid this notation, because it makes longer words smaller.

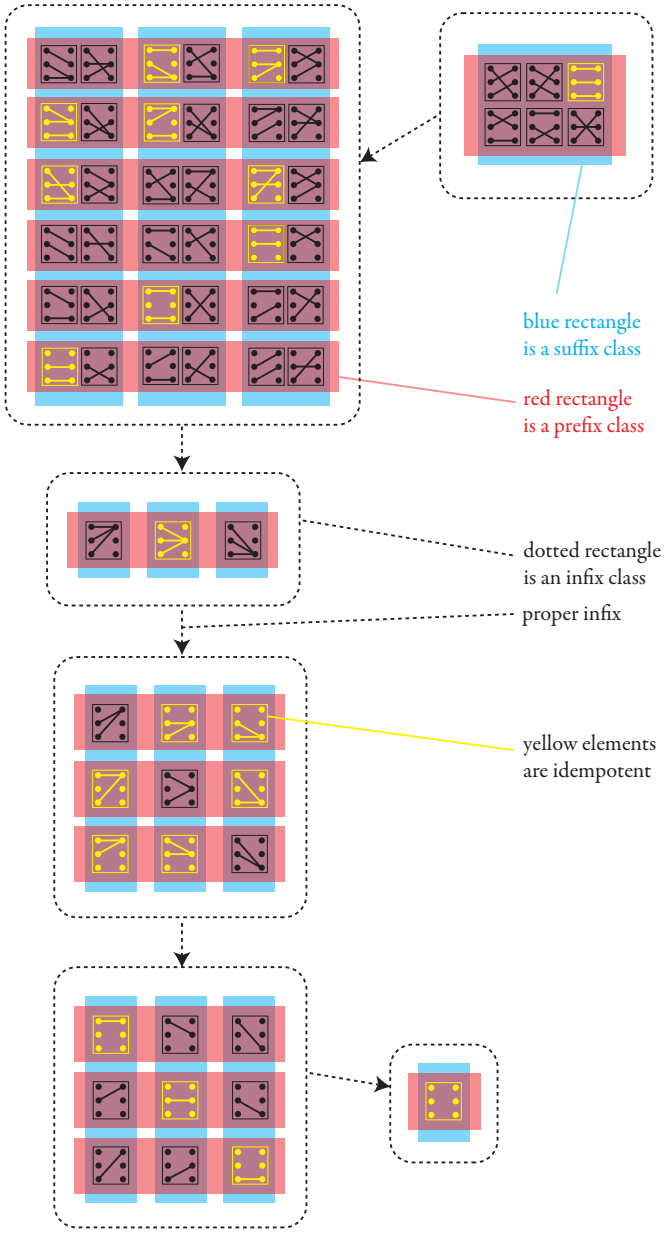


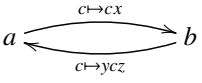
Figure 1.1 The semigroup of partial functions from a three element set to itself, partitioned into prefix, suffix and infix classes. In this particular example, the infix classes are totally ordered, which need not be the case in general.

(1) This item says that distinct prefix classes in the same infix class are incomparable, with respect to the prefix relation. This item of the Eggbox Lemma is the one that will be used most often.

Suppose that a, b are infix equivalent and a is a prefix of b, as witnessed by solutions x, y, z to the equations

$$b = ax \qquad a = ybz.$$

As usual, each of x, y, z could be empty. This can be illustrated as



By the Idempotent Power Lemma, there is some $n \in \{1, 2, \ldots\}$ such that y^n is idempotent. By following the loop around a in the above diagram n times, and then going to b, we get

$$\begin{aligned} b &= && \text{(follow } 2n \text{ times the loop around } a\text{, then go to } b\text{)} \\ y^{2n}a(xz)^{2n}x &= && \text{(}y^n \text{ is an idempotent)} \\ y^n a(xz)^{2n}x &= && \text{(follow } n \text{ times the loop around } a\text{)} \\ a(xz)^n z, \end{aligned}$$

which establishes that b is a prefix of a, and therefore a, b are in the same prefix class.

(2) We now show that prefix and suffix classes in the same infix class must intersect. Suppose that a, b are in the same infix class, as witnessed by

$$a = xby.$$

With respect to the infix relation, by is between b and a, and therefore it must be in the same infix class as both of them. We have

$$\overbrace{x\ \underbrace{b\ y}}^{by \text{ is a suffix of } xby = a} \qquad ,$$

b is a prefix of by

and therefore, thanks to the previous item, by is prefix equivalent to b and suffix equivalent to a. This witnesses that the prefix class of b and the suffix class of a have nonempty intersection.

(3) We now show that all prefix classes in the same infix class have the same size. Take some two prefix classes in the same infix class, given by representatives a, b. We can assume that a, b are in the same suffix class, thanks to the previous item. Let

$$a = xb \qquad b = ya$$

be witnesses for the fact that a, b are in the same suffix class. The following claim implies that the two prefix classes under consideration have the same size.

Claim 1.11. *The following maps are mutually inverse bijections*

$$\text{prefix class of } a \underset{c \mapsto xc}{\overset{c \mapsto yc}{\rightleftarrows}} \text{prefix class of } b$$

Proof Suppose that c is in the prefix class of a, as witnessed by a decomposition $c = az$. If we apply sequentially both maps in the statement of the claim to c, then we get

$$xyc = xyaz \overset{ya=b}{=} xbz \overset{xb=a}{=} az \overset{az=c}{=} c.$$

This, and a symmetric argument for the case when c is in the prefix class of b, establishes that the maps in the statement of the claim are mutually inverse. It remains to justify that the images of the maps are as in the statement of the claim, i.e. the image of the top map is the prefix class of b, and the image of the bottom map is the prefix class of a. Because the two maps are mutually inverse, and they prepend elements to their inputs, it follows that each of the maps has its image contained in the infix class of a, b. To show that the image of the top map is in the prefix class of b (a symmetric argument works for the bottom map), we observe that every element of this image is of the form yaz, and therefore it has $b = ya$ as a prefix, but it is still in the same infix class as a, b as we have observed before, and therefore it must be prefix equivalent to b thanks to the item (1) of the lemma. □

□

The Eggbox Lemma establishes that each infix class has the structure of a rectangular grid (which apparently is reminiscent of a box of eggs), with the rows being prefix classes and the columns being suffix classes. Let us now look at the eggs in the box: define an $\mathcal{H}$-class to be a nonempty intersection of some prefix class and some suffix class. By item (2) of the Eggbox Lemma, every pair of prefix and suffix classes in the same infix class lead to some $\mathcal{H}$-class. The following lemma shows that all $\mathcal{H}$-classes in the same infix class have the same size.

Lemma 1.12. *If a, b are in the same infix class, then there exist possibly empty x, y such that the following is a bijection*

$$\mathcal{H}\text{-class of } a \xrightarrow{c \mapsto xcy} \mathcal{H}\text{-class of } b$$

Proof Consider first the special case of the lemma, when a and b are in the same suffix class. Take the map from Claim 1.11, which maps bijectively the prefix class of a to the prefix class of b. Since this map preserves suffix classes, it maps bijectively the $\mathcal{H}$-class of a to the $\mathcal{H}$-class of b. By a symmetric argument, the lemma is also true when a and b are in the same prefix class.

For the general case, we use item (2) of the Eggbox Lemma, which says that there must be some intermediate element that is in the same prefix class as a and in the same suffix class as b, and we can apply the previously proved special cases to go from the $\mathcal{H}$-class of a to the $\mathcal{H}$-class of the intermediate element, and then to the $\mathcal{H}$-class of b. □

The following lemma shows a dichotomy for an $\mathcal{H}$-class: either it is a group, or the the product of every two elements in that $\mathcal{H}$-class falls outside the infix class.

Lemma 1.13 ($\mathcal{H}$-class Lemma)**.** *The following conditions are equivalent for every $\mathcal{H}$-class G in a finite semigroup:*

(1) *G contains an idempotent;*
(2) *ab is in the same infix class as a and b, for some $a, b \in G$;*
(3) *$ab \in G$ for some $a, b \in G$;*
(4) *$ab \in G$ for all $a, b \in G$;*
(5) *G is a group (with product inherited from the semigroup)*

Proof Implications (5) $\Rightarrow$ (1) $\Rightarrow$ (2) in the lemma are obvious, so we focus on the remaining implications.

(2)$\Rightarrow$(3) Suppose that ab is in the same infix class as a and b. Since a is a prefix of ab, and the two elements are in the same infix class, item (1) of the Eggbox Lemma implies that ab is in the prefix class of a, which is the same as the prefix class of b. For similar reasons, ab is in the same suffix class as a and b, and therefore $ab \in G$.

(3)$\Rightarrow$(4) Suppose that there exist $a, b \in G$ with $ab \in G$. We need to show that G contains the product of every elements $c, d \in G$. Since c is prefix equivalent to a there is a decomposition $a = xc$, and for similar reasons there is a decomposition $b = dy$. Therefore, cd is an infix of

$$\overbrace{xc}^{a}\overbrace{dy}^{b} \in G,$$

and therefore it is in the same infix class as G. Since c is a prefix of cd, and both are in the same infix class, the Eggbox Lemma implies that cd is in the prefix class of c. For similar reasons cd is in the suffix class of d. Therefore, $cd \in G$.

(4)⇒(5) Suppose that G is closed under products, i.e. it is a subsemigroup. We will show that it is a group. By the Idempotent Power Lemma, G contains some idempotent, call it e. We claim that e is an identity element in G, in particular it is unique. Indeed, let $a \in G$. Because a and e are in the same suffix class, it follows that a can be written as xe, and therefore

$$ae = xee = xe = a.$$

For similar reasons, $ea = a$, and therefore e is an identity element in G. The group inverse is defined as follows. For $a \in G$, choose some $k \in \{2, 3, \ldots\}$ such that a^k is idempotent, such k exists by Lemma 1.8. Since there is only one idempotent in G, we have $a^k = e$. Therefore, a^{k-1} is a group inverse of a.

□

Exercise 12. Show that for every finite monoid, the infix class of the monoid identity is a group.

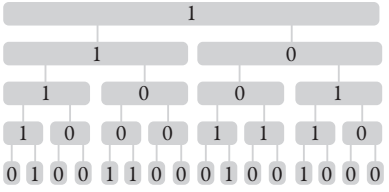
Exercise 13. Consider a finite semigroup. Show that an infix class contains an idempotent if and only if it is *regular*, which means that there exist a, b in the infix class such that ab is also in the infix class.

Exercise 14. Show that if G_1, G_2 are two $\mathcal{H}$-classes in the same infix class of a finite semigroup, and they are both groups, then they are isomorphic as groups[7].

1.3 The Factorisation Forest Theorem

In this section, we show how products in a semigroup can be organised in trees, so that in each node of the tree the products are very simple. The most natural way to do this is to have binary tree, as in the following example, which uses the two semigroup $\{0, 1\}$ with addition modulo 2:

[7] Let us combine Exercises 13 and 14. By Exercises (13) and the $\mathcal{H}$-class lemma, an infix class is regular if and only if it contains an $\mathcal{H}$-class which is a group. By Exercise (14), the corresponding group is unique up to isomorphism. This group is called the *Schützenberger group* of the regular infix class.



We use the name *factorisation tree* for structures as in the above picture: a *factorisation tree over a semigroup* S is a tree, where nodes are labelled by semigroup elements, such that every node is either a leaf, or is labelled by the semigroup product of the labels of its children. A binary factorisation tree is one where every node has zero or two children. For every word in S^+, one can find a corresponding factorisation tree (i.e. one where the word is obtained by reading the leaves left-to-right) whose height (i.e. the maximal number of edges on a root-to-leaf path) is logarithmic in the length of the word.

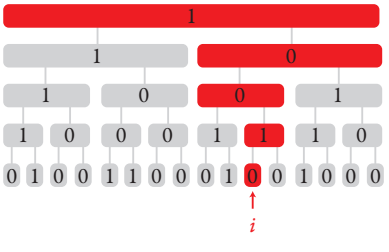
Binary factorisation trees are a natural data structure for several problems about regular languages.

Example 1. Fix a regular language $L \subseteq \Sigma^*$. Consider the following dynamic problem. We begin with some word in Σ^*. We want to build a data structure that handles efficiently the following updates and queries:

Query. Is the current word in L?

Update. Change the label of position i to $a \in \Sigma$.

To solve this problem, as the data structure we can use a binary factorisation tree with respect to some semigroup that recognises the language. If we assume that the regular language is fixed and not part of the input, then the queries are processed in constant time, by checking if the root label of the factorisation tree. The updates are processed in time proportional to the height of the factorisation tree, by updating all of the nodes on the path from the updates position to the root, as in the following picture:

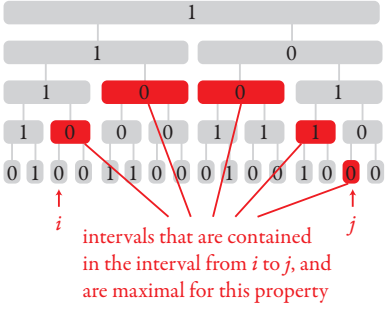


If the factorisation tree is chosen to be balanced, then the updates are processed in logarithmic time. □

Example 2. Fix a regular language $L \subseteq \Sigma^*$. Consider the following dynamic problem. We begin with some word in Σ^*. We want to build a data structure that handles efficiently the following queries (there are no updates):

Query. Given positions $i \leq j$, does L contain the infix from i to j?

As in the previous example, we solve the problem using a binary factorisation tree, with respect to some semigroup recognising the language. Suppose that the tree has height k. Each node of the factorisation tree corresponds to an infix of the underlying word. The infix from i to j can be partitioned into at most $2k$ intervals, each of which corresponds to a node of the tree, as in the following picture:



Therefore, the queries can be processed in time proportional to the height of the tree, which can be assumed to be logarithmic in the length of the underlying word. □

We will now show a data structure which will allow constant time query processing in the problem from Example 2. We will also use a variant of factorisation trees, except that non-binary nodes will need to be used. The problem in Example 1 cannot be solved in constant time[8].

[8] Lower bounds for this problem can be seen in
[5] Frandsen, Miltersen, and Skyum, "Dynamic Word Problems", 1997 , Fig. 1

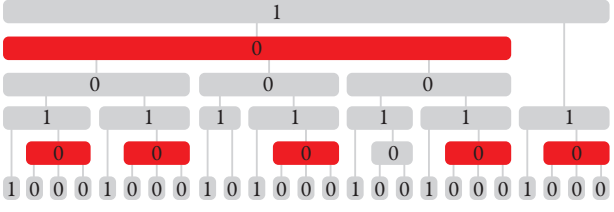
Simon trees

A Simon tree is a factorisation tree which allows nodes of degree higher than 2, but these nodes must have idempotent children. The data structure is named after Imre Simon, who introduced it[9].

Definition 1.14 (Simon Tree)**.** Define a *Simon tree* to be a factorisation tree where every non-leaf node has one (or both) of the following types:

binary: has two children; or
idempotent: all children have the same label, which is an idempotent.

Here is a picture of a Simon tree for the semigroup $\{0, 1\}$ with addition modulo 2, with idempotent nodes drawn in red:



The main result about Simon trees is that their height can be bounded by a constant that depends only on the semigroup, and not the underlying word.

Theorem 1.15 (Factorisation Forest Theorem)**.** *Let S be a finite semigroup. Every word in S^+ admits a Simon tree of height*[10] *$< 5|S|$.*

The rest of this chapter is devoted to proving the theorem.

Groups. We begin with the special case of groups.

Lemma 1.16. *Let G be a finite group. Every word in G^+ admits a Simon tree of height $< 3|G|$.*

Proof Define the *prefix set* of a word $w \in G^+$ to be the set of group elements that can be obtained by taking a product of some nonempty prefix of w. By induction on the size of the prefix set, we show that every $w \in G^+$ has a Simon

[9] Under the name Ramseyan factorisation forests, in
[11] Simon, "Factorization Forests of Finite Height", 1990 , 69

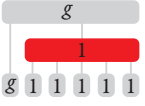
[10] The first version of this theorem was proved in [11, Theorem 6.1], with a bound of $9|S|$. The optimal bound is $3|S|$, which was shown in
[8] Kufleitner, "The Height of Factorization Forests", 2008 , Theorem 1
The proof here is based on Kufleitner, with some optimisations removed.

tree of height strictly less than 3 times the size of the prefix set. Since the prefix set has maximal size $|G|$, this proves the lemma.

The induction base is when the prefix set is a singleton $\{g\}$. This means that the first letter is g, and every other letter h satisfies $gh = g$. In a group, only the group identity $h = 1$ can satisfy $gh = g$, and therefore h is the group identity. In other words, if the prefix set is $\{g\}$, then the word is of the form

$$g \underbrace{1 \cdots 1}_{\text{a certain number of times}}.$$

Such a word admits a Simon tree as in the following picture:



The height of this tree is 2, which is strictly less than three times the size of the prefix set.

To prove the induction step, we show that every $w \in G^+$ admits a Simon tree, whose height is at most 3 plus the size from the induction assumption. Choose some g in the prefix set of w. Decompose w into factors as

$$w = \underbrace{w_1 w_2 \cdots w_{n-1}}_{\substack{\text{nonempty} \\ \text{factors}}} \underbrace{w_n}_{\substack{\text{could} \\ \text{be empty}}}$$

by cutting along all prefixes with product g. For the same reasons as in the induction base, every factor w_i with $1 < i < n$ has a product which is the group identity.

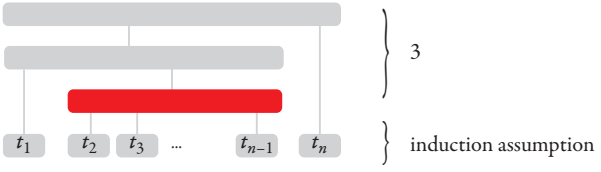
Claim 1.17. *The induction assumption applies to all of* $w_1, \ldots, w_n$.

Proof For the first factor w_1, the induction assumption applies, because its prefix set omits g. For the remaining blocks, we have a similar situation, namely

$$g \cdot (\text{prefix set of } w_i) \subseteq (\text{prefix set of } w) - \{g\} \qquad \text{for } i \in \{2, 3, \ldots, n\},$$

where the left side of the inclusion is the image of the prefix set under the operation $x \mapsto gx$. Since this operation is a permutation of the group, it follows that the left size of the inclusion has smaller size than the prefix set of w, and therefore the induction assumption applies. □

By the above claim, we can apply the induction assumption to compute Simon trees $t_1, \ldots, t_n$ for the factors $w_1, \ldots, w_n$. To get a Simon tree for the whole word, we join these trees as follows:

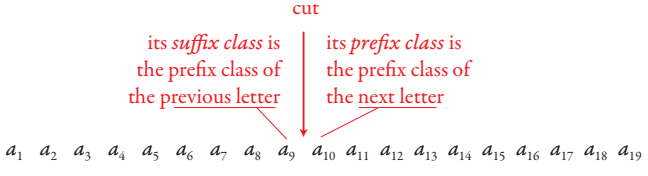


The gray nodes are binary, and the red node is idempotent because every w_i with $1 < i < n$ evaluates to the group identity. □

Smooth words. In the next step, we prove the theorem for words where all infixes come from the same infix class. We say that a word $w \in S^+$ is *smooth* if all of its nonempty infixes have product in the same infix class. The following lemma constructs Simon trees for smooth words.

Lemma 1.18. *If a word is smooth, and the corresponding infix class is $J \subseteq S$, then it admits a Simon tree of height $< 4|J|$.*

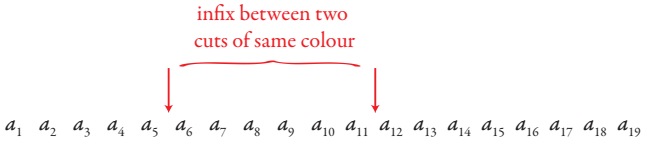
Proof Define a *cut* in a word to be the space between two consecutive letters; in other words this is a decomposition of the word into a nonempty prefix and a nonempty suffix. For a cut, define its *prefix* and *suffix* classes as in the following picture:



For every cut, both the prefix and suffix classes are contained in J, and therefore they have nonempty intersection thanks to item (2) of the Eggbox Lemma. This nonempty intersection is an $\mathcal{H}$-class, which is defined to be the *colour* of the cut. The following claim gives the crucial property of cuts and their colours.

Claim 1.19. *If two cuts have the same colour H, then the infix between them has product in H.*

Proof Here is a picture of the situation:



The infix begins with a letter from the prefix class containing H. Since the infix is still in the infix class J, by assumption on smoothness, it follows from item (1) of the Eggbox Lemma that the product of the infix is in the prefix class of H. For the same reason, the product of the infix is in the suffix class of H. Therefore, it is in H. □

Define the *colour set* of a word to be the set of colours of its cuts; this is a subset of the $\mathcal{H}$-classes in J. Thanks to Lemma 1.11, all $\mathcal{H}$-classes contained in J have the same size, and therefore it makes sense to talk about about the $\mathcal{H}$-class size in J, without specifying which $\mathcal{H}$-class is concerned.

Claim 1.20. *Every J-smooth word has a Simon tree of height at most*

$$|\textit{colour set of } w| \cdot (3 \cdot \mathcal{H}\textit{-class size} + 1).$$

Before proving the claim, we show how it implies the lemma. Since the number of possible colours is the number of $\mathcal{H}$-classes, the maximal height that can arise from the claim is

$$3 \cdot |J| + (\text{maximal size of colour set}) < 4|J|.$$

It remains to prove the claim.

Proof Induction on the size of the colour set. The induction base is when the colour set is empty. In this case the word has no cuts, and therefore it is a single letter, which is a Simon tree of height zero.

Consider the induction step. Let w be a smooth word. To prove the induction step, we will find a Simon tree whose height is at most the height from the induction assumption, plus

$$3 \cdot (\mathcal{H}\text{-class size}) + 1.$$

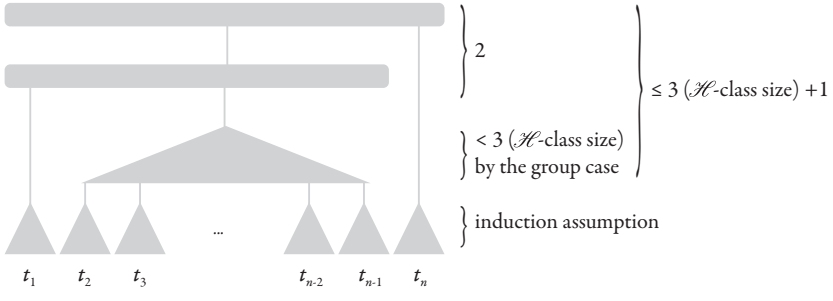
Choose some colour in the colour set of w, which is an $\mathcal{H}$-class H. Cut the word w along all cuts with colour H, yielding a decomposition

$$w = w_1 \cdots w_n.$$

None of the words $w_1, \ldots, w_n$ contain a cut with colour H, so the induction assumption can be applied to yield corresponding Simon trees $t_1, \ldots, t_n$.

If $n \leq 3$, then the Simon trees from the induction assumption can be combined using binary nodes, increasing the height by at most 2, and thus staying within the bounds of the claim.

Suppose now that $n \geq 4$. By Claim 1.19, any infix between two cuts of colour H has product in H. In particular, all $w_2, \ldots, w_{n-1}$ have product in H, and the same is true for $w_2 w_3$. It follows that H contains at least one product of two elements from H, and therefore H is a group thanks to item (3) of the $\mathcal{H}$-class Lemma. Therefore, the we can apply the group case from Lemma 1.16 to join the trees $t_2, \ldots, t_{n-1}$. The final Simon tree looks like this:



□

□

General case. We now complete the proof of the Factorisation Forest Theorem. The proof is by induction on the *infix height* of the semigroup, which is defined to be the longest chain that is strictly increasing in the infix ordering. If the infix height is one, then the semigroup is a single infix class, and we can apply Lemma 1.18 since all words in S^+ are smooth. For the induction step, suppose that S has infix height at least two, and let $T \subseteq S$ be the elements which have a proper infix. It is not hard to see that T is a subsemigroup, and its induction parameter is smaller.

Consider a word $w \in S^+$. As in Lemma 1.18, define a cut to be a space between two letters. We say that a cut is *smooth* if the letters preceding and following the cut give a two-letter word that is smooth.

Claim 1.21. *A word in S^+ is smooth if and only if all of its cuts are smooth.*

Proof Clearly if a word is smooth, then all of its cuts must be smooth. We prove the converse implication by induction on the length of the word. Words of length one or two are vacuously smooth. For the induction step, consider a

word $w \in S^+$ with all cuts being smooth. Since all cuts are smooth, all letters are in the same infix class. We will show that w is also in this infix class. Decompose the word as $w = vab$ where $a, b \in S$ are the last two letters. By induction assumption, va is smooth. Since the last cut is smooth, a and ab are in the same infix class, and therefore they are in the same prefix class by the Eggbox Lemma. This means that there is some x such that $abx = a$. We have

$$va = vabx = wx$$

which establishes that w is in the same infix class as va, and therefore in the same infix class as all the letters in w. □

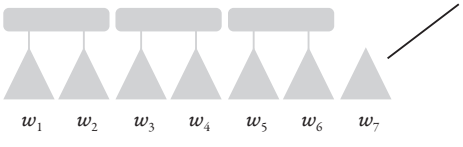
Take a word $w \in S^+$, and cut it along all cuts which are not smooth, yielding a factorisation

$$w = w_1 \cdots w_n.$$

By Claim 1.21, all of the words $w_1, \ldots, w_n$ are smooth, and therefore Lemma 1.18 can be applied to construct corresponding Simon trees of height strictly smaller than

$$4 \cdot (\text{maximal size of an infix class in } S - T).$$

Using binary nodes, group these trees into pairs, as in the following picture:



Each pair corresponds to a word with a non-smooth cut, and therefore each pair has product in T. Therefore, we can combine the paired trees into a single tree, using the induction assumption on a smaller semigroup. The resulting height is the height from the induction assumption on T, plus at most

$$1 + 4 \cdot (\text{maximal size of an infix class in } S - T) < 5|S - T|,$$

thus proving the induction step.

Exercises

Exercise 15. Show that for every semigroup homomorphism

$$h : \Sigma^+ \to S \qquad \text{with } S \text{ finite}$$

there is some $k \in \{1, 2, \ldots\}$ such that for every $n \in \{3, 4, \ldots\}$, every word of length bigger than n^k can be decomposed as

$$w_0 w_1 \cdots w_n w_{n+1}$$

such that all of the words $w_1, \ldots, w_n$ are mapped by h to the same idempotent.

Exercise 16. Show optimality for the previous exercise, in the following sense. Show that for every $k \in \{1, 2, \ldots\}$ there is some semigroup homomorphism

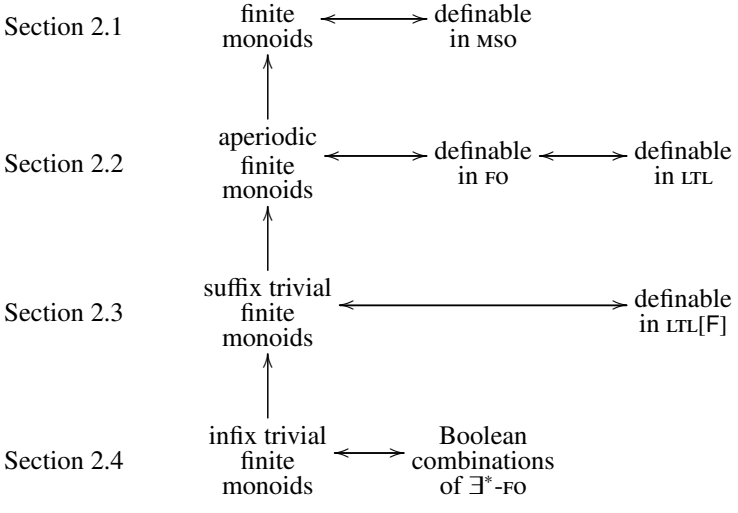
$$h : \Sigma^+ \to S \qquad \text{with } S \text{ finite}$$

such that for every $n \in \{1, 2, \ldots\}$ there is a word of length at least n^k which does not admit a factorisation $w_0 \cdots w_{n+1}$ where all of $w_1, \ldots, w_n$ are mapped by h to the same idempotent.

2

Logics on finite words, and the corresponding monoids

In this chapter, we show how structural properties of a monoid correspond to the logical power that is needed to define languages recognised by this monoid. There are two types of logic: monadic second-order logic MSO and its fragments, as well as linear temporal logic LTL and its fragments. Here is a map of the results, with horizontal arrows being equivalences, and the vertical arrows being strict inclusions.



2.1 All monoids and monadic second-order logic

We begin with a logic that captures the class of all regular languages, namely monadic second-order logic (MSO). This correspondence was proved independently by Büchi, Elgot and Trakhtenbrot.

Logic on words. We assume that the reader is familiar with the basic notions of logic. The following descriptions are meant to fix notation. We use the name *vocabulary* for a set of relation names, each one with associated arity in $\{1, 2, \ldots\}$. A *model* over a vocabulary consists of an underlying set (called the *universe* of the model), together with an interpretation of the vocabulary, which maps each relation name to a relation over the universe of corresponding arity. We allow the universe to be empty. For example, a directed graph is the same thing as a model over the vocabulary which contains one binary relation $E(x, y)$.

To express properties of models, we use first-order logic and MSO. Formulas of first-order logic over a given vocabulary are constructed as follows:

$$\underbrace{\forall x \quad \exists x}_{\substack{\text{quantification over} \\ \text{elements of the universe}}} \qquad \underbrace{\varphi \wedge \psi \quad \varphi \vee \psi \quad \neg\varphi}_{\text{Boolean operations}} \qquad \underbrace{R(x_1, \ldots, x_n)}_{\substack{\text{an } n\text{-ary relation name from} \\ \text{the vocabulary applied} \\ \text{to a tuple of variables}}} \qquad \underbrace{x = y}_{\text{equality}}$$

For the semantics of first-order formulas, defined in the usual way, we use the following notation

$$\underbrace{\mathbb{A}, a_1, \ldots, a_n \models \varphi(x_1, \ldots, x_n)}_{\substack{\varphi \text{ is true in model } \mathbb{A}, \text{ assuming that the} \\ \text{free variables } x_1, \ldots, x_n \text{ are mapped to,} \\ \text{respectively, elements } a_1, \ldots, a_n \in \mathbb{A}}}.$$

A *sentence* is a formula without free variables.

The logic MSO extends first-order logic by allowing two kinds of variables: lower case variables $x, y, z, \ldots$ describe elements of the universe as in first-order logic, while upper case variables $X, Y, Z, \ldots$ describe subsets of the universe (in other words, monadic relations on the universe). Apart from the syntactic constructions of first-order logic, MSO also allows:

$$\underbrace{\forall X \quad \exists Y}_{\substack{\text{quantification over} \\ \text{subsets of the universe}}} \qquad \underbrace{x \in X}_{\text{membership}}$$

We do not use more powerful logics (e.g. full second-order logic can also quantify over binary relations, ternary relations, etc.).

If we describe a word using a model, then we can use logic to express properties of that word. Therefore, a formula of logic can be seen as defining a word language.

Definition 2.1 (Languages definable in first-order logic and MSO)**.** For a word $w \in \Sigma^*$, define its *ordered model* as follows. The universe is the positions in the word. The vocabulary is

$$\underbrace{x \leq y}_{\text{arity 2}} \qquad \{\underbrace{a(x)}_{\text{arity 1}}\}_{a \in \Sigma},$$

with $x \leq y$ representing the order on positions, and with $a(x)$ representing the positions with label a. For a sentence φ of MSO over this vocabulary, we define its *language* to be

$$\{w \in \Sigma^+ : \text{the ordered model of } w \text{ satisfies } \varphi\}.$$

A language is called MSO *definable* if it is of this form. If φ is in first-order logic, then the language is called *first-order definable*.

Example 3. The language $a^*bc^* \subseteq \{a, b, c\}^*$ is first-order definable:

$$\underbrace{\exists x}_{\substack{\text{there is a}\\ \text{position}}} \ \underbrace{b(x)}_{\substack{\text{which has}\\ \text{label } b}} \wedge \underbrace{\forall y \ \overbrace{y < x}^{x \leq y \wedge x \neq y} \Rightarrow a(y)}_{\substack{\text{and every earlier position}\\ \text{has label } a}} \wedge \underbrace{\forall y \ y > x \Rightarrow c(y)}_{\substack{\text{and every later position}\\ \text{has label } b.}}$$

□

Example 4. Here is an MSO formula which defines the language $(aa)^*a \subseteq a^*$ of words of odd length:

$$\underbrace{\exists X}_{\substack{\text{there is a}\\ \text{set of}\\ \text{positions}}} \ \underbrace{\forall x \ \overbrace{\text{first}(x)}^{\forall y \ y \geq x} \vee \overbrace{\text{last}(x)}^{\forall y \ y \leq x} \Rightarrow x \in X}_{\text{which contains the first and last positions,}} \wedge \underbrace{\forall x \forall y \ \overbrace{x = y + 1}^{\substack{x < y \wedge \\ \forall z \ z \leq x \vee y \leq z}} \Rightarrow (x \in X \Leftrightarrow y \notin X)}_{\text{and contains every second position.}}$$

As we will see in Section 2.2, this language is not first-order definable. □

One could imagine other ways of describing a word via a model, e.g. a *successor model* where $x \leq y$ is replaced by a successor relation $x + 1 = y$. The successor relation can be defined in first-order logic in terms of order, but the converse is not true, and there are languages that are first-order definable in the order model but not in the successor model. For the logic MSO, there is no

difference between successor and order, since the order can be defined in MSO as follows

$$x \leq y \quad \text{iff} \quad \forall X \underbrace{(x \in X \wedge (\forall y \forall z\, y \in X \wedge y + 1 = z \Rightarrow z \in X))}_{X \text{ contains } x \text{ and is closed under successors}} \Rightarrow y \in X.$$

The Trakhtenbrot-Büchi-Elgot Theorem. The logic MSO describes exactly the regular languages.

Theorem 2.2 (Trakhtenbrot-Elgot-Büchi)**.** *A language $L \subseteq \Sigma^*$ is* MSO *definable if and only if it is regular*[1].

This result is seminal for two reasons.

The first reason is that it motivates the search for other correspondences

$$\text{machine model} \quad \sim \quad \text{logic},$$

which can concern either generalisations or restrictions of the regular languages. In the case of generalisations, an important example is Fagin's Theorem, which says that NP describes exactly the languages definable in existential second-order logic [2] . In the case of restrictions, important examples are first-order logic and its fragments, which will be described later in this chapter.

The second reason is that the Trakhtenbrot-Büchi-Elgot theorem generalises well to structures beyond finite words. For example, there are natural notions of MSO definable languages for: infinite words, finite trees, infinite trees, graphs, etc. It therefore makes sense to search for notions of regularity – e.g. based on generalisations of semigroups – which have the same expressive power as MSO. This line of research will also be followed in this book.

The rest of Section 2.1 proves the Trakhtenbrot-Büchi-Elgot Theorem.

The easy part is that every regular language is MSO definable. Using the same idea as for the parity language in Example 4, the existence of a run of nondeterministic finite automaton can be formalised in MSO. If the automaton has n states, then the formula looks like this:

$$\underbrace{\exists X_1\, \exists X_2\, \cdots \exists X_n}_{\text{existential set quantification}} \qquad \underbrace{\text{“the sets } X_1, \ldots, X_n \text{ describe an accepting run”}}_{\text{first-order formula}}$$

[1] This result was proved, independently, in the following papers:
[13] Trakhtenbrot, “The synthesis of logical nets whose operators are described in terms of one-place predicate calculus (Russian)”, 1958 , Theorems 1 and 2
[1] Büchi, “Weak second-order arithmetic and finite automata”, 1960 , Theorems 1 and 2
[3] Elgot, “Decision problems of finite automata design and related arithmetics”, 1961 , Theorem 5.3

[2] [4] Fagin, “Generalized first-order spectra and polynomial-time recognizable sets”, 1974 , Theorem 6

A corollary is that if we take any MSO definable language, turn it into an automaton using the hard implication, and come back to MSO using the easy implication, then we get an MSO sentence of the form described above.

The easy part will become the hard part in some generalisations – e.g. for some kinds of infinite words or for graphs – because these generalisations lack a suitable automaton model.

We now turn to the hard part, which says that every MSO definable language is regular. This implication is proved in the rest of Section 2.1. For the definition of regularity, we use finite monoids. In other words, we will show that every MSO definable language is recognised by a finite monoid. The monoid will consist of "MSO types" of some fixed quantifier rank, as described later in this section.

To avoid talking about the two kinds of variables in MSO, instead of working with MSO over ordered models, we will work with first-order logic over models with second-order information, as defined below.

Definition 2.3. Define the *set model* of a word $w \in \Sigma^*$ as follows. The universe is sets of positions, and it is equipped with the following relations:

$$x \subseteq y \qquad \underbrace{x < y}_{\substack{\text{every position in } x \\ \text{is to the left of} \\ \text{every position in } y}} \qquad \underbrace{x = \emptyset}_{x \text{ is empty}} \qquad \underbrace{x \subseteq a}_{\substack{\text{all positions in } x \\ \text{have label } a}}$$

Lemma 2.4. *A language $L \subseteq \Sigma^*$ is* MSO *definable in the ordered model if and only if it is first-order definable in the set model.*

Proof Set quantification in the ordered model corresponds to first-order quantification in the set model. To recover first-order quantification from the ordered model, we use singleton sets in the set model; these can be defined using the inclusion relation. □

Because of the above lemma, instead of MSO over the ordered model, we will work in first-order logic over the set model. Recall that the *quantifier rank* of a first-order formula is the maximal number of quantifiers that can be found on some branch of the syntax tree in the formula. Here is an example:

$$\underbrace{\forall x \, (x \neq \emptyset \wedge x \subseteq a \;\Rightarrow\; \overbrace{(\exists y \; y < x \wedge y \neq \emptyset)}^{\text{quantifier rank 1}} \wedge \overbrace{(\exists y \; x < y \wedge y \neq \emptyset)}^{\text{quantifier rank 1}})}_{\text{quantifier rank 2}}$$

An important property of this size measure is that formulas of quantifier rank at most $k \in \{0, 1, \ldots\}$ are closed under Boolean combinations. For words $w, w' \in$

Σ^*, we write

$$w \equiv_k w'$$

if the corresponding set models satisfy the same first-order sentences of quantifier rank at most k.

Lemma 2.5. *Let Σ be a finite alphabet and $k \in \{0, 1, \ldots\}$. Then $\equiv_k$ is an equivalence relation on Σ^* with the following properties:*

Finite index. *It has finitely many equivalence classes.*

Saturation. *If $L \subseteq \Sigma^*$ is defined by a first-order sentence of quantifier rank k, over the set model, then*

$$w \equiv_k w' \qquad \text{implies} \qquad w \in L \Leftrightarrow w' \in L.$$

Congruence. *It is a congruence with respect to concatenation, i.e.*

$$\bigwedge_{i \in \{1,2\}} w_i \equiv_k w'_i \qquad \text{implies} \qquad w_1 w_2 \equiv_k w'_1 w'_2.$$

Before we prove the lemma, we use it to finish the proof of the Trakhtenbrot-Büchi-Elgot Theorem. Suppose that $L \subseteq \Sigma^*$ is definable in MSO. By Lemma 2.4, L is first-order definable using set models; let $k \in \{0, 1, \ldots\}$ be the quantifier rank of the corresponding first-order sentence. Consider the function

$$w \in \Sigma^* \qquad \mapsto \qquad \underbrace{\text{equivalence class of } w \text{ under } \equiv_k}_{\text{we call this the rank } k \text{ MSO type of } w}$$

By the congruence property from Lemma 2.5, the function is compositional. Therefore, thanks to Lemma 1.3, the image can be equipped with a monoid product so that the function becomes a monoid homomorphism. By the finite index property, the monoid is finite. By the saturation property, the monoid homomorphism recognises the language. We have thus established that L is recognised by a finite monoid, and therefore it is regular.

It remains to prove the lemma.

Proof of Lemma 2.5. The saturation property is an immediate consequence of the definition.

An easy to prove property of quantifier rank is that, once the vocabulary and the free variables are fixed, then up to logical equivalence there are finitely many formulas of quantifier rank at most k. This establishes the finite index property.

We are left with the congruence property. To prove congruence, we use

Ehrenfeucht-Fraïssé games; we assume that the reader is familiar with these[3]. Let $w_1, w_2, w_1', w_2' \in \Sigma^*$. The main idea behind the congruence property is that a set of positions in a concatenation of two words is the same thing as a pair (set of positions in the first word, set of positions in the second word)[4]. Define

$$(\text{set model of } w_1w_2) \xrightarrow{f} (\text{set model of } w_1) \times (\text{set model of } w_2)$$

to be the corresponding bijection; likewise define f' for w_1' and w_2'. Using these bijections, we will combine Duplicator's winning strategies for the Ehrenfeucht-Fraïssé games corresponding to the assumption

$$\underbrace{w_1 \equiv_k w_1' \qquad w_2 \equiv_k w_2'}_{\text{small games}},$$

to get a winning strategy for Duplicator in the Ehrenfeucht-Fraïssé game corresponding to the conclusion

$$\underbrace{w_1w_2 \equiv_k w_1'w_2'}_{\text{big game}}.$$

This is done as follows. Whenever Spoiler makes a move in the big game, then Duplicator uses the bijections to transform this move into a pair of Spoiler moves in the small games. Using Duplicator's winning strategies in the small games, Duplicator gets a pair of responses, and combines them using the bijection into a response in big game.

We now argue that Duplicator's strategy in the big game, as described above, is winning. This argument will depend on the set properties (inclusion, emptiness, etc.) that are given as relations in the set model. Suppose that all k rounds have been played, resulting in tuples

$$\underbrace{a_1, \ldots, a_k}_{\text{sets of positions in } w_1w_2} \quad \text{and} \quad \underbrace{a_1', \ldots, a_k'}_{\text{sets of positions in } w_1'w_2'}.$$

We need to show that the tuples satisfy the same quantifier-free formulas in the corresponding set models. Since Duplicator's strategy in the big game was obtained by combining winnings strategies for Duplicator in the small games,

[3] For an introduction to Ehrenfeucht-Fraïssé games, see
[6] Hodges, *Model Theory*, 1993 , Section 3.2

[4] This explains why monadic second-order logic, and not general second-order logic is used. In general second-order logic, one can use relations of higher arities, e.g. binary relations. A binary relation over positions in w_1w_2 is not the same thing as two binary relations over positions, in w_1 and w_2, respectively.

we know that for every $i \in \{1, 2\}$, the tuples

$$\underbrace{a_{1,i}, \ldots, a_{k,i}}_{\text{sets of positions in } w_i} \qquad \text{and} \qquad \underbrace{a'_{1,i}, \ldots, a'_{k,i}}_{\text{sets of positions in } w'_i},$$

obtained by taking the i-th coordinates of the bijections f and f', satisfy the same quantifier-free formulas. We only prove

$$\underbrace{a_i < a_j}_{\text{in } w_1 w_2} \quad \text{iff} \quad \underbrace{a'_i < a'_j}_{\text{in } w'_1 w'_2} \qquad \text{for every } i, j \in \{1, \ldots, k\}, \tag{2.1}$$

the other relations are left as an exercise. The main observation is that the relation $<$ in $w_1 w_2$ reduces to a quantifier free properties of the corresponding elements in w_1 and w_2, as explained below:

$$\underbrace{a_i < a_j}_{\text{in } w_1 w_2} \quad \text{iff} \quad (\underbrace{a_{i,1} < a_{j,1}}_{\text{in } w_1} \text{ and } \underbrace{a_{j,2} = \emptyset}_{\text{in } w_2}) \text{ or } (\underbrace{a_{j,1} = \emptyset}_{\text{in } w_1} \text{ and } \underbrace{a_{i,2} < a_{j,2}}_{\text{in } w_2}).$$

By the assumption from the small games, the truth value of the right side of the equivalence does not change when we go to w'_1 and w'_2, and therefore the same is true for the left side of the equivalence[5]. □

Exercises

Exercise 17. Define $\mathcal{U}_2$ to be the monoid with elements $\{a, b, 1\}$ and product

$$xy = \begin{cases} y & \text{if } x = 1 \\ x & \text{otherwise.} \end{cases}$$

Show that every finite monoid can be obtained from $\mathcal{U}_2$ by applying Cartesian products, quotients (under semigroup congruences), sub-semigroups, and the powerset construction from Exercise 4.

Exercise 18. For an alphabet Σ, consider the model where the universe is Σ^*, and which is equipped with the following relations:

$$\underbrace{x \text{ is a prefix of } y}_{\text{binary relation}} \qquad \underbrace{\text{the last letter of } x \text{ is } a \in \Sigma}_{\text{one unary relation for each } a \in \Sigma}$$

[5] As mentioned previously, this part of the argument s depends on the choice of relations in the definition of the set model, since the relation $x = \emptyset$ is used when talking about $x < y$. In fact, if we removed the relation $x = \emptyset$ from the vocabulary, then the congruence property in the lemma would fail, even though $x = \emptyset$ is first-order definable in terms of $x \subseteq y$.

Show that a language $L \subseteq \Sigma^*$ is regular if and only if there is a first-order formula $\varphi(x)$ over the above vocabulary such that L is exactly the words that satisfy $\varphi(x)$ in the above structure.

Exercise 19. What happens if the prefix relation in Exercise 18 is replaced by the infix relation?

2.2 Aperiodic semigroups and first-order logic

We now turn to showing how fragments of MSO logic correspond to fragments of regular languages, as described by restrictions on finite monoids. The first – and arguably most important – fragment is first-order logic. As we will see from the Schützenberger-McNaughton-Papert-Kamp Theorem, a language is first-order definable if and only if it is recognised by a finite monoid M which satisfies

$$\underbrace{a^! = a^{!+1} \quad \text{for all } a \in M}_{\text{a monoid or semigroup which satisfies this is called } \textit{aperiodic}}$$

In other words, in an aperiodic monoid the sequence $a, a^2, a^3, \ldots$ is eventually constant, as opposed to having some non-trivial periodic behaviour.

Example 5. Consider the parity language $(aa)^* \subseteq a^*$. We claim that this language is not recognised by any aperiodic monoid, and therefore it is not first-order definable. Suppose that the parity language is recognised by some finite monoid M. By Theorem 1.6, the syntactic monoid of the language – which is not aperiodic because it is the two-element group – must be a quotient of a sub-monoid of M. Since aperiodic monoids are closed under taking quotients and sub-monoids, it follows that M cannot be aperiodic.

The above argument shows that a regular language is first-order definable if and only if its syntactic monoid is aperiodic. Since the syntactic monoid can be computed, and aperiodicity is clearly decidable, it follows that there is an algorithm which decides if a regular language is first-order definable. □

Apart from first-order logic and aperiodic monoids, the Schützenberger-McNaughton-Papert-Kamp Theorem considers also linear temporal logic and star-free regular expressions, so we begin by defining those.

Linear temporal logic Linear temporal logic (LTL) is an alternative to first-order logic which does not use quantifiers. The logic LTL only makes sense for structures equipped with a linear order; hence the name.

Definition 2.6 (Linear temporal logic)**.** Let Σ be a finite alphabet. Formulas of *linear temporal logic* (LTL) over Σ are defined by the following grammar:

$$a \in \Sigma \qquad \varphi \wedge \psi \qquad \varphi \vee \psi \qquad \neg\varphi \qquad \varphi \mathsf{U} \psi.$$

The semantics for LTL formulas[6] is a ternary relation, denoted by

$$\underbrace{w,}_{\substack{\text{word}\\ \text{in } \Sigma^+}} \underbrace{x}_{\substack{\text{position}\\ \text{in } w}} \models \underbrace{\varphi}_{\text{LTL formula}},$$

which is defined as follows. A formula $a \in \Sigma$ is true in positions with label a. The semantics for Boolean combination are defined as usual. For formulas of the form $\varphi \mathsf{U} \psi$, which are called *until formulas*, the semantics[7] are

$$w, x \models \varphi \mathsf{U} \psi \quad \stackrel{\text{def}}{=} \quad \underbrace{\exists y\ x < y}_{\substack{\text{there is some}\\ \text{position strictly}\\ \text{after } x}} \wedge \underbrace{w, y \models \psi}_{\substack{\text{which}\\ \text{satisfies } \psi}} \wedge \underbrace{\forall z\ x < z < y \ \Rightarrow\ w, z \models \varphi}_{\substack{\text{and such that all intermediate}\\ \text{positions satisfy } \varphi}}$$

We say that an LTL formula is true in a word, without specifying a position, if the formula is true in the first position of that word. A language $L \subseteq \Sigma^*$ is called LTL *definable* if there is an LTL formula φ that defines the language on nonempty words:

$$w \in L \quad \text{iff} \quad w \models \varphi \qquad \text{for every } w \in \Sigma^+.$$

.

Example 6. To get a better feeling for LTL, we discuss some extra operators that can be defined using until. We write $\bot$ for any vacuously false formula, e.g. $a \wedge \neg a$, likewise $\top$ denotes any vacuously true formula. We can some commonly used extra operators:

$$\underbrace{\mathsf{X}\varphi \stackrel{\text{def}}{=} \bot \mathsf{U} \varphi}_{\text{the next position satisfies } \varphi} \qquad \underbrace{\mathsf{F}\varphi \stackrel{\text{def}}{=} \top \mathsf{U} \varphi}_{\substack{\text{some strictly later position}\\ \text{satisfies } \varphi\text{s}}} \qquad \underbrace{\varphi \mathsf{U}^* \psi \stackrel{\text{def}}{=} \psi \vee (\varphi \mathsf{U} \psi)}_{\text{non-strict until}}$$

Similarly, we can define a non-strict version F^* of the operator F. For example, the formula

$$\mathsf{F}^*(a \wedge \underbrace{\neg \mathsf{F} \top}_{\text{last position}})$$

[6] The semantics make sense for any linear order, possibly infinite, with positions coloured by Σ.
[7] We use a variant of the until operator which is sometimes called *strict until*.

says that the last position in the word has label a. The formula

$$a \cup b$$

defines the language $\Sigma a^* b \Sigma^*$. □

Almost by definition, every LTL definable language is also first-order definable. This is because to every LTL formula one can associate a first-order formula $\varphi(x)$ that is true in the same positions.

Star-free languages. Apart from first-order logic, aperiodic monoids, and LTL, another equivalent formalism is going to be star-free expressions. As the name implies, star-free expressions cannot use Kleene star, but in exchange they are allowed to use complementation (without star and complementation one could only define finite languages). For an alphabet Σ, the star-free expressions are those that can be defined using the following operations on languages:

$$\underbrace{a \in \Sigma}_{\substack{\text{the language that}\\ \text{contains only}\\ \text{the word } a}} \quad \underbrace{\emptyset}_{\substack{\text{empty}\\ \text{language}}} \quad \underbrace{L \cdot K}_{\text{concatenation}} \quad \underbrace{L + K}_{\text{union}} \quad \underbrace{\overline{L}.}_{\substack{\text{complementation}\\ \text{with respect}\\ \text{to } \Sigma^*}}$$

Note that the alphabet needs to be specified to give meaning to the complementation operation. A language is called *star-free* if it can be defined by a star-free expression.

Example 7. Assume that the alphabet is $\{a, b\}$. The expression $\bar{\emptyset}$ describes the full language $\{a, b\}^*$. Therefore

$$\bar{\emptyset} \cdot a \cdot \bar{\emptyset}$$

describes all words with at least one a. Taking the complement of the above expression, we get a star-free expression for the language b^*. □

Like for LTL formulas, almost by definition every star-free expression describes a first-order definable language. This is because to every star-free expression one can associate a first-order formula $\varphi(x, y)$ which selects a pair of positions $x \leq y$ if and only if the corresponding infix belongs to the language described by the expression.

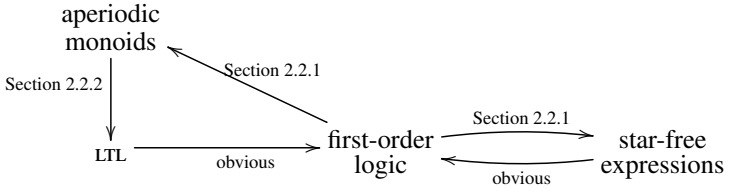
Equivalence of the models. We now show that all of the models discussed so far in this section are equivalent.

Theorem 2.7 (Schützenberger-McNaughton-Papert-Kamp)**.** *The following are equivalent*[8] *for every $L \subseteq \Sigma^*$:*

[8] This theorem combines three equivalences.

(1) *recognised by a finite aperiodic monoid;*
(2) *star-free;*
(3) *first-order definable;*
(4) LTL *definable.*

The rest of Section 2.2 is devoted to proving the above theorem, according to the following plan:



2.2.1 From first-order logic to aperiodic monoids and star-free expressions

In this section, we prove two inclusions: first-order logic is contained in both aperiodic monoids and star-free expressions.

The proof uses a similar compositionality analysis of Ehrenfeucht-Fraïssé games, as in the MSO-to-regular implication of the Trakhtenbrot-Büchi-Elgot Theorem. For $k \in \{0, 1, 2, \ldots\}$ and $w, w' \in \Sigma^*$, we write

$$w \equiv_k w'$$

if the corresponding ordered structures satisfy the same first-order formulas of quantifier rank at most k. We use the blue colour to distinguish the equivalence from the one for MSO that was used in Section 2.1. By the same argument as in Section 2.1, one shows that

$$w \in \Sigma^* \qquad \mapsto \qquad \underbrace{\text{equivalence class of } w \text{ under } \equiv_k}_{\text{we call this the rank } k \text{ FO type of } w}$$

is a monoid homomorphism, into a finite monoid, which recognises every language that can defined by a first-order sentence of quantifier rank k.

We will now show that every equivalence class of $\equiv_k$ is defined by a star-free expression, and that the monoid of these equivalence classes is aperiodic.

The equivalence aperiodic monoids and star-free expressions was shown in
[10] Schützenberger, "On finite monoids having only trivial subgroups", 1965 , p. 190
The equivalence of star-free expressions and first-order logic was shown in
[9] McNaughton and Papert, *Counter-free automata*, 1971 , Theorem 10.5
The equivalence of first-order logic and LTL, not just for finite words, was shown in
[7] Kamp, "Tense Logic and the Theory of Linear Order", 1968

In both cases, we use the following lemma, which characterises equivalence classes of $\equiv_{k+1}$ in terms of equivalence classes of $\equiv_k$ by using only Boolean combinations and concatenation.

Lemma 2.8. *Let $w, w' \in \Sigma^*$. Then $w \equiv_{k+1} w'$ if and only if*

$$w \in LaK \Leftrightarrow w' \in LaK$$

holds for every $a \in \Sigma$ and every $L, K \subseteq \Sigma^*$ which are equivalence classes of $\equiv_k$.

Proof A simple Ehrenfeucht-Fraïssé argument. The letter a corresponds to the first move of player Spoiler. □

We now use the lemma to prove the inclusion of first-order logic in star-free expressions and aperiodic monoids.

From first-order logic to star-free. It is enough to show that every equivalence class of $\equiv k$ is star-free. This is proved by induction on k. For the induction base, there is only one equivalence class, namely all words, which is clearly star-free. Consider now the induction step. Consider an equivalence class of $\equiv_{k+1}$, and let be X the set of triples

$$\underbrace{L}_{\substack{\text{equivalence}\\ \text{class of } \equiv_k}} \quad \underbrace{a}_{\text{letter in } \Sigma} \quad \underbrace{K}_{\substack{\text{equivalence}\\ \text{class of } \equiv_k}}$$

such that some word (equivalently, by the above lemma, every word) in the equivalence class belongs to LaK. The equivalence class is equal to the Boolean combination of concatenations

$$\bigcap_{(L,a,K)\in X} LaK \quad \cap \quad \overline{\bigcap_{(L,a,K)\notin X} LaK}.$$

This is a star-free expression, if we assume that L and K are described by star-free expressions from the induction assumption. Since every first-order definable language is a finite union of equivalence classes of $\equiv_k$ for some k, the result follows.

From first-order logic to aperiodic monoids. As we have argued before, every first-order definable language is recognised by the finite monoid of equivalence classes of $\equiv_k$, for some k. It remains to show that this monoid is aperiodic. To prove this, we will show that

$$w^{3^k} \equiv_k w^{3^k+1} \qquad \text{for every } w \in \Sigma^*.$$

This is a simple application of Lemma 2.8. For every partition of one of the two words above into LaK, as in the lemma, there is an equivalent partition for the other word.

2.2.2 From aperiodic monoids to LTL

The last, and most important, step in the proof is constructing an LTL formula based on an aperiodic monoid[9]. In this part of the proof, semigroups will be more convenient than monoids. Also, we will use LTL to define colourings, which are like languages but with possibly more than two values: a function from Σ^+ to a finite set of colours is called LTL definable if for every colour, the words sent that colour are an LTL definable language. For example, a semigroup homomorphism into a finite semigroup is a colouring.

Lemma 2.9. *Let S be a semigroup, and let $\Sigma \subseteq S$. The colouring*

$$w \in \Sigma^+ \qquad \mapsto \qquad \textit{product of } w$$

LTL *definable.*

By applying the lemma to the special case of S being a monoid, and substituting each monoid element for the letters that get mapped to it in the recognising homomorphism, we immediately get the implication from finite aperiodic monoids to LTL.

It remains to prove the lemma. The proof is by induction on two parameters: the size of the semigroup S, and the size of Σ. These parameters are ordered lexicographically, with the size of S being more important. Without loss of generality, we assume that Σ generates S, i.e. every element of S is the product of some word in Σ^+.

The induction base is treated in the following claim.

Claim 2.10. *If either S or Σ has size one, then Lemma 2.9 holds.*

Proof If the semigroup has one element, there is nothing to do, since functions with one possible value are clearly LTL definable. Consider the case when the Σ contains only one element $a \in S$. By aperiodicity, the sequence

$$a, a^2, a^3, \ldots$$

is eventually constant, which means that all words longer than some threshold have the same value $a^!$. The product is therefore easily seen to be an LTL definable colouring. □

We are left with the induction step. For $c \in S$, consider the function

$$a \in S \mapsto ca \in S.$$

[9] The proof in this section is based on
[14] Wilke, "Classifying Discrete Temporal Properties", 1999 , Section 2

Claim 2.11. *If $a \mapsto ca$ is a permutation of S, then it is the identity.*

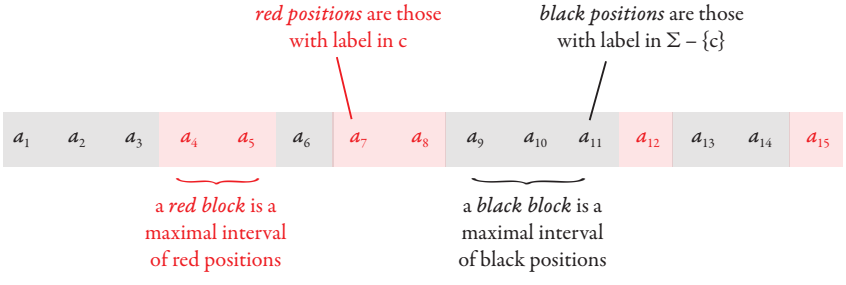
Proof Suppose that $a \mapsto ca$ is a permutation. If n is large enough, then by aperiodicity we know that $a \mapsto c^n a$ and $a \mapsto c^{n+1} a$ are the same permutation, and therefore $a \mapsto ca$ must be the identity. □

If the function $a \mapsto ca$ is the identity for every $c \in \Sigma$, then the product of a word is the same as its last letter; and such a colouring is clearly LTL definable. We are left with the case when there is some $c \in \Sigma$ such that $a \mapsto ca$ is not the identity. Fix this c for the rest of the proof. Define T to be the image of the function $a \mapsto ca$, this is a proper subset of S by assumption on c.

Claim 2.12. *T is a sub-semigroup of S.*

Proof It two semigroup elements have prefix c, then the same is true for their product. □

In the rest of the proof, we use the following terminology for a word $w \in \Sigma^+$:



We first describe the proof strategy. For each black block, its product can be computed in LTL using the induction assumption on a smaller set of generators. The same is true for black blocks. Define a *red-black block* to be any union of a red block plus the following (non-empty) black block; as illustrated below:

For every red-black block, its product is in T because it begins with c and has at least two letters. Furthermore, the product can be computed in LTL, by using the products of the red and black blocks inside it. Using the induction

assumption on a smaller semigroup, we compute the product of the union of all red-black blocks. Finally, the product of the entire word is obtained by taking into account the blocks that are not part of any red-black block.

The rest of this section is devoted to formalising the above proof sketch. In the formalisation, it will be convenient to reason with word-to-word functions. We say that a function a function of type $\Sigma^* \to \Gamma^*$ is an called LTL transduction if it has the form

$$a_1 \cdots a_n \in \Sigma^* \qquad \mapsto \qquad f(a_1 \cdots a_n) \cdot f(a_2 \cdots a_n) \cdots f(a_{n-1})$$

for some LTL definable colouring $f : \Sigma^+ \to X$, where $X \subseteq \Gamma^*$ is finite. By substituting formulas, one easily shows the following composition properties:

$$(\text{LTL transductions}) \circ (\text{LTL transductions}) \subseteq \text{LTL transductions},$$
$$(\text{LTL colourings}) \circ (\text{LTL transductions}) \subseteq \text{LTL colourings}.$$

We use LTL transductions to decorate an input word $w \in \Sigma^+$ with extra information that will serve towards computing its product.

(1) For position that precedes a block (i.e. the next position begins a new block), write the value of the next block. For the remaining positions, do not write anything. Use two disjoint copies of S to distinguish the values of the red and black blocks. Here is a picture:

In the above picture, $a_{i,j}$ denotes the product of the infix $\{i, \ldots, j\}$. The function described in this step is an LTL transduction, thanks to the induction assumption on smaller alphabets.

(2) Take the output of the function in the previous step, and for each red letter (a product of a red block), multiply it with the next letter (which is a product of a black block). As a result, we get the values of all red-black blocks which do not begin in the first position. Here is a picture:

The function in this step is clearly an LTL transduction.

By induction assumption on a smaller semigroup, the product operation $T^+ \to T$ is an LTL colouring. By composing the functions described above with the semigroup product in T, we see that

$$w \in \Sigma^+ \quad \mapsto \quad \text{value of the union of red-black blocks}$$

is an LTL colouring. The values of the (at most two) blocks that do not participate in above union can also be computed using LTL colourings, and therefore the product of the entire word can be computed.

Exercises

Exercise 20. Show that for every sentence of first-order logic, there is a sentence that is equivalent on finite words, and which uses at most three variables (but these variables can be repeatedly quantified).

Exercise 21. Show that the following are equivalent for a finite semigroup:

(1) aperiodic;
(2) $\mathcal{H}$-trivial, which means that all $\mathcal{H}$-classes are singletons;
(3) no sub-semigroup is a non-trivial group.

Exercise 22. Consider the successor model of a word $w \in \Sigma^*$, which is defined like the ordered model, except that instead of $x < y$ we have $x + 1 = y$. Given an example of a regular language that is first-order definable using the ordered model, but not using the successor model.

Exercise 23. Show two languages which have the same syntactic monoid, and such that only one of them is first-order definable in the successor model.

Exercise 24. Define the *syntactic semigroup* of a language to be the subset of the syntactic monoid which is the image of the nonempty words under the syntactic homomorphism. The syntactic semigroup may be equal to the syntactic monoid. Show that if a language is first-order definable in the successor model, then the syntactic semigroup satisfies the following equality

$$ea_1fbea_2f = ea_2fbea_1f \qquad \text{for every } \underbrace{e, f,}_{\text{idempotents}}\ a_1, a_2, b.$$

Exercise 25. Show that the equality in Exercise 24 is not sufficient, i.e. there is a language which satisfies the equality but which is not first-order definable in the successor model.

Exercise 26. Consider the following extension of LTL with group operators. Suppose that G is a finite group, and let

$$\{\varphi_g\}_{g \in G},$$

be a family of already defined formulas such that every position in an input word is selected by exactly one formula φ_g. Then we can create a new formula, which is true in a word of length n if

$$1 = g_1 \cdots g_n,$$

where $g_i \in G$ is the unique group element whose corresponding formula selects position i. Show that this logic defines all regular languages.

2.3 Suffix trivial semigroups and linear logic with F only

In the previous section, we showed that first-order logic corresponds to the monoids without groups, which is the same thing as monoids with trivial $\mathcal{H}$-classes (Exercise 21). What about monoids with trivial suffix classes, prefix classes, or infix classes? Trivial infix classes will be described in Section 2.4. The following theorem describes the expressive power of suffix trivial monoids; a symmetric result holds for prefix trivial monoids. In the theorem, we use the fragment of LTL that cannot use U but only the operators

$$\underbrace{\top \mathsf{U} \varphi}_{\mathsf{F}\varphi} \qquad \underbrace{\neg \mathsf{F} \neg \varphi}_{\mathsf{G}\varphi} \qquad \underbrace{\varphi \vee \mathsf{F}\varphi}_{\mathsf{F}^*\varphi} \qquad \underbrace{\neg \mathsf{F}^* \neg \varphi}_{\mathsf{G}^*\varphi}.$$

Since all of the above operators can be defined in terms of F, we write LTL[F] for the resulting logic.

Theorem 2.13. *The following conditions are equivalent for $L \subseteq \Sigma^*$:*

(1) *Recognised by a finite suffix trivial monoid;*
(2) *Defined by a finite union of regular expressions of the form*

$$\underbrace{\Sigma_0^* a_1 \Sigma_1^* a_2 \cdots a_n \Sigma_n^* \qquad \text{where } a_i \in \Sigma - \Sigma_i \text{ for } i \in \{1, \ldots, n\}.}_{\text{we call such an expression suffix unambiguous}}$$

In the above, some of the sets $\Sigma_i \subseteq \Sigma$ might be empty, in which case $\Sigma_i^* = \{\varepsilon\}$.

(3) *Defined by a Boolean combination of* LTL[F] *formulas of the form* $\mathsf{F}^*\varphi$.

To see why the formulas in item (3) need to be guarded by F^*, consider the LTL[F] formula a which defines the language "words beginning with a". This language is not recognised by any finite suffix trivial monoid.

Proof

(1)⇒ (2) We will show that for every finite suffix trivial monoid M, and every $F \subseteq M$, the language

$$\{w \in M^* : F \text{ contains the product of } w\}$$

is defined by a finite union of suffix unambiguous expressions. It will follow that for every monoid homomorphism into L, the recognised language is defined by a similar expression, with monoid elements substituted by the letters that map to them (such a substitution preserves unambiguity).

If it is of course enough to consider the case when F contains only one element, call it $a \in M$. The proof is by induction on the position of a in the suffix ordering.

The induction base is when a is the monoid identity. By Exercise 12, the suffix class of the identity is a group, and a group must be trivial in a suffix trivial monoid. It follows that a word has product a if and only if it belongs to a^*, which is a suffix unambiguous expression.

We now prove the induction step. Consider a word with product a. This word must be nonempty, since otherwise its product would be the identity. Let i be the maximal position in the word such that the suffix starting in i also has product a. By suffix triviality, every position $< i$ is labelled by a letter in

$$\Sigma_0 = \{b \in M : ba = a\}.$$

Let b be the product of the suffix that starts after i, not including i, and let c be the label of position i. By choice of i, b is a proper suffix of a and $a = cb$. Summing up, words with product a are defined by the expression

$$\bigcup_{\substack{b,c \\ b \text{ is a proper prefix of } c \\ a=cb}} \Sigma_0^* c \cdot (\text{words with product } b),$$

Apply the induction assumption to b, yielding a finite union of suffix unambiguous expressions, and distribute the finite union across concatenation. It remains to justify that the resulting expressions are also suffix unambiguous. This is because none of the expressions that define words with product

b can begin with Σ_1^* with $c \in \Sigma_1$, since otherwise we would contradict the assumption that $cb = a \neq b$.

(2) $\Rightarrow$ (3) Since the formulas from item (3) are closed under union, it is enough to show that every suffix unambiguous expression

$$\Sigma_0^* a_1 \Sigma_1^* a_2 \cdots a_n \Sigma_n^*$$

can be defined by a formula as in (3). For $i \in \{0, \ldots, n\}$, define L_i to be the suffix of the above expression that begins with Σ_i^*. By induction on i, starting with n and progressing down to 0, we show that L_i can be defined by a formula φ_i as in item (3). In the induction base, we say that all positions have label in Σ_n:

$$\varphi_n = \bigwedge_{a \notin \Sigma_n} \neg \mathsf{F}^* a.$$

For the induction step, we first define the language $a_i L_i$, using a formula of LTL[F] (which is not in the shape from item (3)):

$$\psi_i = a_i \wedge \mathsf{G} \bigwedge_{j>i} \varphi_j.$$

Because the expression is suffix unambiguous, the formula ψ_i selects at most one position in a given input word; this property will be used below. The language L_{i-1} is then defined by

$$\varphi_{i-1} = \mathsf{F}^* \psi_i \wedge \underbrace{\mathsf{G}^*((\mathsf{F}\psi_i) \Rightarrow \bigwedge_{a \in \Sigma_0} a)}_{\substack{\text{if a position is to the left} \\ \text{of the unique position} \\ \text{satisfying } \psi_i, \text{ then} \\ \text{it has label in } \Sigma_0.}}$$

(3) $\Rightarrow$ (1) The key observation is the following pumping lemma for formulas as in (3).

Claim 2.14. *For every $\varphi \in$ LTL[F] there is some $n \in \{0, 1, \ldots\}$ such that*

$$w(xy)^n u \models \mathsf{F}^* \varphi \quad \textit{iff} \quad wy(xy)^n u \models \mathsf{F}^* \varphi \qquad \textit{for every } w, x, y, u \in \Sigma^*.$$

Proof Induction on φ. If φ is of the form $a \in \Sigma$, then we can choose $n = 1$, since then both words in the equivalence from the claim use the same letters (possibly in a different order). If the formula has shape $\mathsf{F}\varphi$, then there is nothing to do, since $\mathsf{F}^*\mathsf{F}\varphi$ is equivalent to $\mathsf{F}^*\varphi$, and the induction assumption can be used. Suppose now that φ is a Boolean combination of formulas, and

n is the maximal number from the induction assumption when applied to the subformulas of φ. To prove the conclusion of the claim, we will show

$$\underbrace{w(xy)^{n+1}u}_{v} \models \mathsf{F}^*\varphi \quad \text{iff} \quad \underbrace{wy(xy)^{n+1}u}_{v'} \models \mathsf{F}^*\varphi.$$

For the left-to-right implication, we observe that every suffix of v either intersects w – in which case we can use the induction assumption – or it is already a suffix of v'. For the right-to-left implication, we use the same reasoning, with one extra case, namely if φ is true in a suffix of v' which has form $y_2(xy)^{n+1}w$, for some decomposition $y = y_1y_2$. Here we use the induction assumption as follows:

$$\underbrace{\varepsilon}_{\text{new } w} \qquad \underbrace{y_2}_{\text{new } y} \qquad \underbrace{xy_1y_2}_{\text{new } x}$$

□

By unravelling the definition of the syntactic monoid, in terms of two-sided congruences, we infer from the above claim that for every formula φ of LTL[F], the syntactic monoid M of $\mathsf{F}^*\varphi$ satisfies

$$(xy)^! = y(xy)^! \qquad \text{for all } x, y \in M. \tag{2.2}$$

The same is also true for syntactic monoids of Boolean combinations of such formulas. To finish the proof, we observe that property (2.2) is true in a finite monoid if and only if it is suffix trivial. Indeed, if a monoid is suffix trivial, then $(xy)^!$ and $y(xy)^!$ must be in the same suffix class, and hence equal. Conversely, if a, b are in the same suffix class, then there must be some x, y such that $b = xa$ and $a = yb$; it follows that

$$a = y(xy)^!b \overset{(2.2)}{=} (xy)^!b = b.$$

□

2.4 Infix trivial semigroups and piecewise testable languages

In this section, we describe the languages recognised by finite monoids that are infix trivial. For languages recognised by finite infix trivial monoids, a prominent role will be played the Higman ordering.

Definition 2.15 (Higman ordering)**.** For a finite alphabet Σ, define the *Higman*

ordering on Σ^*, denoted by $\hookrightarrow$, to be the least partial order (transitive and reflexive relation) which satisfies

$$wv \hookrightarrow wav \qquad \text{for all } w, v \in \Sigma^* \text{ and } a \in \Sigma.$$

In other words, $w \hookrightarrow v$ means that w can be obtained from v by removing zero or more positions. Another, equivalent definition, is that $v \hookrightarrow w$ holds if the ordered model of v is a sub-model of the ordered model of w. For example,

$$\text{ape} \hookrightarrow \text{example}.$$

We say that a language $L \subseteq \Sigma^*$ is *upward closed* if

$$v \hookrightarrow w \wedge v \in L \Rightarrow w \in L.$$

Symmetrically, we define downward closed languages. The main result about the Higman ordering is that it is a well-quasi order, as explained in the following lemma.

Lemma 2.16 (Higman's Lemma)**.** *For every upward closed $L \subseteq \Sigma^*$ there is a finite subset $U \subseteq L$ such that*

$$L = \underbrace{\{w \in \Sigma^* : v \leq w \text{ for some } v \in U\}}_{\text{we call this the upward closure of } U}$$

Here is a logical corollary of Higman's lemma.

Theorem 2.17. *A language is upward closed if and only if it can be defined in the ordered model by an $\exists^*$-sentence, i.e. a sentence of the form*

$$\underbrace{\exists x_1\ \exists x_2\ \cdots \exists x_n}_{\text{only existential quantifiers}}\quad \underbrace{\varphi(x_1, \ldots, x_n)}_{\text{quantifier-free}}.$$

Proof Clearly every $\exists^*$-sentence defines an upward closed language. Higman's Lemma gives the converse implication, because the upward closure of every finite set is definable by an $\exists^*$-sentence. □

The Higman ordering will also play an important role in the characterisation of languages recognised by monoids that are infix trivial. Before stating the characterisation, we introduce one more definition, namely zigzags[10]. For languages $L, K \subseteq \Sigma^*$, define a *zigzag between L and K* to be a sequence

$$\underbrace{w_1}_{\in L} \hookrightarrow \underbrace{w_2}_{\in K} < \underbrace{w_3}_{\in L} \hookrightarrow \underbrace{w_4}_{\in K} < \underbrace{w_5}_{\in L} \hookrightarrow \underbrace{w_6}_{\in K} < \cdots.$$

[10] Zigzags and the Zigzag Lemma are inspired by
[2] Czerwinski et al., "A Characterization for Decidable Separability by Piecewise Testable Languages", 2017

In other words, this is a sequence that is growing with respect to the Higman ordering, and such that odd-numbered elements are in L and odd-numbered elements are in K. If L and K are disjoint, then the sequence must be strictly growing.

We are now ready for the characterisation of infix trivial monoids.

Theorem 2.18. *The following conditions are equivalent*[11] *for every* $L \subseteq \Sigma^*$*:*

(1) *recognised by a finite monoid that is infix trivial;*
(2) *is a finite Boolean combination of upward closed languages;*
(3) *there is no infinite zigzag between L and its complement.*

We use the name *piecewise testable* for languages as in item (2) of the above theorem. Equivalence of items (2) and (3) is a corollary of the following lemma, when applied to $K = \Sigma^* - L$.

Lemma 2.19 (Zigzag Lemma)**.** *Let* $L, K \subseteq \Sigma^*$*. The following are equivalent:*

(1) *There are zigzags between L and K of every finite length;*
(2) *There is an infinite zigzag between L and K;*
(3) *There is no piecewise testable language* $M \subseteq \Sigma^*$ *such that*

$$\underbrace{L \subseteq M \quad M \cap K = \emptyset}_{\text{we say that } M \text{ separates } L \text{ and } K}.$$

Proof

(1)⇒(2) Assume that there are zigzags between L and K of arbitrary finite lengths. Define a directed acyclic graph G as follows. Vertices are words in L, and there is an edge $w \to v$ if

$$w \hookrightarrow u \hookrightarrow v \qquad \text{for some } u \in K.$$

For a vertex $v \in L$ of this graph, define its *potential*

$$\alpha(v) \in \{0, 1, \ldots, \omega\}$$

to be the least upper bound on the lengths of paths that start in v, this can be either a finite number, or ω if the paths have unbounded length. By assumption on arbitrarily long zigzags, potentials have arbitrarily high values. By definition of the graph, α is monotone with respect to the (opposite of the) Higman ordering, in the following sense:

$$v \hookrightarrow v' \quad \text{implies} \quad \alpha(v) \geq \alpha(v') \qquad \text{for every } v, v' \in L.$$

[11] Equivalence of items (1) and (2) was first proved in [12] Simon, "Piecewise testable events", 1975

By Higman's Lemma, the language L, like any set of words, has finitely many minimal elements with respect to the Higman ordering. By monotonicity, one of these minimal words must therefore have potential ω. For the same reason, if a word has potential ω, then one of its successors (words reachable in one step in the graph) must also have potential ω; this is because there are finitely many successors that are minimal with respect to the Higman ordering. This way, we can construct an infinite path in the graph which only sees potential ω, as in the König Lemma.

(2)⇒(3) Suppose that there is a zigzag between L and K of infinite length. Every upward closed set selects a suffix (possibly empty) of the zigzag. It follows that every finite Boolean combinations must contain, or be disjoint with, two consecutive elements of the zigzag. Therefore, such a Boolean combination cannot separate L from K.

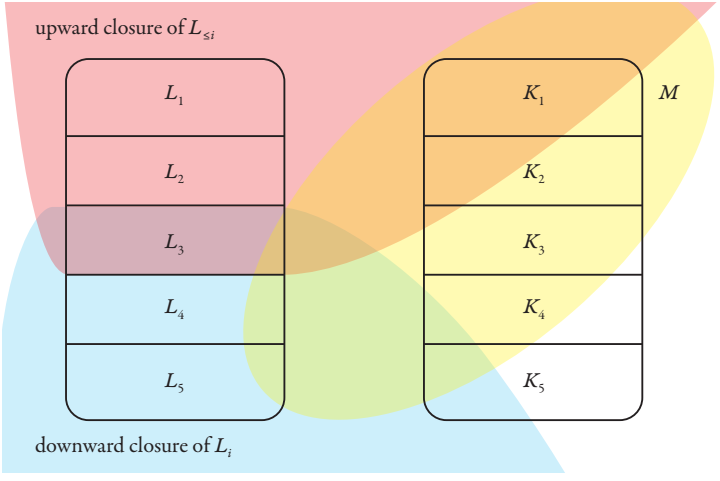
(3)⇒(1) We prove the contra-positive: if zigzags between L and K have bounded length, then L and K can be separated by a piecewise testable language. For $w \in L$ define its *potential* to be the maximal length of a zigzag between L and K s that starts in w; likewise we define the potential for $w \in K$, but using zigzags between K and L. Define $L_i \subseteq L$ to be the words in L with potential exactly $i \in \{1, 2, \ldots\}$, likewise define $K_i \subseteq K$. Our assumption is that the potential is bounded, and therefore L is a finite union of the languages L_i, likewise for K. By induction on $i \in \{0, 1, \ldots\}$, we will show that

$$\underbrace{L_1 \cup \cdots \cup L_i}_{L_{\leq i}} \qquad \text{and} \qquad \underbrace{K_1 \cup \cdots \cup K_i}_{L_{\leq i}}.$$

can be separated by a piecewise testable language, call it M_i. In the induction base, both languages are empty, and can therefore be separated by the empty language, which is clearly piecewise testable. Consider the induction step, where we find the separator M_i. We will use the following sets

$$\underbrace{L_{\leq i}\uparrow}_{\text{upward closure of } L_{\leq i}} \qquad \underbrace{L_i\downarrow}_{\text{downward closure of } L_i} \qquad \underbrace{M}_{\substack{\text{a separator of } K_{<i} \text{ and } L_{<i} \\ \text{from the induction assumption}}}.$$

All of these sets are piecewise testable: the first one is upward closed, the second one is a complement of an upward closed set, and the third one is obtained from the induction assumption. These sets are depicted in the following picture, with $i = 3$:



The set $L_{\leq i}\uparrow$ contains $L_{\leq i}$ by definition, and it is disjoint with K_i, because otherwise there would be some word in $L_{\leq i}$ with potential $i+1$. For similar reasons, the set $L_i\downarrow$ is disjoint with K_i and $L_{\leq i-1}$. Putting these facts together, we see that

$$M_i = L_{\leq i}\uparrow - (M - L_i\downarrow)$$

separates $L_{\leq i}$ from $K_{\leq i}$.

□

The Zigzag Lemma proves that equivalence of the conditions about infinite zigzags and piecewise testability in Theorem 2.18. To finish the proof of the Theorem, we prove that finite infix trivial monoids recognise precisely the languages without infinite zigzags.

Suppose first that L is not recognised by any finite monoid that is infix trivial. We will show that there is an infinite zigzag between L and its complement. If L is not regular, then we are done, since it cannot be piecewise testable, and therefore there must be an infinite zigzag. Assume that L is regular. By assumption, the syntactic monoid of L is not infix trivial. This means that there exist some $a \neq b$ in the syntactic monoid which are in the same equivalence class. By unravelling the definition of the syntactic monoid, we see that there exist words such that for every $n \in \{0, 1, \ldots\}$ we have

$$w_1 \underbrace{(x_1y_1)^n v (x_2y_2)^n}_{\text{represents } a \text{ in the syntactic monoid}} w_2 \in L \qquad \text{iff} \qquad w_1 \underbrace{y_1(x_1y_1)^n v (x_2y_2)^n y_2}_{\text{represents } b \text{ in the syntactic monoid}} w_2 \notin L$$

This sequence of words forms a zigzag.

It remains to show that if L is recognised by a finite infix trivial monoid, then there is no zigzag between L and its complement. For a monoid M and $a, b \in M$, define a zigzag between a and b to be a zigzag, over alphabet M, between the words with product a and the words with product b. If M recognises L, then a zigzag between L and its complement can be used, by extraction, to obtain a zigzag between $a \neq b \in M$. The following lemma shows that this cannot happen, thus completing the proof of Theorem 2.18.

Lemma 2.20. *Let M be finite and infix trivial, and let $a, b \in M$. If there is an infinite zigzag between a and b, then $a = b$.*

Proof The proof is by induction on the infix ordering, lifted tom M^2 coordinatewise:

$$(x, y) \leq (a, b) \qquad \stackrel{\text{def}}{=} \qquad x \text{ is an infix of } a \text{ and } y \text{ is an infix of } b.$$

The induction base is proved the same way as the induction step. Suppose that we have proved the lemma for all pairs $(x, y) \prec (a, b)$.

Claim 2.21. *There exists $n \in \{0, 1, \ldots\}$ and monoid elements $\{a_i, b_i, c_i\}_i$ such that*

$$\begin{array}{lllllllllll} a & = & & c_1 & & c_2 & & \cdots & & c_n & \\ b & = & b_0 & c_1 & b_1 & c_2 & b_2 & \cdots & b_{n-1} & c_n & b_n \\ a & = & a_0 & c_1 & a_1 & c_2 & a_2 & \cdots & a_{n-1} & c_n & a_n \end{array}$$

and for every $i \in \{0, \ldots, n\}$ there is an infinite zigzag between a_i and b_i.

Proof Consider an infinite zigzag between a and b of the form

$$w_1 \hookrightarrow w_2 \hookrightarrow \cdots$$

Let the letters in w_1 be $c_1, \ldots, c_n \in M$. Let $j \in \{2, 3, \ldots\}$. Because of the embedding $w_1 \hookrightarrow w_j$, the word w_j can be factorised as

$$w_j = \underbrace{w_{j,0}}_{M^*} c_1 \underbrace{w_{j,1}}_{M^*} c_2 \cdots c_{n-1} \underbrace{w_{j,n-1}}_{M^*} c_n \underbrace{w_{j,n}}_{M^*}$$

so that for every $i \in \{0, 1, \ldots\}$, the following sequence is a growing chain in the Higman ordering:

$$w_{2,i} \hookrightarrow w_{3,i} \hookrightarrow \cdots .$$

By extracting a subsequence, we can assume that for every $i \in \{0, 1 \ldots, n\}$, the above chain is a zigzag between b_i and a_i, for some $b_i, a_i \in M$. □

Claim 2.22. *There exist $c, c' \in M$ such that $a = cc'$ and $cb = b = bc'$.*

Proof Apply Claim 2.21, yielding monoid elements with

$$\begin{array}{lllllllllll} a & = & & c_1 & & c_2 & & \cdots & & c_n & \\ b & = & b_0 & c_1 & b_1 & c_2 & b_2 & \cdots & b_{n-1} & c_n & b_n \\ a & = & a_0 & c_1 & a_1 & c_2 & a_2 & \cdots & a_{n-1} & c_n & a_n \end{array}$$

For every $i \in \{0, \ldots, n\}$, we can see that $(b_i, a_i) \preceq (b, a)$. If the inclusion is strict, then the induction assumption yields $b_i = a_i$. Otherwise, the inclusion is not strict, and therefore

$$(a_i, b_i) = (a, b).$$

If the inclusion is strict for all i, then the third and second rows in the conclusion of Claim 2.21 are equal, thus proving $a = b$, and we are done. Otherwise, there is some $i \in \{0, \ldots, n\}$ such that $(b_i, a_i) = (b, a)$. By infix triviality, every interval in the second row that contains i will have product b. It follows that

$$\underbrace{c_j b = b}_{\text{for all } j \leq i} \qquad \underbrace{b c_j = b}_{\text{for all } j > i}$$

It is now easy to see that the conclusion of the claim holds if we define c to be the prefix of $a = c_1 \cdots c_n$ that ends with c_i, and define c' to be the remaining suffix. □

Apply the above claim, and a symmetric one with the roles of a and b swapped, yielding elements c, c', d, d' such that

$$a = cc' \quad cb = b = bc' \quad b = dd' \quad da = a = d'. \tag{2.3}$$

We can now prove the conclusion of the lemma:

$$a \overbrace{=}^{(2.3)} (dc)^!(c'd')^! \underbrace{=}_{\text{infix triviality}} (cd)^!(d'c')^! \overbrace{=}^{(2.3)} b.$$

□

Exercises

Exercise 27. Give a polynomial time algorithm, which inputs two nondeterministic automata, and decides if their languages can be separated by a piecwise testable language.

Exercise 28. Consider ω-words, i.e. infinite words of the form

$$a_1 a_2 \cdots \qquad \text{where } a_1, a_2, \ldots \in \Sigma.$$

The Higman ordering naturally extends to ω-words: we write $w \hookrightarrow v$ if w can be obtained from v by removing, possibly infinitely many, positions. Show that the Higman ordering on ω-words is also a well-quasi order, i.e. every upward closed set is the upward closure of finitely many elements.

Bibliography

[1] J. Richard Büchi. “Weak second-order arithmetic and finite automata”. In: *Z. Math. Logik und Grundl. Math.* 6 (1960), pp. 66–92.

[2] Wojciech Czerwinski et al. “A Characterization for Decidable Separability by Piecewise Testable Languages”. In: *Discret. Math. Theor. Comput. Sci.* 19.4 (2017).

[3] Calvin C. Elgot. “Decision problems of finite automata design and related arithmetics”. In: *Trans. Amer. Math. Soc.* 98 (1961), pp. 21–51.

[4] Ronald Fagin. “Generalized first-order spectra and polynomial-time recognizable sets”. In: *Complexity of computation (Proc. SIAM-AMS Sympos. Appl. Math., New York, 1973)*. Providence, R.I.: Amer. Math. Soc., 1974, 43–73. SIAM–AMS Proc., Vol. VII.

[5] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. “Dynamic Word Problems”. In: *J. ACM* 44.2 (Mar. 1997), pp. 257–271.

[6] Wilfrid Hodges. *Model Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1993.

[7] J.A. Kamp. “Tense Logic and the Theory of Linear Order”. PhD thesis. Univ. of California, Los Angeles, 1968.

[8] Manfred Kufleitner. “The Height of Factorization Forests”. In: *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*. Ed. by Edward Ochmanski and Jerzy Tyszkiewicz. Vol. 5162. Lecture Notes in Computer Science. Springer, 2008, pp. 443–454.

[9] Robert McNaughton and Seymour Papert. *Counter-free automata*. The M.I.T. Press, Cambridge, Mass.-London, 1971.

[10] Marcel-Paul Schützenberger. “On finite monoids having only trivial subgroups”. In: *Information and Control* 8 (1965), pp. 190–194.

[11] Imre Simon. “Factorization Forests of Finite Height”. In: *Theoretical Computer Science* 72.1 (1990), pp. 65–94.

[12] Imre Simon. “Piecewise testable events”. In: *Automata Theory and Formal Languages*. Ed. by H. Brakhage. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975, pp. 214–222. ISBN: 978-3-540-37923-2.

[13] Boris A. Trakhtenbrot. “The synthesis of logical nets whose operators are described in terms of one-place predicate calculus (Russian)”. In: *Dokl. Akad. Nauk SSSR* 118.4 (1958), pp. 646–649.

[14] Thomas Wilke. “Classifying Discrete Temporal Properties”. In: *STACS 99*. Ed. by Christoph Meinel and Sophie Tison. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 32–46.

Author index

Subject index