Languages recognised by finite semigroups, and their generalisations to objects such as trees and graphs, with an emphasis on languages definable in monadic second-order logic

Mikołaj Bojańczyk

May 8, 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

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

which 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 bracketing a sequence of elements in the semigroup 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 brackets and write simply

$$abcdef.$$

This means that the product operation in the semigroup can be seen as an operation of type $S^+ \to S$, i.e. it is defined not just on pairs of semigroups elements, but also on finite nonempty words consisting of semigroup elements.

A *semigroup homomorphism* is 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. An equivalent definition of a semigroup homomorphism is requiring the following diagram to commute:

$$\begin{tikzcd}
S^+ \arrow[r, "h^+"] \arrow[d, "\text{product in } S"'] & T^+ \arrow[d, "\text{product in } T"] \\
S \arrow[r, "h"'] & T
\end{tikzcd}$$

In the above, h^+ is the natural lifting of h to words.

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. The product operation in a monoid can be thought of as having type $S^* \to S$, since with the empty word ε being mapped to 1. A *monoid homomorphism* is a semigroup homomorphism that preserves the identity element. In terms of commuting diagrams, a monoid homomorphism is a function which makes the following diagram commute:

$$\begin{tikzcd}
S^* \arrow[r, "h^*"] \arrow[d, "\text{product in } S"'] & T^* \arrow[d, "\text{product in } T"] \\
S \arrow[r, "h"'] & T
\end{tikzcd}$$

Clearly there is a pattern behind the diagrams. This pattern will be explored in the second part of this book, when talking about monads.

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.

[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.

(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 sub-monoid of this one, i.e. the inclusion map is a monoid homomorphism.

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

$$\underbrace{(\{0,1\},+)}_{\text{addition mod 2}} \quad (\{0,1\},\min) \quad \underbrace{(\{0,1\},\pi_1)}_{\text{product } ab \text{ is } a} \quad \underbrace{(\{0,1\},\pi_2)}_{\text{product } ab \text{ is } b} \quad \underbrace{(\{0,1\},(a,b)\mapsto 1)}_{\text{all products are 1}}$$

The first two are monoids.

Compositional functions. 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. □

Commuting diagrams. We finish this section with an alternative description of semigroups which uses commuting diagrams. We include this description, because similar descriptions will be frequently used in this book, e.g. for generalisations of semigroups for infinite words, so we want to start using it as early as possible.

The binary product operation in a semigroup S can be extended to a general operation of type $S^+ \to S$. The following lemma explains, using commuting diagrams, which operations arise this way.

Lemma 1.4. *An operation $\pi : S^+ \to S$ arises from some semigroup operation on S if and only if the following two diagrams commute:*

$$\begin{tikzcd}
S \arrow[d, "\textit{view a letter as a one-letter word}"'] \arrow[dr, "\textit{identity}"] & \\
S^+ \arrow[r, "\pi"'] & S
\end{tikzcd}
\qquad
\begin{tikzcd}
(S^+)^+ \arrow[r, "\textit{product in free semigroup } S^+"] \arrow[d, "\pi^+"'] & S^+ \arrow[d, "\pi"] \\
S^+ \arrow[r, "\pi"'] & S
\end{tikzcd}$$

In the above, π^+ stands for the coordinate-wise lifting of π to words of words.

For monoids, the same lemma holds, with + replaced by $*$. There is no need to add an extra diagram for the monoid identity, since the monoid identity can be defined as the image under π of the empty word ε. The axioms $1 \cdot a$ and $a \cdot 1$ then follow from

$$1 \cdot a = \pi(\varepsilon) \cdot \pi(a) = \pi(\varepsilon a) = \pi(a) = a,$$

and a symmetric reasoning for $a \cdot 1$.

Also homomorphisms can be defined using commuting diagrams. A function $h : S \to T$ is a semigroup homomorphism if and only if the following diagram commutes

$$\begin{tikzcd}
S^+ \arrow[r, "h^+"] \arrow[d, "\text{product in } S"'] & T^+ \arrow[d, "\text{product in } T"] \\
S \arrow[r, "h"'] & T
\end{tikzcd}$$

By replacing + with $*$ we get the definition of a monoid homomorphism.

Exercises

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

Exercise 2. (1) 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. (1) 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 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. (1) 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. (2) 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 $w \sim_X v$.

Exercise 7. (2) Define the *theory of semigroups* to be the set of first-order sentences, which use one ternary relation $x = y \cdot z$, that are true in every semigroup. Show that the theory of semigroups is undecidable, i.e. it is undecidable if a first-order sentence is true in all semigroups.

Exercise 8. (2) Show that the theory of finite semigroups is different from the theory of (all) semigroups, but still undecidable.

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

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 1.5. 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 1.6. *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]

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

[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.

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

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

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

$$\begin{tikzcd}
\forall\exists! & \Sigma^* \arrow[r, two heads, "h"] \arrow[dr, "\text{monoid homomorphism } g \text{ that recognises } L"'] & M \\
& & N \arrow[u, two heads, "\text{monoid homomorphism } f"']
\end{tikzcd}$$

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.

[4] The language need not be regular, and the alphabet need not be finite.

[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 $\forall$ quantifies over surjective homomorphisms, and the same is true for $\exists!$.

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

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

□

Exercise 9. (1) Show that the translation from deterministic finite automata to monoids is exponential in the worst case.

Exercise 10. (1) 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 11. (1) Show that a language $L \subseteq \Sigma^*$ is recognised by a finite commutative monoid if and only if it can be defined by a finite Boolean combination of conditions of the form "letter a appears exactly n times" or "the number of appearances of letter a is congruent to ℓ modulo n".

Exercise 12. (1) Prove that surjectivity of g is important in Theorem 1.7.

Exercise 13. (1) 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.

Exercise 14. (2) Let $\mathcal{L}$ be a class of regular languages with the following closure properties:

- $\mathcal{L}$ is closed under Boolean combinations;
- $\mathcal{L}$ is closed under inverse images of homomorphisms $h : \Sigma^* \to \Gamma^*$;
- Let $L \subseteq \Sigma^*$ be a language in $\mathcal{L}$. For every $u, w \in \Sigma^*$, $\mathcal{L}$ contains the inverse image of L under the following operation:

$$v \mapsto uvw.$$

Show that if L belongs to $\mathcal{L}$, then the same is true for every language recognised by its syntactic monoid.

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 1.8. 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 1.9 (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

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

□

Finiteness is crucial for the above lemma, for example the infinite semigroup

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

contains no idempotents. For $a \in S$, we use the name *idempotent power* for the element a^n, and we use the name *idempotent exponent* for the number n. The idempotent power is unique, but the idempotent exponent is not. It is easy to see that there is always an idempotent exponent which is at most the size of the semigroup, and idempotent exponents are closed under multiplication. Therefore, if a semigroup has n elements, then the factorial $n!$ is an idempotent exponent for every element of the semigroup. This motivates the following notation: we write $a^!$ for the idempotent power of a. The notation usually used in the semigroup literature is a^ω, but we will use ω for infinite words.

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 1.10 (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 1.11 (Egg-box lemma)**.** *The following hold in every finite semigroup.*

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

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

(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 prefixes with suffixes.

Proof

(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 Egg-box 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

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

[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

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

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.

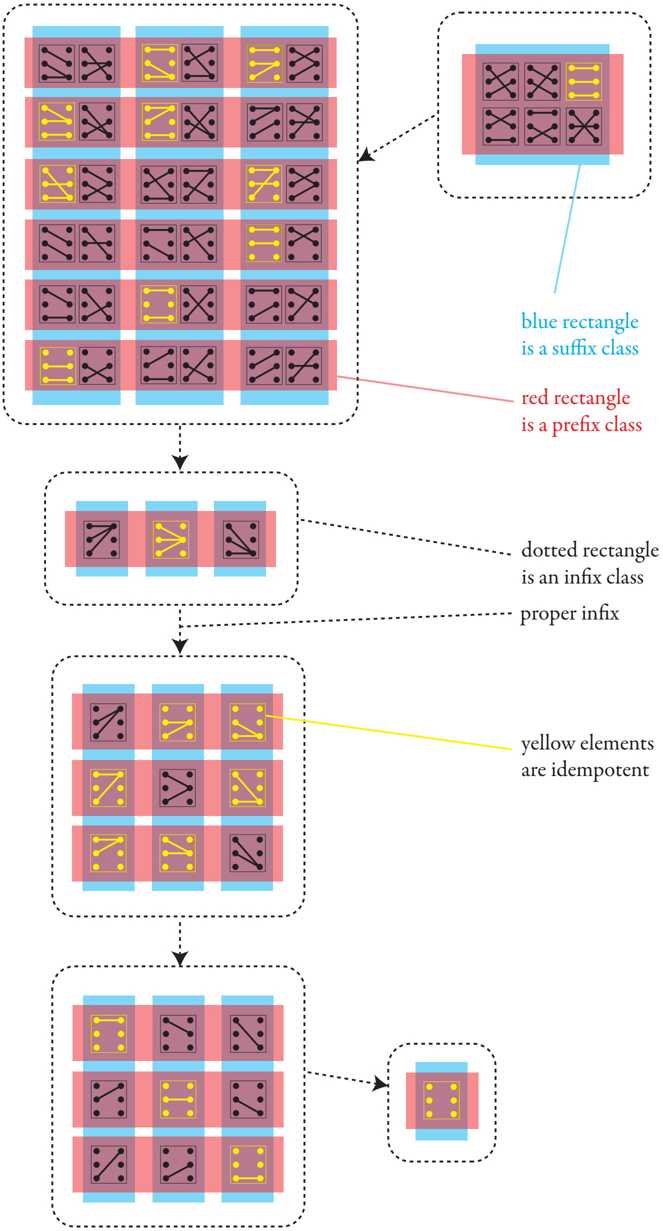


Figure 1.1 The semigroup of partial functions from a three element set to itself, partitioned into prefix, suffix and infix classes. In this particular example, the infix classes are totally ordered, which need not be the case in general.

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

$$\begin{tikzcd} a \arrow[r, bend left, "c\mapsto cx"] & b \arrow[l, bend left, "c\mapsto ycz"] \end{tikzcd}$$

Consider the idempotent exponent $! \in \{1, 2, \ldots\}$ which arises from Idempotent Power Lemma. We have:

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

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

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

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 1.12. *The following maps are mutually inverse bijections*

$$\begin{tikzcd} \textit{prefix class of } a \arrow[r, shift left, "c\mapsto yc"] & \textit{prefix class of } b \arrow[l, shift left, "c\mapsto xc"] \end{tikzcd}$$

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 Egg-box 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 Egg-box 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 1.13. *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 1.12, 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 Egg-box 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 1.14 ($\mathcal{H}$-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) ⇒ (1) ⇒ (2) in the lemma are obvious, so we focus on the remaining implications.

(2)⇒(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 Egg-box 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)⇒(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 Egg-box 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. Take $! \in \{1, 2, \ldots\}$ to be the idempotent exponent which arises from the Idempotent Power Lemma. For every $a \in G$, the power $a^!$ is an idempotent. Since there is only one idempotent in G, we have $a^! = e$. Therefore, $a^{!-1}$ is a group inverse of a.

□

Exercise 15. (1) Show that for every finite monoid, the infix class of the monoid identity is a group.

Exercise 16. (2) 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 17. (2) 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].

Exercise 18. (2) We say that semigroup is *prefix trivial* if its prefix classes are singletons. Show that a finite semigroup S is prefix trivial if and only if it satisfies the identity

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

Exercise 19. (2) 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. We say that a language $L \subseteq \Sigma^*$ is definite if it is a finite Boolean combination of languages of the form $w\Sigma^*$, for $w \in \Sigma^*$. Show that a language is definite if and only if its syntactic semigroup S satisfies the identity

$$x^! = x^! y \qquad \text{for all } x, y \in S.$$

Exercise 20. (2) Show two regular languages such that one is definite and the other is not, but both have isomorphic syntactic monoids.

[7] Let us combine Exercises 16 and 17. By Exercises (16) 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 (17), the corresponding group is unique up to isomorphism. This group is called the *Shützenberger group* of the regular infix class.

Exercise 21. (1) Consider semigroups S which satisfy the following property: (*) that there is an infix class $J \subseteq S$ such that every $a \in S$ is an infix of J, or an absorbing zero element. Show that every finite semigroup is sub-semigroup of a product of finite semigroups that satisfiy (*).

Exercise 22. (2) Show that every finite semigroup satisfies

$$\forall x_1 \, \forall x_2 \, \exists y_1 \, \exists y_2 \, \underbrace{z_1 = z_1 z_1 = z_1 z_2 \land z_2 = z_2 z_2 = z_2 z_1}_{\text{where } z_i = x_i y_i},$$

where quantifiers range over elements of the finite semigroup.

Exercise 23. (2) Show that the following problem is decidable:

- **Input.** Two disjoint sets of variables

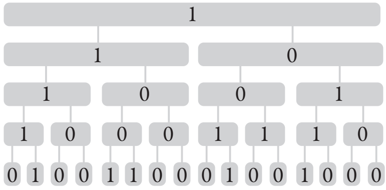
$$X = \{x_1, \ldots, x_n\} \qquad Y = \{y_1, \ldots, y_m\}$$

  and two words $w, w' \in (X \cup Y)^+$.
- **Question.** Is the following true in all finite semigroups:

$$\forall x_1 \, \cdots \, \forall x_n \, \exists y_1 \, \cdots \exists y_m \, \underbrace{w = w'}_{\text{same product}}$$

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:



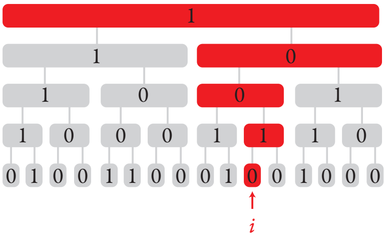
We use the name *factorisation tree* for structures as in the above picture: a *factorisation tree over a semigroup S* is a tree, where nodes are labelled by semigroup elements, such that every node is either a leaf, or is labelled by the semigroup product of the labels of its children. A binary factorisation 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:

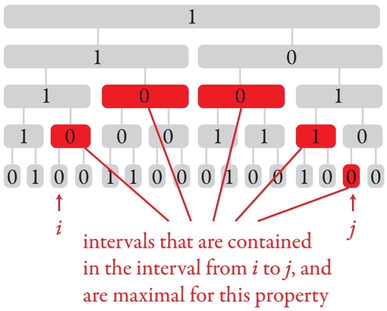


If the factorisation tree is chosen to be balanced, then the updates are processed in logarithmic time. □

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:


1
1 0
1 0 0 1
1 0 0 0 1 1 1 0
0 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0
i
j
intervals that are contained
in the interval from i to j, and
are maximal for this property

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].

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].

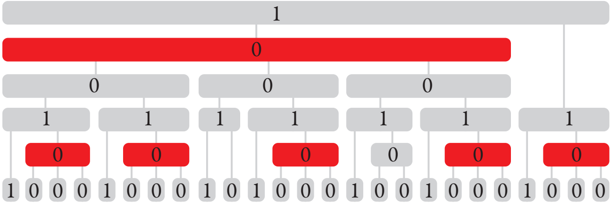
Definition 1.15 (Simon Tree)**.** Define a *Simon tree* to be a factorisation tree where every non-leaf node has one (or both) of the following types:

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:

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

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



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 1.16 (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.

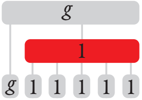
Lemma 1.17. *Let G be a finite group. Every word in G^+ admits a Simon tree of height* $< 3|G|$.

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 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:



[10] The first version of this theorem was proved in [34, Theorem 6.1], with a bound of $9|S|$. The optimal bound is $3|S|$, which was shown in
[21] Kufleitner, “The Height of Factorization Forests”, 2008 , Theorem 1
The proof here is based on Kufleitner, with some optimisations removed.

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.

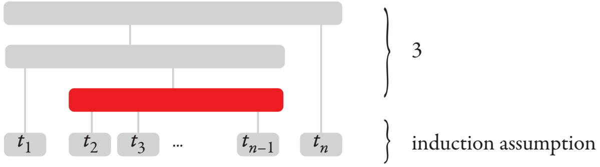
Claim 1.18. *The induction assumption applies to all of* $w_1, \ldots, w_n$.

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:

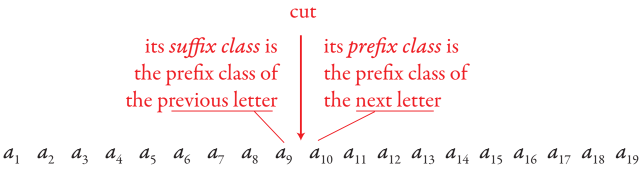


The gray nodes are binary, and the red node is idempotent because every w_i with $1 < i < n$ evaluates to the group identity. □

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

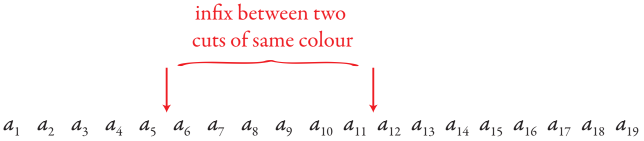
Proof Define a *cut* in a word to be the space between two consecutive letters; in other words this is a decomposition of the word into a nonempty prefix and a nonempty suffix. For a cut, define its *prefix* and *suffix* classes as in the following picture:


cut
its *suffix class* is
the prefix class of
the previous letter
its *prefix class* is
the prefix class of
the next letter
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12} a_{13} a_{14} a_{15} a_{16} a_{17} a_{18} a_{19}

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 Egg-box 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 1.20. *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:


infix between two
cuts of same colour
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12} a_{13} a_{14} a_{15} a_{16} a_{17} a_{18} a_{19}

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 Egg-box 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 1.12, 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 1.21. *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

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

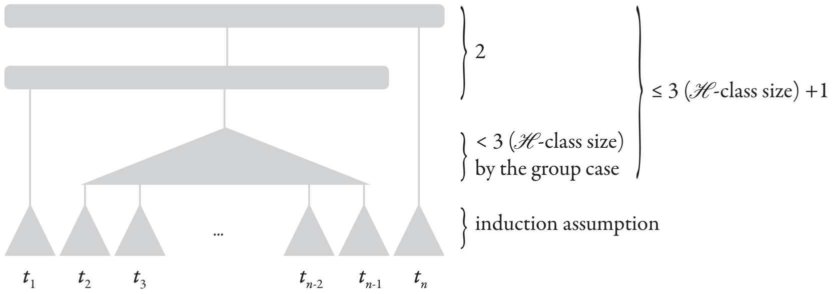
Choose some colour in the colour set of w, which is an $\mathcal{H}$-class H. Cut the word w along all cuts with colour H, yielding a decomposition

$$w = w_1 \cdots w_n.$$

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 1.20, 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 1.17 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 1.19 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 1.19, 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 1.22. *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 Egg-box 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. □

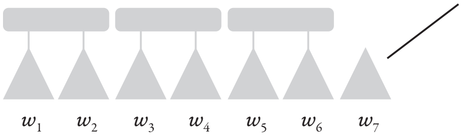
Take a word $w \in S^+$, and cut it along all cuts which are not smooth, yielding a factorisation

$$w = w_1 \cdots w_n.$$

By Claim 1.22, all of the words $w_1, \ldots, w_n$ are smooth, and therefore Lemma 1.19 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 24. (1) 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 25. (2) Show optimality for the previous exercise, in the following sense. Show that for every $k \in \{1, 2, \ldots\}$ there is some semigroup homomorphism

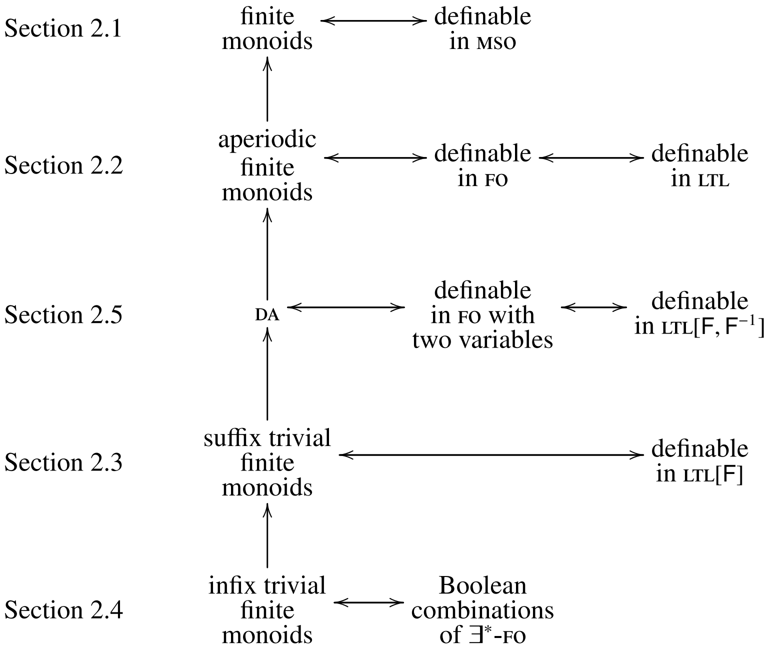
$$h : \Sigma^+ \to S \qquad \text{with } S \text{ finite}$$

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.

Exercise 26. (2) Let $h : \Sigma^* \to M$ be a monoid homomorphism. Consider a regular expression over Σ, which does not use Kleene star L^* but only Kleene plus L^+. Such a regular expression is called h-typed if every subexpression has singleton image under h, and furthermore subexpressions with Kleene plus have idempotent image. Show that every language recognised by h is defined by finite union of h-typed expressions.

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 needed to define languages recognised by this monoid. We consider two kinds of logic: monadic second-order logic MSO and its fragments (notably first-order logic FO), as well as linear temporal logic LTL and its fragments. Here is a map of the results from this chapter, with horizontal arrows being equivalences, and the vertical arrows being strict inclusions.



2.1 All monoids and monadic second-order logic

We begin with monadic second-order logic (MSO), which is the logic that captures exactly the class of regular languages.

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 FO and MSO. Formulas of first-order logic over a given vocabulary are constructed as follows:

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

For the semantics of first-order formulas, we use the notation

$$\mathbb{A}, a_1, \ldots, a_n \models \varphi(x_1, \ldots, x_n)$$

to say that φ is true in the model $\mathbb{A}$, assuming that free variable x_i is mapped to $a_i \in \mathbb{A}$. A *sentence* is a formula without free variables.

The logic MSO extends first-order logic by allowing quantification over subsets of the universe (in other words, monadic relations over the universe, hence the name). The syntax of the logic has 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. 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, which can also quantify over binary relations, ternary relations, etc.).

The following definition associates to each word a corresponding model. With this correspondence, we can use logic to define properties of words.

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 set of positions in the word. The vocabulary is

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

where $x \le y$ is interpreted as the natural order on positions, and with $a(x)$ is interpreted as the set of 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, as witnessed by the sentence:

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

□

Example 4. The language $(aa)^*a \subseteq a^*$ of words of odd length is MSO definable, as witnessed by the sentence:

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

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 $x \le y$ 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: there are languages that are first-order definable in the order model but not in the successor model, see Exercise 38. For the logic MSO, there is no difference between successor and order, since the order can be defined in MSO as follows

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

We now present the seminal Trakhtenbrot-Büchi-Elgot Theorem, which says that MSO describes exactly the regular languages.

Theorem 2.2 (Trakhtenbrot-Büchi-Elgot)**.** *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 restrictions or generalisations of the regular languages. In the case of restrictions, important examples are first-order logic and its fragments, which will be described later in this chapter. In this book, we do not study the generalisations; we are only interested in regular languages. Nevertheless, it is worth mentioning Fagin's Theorem, which says that NP describes exactly the languages definable in existential second-order logic[2] .

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:

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

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 hard part in some generalisations – e.g. for

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

[2] [15] Fagin, "Generalized first-order spectra and polynomial-time recognizable sets", 1974 , Theorem 6

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 MSO over ordered models, we will use 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:

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

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. □

Thanks to the above lemma, instead of MSO over the ordered model, we can work with 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, unlike e.g. formula size, is that formulas of quantifier rank at most $k \in \{0, 1, \ldots\}$ are closed under Boolean combinations. For words $w, w' \in \Sigma^*$, we write

$$w \equiv_k w'$$

if the corresponding set models satisfy the same first-order sentences of quantifier rank at most k. The following lemma is the key to the equivalence of finite semigroups and MSO.

Lemma 2.5 (Compositionality of MSO)**.** *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: *$\equiv_k$ 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*

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

- Congruence: *$\equiv_k$ is a semigroup congruence, i.e.*

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

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. The finite index property is also easy: 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.

We are left with the congruence property. We assume that the reader is familiar with Ehrenfeucht-Fraïssé games[3]. Let $w_1, w_2, w'_1, w'_2 \in \Sigma^*$. The main reason behind the congruence property is that a set of positions in a concatenation of

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

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 a bijection 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

$$\underbrace{w_1w_2 \equiv_k w'_1w'_2}_{\text{big game}}.$$

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 in the vocabulary of the set model, and it would fail if the relation $x = \emptyset$ would be removed from the vocabulary (even though this relation is first-order definable in terms of the remaining ones). Suppose that all k rounds have been played, resulting in tuples

$$\underbrace{a_1, \ldots, a_k}_{\text{sets of positions in } w_1w_2} \qquad \text{and} \qquad \underbrace{a'_1, \ldots, a'_k}_{\text{sets of positions in } w'_1w'_2}.$$

We need to show that the tuples satisfy the same quantifier-free formulas in the corresponding set models. We only prove

$$\underbrace{a_i < a_j}_{\text{in } w_1w_2} \quad \text{iff} \quad \underbrace{a'_i < a'_j}_{\text{in } w'_1w'_2} \qquad \text{for every } i, j \in \{1, \ldots, k\}, \tag{2.1}$$

the other relations from the vocabulary are left as an exercise. For $i \in \{1, \ldots, k\}$

[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. See Exercise 30.

and $\ell \in \{1, 2\}$, define $a_{i,\ell}$ to be the ℓ-th coordinate of $f(a_i)$. Likewise we define $a'_{i,\ell}$, using f'. The main observation is that the relation $<$ in w_1w_2 reduces to a quantifier free properties of the corresponding elements in w_1 and w_2, as explained below:

$$\underbrace{a_i < a_j}_{\text{in } w_1w_2} \quad \text{iff} \quad (\underbrace{a_{i,1} < a_{j,1}}_{\text{in } w_1} \text{ and } \underbrace{a_{j,2} = \emptyset}_{\text{in } w_2}) \text{ or } (\underbrace{a_{j,1} = \emptyset}_{\text{in } w_1} \text{ and } \underbrace{a_{i,2} < a_{j,2}}_{\text{in } w_2}).$$

Since Duplicator's strategy in the big game was obtained by combining winnings strategies for Duplicator in the small games, we know that the truth value of the right side of the equivalence does not change when we go from $a_{i,\ell}$ to $a'_{i,\ell}$, and therefore the same is true for the left side of the equivalence[5]. □

Exercises

Exercise 27. (3) 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 28. (2) For an alphabet Σ, consider the model where the universe is Σ^*, and which is equipped with the following relations:

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

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 29. (1) What happens if the prefix relation in Exercise 28 is replaced by the infix relation?

[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 $x < y$.

Exercise 30. (1) Consider the fragment of second-order logic where one can quantify over: elements, unary relations, and binary relations. (This fragment is expressively complete.) Define $\equiv_k$ to be the equivalence on Σ^* which identifies two words if they a satisfy the same sentences from the above fragment of second-order logic, up to quantifier rank k. Show that this equivalence relation has finite index, but it is not a semigroup congruence.

2.2 Aperiodic semigroups and first-order logic

We now begin the study of fragments of MSO logic. We show that such fragments correspond to structural restrictions on finite monoids. The first – and arguably most important – fragment is first-order logic. This fragment will be described in the Shützenberger-McNaughton-Papert-Kamp Theorem. One part of the theorem says that a language is first-order definable if and only if it is recognised by a finite monoid M which satisfies

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

where $! \in \{1, 2, \ldots\}$ is the idempotent exponent from the Idempotent Power Lemma. 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. Of course the same is true for the complement of the language, which was discussed in Example 4.

Suppose that the parity language is recognised by a homomorphism h into some finite monoid M. By Theorem 1.7, there is a surjective homomorphism from the image of h, which is a sub-monoid of M, into the syntactic monoid. In other words, the syntactic monoid is a quotient (i.e. image under a surjective homomorphism) of a sub-monoid of M. Since the syntactic monoid is the two-element group, which is not aperiodic, and 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 Shützenberger-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:

$$\underbrace{a \in \Sigma}_{\substack{\text{the current}\\ \text{position has}\\ \text{label } a}} \qquad \varphi \wedge \psi \qquad \varphi \vee \psi \qquad \neg\varphi \qquad \underbrace{\varphi \mathsf{U} \psi}_{\varphi \text{ until } \psi} .$$

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 of Boolean combinations are defined as usual. For formulas of the form $\varphi \mathsf{U} \psi$, the semantics[7] are

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

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; this only makes sense for nonempty words. A language $L \subseteq \Sigma^*$ is called LTL *definable* if there is an LTL formula φ that defines the language on nonempty words:

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

For example, the formula $a \mathsf{U} b$ defines the language $\Sigma a^* b \Sigma^*$.

Example 6. To get a better feeling for LTL, we discuss some extra operators that can be defined using until. We write $\perp$ for any vacuously false formula,

[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*.

e.g. $a \wedge \neg a$, likewise $\top$ denotes any vacuously true formula. Here are some commonly used extra operators:

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

Similarly, we define a non-strict version of the operator F, with $\mathsf{F}^*\varphi = \varphi \vee \mathsf{F}\varphi$. For example, the formula

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

says that the last position in the word has label a. □

Almost by definition, every LTL definable language is also first-order definable. Indeed, by unfolding the definition, one sees that for every LTL formula there is 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. However, 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:

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

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^*. □

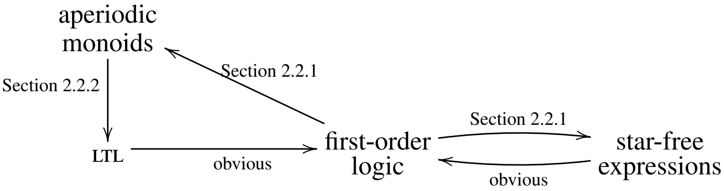
Like for LTL formulas, almost by definition every star-free expression describes a first-order definable language. This is because to every star-free expression one can associate a first-order formula $\varphi(x, y)$ which selects a pair of positions $x \leq y$ if and only if the corresponding infix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

Equivalence of the models. The Shützenberger-McNaughton-Papert-Kamp Theorem says that all of the models discussed so far in this section are equivalent.

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

(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 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 compositionality analysis of Ehrenfeucht-Fraïssé games, similar to 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 ordered models of w and w' satisfy the same first-order formulas of quantifier rank at most k. We use the blue colour to distinguish the equivalence from

[8] This theorem combines three equivalences.
The equivalence aperiodic monoids and star-free expressions was shown in
[31] Schützenberger, “On finite monoids having only trivial subgroups”, 1965 , p. 190
The equivalence of star-free expressions and first-order logic was shown in
[24] 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
[20] Kamp, “Tense Logic and the Theory of Linear Order”, 1968

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 be 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 equivalence classes is aperiodic. 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 M of $\equiv_{k+1}$. Let be X the set of triples

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

By Lemma 2.8, if L, a, K are as above, then M is either contained in LaK or disjoint with LaK. Therefore, the equivalence class M is equal to the following Boolean combination of concatenations

$$\bigcap_{\substack{(L,a,K)\in X\\ M\subseteq LaK}} LaK \qquad \cap \qquad \bigcap_{\substack{(L,a,K)\in X\\ LaK\cap M=\emptyset}} \overline{LaK}.$$

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 use Lemma 2.8 and induction on k to show that

$$w^{2^k-1} \equiv_k w^{2^k} \qquad \text{for every } w \in \Sigma^* \text{ and } k \in \{1, 2, \ldots\}.$$

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. 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*

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

is 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 the subset 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 colourings with one possible colour are clearly LTL definable. Consider the case when the Σ contains only one element $a \in S$. By aperiodicity, the sequence

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

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

is eventually constant, because all powers bigger than the threshold ! give the same result. 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

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

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 of S, call it π. By aperiodicity,

$$\pi^! \circ \pi = \pi^!.$$

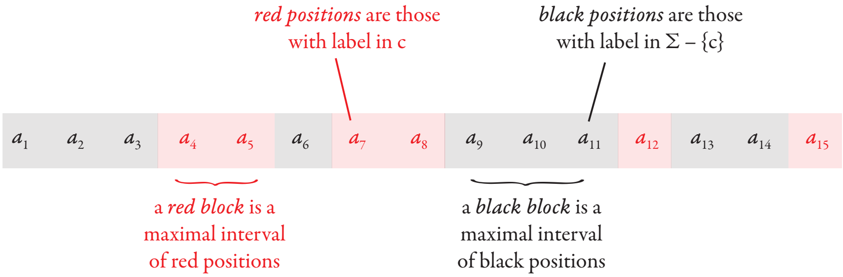
Since permutations form a group, it follows that π is the identity. □

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.

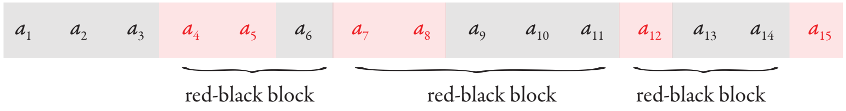
Claim 2.12. *T is a sub-semigroup of S.*

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 $w \in \Sigma^+$:


red positions are those with label in c
black positions are those with label in Σ − {c}
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12} a_{13} a_{14} a_{15}
a *red block* is a maximal interval of red positions
a *black block* is a maximal interval of black positions

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 red 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 LTL *transduction* if it has the form

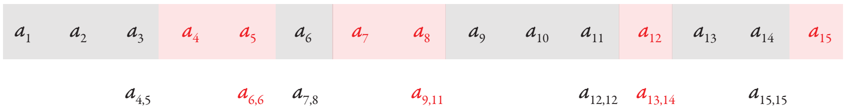
$$a_1 \cdots a_n \in \Sigma^* \qquad \mapsto \qquad f(a_1 \cdots a_n) f(a_2 \cdots a_n) \cdots f(a_n)$$

for some LTL definable colouring $f : \Sigma^+ \to \Gamma + \varepsilon$. By substituting formulas, one easily shows the following composition properties:

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

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

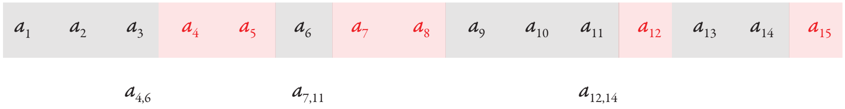
(1) For each position that precedes a block (i.e. the next position begins a new block), write in that position the value of the next block. For the remaining positions, do not write anything. Use two disjoint copies of S to distinguish the values of the red and black blocks. Here is a picture:



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[10].

[10] To make this formal, we need a simple closure property of LTL that is described in Exercise 36.

(2) Take the output of the function in the previous step, and for each red letter (the product of a red block), multiply it with the next letter (which is the 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 31. (2) 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 32. (2) 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 33. (1) 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 34. (1) Show two languages which have the same syntactic monoid, and such that only one of them is first-order definable in the successor model. In particular, one of the closure properties from Exercise 14 must fail for this logic.

Exercise 35. (3) Let Σ be a finite alphbet and let $\vdash, \dashv$ be fresh symbols. For $k, \ell \in \{0, 1, \ldots\}$, we say that $w, w' \in \Sigma^*$ are (k, ℓ)-locally equivalent if

$$\begin{array}{ccc} \vdash w \dashv \text{ has at least } i & \text{iff} & \vdash w' \dashv \text{ has at least } i \\ \text{occurrences of infix } v & & \text{occurrences of infix } v \end{array}$$

holds for every $i \in \{0, \ldots, k\}$ and every $v \in \Sigma^*$ of length at most ℓ. Show that $L \subseteq \Sigma^*$ is first-order definable in the successor model if and only if it is a union of equivalence classes of (k, ℓ)-local equivalence, for some k, ℓ.

Exercise 36. (1) Let $\Gamma \subseteq \Sigma$ and let $L \subseteq \Gamma^*$. If L is definable in LTL, then the same is true for

$$\{w \in \Sigma^* : L \text{ contains the maximal prefix of } w \text{ which uses only letters from } \Sigma\}.$$

Exercise 37. (2) Consider LTL[X], i.e. the fragment of LTL where the only operator is X. Show that this fragment is equal to the definite languages from Exercise 19.

Exercise 38. (1) Show that if a language is first-order definable in the successor model, then the syntactic semigroup satisfies the following equality

$$eafbecf = ecfbeaf \qquad \text{for all } \underbrace{e, f,}_{\text{idempotents}} a, b, c.$$

Exercise 39. (3) Show that the identity in Exercise 38, together with aperiodicity, is equivalent to first-order definability in the successor model.

Exercise 40. (3) 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 32). What about monoids with trivial suffix classes, prefix classes, or infix classes? Trivial infix classes will be described in Section 2.4. In this section, we give a logical characterisation of trivial suffix classes (of course, a symmetric statement holds for trivial prefix classes).

In the characterisation, we use the fragment of LTL where until is replaced by the following 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*

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

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 M, the recognised language is defined by a similar expression, with monoid elements substituted by the letters that map to them (such a substitution preserves suffix unambiguity).

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 15, 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

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

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) ⇒ (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 use the formula

$$\varphi_n = \mathsf{G}^* \underbrace{\bigvee_{a \in \Sigma_n} a}_{\text{all positions have label in } \Sigma_n} .$$

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)):

$$\psi_i = a_i \wedge (\mathsf{F}\varphi_i) \wedge \mathsf{G} \bigvee_{j>i} \varphi_j .$$

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

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

(3) ⇒ (1) Define the rank of a formula in LTL[F] to be the nesting depth of the operator F. For $k \in \{0, 1, \ldots\}$, define $\approx_k$ to be the equivalence relation on Σ^+ which identifies two words if they satisfy the same formulas of rank at most k. The key observation is the following pumping lemma.

Claim 2.14. *For every $k \in \{0, 1, 2, \ldots\}$ we have*

$$w(xy)^i u \quad \approx_k \quad wy(xy)^j u \qquad \textit{for every } w \in \Sigma^+, x, y, u \in \Sigma^* \textit{ and } i, j \geq k.$$

Proof Induction on k. For $k = 0$, we observe that the equivalence class under $\approx_0$ depends only on the first letter, and the two words on both sides in the claim have the same letter because w is nonempty.

Consider now the induction step, when going from k to $k+1$. By unravelling the definition of $\approx_{k+1}$, we need to show that if $i, j \geq k+1$, then for every nonempty proper suffix of the word on one side of equivalence

$$w(xy)^i u \quad \approx_{k+1} \quad wy(xy)^j u$$

there is a nonempty proper suffix on the other side of the equivalence, such that the two suffixes are equivalent under $\approx_k$. Suppose first that v is a nonempty suffix of the left side. If v is a suffix of $(xy)^{k+1}u$, then the same v is a suffix of the right side. Otherwise, we can use the induction assumption. Consider now a nonempty proper suffix v of the right side. Here we argue in the same way as previously, except that there is one extra case, when

$$v = z(xy)^i u \qquad \text{for some } z \text{ that is a suffix of } y.$$

In this case, the $\approx_k$-equivalent suffix on the left side is $z(xy)^k u$. □

By unravelling the definition of the syntactic monoid, in terms of two-sided congruences, we infer from the above claim that for every rank k 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

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

□

Exercises

Exercise 41. (1) Let Σ be an alphabet and let $c \notin \Sigma$ be a fresh letter. Show that $L \subseteq \Sigma^+$ satisfies the conditions of Theorem 2.13 if and only cL is definable in LTL[F].

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 embeddings (also known as the Higman ordering).

Definition 2.15 (Embedding)**.** We say that $w \in \Sigma^*$ *embeds* in $v \in \Sigma^*$, denoted by $w \hookrightarrow v$, if there is an injective function from positions in w to positions in v, which preserves the order on positions and the labels.

In other words, w embeds in v if and only if w can be obtained from v by removing zero or more positions. For example "ape" embeds into "example". It is easy to see that embedding is an ordering: it is reflexive, transitive and anti-symmetric (although it will cease to be anti-symmetric for infinite words). 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 embedding 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*

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

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*

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

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. □

Embeddings 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[11]. For languages $L, K \subseteq \Sigma^*$, define a *zigzag between L and K* to be a sequence

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

In other words, this is a sequence that is growing with respect to embeddings, and such that odd-numbered elements are in L and odd-numbered elements are in K. The zigzag does not need to be strictly growing, but it will be if L and K are disjoint.

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

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

Theorem 2.18. *The following conditions are equivalent*[12] *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 zigzags between L and K can have arbitrarily long 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 in the graph that start in v. This can be either a finite number, or ω if the paths have unbounded length.

We first show that some vertex must have potential ω. 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) embedding, in the following sense:

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

By Higman's Lemma, the language L, like any set of words, has finitely many minimal elements with respect to embedding. By monotonicity, one of these minimal words must therefore have potential ω.

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

For the same reason as above, 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 embedding. 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 either no elements of the zigzag, or all but finitely many elements of the zigzag. It follows that every finite Boolean combination of upward closed sets must contain, or be disjoint with, two consecutive elements of the zigzag. Therefore, such a Boolean combination cannot separate L from K.

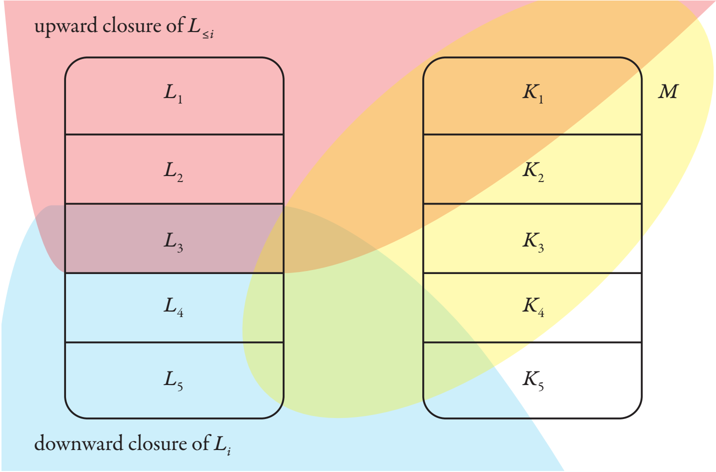
(3)⇒(1) We prove the contra-positive: if zigzags between L and K have bounded length, then L and K can be separated by a piecewise testable language. For $w \in L$ define its *potential* to be the maximal length of a zigzag between L and K that starts in w; likewise we define the potential for $w \in K$, 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 the 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. It is also disjoint with K_i, because otherwise there would be some words

$$\underbrace{w}_{L_{\leq i}} \hookrightarrow \underbrace{v}_{K_i},$$

and therefore w would need to have 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 show that the syntactic monoid of L is finite and infix trivial (which is the same as saying that some recognising monoid is finite and infix trivial) if and only if there is no infinite zigzag between L and its complement.

Suppose first that the syntactic monoid of L is not finite or infix trivial. If the syntactic monoid is not finite, then the language cannot be piecewise testable, since piecewise testable languages are necessarily regular. Assume therefore that the syntactic monoid is finite but not infix trivial. This means that the syntactic monoid is either not prefix trivial, or not suffix trivial. By symmetry, we only consider the case where the syntactic monoid is not suffix trivial. This means that there exist a, b in the syntactic monoid such that

$$(ab)^! \neq b(ab)^!.$$

By unravelling the definition of the syntactic monoid, the above disequality can be easily used to create an infinite zigzag between L and its complement.

It remains to show that if the syntactic monoid of L is finite and infix trivial, then there is no zigzag between L and its complement. Let M be the syntactic monoid. For $a, b \in M$, 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 to pairs:

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

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

Claim 2.21. *If there is an infinite zigzag between a and b, then there exists $n \in \{0, 1, \ldots\}$ and monoid elements $\{a_i, b_i, c_i\}_i$ such that*

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

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 $c_1, \ldots, c_n \in M$. For $j \geq 2$, define an *important position* in w_j to be any position that arises by starting in some position of w_1, and then following the embeddings

$$w_1 \hookrightarrow w_2 \hookrightarrow \cdots \hookrightarrow w_j.$$

By distinguishing the important positions in w_j, we get a factorisation

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

By definition of important positions, for every $i \in \{0, \ldots, n\}$ the following sequence is growing with respect to embedding:

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

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 $b_i, a_i \in M$. This proves the conclusion of the claim. □

Claim 2.22. *If there is an infinite zigzag between a and b, then there exist $c, c' \in M$ such that $a = cc'$ and $cb = b = bc'$.*

Proof Apply Claim 2.21, yielding monoid elements with

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

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 of the lemma 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

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

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 42. (2) Prove Higman's Lemma.

Exercise 43. (2) Give a polynomial time algorithm, which inputs two nondeterministic automata, and decides if their languages can be separated by a piecwise testable language.

Exercise 44. (1) Consider ω-words, i.e. infinite words of the form

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

Embedding naturally extends to ω-words (in fact, any labelled orders). Show that the embedding on ω-words is also a well-quasi order, i.e. every upward closed set is the upward closure of finitely many elements.

2.5 Two-variable first-order logic

We finish this chapter with one more monoid characterisation of a fragment of first-order logic. A corollary of the equivalence of first-order logic and LTL (or of the equivalence of first-order logic and star-free expressions) is that, over finite words, first-order logic is equivalent to its three variable fragment. What about one or two variables?

It is an easy exercise to show that first-order logic with one variable defines exactly the languages that are recognised by monoids that are aperiodic and commutative:

$$a^! = a^{!+1} \quad ab = ba \qquad \text{for all } a, b.$$

The more interesting case is first-order logic with two variables, which we denote by FO^2. This logic is characterised in the following theorem.

Theorem 2.23. *For a language $L \subseteq \Sigma^*$, the following are equivalent:*

(1) *Definable in two variable first-order logic;*
(2) *Recognised by a finite monoid M with the following property*[13]*: M is a aperiodic, and if an infix class $J \subseteq M$ contains an idempotent, then J is a sub-semigroup of M.*

[13] This property appears in
[32] Schützenberger, "Sur Le Produit De Concatenation Non Ambigu", 1976
where it is used to characterise certain unambiguous regular expressions, see Exercise 49.

We use the name DA for the monoids (more generally, finite semigroups) that satisfy the property in item (2). In the exercises, we add several other equivalent conditions for the above theorem, including the temporal logic LTL$[\mathsf{F}, \mathsf{F}^{-1}]$ and the following fragment of first-order logic:

$$(\text{definable by a } \exists^*\forall^*\text{-sentence}) \quad \cap \quad (\text{definable by a } \forall^*\exists^*\text{-sentence}).$$

The rest of Section 2.5 is devoted to proving the theorem. We begin with a equational description of DA. (A stronger equational description is given in Exercise 45.)

Lemma 2.24. *A finite semigroup S is in* DA *if and only if it satisfies:*

$$\underbrace{w^! = w^! v w^!}_{\substack{\text{the equality means that}\\ \text{the two words have the same product}}} \qquad \text{for all } w, v \in M^* \text{ with } v \hookrightarrow w.$$

Proof We first prove that the identity implies that S is DA. Let e be an idempotent. We need to show that if a, b are infix equivalent to e, then the same is true for ab. Because a, b are infixes of e, and e is an idempotent, one can find word w in S^+ which has product e, and where both a and b appear. In particular, $ab \hookrightarrow w$. By the identity in the lemma, we know that $e = eabe$, and therefore ab is an infix of e.

We now show that if S is in DA, then the identity is satisfied. Let $v \hookrightarrow w$ be as in the identity. Let e be the product of $w^!$, and let J be the infix class of e. This infix class is a semigroup, by definition of DA. For every letter a that appears in the word w, there is a suffix of $w^! w^!$ which begins with a and has product in J. Let $a' \in J$ be the product of this suffix. Since J is a semigroup, it follows that $ea' \in J$ and therefore also $ea \in J$. Since $ea \in J$ holds for every letter that appears in w, it follows that $eve \in J$. This means that eve is in the $\mathcal{H}$-class of e, and therefore $e = eve$ by aperiodicity (which is part of the definition of DA), thus establishing the identity. □

We now prove the theorem.

To prove the implication (1)⇒(2), we show that for every language definable in FO^2, its syntactic monoid belongs to DA. By Lemma 2.24 and unravelling the definition of the syntactic monoid, it is enough to show that for every $w_1, w_2, v, w \in \Sigma^*$ and $n \in \{0, 1, \ldots\}$, if $v \hookrightarrow w$ then the words

$$w_1 w^n w_2 \qquad w_1 w^n v w^n w_2$$

satisfy the same FO^2 sentences of quantifier rank at most n. This is shown using a simple Ehrenfeucht-Fraïssé argument.

For the implication (2)⇒(1), we use the following lemma.

Lemma 2.25. *Let M be a monoid in* DA, *and let $a_1, a_2 \in M$. Then*

$$w \in M^* \qquad \mapsto \qquad \underbrace{a_1 \cdot (\textit{product of } w) \cdot a_2}_{\in M}$$

is a colouring definable in FO^2, *i.e. every inverse image is definable in* FO^2.

If we apply the above lemma to a_1 and a_2 being the monoid identity, we conclude that the product operation is definable in FO^2. This implies that every language recognised by the monoid is definable in FO^2, thus proving the implication (1) $\Leftarrow$ (2) in the theorem.

Proof Induction on the following parameters, ordered lexicographically:

(1) size of M;
(2) number of elements that properly extend a_1 in the prefix ordering;
(3) number of elements that properly extend a_2 in the suffix ordering.

The induction base is when M has one element, in which case the colouring in the lemma is constant, and therefore definable in FO^2.

Let us also prove another variant of the induction base, namely when induction parameters (2) and (3) are zero, which means that a_1 is maximal in the prefix ordering and a_2 is maximal in the suffix ordering. It follows that, for

$$\mathcal{H}\text{-class of } a_1 a a_2 \;=\; \mathcal{H}\text{-class of } a_1 b a_2 \qquad \text{for all } a, b \in M.$$

Since DA implies aperiodicity, which implies $\mathcal{H}$-triviality, the colouring in the statement of the lemma is constant, and therefore definable in FO^2.

It remains to prove the induction step. Because of the two kinds of induction base that were considered above, we can assume that one of the parameters (2) or (3) is nonzero. By symmetry, assume that a_1 is not maximal in the prefix ordering.

Claim 2.26. *For every $a \in M$, the following is a sub-monoid of M:*

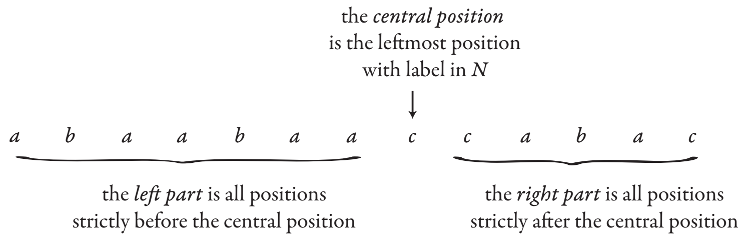
$$\underbrace{\{b \in M : ab \textit{ is prefix equivalent to } a\}}_{\textit{we call this set the prefix stabiliser of } a}.$$

Proof The prefix stabiliser clearly contains the monoid identity. It remains to show that it is closed under composition. Let b, c be in the prefix stabiliser of a. Using the definition of the prefix stabiliser, it is easy to construct a word $w \in M^*$, such that $bc \hookrightarrow w$ and $aw = a$. By Lemma 2.24, it follows that

$$a = aw = aw^! = aw^! bcw^! = abcw^!,$$

which establishes that bc is in the prefix stabiliser of a. □

Let $N \subseteq M$ be the prefix stabiliser of a_1; our assumption says that N is a proper subset of M, and by the above claim it is also a sub-monoid. We decompose a word $w \in M^*$ into three parts, as explained in the following picture:


the *central position*
is the leftmost position
with label in *N*
a b a a b a a c c a b a c
the *left part* is all positions
strictly before the central position
the *right part* is all positions
strictly after the central position

There is an FO^2 formula which selects the central position. Since all labels in the left part are from N, we can use the induction assumption on a smaller monoid to prove that the colouring

$$w \quad \mapsto \quad \text{product of left part}$$

is definable in FO^2. (When using the induction assumption, we restrict all quantifiers of the formulas from the induction assumption so that they quantify over positions that are to the left of the central position.) Let c be the product of the prefix up to and including the central position; as we have shown above, this product can be computed in FO^2. By definition of the central position, we know that a_1 is a proper prefix of a_1c, and therefore we can use the induction assumption to prove that

$$w \quad \mapsto \quad a_1c \cdot (\text{product of right part}) \cdot a_2$$

is a colouring definable in FO^2. The conclusion of the lemma follows. □

Exercises

Exercise 45. (2) Show that a semigroup belongs to DA if and only if it satisfies the identity

$$(ab)^! = (ab)^! a (ab)^! \qquad \text{for all } a, b.$$

Exercise 46. (2) Show that FO^2 has the same expressive power as LTL$[\mathsf{F}, \mathsf{F}^{-1}]$, which is the extension of LTL$[\mathsf{F}]$ with the following past operator:

$$w, x \models \mathsf{F}^{-1}\varphi \quad \stackrel{\text{def}}{=} \quad \exists y\; y < x \;\wedge\; w, y \models \varphi.$$

Exercise 47. (3) Define the *syntactic ordering* on the syntactic monoid, which depends on the accepting set F, as follows:

$$a \leq b \quad \stackrel{\text{def}}{=} \quad \forall x, y \in M \; xay \in F \Rightarrow xby \in F.$$

Show that a language can be defined by a first-order sentence of the form

$$\underbrace{\exists x_1 \cdots \exists x_n \forall y_1 \cdots \forall y_m \quad \overbrace{\varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)}^{\text{quantifier-free}}}_{\text{such a formula is called an } \exists^*\forall^*\text{-sentence}}$$

if and only if

$$w^! \leq w^! v w^! \qquad \text{for all } \underbrace{v \hookrightarrow w}_{\text{Higman ordering}}$$

Hint[14]: use Exercise 26.

Exercise 48. (3) Show that L is definable in FO^2 if and only both L and its complement can be defined using $\exists^*\forall^*$-sentences.

Exercise 49. (2) We say that a regular expression

$$\Sigma_0^* a_1 \Sigma_1^* \cdots \Sigma_{n-1}^* a_n \Sigma_n^*$$

is *unambiguous* if every word w admits at most one factorisation

$$w = w_0 a_1 w_1 \cdots w_{n-1} a_n w_n \qquad \text{where } w_i \in \Sigma_i^* \text{ for all } i \in \{1, \ldots, n\}.$$

Show that a language is a finite disjoint union of unambiguous expressions if and only if its syntactic monoid of L is in DA[15].

[14] An effective characterisation of $\exists^*\forall^*$-sentence was first given in
[2] Arfi, “Polynomial Operations on Rational Languages”, 1987 , Theorem 3.
The proof was simplified in
[25] Pin and Weil, “Polynomial closure and unambiguous product”, 1997 , Theorem 5.8
The solution which uses Exercise 26 is based on [25]. Characterisations of fragments of first-order logic such as $\exists^*\forall^*$ are widely studied, see
[26] Place and Zeitoun, “Going Higher in First-Order Quantifier Alternation Hierarchies on Words”, 2019

[15] This exercise is based on
[32] Schützenberger, “Sur Le Produit De Concatenation Non Ambigu”, 1976

3
Infinite words

In this chapter, we study infinite words.

In Section 3.1, we begin with the classical model of infinite words, namely ω-words. In ω-word, the positions are ordered like the natural numbers. We show how the structure of finite semigroups, which was developed in Section 1.2, can be applied to prove McNaughton's Theorem about determinisation of ω-automata.

In Section 3.2, we move to more general infinite words, where the positions can be any countable linear order, e.g. the rational numbers. For this kind of infinite words, we define a suitable generalisation of semigroups, and show that it has the same expressive power as monadic second-order logic.

3.1 Determinisation of Büchi automata for ω-words

An ω-word is a function from the natural numbers to some alphabet Σ. We write Σ^ω for the set of all ω-words over alphabet Σ. To recognise properties of ω-words, we use Büchi automata.

Definition 3.1 (Büchi automata)**.** The syntax of a *nondeterministic Büchi automaton* is the same as the syntax of a nondeterministic finite automaton for finite words, namely it consists of:

$$\underbrace{Q}_{\text{states}} \qquad \underbrace{\Sigma}_{\substack{\text{input}\\ \text{alphabet}}} \qquad \underbrace{I, F \subseteq Q}_{\substack{\text{inital and}\\ \text{final states}}} \quad \underbrace{\delta \subseteq Q \times \Sigma \times Q}_{\text{transition relation}}.$$

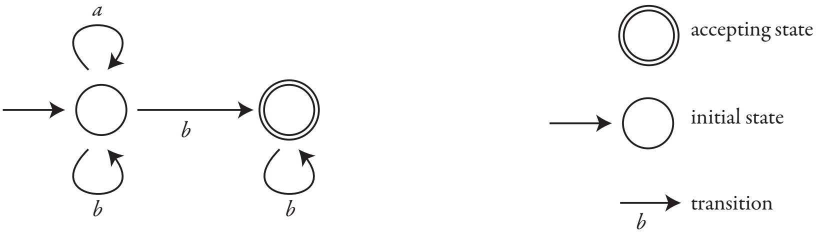
The difference, with respect to nondeterministic automata on finite words, is in the semantics: a word in Σ^ω is accepted by the automaton if there exists a run which begins in an initial state, and which satisfies the *Büchi condition*:

some accepting state appears infinitely often in the run.

A *deterministic Büchi automaton* is the special case when there is one initial state, and the transition relation is a function from $Q \times \Sigma$ to Q.

The following example shows that deterministic Büchi automata are weaker than than nondeterministic ones.

Example 8. Consider the language of ω-words over alphabet $\{a, b\}$ where letter a appears finitely often. This language is recognised by a nondeterministic Büchi automaton as in the following picture:



The idea is that the automaton nondeterministically guesses some position which will not be followed by any a letters; this guess corresponds to the horizontal transition with label b in the picture.

This language is not recognised by any deterministic Büchi automaton. Toward a contradiction, imagine a hypothetical deterministic Büchi automaton which recognises the language. Run this automaton on b^ω. Since a appears finitely often in this ω-word, the corresponding run (unique by determinism) must use an accepting state in some finite prefix. Extend that finite prefix by appending ab^ω. Again, the word must be accepted, so an accepting state must be eventually visited after the first a. By repeating this argument, we get a word which has infinitely many a's and where the (unique) run of the deterministic automaton sees accepting states infinitely often; a contradiction. □

The above shows that languages recognised by deterministic Büchi automata are not closed under Boolean combinations. This turns out to be the only limitation of the model, as shown in the following theorem.

Theorem 3.2. *If a language of ω-words is recognised by a nondeterministic Büchi automaton, then it is a Boolean combination of languages recognised by deterministic Büchi automata*[1].

[1] A Boolean combination of deterministic Büchi automata is the same thing as what is known as a *deterministic Muller automaton*. Therefore, the theorem is the same McNaughton's Theorem,

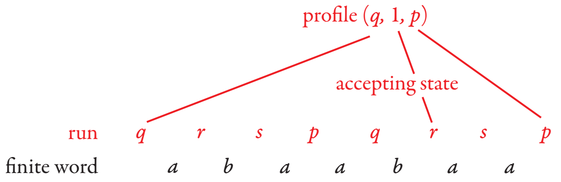
The converse implication in the above theorem is also true, which is left as an exercise for the reader (Exercise 51). A language is called *ω-regular* if it is recognised by a nondeterministic Büchi automaton, or equivalently, by a Boolean combination of deterministic Büchi automata. The ω-regular languages are closed under Boolean combination thanks to the deterministic characterisation. The original application of Büchi automata was Büchi's proof[2] that they recognise exactly the same languages of ω-words as monadic second-order logic; this application is a simple corollary of Theorem 3.2, see Exercise 55.

There are several combinatorial proofs for the determinisation result in Theorem 3.2. In this section, we present an algebraic proof, which leverages the structural theory of finite semigroups developed in Section 1.2.

Let $\mathcal{A}$ be a nondeterministic Büchi automaton, with states Q and input alphabet Σ. For an ω-word, define its *ω-type* to be the set of states from which the word is accepted. We also define the type for finite words, but here we need to store a bit more information. For a run of the automaton over a finite word, define the *profile* of the run to be the triple (q, i, p) where q is the source state of the run, p is the target state of the run, and

$$i = \begin{cases} 0 & \text{if the run does not use any accepting state} \\ 1 & \text{if the run uses some accepting state.} \end{cases}$$

Here is a picture of a run with its profile:



Define the *type* of a finite word $w \in \Sigma^+$ to be the set of profiles of runs over this word. It is not hard to see that the function

$$w \in \Sigma^+ \quad \mapsto \quad \text{type of } w \in \underbrace{\mathsf{P}(Q \times \{0,1\} \times Q)}_{S}$$

[23] McNaughton, "Testing and generating infinite sequences by a finite automaton", 1966 , p. 524
which says that nondeterministic Büchi automata can be determinised into deterministic Muller automata.

[2] [6] Büchi, "On a decision method in restricted second order arithmetic", 1962

is a semigroup homomorphism, with a naturally defined semigroup structure on S. The following lemma shows that types and ω-types are compatible.

Lemma 3.3. *If $w_i \in \Sigma^+$ and $v_i \in \Sigma^+$ have the same type for every $i \in \{1, 2, \ldots\}$, then $w_1 w_2 \cdots \in \Sigma^\omega$ and $v_1 v_2 \cdots \in \Sigma^\omega$ have the same ω-type.*

Proof By substituting parts of an accepting run, while preserving the Büchi condition. □

Thanks to the above lemma, it makes sense to talk about the ω-type of a word $w \in S^\omega$ built out of types. In particular, it makes sense to say whether or not a word $w \in S^\omega$ is accepted by $\mathcal{A}$, since this information is stored in the type. A special case of this notation is ae^ω, where $a, e \in S$, which is the ω-type of the ω-word that begins with letter a and has all other letters equal to e. The importance of this special case is explained by the following lemma about factorisations of ω-words[3]

Lemma 3.4. *For every $w \in S^\omega$ there exist $a, e \in S$, such that e is an idempotent, $ae = e$, and there is a factorisation*

$$w \quad = \quad \overbrace{w_0}^{\text{type } a} \; \overbrace{w_1}^{\text{type } e} \; \overbrace{w_2}^{\text{type } e} \; \overbrace{w_3}^{\text{type } e} \cdots$$

Proof Define a *cut* in w to be the space between two positions. Consider an undirected edge-labelled graph, defined as follows. Vertices are cuts. For every two distinct cuts, there is an undirected edge, labelled by the type of the finite word that connects the two cuts. By Ramsey's Theorem A, see Exercise 50, there exists a type $a \in S$ and an infinite set X of vertices, such every two distinct vertices from X are connected by an edge with label e. Define the decomposition from the lemma to be the result of cutting w along all cuts from X. By assumption on X, every word w_i with $i > 0$ has type e. Idempotence of e follows from

$$\overbrace{\underbrace{w_i}_{e} \; \underbrace{w_{i+1}}_{e}}^{e}.$$

Finally, we can assure that $ae = a$ by joining the first two groups. □

A corollary of Lemmas 3.3 and 3.4 is that $w \in L$ if and only if

(*) there is a factorisation as in Lemma 3.4 such that $ae^\omega \in L$.

[3] This lemma was first observed by Büchi in [6, Lemma 1] where it was used to prove that nondeterministic Büchi automata are closed under complementation, without passing through a deterministic model.

So far, we are doing the same argument as in Büchi's original complementation proof from [6]. In his proof, Büchi observed that variant of (*) with $ae^\omega \notin L$, which characterises the complement of L, can be expressed by a nondeterministic Büchi automaton, and therefore nondeterministic Büchi automata are closed under complementation.

Now, we diverge from Büchi's proof, since we are interested in determinisation and not complementation. Here, semigroups will be helpful. Since it is is not clear how to express condition (*) using a deterministic Büchi automaton, we will reformulate it. This is done in the following lemma. In the lemma, we say that a pair $(a, b) \in S^2$ appears infinitely often in an ω-word $w \in \Sigma^\omega$ if for every $n \in \{1, 2, \ldots\}$ one can find a factorisation

$$w = \underbrace{x}_{\text{type } a}\underbrace{y}_{\text{type } b} z$$

such that x has length at least n.

Lemma 3.5. *An ω-word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if and only if*

(**) *there exist $a, e \in S$, with e idempotent, $ae = e$, and $ae^\omega \in L$, such that all of the following conditions are satisfied:*

(1) *(a, e) appears infinitely often; and*

(2) *if (b, c) appears infinitely often, then c is an infix of e.*

Proof The top-down implication, which says that every word accepted by $\mathcal{A}$ must satisfy (**), is an immediate consequence of Lemma 3.4. We are left with the bottom-up implication. Suppose that w satisfies (**), as witnessed by $a, e \in S$. By condition (1), there is a decomposition

$$w = w_1 v_1 w_2 v_2 w_3 v_3 \cdots$$

such that for every $i \in \{1, 2, \ldots\}$ the word v_i has type e and the word $w_1 v_1 \cdots w_i v_i$ has type a. Let a_i be the type of w_i. The ω-type of w is equal to

$$a_1 e a_2 e a_3 e \cdots,$$

which by Lemma 3.3 and idempotence of e is equal to the ω-type of

$$a_1 e \underbrace{e a_2 e}_{\substack{\text{call this}\\ g_2}} \underbrace{e a_3 e}_{\substack{\text{call this}\\ g_3}} \cdots,$$

By condition (2), there is some n such that ea_i is an infix of e for all $i \geq n$. Therefore, g_i is is an infix of e for $i \geq n$. Since g_i is begins and ends with e, it follows that g is in the $\mathcal{H}$-class of e for all $i > n$. Since this $\mathcal{H}$-class, call it G, contains the idempotent e, it must be a group by the $\mathcal{H}$-class lemma. We now have

$$\begin{aligned} \omega\text{-type of } w &= && (a_1ea_2\cdots a_{n-1}e_{n-1}=a)\\ ag_ng_{n+1}g_{n+2}\cdots &= && (\text{by Lemma 3.4, for some } g,f\in G)\\ agf^{\omega} &= && (\text{because } e \text{ is the unique idempotent in } G)\\ age^{\omega} &= && (\text{some power of } g \text{ is the idempotent } e)\\ ag^{\omega} &= && (\text{for the same reason})\\ ae^{\omega} & && \end{aligned}$$

and therefore w must belong to L. □

To finish the determinisation construction in Theorem 3.2, it remains to show that condition (**) from the above lemma is a finite Boolean combination of languages recognised by deterministic Büchi automata. This will follow from the following lemma.

Lemma 3.6. *For every $a,e\in S$ the property "(a,e) appears infinitely often" is recognised by a deterministic Büchi automaton.*

Proof Let $L\subseteq\Sigma^*$ be the set of words which can be decomposed as

$$w=\underbrace{u}_{\text{type } b}\underbrace{v}_{\text{type } e}\qquad \text{for some } b\in S \text{ such that } aeb=a.$$

This is easily seen to be a regular language, and hence it is recognised by some finite deterministic automaton $\mathcal{D}$. The deterministic Büchi automaton $\mathcal{B}$ recognising the property in the statement of the lemma is defined as follows. Its space is the disjoint union of the set of types S and the states of $\mathcal{D}$. The initial state is the type in S of the empty word. The automaton $\mathcal{B}$ begins to read input letters, keeping in its state the type of the prefix read so far in its state, until the prefix has type ae. Then it switches to the initial state q_0 of the automaton $\mathcal{D}$. For states of $\mathcal{D}$, the state update function of $\mathcal{B}$ is as follows:

$$\delta_{\mathcal{B}}(q,\sigma)\mapsto\begin{cases}\delta_{\mathcal{A}}(q,\sigma) & \text{if } q \text{ is not accepting}\\ \delta_{\mathcal{A}}(q_0,\sigma) & \text{otherwise.}\end{cases}$$

The Büchi accepting states of $\mathcal{B}$ are the same as in $\mathcal{D}$. □

This completes the proof of Theorem 3.2.

Semigroups for ω-words. There is an implicit algebraic structure in the proof of Theorem 3.2, which is formalised in the following definition.

Definition 3.7. An ω-semigroup consists consists of:

- two sets S_+ and S_ω, called the *finite sort* and the *ω-sort*, respectively.
- a finite product $\pi_+ : (S_+)^+ \to S_+$, associative as in the sense of semigroups;
- an ω-product $\pi_\omega : (S_+)^\omega \to S_\omega$, associative in the following sense:

$$\pi_\omega(w_1 w_2 \cdots) = \pi_\omega(\pi_+(w_1)\pi_+(w_2)\cdots) \qquad \text{for every } w_1, w_2, \ldots \in S^+.$$

An example of an ω-semigroup is the automaton types that were used in the proof of Theorem 3.2. Another example is the *free ω-semigroup over a set* Σ, where the finite sort is Σ^+, the ω-sort is Σ^ω, and the two products are defined in the natural way. The same proof as in Theorem 3.2 shows that a language is ω-regular if and only if it is recognised by a homomorphism into an ω-semigroup which is finite (on both sorts). This is discussed in more detail in some of the exercises at the end of this section.

The associativity axiom on the ω-product can represented using a commuting diagram, in the same spirit as for Lemma 1.4:

$$\begin{tikzcd}
((S_+)^+)^\omega \arrow[r, "\omega\text{-product in free } \omega\text{-semigroup over } S_+"] \arrow[d, "(\pi_+)^\omega"'] & (S_+)^\omega \arrow[d, "\pi_\omega"] \\
(S_+)^\omega \arrow[r, "\pi_\omega"'] & S_\omega
\end{tikzcd}$$

In the above diagram, $(\pi_+)^\omega$ denotes the coordinate-wise lifting of π_+ to ω-words of finite words.

Exercises

Exercise 50. (2) Prove the following result, called *Ramsey's Theorem A* [4] . Consider an infinite undirected graph, where every two distinct vertices are a connected by an edge that is labelled by one of finitely many colours. Then the graph contains an infinite monochromatic clique, which means that there exists a colour e and an infinite set X of vertices, such that every two distinct vertices from X are connected by an edge with colour e.

Exercise 51. (1) Prove the converse implication in Theorem 3.2.

[4] [27] Ramsey, "On a problem of formal logic", 1929 , Theorem A

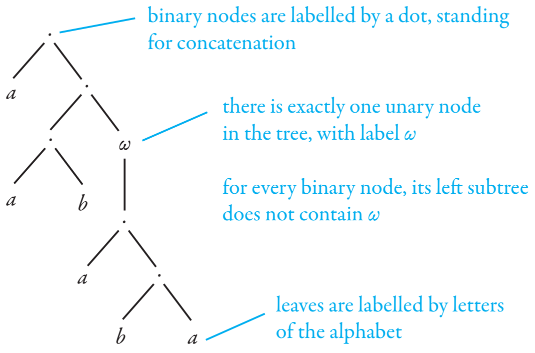
Exercise 52. (1) We say that an ω-word is *ultimately periodic* if it has the form wu^ω, for some finite words $w, u \in \Sigma^\omega$. Show that every nonempty ω-regular language contains an ultimately periodic ω-word.

Exercise 53. (1) Show that two ω-regular languages are equal if and only if they contain the same ultimately periodic ω-words.

Exercise 54. (1) Show that an ω-word w is ultimately periodic if and only if $\{w\}$ is an ω-regular language.

Exercise 55. (2) To an ω-word we associate an ordered model, in the same way as for finite words. Show that a language is MSO definable (using the ordered model) if and only if it is ω-regular.

Exercise 56. (2) Define an ω-term to be any tree as in the following picture:



binary nodes are labelled by a dot, standing for concatenation

there is exactly one unary node in the tree, with label ω

for every binary node, its left subtree does not contain ω

leaves are labelled by letters of the alphabet

Every ω-term represents some ultimately periodic ω-word, but several ω-terms might represent the same ultimately periodic ω-word. Show that two ω-terms represent the same ultimately periodic ω-word if and only if one can be transformed into the other using the equations:

$$(xy)z = x(yz) \qquad (xy)^\omega = x(yx)^\omega \qquad \underbrace{(x^n)^\omega = x^\omega}_{\text{for every } n \in \{1,2,\ldots\}}$$

where x, y, z stand for ω-terms.

Exercise 57. (1) Let $L \subseteq \Sigma^\omega$. Consider the following equivalence relations on Σ^+.

- Right equivalence is defined by

$$w \sim w' \quad \overset{\text{def}}{=} \quad wv \in L \Leftrightarrow w'v \in L \text{ for every } v \in \Sigma^\omega.$$

- Two-sided congruence is defined by

$$w \sim w' \quad \overset{\text{def}}{=} \quad uwv \in L \Leftrightarrow uw'v \in L \text{ for every } u \in \Sigma^*, v \in \Sigma^\omega.$$

- Arnold congruence is defined by

$$w \sim w' \quad \overset{\text{def}}{=} \quad \wedge \begin{cases} u(wv)^\omega \in L \Leftrightarrow u(w'v)^\omega \in L & \text{for every } u, v \in \Sigma^*. \\ uwv \in L \Leftrightarrow uw'v \in L & \text{for every } u \in \Sigma^*, v \in \Sigma^\omega. \end{cases}$$

Show that the latter two, but not necessarily the first one, are semigroup congruences, i.e. they satisfy

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

Exercise 58. (2) Consider the equivalence relations defined in Exercise 57. Prove that the arrows in the following diagram are true implications, and provide counter-examples the missing arrows:

$$\begin{tikzcd}
\begin{array}{c}\text{right}\\\text{congruence}\\\text{has finite index}\end{array} \arrow[r, leftrightarrow] & \begin{array}{c}\text{two-sided}\\\text{congruence}\\\text{has finite index}\end{array} & \begin{array}{c}\text{Arnold}\\\text{congruence}\\\text{has finite index}\end{array} \arrow[l] & \omega\text{-regular} \arrow[l]
\end{tikzcd}$$

Exercise 59. (2) Define the *Arnold semigroup* of a language $L \subseteq \Sigma^\omega$ to be the quotient of Σ^+ under Arnold congruence. Let $L \subseteq \Sigma^\omega$ be a ω-regular. Show that L is definable in first-order logic if and only if its Arnold semigroup is aperiodic.

Exercise 60. (2) The temporal logic LTL[F] can also be used to define languages of ω-words. Let $L \subseteq \Sigma^\omega$ be a ω-regular. Show that L is definable in LTL if and only if its Arnold semigroup is suffix-trivial.

Exercise 61. (1) Show an ω-regular language where the Arnold semigroup is infix trivial, but which cannot be defined by a Boolean combination of $\exists^*$-sentences.

Exercise 62. (1) Define a *safety automaton* to be an automaton on ω-words with the following acceptance condition: all states in the run are accepting.

Show that deterministic and nondeterministic safety automata recognise the same languages.

Exercise 63. (1) Show that an ω-regular language of ω-words is recognised by a safety automaton (deterministic or nondeterministic, does not matter by Exercise 62) if and only if

$$uw^! v \in L \iff u(w^!)^\omega \in L \qquad \text{for every } u, w \in \Sigma^+ \text{ and } v \in \Sigma^\omega,$$

where $! \in \{1, 2, \ldots\}$ is the exponent obtained from the Idempotent Power Lemma as applied to the Arnold semigroup of L.

Exercise 64. (2) For a finite alphabet Σ, we can view Σ^ω as metric space, where the distance between two different ω-words is defined to be

$$\frac{1}{2^{(\text{length of longest common prefix})}}$$

This is indeed a distance, i.e. it satisfies the triangle inequality. Let $L \subseteq \Sigma^\omega$ be ω-regular. Show that L is recognised by a safety automaton if and only if it is a closed set with respect to this distance.

Exercise 65. (2) Find a condition on the Arnold semigroup of an ω-regular language which characterises the clopen languages (i.e. languages which are both closed and open with respect to the distance from Exercise 64)

Exercise 66. (1) We use the topology from Exercise 64. Define a G_δ set to be any countable intersection of open sets. Show that every ω-regular language is a finite Boolean combination of G_δ sets.

Exercise 67. (2) Let $L \subseteq \Sigma^\omega$ be an ω-regular language, and define ! as in Exercise 62. Show that L is recognised by a deterministic Büchi automaton if and only if:

$$u(wv^!)^! v^\omega \in L \implies u(wv^!)^\omega \in L \qquad \text{for every } u, w, v \in \Sigma^+.$$

Exercise 68. (2) Let $L \subseteq \Sigma^\omega$. Define an ω-congruence to be any equivalence relation $\sim$ on Σ^+ which is a semigroup congruence and which satisfies

$$\bigwedge_{i \in \{1,2,\ldots\}} w_i \sim w'_i \qquad \text{implies} \qquad w_1 w_2 \cdots \in L \Leftrightarrow w'_1 w'_2 \cdots \in L. \tag{3.1}$$

Show that a language is ω-regular if and only if it has an ω-congruence of finite index.

Exercise 69. (2) Define *semi-ω-congruence* for a language $L \subseteq \Sigma^\omega$ to be an equivalence relation on finite words which satisfies (3.1), but which is not necessarily a semigroup congruence. Show that if there is a semi-ω-congruence of finite index, then there is an ω-congruence of finite index.

Exercise 70. (2) We say that $\sim$ is the *syntactic ω-congruence* of $L \subseteq \Sigma^\omega$ if it is an ω-congruence, and every other ω-congruence for L refines $\sim$. Show that if a language is ω-regular, then it has a syntactic ω-congruence, which is equal to the Arnold congruence.

Exercise 71. (2) Show a language of ω-words which does not have a syntactic ω-congruence.

3.2 Countable words and $\circ$-semigroups

In this section, we move to $\circ$-words. These are words where the set positions is a countable linear order. The positions could be some finite linear order, as in finite words, or the natural numbers, as in ω-words, but the positions could also be dense, as in the rational numbers. One advantage of $\circ$-words, as compared to ω-words, is that they can be concatenated, which is useful when defining the corresponding generalisation of semigroups.

For finite words, as well as for ω-words, the approach via semigroups can be seen as an alternative to existing automata models. This is no longer the case for $\circ$-words. There is no known corresponding automaton model, and therefore $\circ$-semigroups are the only known model of recognisability.

Definition 3.8 ($\circ$-words)**.** A *Σ-labelled linear order* consists of a set X of *positions*, equipped with a total order and a labelling of type $X \to \Sigma$. Two such objects are considered *isomorphic* if there is a bijection between their positions, which preserves the order and labelling. Define a *$\circ$-word over* Σ to be any isomorphism class of countable[5] Σ-labelled linear orders. We write Σ° for the set of ∘-words[6].

Every finite word is a ∘-word, likewise for every ω-word. Another example is labelled countable ordinals, e.g. any ∘-word where the positions are $\omega + \omega$. Below is a more fancy example, which uses a dense set of positions.

Example 3.9 (Shuffles)**.** A classical exercise on linear orders is that the rational numbers are the unique – up to isomorphism – countable linear order which is dense and has no endpoints (i.e. neither a least nor greatest element). This is proved by constructing, using the back-and-forth method, an isomorphism between any two such orders. The same argument shows that for every countable Σ there is a ∘-word over Σ which has no endpoints, and which satisfies

$$\bigwedge_{a\in\Sigma} \underbrace{\forall x\, \forall y\, \exists z \quad x < z < y \wedge a(z)}_{\text{label } a \text{ is dense}}.$$

We use the name *shuffle of* Σ for the above ∘-word. Shuffles will play an important role in semigroups for ∘-words.

We now define the generalisation of semigroups for ∘-words. We use the approach to associativity via commuting diagrams that was described in Lemma 1.4. Recall from that lemma that a semigroup product on a set S could be defined as any operation π which makes the following diagram commute:

$$\begin{tikzcd}
S \arrow[d, "\text{view a letter as a one-letter word}"'] \arrow[dr, "\text{identity}"] & \\
S^+ \arrow[r, "\pi"'] & S
\end{tikzcd}
\qquad
\begin{tikzcd}
(S^+)^+ \arrow[r, "\text{flattening}"] \arrow[d, "\pi^+"'] & S^+ \arrow[d, "\pi"] \\
S^+ \arrow[r, "\pi"'] & S
\end{tikzcd}$$

For ∘-semigroups, we take the same approach. For a set S, the flattening operation $(S^\circ)^\circ \to S^\circ$ is defined in the natural way, by replacing each position with the ∘-word that is in its label (a formal definition uses a lexicographic product of labelled linear orders).

[5] The reader might wonder why we assume countability. The reason is that the decidability theory that will be described in this section breaks down for uncountable
linear orders. In fact, the mso theory of the order of real numbers $(\mathbb{R}, <)$ is undecidable, as shown
[33] Shelah, "The Monadic Theory of Order", 1975 , Theorem 7
The description of ∘-semigroups in this section is based on
[8] Carton, Colcombet, and Puppis, "An algebraic approach to MSO-definability on countable linear orders", 2018
which itself is based on Shelah's approach to countable linear orders from [33].

[6] Formally speaking, this is not a set, because the linear orders form a class an not a set. However, without loss of generality we can use some fixed countably infinite set, e.g. the natural numbers, for the positions (but the order need not be the same as in the natural numbers). Under this restriction, the labelled linear orders become a set, and no isomorphism types are lost. For this reason, we can refer to Σ° as a set.

Definition 3.10. A $\circ$*-semigroup* consists of an underlying set S equipped with a product operation $\pi : S^\circ \to S$, which is associative in the sense that the following two diagrams commute:

$$\begin{tikzcd}
S \arrow[d, "\text{view a letter as}\atop\text{a one-letter }\circ\text{-word}"'] \arrow[dr, "\text{identity}"] & \\
S^\circ \arrow[r, "\pi"'] & S
\end{tikzcd}
\qquad
\begin{tikzcd}
(S^\circ)^\circ \arrow[r, "\text{flattening}"] \arrow[d, "\pi^\circ"'] & S^\circ \arrow[d, "\pi"] \\
S^\circ \arrow[r, "\pi"'] & S
\end{tikzcd}$$

In the above diagram, π° denotes the coordinate-wise lifting of π to $\circ$-words of $\circ$-words.

Example 9. The *free* $\circ$*-semigroup* over alphabet Σ has Σ° as its underlying set, and its product operation is flattening. To check that this product operation is associative, one needs to prove that the following diagram commutes:

$$\begin{tikzcd}
((\Sigma^\circ)^\circ)^\circ \arrow[r, "\text{flattening for alphabet }\Sigma^\circ"] \arrow[d, "(\text{flattening for alphabet }\Sigma)^\circ"'] & (\Sigma^\circ)^\circ \arrow[d, "\text{flattening for alphabet }\Sigma"] \\
(\Sigma^\circ)^\circ \arrow[r, "\text{flattening for alphabet }\Sigma"'] & \Sigma^\circ
\end{tikzcd}$$

To prove this formally, one uses the formal definition of flattening, in terms of lexicographic products of linear orders (see Example 19). This $\circ$-semigroup is called *free* for the usual reasons; a more formal description of these usual reasons will appear later in the book, when discussing monads. □

Example 10. Recall the semigroups of size two that were discussed in Example 1.2:

$$\underbrace{(\{0,1\},+)}_{\text{addition mod 2}} \quad (\{0,1\},\min) \quad \underbrace{(\{0,1\},\pi_1)}_{\text{product } ab \text{ is } a} \quad \underbrace{(\{0,1\},\pi_2)}_{\text{product } ab \text{ is } b} \quad \underbrace{(\{0,1\},(a,b)\mapsto 1)}_{\text{all products are 1}}$$

Which ones can be extended to $\circ$-semigroups in at least one way?

The first example, i.e. the two-element group, cannot be extended in any way, because the product a of the ω-word 1^ω would need satisfy

$$a = \pi(1^\omega) = \pi(\pi(1)\pi(1^\omega)) = \pi(1a) = 1 + a.$$

The remaining semigroups can be extended to $\circ$-semigroups. As we will see in Example 11, the extensions are not necessarily unique. □

We use $\circ$-semigroups to recognise languages of $\circ$-words. Define a *homomorphism of* $\circ$*-semigroups* to be a function h which makes the following diagram commute:

$$\begin{tikzcd} S^\circ \arrow[r, "h^\circ"] \arrow[d, "\text{product in } S"'] & T^\circ \arrow[d, "\text{product in } T"] \\ S \arrow[r, "h"'] & T \end{tikzcd}$$

Like for semigroups, homomorphisms of ∘-semigroup can be described in terms of compositional functions. Suppose that S is a ∘-semigroup and T is a set. We say that a function $h : S \to T$ is *compositional* if there exists a function $\pi : T^\circ \to T$ which makes the following diagram commute

$$\begin{tikzcd} S^\circ \arrow[r, "h^\circ"] \arrow[d, "\text{product in } S"'] & T^\circ \arrow[d, "\pi"] \\ S \arrow[r, "h"'] & T \end{tikzcd}$$

Using the same proof as for Lemma 1.3, one shows that if h is a compositional and surjective, then π is necessarily associative, thus turning T into a ∘-semigroup, and furthermore h is a homomorphism.

We say that a language $L \subseteq \Sigma^\circ$ is *recognised* by a ∘-semigroup S if there is a homomorphism $h : \Sigma^\circ \to S$ which recognises it, i.e.

$$h(w) = h(w') \quad \text{implies} \quad w \in L \Leftrightarrow w' \in L \qquad \text{for every } w, w' \in L.$$

We are mainly interested in languages recognised by finite ∘-semigroups.

Example 11. Consider un-labelled countable linear orders, which can be viewed as ∘-words over a one letter alphabet $\{a\}$. Consider the function

$$h : \{a\}^\circ \to \{0, 1\}$$

which sends well-founded ∘-words to 1, and the remaining ∘-words to 0. We claim that h compositional (and therefore the language of well-founded ∘-words is recognised by a finite ∘-semigroup). Indeed, take some $v \in (\{a\}^\circ)^\circ$ which flattens to $w \in \{a\}^\circ$. To prove compositionality, need to show that $h^\circ(v)$ uniquely determines $h(w)$. This is because $h(w) = 1$ if and only if the positions of v are well-founded, and every such a position is labelled by a well-founded order. All of this information can be recovered from $h^\circ(v)$. The compositional function h induces an underlying structure of a ∘-semigroup on $\{0, 1\}$. When restricted to finite products, this ∘-semigroup is the same as $(\{0, 1\}, \min)$. Note that a symmetric ∘-semigroup can be constructed, for orders which are well-founded after reversing. The symmetric ∘-semigroup also coincides with $(\{0, 1\}, \min)$ on finite words. □

Example 12. Consider the language $L \subseteq \{a, b, 1\}^\circ$, which contains $\circ$-words where some position with label a is to the left of some position with label b. Consider the following function

$$w \in \{a,b,1\}^\circ \quad \mapsto \quad \begin{cases} 0 & \text{if } w \in L \\ 1 & \text{if all letters are 1} \\ b & \text{if all letters are } b \text{ or 1, and there is some } b \\ ba & \text{if both } b \text{ and } a \text{ appear, but } w \notin L \\ a & \text{otherwise} \end{cases}$$

This function is easily seen to be compositional, and therefore its image is a $\circ$-semigroup. The element 0 is absorbing, and 1 is a monoid identity. The language L is therefore recognised by the corresponding $\circ$-semigroup. □

Monadic second-order logic on $\circ$-words. The *ordered model* of a $\circ$-word is defined in the same way as for finite words: the universe is the positions, and the relations and their meaning are the same as for finite words. We say that a language $L \subseteq \Sigma^\circ$ is definable in MSO if there is an MSO sentence φ, using the vocabulary of the ordered model, such that

$$w \in L \quad \Leftrightarrow \quad \text{the ordered model of } w \text{ satisfies } \varphi \qquad \text{for every } w \in \Sigma^\circ.$$

Example 13. Consider the language of well-founded $\circ$-words that was discussed in Example 11. This language is definable in MSO, by simply writing in MSO the definition of well-foundedness:

$$\underbrace{\forall X}_{\substack{\text{for every}\\ \text{set of}\\ \text{positions}}} \qquad \underbrace{(\exists x \in X)}_{\text{which is nonempty}} \Rightarrow \underbrace{(\exists x \in X\ \forall y \in X\ x \leq y)))}_{\text{there is a least position}}.$$

Another example is the $\circ$-words which contain a sub-order that is dense:

$$\underbrace{\exists X}_{\substack{\text{exists a}\\ \text{set of}\\ \text{positions}}} \qquad \underbrace{(\exists x \in X)}_{\text{which is nonempty}} \wedge \underbrace{(\forall x \in X\ \forall y \in Y\ x < y \Rightarrow\ \exists z \in X\ x < z < y)))}_{\text{and dense in itself}}.$$

An $\circ$-word which violates the second property, i.e. it does not have any dense sub-order, is called *scattered*. □

Once we have built up all the necessary ideas in the Trakhtenbrot-Büchi-Elgot Theorem for finite words, it is very easy to get the extension for $\circ$-words.

Lemma 3.11. *If a language $L \subseteq \Sigma^\circ$ is definable in* MSO*, then it is recognised by a finite* $\circ$*-semigroup.*

Proof We only give a brief sketch using Ehrenfeucht-Fraïssé games – a more detailed proof using a powerset construction on $\circ$-semigroups will be given later in Lemma 3.22. We use the same argument as in Section 2.1. The set model is defined the same way as for finite words – its universe is the subsets of the positions. For $k \in \{0, 1, \ldots\}$, define $\equiv_k$ to be the equivalence relation on Σ° which identifies $\circ$-words if their set models satisfy the same first-order sentences of quantifier rank at most k. Using the same proof as for Lemma 2.5, except that now a $\circ$-word can be divided into more than two parts, one shows that

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

is a compositional function into a finite set. Therefore, the function is a homomorphism of $\circ$-semigroups. Every language definable in MSO is recognised by such a homomorphism, for suitably chosen k. □

The above lemma seems too easy. Is there a catch? Yes: the lemma does not give any algorithm for deciding if an MSO definable language is empty, or any answering any other computational problems. In fact, the lemma would remain true for uncountable words, and satisfiability of MSO sentences for such words is undecidable. The issue of finite representation for $\circ$-semigroups will be addressed in the following section; and countability will play a crucial role.

Another interesting question is about the converse of the lemma: can one define in MSO every language that is recognised by a finite $\circ$-semigroup? For finite words and ω-words, the answer was "obviously yes", because one can use MSO to formalise the acceptance by an automaton. Since we have no automata for $\circ$-words, the question is harder. However, the answer is still "yes", and it will be given in Section 3.4.

Exercises

Exercise 72. (1) Give a formula of MSO which is true in some uncountable well-founded linear order, but is false in all countable well-founded linear orders.

Exercise 73. (1) Find two countable ordinals (viewed as $\circ$-words over a one letter alphabet), which have the same MSO theory.

Exercise 74. (1) We write $\omega*$ for the reverse of ω. An $(\omega* + \omega)$-word is a $\circ$-word where the underlying order is the same as for the integers. Show that the following problem is decidable: given an MSO sentence, decide if it is true in some bi-infinite word.

Exercise 75. (2) We say that a $(\omega* + \omega)$-word v is *recurrent* if every finite word $w \in \Sigma^+$ appears as an infix in every prefix of v and in every suffix of v. Show that all recurrent $(\omega* + \omega)$-words have the same MSO theory.

Exercise 76. (2) Let Σ be an alphabet, and let $x \notin \Sigma$ be a fresh letter. For $w \in \Sigma^\circ$ and $u \in (\Sigma \cup \{x\})^\circ$, define $u[x := w] \in \Sigma^\circ$ to be the result of substituting each occurrence of variable x in u by the argument w. For a language $L \subseteq \Sigma^\circ$, define *contextual equivalence* to be the equivalence relation on Σ° defined by

$$w \sim w' \quad \text{iff} \quad u[x := w] \in L \Leftrightarrow u[x := w'] \in L \text{ for every } u \in (\Sigma \cup \{x\})^\circ.$$

Show that $\sim$ is a $\circ$-congruence (which means that the function that maps w to its equivalence class is compositional) for every language recognised by some finite $\circ$-semigroup.

Exercise 77. (1) Give an example of a language $L \subseteq \Sigma^\circ$ where contextual equivalence is not a $\circ$-congruence.

Exercise 78. (1) Show that every language recognised by a finite $\circ$-semigroup has syntactic $\circ$-semigroup, but there are some languages (not recognised by finite $\circ$-semigroups), which do not have a syntactic $\circ$-semigroup.

Exercise 79. (1) Consider a binary tree (every node has either zero or two children, and we distinguish left and right children), where leaves are labelled by an alphabet Σ. The tree might have infinite branches. Define the *yield* of such a tree to be the $\circ$-word where the positions are leaves of the tree, the labels are inherited from the tree, and the ordering on leaves is lexicographic (for every node, its left subtree is before its right subtree). Show that every $\circ$-word can be obtained as the yield of some tree.

Exercise 80. (1) Show that the following problems are equi-decidable:

- given an MSO sentence, decide if it is true in some $\circ$-word $w \in \Sigma^\circ$

- given an MSO sentence, decide if its true in $(\mathbb{Q}, <)$.

Exercise 81. (1) Assume Rabin's Theorem, which says that the MSO theory of the complete binary tree

$$(\{0,1\}^*, \underbrace{x = y0}_{\substack{\text{left}\\ \text{child}}}, \underbrace{x = y1}_{\substack{\text{right}\\ \text{child}}})$$

is decidable. Show that the problems from Exercise 80 are decidable. (We will also prove this in the next section, without assuming Rabin's theorem.)

3.3 Finite representation of ∘-semigroups

The product operation in a finite semigroup can be seen as an operation of type $S^+ \to S$, or as a binary operation of type $S^2 \to S$. The binary operation has the advantage that a finite semigroup can be represented in a finite way, by giving a multiplication table. In this section, we show that a similar finite representation is also possible for ∘-semigroups. Apart from binary product, we will use two types of ω-iteration – one forward and one backward – and a shuffle operation (which inputs a set of elements, and not a tuple of fixed length).

Definition 3.12 (Läuchli-Leonard operations)**.** For a ∘-semigroup, define its *Läuchli-Leonard operations*[7] to be the following four operations (with their types written in red).

$$\underbrace{ab}_{\substack{\text{binary}\\ \text{product}\\ S^2 \to S}} \qquad \underbrace{a^\omega}_{\substack{\text{product of}\\ aaa\cdots\\ S \to S}} \qquad \underbrace{a^{\omega *}}_{\substack{\text{product of}\\ \cdots aaa\\ S \to S}} \qquad \underbrace{\{a_1, \ldots, a_n\}^\eta}_{\substack{\text{product of the}\\ \text{shuffle of } a_1, \ldots, a_n\\ \mathsf{P}S \to S}}$$

Theorem 3.13. *The product operation in a finite ∘-semigroup is uniquely determined by its Läuchli-Leonard operations.*

Another way of stating the theorem is that if S is a finite set S equipped with Läuchli-Leonard operations, then there is at most one way of extending these operations to an associative product $S^\circ \to S$. We say at most one instead of exactly one, because the Läuchli-Leonard operations need to satisfy certain associativity laws, such as:

$$aa^\omega = a^\omega \qquad (ab)^\omega = a(ba)^\omega \qquad \{a_1, \ldots, a_n\}^\eta = \{\{a_1, \ldots, a_n\}^\eta\}^\eta$$

[7] [22] Läuchli and Leonard, "On the elementary theory of linear order", 1966 , p. 109

We do not worry too much about giving the full set of axioms[8], because we will only consider product operations that arise from compositional functions – e.g. the product operation on MSO types of given quantifier rank k – and such product operations are guaranteed to be associative.

3.3.1 Proof of Theorem 3.13

The key idea in the proof of Theorem 3.13 is that the Läuchli-Leonard operations are enough to generate all sub-algebras, as stated in the following lemma.

Lemma 3.14. *Let S be a $\circ$-semigroup, and let $\Sigma \subseteq S$. Then*

$$\underbrace{\{\text{product of } w : w \in \Sigma^\circ\}}_{\textit{this is called the } \text{sub-algebra generated by } \Sigma} \subseteq S$$

is equal to the smallest subset of S which contains Σ and is closed under the Läuchli-Leonard operations.

Before proving the lemma, we use it to prove Theorem 3.13.

Proof of Theorem 3.13, assuming Lemma 3.14. Suppose that S_1 and S_2 are two $\circ$-semigroups, which have the same underlying set, and product operations which agree on the Läuchli-Leonard operations. We will show that the product operations are the same. Consider the product $\circ$-semigroup $S_1 \times S_2$, defined in the usual coordinate-wise way. Apply Lemma 3.14 to the diagonal

$$\Sigma = \{(a, a) : a \in S\} \subseteq S_1 \times S_2.$$

Since the Läuchli-Leonard operations agree for S_1 and S_2, it follows from the lemma that the sub-algebra generated by Σ is also on the diagonal, which shows that the product operations of S_1 and S_2 are equal. □

The rest of Section 3.3.1 is devoted to proving Lemma 3.14. Define $L \subseteq \Sigma^\circ$ to be the $\circ$-words whose product can be obtained from Σ by applying the Läuchli-Leonard operations. The lemma says that $L = \Sigma^\circ$. This follows immediately from the following lemma (which is stated slightly more generally, because it will be used again later), by taking λ to be the product operation of the $\circ$-semigroup.

Lemma 3.15. *Assume that $L \subseteq \Sigma^\circ$ contains Σ and is closed under binary concatenation. A sufficient condition for $L = \Sigma^\circ$ is that there exists a colouring $\lambda : \Sigma^\circ \to C$, with C a finite set of colours, such that:*

[8] It can be found in
[3] Bloom and Ésik, "The equational theory of regular words", 2005 , Section 7

(*) *Let $w \in (\Sigma^\circ)^\circ$ be such that every position has label in L, and*

$$\underbrace{\lambda^\circ(w) = c^\omega}_{\text{for some } c \in C} \quad \text{or} \quad \underbrace{\lambda^\circ(w) = c^{\omega*}}_{\text{for some } c \in C} \quad \text{or} \quad \underbrace{\lambda^\circ(w) = \text{shuffle of } D.}_{\text{for some } D \subseteq C}$$

Then the flattening of w belongs to L.

Before proving the lemma, we fix some notation for ∘-words. Define an *interval* in a ∘-word to be any set of positions X such that is connected in the following sense:

$$\forall x \in X \; \forall y \in Y \; \forall z \quad x < z < y \Rightarrow z \in X.$$

An infix of a ∘-word is any ∘-word obtained by restricting the positions to some interval. For example, the rationals – viewed as a ∘-word over a one letter alphabet – have uncountably many intervals, but five possible infixes. If $x < y$ are positions in a ∘-word w, then we write $w(x..y)$ for the infix corresponding to the interval $\{z : x < z < y\}$.

Proof Suppose then that L satisfies (*), as witnessed by a colouring λ. We say that $w \in \Sigma^\circ$ is *simple* if all of its infixes are in L. We will show that all of Σ° is simple, thus proving $L = \Sigma^\circ$. For the sake of contradiction, suppose that $w \in \Sigma^\circ$ is not simple. Define $\sim$ to be the binary relation on positions in w, which identifies positions if they are equal, or $w(x..y)$ is simple (where x is the smaller position and y is the bigger position).

Claim 3.16. *The relation $\sim$ is an equivalence relation, every equivalence class is an interval, and this interval induces a simple ∘-word.*

Proof The relation $\sim$ is symmetric and reflexive by definition. Transitivity is because simple words are closed under binary concatenation (which itself easily follows from the fact that L contains all letters and is closed under binary concatenation). This establishes that $\sim$ is an equivalence relation. Because simple ∘-words are closed under infixes by definition, every equivalence class of $\sim$ is an interval.

It remains to show that every (infix induced by an) equivalence class is simple. Here we use countability and the assumption (*). Consider an equivalence class X. Choose some $x \in X$. We will show that the suffix of X that begins in x is simple. A symmetric argument will establish that the prefix leading up to x is simple, and thus X itself is simple by binary concatenation.

If X has a last position, then the suffix that begins in x is simple by definition of $\sim$. Suppose then that there is no last position in X. By countability, chose some sequence of positions

$$x = x_0 < x_1 < x_2 < \ldots \in X$$

which is co-final, i.e. every position in X is smaller than some x_i. By the Ramsey Theorem, we can assume without loss of generality that there is some $c \in C$ such that for every $i \in \{1, 2, \ldots\}$, the infix obtained from $w(x_i..x_{i+1})$ by appending position x_{i+1} has colour c under λ. Furthermore, this infix is simple by definition of $\sim$. Therefore, thanks to assumption (*), we know that the concatenation of all of these infixes is simple. □

Since the equivalence classes of $\sim$ are intervals, they can be viewed as an ordered set, with the order inherited from the original order on positions in w. Because simple words are closed under binary concatenation, the order on equivalence classes is dense, since otherwise two consecutive equivalence classes would need to be merged into a single one. Define $w_\sim \in C^\circ$ to be the result of replacing every equivalence class of $\sim$ by its colour under λ. By assumption that w is not simple, $\sim$ has more than one equivalence class, and therefore the positions of $w_\sim$ are an infinite dense linear order.

Claim 3.17. *Some infix of $w_\sim$ is a shuffle.*

Proof Take some colour $c \in C$. If there is some infinite infix of $w_\sim$ where c does not appear at all, then we can continue working in that infix (its positions are still an infinite dense linear order). Otherwise, positions with colour c are dense. By iterating this argument for all finitely many colours in C, we find an infinite infix where every colour either does not appear at all, or is dense. This infix is a shuffle. □

By (*), the flattening of the infix from the above claim is simple. It follows that the corresponding interval should have been a single equivalence class of $\sim$, contradicting the assumption. □

3.3.2 Decidability of MSO

By Theorem 3.13, a finite $\circ$-semigroup can be represented in a finite way, by giving its underlying set and the multiplication tables for its Läuchli-Leonard operations. We will use this representation to give decision procedure for MSO on $\circ$-words.

A powerset construction. To decide MSO, we will use a powerset construction for finite $\circ$-semigroups, which will correspond to set quantification in MSO. We begin by describing this construction.

Definition 3.18. For a ∘-semigroup S, define the *powerset* ∘*-semigroup* $\mathsf{P}S$ as follows. The underlying set of $\mathsf{P}S$ is the powerset of the underlying set of S, including the empty set[9]. The product operation is defined by

$$w \in (\mathsf{P}S)^\circ \quad \mapsto \quad \{\underbrace{\text{product of } v}_{\text{in } S} : v \in^\circ w\},$$

where $v \in^\circ w$ means that $v \in S^\circ$ can be obtained from $w \in (\mathsf{P}S)^\circ$ by choosing for each position an element of its label[10]. We leave it as an exercise to check that the product operation defined this way is associative[11].

The following lemma shows that the powerset construction is computable. What is not obvious is finding the multiplication tables for the Läuchli-Leonard operations.

Lemma 3.19. *Given the multiplication tables for the Läuchli-Leonard operations in a finite* ∘*-semigroup S, one can compute the multiplication tables for the Läuchli-Leonard operations in the powerset* ∘*-semigroup* $\mathsf{P}S$.

Proof In the proof, we adopt the convention that elements of S are denoted by lower-case letters a, b, c, while elements of $\mathsf{P}S$ are denoted by upper-case letters A, B, C. The multiplication table for binary product of $\mathsf{P}S$, namely

$$A, B \in \mathsf{P}S \quad \mapsto \quad \{\underbrace{ab}_{\text{product in } S} : a \in A, b \in B\},$$

is easily computable using the binary product in S. The hard part is the multiplication tables for the infinitary operations, i.e. ω-power, $\omega*$-power and shuffles.

ω-power. We begin by clarifying some notation. For a $A \in \mathsf{P}S$, the expression A^ω can be understood in three different ways:

(1) $A^\omega \subseteq S^\circ$ is the set of ω-words where all letters are from A;
(2) $A^\omega \in (\mathsf{P}S)^\circ$ is the ω-word where all letters have label equal to A;
(3) $A^\omega \in \mathsf{P}S$ is the product of the word from item (2) in the ∘-semigroup $\mathsf{P}S$.

[9] Whether or not we allow the empty set is not important for the construction.

[10] The relation $v \in^\circ w$ can be formalised by saying that there exists a ∘-word u over alphabet

$$\{(a, A) : a \in A \subseteq S\}$$

such that v is the projection of u to the first coordinate, and w is the projection of u to the second coordinate.

[11] One has to a bit careful. For example, there is no such thing as a powerset group.

To avoid confusion, we use the red type annotation below. The Läuchli-Leonard operation in the powerset $\circ$-semigroup $\mathsf{P}S$ that we are discussing in this lemma uses the third meaning of A^ω:

$$A \in \mathsf{P}T \quad \mapsto \quad A^\omega \in \mathsf{P}S.$$

To compute this operation, we will use, apart from S, one other $\circ$-semigroup. This other $\circ$-semigroup, call it T, is used to recognise the singleton language

$$\{A^\omega \in (\mathsf{P}S)^\circ\} \subseteq (\mathsf{P}S)^\circ.$$

The elements of T are $\{\omega, +, 0\}$ and the product operation is the unique product which makes the following function h into a homomorphism:

$$w \in (\mathsf{P}S)^\circ \mapsto \begin{cases} \omega & \text{if } w = A^\omega \in (\mathsf{P}S)^\circ \\ + & \text{if } v \text{ is a finite word and all letters have label } A \\ 0 & \text{otherwise} \end{cases}$$

For the Läuchli-Leonard operations in T, all outputs are 0 with the following exceptions:

$$++ = + \qquad + \omega = \omega \qquad +^\omega = \omega.$$

Claim 3.20. *Let $a \in S$. Then $a \in A^\omega \in \mathsf{P}S$ if and only if (a, ω) belongs to the sub-algebra of the $\circ$-semigroup $S \times T$ that is generated by $\{(b, +) : b \in A\}$.*

Proof By unravelling the definitions, (a, ω) belongs to the sub-algebra from the claim if and only if it is the product of some $\circ$-word u where every letter is of the form $(b, +)$ for some $b \in A$. Since the product of u has ω on the second coordinate, then u must be an ω-word. Therefore, if we project u onto the first coordinate, we get a word in $A^\omega \subseteq S^\circ$ whose product is a, thus proving the right-to-left implication. The left-to-right implication reverses this reasoning. □

By the above claim, in order to compute $A^\omega \in \mathsf{P}S$, it is enough to compute the sub-algebra from the claim. Thanks to Lemma 3.14, this sub-algebra is the closure of $\{(b, +) : b \in A\}$ under the Läuchli-Leonard operations of $S \times T$, which are simply the coordinate-wise liftings of the Läuchli-Leonard operations in S and T. Therefore, $A^\omega \in \mathsf{P}S$ can be computed.

Shuffle power. The argument for $\omega*$-power is symmetric to the one above, so we are left with the shuffle power in the $\circ$-semigroup $\mathsf{P}S$:

$$\{A_1, \dots, A_n\} \quad \mapsto \quad \{A_1, \dots, A_n\}^\eta$$

We use a similar argument as for the ω-power, except that we need to define a ∘-semigroup which describes the singleton language $\{w\}$ where

$$w \overset{\text{def}}{=} \underbrace{\text{shuffle of } \{A_1, \ldots, A_n\}}_{\text{a } \circ\text{-word over alphabet } \mathsf{P}S}.$$

Such a ∘-semigroup T and a recognising homomorphism

$$h : (\mathsf{P}S)^\circ \to T$$

are left as an exercise for the reader (see Exercise 82). By definition of the powerset ∘-semigroup,

$$\{A_1, \ldots, A_n\}^\eta = \{\underbrace{\text{product of } v}_{\text{in } S} : v \in^\circ w\}.$$

Let $a \in S$. The same proof as for Claim 3.20 shows that

$$a \in \{A_1, \ldots, A_n\}^\eta$$

holds if and only if $(a, h(w))$ belongs to the sub-algebra of $S \times T$ that is generated by elements of the form

$$\{(b, h(A_i)) : i \in \{1, \ldots, n\}, b \in A_i\}.$$

This sub-algebra can be computed, as in the case of ω-power. □

Decidability of MSO. Using the powerset construction on ∘-semigroups, we can now prove decidability of MSO over ∘-words.

Theorem 3.21. *The following problem is decidable:*

Input. *An* MSO *sentence φ, which defines a language $L \subseteq \Sigma^\circ$.*
Question. *Is the language L nonempty?*

The rest of Section 3.3.2 is devoted to proving the above theorem. As in Section 2.1, instead of using MSO in the ordered model, it will be easier to use first-order logic over (the extension for ∘-words of) the set model, see Definition 2.3. By induction on formula size, for every first-order formula over the set model we will construct a homomorphism into a finite ∘-semigroup that recognises the language of the formula. The finite ∘-semigroup will be represented, according to Theorem 3.13, by giving the underlying set and the multiplication tables for its Läuchli-Leonard operations.

For the induction, we need to deal with formulas with free variables. Consider a first-order formula

$$\varphi(\underbrace{x_1, \ldots, x_n}_{\substack{\text{the free variables range} \\ \text{over sets of positons}}})$$

over the vocabulary of the set model (over some fixed input alphabet Σ.) We define the *language of* φ to be the set of $\circ$-words over alphabet $\Sigma \times \{0, 1\}^n$, such that φ is true in the projection to the Σ coordinate, assuming that variable x_i is set to the set of positions that have 1 on the i-th coordinate from $\{0, 1\}$.

Lemma 3.22. *Let Σ be an input alphabet. Given a formula $\varphi(x_1, \ldots, x_n)$ of first-order logic over the vocabulary of the set model, we can compute a finite $\circ$-semigroup S, a (not necessarily surjective) homomorphism*

$$h : (\Sigma \times \{0, 1\}^n)^\circ \to S,$$

and an accepting set $F \subseteq S$ such that the language of φ is exactly $h^{-1}(F)$. The $\circ$-semigroup is represented by multiplication tables for its Läuchli-Leonard operations, and the homomorphism is represented by its restriction to one-letter words.

Once we have proved the lemma, Theorem 3.21 follows immediately. We simply need to check if the image of h contains some accepting element. There is a slightly subtle point: since h is not necessarily surjective, the accepting set could be nonempty but disjoint with the image of h. Therefore, we need to compute the image of h, which is done by closing the images of the one-letter words under the Läuchli-Leonard operations.

It remains to prove the lemma.

Proof of Lemma 3.22 Induction on the size of $\varphi(x_1, \ldots, x_n)$.

Atomic formulas. The atomic formulas are $x \subseteq y$, $x < y$, $x = \emptyset$ and $x \subseteq a$. With the exception of $x < y$, all of the atomic formulas are of the form "the label of every position has some property". Therefore, for every atomic formula except for $x < y$, the corresponding language is recognised by a homomorphism into the semigroup $\{0, 1\}$ with product defined by

$$w \in \{0, 1\}^\circ \quad \mapsto \quad \begin{cases} 0 & \text{if some position has label 0} \\ 1 & \text{if all positions have label 1.} \end{cases}$$

For $x < y$, the appropriate $\circ$-semigroup is the one from Example 12.

Boolean combinations. For negation $\neg\varphi$, we use the same homomorphism as for φ, and we complement the accepting set. For conjunction $\varphi_1 \wedge \varphi_2$ and disjunction $\varphi_1 \wedge \varphi_2$, we use the product $S_1 \times S_2$ of the inductively obtained ∘-semigroups, with a naturally defined homomorphism[12]. Since $S_1 \times S_2$ is defined coordinate-wise, the multiplication tables for its Läuchli-Leonard operations can be computed using those from S_1 and S_2.

Quantification. We are left with quantification. Consider an existentially quantified formula (for universal quantification, the reasoning is the same):

$$\exists x_{n+1}\ \varphi(x_1, \ldots, x_{n+1}).$$

Apply the induction assumption, yielding a homomorphism

$$h : (\Sigma \times \{0,1\}^{n+1})^\circ \to S$$

which recognises the language of φ. Consider the function

$$H : (\Sigma \times \{0,1\}^{n})^\circ \to \underbrace{\mathsf{P}S}_{\text{powerset ∘-semigroup}},$$

such that $H(w)$ is the set of all values $h(v) \in S$, where v ranges over ∘-words over alphabet $\Sigma \times \{0,1\}^{n+1}$ such that w can be obtained from v by erasing the last bit from each letter. It is not hard to see that H is a homomorphism. The homomorphism H recognises the language of the existentially quantified formula; the accepting set consists of those subsets of S which intersect the accepting set for φ. The multiplication tables for the Läuchli-Leonard operations in $\mathsf{P}S$ can be computed thanks to Lemma 3.19.

□

Exercises

Exercise 82. (1) Let Σ be a finite alphabet, and let w be the shuffle of all letters in Σ. Show a finite ∘-semigroup which recognises the singleton language $\{w\}$.

Exercise 83. (2) A ∘-word w is called *regular* if the singleton language $\{w\}$ is recognised by a finite ∘-semigroup. Show that w is regular if and only if it can be constructed from the letters by using the Läuchli-Leonard operations.

[12] The homomorphisms needs to account for the possibility that φ_1 and φ_2 use different subsets of the free variables in $\varphi_1 \wedge \varphi_2$.

Exercise 84. (1) Show that every nonempty MSO definable language $L \subseteq \Sigma^\circ$ contains some regular $\circ$-word.

Exercise 85. (1) Show that if w is a regular $\circ$-word, then $\{w\}$ is MSO definable (without invoking Theorem 3.23).

Exercise 86. (2) Show that for every finite alphabet Σ there exists a $\circ$-word $w \in \Sigma^\circ$ such that

$$h(wvw) = h(w) \qquad \text{for every } \underbrace{h : \Sigma^\circ \to S}_{\substack{\text{homomorphism into}\\ \text{a finite } \circ\text{-semigroup}}} \text{ and } v \in \Sigma^\circ.$$

Exercise 87. (2) For a countable linear order X, let $\{a,b\}^X \subseteq \{a,b\}^\circ$ be the set of $\circ$-words with with positions X. We can equip this set with a probabilistic measure, where for each position $x \in X$, the label is selected independently, with a and b both having probability half. We say that X has a zero-one law if for every MSO definable language L, the probability of $\varphi \cap \{a,b\}^X$ is either zero or one. For which of the following $X = \mathbb{N}, \mathbb{Z}, \mathbb{Q}$ is there a zero-one law?

Exercise 88. (2) A countable linear order can be viewed as a $\circ$-word over a one-letter alphabet. Among these, we can distinguish the countable linear orders that are regular, i.e. generated by the Läuchli-Leonard operations, see Exercise 83. Give an algorithm, which inputs a an countable linear order that is regular in the above sense, and decides if it has a zero-one law (in the sense of Exercise 87).

Exercise 89. (2) Show that every MSO definable language of $\circ$-words belongs to the least class of languages which:

- contains the following two languages over alphabet $\{a,b,c\}$:

$$\underbrace{\exists x a(x)}_{\text{some } a} \qquad \underbrace{\exists x\, \exists y\; a(x) \wedge b(y) \wedge x < y}_{a \text{ before } b}$$

- is closed under Boolean combinations;
- is closed under images and inverse images of letter-to-letter homomorphisms.

Exercise 90. (2) We say that a binary tree (possibly infinite) is *regular* if it has finitely many non-isomorphic sub-trees. Show that a $\circ$-word is regular (in the sense of Exercise 83) if and only if it is the yield (in the sense of Exercise 79) of some regular tree.

Exercise 91. (91) Consider the embedding ordering (Higman ordering) $w \hookrightarrow v$ on ∘-words. Show that for every ∘-words w there is a regular ∘-word v such that $w \hookrightarrow v$ and $v \hookrightarrow w$. Hint: use Lemma 3.15.

Exercise 92. (1) Suppose that we are given a language $L \subseteq \Sigma^\circ$, represented by a finite ∘-semigroup S, a homomorphism $h : \Sigma^\circ \to S$, and an accepting set $F \subseteq S$. Give a algorithm which computes the syntactic ∘-semigroup (which exists by Exercise 78).

Exercise 93. (2) Let $\mathcal{L}$ be a class of languages, such that $\mathcal{L}$ satisfies the following conditions:

- every language in $\mathcal{L}$ is recognised by a finite ∘-semigroup;
- $\mathcal{L}$ is closed under Boolean combinations;
- $\mathcal{L}$ is closed under inverse images of homomorphisms $h : \Sigma^\circ \to \Gamma^\circ$;
- Let $L \subseteq \Sigma^\circ$ be a language in $\mathcal{L}$. For every $w, w_1, \ldots, w_n \in \Sigma^\circ$, $\mathcal{L}$ contains the inverse image of L under the following operations:

$$v \mapsto wv \quad v \mapsto vw \quad v \mapsto v^\omega \quad v \mapsto v^{\omega*} \quad v \mapsto \text{shuffle of } \{w_1, \ldots, w_n, v\}.$$

Show that if L belongs to $\mathcal{L}$, then the same is true for every language recognised by its syntactic ∘-semigroup.

Exercise 94. (2) Let Σ be an alphabet and let $c \notin \Sigma$ be a fresh letter. We say that $L \subseteq \Sigma^\circ$ is definable in LTL[F] if there is a formula of LTL[F] which defines the language cL, see Exercise 41. Give an algorithm which inputs the finite syntactic ∘-semigroup of a language $L \subseteq \Sigma^\circ$, and answers if the language is definable in LTL[F]. Hint: the ∘-semigroup must be suffix trivial, but this is not sufficient.

Exercise 95. (3) Give an algorithm which inputs the finite syntactic ∘-semigroup of a language $L \subseteq \Sigma^\circ$, and answers if the language is definable in two-variable first-order logic FO^2. Hint: the ∘-semigroup must be in DA, but this is not sufficient.

Exercise 96. (2) Show that aperiodicity is not sufficient for first-order definability for ∘-words: give an example of a language $L \subseteq \Sigma^\circ$ that is recognised by a finite aperiodic ∘-semigroup, but which is not definable in first-order logic.

3.4 From $\circ$-semigroups to MSO

In Theorem 3.11, and again in Lemma 3.22, we have shown that if a language of $\circ$-words is definable in MSO, then it is recognised by a finite $\circ$-semigroup. We now show that the converse implication is also true.

Theorem 3.23. *If a language of $\circ$-words is recognised by a finite $\circ$-semigroup, then it is definable in* MSO[13] *which says that*

As mentioned before in this chapter, the theorem would be easy if there was an automaton model, which would assign states to positions, and where the acceptance condition could be formalised in MSO. Unfortunately, no such automaton model is known. Therefore, we need a different proof for the theorem. The rest of Section 3.4 is devoted to such a proof.

We begin by defining regular expressions for $\circ$-words. For a finite family $\mathcal{L}$ of languages of $\circ$-words, define the shuffle of $\mathcal{L}$ to be the $\circ$-words which can be partitioned into intervals so that: (a) every interval induces a word from L for some $L \in \mathcal{L}$; (b) the order type on the intervals is that of the rational numbers; and (c) for every $L \in \mathcal{L}$, the intervals from L are dense.

Lemma 3.24. *Languages definable in* MSO *are closed under Boolean combinations and the following kinds of concatenation:*

$$LK \qquad L^+ \qquad L^\omega \qquad L^{\omega *} \qquad \textit{shuffle of } \underbrace{\mathcal{L}}_{\substack{\textit{a finite family}\\ \textit{of languages}}}$$

Proof For the Boolean operations, there is nothing to do, since Boolean operations are part of the logical syntax. For the concatenations, we observe that MSO can quantify over factorisations, as described below.

Define a *factorisation* of a $\circ$-word to be a partition of its positions into intervals, which are called *blocks*. For a factorisation, define a *compatible colouring* to be any colouring of positions that uses two colours, such that all blocks are monochromatic, and if a block has a successor, then the successor has a different colour. A compatible colouring always exists (there could be uncountably many choices). A factorisation can be recovered from any compatible colouring: two positions are in the same block if and only if the interval connecting

[13] This theorem was first shown in
[8] Carton, Colcombet, and Puppis, “An algebraic approach to MSO-definability on countable linear orders”, 2018 , Theorem 5.1.
The proof presented here is different, and it is based on the proof in
[31] Schützenberger, “On finite monoids having only trivial subgroups”, 1965 , p. 192
which shows that every aperiodic monoid recognises a star-free language. We use the different proof because, after suitable modifications, it allows us to characterise star-free languages of $\circ$-words, see Exercise 102.

them is monochromatic. A compatible colouring can be represented using a single set – namely the positions with one of the two colours. This representation can be formalised in MSO, i.e. one can write an MSO formula $\varphi(x, y, X)$ which says that positions x and y are in the same block of the factorisation which arises from the compatible colouring represented by set X.

Using the above representation, we show closure of MSO under the concatenations in the lemma. For LK, we simply say that there exists a factorisation with two blocks, where the first block is in L and the second block is in K. (To say that a block is in L or K, we observe that MSO sentences can be relativised to a given interval.) For L^+, we say that there exists a factorisation with finitely many blocks, where all blocks are in L. Here is how we express that there are finitely many blocks: there are first and last blocks, and there is no proper subset of positions that contains the first block and is closed under adding successor blocks. For L^ω, we do the same, except that there is no last block. For $L^{\omega *}$, we use a symmetric approach. For the shuffle, we say that the blocks are dense and there is no first and last block. □

In the proof of Theorem 3.23, we will only use the closure properties of MSO from the above lemma. In particular, it will follow that every language recognised by a finite ∘-semigroup can be defined by a regular expression which uses single letters and the closure operations from the lemma (which include intersection and complementation).

To prove Theorem 3.23, we will show that the product operation of every finite ∘-semigroup can be defined in MSO, in the following sense. Let S be a finite ∘-semigroup. We will show that for every $a \in S$, the language

$$L_a = \{w \in S^\circ : w\text{has product } a\}$$

is MSO definable. This will immediately imply that every language recognised by a homomorphism into S is MSO definable, thus proving the theorem.

The proof is by induction on the infix ordering. Fix for the rest of this section an infix class $J \subseteq S$. We partition S into two parts:

$$\underbrace{\text{easy elements}}_{\text{proper prefixes of } J} \quad \cup \quad \underbrace{\text{hard elements}}_{\text{the rest}}.$$

The induction hypothesis says L_a is MSO definable for every easy $a \in S$. We need to prove the same thing for every $a \in J$.

We begin with an observation about smooth products, which follows from the Ramsey argument that was used in Theorem 3.2. We say that $w \in S^\circ$ is *J-smooth* if every finite infix of w has a product in J. This is a lifting to infinite words of the notion of smoothness that was used in Section 1.3. In particular, by Claim 1.22 from that section, a $\circ$-word is J-smooth if and only if all of its infixes of length at most two are J-smooth. The following lemma describes the products of certain J-smooth words.

Lemma 3.25. *For every idempotent $e \in J$ the following holds. Let $w \in J^\circ$ be J-smooth. If w is an ω-word, then its product is ae^ω, where a depends only on e and the first letter in w. If w is an $(\omega{*} + \omega)$-word, then its product is $e^{\omega*}e^\omega$.*

Since the lemma is true for every choice of idempotent e, it follows that $e^{\omega*}e^\omega$ does not depend on the choice of e.

Proof The main observation is the following claim.

Claim 3.26. *If w is J-smooth and has first letter e, then its product is e^ω.*

Proof By Lemma 3.4, the product of w is equal to af^ω, for some a, f. Since w is J-smooth, a and f belong to J. Since the first letter of w is e, we have $ea = a$. Since f is infix equivalent to e, it admits a decomposition $f = xeey$. Therefore

$$af^\omega = ea(xeey)^\omega = \underbrace{eaxe}_{g}(\underbrace{eyxe}_{h})^\omega.$$

We now continue as in the proof of Lemma 3.5: because g, h, e are in the same group, then $g^\omega = e^\omega = h^\omega$, and therefore $gh^\omega = e^\omega$. □

The claim immediately proves the lemma. Indeed, consider a J-smooth ω-word with first letter a. The first letter admits a decomposition axe for some x, because every prefix class intersects the suffix class of e. By the above claim, the product of every J-smooth ω-word that begins with a is equal to axe^ω. A similar argument works when the positions are ordered as the integers: every J-smooth $(\omega{*} + \omega)$-word has the same product as a smooth $(\omega{*} + \omega)$-word with an infix ee, and the latter has product $e^{\omega*}e^\omega$ thanks to the claim and its symmetric version for $\omega{*}$. □

In the rest of the proof, we will use the following terminology. We say that a colouring (a function from $\circ$-words to a finite set of colours) is MSO definable if for every colour, its inverse image is an MSO definable language. We say that a colouring λ is MSO definable on a subset L of inputs if there exists an MSO definable colouring that agrees with λ on inputs from L.

The strategy for the rest of the proof is as follows. Define $L_J \subseteq S^\circ$ to be the $\circ$-words that have product in J. We first show in Lemma 3.27 that the colouring

$$w \in S^\circ \quad \mapsto \quad \text{prefix class of the product of } w$$

is MSO definable on L_J. Next, we use this result about prefixes and a symmetric one for suffixes, to show in Lemma 3.29 that the product operation is MSO definable on L_J. Finally, in Lemma 3.32 we show that the language L_J is definable in MSO. We can then conclude as follows: a ∘-word has product $a \in J$ if and only if it belongs to L_J, and the colouring from Lemma 3.29 step maps it to a. It remains to prove the lemmas.

Lemma 3.27. *The following colouring is* MSO *definable on L_J:*

$$w \in S^\circ \quad \mapsto \quad \textit{prefix class of the product of } w.$$

Proof We write $H \subseteq S^\circ$ for the ∘-words with a hard product. This language is definable in MSO, as the complement of the easy products that are definable by induction assumption. For an interval in w, define its product to be the product of the infix of w that is induced by the interval. An interval is called *easy* if its product is easy, otherwise it is called *hard*. By the induction assumption, we can check in MSO if an interval is easy or hard. An interval is called *almost easy* if it all of its proper sub-intervals are easy.

Claim 3.28. *The product operation is* MSO *definable on easy intervals.*

Proof If there is a last position, then the product can be easily computed: remove the last position, compute the product, and then add the last position. Otherwise, if there is no last position, then we can use Lemma 3.4 to see that an almost easy interval has product $b \in S$ if and only if it belongs to

$$L_a(L_e)^\omega \qquad \text{for some easy } a, e \text{ such that } ae^\omega = a.$$

The above condition can be formalised in MSO thanks to the induction assumption and Lemma 3.24. □

Define a *prefix interval* to be a nonempty downward closed interval. We will compute in MSO the prefix class of some hard prefix interval; if the ∘-word is in L_J then this hard prefix has the same prefix class as w. We do a case disjunction, depending on whether or not there is an easy prefix interval (which can clearly be checked in MSO).

- Suppose first that there is no easy prefix interval. This means that that either w has a first letter which is in J, or $w \in H^{\omega*}$. In the first case, the first letter uniquely determines the prefix class of the product of w. In the second case, when $w \in H^{\omega*}$, then under the assumption of $w \in L_J$, we can use Lemma 3.25 to conclude that the product of w is in the prefix class of $e^{\omega*}$, where e is some arbitrarily chosen idempotent from J.

- Suppose next that there is some easy prefix interval. Let X be the union of all easy prefix intervals. This is an almost easy interval, and therefore its product $a \in S$ can be computed thanks to Claim 3.28. If a is hard, then we know the prefix class of w. Otherwise, if a is easy, it follows that after removing X, we get a $\circ$-word as in the first case, and we can use that case to compute the prefix class.

□

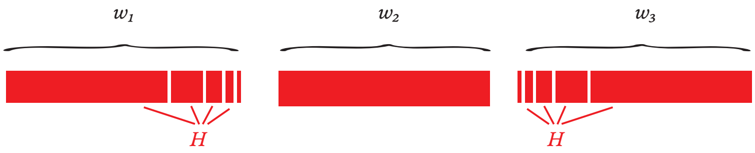
Lemma 3.29. *The product operation of S is* MSO *definable on L_J.*

Proof Let $w \in L_J$. We use the terminology about intervals from the proof of Lemma 3.27 .

Claim 3.30. *There exists a factorisation $w = w_1 w_2 w_3$ such that:*

- *w_1 is either empty or in H^ω;*
- *w_2 is a finite concatenation of almost easy $\circ$-words;*
- *w_3 is either empty or in $H^{\omega*}$.*

Here is a picture of the factorisation:



Proof Define a *limit prefix* of w to be any prefix interval which induces a $\circ$-word in H^ω. Limit prefixes are closed under (possibly infinite) unions. If there is a limit prefix, then there is a maximal one, namely the union of all limit prefixes (if there are not limit prefixes, we define the maximal limit prefix to be empty). Define $w_1 \in H^\omega$ to be the maximal limit prefix of w (if no limit prefix exists, then w_1 is empty). Remove the prefix w_1, and to the remaining part of the word apply a symmetric process, yielding a suffix $w_3 \in H^{\omega*}$ and a remaining part w_2. This is the factorisation in the statement of the claim.

It remains to show that w_2 is a finite concatenation of almost easy $\circ$-words. By construction, the remaining part w_2 does not have any prefix in H^ω, nor does it have any suffix in $H^{\omega*}$. Take the union of all easy prefixes of w_2 (this union exists, because w_2 has no suffix in $H^{\omega*}$, and it is almost easy), and cut it off. After repeating this process a finite number of times, we must exhaust all of w_2, since otherwise there would be a prefix in H^ω. Therefore, w_2 is a finite concatenation of almost easy $\circ$-words. □

Let w_1, w_2, w_3 be as in the above claim. By Lemma 3.25, the product of w_1 is uniquely determined by its prefix class (under the assumption that the entire ∘-word belongs to L_J). Therefore, thanks to Lemma 3.27, we can compute in MSO the product of the w_1. Symmetrically, we can compute the product of w_3. It remains to compute the product of w_2. This is done in the following claim.

Claim 3.31. *If a ∘-word is a finite concatenation almost easy ∘-words, then its product can be computed in* MSO*.*

Proof By the Kleene theorem about regular expressions being equivalent to finite automata, the set of finite concatenations of almost easy intervals can be described using a regular expressions, where the atomic expressions describe almost easy words of given product. Such a regular expression can be formalised in MSO thanks to Lemma 3.24 □

□

Lemma 3.32. *The language L_J is* MSO *definable.*

Proof Define $I \subseteq S$ to be the hard elements which are not in J. This is an ideal in the ∘-semigroup S, i.e. if $w \in S^\circ$ has at least one letter in I, then its product is in I. Define L_I to be the ∘-words with product in I. Again, this is an ideal, this time in the free ∘-semigroup S°. We will show how to define L_I in MSO; it will follow that L_J is MSO definable as

$$L_J = H - L_I.$$

The key is the following characterisation of L_I. Define an *error* to be a ∘-word in S° which satisfies at least one of the following conditions:

- binary error: belongs to $L_a L_b$ for some $a, b \in S - I$ such that $ab \in I$;
- ω-error: belongs to $(L_a)^\omega$, for some $a \in S - I$ such that $a^\omega \in I$;
- $\omega*$-error: belongs to $(L_a)^{\omega*}$, for some $a \in S - I$ such that $a^{\omega*} \in I$;
- shuffle error: is in the shuffle of $\{L_a\}_{a \in A}$ for some $A \subseteq S - I$ such that $A^\eta \in I$.

Note that in the above definition, we can use languages L_a for $a \in J$. These languages are not yet known to be definable in MSO.

Claim 3.33. *A ∘-word belongs to L_I if and only if it has an error infix.*

Proof Clearly every error is in L_I, and since L_I is an ideal, it follows that L_I contains every ∘-word with an error infix. We are left with the converse implication: every ∘-word in L_I contains an error infix. To prove this implication, we will show that the language

$$L = \{w \in S^\circ : \text{if } w \in L_I \text{ then } w \text{ has an error infix}\}$$

satisfies the assumptions of Lemma 3.15, with λ being the product operation in S. The conclusion of Lemma 3.15 will then say that L is equal to S°, thus showing that every $\circ$-word in L_I has an error infix.

The first assumption of Lemma 3.15 says that L is closed under binary concatenation. Suppose that $u, v \in L$. We need to show that $uv \in L$. Suppose that $uv \in L_I$. If $u \in L_I$, then it has an error infix by assumption on $u \in L$, and therefore also uv has an error infix. We argue similarly if $v \in L_I$. Finally, if both u, v have products in $S - I$, then uv is a binary error.

The remaining assumptions of Lemma 3.15 are checked the same way. □

As remarked before Claim 3.33, the definition of errors refers to languages L_a with $a \in J$, which are not yet known to be definable in MSO. We deal with this issue now. By Lemma 3.29, for every $a \in J$ there an MSO definable language which contains all $\circ$-words that have product a, and does not contain any $\circ$-words that have product in $J - \{a\}$. By removing the $\circ$-words with easy products from that language, we get an MSO definable language K_a with

$$L_a \subseteq K_a \subseteq L_a \cup L_I.$$

Define a *weak error* in the same way as an error, except that K_a is used instead of L_a for $a \in J$. Since K_a is obtained from L_a by adding some words from the ideal L_I, it follows from Claim 3.33 that a $\circ$-word is in L_I if and only if it has an infix that is a weak error. Finally, weak errors can be defined by an expression which uses MSO definable languages and the closure operators from Lemma 3.24, and therefore weak errors are MSO definable. It follows that L_I is MSO definable, and therefore L_J is MSO definable. □

As we have already remarked when describing the proof strategy, the above lemma completes the proof of the induction step in Theorem 3.23. Indeed, a $\circ$-word has product $a \in J$ if and only if it belongs to L_J and it is assigned a by the colouring from Lemma 3.29.

Exercises

Exercise 97. (2) The syntax of star-free expression for $\circ$-words is the same as for finite words, except that the complementation operation is interpreted as $\Sigma^\circ - L$ instead of $\Sigma^* - L$. Define a $\circ$-star-free language to be a language $L \subseteq \Sigma^\circ$ that is defined by a star-free expression. Show that if L is $\circ$-star-free, then its syntactic $\circ$-semigroup is aperiodic, but the converse implication fails.

Exercise 98. (1) What is the modification for ∘-star-free expressions that is needed to get first-order logic (over the ordered model)?

Exercise 99. (2) Show that if $L \subseteq \Sigma^\circ$ is ∘-star-free, then the same is true for every language recognised by its syntactic ∘-semigroup.

Exercise 100. (2) Show that if $L \subseteq \Sigma^\circ$ is ∘-star-free, then the same is true for L^ω.

Exercise 101. (2) Show that if S is aperiodic, then the constructions from Lemmas 3.27 and 3.29 can be done using ∘-star-free expressions.

Exercise 102. (2) Show that $L \subseteq \Sigma^\circ$ is ∘-star-free if and only if its syntactic ∘-semigroup is finite, aperiodic and satisfies[14]:

$$e^{\omega*} = e = e^{\omega} \quad \Rightarrow \quad e = \{e\}^{\eta} \qquad \text{for every idempotent } e.$$

Hint: use Exercises 100 and 101.

Exercise 103. (1) Show that languages of ∘-words definable in first-order logic (in the ordered model) are not closed under concatenation LK.

Exercise 104. (2) We say that a product operation $\pi : S^\circ \to S$ is regular-associative if it satisfies the associativity condition from Definition 3.10, but with the diagrams restricted so that only

$$S^\bullet = \{w \in S^\circ : w \text{ is regular}\}$$

is used instead of S°. Show that if S finite and $\pi : S^\circ \to S$ is MSO definable and regular-associative, then π is associative.

Exercise 105. (2) Show that if S is finite and $\pi : S^\bullet \to S$ is regular associative, then it can be extended uniquely to an associative product $\bar{\pi} : S^\circ \to S$. Hint: the MSO formulas defined in the proof of Theorem 3.23 depend only on the Läuchli-Leonard operations of the ∘-semigroup S.

[14] This exercise is based on
[9] Colcombet and Sreejith, "Limited Set Quantifiers over Countable Linear Orderings", 2015 , Theorem 2, item 2.

PART TWO

MONADS

4
Monads

As discussed in Chapter 1, instead of viewing a semigroup as having a binary multiplication operation, one could think of a semigroup as a set S equipped with a multiplication operation $\mu : S^+ \to S$, which is associative in the sense that the following two diagrams commute:

$$\begin{tikzcd}
S \arrow[d, "\text{view a letter as}\\\text{a one-letter word}"', align=center] \arrow[dr, "\text{identity}"] & \\
S^+ \arrow[r, "\mu"'] & S
\end{tikzcd}
\qquad
\begin{tikzcd}
(S^+)^+ \arrow[r, "\text{multiplication in free semigroup } S^+"] \arrow[d, "\mu^+"'] & S^+ \arrow[d, "\mu"] \\
S^+ \arrow[r, "\mu"'] & S
\end{tikzcd}$$

The same is true for monoids, with $*$ used instead of $+$, and for $\circ$-semigroups, with $\circ$ used instead of $+$. In this chapter, we examine the common pattern behind this constructions, which is that they are the Eilenberg-Moore algebras for the monads of $+$-words, $*$-words and $\circ$-words, respectively.

From the perspective of this book, the idea behind monads is the following. Instead of first defining not necessarily free algebras (e.g. semigroups) and then defining free algebras (e.g. the free semigroup) as a special case, an opposite approach is used. We begin with the free algebra (which is the monad), and then other, not necessarily free, algebras are defined as a derived notion (which is the Eilenberg-Moore algebras of the monad). This opposite approach is be useful for less standard algebras such as graphs: axiomatising the not necessarily free algebras is possible but tedious and not intuitive, while the free algebra is very natural, because it consists of graphs with a certain substitution structure.

4.1 Monads and their Eilenberg-Moore algebras

This section, presents the basic definitions for monads and their algebras. These notions make sense for arbitrary categories. However, for simplicity we use the category of sets and functions, because this is where most of our examples live. In later chapters we will consider multi-sorted sets (e.g. sets with sorts $\{+, \omega\}$ for ω-semigroups, or sets with sorts $\{0, 1, \ldots\}$ for hypergraphs), but this is as far as we go with respect to the choice of categories.

Definition 4.1 (Monad)**.** A *monad* in the category of sets[1] consists of the following ingredients:

- *Structures:* for every set X, a set $\mathsf{T}X$;
- *Substitution:* for every function $f : X \to Y$, a function $\mathsf{T}f : \mathsf{T}X \to \mathsf{T}Y$;
- *Unit and free multiplication:* for every set X, two functions

$$\underbrace{\mathsf{unit}_X : X \to \mathsf{T}X}_{\text{the unit of } X} \qquad \underbrace{\mathsf{mult}_X : \mathsf{T}\mathsf{T}X \to \mathsf{T}X}_{\text{free multiplication on } X}.$$

These ingredients are subject to six axioms

$$\underbrace{(4.1)}_{\mathsf{T}\text{ is a functor}} \qquad \underbrace{(4.2) \quad (4.3)}_{\substack{\text{unit and multiplication}\\ \text{are natural transformations}}} \qquad \underbrace{(4.4) \quad (4.5) \quad (4.6)}_{\substack{\mathsf{T}X\text{ with free multiplication}\\ \text{is an Eilenberg-Moore algebra,}\\ \text{and one more associativity axiom}}}$$

which will be described later in this section.

Before describing the monad axioms, we discuss some examples, and define Eilenberg-Moore algebras. The purpose of the monad axioms is to ensure that Eilenberg-Moore algebras are well-behaved. Therefore it is easier to see the monad axioms after the definition of Eilenberg-Moore algebras.

Example 4.2 (Monad of finite words)**.** The monad of finite words is defined as follows. The structures are defined by $\mathsf{T}X = X^*$. For a function f, its corresponding substitution $\mathsf{T}f$ is defined by applying f to every letter in the input word. The unit operation maps a letter to the one-letter word consisting of this letter. Free multiplication flattens a word of words into a word. The monad of $\circ$-words is defined as in the same way, except that one uses $\circ$-words.

For this book, the key notion about monads is their Eilenberg-Moore algebras. The idea is that $\mathsf{T}X$ describes the free algebra, while the Eilenberg-Moore algebras are the algebras that are not necessarily free.

[1] The same definition can be applied to any other category, by using “object” instead of “set”, and “morphism” instead of “function”.

Definition 4.3 (Eilenberg-Moore algebras)**.** An Eilenberg-Moore algebra in a monad T, also called a T-*algebra*, consists of an *underlying set* A and a *multiplication operation* $\mu : \mathsf{T}A \to A$, subject to the following associativity axioms:

$$\begin{tikzcd}
A \arrow[d, "\mathsf{unit}_A"'] \arrow[dr, "\text{identity}"] & \\
\mathsf{T}A \arrow[r, "\mu"'] & A
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{TT}A \arrow[r, "\text{free multiplication on } A"] \arrow[d, "\mathsf{T}\mu"'] & \mathsf{T}A \arrow[d, "\mu"] \\
\mathsf{T}A \arrow[r, "\mu"'] & A
\end{tikzcd}$$

The reader will recognise, of course, the diagramatic definitions of semigroups, monoids and $\circ$-semigroups. Also, the diagramatic definition of ω-semigroup will fall under the scope of the above definition, if we think about ω-semigroups as living in the category of sets with two sorts $\{+, \omega\}$.

By abuse of notation, we use the same letter to denote a T-algebra and its underlying set, assuming that the multiplication operation is clear from the context. Also, if the monad T is clear from the context, we will say algebra instead of T-algebra.

Example 14. [Group monad] The *free group over a set X* is defined to be

$$\underbrace{(X + X)}_{\substack{\text{two copies of } X, \\ \text{one blue, and one red}}}{}^{*}$$

modulo the identities

$$xx = xx = \varepsilon \qquad \text{for every } x \in X,$$

where ε represents the empty word, while x and x represent the blue and red copies of x. The identities can be applied in any context. For example,

$$zx \qquad zyyx \qquad zzzxyy,$$

represent the same element of the free group. Define T to be the monad where $\mathsf{T}X$ is the free group over X, and the remaining monad structure is defined similarly as for finite words, except that we have the two copies of the alphabet, and the identities. The unit operation maps an element to its blue copy.

An algebra over this monad is the same thing as a group. Indeed, if G is an algebra over this monad, with multiplication μ, then the group structure is recovered as follows:

$$\underbrace{1 \overset{\text{def}}{=} \mu(\varepsilon)}_{\text{group identity}} \qquad \underbrace{x^{-1} \overset{\text{def}}{=} \mu(x)}_{\text{group inverse}} \qquad \underbrace{x \cdot y \overset{\text{def}}{=} \mu(xy)}_{\text{group operation}}$$

The axioms of a group are easily checked, e.g.

$$\begin{array}{rcl}
x \cdot x^{-1} & = & \text{(definition of inverse)} \\
x \cdot \mu(x) & = & \text{(unit followed by multiplication is the identity, i.e. axiom 4.4)} \\
\mu(x) \cdot \mu(x) & = & \text{(definition of the group operation)} \\
\mu(\mu(x)\mu(x)) & = & \text{(associativity of multiplication, i.e. axiom 4.5)} \\
\mu(xx) & = & \text{(equality in the free group)} \\
\mu(\varepsilon) & = & \text{(definition of group identity)} \\
1 & &
\end{array}$$

For the converse, we observe that for every group G, its group multiplication can be extended uniquely to an operation of type $\mathsf{T}G \to G$, and the resulting operation will be associative in the sense required by Eilenberg-Moore algebras. □

4.1.1 Axioms of a monad

We now describe the axioms of a monad.

Functoriality. The first group of axioms says that the first two ingredients (the structures and substitutions) of a monad are a functor in the sense of category theory. This means that substitutions preserve the identity and composition of functions. Preserving identity means that if we apply T to the identity function on X, then the result is the identity function on $\mathsf{T}X$. Preserving composition means that the composition of substitutions is the same as the substitution of their composition, i.e. for every functions $f : X \to Y$ and $g : Y \to Z$, the following diagram commutes

$$\begin{tikzcd}
\mathsf{T}X \arrow[r, "\mathsf{T}f"] \arrow[dr, "\mathsf{T}(g\circ f)"'] & \mathsf{T}Y \arrow[d, "\mathsf{T}g"] \\
& \mathsf{T}Z
\end{tikzcd} \tag{4.1}$$

Naturality. The naturality axioms say that for every function $f : X \to Y$, the following diagrams commute.

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\mathsf{unit}_X"'] & Y \arrow[d, "\mathsf{unit}_Y"] \\
\mathsf{T}X \arrow[r, "\mathsf{T}f"'] & \mathsf{T}Y
\end{tikzcd} \tag{4.2}$$

$$\begin{tikzcd}
\mathsf{TT}X \arrow[r, "\mathsf{TT}f"] \arrow[d, "\text{free multiplication on } X"'] & \mathsf{TT}Y \arrow[d, "\text{free multiplication on } Y"] \\
\mathsf{T}X \arrow[r, "\mathsf{T}f"'] & \mathsf{T}Y
\end{tikzcd} \tag{4.3}$$

In the language of category theory, this means that the unit and free multiplication are natural transformations. Also, as we will see later on, the second naturality axiom (naturality of free multiplication) says that the substitution $\mathsf{T}f$ is a homomorphism between the free algebras $\mathsf{T}X$ and $\mathsf{T}Y$.

Associativity. We now turn to the most important monad axioms, which ensure the the Eilenberg-Moore algebras are well behaved. The main associativity axiom says that for every set X, the set $\mathsf{T}X$ equipped with free multiplication on X is a T-algebra (we call this the free T-algebra over X, or simply free algebra if the monad is clear from the context). By unravelling the definitions, this means that the following two diagrams commute:

$$\begin{tikzcd}
\mathsf{T}X \arrow[d, "\mathsf{unit}_{\mathsf{T}X}"'] \arrow[dr, "\text{identity}"] & \\
\mathsf{TT}X \arrow[r, "\text{free multiplication on } X"'] & \mathsf{T}X
\end{tikzcd} \tag{4.4}$$

$$\begin{tikzcd}
\mathsf{TTT}X \arrow[r, "\text{free multiplication on } \mathsf{T}X"] \arrow[d, "\mathsf{T}(\text{free multiplication on } X)"'] & \mathsf{TT}X \arrow[d, "\text{free multiplication on } X"] \\
\mathsf{TT}X \arrow[r, "\text{free multiplication on } X"'] & \mathsf{T}X
\end{tikzcd} \tag{4.5}$$

Apart from the above two, there is one more associativity axiom, namely:

$$\begin{tikzcd}
\mathsf{T}X \arrow[d, "\mathsf{T}(\mathsf{unit}_X)"'] \arrow[dr, "\text{identity}"] & \\
\mathsf{TT}X \arrow[r, "\text{free multiplication on } X"'] & \mathsf{T}X
\end{tikzcd} \tag{4.6}$$

This completes the axioms of a monad, and the definition of a monad.

4.1.2 Homomorphisms and recognisable languages

A homomorphism between two T-algebras is any function between their underlying sets which is consistent with the multiplication operation.

Definition 4.4 (Homomorphism)**.** Let T be a monad. A T-*homomorphism* is a function $h : A \to B$ on the underlying sets of two T-algebras A and B, which makes the following diagram commute

$$\begin{tikzcd}
\mathsf{T}A \arrow[r, "\mathsf{T}h"] \arrow[d, "\text{multiplication in } A"'] & \mathsf{T}B \arrow[d, "\text{multiplication in } B"] \\
A \arrow[r, "h"'] & B
\end{tikzcd}$$

When the monad is clear from the context, we simply write homomorphism, instead of T-homomorphism. Again, the reader will recognise the notion of homomorphism for semigroups, monoids and $\circ$-semigroups.

In the rest of this section, we describe some basic properties of homomorphisms.

Lemma 4.5. *Homomorphisms are closed under composition.*

Proof Consider two homomorphisms

$$A \xrightarrow{g} B \xrightarrow{h} C.$$

Saying that the composition $h \circ g$ is a homomorphism is the same as saying that the perimeter of the following diagram commutes:

$$\begin{tikzcd}
\mathsf{T}A \arrow[rr, "\mathsf{T}(h\circ g)"] \arrow[dr, "\mathsf{T}g"'] \arrow[ddd, "\text{multiplication in } A"'] & & \mathsf{T}C \arrow[ddd, "\text{multiplication in } C"] \\
& \mathsf{T}B \arrow[ur, "\mathsf{T}h"'] \arrow[d, "\text{multiplication in } B"'] & \\
& B \arrow[ddr, "h"] & \\
A \arrow[ur, "g"] \arrow[rr, "h\circ g"'] & & C
\end{tikzcd}$$

The upper face commutes because of the functoriality axioms (substitutions are compatible with composition). The left and right faces commute by assumption that g and h are homomorphisms, and the lower face commutes by definition. □

Recall that we defined the *free algebra over X* to be X equipped with free multiplication on X. The monad axioms say that this is indeed an algebra. It is called free because of the universal property given in the following lemma.

Lemma 4.6 (Free Algebra Lemma)**.** *For every set X, the free algebra $\mathsf{T}X$ has the following universal property:*

$$\forall\exists! \qquad \begin{tikzcd} X \arrow[rr, "\text{function on sets}"] \arrow[drr, "\mathsf{unit}_X"'] & & A \\ & & \mathsf{T}X \arrow[u, "\text{homomorphism of } \mathsf{T}\text{-algebras}"'] \end{tikzcd} \tag{4.7}$$

Proof We begin by showing that there is at least one blue homomorphism h for each red function f; later we show that this homomorphisms is unique. Let $\mu : \mathsf{T}A \to A$ be multiplication in the algebra A. Define h to be the composition of the following functions:

$$\mathsf{T}X \xrightarrow{\mathsf{T}f} \mathsf{T}A \xrightarrow{\mu} A.$$

The axiom on naturality of free multiplication says that $\mathsf{T}f$ is a homomorphism from the free algebra $\mathsf{T}X$ to the free algebra $\mathsf{T}A$. The associativity axiom in the definition of an Eilenberg-Moore algebra says that multiplication μ is a homomorphism from the free algebra $\mathsf{T}A$ to the algebra A. Therefore, h is a homomorphism, as the composition of two homomorphisms $\mathsf{T}f$ and μ.

We now show uniqueness – every homomorphism h which makes the diagram must be equal to the one described above. Consider the following diagram:

$$\begin{tikzcd} & \mathsf{T}X \arrow[dl, "\mathsf{T}f"'] \arrow[d, "\mathsf{Tunit}_X"'] \arrow[dr, "\text{identity}"] & \\ \mathsf{T}A \arrow[dr, "\text{multiplication in } A"'] & \mathsf{TT}X \arrow[l, "\mathsf{T}h"'] \arrow[r, "\text{free multiplication on } X"] & \mathsf{T}X \arrow[dl, "h"] \\ & A & \end{tikzcd}$$

The upper left face commutes by applying T to the assumption that h extends f. (Applying T preserves commuting diagrams, because of the functoriality axioms.) The upper right face commutes by the first associativity axiom. The lower face commutes, because it says that h is a homomorphism. Therefore, the perimeter of the diagram commutes. The perimeter says that says that h must be equal to $\mathsf{T}f$ followed by multiplication in A, and therefore h is unique. □

Compositional functions. Fix a monad T. Suppose that A is an algebra, while B is a set, which is not (yet) equipped with a multiplication operation. We say that a function $h : A \to B$ is *compositional* if

$$\begin{tikzcd}
\exists\mu & \mathsf{T}A \arrow[r, "\mathsf{T}h"] \arrow[d, "\text{multiplication in } A"'] & \mathsf{T}B \arrow[d, "\mu"] \\
& A \arrow[r, "h"'] & B
\end{tikzcd}$$

This is the same notion of compositionality as was used for monoids, semigroups and ∘-semigroups in part I of the book. For the same reason as before, surjective compositional functions are equivalent to surjective homomorphisms, as stated in the following lemma.

Lemma 4.7. *If A is an algebra, B is a set, and $h : A \to B$ is compositional and surjective, then there is a (unique) multiplication operation on B which turns it into a algebra and h into a homomorphism.*

Proof The multiplication operation – no surprises here – is μ from the definition of a compositional function. The diagram in the definition of a compositional function is the same diagram as in the definition of a homomorphism, and therefore if B equipped with μ is an algebra, then h is a homomorphism. It remains to show that B equipped with μ is indeed an algebra. We only prove the more interesting of the two associativity diagrams.

We first observe that T preserves surjectivity of functions. Indeed, if a function $h : A \to B$ is surjective, then it has a one-sided inverse, i.e. a function $h^{-1} : B \to A$ such that $h \circ h^{-1}$ is the identity on B. By the functoriality axioms, $\mathsf{T}h^{-1}$ is a one-sided inverse for $\mathsf{T}h$, and therefore $\mathsf{T}h$ is also surjective. This argument justifies the surjectivity annotation (double-headed arrows) in the following diagram.

$$\begin{tikzcd}
\mathsf{TT}B \arrow[rrr, "\text{free multiplication on } B"] \arrow[ddd, "\mathsf{T}\mu"'] & & & \mathsf{T}B \arrow[ddd, "\mu"] \\
& \mathsf{TT}A \arrow[lu, two heads, "\mathsf{TT}h"'] \arrow[r, "\text{free multiplication on } A"] \arrow[d, "\mathsf{T}(\text{multiplication in } A)"'] & \mathsf{T}A \arrow[ru, two heads, "\mathsf{T}h"] \arrow[d, "\text{multiplication in } A"] & \\
& \mathsf{T}A \arrow[r, "\text{multiplication in } A"'] \arrow[ld, two heads, "\mathsf{T}h"] & A \arrow[rd, two heads, "h"] & \\
\mathsf{T}B \arrow[rrr, "\mu"'] & & & B
\end{tikzcd}$$

The central face commutes by the assumption that A is an algebra. The upper face commutes by naturality of free multiplication. The right and lower faces commute by definition of a compositional function, and the left face commutes by the same definition with T applied to it. It follows that all paths that begin in $\mathsf{TT}A$ and end in B denote the same function. Since h is surjective, it follows that the perimeter of the diagram commutes. This proves the second of the associativity diagrams in the definition of an Eilenberg-Moore algebra. □

Recognisable colourings and languages. In this book, we are most interested in the Eilenberg-Moore algebras as recognisers of languages. A language is a subset L of a free algebra $\mathsf{T}\Sigma$. (Typically we are interested in the case where the alphabet Σ is finite, but this assumption does not seem to play a role in the results we care about, so we omit it.) A language is called *recognisable* if it is recognised by a finite algebra, as explained in the following definition (which uses a slightly more general notion of language, called colourings).

Definition 4.8 (Recognisable colourings)**.** Fix a monad T. An *algebra colouring* is defined to be any function from an algebra to a set of colours[2]. A *finite algebra* is an algebra where the underlying set is finite[3]. An algebra colouring $\lambda : A \to U$ is called *recognisable* if it factors through a homomorphism into a finite algebra, as expressed in the following diagram:

$$\begin{tikzcd}
\exists & A \arrow[r, "\lambda"] \arrow[dr, "\text{homomorphism into a finite algebra}"'] & U \\
& & B \arrow[u, "\text{colouring}"']
\end{tikzcd}$$

Note that a recognisable colouring will necessarily use finitely many colours.

A language can be viewed as the special case of an algebra where the algebra is a free algebra and there are two colours “yes” and “no”. For languages, we prefer set notation, e.g. we can talk about the complement of a language, or order languages by inclusion. The above definition is easily seen to coincide with the notions of recognisability for semigroups, monoids and $\circ$-semigroups that were discussed in the first part of this book. In the next section, we give more examples.

[2] For some monads, it will be more useful to deviate from this definition. For example, in the monad from Example 23 that deals with vector spaces, a more useful notion of colouring is a linear map to the underlying field. Therefore, one could think of a parametrised notion of recognisability, where the notion of “algebra colouring” is taken as a parameter.

[3] Like for algebra colourings, sometimes this notion of finite algebra is not the right one. In the monad from Example 23, the more useful notion is that a finite algebra is one where the underlying set is a vector space of finite dimension. Again, one could think of the notion of “finite algebra” as being a parameter.

Exercises

Exercise 106. Consider a monad T in the category of sets. For a binary relation R on a set X, define

$$R^X \subseteq (\mathsf{T}X) \times (\mathsf{T}X)$$

to be the binary relation on $\mathsf{T}X$ that is defined by

$$R^X = \{((\mathsf{T}\pi_1)(t), (\mathsf{T}\pi_2)(t)) : t \in \mathsf{T}R\}.$$

Does transitivity of R imply transitivity of R^{T}?

Exercise 107. For an algebra A with multiplication operation $\mu : \mathsf{T}A \to A$, define its powerset as follows: the underlying set is the powerset $\mathsf{P}A$, and multiplication is defined by

$$t \in \mathsf{TP}A \quad \mapsto \quad \{\mu(s) : s \in^{\mathsf{T}} t\},$$

where $\in^{\mathsf{T}}$ is defined as in Exercise 106. Show an example of a monad T where this construction does not yield an algebra.

Exercise 108. Find a monad T in the category of sets where the following implication is not true: if $L \subseteq \mathsf{T}\Sigma$ is recognisable, and $h : \mathsf{T}\Sigma \to \mathsf{T}\Gamma$ is a homomorphism, then $h(L)$ is recognisable.

Exercise 109. Show that the monad of groups satisfies the implication from Exercise 108.

Exercise 110. Consider the implication in the previous exercise. Show that even if we restrict h to substitutions, i.e. homomorphisms of the form $\mathsf{T}f$ for some $f : \Sigma \to \Gamma$, then the implication can still be false in some monads.

4.2 A zillion examples

Monads have an abundance of interesting examples. This section is devoted to a collection of such examples, with an emphasis on the algebras arising from the monads, and the languages recognised by the finite algebras.

Example 15. [Finite multisets] Define $\mathsf{T}X$ to be the finite multisets over X, i.e. functions of type $X \to \mathbb{N}$ which have finite support (all but finitely many elements are mapped to 0). Functions are lifted to multisets point-wise, e.g.

$$\underbrace{\{x_1, \ldots, x_n\}}_{\text{red brackets indicate multisets}} \quad \overset{\mathsf{T}f}{\mapsto} \quad \{f(x_1), \ldots, f(x_n)\}.$$

Another perspective on finite multisets is that they are finite words modulo commutativity $xy = yx$. The unit is $x \mapsto \{x\}$, and free multiplication is simply removing nested brackets, e.g.

$$\{\{x, y\}, \{z\}\} \mapsto \{x, y, z\}.$$

This is a monad. An algebra over this monad is the same thing as commutative monoid. Recognisable languages over this monad are the same things are regular languages – in the usual sense – which are commutative, see Exercise 11.

If we lift the restriction on finite supports, then we do not get a monad. The problem is with the substitutions: if $f : X \to \{a\}$ is the constant function with an infinite domain, then there is no way to define

$$(\mathsf{T}f)\{\underbrace{x_1, x_2, \ldots}_{\text{infinitely many distinct elements}}\}.$$

The problem is that the above multiset should contain a infinitely many times. To overcome this problem, we would need to allow the multisets to have infinitely many copies of an element. □

Example 16. [Idempotent finite words] Define $\mathsf{T}X$ to be finite words X^*, modulo the equation $ww = w$. This equation can be applied to arbitrary words, e.g.

$$abcabc = abc.$$

The remaining ingredients of the monad are defined in the natural way. An algebra over this monad is the same thing as an idempotent monoid, i.e. a monoid where all elements are idempotent. Green and Rees show that if X is a finite finite set, then $\mathsf{T}X$ is finite[4] . It follows that for every finite alphabet, there are finitely many languages over this alphabet, and all of them are recognisable. □

Example 17. [Powersets] The *powerset monad*, and its variant the *finite powerset monad*, are defined in the same way a the multiset monad, except that we use sets (or finite sets) instead of multisets. The substitutions are defined

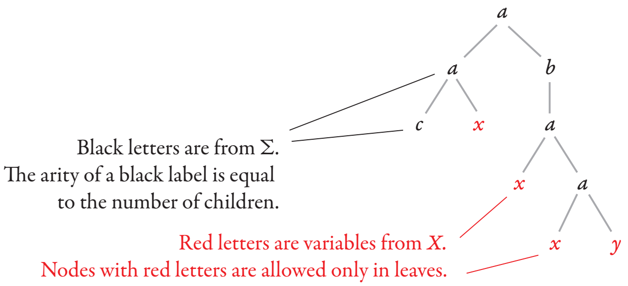
[4] [17] Green and Rees, “On semi-groups in which $x^r = x$”, 1952 , p. 35

via images (in the language of category theory, we use the covariant powerset functor):

$$A \subseteq X \quad \overset{\mathsf{T}f}{\mapsto} \quad \{f(x) : x \in A\} \subseteq Y.$$

The algebras over the finite powerset monad are the same thing as monoids that are commutative and idempotent. If X is a finite set, then both powerset monads generate finite sets; and therefore all languages over finite alphabets are recognisable. □

Example 18. [Terms] Fix a ranked set Σ, i.e. a set where every element has an associated arity in $\{0, 1, \ldots\}$. Define $\mathsf{T}X$ to be the terms over Σ with variables X, i.e. an element of $\mathsf{T}X$ is a tree that looks like this:



Black letters are from Σ.
The arity of a black label is equal
to the number of children.

Red letters are variables from X.
Nodes with red letters are allowed only in leaves.

The unit operation maps $x \in X$ to a term which consists only of x. The substitution $\mathsf{T}f$ is defined in the natural way, by applying f to the variables and leaving the remaining part of the term unchanged. Finally, free multiplication replaces each variable with the corresponding term. It is a simple exercise to check that a T-algebra is the same thing as an *algebra of type* Σ, in the sense of universal algebra[5]. In the terminology of automata theory, both of these notions are the same as deterministic bottom-up tree automata over finite trees, where Σ is the input alphabet[6] . From the above observation it follows that a language $L \subseteq \mathsf{T}X$ is recognisable in the sense of Definition 4.8 if and only if it is a regular tree language in the sense of automata theory[7] , where the input

[5] An algebra of type Σ is defined to be an underlying set A, together with an interpretation which maps each n-ary symbol from Σ to an operation of type $A^n \to A$. See
[30] Sankappanavar and Burris, "A course in universal algebra", 1981 , Definition 1.3

[6] [36] Thatcher and Wright, "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic", 1968 , Section 2

[7] This monad describes finite trees. Finding an algebraic account of languages of infinite trees remains an open problem This problem is discussed in the following papers:
[4] Blumensath, "Regular Tree Algebras", 2018

alphabet is obtained from Σ by adding one letter of arity 0 for each element of X. If the ranked set Σ contains only letters of arity exactly one, then a T-algebra can be seen as a deterministic word automaton with input alphabet Σ, without distinguished initial and final states. □

Example 19. [Linear orders] Define a *chain* over a set X to be a linear order with positions labelled by X, modulo isomorphism of labelled linear orders. For an infinite cardinal κ, a monad T_κ is defined as follows. The set $\mathsf{T}_\kappa X$ consists of chains over X, which have cardinality strictly less than κ. The monad structure is defined in the same way as for $\circ$-words (which are the special case of this construction for where κ is the first uncountable cardinal, which is the same as $\mathfrak{c}$ assuming the Continuum Hypothesis). Nevertheless, we give a more exact description below.

For a function f, the corresponding substitution $\mathsf{T}_\kappa f$ is defined in the natural way, by applying f to the labels in a chain and leaving the positions and ordering unchanged. The unit maps a letter to the one letter chain with that letter. The free multiplication operation is defined using lexicographic products, as follows. Suppose that $w \in \mathsf{T}_\kappa \mathsf{T}_\kappa X$. The positions in the free multiplication of w are pairs (i, j) such that i is a position of w and j is a position in the label $w(i) \in \mathsf{T}_\kappa X$ of position i in the chain w. The label of such a position is inherited from j, and the ordering is lexicographic. The cardinality of the resulting chain is at most κ, since every infinite cardinal satisfies $\kappa = \kappa^2$.

We only prove one of the monad axioms:

$$\begin{tikzcd}
\mathsf{T}_\kappa \mathsf{T}_\kappa \mathsf{T}_\kappa X \arrow[r, "\text{free multiplication on } \mathsf{T}_\kappa X"] \arrow[d, "\mathsf{T}_\kappa(\text{free multiplication on } X)"'] & \mathsf{T}_\kappa \mathsf{T}_\kappa X \arrow[d, "\text{free multiplication on } X"] \\
\mathsf{T}_\kappa \mathsf{T}_\kappa X \arrow[r, "\text{free multiplication on } X"'] & \mathsf{T}_\kappa X
\end{tikzcd}$$

Let $w \in \mathsf{T}_\kappa \mathsf{T}_\kappa \mathsf{T}_\kappa X$. If, in the diagram above, we first go right and then down, then the resulting linear order will have positions of the form $((i, j), k)$, where i is a position in w, j is a position in $w(i)$, and k is a position in $w(i)(j)$. If, in the diagram, we first go down and then right, then we get positions of the form $(i, (j, k))$, where i, j, k satisfy the same conditions as above. In both cases, the tuples of positions are ordered lexicographically, and the label is inherited from k. Therefore

$$((i, j), k) \mapsto (i, (j, k))$$

[5] Bojańczyk and Klin, “A non-regular language of infinite trees that is recognizable by a sort-wise finite algebra”, 2019

is an isomorphism of labelled linear orders, and hence the two outcomes are equal as chains.

If we take $\kappa = \aleph_0$, then we get the monad of finite words. If we take κ to be the first uncountable cardinal, then we get the monad corresponding to $\circ$-words.

If we take κ to be the first cardinal bigger than $\mathfrak{c}$, then T_κ describes chains of cardinality at most $\mathfrak{c}$. In this case, we have the following phenomenon. Recall the powerset construction that was described in Section 3.3. This construction also makes sense for chains of size at most $\mathfrak{c}$. Define $\mathcal{X}$ to be the least class of T_κ-algebras which contains the syntactic algebra of the language "every a is before every b", and which is closed under products and the powerset construction. Using the same proof as for Lemma 3.22, one can show that every MSO definable language $L \subseteq \mathsf{T}_\kappa\Sigma$ is recognised by an algebra from $\mathcal{X}$. Since satisfiability for MSO over the reals is undecidable[8] , it follows that there is no finite way of representing algebras from $\mathcal{X}$. This means that the powerset construction over finite T_κ-algebras is not computable. □

Example 20. Consider a class $\mathcal{X}$ of linear orders which have size bounded by some cardinal κ, and which are closed under free multiplication as defined in the previous example, when viewed as chains over a one letter alphabet. If we restrict the monad from the previous example to chains where the underlying linear order is in $\mathcal{X}$, then we also get a monad. This construction yields the following monads (in all cases, we assume some fixed upper bound on κ on the cardinality, e.g. we can require countability):

- well-founded words (the class of well-founded linear orders);
- scattered words (the class of scattered orders, i.e. those into which one cannot embed the rational numbers)[9].
- dense words (the class which contains two orders: a singleton order for units, and the rational numbers).

□

Example 21. [ω-semigroups] We now describe a monad that corresponds to ω-semigroups, see Definition 3.7. Since an ω-semigroup has two sorts, we leave

[8] [33] Shelah, "The Monadic Theory of Order", 1975 , Theorem 7

[9] Algebras for the monad of countable scattered words are studied in
[29] Rispal and Carton, "Complementation of Rational Sets on Countable Scattered Linear Orderings", 2005

the category of sets, and use instead the category

$$\mathsf{Set}^{\{+,\omega\}}$$

of sets with two sorts $+$ and ω. An object in this category is a set, where every element is assigned exactly one of two sorts, called $+$ and ω. A morphism in this category is any sort-preserving function between sorted sets. We use the name *sorted set* for the objects. We write the objects and morphisms of this category in red, to distinguish them from usual sets and functions. We also use the following notation for sorts:

$$\underbrace{X}_{\text{a sorted set}} = \underbrace{X[+]}_{\substack{\text{elements}\\ \text{of sort } +}} \cup \underbrace{X[\omega]}_{\substack{\text{elements}\\ \text{of sort } \omega}}.$$

We define a monad T over this category as follows. For a sorted set X, the sorted set $\mathsf{T}X$ is defined by:

$$(\mathsf{T}X)[+] = X[+]^+ \qquad \cup \qquad (\mathsf{T}X)[\omega] = (X[+])^*(X[\omega]) \cup (X[+])^\omega.$$

For a morphism $f : X \to Y$, the substitution morphism $\mathsf{T}f$ is defined in the natural way, by applying f to every letter. The unit and free multiplication are defined in the natural way as well. (An element of sort $+$ in $\mathsf{TT}X$ is simply a finite nonempty word of finite nonempty words over $X[+]$, and we can use free multiplication from the monad of finite nonempty words. On sort ω, there are more cases to consider, but the definition is natural as well.) An Eilenberg-Moore algebra over this monad is the same thing as an ω-semigroup, as defined at the end of Section 3.1. □

Example 22. [Vector spaces] Define $\mathsf{T}X$ to be the vector space, over the field of rational numbers, where the basis is X. In other words, elements of $\mathsf{T}X$ are finite linear combinations of elements from X with rational coefficients. For example,

$$3x + 7y - 0.5z \in \mathsf{T}\{x, y, z\}.$$

The action of T on functions is defined by

$$q_1x_1 + \cdots + q_nx_n \quad \overset{\mathsf{T}f}{\mapsto} \quad q_1f(x_1) + \cdots + q_nf(x_n).$$

The unit operation maps $x \in X$ to the corresponding basis vector, and free multiplication is defined in the natural way, as illustrated in the following example:

$$3(4x + 0.5y) - 0.2(5x - 0.1y) \quad \mapsto \quad 12x + 1.5y - x + 0.02y = 11x + 1.48y.$$

An algebra A over this monad, with multiplication μ, is also equipped with the structure of a vector space, because we can add elements

$$a + b \overset{\text{def}}{=} \mu(a + b)$$

and multiply them by scalars $q \in \mathbb{Q}$

$$qa \overset{\text{def}}{=} \mu(qa).$$

If $B \subseteq A$ is a basis for the vector space A, then the algebra A is isomorphic to $\mathsf{T}B$. Therefore, over this monad, every algebra is isomorphic to a free algebra. □

Example 23. [Algebra over a field] Define $\mathsf{T}X$ to be finite linear combinations of words in X^*, with rational coefficients. For example,

$$2xyx + -2xx + 0.5xyz \in \mathsf{T}\{x, y, z\}.$$

In other words, $\mathsf{T}X = \mathsf{T}_{\text{vec}}X^*$, where T_{vec} is the monad of vector spaces from Example 22 and X^* is the monad of finite words[10]. On functions, the monad acts as follows

$$q_1x_1 + \cdots + q_nx_n \quad \overset{\mathsf{T}f}{\mapsto} \quad q_1f^*(x_1) + \cdots + q_nf^*(x_n),$$

where f^* is the substitutions in the monad of finite words. The unit maps x to the linear combination which has the one-letter word x with coefficient 1. The free multiplication is defined like for polynomials, but the variables are not commuting, e.g.:

$$3(4x - 2y)(2xy - yy) \quad \mapsto \quad 24xxy - \underbrace{12xyy + 12yxy}_{\text{this is not } 0} + 6yyy.$$

Every algebra over this monad has the structure of a vector space over the rationals, but there is more structure (e.g. one can multiply two elements of the algebra)[11].

What is a recognisable colouring over this monad? In the context of this monad (and also the simpler monad of vector spaces from Example 22), it is more useful to work with different notions of "finite algebra" and "algebra colouring": instead of finite algebras, one should consider finite dimensional

[10] This is an example of a composite monad that arises via a distributive law of two monads. This type of construction was first described in

[1] Appelgate et al., *Seminar on triples and categorical homology theory*, 1969 , Chapter on distributive laws

[11] Algebras over this monad are known as "algebras over the field of rational numbers", but we avoid this terminology due to the over-loading of "algebra over".

algebras (i.e. those where the underlying vector space has finite dimension), and instead of algebra colourings one should consider linear maps to vector spaces. Under these adapted definitions, the algebra colourings recognised by finite algebras are exactly those which are recognised by weighted automata, see Exercise 123. □

Exercises

Exercise 111. Consider the monad T_Σ from Example 18, where Σ is some ranked set (possibly infinite). Let X be some possibly infinite set of variables, and consider a set

$$\underbrace{\mathcal{E} \subseteq (\mathsf{T}X) \times (\mathsf{T}X)}_{\text{elements of this set will be called identities}}.$$

For a set Y, define $\sim$ to be the least congruence on $\mathsf{T}\Sigma$ that satisfies

$$(\mathsf{T}f)(t_1) \sim (\mathsf{T}f)(t_2) \qquad \text{for every } (t_1, t_2) \in \mathcal{E} \text{ and } f : X \to Y.$$

(This congruence can be obtained by intersecting all congruences with the above property.) Define a new monad as follows: $\mathsf{T}Y$ is equal to $\mathsf{T}_\Sigma Y$ modulo $\sim$, and the remaining components of the monad are defined in the natural way. Show that this is a monad.

Exercise 112. For monads S and T, define a monad morphism from S to T to be a family of functions

$$\{\delta_X : \mathsf{S}X \to \mathsf{T}X\}_{X \text{ is a set}}$$

which is subject to the axioms in Figure 4.1. Using the monads from Section 4.2, give five examples of monad morphisms, and five examples of pairs of monads which do not allow a monad morphism.

Exercise 113. We say that $w \in \mathsf{T}\Sigma$ is *regular* if $\{w\}$ is a recognisable language. Find a monad where there are no regular elements. (Hint: it appears in this section.)

Exercise 114. What is the monad for rings (commutative and non-commutative)? Semirings?

$$\begin{tikzcd}
\mathsf{S}X \arrow[r, "\delta_X"] \arrow[d, "\mathsf{S}f"'] & \mathsf{T}X \arrow[d, "\mathsf{T}f"] \\
\mathsf{S}Y \arrow[r, "\delta_Y"'] & \mathsf{T}Y
\end{tikzcd}$$

$$\begin{tikzcd}
X \arrow[d, "\text{unit in } \mathsf{S}"'] \arrow[dr, "\text{unit in } \mathsf{T}"] & \\
\mathsf{S}X \arrow[r, "\delta_X"'] & \mathsf{T}X
\end{tikzcd}$$

$$\begin{tikzcd}
\mathsf{SS}X \arrow[r, "\mathsf{S}\delta_x"] \arrow[d, "\text{free multiplication in } \mathsf{S}"'] & \mathsf{ST}X \arrow[r, "\delta_{\mathsf{T}X}"] & \mathsf{TT}X \arrow[d, "\text{free multiplication in } \mathsf{T}"] \\
\mathsf{S}X \arrow[rr, "\delta_X"'] & & \mathsf{T}X
\end{tikzcd}$$

Figure 4.1 These three diagrams should commute for all sets X and all functions $f : X \to Y$. The top diagram says that $\{\delta_X\}_X$ is a natural transformation, while the bottom two diagrams say that it is compatible with unit and free multiplication.

Exercise 115. Consider the following monad T. The set $\mathsf{T}X$ is the set of ω-words X^ω, and the substitution $\mathsf{T}f : \mathsf{T}X \to \mathsf{T}Y$ is defined coordinate-wise. The unit is $x \mapsto x^\omega$, and free multiplication is defined by

$$w \in \mathsf{TT}X \qquad \mapsto \qquad (i \mapsto \underbrace{(w[i])[i]}_{\substack{i\text{-th letter of}\\ \text{of } i\text{-th letter of } w}}).$$

Show that this is a monad. Also, show that a language $L \subseteq \mathsf{T}\Sigma$ is recognisable if and only if it is clopen in the sense of Exercise 65.

Exercise 116. Let T be the monad of countable well founded. Show that a finite algebra with universe S is uniquely determined by the operations

$$\underbrace{ab}_{\substack{\text{binary}\\ \text{product}\\ S^2 \to S}} \qquad \underbrace{a^\omega}_{\substack{\text{product of}\\ aaa\cdots\\ S \to S}}$$

Exercise 117. Consider the monad from Example 116. Show that a language $L \subseteq \mathsf{T}\Sigma$ is definable in first-order logic (in the ordered model) if and only if it is recognised by a finite T-algebra S where the underlying semigroup is aperiodic.

Exercise 118. Consider the monad from Example 116. Consider regular expressions defined by the usual operators, plus L^ω. Show that these expressions do not describe all recognisable languages.

Exercise 119. Consider the monad and regular expressions from Exercise 118. Given an effective condition on finite algebras which corresponds exactly to the Boolean combinations of regular expressions.

Exercise 120. Let T be the monad of countable scattered chains. Show that a finite algebra with universe S is uniquely determined by the operations

$$\underbrace{ab}_{\substack{\text{binary}\\ \text{product}\\ S^2 \to S}} \qquad \underbrace{a^\omega}_{\substack{\text{product of}\\ aaa\cdots\\ S \to S}} \qquad \underbrace{a^{\omega*}}_{\substack{\text{product of}\\ \cdots aaa\\ S \to S}}$$

Exercise 121. Consider the monads from Examples 116 and 120. In which of these monads is first-order logic (over ordered models) equivalent to star-free expressions?

Exercise 122. Consider the monad from Example 120. Which class of languages corresponds to aperiodicity (of the semigroup underlying the T-algebra)?

Exercise 123. A weighted automaton over the rationals consists of:

$$\underbrace{\Sigma}_{\substack{\text{input alphabet,}\\ \text{which is a}\\ \text{finite set}}} \quad \underbrace{S}_{\substack{\text{state space,}\\ \text{which is a}\\ \text{vector space of}\\ \text{finite dimension}}} \quad \underbrace{s_0 \in S}_{\text{initial state}} \quad \underbrace{\{\delta_a : S \to S\}_{a\in\Sigma}}_{\substack{\text{state updates,}\\ \text{which are}\\ \text{linear maps}}} \quad \underbrace{F : S \to \mathbb{Q}}_{\substack{\text{final function,}\\ \text{which is a}\\ \text{linear map}}}$$

The semantics of this automaton is a function of type $\Sigma^* \to \mathbb{Q}$ defined as follows. Given an input word, do the following: start with the initial state, apply the state update for the first letter, then the state update for the second letter, and so on for all letters, and at the end apply the final function. The semantics of a weighted automaton can also be naturally extended to finite linear combinations of words over Σ, i.e. to elements of the $\mathsf{T}\Sigma$ as in Example 23.

Show that $L : \mathsf{T}\Sigma \to \mathbb{Q}$ is recognised by a weighted automaton if and only if it is recognised by a finite dimensional T-algebra.

4.3 Syntactic algebras

In this section, we show that every recognisable algebra colouring has a syntactic homomorphism, i.e. a recognising homomorphism that stores the minimal amount of information. This can be viewed as a monad version of the Myhill-Nerode theorem.

Definition 4.9 (Syntactic homomorphism)**.** The *syntactic homomorphism* of an algebra colouring $\lambda : A \to U$ is any surjective homomorphism

$$h : A \to B$$

which recognises λ and which is minimal in the sense explained in the following quantified diagram

$$\begin{tikzcd} \forall\exists! & A \arrow[r, two heads, "h"] \arrow[dr, two heads, "\text{surjective homomorphism that recognises } \lambda"'] & B \\ & & C \arrow[u, two heads, "\text{homomorphism}"'] \end{tikzcd}$$

The algebra used by the syntactic homomorphism is called the *syntactic algebra*. The syntactic algebra, if it exists, is unique up to isomorphism of algebras. Also the syntactic homomorphism is unique in the following sense: every two syntactic homomorphisms will have the same kernel (equivalence relation on A that identifies two elements with the same homomorphic image). This kernel is called the *syntactic congruence* of λ.

We are mainly interested in the case where λ describes a language, i.e. A is a free algebra, and there are two colours.

There are two main results in this section. The first one, Theorem 4.11, says that if an algebra colouring is recognisable, then it has a syntactic homomorphism. The second one, Theorem 4.16, says that a monad is finitary (roughly speaking, this means that all structures described by the monad are finite) if and only if every (not necessarily recognisable) algebra colouring has a syntactic homomorphism. To illustrate these theorems, we begin with an example of an algebra colouring that does not have a syntactic homomorphism. By Theorem 4.11, this colouring is necessarily not recognisable.

Example 24. Consider the monad of ∘-words. Define L to be the set of ∘-words over a one letter alphabet which contain every finite word as an infix. More formally,

$$L = \{w \in \{a\}^\circ : a^n \text{ is an infix of } w \text{ for every } n \in \{0, 1, \ldots\}.\},$$

which is a language over alphabet a, and therefore a special case of an algebra colouring. This language is not recognisable, because all finite words must have different images under any recognising homomorphism (we leave this as an exercise for the reader). We will show that L does not have a syntactic homomorphism. For $n \in \{1, 2, \ldots\}$ define

$$w_n = (\text{shuffle of } \{a\}) \cdot a^n \cdot (\text{shuffle of } \{a\})$$

and define h_n to be the function

$$w \in \{a\}^\circ \quad \mapsto \quad \begin{cases} w_n & \text{if } w = w_{n+1} \\ w & \text{otherwise.} \end{cases}$$

This function is compositional for every n, and therefore it can be viewed as a homomorphism. If there would be a syntactic homomorphism, then it would need to factor through h_n. Since h_n gives the same result for w_n and w_{n+1}, therefore the same would have to be true for the syntactic homomorphism. Therefore, the syntactic homomorphism h, if it existed, would need to satisfy

$$h(w_1) = h(w_2) = \cdots .$$

By associativity, we would have

$$h(\underbrace{w_1 w_1 w_1 \cdots}_{\notin L}) = h(\underbrace{w_1 w_2 w_3 \cdots}_{\in L}).$$

Therefore the syntactic homomorphism does not exist. □

As we will see later in this section, the monad of ∘-words from the above example is not finitary.

4.3.1 Terms, polynomials and congruences

To prove the existence of syntactic homomorphisms, we will use classical concepts from universal algebra such as a terms, polynomials and congruences. We begin by defining these notions.

Terms and polynomials. If X is a set, then a *term over variables X* is defined to be any element of $\mathsf{T}X$. Given an algebra A, a term is interpreted as the following operation

$$\eta \in A^X \quad \mapsto \quad \text{multiply } (\mathsf{T}\eta)(t) \text{ in } A.$$

We write t^A for the above operation; and we use the name *variable valuation* for η. An operation of this form is called a *term operation* in the algebra A. We distinguish between a term (which can be viewed as syntax) and the term operation that it generates in a given algebra (which can be viewed as semantics). If a term uses a finite set of variables $\{x_1, \ldots, x_n\}$, then for $a_1, \ldots, a_n \in A$ we write

$$t^A(a_1, \ldots, a_n)$$

for the result of applying t^A to the variable valuation $x_i \mapsto a_i$.

Example 25. Consider the monad of finite words. The word xy is a term, and the term operation induced it in an algebra (which is the same thing as a monoid) is binary multiplication. The operation induced by the term ε, which has an empty set of variables, is the constant that represents the monoid identity. Another example of a term operation is squaring, which is given by the term xx. A non-example is the idempotent power operation $a \mapsto a^!$. This is not a term operation, because the number ! depends on the algebra at hand (also, this number does not exist in some infinite algebras).

In the monad of ∘-chains, the Läuchli-Leonard operations x^ω and $x^{\omega *}$ are also term operations. To model the remaining Läuchli-Leonard operation, namely shuffling, we use an infinite family of terms, with the n-th one being the shuffle $\{x_1, \ldots, x_n\}$. □

A *polynomial* in an algebra is like a term except that some variables are fixed elements of the algebra. More formally, a *polynomial with variables X in an algebra A* is defined to be any element $t \in \mathsf{T}(A + X)$. The semantics of such a polynomial is the operation

$$\eta \in A^X \qquad \mapsto \qquad t^A(\underbrace{\mathrm{id}_A + \eta}_{\eta \text{ extended by the identity on } A}).$$

Such an operation is called a *polynomial operation* in A. We will be mainly interested in unary polynomial operations, which are functions of type $A \to A$ that arise from polynomials with one variable.

Congruences. Consider an equivalence relation $\sim$ on the underlying set in an algebra A. It is not hard to see that the following conditions are equivalent:

- $\sim$ is the kernel of some homomorphism from A to some algebra B;
- the function which maps $a \in A$ to its equivalence class is compositional;
- $\sim$ is compatible with every term operation $f : A^X \to A$, which means:

$$\underbrace{\eta_1 \sim \eta_2}_{\substack{\eta_1(x) \sim \eta_2(x) \\ \text{for every } x \in X}} \quad \Rightarrow \quad f(\eta_1) \sim f(\eta_2) \qquad \text{for every } \eta_1, \eta_2 \in A^X.$$

Equivalence of the first two conditions is Lemma 4.7, and equivalence of the second two conditions follows by unravelling the definition of compositionality. We say that $\sim$ is *congruence* if it satisfies any of the above conditions. Every congruence induces a quotient algebra, where the universe is equivalence classes.

Contextual equivalence. To construct the syntactic homomorphism, we will use contextual equivalence, which is defined similarly as in the usual proof of the Myhill-Nerode Theorem, see e.g. Theorem 1.7. The only difference is that instead of two-sided contexts (as in monoids and semigroups), we use unary polynomial operations.

Definition 4.10 (Contextual equivalence)**.** Define *contextual equivalence* of an algebra colouring $\lambda : A \to U$ to be the equivalence relation on A, which identifies $a, a' \in A$ if they have the same image under $\lambda \circ f$ for every unary polynomial operation $f : A \to A$.

The following example shows that, in general, contextual equivalence is not a congruence.

Example 26. Consider the language L from Example 24. We show that contextual equivalence for this language (viewed as a colouring with values "yes" and "no") is not a congruence. A unary polynomial operation in $\{a\}^\circ$ corresponds to a $\circ$-word over alphabet $\{a, x\}$. From the point of view of L, there are two kinds of unary polynomials operations. If f is a unary polynomial such that the corresponding $\circ$-word over $\{a, x\}$ contains a^n as an infix for every n, then $f(w) \in L$ for every $w \in \{a\}^\circ$. Otherwise, if the corresponding $\circ$-word does not contain a^n as an infix for some n, then $f(w) \in L \Leftrightarrow w \in L$. These observations imply that contextual equivalence for L has two equivalence classes: namely L and its complement. In particular, contextual equivalence is not a congruence, since otherwise L would be recognisable. □

4.3.2 Syntactic homomorphisms for recognisable colourings

We now state the first result about syntactic homomorphisms, which says that they always exist for algebra colourings that are recognisable.

Theorem 4.11. *Let* T *be a monad in the category of sets. Every recognisable algebra colouring has a syntactic homomorphism.*

If $\sim$ is a equivalence relation on the underlying set of an algebra A, then we say that $\sim$ recognises an algebra colouring $\lambda : A \to U$ if all equivalence classes are monochromatic, i.e.

$$a_1 \sim a_2 \quad \text{implies} \quad \lambda(a_1) = \lambda(a_2).$$

An algebra colouring is recognisable if and only if there is equivalence relation that recognises it, is a congruence, and has finitely many equivalence classes.

We say that an equivalence relation $\approx$ refines an equivalence relation $\sim$ if $a \approx b$ implies $a \sim b$. If we view equivalence relations as sets of pairs, this is the same as set inclusion. The following lemma shows that contextual equivalence refines every recognising congruence. (Note that the lemma does not say that contextual equivalence is a congruence. This will be proved later, using the assumption on recognisable.)

Lemma 4.12. *For every algebra colouring* λ*, contextual equivalence of* λ *is an equivalence relation that refines every congruence that recognises* λ*.*

Proof Let f be a unary polynomial operation in the underlying algebra. Let $\approx$ be a congruence that recognises λ. As a congruence, $\approx$ is compatible with every term operation, and therefore it is compatible with every unary polynomial operation (see Exercise 124). This means that if $a \approx b$, then also $f(a) \approx f(b)$, and therefore $f(a)$ has the same colour under λ as $f(b)$, by assumption that $\approx$ is a recognising congruence. By arbitrary choice of f, we have established that $a \approx b$ implies $a \sim b$. □

We now use the assumption on recognisability to prove that contextual equivalence is a congruence.

Lemma 4.13. *If an algebra colouring* λ *is recognisable, then its contextual equivalence is a congruence.*

Proof Let $\lambda : A \to U$ be an algebra colouring, and let $\sim$ be its contextual equivalence relation. We will show that $\sim$ is a congruence in three steps.

(1) In the first step, we show that $\sim$ is compatible with all unary polynomial operations. The key observation is that unary polynomial operations are closed

under composition. Indeed, consider two unary polynomial operations

$$A \xrightarrow{f} A \xrightarrow{g} A.$$

Take the unary polynomial in $\mathsf{T}(A + \{x\})$ that defines f, and for the variable x substitute the unary polynomial in $\mathsf{T}(A + \{x\})$ that defines g. The resulting unary polynomial defines the composition $g \circ f$.

Equipped with this observation, we prove that $\sim$ is compatible with all unary polynomial operations. Let f be a unary polynomial operation. We need to show

$$a_1 \sim a_2 \quad \text{implies} \quad f(a_1) \sim f(a_2).$$

The assumption of the implication says that a_1, a_2 have the same image under all functions of the form $\lambda \circ g$, where g is a unary polynomial operation, while the conclusion of the implication says that a_1, a_2 have the same image under all functions of the form $\lambda \circ g \circ f$, where g is a unary polynomial operation. Because $g \circ f$ is a unary polynomial operation, the implication follows.

(2) We now show that $\sim$ is compatible with all term operations that have finitely many variables. Consider an n-ary term operation $f : A^n \to A$. We need to show

$$\bigwedge_{i \in \{1,\ldots,n\}} a_i \sim b_i \quad \text{implies} \quad f(a_1, \ldots, a_n) \sim f(b_1, \ldots, b_n). \tag{4.8}$$

Assume the assumption of the above implication. For $i \in \{1, \ldots, n\}$ consider the following unary polynomial operation

$$f_i(x) = f(a_1, \ldots, a_{i-1}, x, b_{i+1}, \ldots, b_n).$$

Because $a_i \sim b_i$ and $\sim$ is compatible with all unary polynomial operations,

$$f(a_1, \ldots, a_{i-1}, b_i, \ldots, b_n) = f_i(a_i) \sim f_i(b_i) = f(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n).$$

By iterating the above for $i = 1, \ldots, n$, we get the conclusion of (4.8).

(3) Finally, we show that $\sim$ is compatible with all term operations, possibly with infinitely many variables. In this step, we use the assumption that λ is recognisable. Consider a term $t \in \mathsf{T}X$. We need to show that every valuations $\eta_1, \eta_2 \in A^X$ satisfy

$$\eta_1 \sim \eta_2 \quad \text{implies} \quad t^A(\eta_1) \sim t^A(\eta_2).$$

Assume the assumption of the above implication. Because λ is recognisable, there is some congruence $\approx$ on A which recognises λ and has finitely many equivalence classes. Consider a choice function $\tau : A \to A$ which maps each

$\approx$-equivalence class in A to a chosen element in that equivalence class. Let $i \in \{1, 2\}$. Because the choice function respects $\approx$-equivalence, we have

$$\eta_i \approx \tau \circ \eta_i. \tag{4.9}$$

Since $\approx$ is a congruence,

$$t^A(\eta_i) \approx t^A(\tau \circ \eta_i). \tag{4.10}$$

Since each of the valuations $\tau \circ \eta_i$ for $i = 1, 2$ has finite image, we can find a finite set of variables Y (think of pairs of equivalence classes of $\approx$) and functions δ and γ_i which make the following diagram commute:

$$\begin{tikzcd}
& X \arrow[dl, "\tau\circ\eta_1"'] \arrow[d, "\sigma"'] \arrow[dr, "\tau\circ\eta_2"] & \\
A & Y \arrow[l, "\gamma_1"] \arrow[r, "\gamma_2"'] & A
\end{tikzcd} \tag{4.11}$$

Define $s \in \mathsf{T}Y$ to be the term that results from t by the substitution $\mathsf{T}\sigma$. By definition,

$$s^A(\gamma_i) = t^A(\gamma_i \circ \sigma) = t^A(\tau \circ \eta_i) \overset{(4.10)}{\approx} t^A(\eta_i).$$

By the first step of the proof, $\approx$ refines $\sim$. Therefore, to finish the proof, it remains to show

$$s^A(\gamma_1) \sim s^A(\gamma_2).$$

Since s is a term with finitely many variables, and we have already established that $\sim$ is compatible with terms that have finitely many variables, it is enough to show $\gamma_1 \sim \gamma_2$. This is because

$$\tau \circ \eta_1 \sim \eta_1 \sim \eta_2 \sim \tau \circ \eta_2.$$

In the above, the first and last equivalence are due to (4.9) and the fact that $\approx$ refines $\sim$. Since the first valuation in the above is equal to $\gamma_1 \circ \sigma$, and the last one is equal to $\gamma_2 \circ \sigma$, it follows that $\gamma_1 \sim \gamma_2$.

We have proved that $\sim$ is a congruence. □

We now complete the proof of Theorem 4.11. Let $\sim$ be contextual equivalence of a recognisable algebra colouring $\lambda : A \to U$. By Lemma 4.13, $\sim$ is a congruence. Because $\sim$ is a congruence, the quotient function, call it h, is a homomorphism. Since the identity function is a unary polynomial operation, it follows that equivalence classes of $\sim$ are monochromatic, and therefore h recognises λ.

It remains to show that every other recognising homomorphism factors through h. Suppose that

$$g : A \to B$$

is a surjective homomorphism that recognises λ. The kernel of g is a congruence that recognises λ, and therefore the kernel of g refines $\sim$. In other words, g factors through h, i.e. there is some function f such that $g = f \circ h$. The last thing to show is that f is in fact a homomorphism. This is shown in the following lemma, with C being the quotient $A_{/\sim}$.

Lemma 4.14. *Let A, B, C be algebras, let g, h be surjective homomorphisms, and let f be a function which makes the following diagram commute.*

$$\begin{tikzcd}
 & B \arrow[dd, "f"] \\
A \arrow[ru, "g"] \arrow[rd, "h"'] & \\
 & C
\end{tikzcd}$$

Then f is a homomorphism.

Proof Consider the following diagram:

$$\begin{tikzcd}
\mathsf{T}B \arrow[rrr, "\text{multiplication in } B"] \arrow[dd, "\mathsf{T}f"'] & & & B \arrow[dd, "f"] \\
 & \mathsf{T}A \arrow[lu, "\mathsf{T}g"'] \arrow[ld, "\mathsf{T}h"] \arrow[r, "\text{multiplication in } A"] & A \arrow[ru, "g"] \arrow[rd, "h"'] & \\
\mathsf{T}C \arrow[rrr, "\text{multiplication in } C"'] & & & C
\end{tikzcd}$$

The upper and lower faces commute because g and h are homomorphisms, and the left and right faces commute by definition of f. Since all arrows in the diagram are surjective, it follows that the perimeter of the diagram commutes, which means that f is a homomorphism. □

4.3.3 Finitary monads

In the monad of finite words, every language – even not necessarily recognisable – has a syntactic homomorphism. In the monad of ∘-words, this is no longer true, as witnessed by Example 24. What is the difference?

The difference is that every finite word uses only a finite subset of the alphabet, which is no longer true for $\circ$-words. This is made precise by the following definition.

Definition 4.15 (Finitary elements and monads.)**.** Let T be a monad in the category of sets. We say that an element $t \in \mathsf{T}X$ is *finitary*

$$t = (\mathsf{T}f)(t) \qquad \text{for some } f : X \to X \text{ with finite image.}$$

We say that T is finitary if for every X, all elements of $\mathsf{T}X$ are finitary.

For example, the monad of finite words is finitary, while the monads of $\circ$-words is not. The following theorem shows that finitary monads are exactly those monads where all algebra colourings have syntactic homomorphisms.

Theorem 4.16. *Let* T *be a monad in the category of sets. Then* T *is finitary if and only if every algebra colouring has a syntactic homomorphism.*

Proof We begin with the left-to-right implication. Suppose that T is finitary, and let $\lambda : A \to U$ be some algebra colouring, not necessarily recognisable. We use the same proof as in Theorem 4.11. The only place where the proof used the assumption on recognisability was in step (3) of Lemma 4.13, where contextual equivalence was shown to be compatible with all term operations. If the monad is finitary, then every term operation uses finitely many variables, and therefore step (3) of the proof follows immediately from step (2), without any need for invoking recognisability.

We now prove the converse implication. Fix some infinite set X. We will show that all elements of X are finitary. Let ${\color{red}X}$ be a disjoint copy of X. For a finite subset ${\color{red}Y} \subseteq {\color{red}X}$, define

$$f_{\color{red}Y} : X + {\color{red}X} \to X + {\color{red}X}$$

to be the function which maps all elements of ${\color{red}Y}$ to their corresponding black copies, and which is the identity on the remaining elements. Define $\sim_{\color{red}Y}$ to be the equivalence relation on $\mathsf{T}(X + {\color{red}X})$ which is the kernel of the homomorphism

$$\mathsf{T}f_{\color{red}Y} : \mathsf{T}(X + {\color{red}X}) \to \mathsf{T}(X + {\color{red}X})$$

The mapping ${\color{red}Y} \mapsto \sim_{\color{red}Y}$ is monotone with respect to inclusion:

$${\color{red}Y} \subseteq {\color{red}Z} \quad \text{implies} \quad \sim_{\color{red}Y} \subseteq \sim_{\color{red}Z} .$$

In the conclusion of the above implication, we view equivalence relations as sets of pairs; under this view smaller equivalence relations refine bigger ones. Viewing equivalences relations as sets of pairs, define $\sim$ to be the union of the equivalences $\sim_{\color{red}Y}$, ranging over all finite subsets ${\color{red}Y} \subseteq {\color{red}X}$. This is an equivalence relation; transitivity follows from the monotonicity property above.

Consider the two natural injections

$$\begin{tikzcd} X + {\color{red}X} & X \arrow[l, "f"'] \arrow[r, "{\color{red}f}"] & X + {\color{red}X}, \end{tikzcd}$$

such that f maps all elements to their black copies, and ${\color{red}f}$ maps all elements to their red copies. By assumption on the monad, there is a syntactic homomorphism for the quotient function of $\sim$, call it h. For every $x \in X$, the units of x and ${\color{red}x}$ are equivalent under the congruence $\sim_{\{{\color{red}x}\}}$. Since the latter is a congruence that recognises $\sim$, it follows that the units of x and ${\color{red}x}$ must have the same image under h. By arbitrary choice of x, we see that

$$h \circ \mathsf{T}f = h \circ \mathsf{T}{\color{red}f}. \tag{4.12}$$

We now prove that every $t \in \mathsf{T}X$ is finitary. Fix t. Since h recognises $\sim$, the equality (4.12) implies that

$$(\mathsf{T}f)(t) \sim (\mathsf{T}{\color{red}f})(t)).$$

By definition of $\sim$, there must be some finite ${\color{red}Y}$, which depends on t, such that

$$(\mathsf{T}f)(t) \sim_{\color{red}Y} (\mathsf{T}{\color{red}f})(t)) \tag{4.13}$$

Choose an element of Y and let

$$g : X + {\color{red}X} \to X$$

be the function which is the identity on X and sends all red elements to the chosen element of Y. We have

$$\begin{aligned} t &= && \text{(because } g \circ f \text{ is the identity on } X\text{)} \\ (\mathsf{T}(g \circ f))(t) &= && \text{(because } f_Y \circ f = f\text{)} \\ (\mathsf{T}(g \circ f_Y \circ f))(t) &= && \text{((4.13))} \\ (\mathsf{T}(g \circ f_Y \circ {\color{red}f}))(t) & && \end{aligned}$$

The image of the function $g \circ f_Y \circ {\color{red}f}$ is contained in Y, and therefore we have established that t is finitary. □

Exercises

Exercise 124. Fix a monad. Let A be an algebra, and let $p : A^X \to A$ be a polynomial operation. Show that for every homomorphism $h : A \to B$ there is a polynomial operation $p^h : B^X \to B$ which commutes with the homomorphism in the following sense:

$$\begin{tikzcd}
A^X \arrow[r, "p"] \arrow[d, "h^X"'] & A \arrow[d, "h"] \\
B^X \arrow[r, "p^h"'] & A
\end{tikzcd}$$

Exercise 125. Consider an algebra colouring $\lambda : A \to U$. Consider the family of congruences on A that recognise U, ordered by inclusion. Show that this family is a lattice, i.e. every two elements have a greatest lower bound and also a least upper bound.

Exercise 126. Consider the monad from Example 23. Show that if $\lambda : \mathsf{T}\Sigma \to \mathbb{Q}$ is recognised by a weighted automaton, then the same is true for the syntactic homomorphism.

Exercise 127. Let T be a monad in the category of sets. Show that if every algebra colouring with two colours has a syntatic homomorphism, then every algebra colouring with an arbitrary number of colours has a syntactic homomorphism.

Exercise 128. Give an example of a monad T which is not finitary, but such that all elements of $\mathsf{T}X$ are finitary for countable X.

Exercise 129. Let S be a finite set of sort names, and consider the category

$$\mathsf{Set}^S$$

of S-sorted sets with sort-preserving functions. Prove Theorem 4.11 for monads over this category.

Exercise 130. Consider a category of sorted sets, as in the previous exercise, but with infinitely many sort names. Define a finite algebra to be one that is finite on every sort. Show that Theorem 4.11 fails.

Exercise 131. Show that a monad in the category of sets is finitary if and only if it arises as a result of the construction described in Exercise 111.

Exercise 132. Let T be a monad in the category of sets, and consider a set of terms $\mathscr{B}$, each one using finitely many variables. We say that $\mathscr{B}$ is a *term basis* if for every finite algebra A and subset $\Gamma \subseteq A$, the sub-algebra generated by Γ is equal to the least subset of A that contains Γ and which is closed under applying terms from $\mathscr{B}$. Show that if $\mathscr{B}$ is a term basis, then a finite algebra A is uniquely determined by its $\mathscr{B}$-multiplication tables, which is the family of term operations $\{t^A\}_{t \in \mathscr{B}}$.

Exercise 133. Let T be a monad which has a finite term basis $\mathscr{B}$. Show that given the $\mathscr{B}$-multiplication tagbles in an algebra A, one can compute the $\mathscr{B}$-multiplication tables of the powerset algebra $\mathsf{P}A$ (as defined in Exercise 107, assuming that $\mathsf{P}A$ is indeed an algebra).

Exercise 134. Find a notion of *computable term basis* which generalises the previous exercise so as to capture the Läuchli-Leonard operations in the monad of $\circ$-words.

Exercise 135. Recall the notion of *regular elements* from Exercise 113. Show that if t is regular, then it is finitary.

Exercise 136. Assume that the regular elements, as considered in the previous exercise, are closed under free multiplication in the following sense: if $t \in \mathsf{T}X$ is a regular term operation, and $\eta : X \to \mathsf{T}Y$ is a valuation of its variables that uses only regular elements, then $t^{\mathsf{T}Y}(\eta)$ is a regular element. Under these assumptions, define a monad of regular elements.

4.4 The Eilenberg Variety Theorem

In Chapter 2, we proved several theorems of the kind

$$\text{class of languages} \qquad \sim \qquad \text{class of semigroups.}$$

For example, a language of finite words is definable in first-order logic if and only if it is recognised by an aperiodic semigroup. In this section, we prove a result, originally due to Eilenberg, which says that every class of languages with good closure properties will correspond to a class of algebras with good closure properties. Eilenberg proved the theorem for monoids and semigroups, but the proof, as presented below, is the same in the abstract setting of monads.

The classes with good closure properties will be called varieties, in analogy with the varieties that appear in Birkhoff's theorem from universal algebra. There will be two kinds of varieties: for algebras and for languages. We begin with the algebras.

Definition 4.17 (Algebra variety)**.** Let T be a monad in the category of sets. An *algebra variety* is a class $\mathscr{A}$ of finite T-algebras with the following closure properties:

- *Quotients.* If $\mathscr{A}$ contains A, then it contains every quotient of A.
- *Sub-algebras.* If $\mathscr{A}$ contains A, then it contains every sub-algebra of A.
- *Products.* If $\mathscr{A}$ contains A and B, then it contains $A \times B$.

Example 27. Consider the monad of finite words, where algebras are monoids. Examples algebra varieties include: finite groups, finite aperiodic monoids, finite infix trivial monoids, or finite prefix trivial monoids. □

Example 28. Here is a non-example. Consider the monad of nonempty finite words, where algebras are semigroups. The class of monoids (i.e. semigroups which have an identity element) is not an algebra variety, because it is not closed under sub-algebras. □

Example 29. Consider a monad T. Define an *identity* to be a pair of terms $s, t \in \mathsf{T}X$ over a common set of variables. An algebra A is said to satisfy the identity if

$$s^A(\eta) = t^A(\eta) \qquad \text{for every } \eta \in A^X.$$

The class of finite algebras that satisfy a given identity (more generally, all identities in a given set of identities) is easily seen to be an algebra variety. For example, the algebra variety of commutative semigroups arises from the identity

$$xy = yx.$$

Unlike in Birkhoff's theorem, some algebra varieties do not arise this way. For example, varieties discussed in Example 27 do not arise from (even possibly infinite sets of) identities[12] □

[12] If we use the more general form of *profinite identities*, then every variety can be axiomatised equationally, which was shown in:
[28] Reiterman, "The Birkhoff theorem for finite algebras", 1982 , Theorem 3.1.

We now describe language varieties. In Eilenberg's original formulations, this is a class of regular languages that is closed under Boolean combinations, inverse images of homomorphisms, and inverse images of operations of the form

$$w \in \Sigma^+ \mapsto v_1 w v_2 \in \Sigma^+ \qquad \text{for fixed } v_1, v_2 \in \Sigma^*.$$

In the more abstract setting of monads, the role of these operations will be played by unary polynomial operations, as described in the following definition.

In the following definition, by recognisable languages we mean recognisable subsets of free algebras.

Definition 4.18 (Language variety)**.** Let T be a monad in the category of sets. A *language variety* is a class of recognisable languages with the following closure properties:

- *Boolean combinations.* $\mathscr{L}$ is closed under Boolean combinations, including complementation.
- *Inverses of homomorphisms.* If $h : \mathsf{T}\Sigma \to \mathsf{T}\Gamma$ is a homomorphism of free algebras, then $\mathscr{L}$ is closed under inverse images of h.
- *Inverses of unary polynomials.* If $f : \mathsf{T}\Sigma \to \mathsf{T}\Sigma$ is a unary polynomial operation in a free algebra $\mathsf{T}\Sigma$, then $\mathscr{L}$ is closed under inverse images of f.

Example 30. Consider the monad of finite words, where algebras are monoids. The languages definable in first-order logic are a language variety. Closure under Boolean combinations is immediate, because we are dealing with a logic. Closure under inverse images of homomorphism or unary polynomials can be proved using Ehrenfeucht-Fraïssé games: if f is either a homomorphism or a unary polynomial operation, then a strategy copying argument shows

$$\begin{array}{ccc} \text{Duplicator wins the} & \text{implies} & \text{Duplicator wins the} \\ k \text{ round game on } w \text{ and } w' & & k \text{ round game on } f(w) \text{ and } f(w'). \end{array}$$

This implies that first-order definable languages are closed under inverse images of homomorphisms and unary polynomial operations. The same is true for first-order logic on $\circ$-words. □

Example 31. Consider again the monad of finite words, where algebras are monoids. The definite languages from Example 19 are not a variety, because

Another answer is given in:
[13] Eilenberg and Schützenberger, *On pseudovarieties*, 1975 , Theorem 1,
where it is shown that every variety can be axiomatised as follows: one gives an infinite set of identities, but requires only that all but finitely many identities are satisfied.

it is not closed under inverse images of the homomorphisms. Indeed, the language

$$a\{a,b\}^* \subseteq \{a,b\}^*$$

is definite. If we take the inverse image under the homomorphism

$$h : \{a,b,c\}^* \to \{a,b\}^*,$$

which erases the c letters, then we get the language

$$c^*a\{a,b,c\}^* \subseteq \{a,b,c\}^*,$$

which is not definite. The problem is with homomorphism that erase letters. If we would consider the same class of languages but in the monad of nonempty finite words, where algebras are semigroups, then we would get a variety. □

The Eilenberg Pseudo-Variety Theorem says that the two notions of variety are two sides of the same coin.

Theorem 4.19 (Eilenberg Variety Theorem)**.** *Let* T *be a monad in the category of sets. The following maps are mutually inverse bijections*

$$\begin{tikzcd}
\textit{algebra varieties} \arrow[r, bend left, "\mathscr{A} \mapsto \textit{languages recognised by algebras from } \mathscr{A}"] & \textit{language varieties} \arrow[l, bend left, "\mathscr{L} \mapsto \textit{finite algebras which recognise only languages from } \mathscr{L}"]
\end{tikzcd}$$

Proof Let us write L for the left-to-right map, and A for the right-to-left map. We first show that each of these two maps take varieties to varieties, and then we show that the two maps are mutual inverses.

(1) We first show that if $\mathscr{A}$ is a class of finite algebras closed under products (which covers the special case of algebra varieties), then $\mathsf{L}\mathscr{A}$ is a language variety. We begin with Boolean combinations. If L is recognised by an algebra $A \in \mathscr{A}$, then its complement is recognised by the same algebra. If furthermore K is recognised by $B \in \mathscr{A}$, then then $L \cup K$ and $L \cap K$ are both recognised by the product $A \times B$, which belongs to $\mathscr{A}$ by closure under products. Consider now the inverse images. Let L be recognised by a homomorphism

$$h : \mathsf{T}\Sigma \to A \in \mathscr{A}.$$

If $g : \mathsf{T}\Sigma \to \mathsf{T}\Gamma$ is a homomorphism, then the inverse image $g^{-1}(L)$ is recognised by the composition $h \circ g$. Consider now a unary polynomial operation

$f : \mathsf{T}\Sigma \to \mathsf{T}\Sigma$. As we have remarked in the proof of Lemma 4.12, congruences are compatible with unary polynomial operation, which means that

$$h(w_1) = h(w_2) \quad \text{implies} \quad h(f(w_1)) = h(f(w_2)).$$

This, in turn, means that h also recognises the inverse image $h^{-1}(L)$.

(2) Next we show that if $\mathscr{L}$ is a class of recognisable languages closed under unions and intersections (which covers the case of language varieties), then $\mathsf{A}\mathscr{L}$ is an algebra variety[13]. Every language recognised by a sub-algebra of A is also recognised by A, and the same is true for quotients, and therefore $\mathsf{A}\mathscr{L}$ is closed under sub-algebras and quotients of A. Consider now products. Suppose that a language L is recognised by a homomorphism

$$h : \mathsf{T}\Sigma \to A \times B \qquad \text{with } A, B \in \mathsf{A}\mathscr{L}.$$

For every $a \in A$, the inverse image

$$L_a = h^{-1}(\{a\} \times B)$$

is recognised by the homomorphic

$$h_A : \mathsf{T}\Sigma \to A,$$

which is the composition of h with the projection to A. Since the latter homomorphism has domain A, it follows that $L_a \in \mathscr{L}$. For similar reasons, if $b \in B$ then $\mathscr{L}$ contains the language

$$L_b = h^{-1}(A \times \{b\}).$$

The intersection $L_a \cap L_b$ is the inverse image under h of the pair (a, b). Every language recognised by h is a finite union of such languages; and therefore it belongs to $\mathscr{L}$ by closure under unions and intersections.

(3) We now show that the maps A and L are mutual inverses. We first show that every algebra variety $\mathscr{A}$ satisfies

$$\mathscr{A} = \mathsf{AL}\mathscr{A}.$$

This is the same as showing that $A \in \mathscr{A}$ if and only if

(*) every language recognised by A is recognised by some algebra in $\mathscr{A}$.

[13] The first two steps of this proof establish that the maps L and A form what is known as a Galois connection, between

- classes of finite algebras closed under products; and
- classes of recognisable languages closed under unions and intersections.

In the terminology of Galois connections, the varieties are the closed sets, with respect to this Galois connection.

Clearly every algebra $A \in \mathscr{A}$ satisfies (*). We now prove the converse implication. Suppose that an algebra A satisfies (*). The multiplication operation

$$\mu : \mathsf{T}A \to A$$

in the algebra A is a homomorphism from the free algebra $\mathsf{T}A$ to A. By the assumption that A satisfies (*), every language recognised by this homomorphism is recognised by some algebra from $\mathscr{A}$. In particular, for every $a \in A$ the language $\mu^{-1}(a)$ is recognised by some homomorphism

$$h_a : \mathsf{T}A \to B_a \in \mathscr{A}.$$

Consider the product homomorphism

$$h : \mathsf{T}A \to \prod_{a \in A} B_a \qquad t \mapsto (h_a(t))_{a \in A}$$

Define B to be the image of h. This is a sub-algebra of the a product of algebras from $\mathscr{A}$, and therefore it belongs to $\mathscr{A}$. From now on we view h as surjective homomorphism onto its image B. This homomorphism recognises all languages $\mu^{-1}(a)$, and therefore μ factors through h, i.e. there is some function f which makes the following diagram commute:

$$\begin{tikzcd}
\mathsf{T}A \arrow[r, "\mu"] \arrow[dr, "h"'] & A \\
& B \arrow[u, "f"']
\end{tikzcd}$$

By Lemma 4.14, f is not just a function but also a homomorphism of algebras. This means that A is the image of B under a surjective homomorphism. In other words, A is a quotient of B, and therefore $A \in \mathscr{A}$.

(4) In the final step, we show that every language variety $\mathscr{L}$ satisfies

$$\mathscr{L} = \mathsf{LA}\mathscr{L}.$$

This is the same as showing that $L \in \mathscr{L}$ if and only if

(*) L is recognised by an algebra that only recognises languages from $\mathscr{L}$.

Clearly (*) implies $L \in \mathscr{L}$, so we focus on the converse implication. Suppose that $L \in \mathscr{L}$. Since L is recognisable, it has a syntactic homomorphism

$$h : \mathsf{T}\Sigma \to A$$

thanks to Theorem 4.11. To prove (*), we will show that all languages recognised by the syntactic algebra A belong to $\mathscr{L}$.

By the proof of Theorem 4.11, the kernel of the syntactic homomorphism

h is the same as contextual equivalence for L. Therefore, by definition of contextual equivalence, we know that every $v, w \in \mathsf{T}\Sigma$ satisfy

$$h(v) = h(w) \quad \text{iff} \quad \underbrace{p(w) \in L \Leftrightarrow p(v) \in L}_{\substack{\text{for every unary polynomial operation}\\ p : \mathsf{T}\Sigma \to \mathsf{T}\Sigma}}.$$

Unary polynomial operations are compatible with congruences, which means that for every unary polynomial operation p there some function $p^A : A \to A$ which makes the following diagram commute

$$\begin{tikzcd}
\mathsf{T}\Sigma \arrow[r, "h"] \arrow[d, "p"'] & A \arrow[d, "p^A"] \\
\mathsf{T}\Sigma \arrow[r, "h"'] & A
\end{tikzcd}$$

Since p^A can be chosen in finitely many ways, and $p^{-1}(L)$ depends only on p^A, it follows that there is a finite set $\mathcal{P}$ of unary polynomial operations in $\mathsf{T}\Sigma$ such that

$$h(v) = h(w) \quad \text{iff} \quad \underbrace{p(v) \in L \Leftrightarrow p(w) \in L}_{\text{for every unary polynomial operation } p \in \mathcal{P}}. \tag{4.14}$$

We now show that $\mathscr{L}$ contains every language recognised by A. Let then

$$g : \mathsf{T}\Gamma \to A$$

be some homomorphism. By surjectivity of the syntactic homomorphism and the universal property of the free algebra $\mathsf{T}\Gamma$, we can choose some homomorphism f which makes the following diagram commute

$$\begin{tikzcd}
\mathsf{T}\Gamma \arrow[d, "f"'] \arrow[dr, "g"] & \\
\mathsf{T}\Sigma \arrow[r, "h"'] & A
\end{tikzcd}$$

We will show that for every $a \in A$, the language $g^{-1}(a)$ belongs to in $\mathscr{L}$. Since $\mathscr{L}$ is closed under finite unions, it will follow that every language recognised by g belongs to $\mathscr{L}$, thus completing the proof of (*) for L.

Fix some $a \in A$. Since the syntactic homomorphisms is surjective, one can choose some $v \in \mathsf{T}\Sigma$ such that $h(v) = a$. By (4.14), every $w \in \mathsf{T}\Gamma$ satisfies

$$g(w) = h(f(w)) = a = h(v) \quad \text{iff} \quad \underbrace{p(f(w)) \in L \Leftrightarrow p(v) \in L}_{\text{for every unary polynomial operation } p \in \mathcal{P}}.$$

If v is fixed, then the right side of the above is a finite Boolean combination of constraints of the form

$$p(f(w)) \in L \qquad \text{which is the same as} \qquad w \in \underbrace{f^{-1}(p^{-1}(L))}_{\text{a language in } \mathscr{L}}.$$

Summing up, we can express the constraint $g(w) = a$ as a finite Boolean combination of constraints $w \in K$, where K is some language in $\mathscr{L}$. Therefore, $g^{-1}(a)$ belongs to $\mathscr{L}$.

□

Exercises

Exercise 137. Give an example of a monad which is not finitary, but where every language $L \subseteq \mathsf{T}\Sigma$ with a finite alpahbet Σ has a syntactic homomorphism.

Exercise 138. Consider the monad of finite words. Show that a class of languages $\mathscr{L}$ is a variety if and only if it is closed under Boolean combinations, inverse images under homomorphisms, and inverse images of unary polynomial operations of the form:

$$w \mapsto v_1 w v_2 \qquad \text{for every choice of parameters } v_1, v_2 \in \Sigma^*.$$

.

Exercise 139. Consider the monad of finite words. Show that there are uncountably many algebra varieties. In particular, for some algebra varieties, the membership problem $A \overset{?}{\in} \mathscr{A}$ is undecidable.

Exercise 140. Consider the monad of ∘-words. Show that a class of languages $\mathscr{L}$ is a variety if and only if it is closed under Boolean combinations, inverse images under homomorphisms, and inverse images of unary polynomial operations of the following forms:

$$\begin{aligned} w &\mapsto v_1 w v_2 && \text{for every choice of parameters } v_1, v_2 \in \Sigma^\circ \\ w &\mapsto w^\omega \\ w &\mapsto w^{\omega*} \\ w &\mapsto \text{shuffle of } \{w, v_1, \ldots, v_n\} && \text{for every choice of parameters } v_1, l.dots, v_n \in \Sigma^\circ \end{aligned}$$

Exercise 141. Let S be a finite set, and consider the category

$$\mathsf{Set}^S$$

of S-sorted sets with sort-preserving functions. State and prove the Eilenberg Pseudo-Variety Theorem for monads over this category.

Exercise 142. Consider the monad from Example 23, which corresponds to weighted automata. We adapt to varieties to the weighted setting as follows. Define an algebra variety to be class of finite-dimensional algebras which is closed under sub-algebras, quotients and products. Define a language variety to be a class $\mathscr{L}$ of linear maps $\mathsf{T}\Sigma \to \mathbb{Q}$, recognised by finite-dimensional algebras, which is closed under inverse images of homomorphisms and polynomials, and which is closed under combinations in the following sense: if $\mathscr{L}$ contains

$$\{\lambda_i : \mathsf{T}\Sigma \to U_i\}_{i\in\{1,2\}},$$

and $f : \mathbb{Q}^2 \to \mathbb{Q}$ is a linear map, then $\mathscr{L}$ contains also

$$w \mapsto f(\lambda_1(w), \lambda_2(w)).$$

Show that the Eilenberg Pseudo-Variety Theorem holds for varieties understood in this way.

Exercise 143. Consider the weighted varieties from the previous example. What is the weighted analogue of star-free languages? Hint: consider the concatenation of two linear maps

$$\lambda_1, \lambda_2 : \mathsf{T}\Sigma \to U$$

to be the linear map which is defined as follows on Σ^*

$$(\lambda_1 \cdot \lambda_2)(a_1 \cdots a_n) = \sum_{i\in\{0,\ldots,n\}} \lambda_1(a_1 \cdots a_i) \cdot \lambda_2(a_{i+1} \cdots a_n),$$

and which is extended to $\mathsf{T}\Sigma$ by linearity.

PART THREE

TREES AND GRAPHS

5
Forest algebra

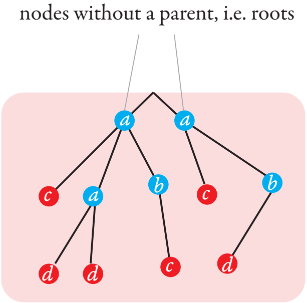
In this chapter, we present a monad that models trees. The trees are going to be finite, node labelled, unranked (no restriction on the number children for a given node), and without a sibling order. Other kinds of trees can be modelled by other monads.

5.1 The forest monad

In fact, the monad will represent a slightly more general object, namely forests, i.e. multisets of trees. The algebras are going to be two-sorted, with the sort names being "forest" and "context". For the rest of this chapter, define a *two-sorted set* to be a set together with a partition into elements of forest sort and elements of context sort. We use a convention where forest-sorted elements are written in red, context-sorted elements are written in blue, and black is used for elements whose sort is not known or which come from a set without sorts.

A *forest* over a two-sorted set Σ consists of:

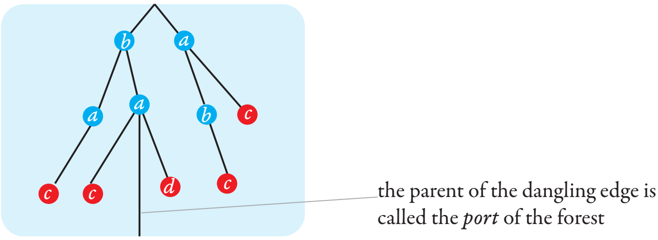
- a set of nodes;
- a parent function, which is a partial function from nodes to nodes;
- a labelling form nodes to Σ.



The parent function must be acyclic, and the labelling function must respect the following constraint: leaves (nodes that are not parents of any other node) have labels of sort "forest", while non-leaves have labels of sort "context". We assume that forests are nonempty, i.e. there is at least one node.

We use the usual tree terminology, such as root (a node without a parent), ancestor (transitive reflexive closure of the parent relation), child (opposite of the parent relation), descendant (opposite of ancestor) and sibling (children of a common parent).

Apart from forests, the forest monad will also talk about *contexts*, which are forests with an extra dangling edge that is attached to a node with a context label, as in the following picture:



the parent of the dangling edge is called the *port* of the forest

The forest monad. We now define a monad structure on forests and contexts. The underlying category is two-sorted sets, where objects are two-sorted sets and the morphisms are sort-preserving functions between two-sorted sets.

Definition 5.1. Define the *forest* monad as follows.

- The category is two-sorted sets, with the sorts called "forest" and "context".
- For a two-sorted set Σ, the forest-sorted elements in $\mathsf{F}\Sigma$ are forests over Σ, while the context-sorted elements are contexts over Σ. A sort-preserving function $f : \Sigma \to \Gamma$ is lifted to a sort-preserving function $\mathsf{F}f : \mathsf{F}\Sigma \to \mathsf{F}\Gamma$ by applying f to the label of every node and leaving the rest of the structure unchanged.
- The unit operation maps a label $a \in \Sigma$ to the unique forest/context that has one node with label a, as in the following pictures:

units of forest labels units of context labels

- Free multiplication is the operation of type $\mathsf{FF}\Sigma \to \mathsf{F}\Sigma$ that is illustrated in Figure 5.1. More formally, the free multiplication of $t \in \mathsf{FF}\Sigma$ is defined as follows. The nodes are pairs (u, v) such that u is a node of t and v is a node

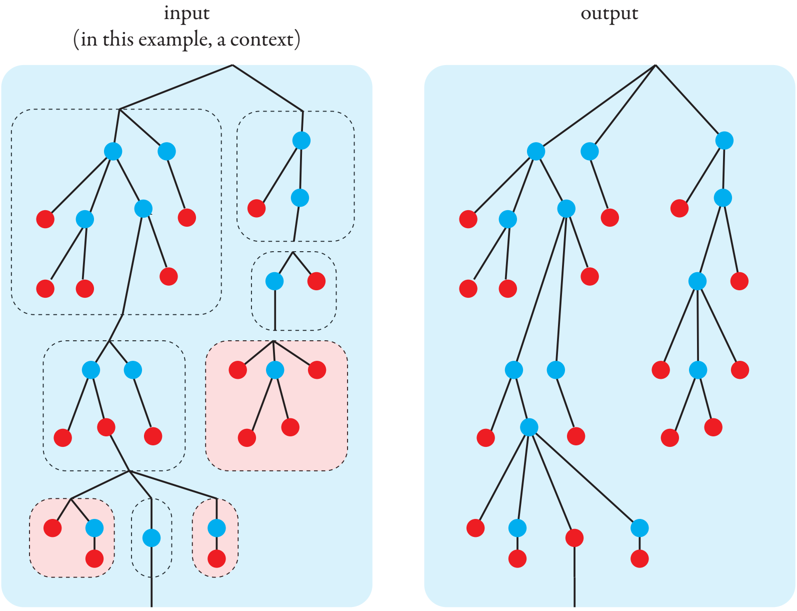


Figure 5.1 Free multiplication in the forest monad

in the label of v. The label is inherited from v, while the parent function is defined as follows (in the following $t_u \in \mathsf{F}\Sigma$ is the label of node u in t):

$$\begin{cases} (u, t_u\text{-parent of } v) & \text{if } v \text{ is not a root in } t_u; \\ (t\text{-parent of } u, \text{port of } t\text{-parent of } u) & \text{if } v \text{ is a root in } t_u \text{ and } u \text{ is not a root in } t; \\ \text{undefined} & \text{otherwise} \end{cases}$$

If t is a context, then the port in the free multiplication is defined to be the port of the context that labels the port of t.

We leave it as an exercise for the reader to check that the monad axioms are satisfied by the above definition. We use the name *forest algebra* for Eilenberg-Moore algebras over this monad.

5.2 Recognisable languages

The rest of this chapter is devoted to a study of the languages recognised by forest algebras. We care mainly about languages recognised by finite forest algebras, which are forest algebras that have finitely many elements on both sorts. We begin with some examples.

Example 32. Let $\Gamma \subseteq \Sigma$ be two-sorted alphabets. We can view

$$\mathsf{F}\Gamma \subseteq \mathsf{F}\Sigma$$

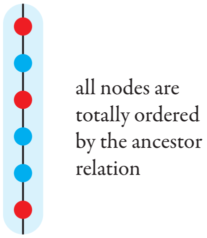
as a language, which only contains those forests and context over alphabet Σ where all labels are from Γ. This language is recognised by homomorphism

$$h : \mathsf{F}\Sigma \to A$$

into a finite forest algebra A defined as follows. The underlying sorted set A is two copies of $\{0, 1\}$, one on the forest sort and one on the context sort. The multiplication operation in the algebra A maps $t \in \mathsf{F}A$ to the maximal number from $\{0, 1\}$ that appears as a label in t, with the number cast into the appropriate sort. It is not hard to see that this is indeed a forest algebra, i.e. the operation μ is associative in the sense required of Eilenberg-Moore algebras. The homomorphism h maps an input to 1 (in the appropriate sort) if it belongs to the language, and to 0 otherwise. □

Example 33. Let Σ be a sorted alphabet, let $n \in \{1, 2, \ldots\}$. Consider the function h which inputs a forest or context in $\mathsf{F}\Sigma$, and outputs the following information: (a) is it a forest or context; (b) what is the number of nodes modulo n. This function is compositional, in the sense of Section **??**. Lemma 4.7 is also true for monads in the category of sorted sets, and therefore h can be viewed as a homomorphism of forest algebras. This homomorphism recognises the language of forests/contexts where the number of nodes is divisible by n. □

Example 34. Consider an alphabet Σ where all letters have context type. In this case, there are no forests over Σ, because there can be no leaves. For the same reason, every context over Σ looks like this:



In other words, $\mathsf{F}\Sigma$ is empty on the forest sort, and is isomorphic to the free semigroup Σ^+ on the context sort. Since the monad structure of the free semigroup agrees with the monad structure of the forest monad, it follows that a forest algebra with an empty forest sort is the same thing as a semigroup. □

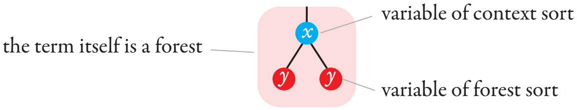
5.2.1 A finite representation

As usual with the monad approach, one needs to explain how the algebras can be finitely represented. Even if the underlying sorted set is finite, the multiplication operation

$$\mu : \mathsf{F}A \to A$$

is in principle an infinite object. We show below a finite representation for the multiplication operation, in analogy to semigroups, where one only needs to define the multiplication operation for inputs of length two. When discussing this finite representation, we use as much as possible the abstract language of monads; this will allow us to see analogies with other finite representations in this book.

A term basis. Like for any monad, a *term* in the forest monad is defined to be an element of $\mathsf{F}X$ for some set of variables X. Here is a picture of a term:



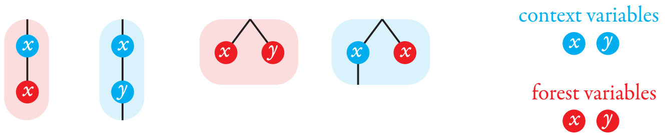
A difference with respect to terms for monads in the category of sets is that in the forest monad – which lives in the category of two-sorted sets – the variables are sorted, which means that there are forest variables, and context variables. Also, the term itself has a sort (call this the output sort). When interpreted in an algebra A, a term $t \in \mathsf{T}X$ induces an operation

$$\eta \in A^X \quad \mapsto \quad \text{multiplication in } A \text{ applied to } (\mathsf{F}\eta)(t).$$

The input to this operation, which is denoted by t^A and called a *term operation* in A, is a sort-preserving valuation of the variables, while the output is an element of the algebra whose sort is the output sort of the term. For example, the term in the picture above induces an operation, which inputs a context sorted x and a forest-sorted y, and outputs a forest sorted element. Note that term operations are not morphisms in the category of two-sorted sets, if only because there is no clear way of assigning a sort to the input valuation.

We distinguish the following terms in forest algebra.

Definition 5.2. Define the *basic forest algebra terms* to be the following terms:



The *basic operations* in a forest algebra A are defined to be the term operations that are induced in A by these terms.We also use the following notation for the basic operations (the colour of an operator is the colour of the output sort):

$$\underbrace{x\cdot x}_{\substack{\text{inputs a}\\ \text{context } x\\ \text{and forest } x\\ \text{and outputs}\\ \text{a forest}}} \quad \underbrace{x\cdot y}_{\substack{\text{inputs a}\\ \text{context } x\\ \text{and context } y\\ \text{and outputs}\\ \text{a context}}} \quad \underbrace{x+y}_{\substack{\text{inputs a}\\ \text{forest } x\\ \text{and forest } y\\ \text{and outputs}\\ \text{a forest}}} \quad \underbrace{x+x}_{\substack{\text{inputs a}\\ \text{forest } x\\ \text{and context } x\\ \text{and outputs}\\ \text{a context}}}$$

The basic operations uniquely determine a forest algebra.

Theorem 5.3. *The multiplication operation in a forest algebra is uniquely determined by the basic operations.*

Proof Every forest or context can be constructed from the units by applying the basic operations. □

The forest algebra in the above theorem does not need to be finite. If the forest algebra is finite, then it can be finitely represented by giving the multiplication tables for the basic operations.

We can also give simple list of axioms which says when the basic operations on a two-sorted set can be extended to a forest algebra multiplication. The reader can easily check that the basic operations in a forest algebra must satisfy all of the following axioms (the colour of the brackets indicates the sort of the bracket, and the colour of the equality sign indicates the sort of the compared elements):

$x+(y+z) = (x+y)+z$ (F1) forests with $+$ are a semigroup

$x+y = y+x$ (F2) the forest semigroup is commutative

$x\cdot(yz) = (xy)\cdot z$ (F3) contexts with $\cdot$ are a semigroup

$x\cdot(y\cdot z) = (xy)\cdot z$ (F4) action of contexts on forests

$x+(y+z) = (x+y)+z$ (F5) action of forests on contexts

$(x\cdot y)+z = (x+z)\cdot y$ (F6) compatibility of the two actions

The above axioms are complete, see the exercises, in the sense that if one supplies four operations on a two-sorted set that satisfy the above axioms, then these operations can be extended to a forest algebra. Using this complete axiomatisation, we can effectively check if a finite representation of a forest algebra is correct, i.e. it comes from some forest algebra.

5.2.2 Syntactic algebras

We now discuss syntactic algebras must necessarily exist in the forest monad. This is proved by a minor adaptation of the results from Section 4.3.

The notion of algebra colouring from Definition 4.8 makes sense also for the forest monad. In fact, this notion makes sense for any monad in any category, not just the category of sets: an algebra colouring is a morphism

$$\lambda : A \to C$$

from the underlying object in an algebra to some object in the category. Also the notion of a syntactic homomorphism from Definition 4.9 makes sense for monads in any category, if one interprets "surjective functions" as "epimorphisms".

In the forest monad, where the underlying category is two-sorted sets, an algebra colouring is a sort-preserving function from the underlying two-sorted set in a forest algebra to some two-sorted set of colours. A subset $L \subseteq A$ can be seen as special case of algebra colouring which uses four colours

$$\underbrace{\{\text{yes, no}\}}_{\text{forest sort}} \cup \underbrace{\{\text{yes, no}\}}_{\text{context sort}}.$$

For the category of two-sorted sets, surjective functions are those which are surjective on both sorts.

The results on existence of syntactic homomorphisms from Section 4.3 can be easily adapted to the forest monad – more generally, to every monad in every category of sorted sets – as explained in the following theorem and its proof.

Theorem 5.4. *Let* T *be a monad in a category of sorted sets (there could be more than two sorts, even infinitely many).*

(1) *If* T *is finitary, then every algebra colouring has a syntactic homomorphism;*

(2) *If the monad is not necessarily finitary, but there are finitely many sort names, then every algebra colouring recognised by a finite algebra (finite on every sort) has a syntactic homomorphism.*

Proof For item (1) we use the same proof as in the left-to-right implication for Theorem 4.16, while for item (2) we use the same proof as in Theorem 4.11. In both cases the proofs use contextual equivalence. The only difference is that the definition of unary polynomial operations used for contextual equivalence from Definition 4.10 needs to be adapted to the sorted setting. In the sorted setting, a unary polynomial operation has an input sort (the sort of the variable) and an output sort (the sort of the polynomial). In item (2), the assumption on finitely many sort names is used[1] to conclude that the equivalence relation $\approx$ in step (3) of Lemma 4.13 has finitely many equivalence classes; if there would be infinitely many sorts then there would be at least one equivalence class for each sort. Apart from this difference, the rest of the proof is the same. □

In particular, since the forest monad is finitary, it follows that every language $L \subseteq \mathsf{F}\Sigma$ in the forest monad has a syntactic algebra. As discussed in the exercises, the syntactic algebra for a language recognised by a finite forest algebra can be computed.

Also, the Eilenberg variety theorem holds for the forest monad. In the statement, the unary polynomial operations are the sorted version that is described in the proof of Theorem 5.4, apart from this change the statement of the theorem and its proof are the same as in Section 4.4. More generally, the Eilenberg variety theorem works for every monad in every category of sorted sets, assuming that there are finitely many sorts. When generalising the proof of the Eilenberg Variety Theorem to multi-sorted algebras, we use the assumption on finitely many sorts in step (4) of the proof, to show that there are finitely many possible unary polynomial operations in a finite algebra.

5.2.3 Infinite trees

Define a monad F_∞ in the same way as the monad F, except that we allow the forests and contexts to be infinite. A node might have infinitely many children, and there might be infinite branches. This monad is no longer finitary.

We do not discuss this monad in more detail, apart from the following example, which shows that it is not clear what a "finite algebra" should be for this monad.

Example 35. Consider two-sorted alphabet $\Sigma = \{a, b\}$. Even though there are not forest-sorted letters, it is still possible to construct an infinite forest over

[1] This assumption is indeed necessary, which can be proved using ideas from
[5] Bojańczyk and Klin, "A non-regular language of infinite trees that is recognizable by a sort-wise finite algebra", 2019

this alphabet, because there is no need for leaves. Let $L \subseteq \Sigma^\omega$ be some prefix-independent language of ω-words, not necessarily regular. For example

$$L = \{b^{n_1} a b^{n_2} a \cdots : \text{the sequence } n_1, n_2, \ldots \text{ contains infinitely many primes}\}.$$

Define a *branch* in a forest to be a set of nodes that is linearly ordered by the descendant relation, and which is maximal inclusion-wise for this property. An L-branch is a branch where the sequence of labels, starting from the root, belongs to L.

Define $L' \subseteq \mathsf{F}_\infty \Sigma$ to be the set of infinite forests where every node has at least two children and every node belongs to some L-branch. We claim that L' is recognised by a finite algebra, with at most 36 elements, regardless of the choice of L (as long as it is prefix independent). Since there are uncountably many possible choices for L, it follows that there is no finite way of representing algebras in this monad that have at most 36 elements. In particular, "finite on every sort" is not a reasonable choice of "finite algebra" for this monad.

Define a function h from $\mathsf{F}\Sigma$ to a finite two-sorted set as follows. For forests, the function h gives the answers to the following questions:

1. is the forest in K?
2. are there are at least two roots?

For contexts, the function h gives the answers to the following questions:

1. is it possible to fill the port with some forest so that the result is in K?
2. are there are at least two roots?
3. does the port have a sibling?
4. is the context equal to the unit of a?
5. is the context equal to the unit of b?

The red questions have at most 4 possible answers, and the blue questions have at most 32 possible answers hence the number 36. In fact, this number can easily be reduced; for example in case of a "no" answer to question 1, there is not need to store the answers for the remaining questions. We leave it as an exercise for the reader to check that the function h is compositional. It follows that the image of the function h, call it A, is a finite algebra for the monad F_∞. □

Exercises

Exercise 144. (1) Prove completeness for the axioms (F1)–(F6).

Exercise 145. Show that the syntactic algebra can be computed for a language $L \subseteq \mathsf{F}\Sigma$ that is recognised by a finite forest algebra. The input to the algorithm is a homomorphism

$$h : \mathsf{F}\Sigma \to A$$

into a finite forest algebra, together with an accepting set $F \subseteq A$. The forest algebra is represented using its basic operations, as in Theorem 5.3, and the homomorphism is represented by its values on the units.

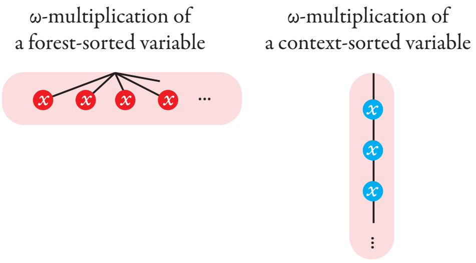
Exercise 146. Show that recognisable languages in the forest monad are not closed under images of (not necessarily letter-to-letter) homomorphisms

$$h : \mathsf{F}\Sigma \to \mathsf{F}\Gamma.$$

Exercise 147. Consider the monad F_∞. We say that a forest/context in this monad is *thin* if it has countably many branches. Define $\mathsf{F}_{\mathrm{thin}}\Sigma \subseteq \mathsf{F}_\infty\Sigma$ to be thin forests/contexts. Show that this is a monad

Exercise 148. Show that a forest/context is thin, in the sense of Exercise 147, if and only if one can assign countable ordinal numbers to its children so that if a node is labelled by ordinal number α, then all of its children are labelled by ordinal numbers $\leq \alpha$, and at most one child is labelled by α.

Exercise 149. Consider the monad $\mathsf{F}_{\mathrm{thin}}$ from Exercise 147. Show that a finite algebra over this monad is determined uniquely by its forest algebra operations (as in Theorem 5.3) plus the following two term operations:


ω-multiplication of a forest-sorted variable

ω-multiplication of a context-sorted variable

5.3 Logics for forest algebra

In this section, we discuss languages in the forest monad that can be defined using logic, and the structural properties of the forest algebras that define them.

5.3.1 Monadic second-order logic

We begin with monadic second-order logic. The idea is as usual: to each forest/context we associate a model, and then we can use monadic second-order logic to describe properties of that model. There will be one twist: because siblings in a forest/context are not ordered, we will need to extend MSO with modulo counting in order to make it expressively complete for all recognisable languages.

Definition 5.6. Define the *ordered model* of a forest or context as follows: the universe is the nodes, and it is equipped with the following relations:

$$\underbrace{x \le y}_{\text{ancestor}} \qquad \underbrace{a(x)}_{x \text{ has label } a \in \Sigma} \qquad \underbrace{port(x)}_{x \text{ is the port}}$$

The arguments to the relations are x, y, while the letter a is a parameter. Each choice of $a \in \Sigma$ gives a different relation.

By using different logics on the ordered model, we get different classes of languages.

We begin with monadic second-order logic. A language $L \subseteq \mathsf{F}\Sigma$ is called MSO *definable* if it can be defined by a formula of monadic second-order logic using the ordered model. The logic MSO is not enough to define all recognisable languages, because it cannot count the number of nodes modulo two (or three, etc.). The problem is that there is no order on the siblings, so if we get a forest

$$a + \cdots + a$$

that consists of n nodes that are both roots and leaves, then we cannot use the usual trick of selecting even-numbered nodes to count parity. (A more formal argument will be given below.) For these reasons, we extend MSO with modulo counting. In this extension – called *counting* MSO – for every set variable X and numbers $n \in \{2, 3, \ldots\}$ and $\ell \in \{0, \ldots, n-1\}$ we can write a formula

$$|X| \equiv \ell \mod n$$

which says that the size of the set X is congruent to ℓ modulo n. This extension is expressively complete for the recognisable languages, as shown in the following theorem.

Theorem 5.7. *A language $L \subseteq \mathsf{F}\Sigma$ is recognised by a finite forest algebra if and only if it is definable in counting* MSO.

Proof Both implications in the theorem are proved in a similar way as for finite words, so wo only give a proof sketch.

From a finite forest algebra to counting MSO. Suppose that $L \subseteq \mathsf{F}\Sigma$ is recognised by a homomorphism

$$h : \mathsf{F}\Sigma \to A$$

into a finite forest algebra. We will show that for every $a \in A$, the inverse image $h^{-1}(a)$ is definable in counting MSO. By taking finite unions of such inverse images, we can get any language recognised by h, in particular L.

The idea is the same as for finite words: the defining formula inductively computes the value under h for every subtree in the input tree/context. The induction corresponds to a bottom-up pass through the input[2]. For $t \in \mathsf{F}\Sigma$ and a node x in t, define the *subtree of* x, to be the forest/context which is obtained from t by restricting the nodes to x and its descendants. The subtree is a context if the port is a descendant of x, otherwise it is a forest. Define the *type* of a node to be the image under h of its subtree. The following claim shows that the type of a node can be inferred from the types of its children using counting.

Claim 5.8. *For every $t \in \mathsf{F}\Sigma$ and every node x in t, the type of x depends only on the answers to the following questions:*

- *what is the label of node x?*
- *are there exactly n children of x type a?*
- *does n divide the number of children of x with type a?*

where a ranges over elements of the forest algebra A and $n \in \{0, \dots, |A|\}$.

Proof Let $a_1, \dots, a_m$ be the types of the subtrees of the children of x. At most one of these types is a context, because there is at most one port. We only consider the case where all of the types are forests (and hence they will be written in red below); the case when one type is a context is treated similarly. If a is the label of node x, then the type of x is equal to

$$a \cdot (a_1 + \cdots + a_m).$$

[2] This idea works for objects such as finite words or forest algebra, because they have a canonical way of parsing (left-to-right for words, or bottom-up for forest algebra), which can be defined in MSO. For $\circ$-words, we do not know any simple parsing method, which is the reason why the implication from finite algebras to logic in Section 3.4 was hard. A similar phenomenon will appear for graphs, which will be discussed in the next chapter, which also do not have any simple definable canonical way of parsing.

The label a is known, while the red part is multiplication in the forest semigroup of A, which is a commutative semigroup. In a commutative semigroup, the result of multiplication depends only on the number of times each argument is used. Furthermore, since the forest semigroup has size at most $|A|$, then the number of times an argument is used needs to be remembered only up to threshold $|A|$ and modulo some number that is at most $|A|$, see Exercise 11. □

Consider some enumeration $A = \{a_1, \ldots, a_n\}$ of the elements in the algebra. Using the above claim, we can write a formula

$$\varphi(X_1, \ldots, X_n)$$

of counting MSO which holds if and only if for every $i \in \{1, \ldots, n\}$, the set X_i is exactly the set of nodes with type a_i. The formula simply checks that the types for each node are consistent with the types of its children, as described in the above claim. Finally, the image $h(t)$ of a forest/context can be computed in counting MSO by guessing the sets $X_1, \ldots, X_n$ that satisfy the formula φ above, and then inferring $h(t)$ from the types of the root nodes (with the same argument as in the above claim).

From counting MSO to a finite forest algebra. As for finite words, instead of using MSO over the ordered model, it will be more convenient to use first-order logic over a model where elements are sets of nodes. For a forest/context, define its *set model* as follows. The universe is all subsets of the nodes, and it is equipped with the following relations:

$$\underbrace{x \subseteq y}_{\text{set inclusion}} \qquad \underbrace{x \subseteq a}_{\substack{\text{every node}\\ \text{in } x \text{ has}\\ \text{label } a \in \Sigma}} \qquad \underbrace{x \leq y}_{\substack{\text{every node in } x\\ \text{is an anecstor of}\\ \text{every node in } y}} \qquad \underbrace{port(x)}_{\substack{\text{every node}\\ \text{in } x \text{ is a port}}} \qquad \underbrace{|x| \equiv \ell \mod n}_{\substack{\text{the number of nodes}\\ \text{in } x \text{ is congruent}\\ \text{with } \ell \text{ modulo } n}}.$$

In the above relations, the arguments are x, y (which are elements in the set model, and therefore sets of nodes), while a and n, ℓ are parameters. Each choice of parameter gives a new relation. It is not hard to see that a language is first-order definable in the set model if and only if it is definable in counting MSO in the ordered model. Like for finite words, to every first-order formula

$$\varphi(\underbrace{x_1, \ldots, x_n}_{\substack{\text{free variables}\\ \text{describe sets of nodes}}})$$

in the set model over alphabet Σ, we can associate a language L_φ whose alphabet is 2^n disjoint copies of Σ. By induction on formula size, one shows that for every formula, the associated language is recognised by a finite forest algebra. For the atomic formulas $x \subseteq y$ and $x \subseteq a$, one uses the forest algebra from Example 32. For the atomic formulas with modulo counting one uses Example 33. Finite forest algebras for the remaining atomic formulas $x \leq y$ and $port(x)$ are left as an exercise for the reader. In the induction step, one uses products of forest algebras (for Boolean combinations) and a powerset forest algebra (for the quantifiers)[3]. □

The construction of an algebra in the above theorem is effective: given a sentence of MSO, we can construct a recognising homomorphism

$$h : \mathsf{F}\Sigma \to A$$

into a finite forest algebra (and compute an accepting set $F \subseteq A$). The finite forest algebra is represented by its basic operations, as discussed in the previous section, and the homomorphism is represented by its images for the units.

As discussed above, counting is needed to define all recognisable languages. The exact role of counting is explained in the following theorem.

Theorem 5.9. *A language $L \subseteq \mathsf{F}\Sigma$ is definable in* MSO *(without counting) if and only if it is recognised by a forest algebra where the forest semigroup (forests equipped with $+$) is aperiodic.*

A corollary of this theorem is that modulo counting is needed to define the language "even number of nodes", since this language cannot be defined by a forest algebra with an aperiodic forest semigroup.

Proof For the left-to-right implication, we use the same proof as in the left-to-right implication of Theorem 5.7. The only difference is that in Claim 5.8 we do not need modulo counting. This is because for every commutative aperiodic semigroup, the outcome of multiplication depends only on the number of times that each argument is used up to some finite threshold, without modulo counting.

Consider now the right-to-left implication, which says that if a language is definable in MSO without counting, then it is recognised by a finite forest algebra with an aperiodic forest semigroup. Here, again, we use the same proof as in Theorem 5.7, where a recognising forest algebra is constructed by starting with some atomic forest algebras, and then applying products and the powerset construction. Since we do not need the relation

$$|x| \equiv \ell \mod n$$

from the set model, all of the atomic forest algebras have forest semigroups that

[3] The powerset construction for algebras, and sufficient conditions for the monad which ensure that it is well defined, will be discussed in the next chapter, in Lemma 6.13. This lemma can be applied to the forest monad.

are aperiodic. Products clearly preserve aperiodicity of the forest semigroup, and the same is true powersets, as explained in the following lemma.

Lemma 5.10. *If S is a commutative*[4] *aperiodic semigroup, then the same is true for its powerset semigroup $\mathsf{P}S$.*

Proof In this proof, we use multiplicative notation for the semigroup operation. By aperiodicity of S, there is some $! \in \{1, 2, \ldots\}$ such that every element of $b \in S$ satisfies $b^! = b^! b$. To establish aperiodicity of the powerset semigroup, we will show that every element $A \subseteq S$ of the power set semigroup satisfies

$$A^n = A^{n+1}$$

where n is the size of S times $! + 1$. We only show the inclusion $A^{n+1} \subseteq A^n$, the same proof can be used to establish the opposite inclusion. Let

$$a = a_1 \cdots a_{n+1} \in A^{n+1}.$$

By the pigeon-hole principle and choice of n, some $b \in A$ must appear at least $! + 1$ times in the sequence $a_1, \ldots, a_{n+1}$. By commutativity and aperiodicity of S, one extra occurrence of b can be eliminated from the product, proving $A^{n+1} \subseteq A^n$. □

□

Corollary 5.11. *A language is definable in* MSO *without counting if and only if its syntactic forest algebra is finite and has an aperiodic forest semigroup.*

Proof Aperiodicity of the forest semigroup is preserved when taking subalgebras and quotients (images under surjective homomorphisms). Since the syntactic forest algebra can be obtained from any recognising forest algebra by taking a subalgebra and then a quotient, the result follows from Theorem 5.9. □

Since the syntactic forest algebra can be computed for recognisable languages, it follows that given a sentence of counting MSO, we can decide if there is a sentence of MSO which does not use counting and which is equivalent on forests and contexts.

[4] Commutativity is important in the proof of the lemma. For example, when proving the equivalence of MSO and finite semigroups in the first part of the book, we started out with atomic semigroups that are aperiodic, and all other semigroups (including groups) could be obtained by applying products and powersets.

5.3.2 First-order logic

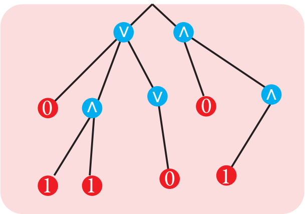
For finite words, the king of algebraic characterisations was the Shützenberger-McNaughton-Papert-Kamp Theorem, which described the languages of finite words that can be defined in first-order logic (using the ordered model). Unfortunately, finding a generalisation of this theorem to forest algebra (or any other algebra modelling trees) remains an open problem. Therefore, our discussion of first-order logic in the forest monad is limited to some remarks and one example.

As discussed in Section 5.2.2, The Eilenberg Variety Theorem works also for the forest monad. One can show that, in the forest monad, the class of languages definable in first-order logic is a language variety, see Exercise 150. Therefore, from the Eilenberg Variety Theorem it follows that whether or not a language $L \subseteq \mathsf{F}\Sigma$ is definable in first-order logic depends only on the syntactic algebra of the language. However, it is not known if the corresponding property of syntactic algebras is decidable. Here is an example which shows that aperiodicity – which characterised the syntactic algebras for first-order definable languages of finite words – is not enough for forest algebra.

Example 36. Consider an alphabet

$$\Sigma = \{ \underbrace{\vee, \wedge}_{\text{context sort}}, \underbrace{0, 1}_{\text{forest sort}} \}.$$

A forest over this alphabet is the same thing as a multiset of positive Boolean formulas, as in the following picture:



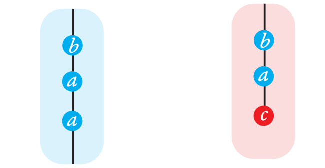
We define the *value* of a node in a forest over this alphabet to be the value of the Boolean formula in the subtree of the node. Consider the language

$$L = \{t \in \mathsf{F}\Sigma : t \text{ is a forest where all roots have value } 1\}$$

If we look at the syntactic forest algebra of this language, then both the forest semigroup and the context semigroup are aperiodic (in fact, they are idempotent). Nonetheless, the language is not definable in first-order logic, see Exercise 152. □

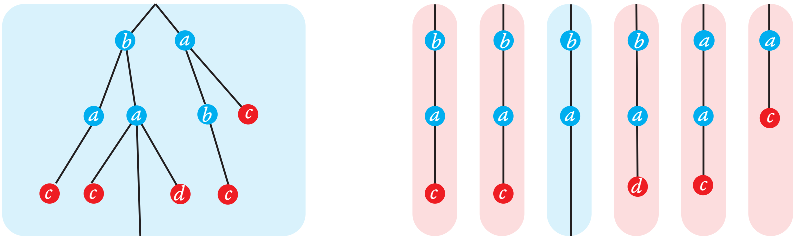
5.3.3 Branch languages

In this section, we discuss languages which are defined only by looking at branches in a forest/context. We say that a forest/context is a *branch* if all nodes are linearly ordered by the ancestor relation. Here is a picture:



a context branch a forest branch

A branch can be viewed as a word, consisting of the labels of the nodes in the branch, listed in root-to-leaf order. For a forest/context t, define a *branch of t* to be any branch that can be obtained from selecting some x which is either the port or a leaf, and restricting t to the ancestors of x. The branch is a context if x is the port, otherwise the branch is a forest. Here is a picture:



a context all of its branches

The following theorem gives a characterisation of languages that are determined by only looking at branches.

Theorem 5.12. *For every language $L \subseteq \mathsf{F}\Sigma$, not necessarily recognisable, the following conditions are equivalent*

(1) membership $t \in L$ depends only on the set of branches in t;
(2) the syntactic forest algebra of L satisfies the identities

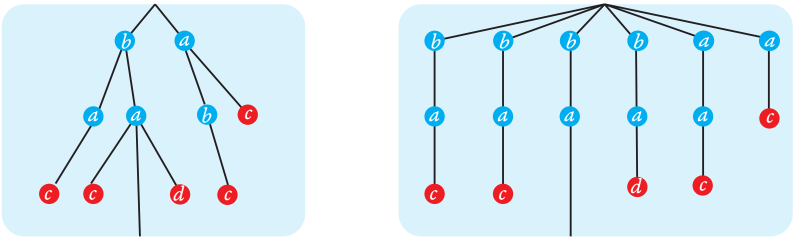
$$\underbrace{a \cdot (b + c) = (a \cdot b) + (a \cdot c)}_{\text{distributivity}} \quad \text{and} \quad \underbrace{b + b = b}_{\substack{\text{idempotence of}\\ \text{the forest semigroup}}} \qquad \text{for every } a, b, c \in A.$$

If L is recognisable, then the above conditions are also equivalent to:

(3) *L is a finite Boolean combination of languages of the form "for some branch, the corresponding word is in $K \subseteq \Sigma^*$", where K is regular.*

Proof

- (1)⇒(2) Applying the identities in the free algebra $\mathsf{F}\Sigma$ does not affect the set of branches.
- (2) ⇒ (1) For $t \in \mathsf{F}\Sigma$, define its *branch normal form* to be the forest/context that is the union of all branches in t, as described in the following picture:



a context

its branch normal form

If the syntactic algebra satisfies the distributivity identity in the theorem, then a forest/context has the same image under the syntactic homomorphism as its branch normal form. Since the branch normal form is determined uniquely by the multiset of branches, it follows that the image under the syntactic homomorphism depends only on the multiset of branches[5]. Thanks to the idempotence identity, it is only the set of branches that matters, and therefore the language must be branch testable.

- (3) ⇔ (1) for recognisable languages. Clearly (3) implies (1). Consider now the converse implication. Let h be the syntactic homomorphism of a recognisable language. By condition (1) and the definition of a syntactic homomorphism, membership $t \in L$ depends only on the set

$$H(t) = \{h(s) : s \text{ is a branch in } t\}.$$

For every a in the syntactic algebra, define $K_a \subseteq \Sigma^*$ to be the words that correspond to branches which have value a under the syntactic homomorphism. This language is recognised by a finite semigroup (which is easily constructed from the forest algebra used by h), and therefore it is regular. Finally, $a \in H(t)$ if and only if for some branch the corresponding word is

[5] One could think that the distributivity identity alone (without the one for idempotence) characterises exactly the languages where membership depends only on the multiset of branches. This is not true, see Exercise 155.

in K_a. Therefore, $H(t)$ can be computed using a finite Boolean combination of languages of the form K_a.

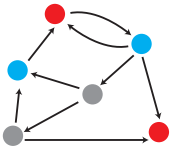
□

Condition (2) in the above theorem can be effectively checked given the syntactic algebra. Since the syntactic algebra can be computed for recognisable languages, it follows that one can decide if a recognisable language satisfies any of the conditions in the above theorem.

5.3.4 Modal logic

In this section, we discuss the tree variants of some of the temporal logics that were discussed in Chapter 2. When working with trees and forests, we use the terminology of modal logic, described as follows.

Define a *Kripke model* to be a directed graph with vertices labelled by some alphabet Σ. Here is a picture of a Kripke model:



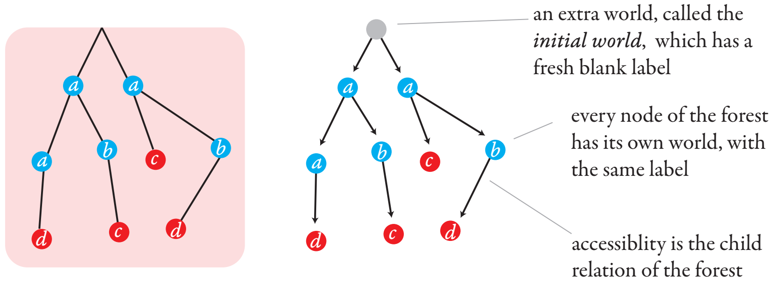
Vertices of the Kripke model are called *worlds*, and the edge relation is called *accessibility*. Accessibility does not need to be a transitive relation. In this section, we will only study Kripke models where accessibility is acyclic. To express properties of worlds in Kripke models we use modal logic, whose formulas are constructed as follows:

$$\underbrace{a}_{\substack{\text{the current}\\ \text{world has}\\ \text{label } a \in \Sigma}} \qquad \underbrace{\Diamond\varphi}_{\substack{\text{some}\\ \text{accessible}\\ \text{world}\\ \text{satisfies } \varphi}} \qquad \underbrace{\Box\varphi}_{\substack{\text{every}\\ \text{accessible}\\ \text{world}\\ \text{satisfies } \varphi}} \qquad \underbrace{\neg\varphi \quad \varphi\wedge\psi \quad \varphi\vee\psi}_{\text{Boolean combinations}}.$$

We use the following notation for the semantics of modal logic:

$$\underbrace{M}_{\substack{\text{Kripke}\\ \text{model}}}, \underbrace{v}_{\substack{\text{world}\\ \text{of } M}} \models \underbrace{\varphi}_{\substack{\text{formula}\\ \text{of modal}\\ \text{logic}}}.$$

We will use modal logic to define properties of forests, by assigning a Kripke model to each forest[6], as explained in the following picture:



an extra world, called the *initial world*, which has a fresh blank label

every node of the forest has its own world, with the same label

accessiblity is the child relation of the forest

a forest

its Kripke model

Definition 5.13 (Forest languages definable in modal logic)**.** We say that a formula of modal logic is true in a forest if it is true in the initial world of its Kripke model. A forest language is called *definable in modal logic* if there is a formula of modal logic that is true in exactly the forests from the language.

The following theorem characterises modal logic in terms of two identities. A corollary of the theorem is that one can decide if a language is definable in transitive modal logic; this is because it suffices to check if the identities hold in the syntactic algebra of a language.

Theorem 5.14. *Let $L \subseteq \mathsf{F}\Sigma$ be a language that contains only forests. Then L is definable in modal logic if and only if its syntactic forest algebra is finite and satisfies the identities*

$$a + a = a \qquad c^{!}\cdot a = c^{!}\cdot b,$$

where $! \in \{1, 2, \ldots\}$ is the idempotent exponent of the context semigroup.

Proof The rough idea is that the identities say that the membership in the language is invariant under bisimulation (the first identity) and depends only on nodes at constant depth (the second identity). These are exactly the properties that characterise modal logic. A more detailed proof is given below.

Define the *modal rank* of a formula to be the nesting depth of the modal operators $\Diamond$ and $\Box$. Here are some examples:

$$\underbrace{a}_{\text{modal rank 0}} \qquad \underbrace{(\Diamond a) \wedge (\Diamond b)}_{\text{modal rank 1}} \qquad \underbrace{(\Diamond(a \wedge \Box b) \wedge (\Diamond b)}_{\text{modal rank 2}}.$$

[6] One could also assign a Kripke model to a context, by doing the same construction, except with a special marker for the port node. We choose not to do this, without any deeper reasons, and therefore in what follows we only discuss languages that contain only forests.

When the alphabet is finite and fixed, then there are finitely many formulas of given modal rank, up to logical equivalence. To prove the theorem, we use a slightly more refined result, in the following claim, which characterises the expressive power of modal logic of given modal rank.

Claim 5.15. *A forest language can be defined by a formula of modal rank* $n \in \{0, 1, \ldots\}$ *if and only if its syntactic algebra satisfies the identities*

$$a + a = a \qquad c^n \cdot a = c^n \cdot b$$

Proof We say that a Kripke model is tree-shaped if the accessibility relation gives a finite tree, with edges directed away from the root (this is the case for the Kripke models that we assign to forests). We say that two tree-shaped Kripke models are *bisimilar* if one can be transformed into the other by applying the identity $a + a = a$, i.e. duplicating or de-duplicating identical sibling subtrees[7]. For $n \in \{0, 1, \ldots\}$, we say that two tree-shaped Kripke models are n-bisimilar if, after removing all worlds that are separated by more than n edges from the root, they are bisimilar. By induction on n one shows that every equivalence class of n-bisimilarity can be defined by a formula of modal logic with modal rank n; and conversely formulas of modal rank n are invariant under n-bisimilarity. The identities in the statement of the claim say that the forest language is invariant under n-bisimilarity, and hence the claim follows. □

The theorem follows immediately from the above claim. Indeed, if the identities in the theorem are satisfied, then the language can be defined by a formula of modal logic with modal rank $!$. Conversely, if the language is defined by a formula of nesting depth n, then membership in the language is not affected by nodes which are more than n edges away from the root, and therefore the syntactic algebra must satisfy

$$\begin{aligned}
c^! a \quad &= \quad \text{(because } c^! \text{ is idempotent)}\\
c^{!n} a \quad &= \quad \text{(because } a \text{ is more than } n \text{ edges away from the root)}\\
c^{!n} b \quad &= \quad \text{(because } c^! \text{ is idempotent)}\\
c^! b. &
\end{aligned}$$

□

Transitive modal logic. A formula of modal logic can only talk about nodes that are at some constant distance from the root. We now discuss a variant of modal logic which can talk about arbitrarily deep nodes.

[7] For tree-shaped Kripke models this notion coincides with the usual notion of bisimulation for general Kripke models.

For a forest, define its *transitive Kripke model* of a forest in the same way as the Kripke model, except that the accessibility relation now models the transitive closure of the child relation. In other words, accessibility now represents the proper descendant relation.

Definition 5.16 (Forest languages definable in transitive modal logic)**.** A language that contains only forests is called *definable in transitive modal logic*[8] if there is a formula of modal logic that is true in (the initial world) of exactly the forests from the language.

The following theorem characterises transitive modal logic in terms of two identities. A corollary of the theorem is that one can decide if a language is definable in transitive modal logic.

Theorem 5.17. *Let $L \subseteq \mathsf{F}\Sigma$ be a language that contains only forests. Then L is definable in modal logic if and only if it is recognised by a finite forest algebra which satisfies the identities*

$$a + a = a \qquad c{\cdot}a{=}(c{\cdot}a){+}a.$$

Proof It is easy to see that the identities must be true in the syntactic algebra of every language definable in transitive modal logic. The first identity says that the language must be invariant under bisimulation, which is clearly true for transitive modal logic. For the second identity, we observe that going from $c{\cdot}a$ to $(c \cdot a) + a$ does not affect the transitive Kripke model, up to bisimulation.

The rest of this proof is devoted to the right-to-left implication. Let

$$h : \mathsf{F}\Sigma \to A$$

be a homomorphism into an algebra A that satisfies the identities. By induction on the size of A, we will show that for every forest-sorted $a \in A$, the inverse image $h^{-1}(a)$ is definable in transitive temporal logic (we say that such a is definable in the rest of the proof). This immediately yields the right-to-left implication. In the rest of the proof, we define the *type* of an element of $\mathsf{F}\Sigma$ to be its image under h.

In the proof, we use use a reachability ordering on the algebra A which is defined as follows. We say that $a \in A$ is *reachable* from $b \in A$, denoted by $a \leq b$, if there is some $t \in \mathsf{F}A$ which uses b at least once, and which gives a under the multiplication operation of A. (Reachability can be seen as the forest algebra variant of the infix relation for semigroups.) Reachability is easily seen to be a pre-order, i.e. it is transitive and reflexive. We draw the

[8] In the terminology of temporal logic, this logic is also called EF, which refers to the "exists finally" operator of (branching time) temporal logic.

reachability ordering in red when comparing forest-sorted elements. Thanks to the identities in the theorem, reachability is anti-symmetric when restricted to the forest sort, as explained in the following claim.

Claim 5.18. *If forest-sorted $a, b \in A$ are reachable from each other, then $a = b$.*

Proof For forest-sorted $a, b \in A$, reachability $a \geq b$ is equivalent to

$$\underbrace{a = c \cdot b}_{\text{for some context-sorted } c} \qquad \text{or} \qquad \underbrace{a = c + b}_{\text{for some forest-sorted } c} \qquad \text{or} \qquad a = b.$$

In the presence of the identities from the assumption of the theorem, all three conditions above imply $a = a + b$. For the same reason, $a \leq b$ implies $b = a + b$, and therefore $a = b$. □

In every finite forest algebra there is a maximal forest-sorted element with respect to reachability, because every two forests can be combined using $+$, into a forest that is bigger than both of them. By the above, the maximal element is unique. Fix the maximal element a for the rest of the proof.

The following claim uses the induction assumption on algebra size to give a sufficient condition for definability.

Claim 5.19. *Assume that $c \in A$ is forest-sorted, and there is some non-maximal forest-sorted $b \in A$ from which c is not reachable. Then c is definable.*

Proof Let I be the set of elements in A that are reachable from b. This is an ideal, which means that if $t \in \mathsf{F}A$ contains at least one letter from I, then its multiplication is in I. Furthermore, this ideal contains at least two forest-sorted elements, by assumption that b is non-maximal. Define $\sim$ to be the equivalence relation on A which identifies two elements if they are equal, or both have the same sort and belong to I. Because I is an ideal, $\sim$ is a congruence which means that the quotient function g which maps an element of A to its equivalence class is a homomorphism. Because b is non-maximal, the homomorphism makes the algebra smaller, and therefore the induction assumption on algebra size can be used to show that every language recognised by $g \circ h$ is definable in transitive modal logic. Because c is not reachable from b, it does not not belong to the ideal I, and thus the equivalence class of c under $\sim$ consists of c only. This means that h and $g \circ h$ have the same inverse images for c, and hence this inverse image is definable in transitive modal logic. □

We will use the above claim to show that, with at most two exceptions, all forest-sorted elements of A are definable. Call an element b *sub-maximal* if it is not maximal and $c > b$ implies that c is maximal. If c is neither maximal nor sub-maximal then it is definable by the above claim, because there is some sub-maximal b from which c is not reachable. For the same reason, if there are at least two sub-maximal elements, then all sub-maximal elements are definable. In particular, if there are at least two sub-maximal elements, then all elements are definable: the non-maximal elements are all definable, and the maximal element is definable as the complement of the remaining elements.

We are left with the case when there is exactly one sub-maximal element, call it b. We will show that the maximal element a is definable, and therefore b is definable (as the complement of the remaining elements). Define the *descendant forest* of a node x to be the forest that is obtained by keeping only the proper descendants of x. Since we do not allow empty forests, the descendant forest is defined only when x is not a leaf.

Claim 5.20. *A forest has type a if and only if it contains a node x, with label $\sigma \in \Sigma$, such that one of the following conditions hold:*

(1) *the descendant forest of x has type $d < b$ and $h(\sigma) \cdot d{=}a$; or*
(2) *the descendant forest of x has type $d \geq b$ and $h(\sigma) \cdot b{=}a$; or*
(3) *the node x is a leaf and $h(\sigma) = a$.*

Proof To prove the bottom-up implication, we observe that each of the conditions (1,2,3) implies that the subtree of x has type a; by maximality of a the entire forest must then also have type a. For conditions (1,3) the observation is immediate. For condition (2), there are two cases to consider: either the descendant forest has type a and the subtree of x has type a by maximality, or the descendant forest has type b and the subtree of x has type a by $h(\sigma) \cdot b{=}a$.

We now prove the top-down implication. We show that if every node in a forest violates all of the conditions (1,2,3), then the forest has type $\leq b$. If the forest has only one node, then we use condition (3). The second case is when the forest has at least two trees, i.e. it can be decomposed as $t = t_1 + t_2$. By induction assumption, both t_1 and t_2 have types $b_1, b_2 \leq b$. By the identity in the theorem, we get

$$b = b + b_1 + b_2,$$

which implies that $b_1 + b_2 \leq b$. The final case is when t is a tree, whose root x has label σ and descendant forest s. By induction assumption, s has a type $d \leq b$. If $d < b$ then we use condition (1) to infer that t has type b, otherwise we use condition (2). □

To finish the proof of the theorem, it remains to show that the conditions in the above claim can be expressed using transitive modal logic. Condition (3) can easily be checked. In condition (1), the element d is definable because it is neither maximal nor sub-maximal. Therefore, the there is a formula of modal logic which is true in the world corresponding to a node x (in the descendant Kripke model) if and only if the descendant forest of x has type d. Therefore, there is a formula of modal logic which is true in the world corresponding to x if and only if it satisfies (1). For similar reasons, we can define condition (2), since the union of the languages for a and b is definable, by taking the complement of the remaining definable languages. □

Exercises

Exercise 150. Prove that first-order logic, as discussed in Section 5.3.2, is a variety in the sense of the Eilenberg variety theorem.

Exercise 151. Show that the language of forests where some leaf has even depth is not definable in first-order logic.

Exercise 152. Show that the language from Example 36 is not definable in first-order logic.

Exercise 153. Consider a two-sorted alphabet

$$\Sigma = \{\text{left, right}, \text{left, right}\}.$$

A *binary tree* over this alphabet is a tree where every node is either a leaf, or it has exactly two children, with labels “left” and “right” in the appropriate sort. There are no constraints on the root label. Define $L \subseteq \mathsf{F}\Sigma$ to be the set of binary trees where all leaves are at even depth. Show that this language is first-order definable. Hint: show first that there is a first-order language which separates L from the set of binary trees where all leaves are at odd depth.

Exercise 154. Define *anti-chain logic* to be the variant of MSO where set quantification is restricted to anti-chains, i.e. sets of nodes that are pairwise incomparable with respect to the descendant relation. Show that anti-chain logic can define all recognisable languages that contain only binary trees, as defined in Example 153.

Exercise 155. Give an example of a language $L \subseteq \mathsf{F}\Sigma$ where membership depends only on the multiset of branches, but where the syntactic algebra violates the distributivity identity from Theorem 5.12.

Exercise 156. A unary node in a forest/context is a node with exactly one child. Show that anti-chain logic with modulo counting can define every recognisable language where every element has no unary nodes.

Exercise 157. Give an algorithm which decides if a recognisable language $L \subseteq \mathsf{F}\Sigma$ is of the form: "for some branch, the corresponding word is in $K \subseteq \Sigma^*$", for some regular K.

6
Graphs of bounded treewidth

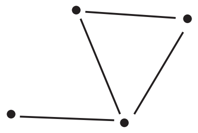
In this chapter, we discuss graphs and algebras for them. Although in principle the algebras can describe arbitrary graphs, the associated algorithms will work on graphs of bounded treewidth.

6.1 Graphs, logic, and treewidth

We begin with the simplest definition of a graph: simple undirected graphs. Later on, we will add several features to graphs: labels, directed hyperedges and ports. These features will be needed to impose a monad structure on graphs.

Definition 6.1 (Graph)**.** A *graph* consists of a set of vertices, together with a binary symmetric edge relation.

Here is a picture of a graph:



We use logic, mainly MSO, to define properties of graphs.

Definition 6.2 (Graph languages definable in MSO)**.** Define the *incidence model* of a graph as follows. The universe is the disjoint union of the vertices and the edges, and there is a binary *incidence* relation, which is interpreted as

$$\{(v, e) : \text{vertex } v \text{ is incident with edge } e\}.$$

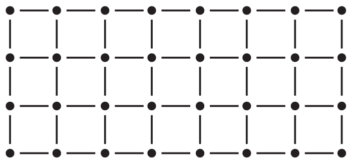
The two kinds of elements in the universe of the incidence model – vertices and edges – can be distinguished using first-order logic: an element of the universe is an edge if it is incident to some vertex, otherwise it is a vertex.

The incidence model is sometimes times called the MSO$_2$ model. An alternative is the MSO$_1$ model, where the universe is only the vertices, and there is a binary relation for the edges. For first-order logic, the two models are equivalent. The edge relation of the relation can be defined in the incidence model by

$$E(v,w) \quad \Leftrightarrow \quad \exists e \; \text{incident}(v,e) \wedge \text{incident}(w,e),$$

while first-order quantification over edges in the incidence model can be replaced by first-order quantification over pairs of vertices in the MSO$_1$ model. For monad second-order logic, the MSO$_2$ model gives more expressive power than the MSO$_1$ model, as explained in the following example.

Example 37. Define a *grid* to be a graph that looks like this:



We leave it as an exercise for the reader to check that the set of grids can be defined in MSO (this is possible in both the incidence model and the MSO$_1$ model). Consider now the set of graphs which are cliques of prime size. This language can be expressed in MSO using the incidence model: one cannot remove edges so as to get a grid which has at least two rows and at least two columns. On the other hand, for graphs which are cliques, monadic second-order logic over the MSO$_1$ model has the same expressive power as first-order logic. For cliques, every sentence of first-order logic will select finitely many or all but finitely many cliques. □

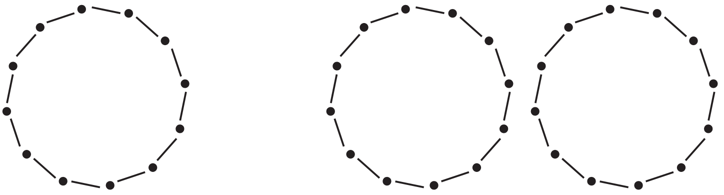
In this chapter, we will not be studying first-order definable properties of graphs. Such properties are necessarily local[1], e.g. the existence of a cycle of length three:

$$\exists u \, \exists v \, \exists v \qquad E(u,v) \wedge E(v,w) \wedge E(w,u).$$

Connectivity is an example of a non-local property that cannot be expressed in first-order logic. Using an Ehrenfeucht-Fraïssé argument, one can show that

[1] The notion of locality is made precise by the Gaifman Theorem, see [18] Heinz-Dieter Ebbinghaus, *Finite Model Theory*, 2006 , Theorem 2.5.1

a sentence of first-order logic cannot distinguish between a large cycle and a disjoint union of two large cycles:



a connected graph

a disconnected graph

On the other hand, connectivity can be expressed in monadic second-order logic, already in the MSO$_1$ model, as witnessed by the following sentence

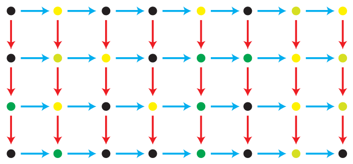
$$\underbrace{\exists X}_{\substack{\text{there is a set}\\ \text{of vertices,}}} \quad \underbrace{(\exists v\ v \in X)\ \wedge (\exists v\ v \notin X)}_{\text{which is neither empty nor full,}} \wedge \underbrace{(\forall v\ \forall w\ E(v,w) \wedge v \in X \Rightarrow w \in X).}_{\text{and which is closed under taking edges}}$$

Example 38. [Hamiltonian cycles] Using MSO over the incidence model, we can say that a graph has a Hamiltonian cycle, i.e. a directed cycle that visits every vertex exactly once. A Hamiltonian cycle can be seen as a set of edges X such that: (a) every vertex in the graph has exactly one incoming and one outgoing edge from the set; (b) if only the edges from the set are used, then the graph is connected. The incidence model is important here. In the MSO$_1$ model, where quantification over sets of edges is not allowed, one cannot express the existence of a Hamiltonian path[2] . □

Already first-order logic is undecidable on graphs, i.e. it is undecidable whether or not a sentence of first-order logic is true in some graph. This undecidability is explained in the following example.

Example 39. Consider directed graphs with coloured vertices and edges. These extra features can be easily encoded, using first-order logic, in (undirected and unlabelled) graphs, see the exercises. A computation of a Turing machine can be visualised as a coloured grid, where: the rows represent configurations, and the colours of the vertices represent cell contents, as in the following picture:

[2] [10] Courcelle and Engelfriet, *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, 2012 , Proposition 5.13



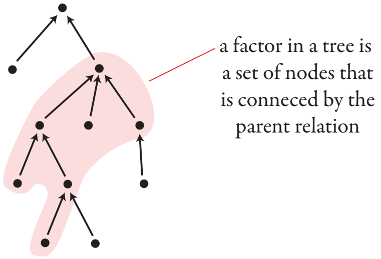
One can write a sentence of first-order logic which is true in exactly those grids that represent accepting computations of a given Turing machine. Therefore, the halting problem reduces to satisfiability for first-order logic over finite graphs. □

6.1.1 Treewidth

The undecidability problems described in Example 39s will be avoided if we consider graphs that are similar to trees. In this chapter, the notion of similarity that we care about is treewidth, as defined below[3].

Definition 6.3 (Tree decompositions and treewidth)**.** A *tree decomposition* of a graph G consists of a set of *nodes*, equipped with a binary parent relation, and a function which maps each node to its bag, which is a nonempty set of vertices. These should satisfy the following constraints:

(1) the parent relation induces a forest structure on nodes, i.e. it is acyclic, every node has at most one parent, and there is one root (node without a parent);
(2) for every edge, there is some bag that contains both endpoints of the edge;
(3) for every vertex v, the set of nodes with v in their bag is a nonempty *factor* in the tree, as explained in the following picture:

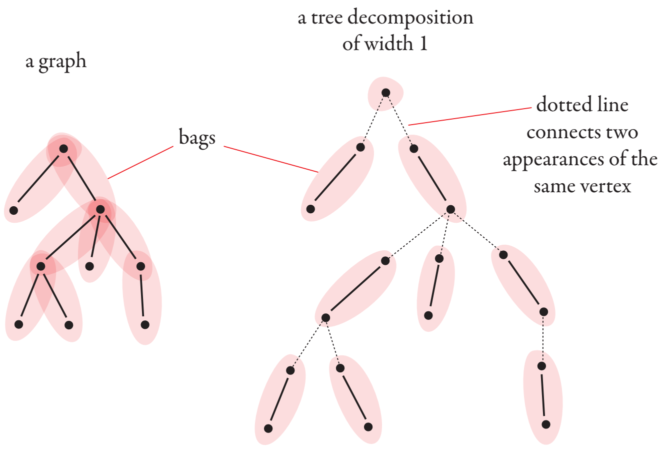


The *width* of a decomposition is defined to be the maximal size of bags, minus

[3] For an introduction to treewidth, including a brief history, see [12] Diestel, *Graph theory (electronic edition)*, 2006 , Section 12.

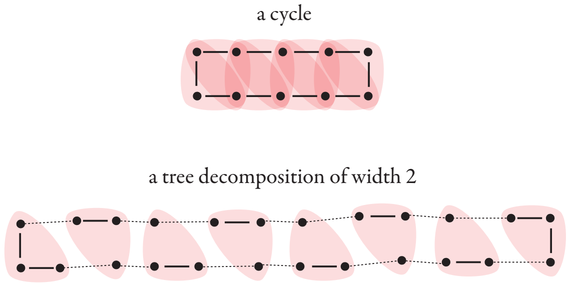
one. The *treewidth* of a graph is the minimal width of a tree decomposition for the graph.

If a graph is a tree (i.e. it is an acyclic undirected graph), then it has treewidth one. The nodes of the tree decompositions are edges of the graph, plus one extra root node. Here is a picture of a tree decomposition for a tree, where the underlying graph is a tree:



The bags in have size at most two, and therefore the treewidth is one (the width is defined to be the size of bags minus one so that trees have treewidth one). Forests (i.e. disjoint unions of trees) are the only graphs of treewidth one.

Cycles have treewidth 2, as illustrated in the following example:



The tree decomposition in the above picture is a *path decomposition*, i.e. every node in the tree decomposition has at most one child.

If a graph has $k + 1$ vertices, then it has treewidth at most k, since one can always use a trivial tree decomposition where all vertices of the graph are in the same bag. For cliques, the trivial tree decomposition is optimal, as shown in the following example.

Example 40. Consider a clique, i.e. a graph where every two vertices are connected by an edge. We show that the trivial tree decomposition is optimal, i.e. in every tree decomposition, there must be some node whose bag contains all of the vertices in the clique. Consider a tree decomposition of a clique. For each vertex of the clique, consider the factor which contains nodes that have the vertex in their bag. Since the graph is a clique, for every two vertices the corresponding factors have nonempty intersection. We will show that if $\mathcal{X}$ is a family of pair-wise intersecting factors in a tree, then the intersection $\bigcap \mathcal{X}$ is nonempty; this will imply that some node of the tree decomposition has all vertices of the clique in its bag.

Indeed: each factor has a root, i.e. a node in the factor that is an ancestor of all other nodes in the factor. If two factors have roots that are not comparable by the ancestor relation, then these factors are disjoint. Therefore, if $\mathcal{X}$ is a family of pairwise intersecting factors, then their roots must be totally ordered by the ancestor relation. The root of the factor which is furthest away from the root of the entire tree must be in the intersection $\bigcap \mathcal{X}$. □

Another example of graphs with unbounded treewidth is grids, see the exercises. In fact, the Grid Theorem, which is stated but not proved in the exercises, says that a class of graphs has unbounded treewidth if and only if it contains all grids as minors. We will show later in this chapter that for every $k \in \{1, 2, \ldots\}$, the class of graphs of treewidth at most k has a decidable MSO theory. A corollary of the Grid Theorem is that this result is also optimal: if the MSO theory of a class of graphs is decidable, then the class has bounded treewidth.

Exercises

Exercise 158. (1) Show that the grids, as described in Example 39, can defined in first-order logic (with two extra unary predicates, which select the blue and red edges).

Exercise 159. (2) Show that the existence of a Hamiltonian cycle cannot be expressed using the MSO_1 representation of graphs as a models.

Exercise 160. (2) Show that the existence of an Euler cycle (every edge is visited exactly once) cannot be expressed using the MSO$_1$ representation of graphs as a models.

Exercise 161. (2) For $\ell \in \{1, 2, \ldots\}$ define the ℓ-reduction of a graph to be the result the following operation: for each pair of vertices v, w we only keep the first ℓ edges that go from v to w. Show that for every MSO sentence φ there is some ℓ such that φ is true in a graph if and only if it is true in its ℓ-reduction.

Exercise 162. (2) Consider undirected graphs (the special case of graphs where for each edge there is an opposite edge, and there are no parallel edges). We say that an undirected graph G is a *minor* of an undirected graph H if one can find a family of disjoint vertex sets

$$\{X_v \subseteq \text{vertices of } H\}_{v \in \text{vertices of } G}$$

such that every set of vertices U in G satisfies:

$$\underbrace{U \text{ is connected in } G}_{\substack{\text{a subset of vertices is connected if} \\ \text{the induced subgraph is connected}}} \qquad \text{implies} \qquad (\bigcup_{v \in U} X_v) \text{ is connected in } H.$$

(It is enough to check the implication for sets U with at most two vertices; and we assume that one vertex sets are connected). Show that if L is a property of undirected graphs that is definable in MSO, then also "some minor satisfies L" is definable in MSO.

Exercise 163. (1) The Grid Theorem says that if a class of undirected graphs has unbounded treewidth, then for every $n \in \{1, 2, \ldots\}$ there is some graph in the class which has an $n \times n$ grid as a minor. Using the Grid Theorem prove that if a class of undirected graphs has decidable MSO theory, then it has bounded treewidth.

Exercise 164. (1) Building on the previous exercises, show that if a class of (not necessarily unidrected) graphs has decidable MSO theory, then it has bounded treewidth.

Exercise 165. (2) Show that the $n \times n$ grid has treewidth at least n.

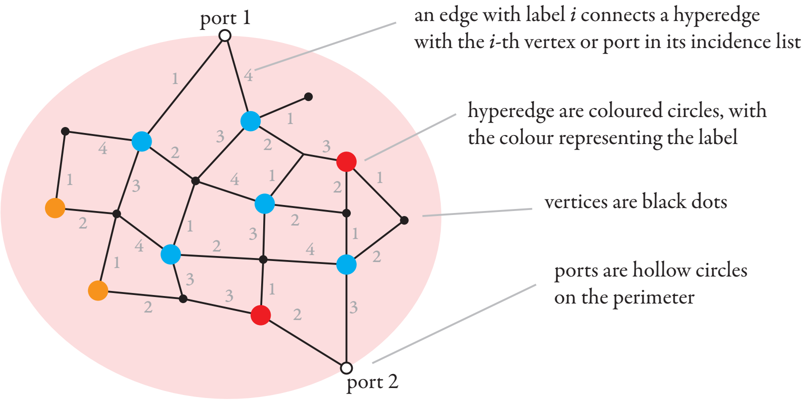
6.2 The hypergraph monad

In this section, we describe the hypergraph monad[4], which impose an algebraic structure graphs and their generalisation to hypergraphs. We use the hypergraph generalisation, because the extra features of hypergraphs – labels, arities, ports – will be used to define the monad structure. Like any monad, the hypergraph monad will allow us to talk about algebras, homomorphisms, terms, recognisable languages, syntactic algebras, etc. The main result of this section is going to be Courcelle's Theorem, which says that every graph property definable in MSO is necessarily recognisable. In the next section, we will also present a converse to Courcelle's Theorem, which will say that for bounded treewidth, recognisability implies definability in MSO.

Definition 6.4. A *hypergraph* consists of:

- A set V of vertices.
- A set E of hyperedges. Each hyperedge has an arity in $\{0, 1, \dots\}$.
- A Σ of *labels*. Each label has an associated arity in $\{0, 1, \dots\}$.
- An arity $k \in \{0, 1, \dots\}$ and an injective port sequence $\{1, \dots, k\} \to V$.
- For each hyperedge of arity n, a label in Σ of arity n, and an injective *incidence list* of type $\{1, \dots, n\} \to V$.

We use the name *ports* for the vertices in the port sequence, and *non-port vertices* for the remaining vertices. We draw hypergraphs like this:



an edge with label i connects a hyperedge with the i-th vertex or port in its incidence list

hyperedge are coloured circles, with the colour representing the label

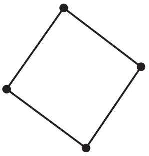
vertices are black dots

ports are hollow circles on the perimeter

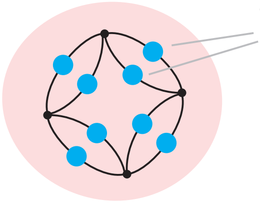
To avoid clutter in the pictures, we skip the numbers on the edges, if they are not important for the picture or implicit from the context.

[4] This monad is based on the hyperedge replacement algebras of Courcelle.

A directed graph can be represented as a hypergraph. The representing hypergraph has zero ports, and its vertices are the same in the graph. Each edge of the graph is represented by two binary hyperedges (with some fixed label), one in each direction. Here is a picture:



a graph



directed in opposing directions

its representation as a hypergraph

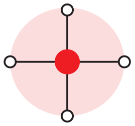
The hypergraph monad. The idea behind the hypergraph monad is that a hyperedge can be replaced by a hypergraph of matching arity. This replacement, which will be the free multiplication in the monad, is illustrated in the Figure 39.

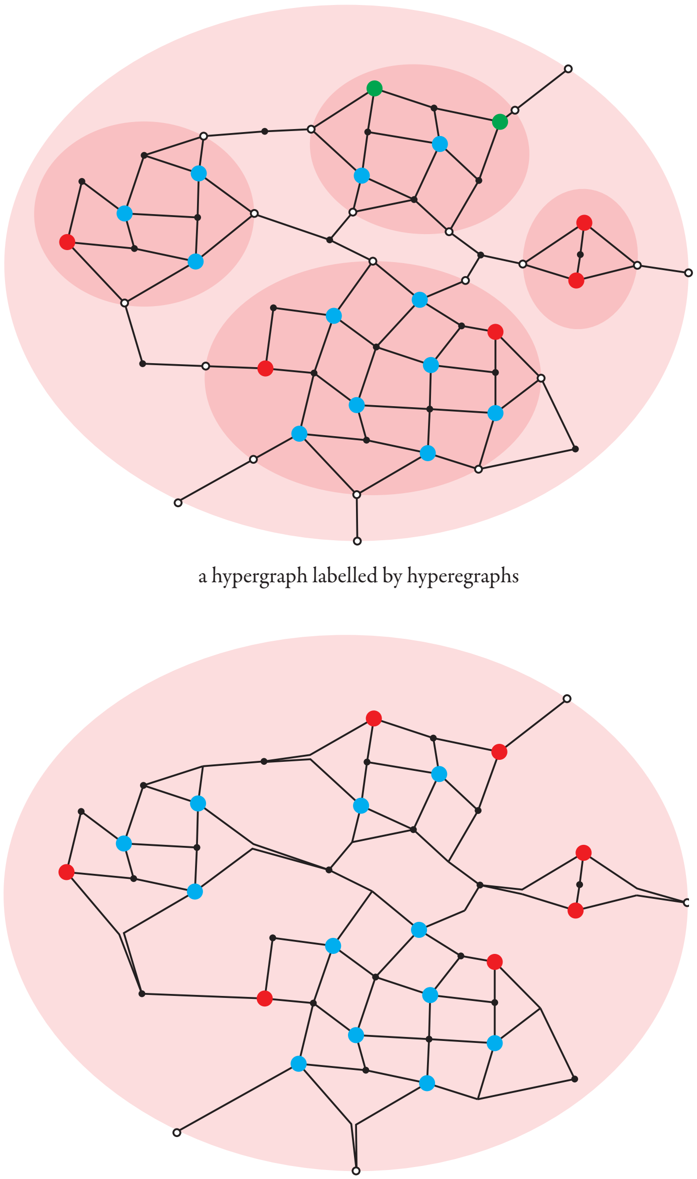
Definition 6.5 (Hypergraph monad)**.** The hypergraph monad, denoted by H, is defined as follows.

- The underlying category is the *category of ranked sets*
$$\mathsf{Set}^{\{0,1,\ldots\}}.$$
Objects are *ranked sets*, i.e. sets where every element has an associated arity in $\{0, 1, \ldots\}$. Morphisms are arity-preserving functions between ranked sets.
- For a ranked set Σ, the ranked set $\mathsf{H}\Sigma$ consists of hypergraphs labelled by Σ, modulo isomorphism. Recall that the arity of a hypergraph is the number of ports.
- For a function $f : \Sigma \to \Gamma$, the function $\mathsf{H}f : \mathsf{H}\Sigma \to \mathsf{H}\Gamma$ applies f to the labels, without changing the rest of the hypergraph structure.
- The unit of the monad maps an n-ary label $a \in \Sigma$ to the hypergraph with ports $\{1, \ldots, n\}$, no other vertices, and one hyperedge labelled by a with incidence list $i \mapsto i$. Here is a picture:

a letter

its unit hypergraph





a hypergraph labelled by hyperegraphs

its free multiplication

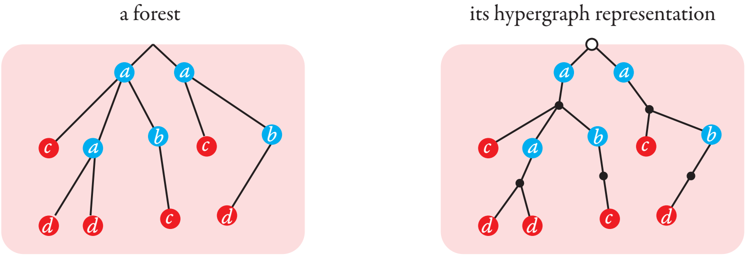
Figure 6.1 Free multiplication in the monad H

- Let $G \in \mathsf{HH}\Sigma$ be a hypergraph. Its free multiplication is defined as follows. The vertices are vertices of G, plus pairs (e, v) such that e is a hyperedge of G and v is a vertex in the hypergraph G_e that is the label of the hyperedge e. The hyperedges are pairs (e, f), where e is a hyperedge of G and f is a hyperedge in G_e. The arities and labels of hyperedges are inherited from the second coordinate, while the incidence lists are defined by

$$(e, f)[i] = \begin{cases} f[i] & \text{if } f[i] \text{ is a non-port vertex} \\ e[j] & \text{if } f[i] \text{ is the } j\text{-th port.} \end{cases}$$

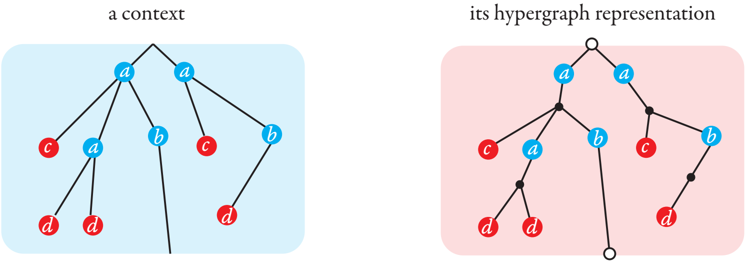
We leave it as an exercise for the reader to check that the above definition satisfies the monad axioms. This completes the definition of the hypergraph monad.

Example 41. The hypergraph monad can be seen as a generalisation of the forest monad. A forest can be represented as a hypergraph of arity one, as explained in the following picture:


a forest

its hypergraph representation

A context can be represented as a hypergraph of arity two:


a context

its hypergraph representation

This representation is consistent with the monad structures of the forest monad and the context monad. Therefore, we can think of the forest monad as being a sub-monad of the hypergraph monad (when we identify the forest sort with arity 1, and the context sort with arity 2). In particular, from every algebra of the hypergraph monad we can extract an algebra of the forest monad. □

The rest of this section is devoted to discussing the algebraic notions that arise from this monad, such as terms and algebras.

6.2.1 Recognisable languages

We begin by discussing recognisable languages.

Example 42. Let M be a commutative monoid. Define an algebra for the monad H as follows. The underlying ranked set A has a copy of M on each arity. (More formally, an n-ary element of A is a pair (a, n) with $a \in M$.) The multiplication operation in the algebra maps a hypergraph $G \in \mathsf{H}A$ to the multiplication – in the monoid M – of all the labels of its hyperedges. (Because the monoid is commutative, the order of multiplication is not important.) The result of this multiplication is viewed as an element of the n-th copy of M, with n being the arity of G. It is not hard to see that this operation is associative, i.e. it satisfies the axioms of Eilenberg-Moore algebras.

The algebra constructed this way can be used to recognise some simple languages of hypergraphs. Consider first the monoid

$$(\{0, \ldots, n\}, \max).$$

If A is the algebra constructed for this monoid as above, then it can be used to recognise the hypergraph language

$$\{G \in \mathsf{H}\Sigma : \text{at least } n \text{ hyperedges have label in } \Gamma\} \qquad \text{for ranked sets } \Gamma \subseteq \Sigma.$$

The homomorphism maps a hypergraph to the number of hyperedges with label in Γ, up to a threshold of n, with the number stored in the copy of the monoid that matches the arity of the input graph. Another application of this construction is recognising the language of hypergraphs with an even number of hyperedges; here the appropriate monoid is the two-element group. □

The algebras in the above example are infinite, but finite on every arity. In the hypergraph monad, it is impossible for an algebra to have an underlying set that is finite altogether. This is because the multiplication operation $\mu : \mathsf{H}A \to A$ in an algebra is arity-preserving, and $\mathsf{H}A$ is nonempty on every arity (think of the hypergraphs that have only ports, and no vertices or hyperedges). Therefore, the underlying set of an algebra must be nonempty on every arity. In the following definition, we assume that "finite algebras" are those which have finitely many elements for each arity.

Definition 6.6 (Recognisable language of hypergraphs)**.** We say that a language $L \subseteq \mathsf{H}\Sigma$ is *recognisable* if it is recognised by a homomorphism into an algebra which has finitely many elements on every arity.

This definition will turn out to be not restrictive enough, as far as general hypergraphs are concerned, see Example 45. In fact, no entirely satisfactory definition of "finite algebra" for general hypergraphs is known, and possibly does not exist. However, for bounded treewidth, the above notion will be fully satisfactory, as we will see in Section **??**.

Example 43. Define a *path* in a hypergraph to be sequence of the form

$$v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} v_{n-1} \xrightarrow{e_n} v_n,$$

where $v_0, \ldots, v_n$ are vertices or ports and $e_1, \ldots, e_n$ are hyperedges, such that each hyperedge e_i is incident with both v_{i-1} and v_i. Note that the notion of path does not depend on the order of the incidence lists for the hyperedges, e.g. reversing all arrows would also give a path. The *source* of the path is v_0, and v_n is its *target*. We say that the path *connects* v_0 with v_n. A hypergraph is called *connected* if every vertex or port can be connected to every other vertex or port via a path. Define h to be the function which maps a hypergraph to the following information: (a) its arity; (b) is the hypergraph connected; (c) is there some vertex that is not connected to any port; (c) which pairs of ports are connected. For hypergraphs with at least one port item (b) is redundant. One can check that this is function is compositional, and therefore its image can be equipped with the structure of an algebra so that h is a homomorphism. The corresponding algebra is finite on every arity. Therefore, the language of connected hypergraphs is recognisable. □

Example 44. In this example, we show that the language of k-colourable hypergraphs is recognisable. Define a *3-colouring* of a hypergraph to be a function from ports and vertices to $\{1, \ldots, k\}$ such that every hyperedge has an incidence list that uses each colour at most once. (In particular, there all hyperedges must have arity at most k.) Define h to be the function which maps a hypergraph to the following information: (a) its arity; (b) which functions from the ports to $\{1, \ldots, k\}$ can be extended to k-colourings. If the hypergraph has arity zero, then (b) is just one bit of information: is there a k-colouring or not. This function is compositional, and has finitely many values for each arity, and therefore the language of k-colourable hypergraphs is recognisable. □

The following example illustrates a problem with of our notion of recognisability, which is that it allows for too many algebras.

Example 45. We say that a hypergraph is a clique if it has no ports, and for every two vertices are adjacent (i.e. connected by some hyperedge). Let P be any set of natural numbers, possibly undecidable. We will show that the language "cliques whose size is in P" is recognisable. Let h be the function which maps a hypergraph to the following information: (a) its arity; (b) does there exist a vertex that is not adjacent to some vertex or port; (c) which ports are adjacent. If the arity is zero, then h also stores (c) is the number of vertices in P. This function is compositional, and has finitely many values for each arity, and therefore the language "cliques whose size is in P" is recognisable. □

As we will see later on, the problem from the above example will disappear once we restrict attention to hypergraphs of bounded treewidth.

6.2.2 Terms and tree decompositions.

Tree decompositions and treewidth can be naturally extended to hypergraphs.

Definition 6.7 (Tree decompositions for hypergraphs)**.** Tree decompositions are defined for hypergraphs in the same way as for graphs, with the following differences: (a) bags can also contain ports; (b) for every hyperedge there must be some bag which contains its entire incidence list; (c) for every port of the hypergraph, the nodes with the port in their bag are a nonempty prefix of the tree (a prefix is a set closed under taking parents).

A corollary of item (c) is that all ports can be found in the root bag of the tree decomposition. As before, the width of a tree decomposition is the maximal bag size minus one, and the treewidth of a hypergraph is the minimal width of a tree decomposition. For hypergraphs which represent graphs (i.e. no ports, and every edge is represented by two binary hyperedges in opposing directions), the above definition coincides with Definition 6.3.

A bag in a tree decomposition can contain an unbounded number of hyperedges. This will not be a problem for our intended applications, since the properties of hypergraphs that we study will not depend in an important way on parallel hyperedges (i.e. hyperedges with the same incidence lists).

Tree decompositions as terms. The algebraic structure of the hypergraph monad can be used to give an alternative description of treewidth. Recall the notion of terms from Section 4.3.1: a term over variables X is any element of $\mathsf{H}X$. As was the case for the forest monad, the terms in the hypergraph monad are sorted, which means that each variable has an arity, and the term itself has an arity. If A is an algebra, then a term $t \in \mathsf{H}X$ induces a term operation

$$\underbrace{\eta \in A^X}_{\substack{\text{an arity-preserving}\\ \text{valuation of the variables}}} \qquad \mapsto \qquad \underbrace{\text{multiplication in } A \text{ applied to } (\mathsf{H}\eta)(t).}_{\substack{\text{an element of the algebra } A,\\ \text{whose arity is the same as the arity of } t}}.$$

We denote this operation by t^A, and call it a *term operation* in the algebra A. As was the case for forest algebra, term operations are in general not arity-preserving.

Since a term is a hypergraph, it has some treewidth. The following lemma shows that hypergraphs of treewidth at most k are closed under applying (term operations induced by) terms of treewidth at most k. The algebra used in the lemma is the free algebra.

Lemma 6.8. *Let $t \in \mathsf{H}X$ be a term and let $\eta \in (\mathsf{H}\Sigma)^X$ be a valuation of its variables. If the hypergraphs t and $\{\eta(x)\}_{x \in X}$ have treewidth at most k, then the same is true for the result of applying the term operation $t^{\mathsf{H}\Sigma}$ to η.*

Proof Take a tree decomposition for the term t. For every hyperedge e which is labelled by a variable x, find a node whose bag contains the incidence list of the hyperedge, remove the hyperedge, and add a child to this node with a tree decomposition of $\eta(x)$. □

A corollary of the above lemma is that there is a well-defined monad for hypergraphs of treewidth at most k. This monad, call it H_k, uses only hypergraphs with treewidth at most k, with all the monad structure inherited from H. The underlying category is ranked sets with arities at most $k+1$; since hypergraphs with bigger arities will have treewidth at least $k+1$. In particular, the alphabet Σ can have letters of arity at most $k+1$.

The treewidth terms. We now show a family of terms which can be used to generate all hypergraphs of given treewidth. Define the *treewidth terms* to be the terms from Figure 6.2. For an algebra, define its *treewidth k operations* to be the term operations induced in the algebra by treewidth terms that have treewidth at most k. The following theorem shows that the treewidth terms can be used to generate all hypergraphs of given treewidth.

Theorem 6.9. *Let $k \in \{1, 2, \ldots\}$. A hypergraph has treewidth at most k if and only if it can be generated (in the free algebra) from hypergraphs with no vertices and at most $k+1$ ports, by applying treewidth k operations.*

Proof The right-to-left implication follows from Lemma 6.8.

Consider now the left-to-right implication. Every tree decomposition can easily be modified, without affecting its width, into a tree decomposition which

Forgetting. Let x be a variable of arity $k+1$. The *k-forgetting term* is defined to be the hypergraph in $\mathsf{H}\{x\}$ which has ports $\{1, \ldots, k\}$, one non-port vertex v, and one hyperedge with label x and incidence list $(1, \ldots, k, v)$.

Fusion. Let x, y be two variables of arity k. The *k-fusion term* is defined to be the hypergraph in $\mathsf{H}\{x, y\}$ which has k ports, no vertices, and two hyperedges with labels x and y and incidence list $(1, \ldots, k)$.

Rearrangement. Let $f : \{1, \ldots, k\} \to \{1, \ldots, \ell\}$ be an injective function. Let x be a variable of arity k. The *f-rearrangement term* is defined to be the hypergraph in $\mathsf{H}\{x\}$ which has ℓ ports, no vertices, and one hyperedge with label x and incidence list $(f(1), \ldots, f(k))$.

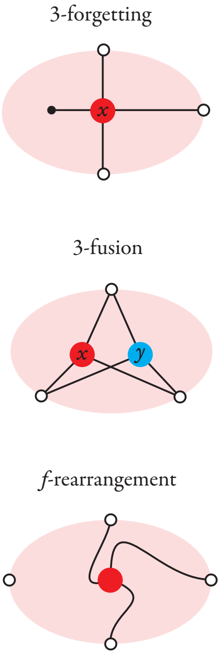


Figure 6.2 The treewidth terms. The parameters k, ℓ are from $\{0, 1, \ldots\}$. In the pictures, the ports are ordered clockwise from top, and the same is true for the vertices incident to a hyperedge.

satisfies: (*) the root bag contains only ports, and if a node has at least two children, then the bag of the node is equal to the bags of all of its children. To ensure condition (*), we can insert an extra node on every parent-child edge which has the same bag as the parent. By a simple induction on the size number of nodes, one shows that for every width k tree decomposition satisfying (*), the underlying hypergraph can be generated using the treewidth k operations as in the statement of the lemma. □

Using the above theorem, and the same argument as in Theorem 3.13, we get the following corollary.

Corollary 6.10. *Let $k \in \{1, 2, \ldots\}$ and consider the monad H_k of hypergraphs with treewidth at most k. The multiplication operation in an algebra over this monad is uniquely determined by its treewidth k operations.*

6.2.3 Courcelle's Theorem

In this section, we prove Courcelle's Theorem, which says that all languages definable in MSO are recognisable. To define properties of hypergraphs in MSO, we use the incidence model defined as follows.

Definition 6.11 (Incidence model)**.** The incidence model of a hypergraph is defined as follows. The universe is

$$\text{ports} + \text{vertices} + \text{hyperedges}$$

and it is equipped with the following relations:

$$\underbrace{e[i] = v}_{\substack{v \text{ is the } i\text{-th} \\ \text{element of the} \\ \text{incidence list of } e}} \qquad \underbrace{\mathrm{port}_i(v)}_{v \text{ is the } i\text{-th port}} \qquad \underbrace{a(e)}_{\substack{\text{hyperedge } e \\ \text{has label } a \in \Sigma.}}$$

The arguments of the relations are e and v, while i and a are parameters. Each choice of parameters gives a different relation.

Recognisability holds also for slight strengthening of MSO, called *counting* MSO, which also allows the following form of modulo counting: for every $n \in \{2, 3, \ldots\}$ there is a predicate

$$|X| \equiv 0 \mod n,$$

which inputs a set and says if the size of this set is divisible by n. The modulo counting predicate is a second-order predicate, since it inputs a subset of the universe, and not an element (or tuple of elements) in the universe.

Counting MSO is more powerful than MSO without counting, e.g. "the number of vertices is even" can be defined in counting MSO but not in MSO.

Theorem 6.12 (Courcelle's Theorem)**.** *If a language $L \subseteq \mathsf{H}\Sigma$ is definable in counting* MSO*, over the incidence model, then it is recognisable, i.e. recognised by a homomorphism into an algebra that is finite on every arity.*

The theorem can be proved using Ehrenfeucht-Fraïssé games, but we choose a more algebraic approach, which uses powerset algebras. These algebras are introduced in the following lemma.

Lemma 6.13. *The recognisable languages images under functions of the form*

$$\underbrace{\mathsf{H}f : \mathsf{H}\Sigma \to \mathsf{H}\Gamma \qquad \textit{for } f : \Sigma \to \Gamma}_{\textit{such functions are called letter-to-letter homomorphisms.}}$$

Proof We will use a powerset construction for algebras. Since we have already used powerset constructions before, we try to discuss this construction here in greater generality. This will give us an opportunity to think about the kinds of monads that allow a powerset construction for algebras (hint: these are not all monads, e.g. the group monad does not have a powerset construction).

For a ranked set X, define its *powerset* to be the ranked set $\mathsf{P}X$ where elements of arity n are sets of elements from A that have arity n. For an arity-preserving function $f : X \to Y$ on ranked sets, define

$$\mathsf{P}f : \mathsf{P}X \to \mathsf{P}Y$$

to be the arity-preserving function that maps a set to its image. It is easy to see that P is a functor. Define *distribution on* X to be the function of type

$$\mathsf{HP}X \to \mathsf{PH}X$$

which inputs a hypergraph G, and outputs the set of hypergraphs that can be obtained from G by choosing for each edge an element of its label.

Claim 6.14. *Distribution is a natural transformation, which means that the following diagram commutes for every arity-preserving function $f : X \to Y$*

$$\begin{tikzcd}
\mathsf{HP}X \arrow[r, "\mathsf{HP}f"] \arrow[d, "distribute\ on\ X"'] & \mathsf{HP}Y \arrow[d, "distribute\ on\ Y"] \\
\mathsf{PH}X \arrow[r, "\mathsf{PH}f"'] & \mathsf{PH}Y
\end{tikzcd}$$

Proof A simple check. One useful property of distribution is that if we apply distribution on X to some hypergraph $G \in \mathsf{HP}X$, then every hypergraph in the resulting set will have the same vertices, ports and hyperedges as G. This useful property would not hold, for example, in the group monad; in fact the claim would be false in the group monad[5]. □

We will use the powerset and distribution to prove the lemma. Suppose that a language L is recognised by a homomorphism

$$h : \mathsf{H}\Sigma \to A,$$

and consider a letter-to-letter homomorphism

$$\mathsf{H}f : \mathsf{H}\Sigma \to \mathsf{H}\Gamma.$$

[5] In fact, there is no powerset construction for algebras in the group monad. Nevertheless, by a proof that does not use powerset algebras, one can show that in the group monad the recognisable languages are closed under images of letter-to-letter homomorphisms.

Define g to be the composition of the following two functions:

$$\begin{tikzcd}
\mathsf{HP}\Sigma \arrow[r, "\text{distribute on } \Sigma"] & \mathsf{PH}\Sigma \arrow[r, "\mathsf{P}h"] & \mathsf{P}A.
\end{tikzcd}$$

Claim 6.15. *The function g is compositional.*

Proof Consider the following diagram.

$$\begin{tikzcd}
\mathsf{HHP}\Sigma \arrow[rr, "\mathsf{H}g"] \arrow[dr, "\mathsf{H}(\text{distribute on } \Sigma)"'] \arrow[dddd, "\begin{array}{c}\text{free}\\ \text{multiplication}\\ \text{on } \mathsf{P}\Sigma\end{array}"'] & & \mathsf{HP}A \arrow[dd, "\begin{array}{c}\text{distribute}\\ \text{on } A\end{array}"] \\
& \mathsf{HPH}\Sigma \arrow[ur, "\mathsf{HP}h"'] \arrow[d, "\text{distribute on } \mathsf{H}\Sigma"'] & \\
& \mathsf{PHH}\Sigma \arrow[r, "\mathsf{PH}h"'] \arrow[d, "\mathsf{P}(\text{free multiplication on } \Sigma)"'] & \mathsf{PH}A \arrow[dd, "\begin{array}{c}\mathsf{P}(\text{multiplication}\\ \text{in the algebra } A)\end{array}"] \\
& \mathsf{PH}\Sigma \arrow[dr, "\mathsf{P}h"] & \\
\mathsf{HP}\Sigma \arrow[ur, "\text{distribute on } \Sigma"] \arrow[rr, "g"'] & & \mathsf{P}A
\end{tikzcd}$$

If we prove that the perimeter of the diagram commute, then we will prove that g is compositional. The upper and lower triangular faces commute by definition of g. The upper rectangular face commutes by naturality of distribution from Claim 6.14, and the lower rectangular face commutes because h is a homomorphism. It remains to show that the five-sided face on the left commutes. This again, is proved via simple check, similarly to Claim 6.14. □

Like for any compositional function, the image of g can be equipped with a multiplication operation which turns g into a homomorphism. (This multiplication operation is the two arrows on the right side from the diagram in the proof of the claim.) We will use the homomorphism g to recognise the image of L under $\mathsf{H}f$. Consider the following diagram:

$$\begin{tikzcd}
\mathsf{H}\Gamma \arrow[r, "\mathsf{H}(\text{inverse image under } f)"] \arrow[d, "\text{inverse image under } \mathsf{H}f"'] & \mathsf{HP}\Sigma \arrow[d, "g"] \arrow[dl, "\text{distribute on } \Sigma"'] \\
\mathsf{PH}\Sigma \arrow[r, "\mathsf{P}h"'] & \mathsf{P}A
\end{tikzcd}$$

The top-left triangular face in the diagram commutes by definition of distribution, and the bottom-right triangular face in the diagram commutes by definition of g. A hypergraph $G \in \mathsf{H}\Gamma$ belongs to the image of L under $\mathsf{H}f$ if and only if applying the function on the down-right path in the diagram gives a set that intersects the image $h(L)$. Since the diagram commutes, it follows that the right-down path in the diagram recognises the image $(\mathsf{H}f)(L)$. The right-down path is a homomorphism, as a composition of two homomorphisms. Also, $\mathsf{P}A$ is finite on every arity, because finiteness on every arity is preserved by powersets. □

We now translate the closure property from the above lemma into quantification in logic. Since the letters used by the letter-to-letter homomorphisms in the above lemmas are labels of hyperedges, it will turn out that images under letter-to-letter homomorphisms will correspond to quantification over sets of hyperedges. This motivates the following definition, which will yield a logic that only allows quantification over sets of hyperedges.

Definition 6.16. For a hypergraph $G \in \mathsf{H}\Sigma$, define its *hyperedge set model* as follows. The universe is the subsets of the hyperedges, and it is equipped with the following relations:

$$\underbrace{X \subseteq Y}_{\text{set inclusion}} \qquad \underbrace{X \subseteq a}_{\substack{\text{every}\\ \text{hyperedge}\\ \text{in } X \text{ has}\\ \text{label } a \in \Sigma}} \qquad \underbrace{i \in X[j]}_{\substack{\text{there exists}\\ e \in X \text{ such}\\ \text{that } e[j] \text{ is}\\ \text{the } i\text{-th port}}} \qquad \underbrace{X[i] \cap Y[j] \neq \emptyset}_{\substack{\text{there exist hyperedges}\\ e \in X \text{ and } f \in Y\\ \text{such that } e[i] = f[j]}} \qquad \underbrace{|X| \equiv 0 \mod n.}_{\substack{\text{the number of hyperedges}\\ \text{in } X \text{ is dvisible by } n}}$$

In the above relations, the arguments are the sets X, Y. The labels $a \in \Sigma$ and numbers $i, j, n \in \{1, 2, \ldots\}$ are parameters. Each choice of parameters gives a different relation.

Lemma 6.17. *If a language $L \subseteq \mathsf{H}\Sigma$ is definable in first-order logic using the hyperedge set model, then it is recognisable.*

Proof Same proof as in Lemma 3.22. For a first-order formula

$$\varphi(x_1, \ldots, x_n)$$

define its language to be the set hypergraphs

$$G \in \mathsf{H}(\Sigma \times 2^n)$$

such that the projection of G onto the $\mathsf{H}\Sigma$ satisfies the formula φ, under the valuation which maps variable x_i to the hyperedges which have "true" on the i-th bit of their corresponding bit-vector from 2^n. By induction on formula size, we show that for every first-order formula, its language is recognisable.

The induction step is proved in the same way as in Lemma 3.22: for Boolean combinations we use homomorphisms into product algebras, while for the quantifiers we use the powerset construction from Lemma 6.13.

We are left with the induction base. For the formulas $X \subseteq Y$ and $X \subseteq a$, the corresponding hypergraph language is of the form "every hyperedge has a label in $\Gamma \subseteq \Sigma$". Such languages were shown to be recognisable in Example 42. In the same example, we showed how to count hyperedges modulo some number, thus showing recognisability of the modulo counting relation. Consider now the language which corresponds to the relation

$$i \in X[j].$$

Let h be the function h which maps a hypergraph to the following information: (a) its arity; (b) which ports belong to $X[j]$. This function is easily seen to be compositional, and it has finite image on every arity, and therefore it witnesses recognisability of the language corresponding to $i \in X[j]$. A similar argument works for

$$X[i] \cap Y[j] \neq \emptyset.$$

□

Using the above lemma, we finish the proof of Courcelle's Theorem. There is a minor issue that needs to be resolved. Because the logic from the above lemma can only quantify over sets of hyperedges, it cannot express properties of isolated vertices, i.e. vertices which are not adjacent to any hyperedge. To finish the proof of Courcelle's Theorem, we need to take into account the isolated vertices.

For a hypergraph G, define $\alpha(G)$ to be the hypergraph of same arity obtained from G by removing all isolated vertices, and define $\beta(G)$ to be the hypergraph obtained by removing all non-isolated edges and all hyperedges. By induction on formula size, one can show that every sentence of counting MSO is a equivalent to Boolean combination of sentences, each of which talks about only $\alpha(G)$ or $\beta(G)$. Therefore, to prove Courcelle's Theorem, it is enough to show that for every sentence φ of counting MSO, both of the following languages are recognisable:

$$\underbrace{\{G \in \mathsf{H}\Sigma : \alpha(G) \models \varphi\}}_{L_\alpha} \qquad \underbrace{\{G \in \mathsf{H}\Sigma : \beta(G) \models \varphi\}}_{L_\beta}$$

For the language L_α we use Lemma 6.17. Every set of non-isolated vertices can be represented as

$$X_1[1] \cup \cdots \cup X_n[n],$$

where n is the maximal arity of letters in the finite alphabet. Using this representation, we can quantify over sets of non-isolated vertices by using quantification over sets of hyperedges. It follows that the language L_α can be defined in first-order logic over the hyperedge set model, and is therefore recognisable by Lemma 6.17.

For the language L_β, we observe that it can be defined by checking an ultimately periodic property of the numbers of ports and isolated vertices. For a hypergraph $G \in \mathsf{H}\Sigma$, define $\gamma(G)$ to be the word $a^n b^m$ where n is the number of ports and m is the number of isolated vertices. This word is roughly the same thing as $\beta(G)$, except that it has an order. Therefore, for every sentence φ of counting MSO on graphs, one can easily find a sentence ψ of counting MSO over finite words which makes the following diagram commute:

$$\begin{tikzcd}
\mathsf{H}\Sigma \arrow[r, "\gamma"] \arrow[d, "\beta"'] & a^* b^* \arrow[d, "\psi"] \\
\mathsf{H}\Sigma \arrow[r, "\varphi"'] & \{\text{yes, no}\}
\end{tikzcd}$$

For finite words, counting MSO has the same expressive power as MSO, because modulo counting can be expressed using the order of the word. Since MSO on finite words can only define regular languages, it follows that ψ defines a regular language contained in $a^* b^*$. Every such regular language is defined as the intersection of $a^* b^*$ with a finite Boolean combination of constraints of the form

$$\underbrace{\#_\sigma = n}_{\substack{\text{letter } \sigma \in \{a,b\} \text{ appears} \\ \text{exactly } n \text{ times}}} \qquad \underbrace{\#_\sigma \equiv k \mod n}_{\substack{\text{the number of} \\ \text{appearances of } \sigma \in \{a,b\} \\ \text{is congruent to } k \text{ modulo } p}}$$

It follows that L_β is a finite Boolean combination of languages of the form "exactly n ports", "exactly n isolated vertices", "the number of ports is congruent to k modulo n", "the number of isolated vertices is congruent to k modulo n". All of these languages are easily seen to be recognisable, using a construction similar to Example 42.

This completes the proof of Courcelle's Theorem.

6.2.4 Satisfiability for bounded treewidth

We finish this section with an algorithm for deciding satisfiability of counting MSO. Recall that already first-order logic on graphs has undecidable satisfiability. Of course the undecidability result carries over to the more general setting of hypergraphs and counting MSO. We can recover decidability if we restrict attention to hypergraphs of bounded treewidth.

Theorem 6.18. *The following problem is decidable:*

- **Input.** *A sentence of counting* MSO *and* $k \in \{1, 2, \ldots\}$.
- **Question.** *Is the sentence true in some hypergraph of treewidth at most* k*?*

Proof We use the proof of Courcelle's Theorem, with an emphasis on computability. We say that ranked set is *computable* if its elements can be represented in a finite way, and there is an algorithm which inputs an arity k and either outputs the finite list of all elements of arity k (if there are finitely many), or starts enumerating these elements (if there are infinitely many). We say that an algebra A is *computable* if its underlying ranked set is computable, and its multiplication operation is also computable (the inputs to the multiplication are finite hypergraphs, which can be represented in a finite way).

Free algebras over computable alphabets are computable, all of the algebras that we used as recognisers for the atomic relations in the proof Courcelle's Theorem are computable, and computability is preserved under the products and the powerset construction. Therefore, we get the following computable strengthening of Courcelle's Theorem: given a sentence of counting MSO, which defines a property of hypergraphs over a finite alphabet Σ, we can compute a recognising homomorphism

$$h : \mathsf{H}\Sigma \to A$$

into a computable algebra. The algebra, homomorphism, and accepting set are represented by the corresponding algorithms.

Let $A_k \subseteq A$ be the image under h of all hypergraphs with treewidth at most k. By Theorem 6.9, A_k is equal to the smallest subset of A that contains the letters and which is closed under applying the treewidth k operations in the algebra A. Since A_k is contained in the finite part of A which has arity at most $k + 1$, and there are finitely many treewidth k operations, it follows that A_k can be computed. Finally, we check if A_k contains at least one element of the accepting set. □

Exercises

Exercise 166. (1) Show that H satisfies the monad axioms.

Exercise 167. (1) Consider graphs (not hypergraphs). Show that the existence of an Eulerian cycle can be defined in counting MSO, but not in MSO.

Bibliography

[1] H Appelgate et al. *Seminar on triples and categorical homology theory*. Springer, 1969.

[2] Mustapha Arfi. “Polynomial Operations on Rational Languages”. In: *Symposium on Theoretical Aspects of Computer Science, STACS, Passau, Germany*. 1987, pp. 198–206.

[3] Stephen L. Bloom and Zoltán Ésik. “The equational theory of regular words”. In: *Information and Computation* 197.1 (2005), pp. 55 –89.

[4] Achim Blumensath. “Regular Tree Algebras”. In: *CoRR* abs/1808.03559 (2018).

[5] Mikołaj Bojańczyk and Bartek Klin. “A non-regular language of infinite trees that is recognizable by a sort-wise finite algebra”. In: *Logical Methods in Computer Science* 15.4 (2019).

[6] J. Richard Büchi. “On a decision method in restricted second order arithmetic”. In: *Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr .)* Stanford, Calif.: Stanford Univ. Press, 1962, pp. 1–11.

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

[8] Olivier Carton, Thomas Colcombet, and Gabriele Puppis. “An algebraic approach to MSO-definability on countable linear orders”. In: *The Journal of Symbolic Logic* 83.3 (2018), pp. 1147–1189.

[9] Thomas Colcombet and A. V. Sreejith. “Limited Set Quantifiers over Countable Linear Orderings”. In: *International Colloquium on Automata, Languages and Programming, ICALP, Kyoto, Japan*. Ed. by Magnús M. Halldórsson et al. Vol. 9135. Lecture Notes in Computer Science. Springer, 2015, pp. 146–158.

[10] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Vol. 138. En-

cyclopedia of Mathematics and Its Applications. Cambridge University Press, 2012.

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

[12] Reinhard Diestel. *Graph theory (electronic edition)*. Vol. 173. Graduate texts in mathematics. Springer-Verlag, 2006.

[13] Samuel Eilenberg and Marcel-Paul Schützenberger. *On pseudovarieties*. IRIA. Laboratoire de Recherche en Informatique et Automatique, 1975.

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

[15] 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.

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

[17] J. A. Green and D. Rees. "On semi-groups in which $x^r = x$". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 48.1 (1952), pp. 35–40.

[18] Jörg Flum Heinz-Dieter Ebbinghaus. *Finite Model Theory*. 2nd. Springer Monographs in Mathematics. Springer, 2006.

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

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

[21] 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.

[22] H Läuchli and J Leonard. "On the elementary theory of linear order". In: *Fundamenta Mathematicae* 59.1 (1966), pp. 109–116.

[23] Robert McNaughton. "Testing and generating infinite sequences by a finite automaton". In: *Information and Control* 9 (1966), pp. 521–530.

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

[25] J.-E. Pin and P. Weil. "Polynomial closure and unambiguous product". In: *Theory Comput. Syst.* 30.4 (1997), pp. 383–422.

[26] Thomas Place and Marc Zeitoun. “Going Higher in First-Order Quantifier Alternation Hierarchies on Words”. In: *J. ACM* 66.2 (2019), 12:1–12:65.

[27] Frank D. Ramsey. “On a problem of formal logic”. In: *Proc. of the London Math. Soc.* 30 (1929), pp. 338–384.

[28] Jan Reiterman. “The Birkhoff theorem for finite algebras”. In: *Algebra Universalis* 14.1 (1982), pp. 1–10.

[29] Chloé Rispal and Olivier Carton. “Complementation of Rational Sets on Countable Scattered Linear Orderings”. In: *International Journal of Foundations of Computer Science* 16.04 (Aug. 2005), pp. 767–786.

[30] Hanamantagouda P Sankappanavar and Stanley Burris. “A course in universal algebra”. In: *Graduate Texts Math* 78 (1981).

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

[32] Marcel-Paul Schützenberger. “Sur Le Produit De Concatenation Non Ambigu”. In: *Semigroup Forum* 13 (1976), pp. 47–75.

[33] Saharon Shelah. “The Monadic Theory of Order”. In: *Annals of Mathematics* (1975), pp. 379–419.

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

[35] 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.

[36] J. W. Thatcher and J. B. Wright. “Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic”. In: *Mathematical systems theory* 2.1 (Mar. 1968), pp. 57–81.

[37] 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.

[38] 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