LANGUAGES
RECOGNISED BY FINITE SEMIGROUPS
and their generalisations to objects
such as
TREES AND GRAPHS
with an emphasis on definability in
MONADIC SECOND-ORDER LOGIC.



Mikołaj Bojańczyk

lecture notes for a course given at the
University of Warsaw

2020

Contents

Preface *page* iii

Part I Words 1

1 Semigroups, monoids and their structure 3
1.1 Recognising languages 8
1.2 Green's relations and the structure of finite semigroups 11
1.3 The Factorisation Forest Theorem 20

2 Logics on finite words, and the corresponding monoids 29
2.1 All monoids and monadic second-order logic 30
2.2 Aperiodic semigroups and first-order logic 37
2.3 Suffix trivial semigroups and temporal logic with F only 50
2.4 Infix trivial semigroups and piecewise testable languages 53
2.5 Two-variable first-order logic 61

3 Infinite words 67
3.1 Determinisation of Büchi automata for ω-words 67
3.2 Countable words and $\circ$-semigroups 79
3.3 Finite representation of $\circ$-semigroups 87
3.4 From $\circ$-semigroups to MSO 95

Part II Monads 105

4 Monads 107
4.1 Monads and their Eilenberg-Moore algebras 108
4.2 A zillion examples 117
4.3 Syntactic algebras 127
4.4 The Eilenberg Variety Theorem 139

4.5 Identities and Birkhoff's Theorem 152

Part III Trees and graphs 157

5 Forest algebra 159
5.1 The forest monad 159
5.2 Recognisable languages 162
5.3 Logics for forest algebra 170

6 Hypergraphs of bounded treewidth 186
6.1 Graphs, logic, and treewidth 186
6.2 The hypergraph monad and Courcelle's Theorem 194
6.3 Definable tree decompositions 213

Bibliography 245
Author index 251
Subject index 252

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 use them to recognise languages of finite words.

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

$$(a, b) \in S^2 \quad \mapsto \quad ab \in S,$$

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 semigroup multiplication 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 semigroup multiplication 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.

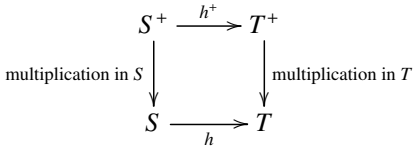
A *semigroup homomorphism* is a function between (underlying sets of) semigroups that preserves the structure of semigroups, i.e. a function

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

which is consistent with the multiplication operation in the sense that

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

where the semigroup multiplication on the left is in S, and the semigroup multiplication on the right is in T. An equivalent definition of a semigroup homomorphism, which views semigroup multiplication as defined on entire words and not just pairs of letters, says that the following diagram must commute:

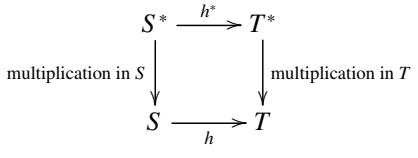


In the above, h^+ is the natural lifting of h to words, which applies h to every letter.

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

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

The identity element, if it exists, must be unique. This is because if there are two candidates for the identity, then multiplying them reveals the true identity. The multiplication operation in a monoid can be thought of as having type $S^* \to S$, 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:



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 con-

catenation, is a semigroup, called the *free*[1] *semigroup over generators* Σ. The *free monoid* is the set Σ^* of possibly empty words.

(2) Every group is a monoid.

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

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

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

The monoid identity is the identity function. The monoid from the previous item is a 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:

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

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

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

[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.

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

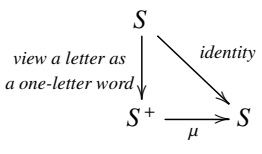
$$(h(a) \circ h(b)) \circ h(c) \overset{(1.1)}{=} (h(ab)) \circ h(c) \overset{(1.1)}{=} h(abc).$$

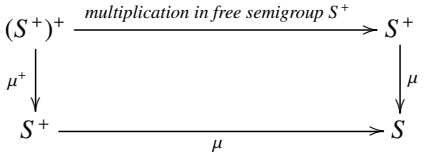
The same reasoning shows that $h(a) \circ (h(b) \circ h(c))$ is equal to $h(abc)$, thus establishing associativity. □

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

As mentioned before, the binary multiplication operation in a semigroup S can be extended to an operation of type $S^+ \to S$. The following lemma explains, using commuting diagrams, which operations of type $S^+ \to S$ arise this way.

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





In the above, μ^+ stands for the coordinate-wise lifting of μ 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 μ of the empty word ε. The axioms

$$1 \cdot a = a = a \cdot 1$$

can then be derived as

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

with a symmetric reasoning used for $a \cdot 1$.

Exercises

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

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

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

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

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

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

Exercise 6. Let Σ be an alphabet, and let

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

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

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

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

Exercise 7. 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. 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[3]. 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 the membership relation $w \in L$ is determined uniquely by $h(w)$. In other words, there is an *accepting 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 $\Rightarrow$ 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 w, the state of the automaton is equal to the homomorphic image of w under the recognising homomorphism, and therefore the accepting subset for the monoid homomorphisms can be used. This

[3] The semigroup approach to languages can be credited to
[50] Schützenberger, "Une théorie algébrique du codage", 1955-1956
On page 10 of this paper, which is primarily devoted to codes, Shützenberger remarks that semigroups can be used to recognise languages and defines the syntactic congruence (in fact, the syntactic pre-order). Apparently, the syntactic congruence dates back to
[24] Dubreil, "Contribution à la théorie des demi-groupes. I", 1941
but I have been unable to obtain a copy of this paper. These are the early days of automata theory, and Shützenberger's paper is contemporary to
[40] Moore, "Gedanken-experiments on sequential machines", 1956 , Theorem 4,
which is the first place that I know where minimisation of automata appears.

automaton computes the monoid multiplication according to the choice of parentheses illustrated in this example:

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

1 ⇒ 2 Let Q be the states of a nondeterministic automaton recognising L. Define a function[4]

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

which sends a word w to the binary relation

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

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

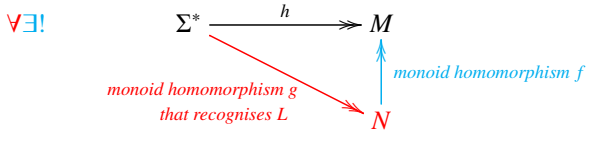
□

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*[5] $L \subseteq \Sigma^*$ *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[6]



[4] This transformation from a nondeterministic (or deterministic) finite automaton to a monoid incurs an exponential blow-up, which is unavoidable in the worst case.

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

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

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

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

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

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

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

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

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

□

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

Exercise 10. 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. 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. Prove that surjectivity of g is important in Theorem 1.7.

Exercise 13. 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. 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[7], which are pre-orders in a semigroup that correspond to 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

$$ee = e.$$

[7] J. A. Green. "On the Structure of Semigroups". In: *Annals of Mathematics* 54.1 (1951), pp. 163–172.

Example 1.8. In a group, there is a unique idempotent, namely the group identity. There can be several idempotents, 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

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

□

Finiteness is important in 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 illustrates Green's relations on the example of the monoid of partial functions on a three element set. The prefix, suffix and infix relations are pre-orders, i.e. they are transitive and reflexive[8]. They need not be anti-symmetric, for example in a group every element is a prefix (and also a suffix and 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.

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

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

for the prefix relation. Likewise, $a \geq_{\mathcal{L}} b$ is used for the suffix 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.

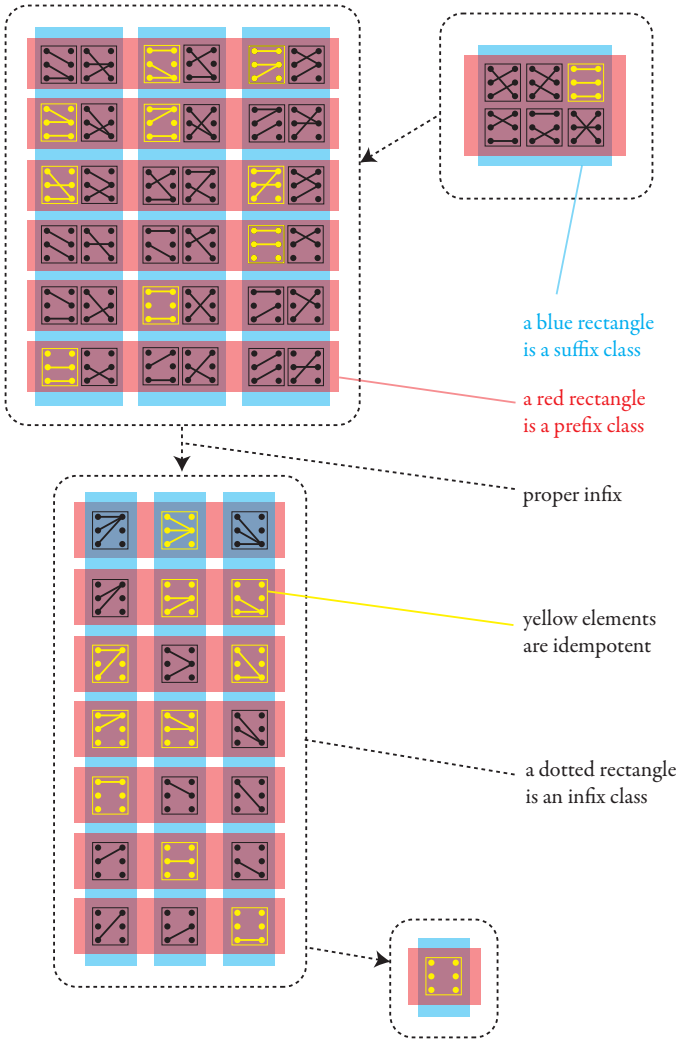


Figure 1.1 The monoid of partial functions from a three element set to itself, partitioned into prefix, suffix and infix classes. In this particular example, the infix classes are totally ordered, which need not be the case in general.

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:*

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

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

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

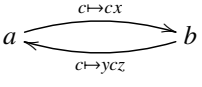
Proof

(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

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

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



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 $a = xby$, and therefore it must be in the same infix class as both of them. We have

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

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

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

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

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

Claim 1.12. *The following maps are mutually inverse bijections*

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

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

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

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

□

The Egg-box Lemma establishes that each infix class has the structure of a rectangular grid (which apparently reminded hungry author 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 an intersection of some prefix class and some suffix class, both taken from some common infix class. By item (2) of the Egg-box Lemma, this intersection is nonempty. 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 multiplying any two elements from the $\mathcal{H}$-class not only falls outside the $\mathcal{H}$-class, but even outside the corresponding 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 multiplication inherited from the semigroup).*

Proof Implications (5) $\Rightarrow$ (1) $\Rightarrow$ (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 multiplication of every two 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. By the reasoning in the previous item, $cd \in G$.

(4)⇒(5) Suppose that G is closed under multiplication, 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 the unique 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. Show that for every finite monoid, the infix class of the monoid identity is a group.

Exercise 16. 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. 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[9].

[9] Let us combine Exercises 16 and 17. By Exercises (16) and the $\mathcal{H}$-class lemma, an infix class

Exercise 18. 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. 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. Show two regular languages such that one is definite and the other is not, but both have isomorphic syntactic monoids.

Exercise 21. 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 satisfy (*).

Exercise 22. Show that every finite semigroup satisfies

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

where quantifiers range over elements of the finite semigroup.

Exercise 23. 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 $w, w' \in (X \cup Y)^+$.

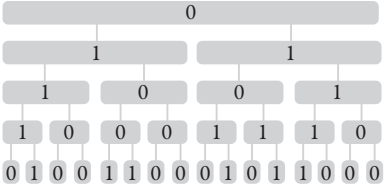
is regular if and only if it contains an $\mathcal{H}$-class which is a group. By Exercise (17), the corresponding group is unique up to isomorphism. This group is called the *Shützenberger group* of the regular infix class.

- **Question.** Is the following true in all finite semigroups:

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

1.3 The Factorisation Forest Theorem

In this section, we show how the multiplication of a long sequence of elements in a semigroup can be organised as a tree, so that in each node of the tree the multiplication is 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:



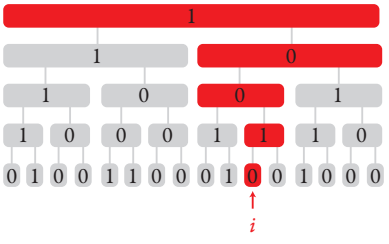
We use the name *factorisation tree* for structures as in the above picture. More formally, a *factorisation tree over a semigroup* S is a tree, where nodes are labelled by semigroup elements, such that every node is either a leaf, or is labelled by the semigroup multiplication of the labels of its children. Since the semigroup in question need not be commutative, the children in a tree are ordered, i.e. there is a first child, second child, etc.

A *binary factorisation* tree is one where every node has zero or two children. For every word in S^+, one can find a corresponding binary 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 finite semigroup that recognises the language. If we assume that the 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 belongs to the accepting set. 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 updated position to the root, as in the following picture:

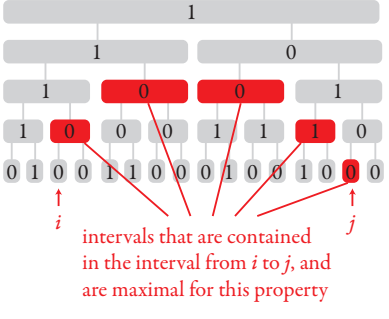


If the factorisation tree is chosen to be balanced, then the updates are processed in logarithmic time. □

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?

Of course, one obvious solution is to pre-compute in quadratic time a table of answers to all possible queries. If we want to solve the problem with linear time pre-computation, then we can use a binary factorisation tree, with respect to some semigroup recognising the language. Suppose that the tree has height k. Each node of the factorisation tree corresponds to an infix of the underlying word. The infix from i to j can be partitioned into at most $2k$ intervals, each of which corresponds to a node of the tree, as in the following picture:



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

In this section, we 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[10].

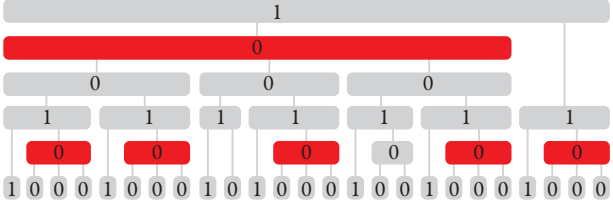
Simon trees

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[11].

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

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

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



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

Theorem 1.16 (Factorisation Forest Theorem)**.** *Let S be a finite semigroup. Every word in S^+ admits a Simon tree of height*[12] $< 5|S|$.

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

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

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

[12] The first version of this theorem was proved in [52, Theorem 6.1], with a bound of $9|S|$. The optimal bound is $3|S|$, which was shown in

Groups. We begin with the special case of groups.

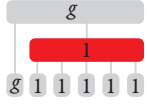
Lemma 1.17. *Let G be a finite group. Every word in G^+ has a Simon tree of height $< 3|G|$.*

Proof Define the *prefix set* of a word $w \in G^+$ to be the set of group elements that can be obtained by multiplying 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:



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 that multiply to g. For the same reasons as in the induction base, every factor w_i with $1 < i < n$ yields the group identity under multiplication.

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

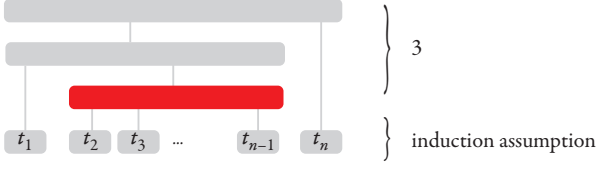
[36] Kufleitner, "The Height of Factorization Forests", 2008 , Theorem 1
The proof here is based on Kufleitner, with some optimisations removed.

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:

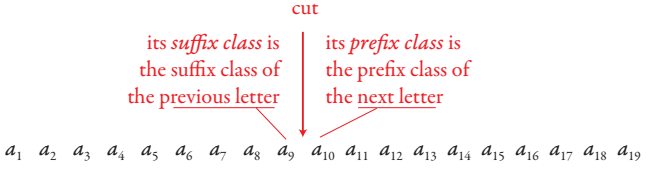


The gray nodes are binary, and the red node is idempotent because every w_i with $1 < i < n$ evaluates to the group identity. □

Smooth words. In the next step, we prove the theorem for words where all infixes have multiplication in the same infix class. We say that a word $w \in S^+$ is *smooth* if every nonempty infix multiplies to 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 has a Simon tree of height $< 4|J|$.*

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

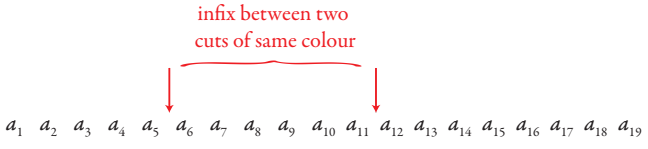


For every cut, both the prefix and suffix classes are contained in J, and therefore they have nonempty intersection thanks to item (2) of the Egg-box Lemma.

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 returns an element of H under multiplication.*

Proof Here is a picture of the situation:



The infix begins with a letter from the prefix class containing H. Since the infix is still in the infix class J, by assumption on smoothness, it follows from item (1) of the Egg-box Lemma that the result of multiplying the infix is in the prefix class of H. For the same reason, the result is also 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 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*

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

Since the number of possible colours is the number of $\mathcal{H}$-classes, the maximal height that can arise from the claim is

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

which proves the lemma. 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

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

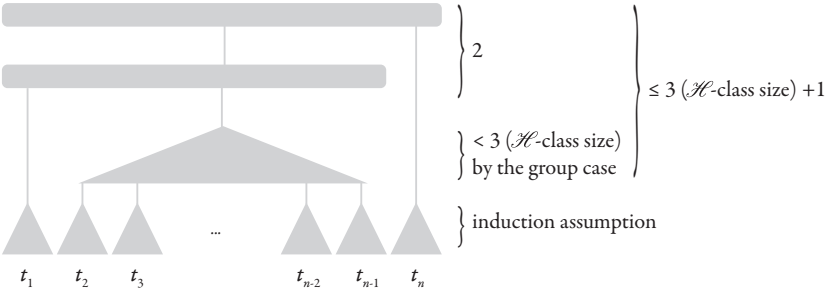
Choose some colour in the colour set of w, which is an $\mathcal{H}$-class H. Cut the word w along all cuts with colour H, yielding a decomposition

$$w = w_1 \cdots w_n.$$

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, multiplying an infix between any two cuts of colour H returns a value in H. In particular, all $w_2, \ldots, w_{n-1}$ yield results in H under multiplication, and the same is true for $w_2 w_3$. It follows that H contains at least one multiplication of two elements from H, and therefore H is a group thanks to item (3) of the $\mathcal{H}$-class Lemma. Therefore, 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. □

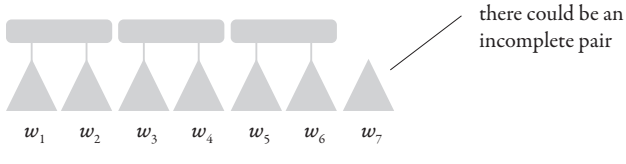
Take a word $w \in S^+$, and cut it along all cuts which are not smooth, yielding a factorisation

$$w = w_1 \cdots w_n.$$

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 multiplying each pair yields a result 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. 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. Show optimality for the previous exercise, in the following sense. Show that for every $k \in \{1, 2, \ldots\}$ there is some semigroup homomorphism

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

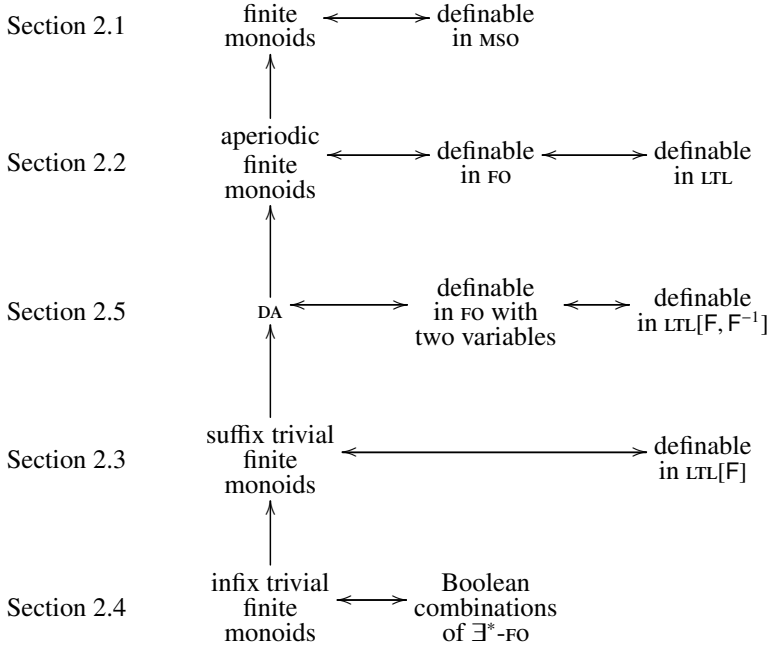
such that for every $n \in \{1, 2, \ldots\}$ there is a word of length at least n^k which does not admit a factorisation $w_0 \cdots w_{n+1}$ where all of $w_1, \ldots, w_n$ are mapped by h to the same idempotent.

Exercise 26. 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, such as formula, model, quantifier or free variable. The following description is meant to fix notation. We use the word *vocabulary* to denote a set of relation names, each one with associated arity in $\{1, 2, \ldots\}$. A *model* over a vocabulary consists of an *underlying set* (also called the *universe* of the model), together with an interpretation, which maps each relation name from the vocabulary to a relation over the universe of same arity. We allow the universe to be empty. For example, a directed graph is the same thing as a model where the universe is the vertices and the vocabulary has one binary relation $E(x, y)$ that represents the edge relation.

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:

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

We use the notation

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

to say that formula 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.

Apart from first-order logic, we also use monadic second-order logic MSO; in fact MSO is the central logic for this book. 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.). This is because

more powerful logic will not be subject to compositionality methods that are discussed in this book.

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, and it is equipped with the following relations:

$$\underbrace{x \leq y}_{\substack{\text{position } x \\ \text{is before} \\ \text{position } y}} \qquad \{\underbrace{a(x)}_{\substack{\text{position } x \\ \text{has label } a}}\}_{a \in \Sigma}.$$

For a sentence φ of MSO over the vocabulary used in the ordered model (this vocabulary depends only on the alphabet Σ), 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, i.e. it does not use set quantification, 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:

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

□

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

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

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

One could imagine other ways of describing a word via a model, e.g. a *successor model* where $x \leq y$ is replaced by a successor relation $x + 1 = y$. The successor relation can be defined in first-order logic in terms of order, but the

converse is not true. Indeed, there are languages that are first-order definable in the ordered model but are not first-order definable in the successor model, see Exercise 39. For the logic MSO, there is no difference between successor and order, since the order can be defined in terms of successor using the logic MSO as follows

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

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, an important example is first-order logic; this restriction and others 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 Trakhtenbrot-Büchi-Elgot theorem is also seminal because it generalises well to structures beyond finite words. For example, there are obvious 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

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

[2] [30] Fagin, "Generalized first-order spectra and polynomial-time recognizable sets", 1974 , Theorem 6

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.

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. The proof is so generic that it will be reused multiple times in future chapters, for structures such as infinite words, trees or graphs.

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 idea is to construct the finite monoid by induction on formula size. In the induction, we also construct monoids for formulas with free variables, so we begin by dealing with those.

Definition 2.3 (Language of formulas with free variables)**.** For an MSO formula

$$\varphi(\underbrace{X_1, \ldots, X_n}_{\text{all free variables are set variables}})$$

which uses the vocabulary of the ordered model for words over alphabet Σ, define its *language* to be the set of words w over alphabet $\Sigma \times \{0,1\}^n$ such that

$$\pi_\Sigma(w) \models \varphi(X_1, \ldots, X_n),$$

where π_Σ is the projection of w onto the Σ coordinate, and X_i is the set of positions whose label has value 1 on the i-th bit of the bit vector from $\{0,1\}^n$.

If the formula φ in the above definition has no free variables, then the above notion of language coincides with Definition 2.1. Therefore, the hard part of the Trakhtenbrot-Büchi-Elgot Theorem will follow immediately from Lemma 2.4 below.

Lemma 2.4. *If $\varphi(X_1, \ldots, X_n)$ is a formula of* MSO *(where all free variables are set variables), then its language is recognised by a finite monoid.*

Proof Before proving the lemma, we observe that first-order variables can be eliminated from MSO, and therefore we can assume that in φ and all of its

sub-formulas, all free variables are set variables. Suppose that we extend MSO with the following predicates that express properties of sets

$$\underbrace{X \subseteq Y}_{\text{set inclusion}} \qquad \underbrace{X \leq Y}_{\substack{x \leq y \text{ holds for} \\ \text{every } x \in X \text{ and} \\ \text{every } y \in Y}} \qquad \underbrace{X \subseteq a}_{\substack{a(x) \text{ holds for} \\ \text{every } x \in X}}. \tag{2.1}$$

The above predicates are second-order predicates in the sense that they express properties of sets; in contrast to the first-order predicates $x \leq y$ and $a(x)$ which express properties of elements. Using the second-order predicates, we can eliminate the first-order variables: instead of quantifying over a position x, we can quantify over a set of positions X, and then say that this set is a singleton:

$$\underbrace{X \neq \emptyset}_{X \text{ is nonempty}} \quad \underbrace{\forall Y\ Y \subseteq X \Rightarrow Y = \emptyset \vee Y = X}_{\text{and every proper subset of } X \text{ is empty}}.$$

Once elements are represented as singleton sets, the first-order predicates $x \leq y$ and $a(x)$ can be simulated using the second-order predicates from 2.1.

Using the transformation described above, from now on we assume that MSO has only set variables, and it uses the second-order predicates from 2.1. For such formulas, we prove the lemma by induction on formula size.

- *Induction base.* In the induction base, we need to show that for every atomic formula as in (2.1), its language is recognised by a finite monoid. Consider for example the formula $X \subseteq a$. The language of this formula consists of words over alphabet $\Sigma \times 2$ where every position has a label that satisfies the following implication:

  $$\text{second coordinate is } 1 \quad \Rightarrow \quad \text{first coordinate is } a.$$

  This language is recognised by the homomorphism into the monoid

  $$(\{0, 1\}, \min)$$

  which maps letters that satisfy the implication to 1 and other letters to 0. Similar constructions can be done for the remaining predicates, in the case of $X \leq Y$ the monoid is not going to be commutative.
- Boolean combinations. For negation, the language of

  $$\neg\varphi(X_1, \ldots, X_n)$$

  is recognised by the same homomorphism as the language of $\varphi(X_1, \ldots, X_n)$, only the accepting set needs to be complemented. For conjunction

  $$\varphi_1(X_1, \ldots, X_n) \wedge \varphi_2(X_1, \ldots, X_n),$$

one uses a product homomorphism

$$(h_1, h_2) : (\Sigma \times 2^n)^* \to M_1 \times M_2 \quad \text{where } h_i : (\Sigma \times 2^n)^* \to M_i \text{ recognises } \varphi_i,$$

with the accepting set consisting of pairs that are accepting on both coordinates. Disjunction $\vee$ reduces to conjunction and negation using De Morgan's Laws.

- *Set quantification.* By De Morgan's Laws, it is enough to consider existential set quantification

$$\exists X_n \, \varphi(X_1, \ldots, X_n)$$

The language of the quantified formula uses alphabet $\Sigma \times 2^{n-1}$. Let

$$h : (\Sigma \times 2^n)^* \to M$$

be a homomorphism that recognises the language of the formula $\varphi(X_1, \ldots, X_n)$, which is obtained by induction assumption. To recognise the quantified formula, we use a powerset construction. Define

$$\pi : (\Sigma \times 2^n)^* \to (\Sigma \times 2^{n-1})^*$$

to be the letter-to-letter homomorphism which removes the last bit from every input position, and define

$$H : (\Sigma \times 2^{n-1})^* \to \mathsf{P}M \qquad H(w) = \{h(v) : \pi(v) = w\}.$$

It is not hard to see that the function H is a homomorphism, with the monoid structure on the powerset $\mathsf{P}M$ defined by

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

The powerset construction clearly preserves finiteness, although at the cost of an exponential blow up. The accepting set consists of those subsets of M which have at least one accepting element.

□

The construction in the above lemma is effective, which means that given a sentence of MSO, we can compute in finite time a recognising monoid homomorphism with an accepting set. Therefore, it is decidable if a sentence of MSO is true in at least one finite word: check if the image of the monoid homomorphism contains at least one accepting element.

As mentioned before, the proof of the "hard" implication in the Trakhtenbrot-Büchi-Elgot Theorem is very generic and will work without substantial changes in other settings, such as infinite words, trees or graphs. The "easy part" will become hard part in some generalisations – e.g. for some kinds of infinite

words or for graphs – because these generalisations lack a suitable automaton model.

Exercises

Exercise 27. Define $\mathcal{U}_2$ to be the monoid with elements $\{a, b, 1\}$ and multiplication

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

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. For an alphabet Σ, consider the model where the universe is the set Σ^* of all finite words, 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. What happens if the prefix relation in Exercise 28 is replaced by the infix relation?

Exercise 30. 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.

Exercise 31. In the proof of the Trakhtenbrot-Büchi-Elgot Theorem, there was an exponential blowup incorred by every set set quantifier. Show that this is optimal, i.e. for every n there is an MSO formula with $O(n)$ set quantifiers such that the smallest model of this formula is a word that has length which is a tower of n exponentials.

2.2 Aperiodic semigroups and first-order logic

Having shown that MSO corresponds to all finite monoids, we now begin the study of fragments of MSO and the corresponding 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, namely the words of odd length which were discussed in Example 4.

Suppose that the parity language is recognised by a homomorphism h into some finite monoid M. By Theorem 1.7 on syntactic monoids, 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. □

As can be guessed from the name, the Shützenberger-McNaughton-Papert-Kamp Theorem is an amalgam of several results, which consider several formalisms. Apart from first-order logic and aperiodic monoids, these formalisms include linear temporal logic and star-free regular expressions, so we begin by defining those.

Linear temporal logic Linear temporal logic[3] (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 2.5 (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 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[4] are

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

We say that an LTL formula is true in a word, without specifying a position, if the formula is true in the first position of that word; 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 $\Sigma a^* b \Sigma^*$. If we add the empty word to this language, then it is still defined by the formula $a \mathsf{U} b$, because the notion of LTL definable language does not take into account the empty word.

Example 6. To get a better feeling for LTL, we discuss some extra operators that can be defined using until, and which will be used later in this chapter. We

[3] This logic, and the theorem about its expressive completeness for first-order logic, is due to [35] Kamp, "Tense Logic and the Theory of Linear Order", 1968 , Theorem II.1. This theorem considers all words where the set of positions is a (possibly infinite) complete linear ordering, which covers the special case of finite words that is considered in this chapter.

[4] We use a variant of the until operator which is sometimes called *strict until*. Strict until is the variant that was originally used by Kamp, see [35, p. viii].

write $\bot$ for any vacuously false formula, such as $a \wedge \neg a$. Likewise $\top$ denotes any vacuously true formula. Here are some commonly used extra operators:

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

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. We now present the final formalism that will appear in the Shützenberger-McNaughton-Papert-Kamp Theorem, namely star-free expressions[5]. 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:

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

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

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

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

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

[5] These were introduced in
[48] Schützenberger, "On finite monoids having only trivial subgroups", 1965 , p. 190.

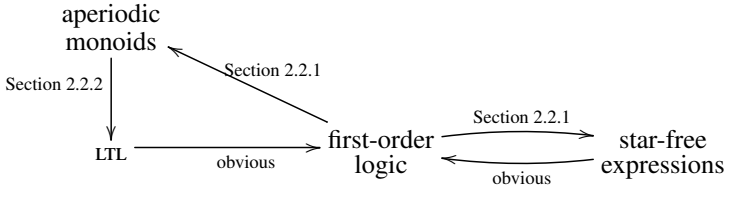
Like for LTL formulas, almost by definition every star-free expression describes a first-order definable language. This is because to every star-free expression one can associate a first-order formula $\varphi(x, y)$ which selects a pair of positions $x \leq y$ if and only if the corresponding infix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

Theorem 2.6 (Shützenberger-McNaughton-Papert-Kamp)**.** *The following are equivalent*[6] *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.

[6] This theorem combines three equivalences.
The equivalence of aperiodic monoids and star-free expressions was shown in
[48] Schützenberger, "On finite monoids having only trivial subgroups", 1965 , p. 190.
The equivalence of star-free expressions and first-order logic was shown in
[39] 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
[35] Kamp, "Tense Logic and the Theory of Linear Order", 1968 , Theorem II.1.

Ehrenfeucht-Fraïssé games. In the proof, we use Ehrenfeucht-Fraïssé games, which are described as follows. An Ehrenfeucht-Fraïssé game is played by two players, called Spoiler and Duplicator. A configuration of the game is a pair of words (one red and one blue), each one with a n-tuple of distinguished word positions

$$\underbrace{w}_{\text{in } \Sigma^*}, \underbrace{x_1, \ldots, x_n}_{\text{positions in } w} \qquad \underbrace{w}_{\text{in } \Sigma^*}, \underbrace{x_1, \ldots, x_n}_{\text{positions in } w}$$

For such a configuration and $k \in \{0, 1, \ldots\}$, the k-round game is played as follows. If there is a quantifier-free formula that distinguishes the two sides (red and blue), then Spoiler wins immediately and the game is stopped. Otherwise, the game continues as follows. If $k = 0$, then Duplicator wins. If $k > 0$ then Spoiler chooses one of the colours red or blue, and a distinguished position x_{n+1} in the word of the chosen colour. Duplicator responds with a matching distinguished position in the word of the other colour, and the game continues with $k - 1$ rounds from the configuration with $n + 1$ distinguished positions, which is obtained by adding the new distinguished positions. This completes the definition of Ehrenfeucht-Fraïssé games.

The point of Ehrenfeucht-Fraïssé games is that they characterise the expressive power of first-order logic, as stated in Theorem 2.7 below. The number of rounds in the games corresponds to quantifier rank of a formula, which is the nesting depth of quantifiers, as illustrated in the following example

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

The correspondence of logic and games is given in the following theorem:

Theorem 2.7. *For every configuration of the game and $k \in \{0, 1, \ldots\}$, Duplicator has a winning strategy in the game if and only if the two sides of the configuration satisfy the same formulas of first-order logic with quantifier rank at most k.*

Proof Straightforward induction on k. □

For $k \in \{0, 1, 2, \ldots\}$ and $w, w \in \Sigma^*$, we write

$$w \equiv_k w$$

if the two words satisfy the same sentences of first-order logic with quantifier rank at most k, or equivalently, Duplicator has a winning strategy in the k-round game over the two words (with no distinguished positions). The follow-

ing lemma 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. *For every $k \in \{0, 1, \ldots\}$ and finite alphabet Σ, the equivalence relation $\equiv_k$ on Σ^* has finitely many equivalence classes. Furthermore, for every words $w, w \in \Sigma^*$ we have $w \equiv_{k+1} w$ 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 Induction on k. The "furthermore" part immediately implies that there are finitely many equivalence classes, since there are finitely many choices for the letter a, and also finitely many choices for the equivalence classes L, K thanks to the induction assumption. Note that the number of equivalence classes for $\equiv_{k+1}$ is exponential in the number of equivalence classes for $\equiv_k$.

It remains to prove the "furthermore" part.

For the left-to-right implication, we observe that LaK can be defined by a first-order sentence of quantifier rank $k+1$, which existentially quantifies over some position x with label a and then checks (using quantifier rank k) that the part before x belongs to L and the part after x belongs to K. Therefore, if w and w satisfy the same sentences of quantifier rank $k+1$, they must belong to the same languages of the form LaK.

Consider now the right-to-left implication. Here it will be useful to consider variant of the Ehrenfeucht-Fraïssé game, call it the *local game*. Consider a configuration of the Ehrenfeucht-Fraïssé game of the form

$$w, x_1, \ldots, x_n \qquad w, x_1, \ldots, x_n.$$

where the red distinguished positions are listed in strictly increasing order $x_1 < \cdots < x_n$, and the same is true for the blue positions. Let us partition the positions x of w into the following $2n+1$ sets, some of which may be empty:

$$\underbrace{X_0}_{x < x_1} \quad \{x_1\} \quad \underbrace{X_1}_{x_1 < x < x_2} \quad \{x_2\} \quad \cdots \quad \underbrace{X_{n-1}}_{x_{n-1} < x < x_n} \quad \{x_n\} \quad \underbrace{X_n}_{x_n < x} \quad . \tag{2.2}$$

Similarly, we partition the positions in the blue word w. We say that a strategy of player Spoiler is local if there is some $i \in \{0, \ldots, n\}$ such that all positions chosen by Spoiler in the strategy belong to X_i (for positions in the red word w) or X_i (for positions in the blue word w).

Claim 2.9. *If Spoiler has a winning strategy, then he also has a local one.*

Proof There is no benefit for Spoiler in using two different blocks of the partition described in (2.2). □

A corollary of this claim is that if $n = 1$, then Spoiler has a winning strategy in the $(k+1)$-round game for the configuration

$$w, x_1 \qquad w, x_1$$

if and only if: (1) the distinguished positions have different labels; or (2) Spoiler has a winning strategy in the k-round game for the parts strictly before the distinguished position; or (3) Spoiler has a winning strategy for the parts strictly after the distinguished position. This gives the right-to-left implication in the lemma. □

We use the lemma above to prove the inclusion of first-order logic in both 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 of $k = 0$, there is only one equivalence class, namely all words, which is clearly a star-free language. Consider now the induction step. Consider an equivalence class M of $\equiv_{k+1}$. Let X be 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, the equivalence class M is equal to the following finite Boolean combination of concatenations

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

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. A corollary of Lemma 2.8 is the following compositionality property for first-order logic on words.

Corollary 2.10. *For every alphabet Σ and $k \in \{0, 1, \ldots\}$, the equivalence relation $\equiv_k$ on Σ^* is a monoid congruence with finitely many equivalence classes.*

Proof Induction on k. To see that there are finitely many equivalence classes, we use Lemma 2.8, which says that an equivalence class of $\equiv_{k+1}$ can be viewed as a set of triples (equivalence class of $\equiv_k$, letter from Σ, equivalence

class of $\equiv_k$), and there are finitely many possible sets of such triples. We now show that $\equiv_{k+1}$ is a monoid congruence, i.e.

$$w \equiv_{k+1} w \text{ and } v \equiv_{k+1} v \qquad \text{implies} \qquad wv \equiv_{k+1} wv.$$

By Lemma 2.8, to prove the conclusion of the above implication, it is enough to show that wv and wv belong to the same languages of the form LaK as in the lemma. This follows immediately from the assumption of the implication, and the induction assumption of the lemma which that $\equiv_k$ is a monoid congruence. (In the proof we also need the observation that $\equiv_{k+1}$ refines $\equiv_k$, which follows from the definition of $\equiv_k$.) □

By the above corollary, the function h_k which maps a word to its equivalence class under $\equiv_k$ is a monoid homomorphism into a finite monoid. This homomorphism recognises every language that is defined by a first-order sentence of quantifier rank at most k, by definition of $\equiv_k$. Therefore, every first-order definable language is recognised by h_k for some k. It remains to show that the monoid used by such a homomorphism is aperiodic. To prove this, we use Lemma 2.8 and a simple 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[7]. 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 2.11. *Let S be a finite aperiodic semigroup, and let $\Sigma \subseteq S$. The colouring*

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

is LTL *definable.*

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

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 multiplication of some word in Σ^+.

The induction base is treated in the following claim.

Claim 2.12. *If either S or Σ has size one, then Lemma 2.11 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

$$a, a^2, a^3, \dots$$

is eventually constant, because all powers bigger than the threshold ! give the same result. The multiplication is therefore easily seen to be an LTL definable colouring, because for every $n \in \{1, 2, \dots\}$ the singleton language

$$\{a^n\} \subseteq \{a\}^+$$

is definable in LTL. For example, when $n = 3$, then the defining formula is

$$\underbrace{\mathsf{FF}\top}_{\substack{\text{there are at least}\\ \text{3 positions}}} \wedge \underbrace{\neg(\mathsf{FFF}\top)}_{\substack{\text{there are strictly}\\ \text{less than 4 positions}}}.$$

□

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

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

Claim 2.13. *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,

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

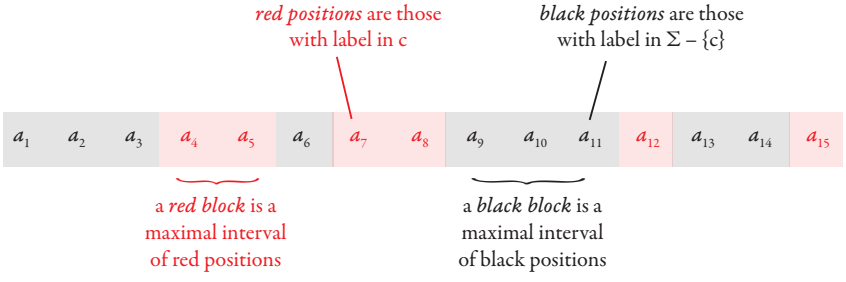
Since permutations form a group, we can multiply both sides by the inverse of $\pi^!$ and conclude that π is the identity permutation. □

If the function $a \mapsto ca$ is the identity for every $c \in \Sigma$, then the multiplication 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 2.14. *T is a sub-semigroup of S.*

Proof Multiplying two elements with prefix c gives an element with prefix c. □

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



We first describe the proof strategy. For each black block, its multiplication 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 red-black blocks, the multiplication operation can be computed in LTL, by using multiplication for the red and black blocks inside it. Also, for every red-black block, its multiplication is in T because it begins with c and has at least two letters. Therefore, we can use the induction assumption on a smaller semigroup, to compute the multiplication of the union of all red-black blocks. Finally, the multiplication 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 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 $f : \Sigma^+ \to \Gamma + \varepsilon$. Under this definition, the output length is at most the input length for LTL transductions. By substituting formulas, one easily[8] 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 multiplication.

(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 multiplication of the infix $\{i, \ldots, j\}$. The function described in this step is an LTL transduction, thanks to the induction assumption on smaller alphabets[9].

(2) Take the output of the function in the previous step, and for each red letter (the multiplication of a red block), multiply it with the next letter (which is the multiplication 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:



[8] The result would also hold for generalisation of LTL transductions where the colouring f has type $\Sigma^+ \to \Gamma^*$, but the proof is easier when the type is $\Sigma^+ \to \Gamma + \varepsilon$, and only the latter case is needed here.

[9] To make this formal, we need a simple closure property of LTL that is described in Exercise 37.

The function in this step is clearly an LTL transduction.

By induction assumption on a smaller semigroup, the multiplication operation $T^+ \to T$ is an LTL colouring. By composing the functions described above with the semigroup multiplication 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 multiplication of the entire word can be computed.

Exercises

Exercise 32. 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 33. 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 34. 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 $x + 1 = y$. Give an example of a regular language that is first-order definable using the ordered model, but not using the successor model.

Exercise 35. 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 36. 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

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

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 37. 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 38. 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 39. 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 40. Show that the identity in Exercise 39, together with aperiodicity, is equivalent to first-order definability in the successor model.

Exercise 41. 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 temporal 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 finite monoids with trivial $\mathcal{H}$-classes (Exercise 33). 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. 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 2.15. [10] *The following conditions are equivalent for $L \subseteq \Sigma^*$:*

(1) *is recognised by a finite suffix trivial monoid;*
(2) *is defined by a finite union of regular expressions of the form*

$$\underbrace{\Sigma_0^* a_1 \Sigma_1^* a_2 \cdots a_n \Sigma_n^* \qquad \text{where } a_i \in \Sigma - \Sigma_i \text{ for } i \in \{1, \ldots, n\};}_{\substack{\text{We call such an expression suffix unambiguous.} \\ \text{A set } \Sigma_i \subseteq \Sigma \text{ is allowed to be empty, in which case } \Sigma_i^* = \{\varepsilon\}.}}$$

(3) *is 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

$$\{w \in M^* : \text{the multiplication of } w \text{ is in } F\}$$

is defined by a finite union of suffix unambiguous expressions. It will follow that for every monoid homomorphism into M, the recognised language is

[10] This theorem is based on
[29] Etessami and Wilke, "An Until Hierarchy and Other Applications of an Ehrenfeucht-Fraïssé Game for Temporal Logic", 2000 , Theorem 6.1
[18] Cohen, Perrin, and Pin, "On the expressive power of temporal logic", 1993 , Theorem 4.2
The result itself is taken from [29]. However, the use of Green's relations in the proof is more in the spirit of [18], which considers an stronger logic that is obtained from LTL[F] by adding a "next" operator.

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

Since our target expressions are closed under finite unions, 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 a suffix of every monoid element, which means that a is a suffix of the monoid identity. By Exercise 15, the infix class of the identity is a group, and a group must be trivial in a suffix trivial monoid. It follows that a word multiplies to a if and only if it belongs to a^*, which is a suffix unambiguous expression.

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

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

Let b be the multiplication 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 that multiply to a are defined by the expression

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

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

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

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

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

by a formula φ_i as in item (3). In the induction base, we use the formula

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

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

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

(3) $\Rightarrow$ (1) Define the rank of a formula in LTL[F] to be the nesting depth of the operator F. (We assume here that F is the only temporal operator used in the formula, and the remaining operators such as G or F^* are replaced by their definitions using 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 2.16. *For every $k \in \{0, 1, 2, \ldots\}$ we have*

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

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 words on both sides of the equivalence

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

have the same first letter, and for every nonempty proper suffix of a word on one side of the equivalence, there there is a nonempty proper suffix on the other side of the equivalence, such that the two suffixes are equivalent under $\approx_k$. Clearly the first letters are the same, because w is nonempty. Consider

now the suffixes. Suppose first that v is a nonempty proper 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

$$(xy)^! = y(xy)^! \qquad \text{for all } x, y \in M. \tag{2.3}$$

The same is also true for syntactic monoids of Boolean combinations of such formulas. To finish the proof, we observe that property (2.3) 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

$$a = y(xy)^! b \overset{(2.3)}{=} (xy)^! b = b.$$

□

Exercises

Exercise 42. Let Σ be an alphabet and let $c \notin \Sigma$ be a fresh letter. Show that $L \subseteq \Sigma^+$ satisfies the conditions of Theorem 2.15 if and only cL is definable in LTL[F].

2.4 Infix trivial semigroups and piecewise testable languages

Having discussed monoids that are $\mathcal{H}$-trivial, prefix-trivial and suffix-trivial in the previous sections, we turn to finite monoids that are infix-trivial. For languages recognised by finite infix trivial monoids, a prominent role will be played embeddings of words (also known as the Higman ordering).

Definition 2.17 (Embedding)**.** We say that a word $w \in \Sigma^*$ *embeds* in a word $v \in \Sigma^*$, 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 on words: 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 2.18 (Higman's Lemma)**.** *For every upward closed $L \subseteq \Sigma^*$ there is a finite subset $U \subseteq L$ such that*

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

Proof Consider the set of minimal elements in L, i.e. the set

$$U = \{w \in L : \text{there is no } v \in L \text{ such that } v \hookrightarrow w \text{ and } v \neq w\}.$$

Because the embedding ordering is well-founded (there are no infinite decreasing chains) it follows that L is equal to the upward closure of its minimal elements U. By definition, U is an *antichain*, which means that every two elements of U are incomparable with respect to embedding. Therefore, to prove the lemma it remains to show that antichains are finite.

Claim 2.19. *There is no infinite antichain with respect to embedding.*

Proof Define a *growth* in a finite or infinite sequence $w_1, w_2, \ldots$ to be a pair of indices $i < j$ such that $w_i \hookrightarrow w_j$. We will show that every infinite sequence contains at least one growth. This implies that there cannot be any infinite antichains.

Suppose, toward a contradiction, that there is an infinite sequence of words without growths. Define the radix ordering on finite words as follows: shorter words come before longer ones, and same length words are ordered lexicographically. Define a sequence $w_1, w_2, \ldots$ by induction as follows. The word w_1 is the least word, in the radix ordering (which is well-founded, so it makes sense to talk about least words), which can be extended to an infinite sequence without growths. For $n > 1$, define w_n to be the least word in the radix ordering such that $w_1, \ldots, w_n$ can be extended to an infinite sequence without growths,

in particular $w_1, \ldots, w_n$ has no growths. Sequences without growths are closed under limits, and therefore $w_1, w_2, \ldots$ has no growths.

Consider the sequence $w_1, w_2, \ldots$ defined in the previous paragraph. Because the alphabet is finite, there must be some letter, call it a, such that infinitely many words in the sequence, say with indexes $n_1 < n_2 < \cdots$, begin with the letter a. Define a new sequence as follows:

$$w_1, \ldots, w_{n_1-1}, \underbrace{w_{n_1}, w_{n_2}, \ldots}_{\substack{\text{the word } w_n \text{ is} \\ \text{is obtained from} \\ \text{the word } w_n \text{ by} \\ \text{removing the} \\ \text{first letter}}}.$$

Since w_{n_1} is shorter than w_{n_1}, it follows from the construction in the previous paragraph that the above sequence must have some growth. However, it is easy to see that any growth in the above sequence would also translate to some growth in the sequence from the previous paragraph, hence a contradiction. □

□

Here is a logical corollary of Higman's lemma.

Theorem 2.20. *A language is upward closed if and only if it can be defined in the ordered model by an $\exists^*$-sentence, i.e. a sentence of the form*

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

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

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. 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 even-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.

Theorem 2.21. *The following conditions are equivalent*[11] *for every* $L \subseteq \Sigma^*$*:*

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

We use the name *piecewise testable* for languages as in item (2) of the above theorem. Equivalence[12] of items (2) and (3) is a corollary of the following lemma, when applied to $K = \Sigma^* - L$.

Lemma 2.22 (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*

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

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:

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

[11] Equivalence of items (1) and (2) was first proved in
[53] Simon, "Piecewise testable events", 1975 , p. 220.
Equivalence of items (2) and (3) was first proved in
[22] Czerwiński, Martens, and Masopust, "Efficient Separability of Regular Languages by Subsequences and Suffixes", 2013 , Theorem 3.

[12] Both conditions (3) and (1) can be checked by algorithms. For (1) this is immediate, while condition (3) is discussed in Exercise 44. Therefore, condition (1) would not be useful for a hypothetical person that only cares about deciding if a regular language is piecewise testable.

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.

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, using the same reasoning as in the proof of König's 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.

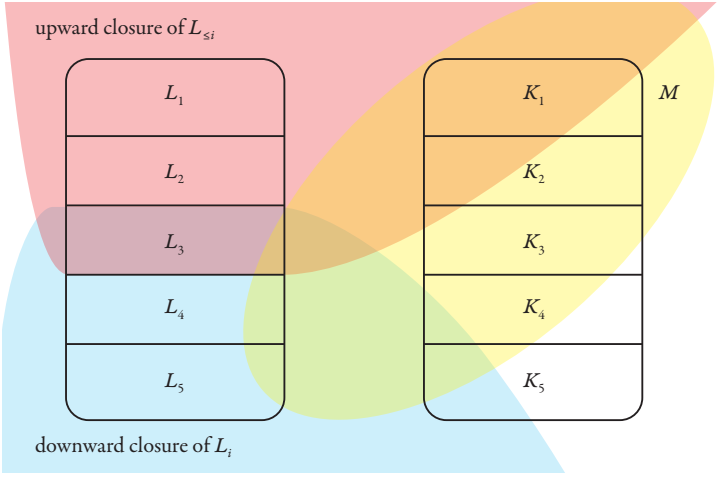
(3)⇒(1) We prove the contra-positive: if zigzags between L and K have bounded length, then L and K can be separated by a piecewise testable language. For $w \in L$ define its *potential* to be the maximal length of a zigzag between L and K that starts in w; likewise we define the potential for $w \in K$, 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 the languages

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

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 go from $i - 1$ to i. We write $L_{<i}$ instead of $L_{\leq i-1}$. We will use the following sets

- the upward closure of $L_{\leq i}$;
- the downward closure of L_i;
- a piecewise testable set M that contains $K_{<i}$ and is disjoint with $L_{<i}$.

The first two sets are piecewise testable because they are upward or downward closed, and the third set is obtained from the induction assumption. These sets are depicted in the following picture, with $i = 3$:



The separator M from the induction assumption contains $K_{<i}$ and is disjoint with $L_{<i}$. Therefore, the piecewise testable language

$$M' = \underbrace{(\text{upward closure of } L_{\leq i})}_{\text{contains } L_{\leq i}} - \underbrace{(M - (\text{downward closure of } L_i))}_{\text{disjoint with } L_{\leq i}}$$

contains $L_{\leq i}$. We will now show that M' is disjoint with $K_{\leq i}$, thus finding a separator as required in the induction. First observe that the upward closure of $L_{\leq i}$ is disjoint with K_i, because otherwise there would be some words

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

and therefore the word w would have potential $i + 1$, and would not belong to $L_{\leq i}$. Therefore, K_i is disjoint with $M' \subseteq L_{\leq i}$. The downward closure of L_i is disjoint with $K_{<i}$, since otherwise there would be some words

$$\underbrace{w}_{K_{<i}} \hookrightarrow \underbrace{v}_{L_i},$$

contradicting the definition of $K_{<i}$. Therefore $K_{<i}$ is contained in

$$\underbrace{M}_{\text{contains } K_{<i}} - \underbrace{(\text{downward closure of } L_i)}_{\text{disjoint with } K_{<i}},$$

and thus $K_{<i}$ is disjoint with M'.

□

The Zigzag Lemma proves the equivalence of the conditions about infinite zigzags and piecewise testability in Theorem 2.21. 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 either infinite or finite but not infix trivial. If the syntactic monoid is infinite, 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 infinite 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 between the languages

$$\{w \in M^* : w \text{ multiplies to } a\} \qquad \{w \in M^* : w \text{ multiplies to } b\}.$$

If M recognises L, then a zigzag between L and its complement can be used, by extraction, to obtain a zigzag between some two distinct monoid elements $a, b \in M$. The following lemma shows that this cannot happen, thus completing the proof of Theorem 2.21.

Lemma 2.23. *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:

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

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

Claim 2.24. *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*

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

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

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

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

Let the letters in w_1 be $c_1, \ldots, c_n \in M$. 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 2.25. *If there is an infinite zigzag between a and b, then either $a = b$, or there exist $c, c' \in M$ such that $a = cc'$ and $cb = b = bc'$.*

Proof Apply Claim 2.24, yielding monoid elements which satisfy the following equalities:

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

For every $i \in \{0, \ldots, n\}$, we can see that $(b_i, a_i) \preceq (b, a)$. If the inclusion is strict, then the induction assumption 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 2.24 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 multiplication 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 and c' as follows:

$$a = \underbrace{c_1 \cdots c_i}_{c} \underbrace{c_{i+1} \cdots c_n}_{c'}.$$

□

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

$$a = cc' \quad cb = b = bc' \quad b = dd' \quad da = a = ad'. \tag{2.4}$$

We can now prove the conclusion of the lemma:

$$a \overbrace{=}^{(2.4)} (dc)^!(c'd')^! \underbrace{=}_{\text{infix triviality}} (cd)^!(d'c')^! \overbrace{=}^{(2.4)} b.$$

□

Exercises

Exercise 43. Prove Higman's Lemma.

Exercise 44. Give a polynomial time algorithm, which inputs two nondeterministic automata, and decides if their languages can be separated by a piecewise testable language.

Exercise 45. 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?

First-order logic with one variable defines exactly the languages which are Boolean combinations for sentences of the form $\exists x\, a(x)$. These languages are 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 2.26. [13] *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: M is aperiodic, and if an infix class $J \subseteq M$ contains an idempotent, then J is a sub-semigroup of M.*

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 $\text{LTL}[\mathsf{F}, \mathsf{F}^{-1}]$ 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 an equational description of DA, which uses the embedding ordering on words that featured prominently in the previous section. (A stronger equational description is given in Exercise 46.)

Lemma 2.27. *A finite monoid M is in* DA *if and only if it satisfies:*

$$\underbrace{w^! = w^! v w^!}_{\textit{same multiplication}} \qquad \textit{for all } w, v \in M^* \textit{ with } v \hookrightarrow w.$$

Proof We first prove that the identity implies that M is DA. The identity clearly implies aperiodicity, by taking $w = v$. Let e be an idempotent. We need to show

[13] The class of monoids from item 2 appears, under the name Df, in
[49] Schützenberger, "Sur Le Produit De Concatenation Non Ambigu", 1976 , p. 47,
where it is used to characterise certain unambiguous regular expressions, see Exercise 50. Subsequent articles use the name DA, which we use here as well. The connection with two variable first-order logic, which is the content of the theorem, is from
[55] Thérien and Wilke, "Over words, two variables are as powerful as one quantifier alternation: $\text{FO}^2 = \Sigma^2 \cap \Pi^2$", 1998 , Theorem 4.

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 a word w in S^+ which multiplies to e and contains both a and b. 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 M is in DA, then the identity is satisfied. Let $v \hookrightarrow w$ be as in the identity. Let e be the multiplication of $w^!$, and let J be the infix class of e. This infix class is a monoid, 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 multiplication in J. Let $a' \in J$ be the multiplication of this suffix. Since J is a monoid, 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 $ev \in J$, and therefore also $eve \in J$. 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 2.27 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

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

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

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

is a colouring definable in FO^2, which means that for every $c \in M$, the inverse image of c under the colouring is a language that 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 multiplication 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) ⇐ (2) in the theorem. It remains to prove the lemma.

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 the 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

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

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

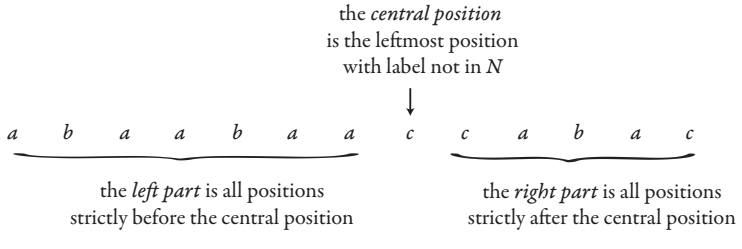
$$\underbrace{\{b \in M : ab \text{ is prefix equivalent to } a\}}_{\text{we call this set the prefix stabiliser of } a}.$$

Proof The prefix stabiliser clearly contains the monoid identity. It remains to show that it is closed under multiplication. 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 2.27, 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:



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

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

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 in the left part.) Let c be the multiplication of the prefix up to and including the central position; as we have shown above, this multiplication can be computed in FO^2. By definition of the central position, we know that a_1 is a proper prefix of $a_1 c$, and therefore we can use the induction assumption to prove that

$$w \quad \mapsto \quad a_1 c \cdot (\text{multiplication of right part}) \cdot a_2$$

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

Exercises

Exercise 46. Show that a monoid belongs to DA if and only if it satisfies the identity

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

Exercise 47. Show that FO^2 has the same expressive power as $\text{LTL}[\mathsf{F}, \mathsf{F}^{-1}]$, which is the extension of $\text{LTL}[\mathsf{F}]$ 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 48. 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 49. Show that L is definable in FO^2 if and only both L and its complement can be defined using $\exists^*\forall^*$-sentences.

Exercise 50. 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
[42] Pin and Weil, "Polynomial closure and unambiguous product", 1997 , Theorem 5.8
The solution which uses Exercise 26 is based on [42]. Characterisations of fragments of first-order logic such as $\exists^*\forall^*$ are widely studied, see
[43] Place and Zeitoun, "Going Higher in First-Order Quantifier Alternation Hierarchies on Words", 2019

[15] This exercise is based on
[49] 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 an ω-word, the positions are ordered like the natural numbers. We show how the structure of finite semigroups described by Green's relations 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 defined to be 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. These have the same syntax as nondeterministic automata on finite words, but they are used to accept or reject ω-words.

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:

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

An ω-word over the input alphabet 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 literature on automata for ω-words has other acceptance conditions, which will not be used in this book. One example is the *Muller condition*, where the accepting set is a family of subsets of states, and a run is accepting if the set of states used infinitely often is a subset that belongs to the accepting family. Another example is the *parity condition*: there is a linear order on the states, and a subset of accepting states, and a run is accepting if the maximal state used infinitely often is accepting.

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. *The following formalisms describe the same languages of ω-words:*

- *nondeterministic Büchi automata;*
- *Boolean combinations of deterministic Büchi automata*[1].

A language is called *ω-regular* if it satisfies either of the two equivalent conditions in the above theorem. 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.

The easier bottom-up implication in Theorem 3.2 follows from the following lemma.

Lemma 3.3. *Languages recognised by nondeterministic Büchi automata are closed under union and intersection, and contain all languages recognised by deterministic Büchi automata and their complements.*

Proof Closure under union is immediate for nondeterministic automata. Consider now the intersection of two nondeterministic Büchi automata $\mathcal{A}$ and $\mathcal{B}$. A nondeterministic Büchi automaton C for the intersection is defined as follows. Take two copies of the product automaton $\mathcal{A} \times \mathcal{B}$. The accepting states are

$$\underbrace{(\text{accepting state}, \text{any state})}_{\text{first copy}} \quad \cup \quad \underbrace{(\text{any state}, \text{accepting state})}_{\text{second copy}}$$

Whenever C sees an accepting state as described above, it switches to the other copy. An accepting run of C must see accepting states of both copies infinitely often, and hence it recognises the intersection of the languages of $\mathcal{A}$ and $\mathcal{B}$.

Since deterministic Büchi automata are a special case of nondeterministic ones, it remains to show that complements of deterministic Büchi automata can be simulated by nondeterministic Büchi automata. The complement of the

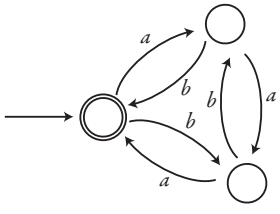
[1] A Boolean combination of deterministic Büchi automata is the same thing as a deterministic automaton with the Muller condition. Therefore, the theorem is the same McNaughton's Theorem,

[38] McNaughton, "Testing and generating infinite sequences by a finite automaton", 1966 , p. 524

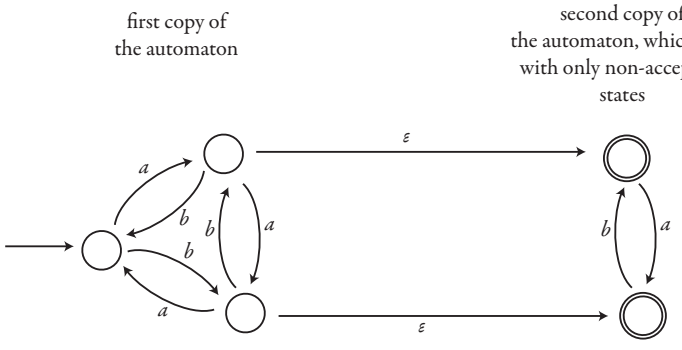
which says that nondeterministic Büchi automata can be determinised into deterministic Muller automata.

[2] [14] Büchi, "On a decision method in restricted second order arithmetic", 1962

language of a deterministic Büchi automaton consists of those words where final states are seen finitely often in the unique run. This can be checked by a nondeterministic Büchi automaton, which nondeterministically guesses the moment where accepting states of the original automaton will no longer be seen. Here is an example. Suppose that we want to complement the deterministic Büchi automaton



which checks that infinitely often, the number of a's is equal to the number of b's modulo 3. The nondeterministic Büchi automaton for the complement looks like this:



In the simulating nondeterministic automaton, the initial states are inherited in the first copy, and the accepting states are all states in the second copy (which correspond to non-accept states in the original deterministic automaton). The above picture uses ε-transitions, which can be easily eliminated. □

We are left with the harder top-down implication in the theorem, which says that every nondeterministic Büchi automaton can be simulated by a Boolean combination of deterministic Büchi automata. There are several combinatorial proofs for the determinisation result in harder implication[3]. In this section,

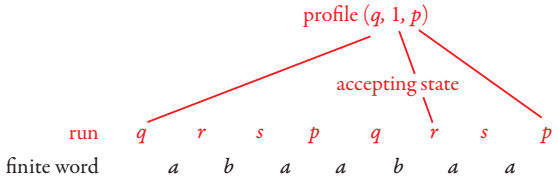
[3] Apart from McNaughton's original proof from [38], another well-known construction is given in

we present an algebraic proof, which leverages the structural theory of finite semigroups described earlier in this book.

Let $\mathcal{A}$ be a nondeterministic Büchi automaton, with states Q and input alphabet Σ. The rest of this section is devoted to finding a Boolean combination of deterministic Büchi automata that is equivalent to $\mathcal{A}$. 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

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

is a semigroup homomorphism, with a naturally defined semigroup structure on S.

The following lemma shows that types for finite and ω-words are compatible with each other.

Lemma 3.4. *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. □

[46] Safra, "On the complexity of the ω-automata", 1988 , Theorem 1.
Another approach, which is based on a construction of Muller and Schupp, is described in [9] Bojańczyk and Czerwiński, *An Automata Toolbox*, 2018 , Section 1.

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; this is the ω-type of some (equivalently, every) ω-word that is obtained by concatenating ω-many finite words with the respective finite 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[4]

Lemma 3.5. *For every $w \in S^\omega$ there exist $a, e \in S$, such that e is an idempotent, $ae = a$, 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 51, 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 3.4 and 3.5 is that $w \in L$ if and only if

(*) there is a factorisation as in Lemma 3.5 such that $ae^\omega \in L$.

So far, we are doing the same argument as in Büchi's original complementation proof from [14]. 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.

[4] This lemma was first observed by Büchi in [14, Lemma 1] where it was used to prove that nondeterministic Büchi automata are closed under complementation, without passing through a deterministic model.

This is the place where we diverge from Büchi's proof, since we are interested in determinisation, while Büchi was interested in complementation. For determinisation, more insight into the structure of finite semigroups will be helpful. Since it is immediately not clear how to express condition (*) using a deterministic Büchi automaton, we will reformulate it. In the reformulation, 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 3.6. *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, $ae = a$, and $ae^\omega \in L$, such that both conditions below 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 3.5. 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 prefix ending in w_i has type a. Let a_i be the type of w_i. The ω-type of w is equal to

$$a_1 e a_2 e a_3 e \cdots .$$

By condition (2), there is some n such that

$$g_i \stackrel{\text{def}}{=} e a_i e$$

is an infix of e for all $i \geq n$. 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 complete the

proof of the lemma as follows:

$$\begin{aligned}
\omega\text{-type of } w &= && (a_1 e a_2 \cdots a_{n-1} e_{n-1} = a)\\
aea_n e a_{n+1} e a_{n+2} e \cdots &= && (e \text{ is idempotent and Lemma 3.4})\\
aea_n eea_{n+1} eea_{n+2} ee \cdots &= && (\text{definition of } g_i)\\
ag_n g_{n+1} g_{n+2} \cdots &= && (\text{by Lemma 3.5, for some } g, f \in G)\\
agf^\omega &= && (\text{because } e \text{ is the unique idempotent in } G)\\
age^\omega &= && (\text{some power of } g \text{ is the idempotent } e)\\
ag^\omega &= && (\text{for the same reason})\\
ae^\omega
\end{aligned}$$

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 3.7. *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 defined as follows:

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

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 3.8. An ω-semigroup[5] consists of:

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

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

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 multiplication operations 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 ω-multiplication can be represented using a commuting diagram, in the same spirit as for Lemma 1.4:

$$\begin{array}{ccc} ((S_+)^+)^\omega & \xrightarrow{\omega\text{-multiplication in free } \omega\text{-semigroup over } S_+} & (S_+)^\omega \\ \downarrow{\scriptstyle (\mu_+)^\omega} & & \downarrow{\scriptstyle \mu_\omega} \\ (S_+)^\omega & \xrightarrow[\mu_\omega]{} & S_\omega \end{array}$$

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

Exercises

Exercise 51. Prove the following result, called *Ramsey's Theorem A* [6] . 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

[5] [41] Perrin and Pin, "Semigroups and automata on infinite words", 1995 , Section 7

[6] [44] Ramsey, "On a problem of formal logic", 1929 , Theorem A

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.

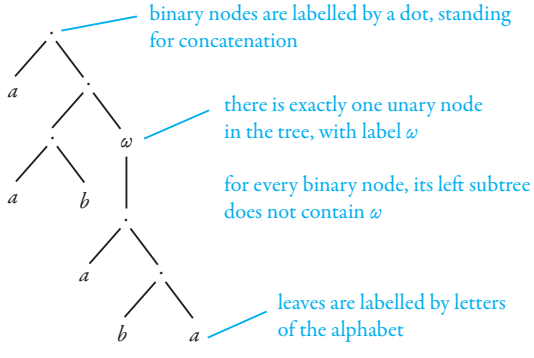
Exercise 52. We say that an ω-word is *ultimately periodic* if it has the form wu^ω, for some finite words $w, u \in \Sigma^\omega$. Show that every nonempty ω-regular language contains an ultimately periodic ω-word.

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

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

Exercise 55. 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. Define an ω-term to be any tree as in the following picture:



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. Let $L \subseteq \Sigma^\omega$. Consider the following equivalence relations on Σ^+.

- Right equivalence is defined by

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

- Two-sided congruence is defined by

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

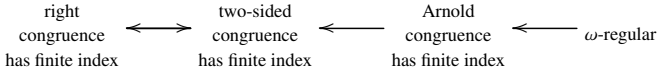
- Arnold congruence is defined by

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

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. 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:



Exercise 59. 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. 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. 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. 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. 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

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

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

Exercise 64. 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. 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. 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. 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:

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

Exercise 68. 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. 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. 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. Show a language of ω-words which does not have a syntactic ω-congruence.

3.2 Countable words and ∘-semigroups

In this section, we move to ∘-words. These are words where the set of 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 some dense set, like the rational numbers. One advantage of ∘-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 ∘-words. There is no known corresponding automaton model, and therefore ∘-semigroups are the only known model of recognisability.

Definition 3.9 (∘-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 ∘-*word over* Σ to be

any isomorphism class of countable[7] Σ-labelled linear orders. We write Σ° for the set of $\circ$-words[8].

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

Example 3.10 (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). The same argument, see below, shows that for every countable Σ there is a unique $\circ$-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 $\circ$-word. Shuffles will play an important role in semigroups for $\circ$-words.

In case the reader is not familiar with back-and-forth arguments, we explain why the shuffle is unique. Define a *finite partial isomorphism* between two $\circ$-words to be a bijection between two finite subsets of their positions which respects the order and labels. Because shuffles have no endpoints and all labels are dense, we conclude the following property:

(*) If f is finite partial isomorphism between two shuffles, and x is a position in the first (respectively, second shuffle), then f can be extended to a finite partial isomorphism whose domain (respectively, co-domain) contains x.

Using the above property and countability, for every two shuffles one can define a sequence

$$f_0 \subseteq f_1 \subseteq f_2 \subseteq \cdots$$

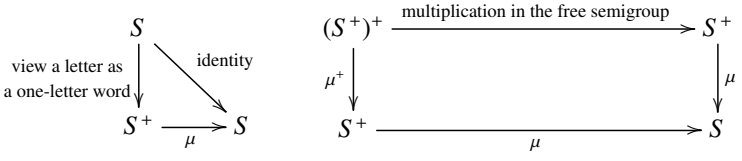
of finite partial isomorphisms such that every position is eventually covered

[7] Why do we assume countability? It turns out that the decidability results 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
[51] Shelah, "The Monadic Theory of Order", 1975 , Theorem 7.
The description of $\circ$-semigroups in this section is based on [51] and
[16] Carton, Colcombet, and Puppis, "An algebraic approach to MSO-definability on countable linear orders", 2018

[8] 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. The same issue and the same solution appears in other places in this book, and we do not mention it explicitly any more.

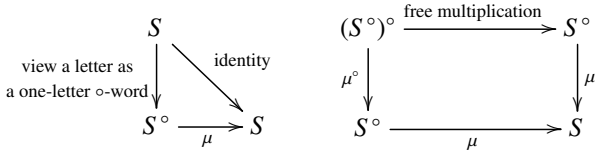
by some f_n. The limit (set union) of these finite partial isomorphisms is an isomorphism between the two shuffles.

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 multiplication on a set S could be defined as any operation $\mu : S^+ \to S$ which makes the following diagram commute:



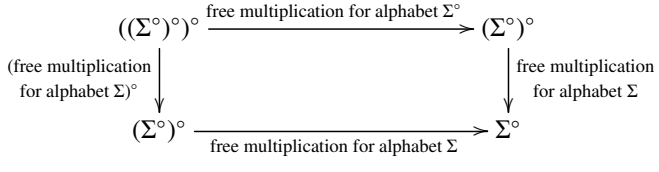
For ∘-semigroups, we take the same approach: we begin by defining a free multiplication operation (which corresponds to multiplication in the free ∘-semigroup), and then define other ∘-semigroups in terms of that. For a set S, define *free multiplication* to be the operation $(S^\circ)^\circ \to S^\circ$ which replaces each position in the input ∘-word with the ∘-word that is in its label (a formal definition uses a lexicographic product of labelled linear orders).

Definition 3.11. A ∘-*semigroup* consists of an underlying set S equipped with a multiplication operation $\mu : S^\circ \to S$, which is associative in the sense that the following two diagrams commute:



In the above diagram, μ° denotes the coordinate-wise lifting of μ to ∘-words of ∘-words.

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



To prove this formally, one uses the formal definition of free multiplication, in terms of lexicographic products of linear orders (see Example 15). 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. $\square$

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

$$\underbrace{(\{0,1\},+)}_{\text{addition mod 2}} \quad (\{0,1\},\min) \quad \underbrace{(\{0,1\},\pi_1)}_{(a,b)\mapsto a} \quad \underbrace{(\{0,1\},\pi_2)}_{(a,b)\mapsto b} \quad (\{0,1\},(a,b)\mapsto 1)$$

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 multiplication a of the ω-word 1^ω would need satisfy

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

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

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:

$$\begin{array}{ccc} S^\circ & \xrightarrow{h^\circ} & T^\circ \\ \text{multiplication in } S \downarrow & & \downarrow \text{multiplication in } T \\ S & \xrightarrow[h]{} & T \end{array}$$

Like for semigroups, homomorphisms of $\circ$-semigroup can be described in terms of compositional functions. Suppose that S is a $\circ$-semigroup and T is a set, which is not yet known to have the structure of a $\circ$-semigroup. We say that a function $h : S \to T$ is *compositional* if there exists a function $\mu : T^\circ \to T$ which makes the following diagram commute

$$\begin{array}{ccc} S^\circ & \xrightarrow{h^\circ} & T^\circ \\ \text{multiplication in } S \downarrow & & \downarrow \mu \\ S & \xrightarrow[h]{} & T \end{array}$$

Using the same proof as for Lemma 1.3, one shows that if h is a compositional and surjective, then μ is necessarily associative, thus turning T into a $\circ$-semigroup, and furthermore h is a homomorphism. A generalised version of

this result, which works not just for ∘-semigroups but also for a wider class of algebraic structures, will be proved in Part II of this book about monads.

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, i.e. ∘-semigroups where the underlying set is finite. Note that it is not immediately clear how to present the multiplication operation of a finite ∘-semigroup in a finite way; this question will be addressed later in this section.

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 gives $w \in \{a\}^\circ$ under free multiplication. 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 multiplications, 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 ∘-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 ∘-semigroup. The element 0 is absorbing, and 1 is a monoid identity. The language L is therefore recognised by the corresponding ∘-semigroup. □

3.2.1 Monadic second-order logic on ∘-words

As usual in this book, we are interested in properties of ∘-words that can be defined using MSO. Part of the appeal of MSO is the ease with which it can be applied to different setups (such as finite or infinite words, trees, graphs, etc.) and ∘-words are no exception. It is immediately clar what MSO for ∘-words should mean. Define the *ordered model* of a ∘-word 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 ∘-words that was discussed in Example 11. This language is definable in MSO, by simply writing in MSO the definition of well-foundedness:

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

Another example is the ∘-words which contain a sub-order that is dense:

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

An ∘-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 ∘-words. The same proof as for finite words (using a powerset construction on ∘-semigroups) gives the following result.

Theorem 3.12. *If a language $L \subseteq \Sigma^\circ$ is definable in* MSO*, then it is recognised by a finite ∘-semigroup.*

The above theorem seems all too easy. Is there a catch? Yes: the theorem alone does not give an algorithm for deciding if an MSO definable language is empty. In the case of finite words, we could remark that all of the constructions used in the proof (products and powersets) are effective, with finite semigroups represented by their multiplication tables. But, so far, we do not have any finite

representation of ∘-semigroups yet, and therefore we cannot talk about effectivity. Such finite representations, and their application to deciding MSO, will be developed in the next section.

To see the difficulty in getting finite representations, consider uncountable words. Theorem 3.12 remains true for uncountable words. However, satisfiability of MSO sentences over uncountable words (e.g. the reals) is undecidable[9] . This means that for uncountable words the constructions in the lemma cannot be made effective. Hence, countability will play a crucial role in finding finite representations.

Another interesting question is about the converse of the theorem: can one define in MSO every language that is recognised by a finite ∘-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 ∘-words, the question is harder. However, the answer is still "yes", and it will be given in Section 3.4.

Exercises

Exercise 72. 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. Find two countable ordinals (viewed as ∘-words over a one letter alphabet), which have the same MSO theory.

Exercise 74. We write $\omega*$ for the reverse of ω. An $(\omega* + \omega)$-word is a ∘-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. 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. 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

[9] [51] Shelah, "The Monadic Theory of Order", 1975 , Theorem 7.

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

Exercise 78. 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. 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. 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. 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 multiplication 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 of quadratic size. In this section, we show that a similar finite representation is also possible for ∘-semigroups. Apart from binary multiplication, 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 3.13 (Läuchli-Leonard operations)**.** For a ∘-semigroup, define its *Läuchli-Leonard operations*[10] to be the following four operations (with their types written in red).

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

The main result of this section is the following theorem, which says that the role played by the Läuchli-Leonard operations in a finite ∘-semigroup is the same as the role played by binary multiplication in a semigroup. A difference with respect to semigroups is that the Läuchli-Leonard operations are complete only for *finite* ∘-semigroups, see Exercise 82.

Theorem 3.14. *The multiplication 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 equipped with the Läuchli-Leonard operations, then there is at most one way of extending these operations to an associative multiplication $S^\circ \to S$. We say at most one instead of exactly one, because the Läuchli-Leonard operations need to satisfy certain associativity axioms, such as:

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

Because the full list of axioms and its completeness proof are both long, we do not consider them here[11]. This will not be a big issue, because we will only consider multiplication operations that arise from compositional functions –

[10] [37] Läuchli and Leonard, "On the elementary theory of linear order", 1966 , p. 109.

[11] They can be found in
[5] Bloom and Ésik, "The equational theory of regular words", 2005 , Section 7.

e.g. the multiplication operation on MSO types of given quantifier rank k – and such multiplication operations are guaranteed to be associative.

Exercises

Exercise 82. Find two infinite $\circ$-semigroups which have the same underlying set and the same Läuchli-Leonard operations, but different multiplication operations.

3.3.1 Proof of Theorem 3.14

The key idea in the proof of Theorem 3.14 is that the Läuchli-Leonard operations are enough to generate all sub-algebras, as stated in the following lemma.

Lemma 3.15. *Let S be a finite $\circ$-semigroup, and let $\Sigma \subseteq S$. Then*

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

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 3.14.

Proof of Theorem 3.14, assuming Lemma 3.15. Suppose that S_1 and S_2 are two $\circ$-semigroups, which have the same underlying set, and where the multiplication operations agree on the Läuchli-Leonard operations. We will show that the multiplication operations are the same. Consider the product $\circ$-semigroup $S_1 \times S_2$, defined in the usual coordinate-wise way. Apply Lemma 3.15 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 the diagonal, which shows that the multiplication operations of S_1 and S_2 are equal. □

The rest of Section 3.3.1 is devoted to proving Lemma 3.15. Define $L \subseteq \Sigma^\circ$ to be the $\circ$-words whose multiplication can be obtained from Σ by applying the Läuchli-Leonard operations. To prove Lemma 3.15, we need to show $L = \Sigma^\circ$. This will follow immediately from the following lemma.

Lemma 3.16. *Let S be a finite ∘-semigroup, and let $L \subseteq S^\circ$ be such that:*

(1) *if $w_1, w_2 \in L$ then $w_1 w_2 \in L$;*
(2) *if $w_1, w_2, \ldots \in L$ have the same multiplication, then $w_1 w_2 \cdots \in L$;*
(3) *if $w_1, w_2, \ldots \in L$ have the same multiplication, then $\cdots w_2 w_1 \in L$;*
(4) *if $w \in L^\circ$ is such that $\mu^\circ(w)$ is a shuffle, then its free multiplication is in L.*

If L contains all letters in a subset $\Sigma \subseteq S$, then L also contains Σ°.

Proof We begin with some notation for ∘-words. Define an *interval* in a ∘-word to be any set of positions X that is connected in the following sense:

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

An *infix* of a ∘-word is defined to be any ∘-word that is obtained by restricting the positions to some interval. For example, the rational numbers – viewed as a ∘-word w over a one letter alphabet $\{a\}^\circ$ – have uncountably many intervals, but five possible infixes, namely a, w, aw, wa and awa.

We now proceed with the proof of the lemma. Suppose that L has all of the closure properties in the assumption of the lemma. We say that $w \in \Sigma^\circ$ is *simple* if not only w, but also all of its infixes are in L. We will show that every ∘-word in 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 the infix corresponding to the interval

$$\underbrace{\{z : x < z \leq y\}}_{\text{an open-closed interval}}$$

is simple (where x is the smaller position and y is the bigger position).

Claim 3.17. *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 holds because simple words are 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 items (1)–(3) from the assumption of the lemma. Consider an equivalence class X. Choose some position $x \in X$. We will show that both intervals

$$\underbrace{\{y \in X : y > x\}}_{Y} \qquad \underbrace{\{y \in X : y \leq x\}}_{Z}$$

are simple, and therefore X itself is simple thanks to binary concatenation. We only consider the first interval Y, and Z is treated with a symmetric argument.

If Y has a last position, then it is simple by definition of $\sim$. If there is no last position, than thanks to countability we can decompose Y as a union

$$Y = Y_0 \cup Y_1 \cup Y_2 \cup \cdots,$$

of consecutive open-closed intervals. By definition of $\sim$, each interval Y_n is simple. Since L is closed under binary concatenation by assumption (1), also every finite union of consecutive intervals Y_n is simple. Therefore, by the Ramsey Theorem, we can assume without loss of generality that all of the intervals $Y_1, Y_2, \ldots$ (but not necessarily Y_0) have the same multiplication $a \in S$. It follows that $Y_1 \cup Y_2 \cup \cdots$ is simple, thanks to assumption (2) about closure of L under ω-concatenation of $\circ$-words with same multiplication. Finally, Y_0 can be added using binary concatenation, thus proving that Y 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 $w_\sim \in S^\circ$ to be the result of replacing every equivalence class of $\sim$ by its multiplication in S. 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 3.18. *Some infix of $w_\sim$ is a shuffle.*

Proof Take some $a \in S$. If there is some infinite infix of $w_\sim$ where no position is labelled by a, then we can continue working in that infix (its positions are still an infinite dense linear order). Otherwise, positions with label a are dense. By iterating this argument for all finitely many elements of S, we find an infinite infix where every $a \in S$ either does not appear at all, or is dense. This infix is a shuffle. □

By the closure of L under shuffles, the free multiplication 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

Thanks to Theorem 3.14, 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 ∘-words.

Recall the proof of Theorem 3.12, which showed that every MSO definable language is recognised by a finite ∘-semigroup. We will show that the constructions in the proof can be made effective, with finite ∘-semigroups being represented using the Läuchli-Leonard operations. In the proof of Theorem 3.12, we inductively transformed the MSO formula into a recognising ∘-semigroup, starting from ∘-semigroups corresponding to the atomic relations in MSO, and then by applying products $S_1 \times S_2$ and powersets $\mathsf{P}S$. It is not hard to find representations (in terms of Läuchli-Leonard operations) for the ∘-semigroups that correspond to the atomic relations. It is also easy to see that given representations of ∘-semigroups S_1 and S_2, one can compute a representation of the product ∘-semigroup $S_1 \times S_2$, because the Läuchli-Leonard operations work coordinate-wise. The interesting case is the powerset construction, which is treated in the following lemma.

Lemma 3.19. *Given a representation (using Läuchli-Leonard operations) of a finite ∘-semigroup S, one can compute a representation of the powerset ∘-semigroup $\mathsf{P}S$.*

Proof In the proof, we use lower-case letters a, b, c for elements of S, and we use upper-case letters A, B, C for elements of the powerset $\mathsf{P}S$. We only show how to compute the multiplication table for the shuffle operation

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

in the powerset ∘-semigroup $\mathsf{P}S$. The remaining Läuchli-Leonard operations are treated in a similar way.

By definition of the powerset ∘-semigroup, an element belongs to the set $\{A_1, \ldots, A_n\}^\eta$ if and only if it can be obtained as follows: take the ∘-word

$$\text{shuffle of } \{A_1, \ldots, A_n\} \quad \in (\mathsf{P}S)^\circ, \tag{3.2}$$

choose for each position an element of its label, and then apply the multiplication operation of S. In other words, a belongs to $\{A_1, \ldots, A_n\}^\eta$ if and only if there exists a word

$$v \in (S \times \mathsf{P}S)^\eta$$

which satisfies the following properties:

(a) after projecting v to the first coordinate and multiplying in S, the result is a;
(b) in every letter of v, the first coordinate belongs to the second coordinate;
(c) after projecting v to the second coordinate, the result is (3.2).

We will construct a homomorphism

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

that recognises the set of $\circ$-words which satisfies conditions (b) and (c) above. The homomorphism h maps a $\circ$-word v to the following information: (i) is condition (b) satisfied; (ii) is v an infix of some $\circ$-word that satisfies (c); (iii) if v is a single letter, then what is the letter; and (iv) does v have a first/last position. The function h defined this way is compositional – and therefore it is a homomorphism – and the Läuchli-Leonard operations on its image T can be computed. Also, the accepting set $F \subseteq T$ can be computed, it consists of elements where the answers to questions (i) and (ii) are both "yes", the answer to question (iii) is "not a single letter", and the answer to question (iv) is "there is neither a first nor last position".

By the above discussion, the set $\{A_1, \ldots, A_n\}^\eta$ that we want to compute consists of those elements $a \in S$ that can be obtained by taking some $v \in h^{-1}(F)$, projecting to the first coordinate, and then applying the multiplication operation of S. Here is alternative description of this set: take the subalgebra of $S \times T$ that is generated by

$$\{(b, h(b, B)) : b \in B \subseteq S\} \quad \subseteq S \times T, \tag{3.3}$$

keep only the pairs from this subalgebra where the second coordinate belongs to the accepting set F, and then project these pairs to the first coordinate. The alternative description can be computed, because we can compute a representation of the product $\circ$-semigroup $S \times T$, and we can compute subalgebras by saturating with respect to the Läuchli-Leonard operations thanks to Theorem 3.14. □

Using the above lemma, we can deduce decidability of MSO over $\circ$-words.

Theorem 3.20. *The following problem is decidable:*

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

Proof By induction on formula size, we compute for each formula of MSO (possibly with free variables), a homomorphism into a finite $\circ$-semigroup that recognises its language, together with an accepting subset of the $\circ$-semigroup. The $\circ$-semigroup is represented using the Läuchli-Leonard operations, and the homomorphism is represented by its images for the letters of the alphabet. In the induction step, we use Lemma 3.19 to compute a finite representation of a powerset $\circ$-semigroup. □

Exercises

Exercise 83. Let Σ be a finite alphabet, and let w be the shuffle of all letters in Σ. Show a finite $\circ$-semigroup which recognises the singleton language $\{w\}$.

Exercise 84. A $\circ$-word w is called *regular* if the singleton language $\{w\}$ is recognised by a finite $\circ$-semigroup. Show that w is regular if and only if it can be constructed from the letters by using the Läuchli-Leonard operations.

Exercise 85. Show that every nonempty MSO definable language $L \subseteq \Sigma^\circ$ contains some regular $\circ$-word.

Exercise 86. Show that if w is a regular $\circ$-word, then $\{w\}$ is MSO definable (without invoking Theorem 3.21).

Exercise 87. 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 88. 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 89. 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 84. 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 88).

Exercise 90. 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 91. 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 84) if and only if it is the yield (in the sense of Exercise 79) of some regular tree.

Exercise 92. (91) Consider the embedding ordering (Higman ordering) $w \hookrightarrow v$ on $\circ$-words. Show that for every $\circ$-words w there is a regular $\circ$-word v such that $w \hookrightarrow v$ and $v \hookrightarrow w$. Hint: use Lemma 3.16.

Exercise 93. Suppose that we are given a language $L \subseteq \Sigma^\circ$, represented by a finite $\circ$-semigroup S, a homomorphism $h : \Sigma^\circ \to S$, and an accepting set $F \subseteq S$. Give a algorithm which computes the syntactic $\circ$-semigroup (which exists by Exercise 78).

Exercise 94. 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 $\circ$-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 $\circ$-semigroup.

Exercise 95. 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 42. Give an algorithm which inputs the finite syntactic $\circ$-semigroup of a language $L \subseteq \Sigma^\circ$, and answers if the language is definable in LTL[F]. Hint: the $\circ$-semigroup must be suffix trivial, but this is not sufficient.

Exercise 96. 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 97. 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 ∘-semigroups to MSO

In Theorem 3.12 we have shown that if a language of ∘-words is definable in MSO, then it is recognised by a finite ∘-semigroup. We now show that the converse implication is also true.

Theorem 3.21. *If a language of ∘-words is recognised by a finite ∘-semigroup, then it is definable in* MSO[12].

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 ∘-words. For a finite family $\mathcal{L}$ of languages of ∘-words, define the shuffle of $\mathcal{L}$ to be the ∘-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 3.22. *Languages definable in* MSO *are closed under Boolean combinations and the following kinds of concatenation:*

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

[12] This theorem was first shown in
[16] 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
[48] 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 ∘-words, see Exercise 103.

Proof For the Boolean operations, there is nothing to do, since Boolean operations are part of the logical syntax. For the remaining operations, which are all variants of concatenation, 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 such that for every two distinct blocks with the same colour, there is a block between them with 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 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 by an MSO formula $\varphi(x, y, X)$ which says that positions x and y are in the same block of the factorisation (i.e. they have the same colour and they are not separated by any position with a different colour).

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 or last block. □

In the proof of Theorem 3.21, 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 $\circ$-semigroup can be defined by a regular expression which uses single letters and the closure operations from the lemma.

To prove Theorem 3.21, we will show that the multiplication operation of every finite $\circ$-semigroup can be defined in MSO, in the following sense. Let S be a finite $\circ$-semigroup. We will show that for every $a \in S$, the language

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

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 position of a in 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 will prove the same thing for every $a \in J$.

We begin with an observation about smooth multiplications, which follows from the Ramsey argument that was used in Theorem 3.2. We say that $w \in S^\circ$ is *J-smooth* if the multiplication of every *finite* infix w is in J. This is a lifting to infinite words of the notion of smoothness that was used in Section 1.3. By the same proof as in 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 multiplication of certain J-smooth words.

Lemma 3.23. *Let $e \in J$ be idempotent, and let $w \in J^\circ$ be J-smooth. If w is an ω-word, then its multiplication is ae^ω, where a depends only on e and the prefix class of the first letter in w. If w is an $(\omega* + \omega)$-word, i.e. its positions are ordered like the integers, then its multiplication is $e^{\omega*}e^\omega$.*

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

$$e^{\omega*}e^\omega = f^{\omega*}f^\omega$$

holds for every two idempotents e, f in the same infix class.

Proof The main observation is the following claim.

Claim 3.24. *If w is an ω-word that is J-smooth and has first letter e, then its multiplication is e^ω.*

Proof By Lemma 3.5, the multiplication 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

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

We now continue as in the proof of Lemma 3.6: 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 b. The first letter admits a decomposition $b = aex$ for

some $a, x \in J$, and furthemore a depends only on the prefix class of b. By the above claim, the multiplication of every J-smooth ω-word that begins with b is equal to ae^ω. A similar argument works when the positions are ordered as the integers: every J-smooth $(\omega* + \omega)$-word has the same multiplication as a smooth $(\omega* + \omega)$-word with an infix ee, and the latter has multiplication $e^{\omega*}e^\omega$ thanks to the claim and its symmetric version for $\omega*$. □

We say that a colouring $\lambda : S^\circ \to C$ which uses a fintie set C of colours is MSO definable on a subset $L \subseteq S^\circ$ 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 multiplication in J. We first show in Lemma 3.25 that the colouring

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

is MSO definable on L_J. Next, in Lemma 3.27, we use this result about prefixes and a symmetric one for suffixes to show that the multiplication operation is MSO definable on L_J. Finally, in Lemma 3.30 we show that the language L_J is definable in MSO. We then conclude as follows: a $\circ$-word has multiplication $a \in J$ if and only if it belongs to L_J, and the colouring from Lemma 3.27 maps it to a. It remains to prove the lemmas.

Lemma 3.25. *The following colouring is* MSO *definable on* L_J*:*

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

Proof We write $H \subseteq S^\circ$ for the $\circ$-words which multiply to a hard element, i.e. an element that is at least as big as J in the infix ordering, or incomparable with J. This language is definable in MSO, as the complement of the language of $\circ$-words that multiply to an easy element, which is definable by induction assumption. For an interval in w, define its multiplication to be the multiplication of the infix of w that is induced by the interval. An interval is called *easy* if its multiplication is easy, otherwise it is called *hard*. The family of easy intervals is closed under subsets. 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 3.26. *The multiplication operation is* MSO *definable on almost easy intervals.*

Proof If there is a last position, then the multiplication can be easily computed: remove the last position, compute the multiplication, and then add the last position. Otherwise, if there is no last position, then we can use Lemma 3.5

to see that an almost easy interval has multiplication $b \in S$ 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 3.22. □

Define a *prefix interval* to be a interval that is downward closed in the ordering of positions. We will compute in MSO the prefix class of some nonempty hard prefix interval; if the ∘-word is in L_J then this hard prefix has the same prefix class as w. Define X to be the union of all easy prefix intervals; if there is no easy prefix interval then this union is empty. This union is an almost easy interval, and therefore its multiplication, call it a, can be computed using Claim 3.26. If a is hard, then we are done, since $a \in J$ by the assumption that $w \in L_J$, and therefore thanks to the Egg-box lemma we can conclude that the prefix class for the multiplication of w is the same as for a.

Suppose now that a is not hard. Define Y to be the suffix interval which is the complement of X. By definition of a, if b is the multiplication of some prefix of Y, then ab is hard, and therefore by the same argument as in the previous paragraph, the prefix class of the multiplication of w is the same as that for ab. Therefore, it remains to compute in MSO the multiplication of some (does not matter which one) prefix of Y. If Y has a first position, then we can simply use the letter in that position. Otherwise, by the Ramsey Theorem, Y can be decomposed as a union of consecutive intervals

$$Y = \cdots \cup Y_2 \cup Y_1 \cup Y_0$$

such that all of the intervals $Y_1, Y_2, \ldots$ have the same multiplication, call it c. If c is easy, which can be defined in MSO thanks to the induction assumption and the closure properties from Lemma 3.22, we know that Y has a prefix which multiplies to $c^{\omega*}$. Otherwise, c is hard, and therefore by Lemma 3.23, we know that Y has a prefix which multiplies to $e^{\omega*}$, where e is some arbitrarily chosen idempotent from J. □

In the above lemma, we have shown how to compute in MSO the prefix class of a ∘-word, conditionally under the assumption that its multiplication is in J. A symmetric argument works for suffix classes. Now we use that result to compute the actual value, still conditionally under the assumption that the multiplication is in J.

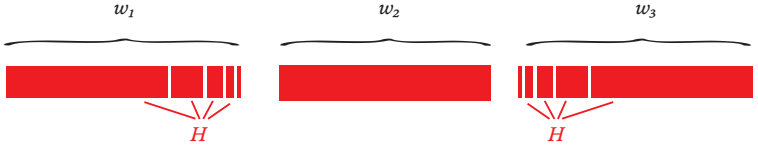
Lemma 3.27. *The multiplication operation of S is* MSO *definable on L_J.*

Proof Let $w \in L_J$. We use the terminology about intervals from the proof of Lemma 3.25 .

Claim 3.28. *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, in the case when w_1 and w_3 are nonempty:



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. 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 is nonempty, because there must be some nonempty easy prefix of w_2 thanks to the assumption that w_2 has no suffix in $H^{\omega *}$), 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 3.23, the multiplication of w_1 is uniquely determined by its prefix class (under the assumption that the entire $\circ$-word belongs to L_J). Therefore, thanks to Lemma 3.25, we can compute in MSO the multiplication of the w_1. Symmetrically, we can compute the multiplication of w_3. It remains to compute the multiplication of w_2. This is done in the following claim.

Claim 3.29. *If a $\circ$-word is a finite concatenation almost easy $\circ$-words, then its multiplication 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 expression, where the atomic expressions describe almost easy words of that multiply to a given element. Such a regular expression can be formalised in MSO thanks to Lemma 3.22 □

□

Lemma 3.30. *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 multiplication is in I. Define L_I to be the ∘-words that multiply to an element of 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, but they are conditionally definable in the sense used by Lemma 3.27.

Claim 3.31. *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 every ∘-word with an error infix is in L_I. 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 3.16, with λ being the multiplication operation in S. The conclusion of Lemma 3.16 will then say that L is equal to S°, thus showing that every ∘-word in L_I has an error infix.

The first assumption of Lemma 3.16 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 multiply to elements in $S - I$, then uv is a binary error.

The remaining assumptions of Lemma 3.16 are checked the same way. □

As remarked before Claim 3.31, 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 3.27, for every $a \in J$ there an MSO definable language which contains all $\circ$-words that have multiplication a, and does not contain any $\circ$-words that have multiplication in $J - \{a\}$. By removing the $\circ$-words with easy multiplications 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 3.31 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 3.22, 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 3.21. Indeed, a $\circ$-word has multiplication $a \in J$ if and only if it belongs to L_J and it is assigned a by the colouring from Lemma 3.27.

Exercises

Exercise 98. 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 99. What is the modification for $\circ$-star-free expressions that is needed to get first-order logic (over the ordered model)?

Exercise 100. Show that if $L \subseteq \Sigma^\circ$ is $\circ$-star-free, then the same is true for every language recognised by its syntactic $\circ$-semigroup.

Exercise 101. Show that if $L \subseteq \Sigma^\circ$ is $\circ$-star-free, then the same is true for L^ω.

Exercise 102. Show that if S is aperiodic, then the constructions from Lemmas 3.25 and 3.27 can be done using $\circ$-star-free expressions.

Exercise 103. Show that $L \subseteq \Sigma^\circ$ is $\circ$-star-free if and only if its syntactic $\circ$-semigroup is finite, aperiodic and satisfies[13]:

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

Hint: use Exercises 101 and 102.

Exercise 104. Show that languages of $\circ$-words definable in first-order logic (in the ordered model) are not closed under concatenation LK.

Exercise 105. We say that a multiplication operation $\mu : S^\circ \to S$ is regular-associative if it satisfies the associativity condition from Definition 3.11, 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 $\mu : S^\circ \to S$ is MSO definable and regular-associative, then μ is associative.

Exercise 106. Show that if S is finite and $\mu : S^\bullet \to S$ is regular associative, then it can be extended uniquely to an associative multiplication $\bar{\mu} : S^\circ \to S$. Hint: the MSO formulas defined in the proof of Theorem 3.21 depend only on the Läuchli-Leonard operations of the $\circ$-semigroup S.

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

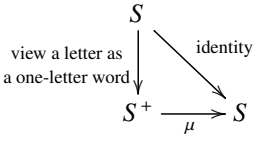
PART TWO

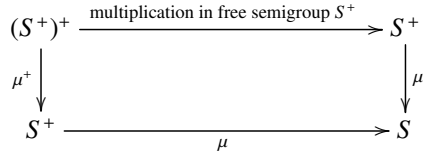
MONADS

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:





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 these 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 useful for less standard algebras such as graphs, where 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 multi-sorted sets 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, and therefore it is easier to see the monad axioms after the definition of Eilenberg-Moore algebras. But even before that, we begin with an example of the monad of finite words, where the Eilenberg-Moore algebras are monoids, to illustrates what we want to do with monads.

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 $f : X \to Y$, its corresponding substitution

$$\underbrace{\mathsf{T}f : \mathsf{T}X \to \mathsf{T}Y}_{\substack{\text{can also be written as} \\ f^* : X^* \to Y^* \\ \text{for this particular monad}}}$$

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

is defined by applying f to every letter in the input word. For a set X, the unit operation of type

$$\underbrace{X \to \mathsf{T}X}_{X \to X^*}, \quad \text{for this particular monad}$$

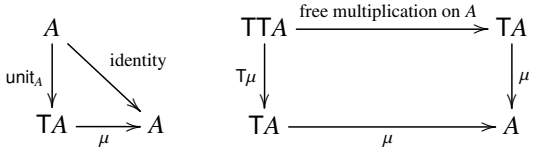
maps a letter to the one-letter word consisting of this letter. Free multiplication, which is a function of type

$$\underbrace{\mathsf{T}\mathsf{T}X \to \mathsf{T}X}_{(X^*)^* \to X^*} \quad \text{for this particular monad}$$

flattens a word of words into a word. The monad of $\circ$-words is defined in the same way, except that it uses $\circ$ instead of $*$.

For this book, the key notion for monads is 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.

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:



The above definition also makes sense for categories other than the category of sets, with A being an object in the category and μ being a morphism. In the definition, the reader will recognise, of course, the diagramatic definitions of semigroups, monoids and $\circ$-semigroups. Also, the diagramatic definition of ω-semigroups 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:

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

The axioms of a group are easily checked, e.g. the axiom $x \cdot x^{-1}$ is proved as follows:

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

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

Having described some intuition behind monads and their Eilenberg-Moore algebras, 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 the 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

$$\begin{array}{ccc} \mathsf{T}X & \xrightarrow{\mathsf{T}f} & \mathsf{T}Y \\ & \searrow^{\mathsf{T}(g \circ f)} & \downarrow{\scriptstyle \mathsf{T}g} \\ & & \mathsf{T}Z \end{array} \tag{4.1}$$

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

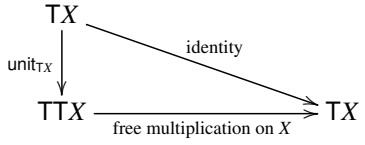
$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ \downarrow{\scriptstyle \mathsf{unit}_X} & & \downarrow{\scriptstyle \mathsf{unit}_Y} \\ \mathsf{T}X & \xrightarrow[\mathsf{T}f]{} & \mathsf{T}Y \end{array} \tag{4.2}$$

$$\begin{array}{ccc} \mathsf{TT}X & \xrightarrow{\mathsf{TT}f} & \mathsf{TT}Y \\ \downarrow{\scriptstyle \text{free multiplication on } X} & & \downarrow{\scriptstyle \text{free multiplication on } Y} \\ \mathsf{T}X & \xrightarrow[\mathsf{T}f]{} & \mathsf{T}Y \end{array} \tag{4.3}$$

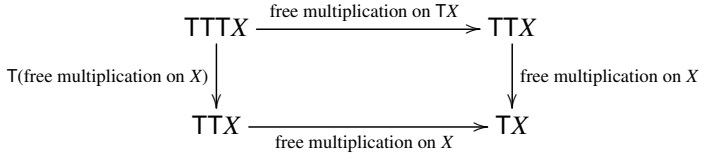
In the language of category theory, this means that the unit and free multiplication are natural transformations. Also, as we will see later on, the second naturality axiom (naturality of free multiplication) says that the substitution $\mathsf{T}f$ is a homomorphism between the free algebras $\mathsf{T}X$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

Associativity. We now turn to the most important monad axioms, which ensure the 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:

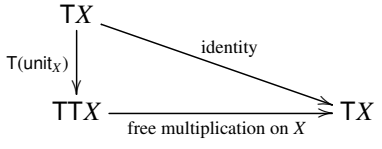


(4.4)



(4.5)

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



(4.6)

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

Exercises

Exercise 107. Consider a monad T in the category of sets. For a binary relation R on a set X, define

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

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

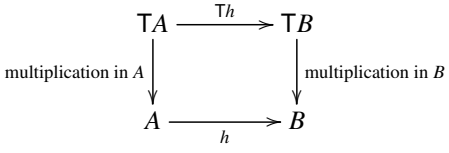
$R^{\mathsf{T}} = \{((\mathsf{T}\pi_1)(t), (\mathsf{T}\pi_2)(t)) : t \in \mathsf{T}R\}$ where $\pi_i : X \times X \to X$ is the i-th projection.

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

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, as formalised in the following definition.

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



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.

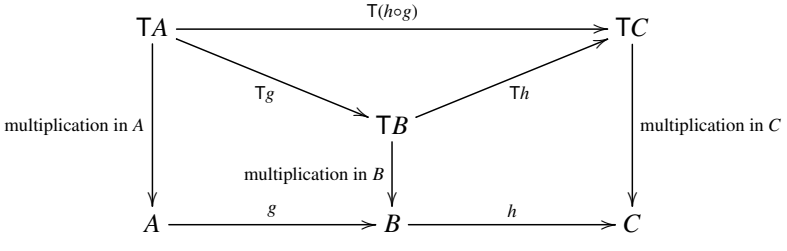
In the rest of this section, we describe some basic properties of homomorphisms.

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:

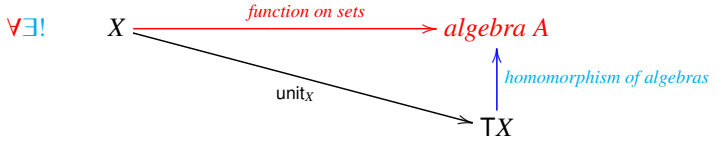


The upper triangular face commutes because of the functoriality axioms (substitutions are compatible with composition). The left and right triangular faces commute by assumption that g and h are homomorphisms. □

Recall that we defined the *free algebra over a set X* to be $\mathsf{T}X$ equipped with the free multiplication operation of type $\mathsf{TT}X \to \mathsf{T}X$. 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:



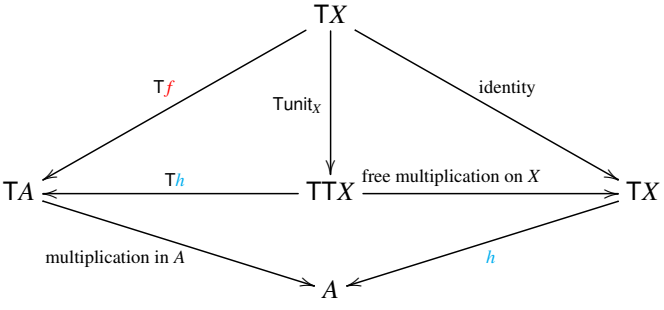
(4.7)

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. Thanks to Lemma 4.5, h is a homomorphism, as the composition of two homomorphisms $\mathsf{T}f$ and μ.

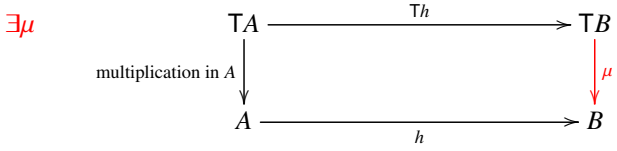
We now show uniqueness – every homomorphism h which makes the diagram must be equal to the one described above. Consider the following diagram:



The upper left triangular 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 triangular face commutes by the first associativity axiom. The lower four-sided 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 $h : A \to B$ on sets is *compositional* if

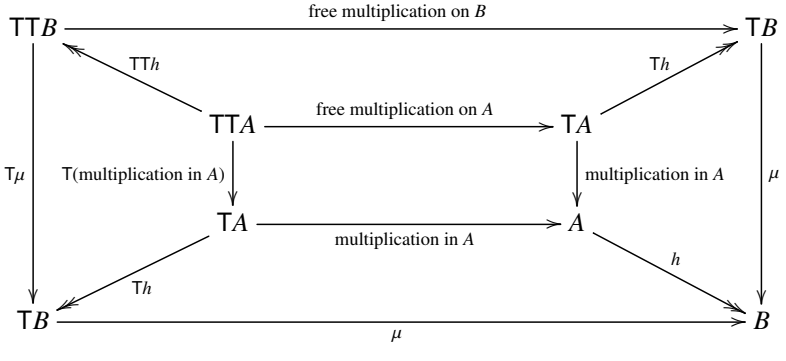


This is the same notion of compositionality as was used for monoids, semigroups and $\circ$-semigroups in part I of the book. For the same reason as before, surjective compositional functions are equivalent to surjective homomorphisms, as stated in the following lemma.

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 an 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, namely the one with a rectangular diagram.

We first observe that T preserves surjectivity of functions[2]. 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.

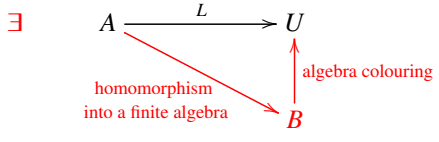


[2] This part of the argument is true for the category of sets, and also for multi-sorted sets, but fails in general, since functors do not need to preserve epimorphisms in general categories.

The central rectangular face commutes by the assumption that A is an algebra. The upper trapezoid face commutes by naturality of free multiplication. The right and lower trapezoid faces commute by definition of a compositional function, and the left trapezoid 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 that we care about, so we omit it.) A language is called *recognisable*[3] 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[4]. A *finite algebra* is an algebra where the underlying set is finite[5]. An algebra colouring $L : A \to U$ is called *recognisable* if it factors through a homomorphism into a finite algebra, as expressed in the following diagram:



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

[3] The definition of recognisable languages for monads appears first in [27] Eilenberg and Wright, "Automata in General Algebras", 1967 , Section 11. The above paper uses Lawvere theories, which correspond to finitary monads (see Section 4.3.3). The main result of [27], Theorem III, concerns free Lawvere theories, which correspond to the monads described in Example 21, and says that recognisable languages for such monads can be described using least fix-points. With the exception of Example 21, none of the monads studied in this book are free.

[4] For some monads, it would 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. Nevertheless, for all monads that are studied in more detail in this book, Definition 4.8 is good enough.

[5] Like for algebra colourings, sometimes this notion of finite algebra is not the right one. In the monad from Example refex:monad-algebra-over-field, 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.

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 use Boolean operations for languages. 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.

Exercises

Exercise 108. 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 107. Show an example of a monad T where this construction does not yield an algebra.

Exercise 109. Does the group monad satisfy the following implication:

(*) 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.

What about surjective homomorphisms?

Exercise 110. Consider the implication in the previous exercise. Show that even if we restrict h to functions of the form $\mathsf{T}f$ for some surjective $f : \Sigma \to \Gamma$, 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.

4.2.1 Monads for words

We begin with several examples of monads that study words, both finite and infinite. We have already discussed finite words in Example 4.2. The following example discusses infinite words, up to a fixed cardinality.

Example 15. [Chains] 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, define a monad T_κ as follows. The set $\mathsf{T}_\kappa X$ consists of chains over X, which have cardinality strictly less than κ. For example, if κ is the first infinite cardinal $\aleph_0$ then the monad describes finite words, and if κ is the first uncountable cardinal then the monad describes $\circ$-words. The monad structure is defined in the same way as for finite words and $\circ$-words. Nevertheless, we give a more exact description below.

For a function f, the corresponding substitution $\mathsf{T}_\kappa f$ is defined by applying f to the labels in the input chain and leaving the positions and ordering unchanged. The unit maps a letter to the chain with one position labelled by 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 of position i in the chain w, call this label $w(i) \in \mathsf{T}_\kappa X$. 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 $\kappa = \kappa^2$.

This is a monad. We only prove one of the monad axioms, namely

$$\begin{array}{ccc} \mathsf{T}_\kappa \mathsf{T}_\kappa \mathsf{T}_\kappa X & \xrightarrow{\text{free multiplication on } \mathsf{T}_\kappa X} & \mathsf{T}_\kappa \mathsf{T}_\kappa X \\ \Big\downarrow{\scriptstyle \mathsf{T}_\kappa(\text{free multiplication on } X)} & & \Big\downarrow{\scriptstyle \text{free multiplication on } X} \\ \mathsf{T}_\kappa \mathsf{T}_\kappa X & \xrightarrow[\text{free multiplication on } X]{} & \mathsf{T}_\kappa X \end{array}$$

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

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

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

If we take κ to be the first cardinal bigger than the continuum cardinal $\mathfrak{c}$, 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 $\mathscr{A}$ 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 in the Trakhtenbrot-Büchi-Elgot Theorem and in Theorem 3.12, one can show that every MSO definable language $L \subseteq \mathsf{T}_\kappa\Sigma$ is recognised by an algebra from $\mathscr{A}$. As we have mentioned on page 3.2.1, satisfiability for MSO over the reals is undecidable, and therefore there is no finite way of representing algebras from $\mathscr{A}$. This means that the powerset construction over finite T_κ-algebras is not computable. □

In the above example, we consider all chains of given cardinality. One can also consider subclasses of chains, subject to some condition on the underlying linear order, as described in the following example.

Example 16. Consider a set $\mathcal{X}$ of linear orders which is 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)[6];
- dense words (the class which contains two orders: a singleton order for units, and the rational numbers).

□

Example 17. [ω-semigroups] We now describe a monad that corresponds to ω-semigroups, see Definition 3.8. Since an ω-semigroup has two sorts, we leave 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

[6] Algebras for the monad of countable scattered words are studied in
[45] Rispal and Carton, "Complementation of Rational Sets on Countable Scattered Linear Orderings", 2005

element is assigned exactly one of two sorts, called + and ω. We use the name *sorted set* for the objects, for the purpose of this example. A morphism in this category is any sort-preserving function between sorted sets. We also use the following notation for sorts:

$$\underbrace{X}_{\text{a sorted set}} = \underbrace{X[+]}_{\substack{\text{elements}\\ \text{of sort } +}} \cup \underbrace{X[\omega]}_{\substack{\text{elements}\\ \text{of sort } \omega}}.$$

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. □

4.2.2 Other monads

We now present two monads – finite multisets and finite sets – which can be viewed as finite words modulo some equalities. Because these monads arise by imposing equalities on finite words, their Eilenberg-Moore algebras for these monads are going to be special cases of monoids.

Example 18. [Finite multisets] Define $\mathsf{T}X$ to be the finite multisets over X. We write finite multisets using red brackets like this

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

A multiset is finite if it has finitely many elements, and each element appears finitely many times. Functions are lifted to multisets point-wise, e.g.

$$\{x_1, \ldots, x_n\} \qquad \overset{\mathsf{T}f}{\mapsto} \qquad \{f(x_1), \ldots, f(x_n)\}.$$

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 output multiset should contain a infinitely many times. To overcome this problem, we could allow multisets with infinitely many copies of an element. □

Example 19. [Idempotent finite words] Define $\mathsf{T}X$ to be finite words X^*, modulo the equation $ww = w$. For example,

$$abcababc = abc(ab)^2c = abcabc = (abc)^2 = abc = (ab)^2c = ababc.$$

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 set, then $\mathsf{T}X$ is finite[7] . It follows that for every finite alphabet, there are finitely many languages over this alphabet, and all of them are recognisable. □

Example 20. [Powersets] The *powerset monad*, and its variant the *finite powerset monad*, are defined in the same way as the multiset monad, except that we use sets (or finite sets) instead of multisets. The substitutions are defined via images (in the language of category theory, we use the co-variant powerset functor, as opposed to the contra-variant powerset functor, which uses inverse images):

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

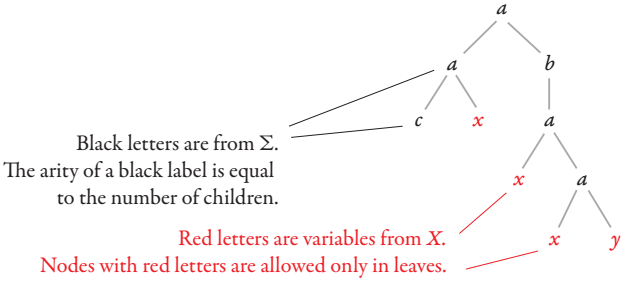
Algebras over the finite powerset monad are the same thing as monoids that are commutative and idempotent. If X is a finite set, then both powerset monads generate finite sets; and therefore all languages over finite alphabets are recognisable. □

[7] [33] Green and Rees, "On semi-groups in which $x^r = x$", 1952 , p. 35

Example 21. [Terms] Fix a ranked set Σ, i.e. a set where every element has an associated arity in $\{0, 1, \ldots\}$. For example, we could have

$$\Sigma = \{\underbrace{a}_{\text{arity 2}}, \underbrace{b}_{\text{arity 1}}, \underbrace{c}_{\text{arity 0}}\}.$$

Based on Σ, we define a monad T_Σ as follows. Define $\mathsf{T}_\Sigma X$ to be the terms over Σ with variables X, i.e. an element of $\mathsf{T}_\Sigma X$ is a tree that looks like this:



The unit operation maps $x \in X$ to a term which consists only of x. The substitution $\mathsf{T}_\Sigma f$ is defined 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 an algebra over the monad T_Σ is the same thing as an *algebra of type* Σ, in the sense of universal algebra, i.e. it consists of an underlying set equipped, with one operation for every letter in Σ[8] . 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[9] . 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[10] , where the input 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. □

[8] [47] Sankappanavar and Burris, "A course in universal algebra", 1981 , Definition 1.3

[9] [54] Thatcher and Wright, "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic", 1968 , Section 2

[10] This monad describes finite trees. Finding an algebraic account for languages of infinite trees remains an open problem. This problem is discussed in the following papers:
[6] Blumensath, "Regular Tree Algebras", 2018
[10] Bojańczyk and Klin, "A non-regular language of infinite trees that is recognizable by a sort-wise finite algebra", 2019

Example 22. [Vector spaces] In this example, we discuss vector spaces over some field. For the sake of concreteness, we use the field of rational numbers. 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

$$q_1 x_1 + \cdots + q_n x_n \qquad \overset{\mathsf{T}f}{\mapsto} \qquad q_1 f(x_1) + \cdots + q_n f(x_n).$$

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:

$$3(4x + 0.5y) - 0.2(5x - 0.1y) \quad \mapsto \quad 12x + 1.5y - x + 0.02y = 11x + 1.52y.$$

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 from the field of rational numbers:

$$qa \overset{\text{def}}{=} \mu(qa).$$

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,

$$2xyx + -2xx + 0.5xyz \in \mathsf{T}\{x, y, z\}.$$

We can view elements of this monad as polynomials with non-commuting variables. 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[11]. On functions, the monad acts as follows

$$q_1 x_1 + \cdots + q_n x_n \qquad \overset{\mathsf{T}f}{\mapsto} \qquad q_1 f^*(x_1) + \cdots + q_n f^*(x_n),$$

where f^* is the substitutions in the monad of finite words. The unit maps x

[11] 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

to the linear combination which has the one-letter word x with coefficient 1. Free multiplication is defined like for polynomials, but the variables are non-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)[12].

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 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 21, where Σ is some ranked set (possibly infinite). Let X be some possibly infinite set of variables, and consider a set

$$\underbrace{\mathcal{E} \subseteq (\mathsf{T}_\Sigma X) \times (\mathsf{T}_\Sigma X)}_{\text{elements of this set will be called identities}}.$$

For a set Y, define $\sim$ to be the least congruence on $\mathsf{T}_\Sigma Y$ that satisfies

$$(\mathsf{T}_\Sigma f)(t_1) \sim (\mathsf{T}_\Sigma f)(t_2) \qquad \text{for every } (t_1, t_2) \in \mathcal{E} \text{ and } f : X \to Y.$$

(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.

[12] 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".

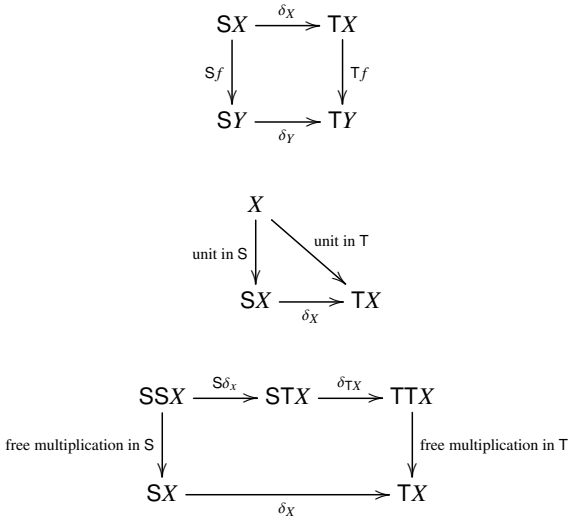


Figure 4.1 These three diagrams should commute for all sets X and all functions $f : X \to Y$. The top diagram says that $\{\delta_X\}_X$ is a natural transformation, while the bottom two diagrams say that it is compatible with unit and free multiplication.

Exercise 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?

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

$$w \in \mathsf{T}\mathsf{T}X \qquad \mapsto \qquad (i \mapsto \underbrace{(w[i])[i]}_{\substack{i\text{-th letter of} \\ \text{of } i\text{-th letter of } w}}).$$

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 chains. Show that a finite algebra with universe S is uniquely determined by the operations

$$\underbrace{ab}_{\substack{\text{binary} \\ \text{multiplication} \\ S^2 \to S}} \qquad \underbrace{a^\omega}_{\substack{\text{multiplication of} \\ aaa\cdots \\ S \to S}}$$

Exercise 117. Consider the monad from Exercise 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 Excercise 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. Give 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

$$\underbrace{ab}_{\substack{\text{binary} \\ \text{multiplication} \\ S^2 \to S}} \qquad \underbrace{a^\omega}_{\substack{\text{multiplication of} \\ aaa\cdots \\ S \to S}} \qquad \underbrace{a^{\omega *}}_{\substack{\text{multiplication of} \\ \cdots aaa \\ S \to S}}$$

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 if an algebra colouring is recognisable, then it has a syntactic homomorphism, i.e. a recognising homomorphism that stores the minimal amount of information[13].

Definition 4.9 (Syntactic homomorphism)**.** Fix a monad T in the category of sets[14]. The *syntactic homomorphism* of an algebra colouring $L : A \to U$ is any surjective homomorphism

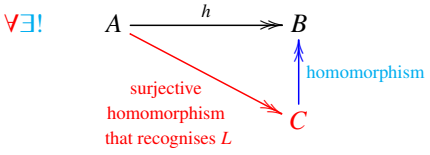
$$h : A \to B$$

which recognises L and which is minimal in the sense explained in the follow-

[13] The results of this section, with the exception of Section 4.3.3, are based on [7] Bojańczyk, "Recognisable languages over monads", 2015 , Part I.

[14] This definition also makes sense for monads in other categories, assuming that one one interprets "surjective functions" as "epimorphisms". However, the results in this book about the existence of syntactic homomorphisms will depend on the category of sets, and its generalisation to sorted sets.

ing quantified diagram



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 unique kernel is called the *syntactic congruence* of L.

We are mainly interested in the case where the algebra colouring describes a language, i.e. A is a free algebra, and there are two colours "yes" and "no".

There are two main results in this section. The first one, Theorem 4.13, says that if an algebra colouring is recognisable, then it has a syntactic homomorphism. In general, colourings that are not recognisable need not have syntactic homomorphisms. The second one, Theorem 4.19, 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. In light of Theorems 4.13 and 4.19, the example uses a colouring that is not recognisable and a monad that is not finitary.

Example 24. Consider the monad of $\circ$-words. (As we will see later, this monad is not finitary.) Define L to be the set of $\circ$-words over a one letter alphabet $\{a\}$ which contain every finite word as an infix. More formally,

$$L = \{w \in \{a\}^\circ : a^n \text{ is an infix of } w \text{ for every } n \in \{0, 1, \ldots\}\}.$$

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

$$w_n = (\text{shuffle of } \{a\}) \cdot a^n \cdot (\text{shuffle of } \{a\}).$$

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 give the same result for all $\circ$-words $w_1, w_2, \ldots$. 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. □

4.3.1 Terms and congruences

To construct the syntactic homomorphism, we will use classical notions from universal algebra, such as terms and congruences, adapted to the monad setting. These notions and their basic properties are described below.

Terms. If X is a set, possibly infinite, then a *term over variables* X is defined simply to be any element of $\mathsf{T}X$. Given an algebra A, a term $t \in \mathsf{T}X$ is interpreted as the following operation

$$\underbrace{\eta \in A^X \quad \mapsto \quad \text{multiply } (\mathsf{T}\eta)(t) \text{ in } A}_{\text{we write } t^A \text{ for this operation}}.$$

Arguments of t^A are called *variable valuations*. An operation of the form t^A, for some t, is called a *term operation*[15] 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 n variables with some implicit ordering, then we write

$$t^A(a_1, \ldots, a_n)$$

for the result of applying t^A to the variable valuation which maps the i-th variable to 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

[15] If we are working in a category other than the category of sets, then the term operation $t^A : A^X \to A$ is not necessarily a morphism in the category. For example, in a category of sorted sets with at least two sorts, the set A^X is not a sorted set.

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 $\circ$-chains, the Läuchli-Leonard operations of ω-power and $\omega*$-power are term operations which arise from the univariate terms x^ω and $x^{\omega*}$. To model shuffling, we use an infinite family of terms, with the n-th one being the shuffle of $\{x_1, \ldots, x_n\}$. □

Term operations commute with homomorphisms, as shown below.

Lemma 4.10. *If $h : A \to B$ is a homomorphism, and $t \in \mathsf{T}X$ is a term, then the following diagram commutes:*

$$\begin{array}{ccc} A^X & \xrightarrow{\eta \mapsto h \circ \eta} & B^X \\ {\scriptstyle t^A}\downarrow & & \downarrow{\scriptstyle t^B} \\ A & \xrightarrow[h]{} & B \end{array} .$$

Proof Consider a term $t \in \mathsf{T}X$ and a valuation $\eta \in A^X$. We show below that if we start in a valuation $\eta \in A^X$, and follow the down-right and the right-down paths in the diagram from the statement, then we get the same element of B.

$$\begin{array}{rcl} \text{down-right path applied to } \eta = & & \\ h(t^A(\eta)) & = & \text{(definition of } t^A\text{)} \\ h((\text{multiplication in } A)(\mathsf{T}\eta)) & = & (h \text{ is a homomorphism}) \\ (\text{multiplication in } B)(h \circ \mathsf{T}\eta)) & = & (\mathsf{T} \text{ is a functor}) \\ (\text{multiplication in } B)(\mathsf{T}(h \circ \eta)) & = & \text{(definition of } t^B\text{)} \\ t^B(h \circ \eta) = & & \\ \text{right-down path applied to } \eta. & & \end{array}$$

□

Congruences. Define a *congruence* in an algebra A to be an equivalence on the underlying set that satisfies any of the equivalent conditions in the following lemma.

Lemma 4.11. *Let A be an algebra. For every equivalence relation $\sim$ on its underlying set, the following conditions are equivalent:*

(1) *the function which maps $a \in A$ to its equivalence class is compositional;*

(2) $\sim$ *is the kernel of some homomorphism from A to some algebra B;*
(3) $\sim$ *commutes with every term operation, which means that:*

$$\underbrace{\eta_1 \sim \eta_2}_{\substack{\eta_1(x) \sim \eta_2(x) \\ \text{for every } x \in X}} \quad \Rightarrow \quad t^A(\eta_1) \sim t^A(\eta_2) \qquad \text{for every } t \in \mathsf{T}X \text{ and } \eta_1, \eta_2 \in A^X.$$

Proof The implication (1)⇒ (2) follows from Lemma 4.7 which says that compositional functions are the same as homomorphisms. The implication (2)⇒(3) follows from Lemma 4.10, which says that term operations commute with homomorphisms. The implication (3)⇒(1) follows from the definition of compositional functions. □

By condition (2), every congruence induces a quotient algebra, where the universe is equivalence classes.

Exercises

Exercise 124. Fix a monad in the category of sets. Consider a set of terms $\mathscr{B}$. 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 term operations corresponding to 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 125. Let T be a monad which has a finite term basis $\mathscr{B}$. Show that given the $\mathscr{B}$-multiplication tables in an algebra A, one can compute the $\mathscr{B}$-multiplication tables of the powerset algebra $\mathsf{P}A$ (as defined in Exercise 108, assuming that $\mathsf{P}A$ is indeed an algebra).

Exercise 126. 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 127. Consider a monad with a term basis $\mathscr{B}$. Show that $\sim$ is an equivalence relation on the underlying set of a finite algebra, then $\sim$ is a congruence if and only if it commutes with all term operations from the term basis $\mathscr{B}$.

Exercise 128. Define a *vectorial term* to be any function

$$f : \underbrace{Y}_{\substack{\text{output}\\ \text{variables}}} \to \mathsf{T} \underbrace{X}_{\substack{\text{input}\\ \text{variables}}}.$$

For an algebra A and a vectorial term f as above, define

$$\underbrace{f^A : A^X \to A^Y}_{\substack{\text{such a function is called}\\ \text{a } \textit{vectorial term operation}}}$$

to be the function which maps $\eta \in A^X$ to the following function:

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

Show that vectorial term operations are closed under composition.

4.3.2 Syntactic homomorphisms for recognisable colourings

In this section, we prove the first result about syntactic homomorphisms, which says that they always exist for algebra colourings that are recognisable.

Theorem 4.13. *Let* T *be a monad in the category of sets. Every recognisable algebra colouring has a syntactic homomorphism.*

The proof is based on congruences. The main result is (a strengthening of) the observation that congruences of finite index in a given algebra, ordered by inclusion when viewed as sets of pairs, form a lattice. This means that every two congruences of finite index have a least upper bound and greatest lower bound. For the greatest lower bound (which is called the *meet* in the terminology of lattices), the observation is straightforward: if $\sim_1$ and $\sim_2$ are congruences, not necessarily of finite index, then their intersection (when viewed as a set of pairs) clearly commutes with all term operations, and is therefore a congruence. Since every lower bound must be contained in the intersection, it follows that the intersection is the greatest lower bound.

The least upper bound (which is called the *join* in the terminology of lattices) is more interesting. Here, we use the assumption on finite index (it is enough that one of the congruences has finite index), see Exercise 129 for why this assumption is needed.

Lemma 4.14. *Let* A *be an algebra. If* $\sim_1$ *and* $\sim_2$ *are congruences and* $\sim_1$ *has finite index, then they have a join congruence, i.e. a least upper bound among all congruences in* A.

Proof Define the join to be the transitive closure of the union of $\sim_1$ and $\sim_2$. In other words, two elements of the algebra are related by the join if one can be reached from the other using a finite number of steps which use either one of the congruences $\sim_1$ or $\sim_2$. This is the same as the join in the lattice of equivalence relations, and therefore every congruence that contains both $\sim_1$ and $\sim_2$ must contain the join defined above. It remains to prove that this join is in fact a congruence, and not just an equivalence relation. To prove this, we use the definition of congruences which says that they commutes with all term operations in the algebra. We begin with the special case of term operations with finitely many variables.

Claim 4.15. *The join commutes with all term operations that have finitely many variables.*

Proof Consider a term operation

$$t^A(x_1, \ldots, x_n)$$

with finitely many variables. We need to show that every two outputs

$$t^A(a_1, \ldots, a_n) \qquad \text{and} \qquad t^A(b_1, \ldots, b_n)$$

are equivalent under the join, assuming that the inputs are pairwise equivalent under the join. This is proved using a finite number of steps, where in each step we use commutation of t^A with either $\sim_1$ or $\sim_2$. □

Using the above claim, and the assumption that $\sim_1$ has finite index, we show that the join commutes with all term operations, even those with infinitely many variables. Consider a term $t \in \mathsf{T}X$ and two valuations

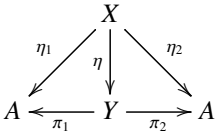
$$\eta_1, \eta_2 : X \to A$$

which are pointwise equivalent with respect to the join. We need to show that applying t^A to both of these valuations gives outputs that are equivalent with respect to the join. We begin with the special case when the two valuations have finite images.

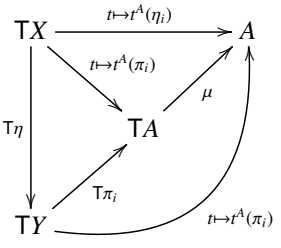
Claim 4.16. *If η_1, η_2 have finite image $B \subseteq A$, then $t^A(\eta_1) = t^A(\eta_2)$.*

Proof Define Y to be the set of pairs in $B \times B$ which are equivalent under $\sim$ and let $\eta : X \to Y$ be the function which maps an variable from X to the pair of its images under η_1 and η_2. Each valuation η_i can be decomposed as first

applying η, and then taking the i-th projection, as in the following diagram:



Applying the term operation t^A to a valuation η_i is the same as applying the term operation $(\mathsf{T}\eta(t))^A$ to the valuation π_i, which is proved by chasing the following diagram:



Since the term operation $(\mathsf{T}\eta(t))^A$ uses a finite set of variables Y, and since the valuations π_1 and π_2 are equivalent under $\sim$, the statement of this claim follows from Claim 4.15. □

Using the above claim, and the assumption that $\sim_1$ has finite index, we conclude the proof of the lemma. Choose a function

$$\alpha : A \to A$$

that maps every element of A to some chosen element in its equivalence class under $\sim_1$. This function has finite image, because $\sim_1$ has finite index. We now conclude the proof of the lemma as follows:

$$\begin{array}{rll} t^A(\eta_1) & \sim_1 & (\sim_1 \text{ is a congruence}) \\ t^A(\alpha \circ \eta_1) & \sim & (\alpha \text{ has finite image and Claim 4.16}) \\ t^A(\alpha \circ \eta_2) & \sim_1 & (\sim_1 \text{ is a congruence}) \\ t^A(\eta_2) & & \end{array}$$

□

Using the above lemma, we complete the proof of Theorem 4.13.

Proof of Theorem 4.13 Consider algebra colouring $L : A \to U$ that is recognisable. Let C be the set of congruences in A that recognise the colouring in

the following sense

$$a \sim b \qquad \text{implies} \qquad L(a) = L(b).$$

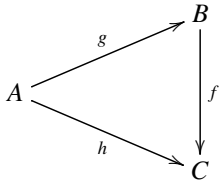
Because the colouring is recognisable, the there is at least one congruence $\approx$ of finite index in C. Define $C_{\approx} \subseteq C$ to be the congruences which contain $\approx$. This is a finite set, since every congruence in $C_{\approx}$ is obtained by merging some of the finitely many equivalence classes in $\approx$. By Lemma 4.14, $C_{\approx}$ is a finite lattice, and therefore it has a greatest element, call it $\sim$. Again by Lemma 4.14, every congruence in C has an upper bound in $C_{\approx}$, and therefore $\sim$ is the greatest element also of C. We will prove that the quotient homomorphism

$$h : A \to A_{/\sim},$$

is the syntactic homomorphism of L.

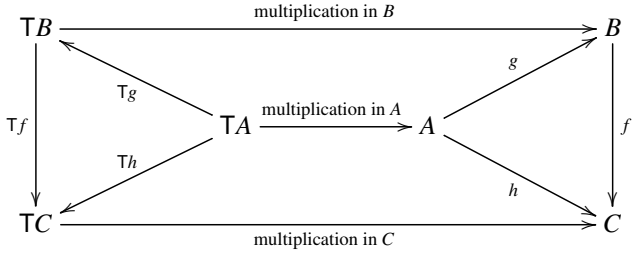
By translating maximality of $\sim$ into the language of homomorphisms, it follows that that every surjective homomorphism $g : A \to B$ that recognises L must factor through h, i.e. must be some function f such that $g = f \circ h$. The last thing to show is that f is in fact a homomorphism, and not just any function on the underlying sets of the algebras. This is shown in the following claim, with C being the quotient $A_{/\sim}$.

Claim 4.17. *Let A, B, C be algebras, let g, h be surjective homomorphisms, and let f be a function which makes the following diagram commute.*



Then f is a homomorphism.

Proof Consider the following diagram:



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. □

□

Exercises

Exercise 129. Show that the assumption on finite index of $\sim_1$ in Lemma 4.14 is needed.

Exercise 130. 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 131. This exercise can be seen as a variant of Moore's algorithm for computing the syntactic congruence. Consider a monad in the category of sets, together with a term basis, see Exercise 124. Consider an algebra colouring $\lambda : A \to U$ where A is a finite algebra. Show that the syntactic congruence of λ is the greatest (coarsest) equivalence relation on A recognises λ and which is stable under all term operations from the term basis.

4.3.3 Finitary monads

In the monad of finite words, every language – not just recognisable ones – has a syntactic homomorphism. For example, in the monad of finite words, the syntactic homomorphism of the non-recognisable language "the number of a letters is equal to the number of b letters" maps a word to the difference (number of a's − number of b's). In the monad of $\circ$-words, some non-recognisable languages do not have syntactic homomorphisms, as witnessed by Example 24. What is the difference?

The difference, as will be shown in Theorem 4.19 below, 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 4.18 (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* if

$$t = (\mathsf{T}f)(t) \qquad \text{for some } f : X \to X \text{ with finite image.}$$

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 4.19. *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*[16].

Proof For the left-to-right implication, we use the same proof as for Theorem 4.13. Define the join of a possibly infinite set of congruences to be the transitive closure of their union. By the same reasoning as in Claim 4.15, the join commutes with all term operations that have finitely many variables. Because the monad is finitary, all term operations are like this, and therefore the join commutes with all term operations, and is therefore a congruence. (We have thus shown that for finitary monads, the congruences in an algebra form a complete lattice.) If we now take the join of all congruences that recognise a given algebra colouring, then we get the syntactic congruence, and the quotient homomorphism is the syntactic homomorphism.

We now prove the converse implication. Suppose that every algebra colouring has a syntactic homomorphism. Fix some set X. We will show that all elements of $\mathsf{T}X$ are finitary. Let $\color{red}{X}$ be a disjoint copy of X. For a finite subset $Y \subseteq X$, define

$$f_Y : \color{red}{X} + X \to \color{red}{X} + X$$

to be the function which maps each element to itself, with the exception of the red copies of elements from Y, which are mapped to their corresponding black copies. Define $\sim$ to be the equivalence relation on $\mathsf{T}(X + \color{red}{X})$ which identifies two elements if, for some finite $Y \subseteq X$, they have the same image under $\mathsf{T}f_Y$.

Claim 4.20. $\sim$ *is a congruence on* $\mathsf{T}(X + \color{red}{X})$.

Proof We first argue that $\sim$ is an equivalence relation. Transitivity argued as follows:

$$w_1 \underbrace{\sim}_{\substack{\text{as witnessed}\\ \text{by } Y_1 \subseteq X}} w_2 \quad \text{and} \quad w_2 \underbrace{\sim}_{\substack{\text{as witnessed}\\ \text{by } Y_2 \subseteq X}} w_3 \qquad \text{implies} \qquad w_1 \underbrace{\sim}_{\substack{\text{as witnessed}\\ \text{by } Y_1 \cup Y_2 \subseteq X}} w_3.$$

Consider the syntactic homomorphism of $\sim$, which exists by the assumption that syntactic congruences exist. Let $\approx$ be the kernel of the syntactic homomorphism, which means that (a) $\approx$ is a congruence that is contained in $\sim$ when

[16] This theorem is unpublished work of Gordon Plotkin and the author.

viewed as a set of pairs, and (b) $\approx$ contains every congruence that is contained in $\sim$. We will show that $\sim$ is actually equal to $\approx$, and therefore $\sim$ is a congruence. In light of (a), it is enough to show that $\sim$ is contained in $\approx$. Indeed, suppose that two elements are equivalent under $\sim$. By definition, this means that they have the same image under $\mathsf{T}f_Y$ for some finite Y. Since $\mathsf{T}f_Y$ is a homomorphism that recognises $\sim$, it follows by (b) that the two elements are equivalent under $\approx$. □

Consider the functions

$$f, f : X \to X + X$$

such that f maps each argument to its red copy, and f is the identity. For every $x \in X$, its unit is mapped by $\mathsf{T}f$ and $\mathsf{T}f$ to elements which have the same image under $\mathsf{T}f_{\{x\}}$, and therefore are equivalent under $\sim$. Since $\sim$ is a congruence, and all units in $\mathsf{T}X$ are mapped by $\mathsf{T}f$ and $\mathsf{T}f$ to elements equivalent under $\sim$, it follows that for every $w \in \mathsf{T}X$, its images under $\mathsf{T}f$ and $\mathsf{T}f$ are equivalent under $\sim$. By definition of $\sim$, this means that for every $w \in \mathsf{T}X$ there must be some finite $Y \subseteq X$ such that

$$(\mathsf{T}(f_Y \circ f))(w) = (\mathsf{T}(f_Y \circ f))(w) \tag{4.8}$$

We now complete the proof that every element of $\mathsf{T}X$ is finitary. Let $w \in \mathsf{T}X$, and let Y be such that the above equivalence holds. Choose an element $y \in Y$ and consider the function

$$g : X + X \to X$$

which is the identity on X and maps all red letters to y. We have

$$\begin{aligned}
w &= && \text{(because } g \circ f \text{ is the identity on } X\text{)} \\
(\mathsf{T}(g \circ f))(w) &= && \text{(because } f_Y \text{ is the identity on black letters)} \\
(\mathsf{T}(g \circ f_Y \circ f))(w) &= && \text{(by (4.8))} \\
(\mathsf{T}(g \circ f_Y \circ f))(w). &&&
\end{aligned}$$

The image of the function $g \circ f_Y \circ f$ is contained in Y, and therefore we have established that w is finitary. □

Exercises

Exercise 132. 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 133. 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 134. 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 135. 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 4.13 for monads over this category.

Exercise 136. 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 4.13 fails.

Exercise 137. 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 138. Recall the notion of *regular elements* from Exercise 113. Show that if t is regular, then it is finitary.

Exercise 139. 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

$$\text{class of languages} \quad \sim \quad \text{class of semigroups.}$$

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 that every class of languages with good closure properties will correspond to a class of algebras with good closure properties. The theorem was originally proved by Eilenberg for monoids[17] , but with some extra care one can make the proof work in the abstract setting of monads.

4.4.1 Unary polynomials

Before stating and proving the theorem, we describe unary polynomials, which are used in the definition of language varieties. For an algebra A, define a *unary polynomial*[18] to be any function of the form

$$a \in A \quad \mapsto \quad t^A(a, c_1, \ldots, c_n) \in A,$$

which is obtained for some choice of $n \in \{0, 1, \ldots\}$, some term t with $n + 1$ variables[19] and some parameters $c_1, \ldots, c_n \in A$.

Lemma 4.21. *In every algebra, unary polynomials are closed under composition.*

Proof Consider two unary polynomials

$$\begin{aligned} a \in A \quad &\mapsto \quad t^A(a, c_1, \ldots, c_n) \\ a \in A \quad &\mapsto \quad s^A(a, d_1, \ldots, d_m). \end{aligned}$$

To prove that the composition of the above two unary polynomials is also a unary polynomial, we will show that there is a term u with $1 + m + n$ variables which satisfies the following equality:

$$\underbrace{t^A(s^A(a, d_1, \ldots, d_m), c_1, \ldots, c_n)}_{\text{a composition of two unary polynomials}} = \underbrace{u^A(a, d_1, \ldots, d_m, c_1, \ldots, c_n)}_{\text{a single unary polynomial}}. \tag{4.9}$$

[17] [25] Eilenberg, *Automata, languages, and machines. Vol. B*, 1976 , Theorem 13.2

[18] The terminology of "terms" and "polynomials" comes from universal algebra, see [47] Sankappanavar and Burris, "A course in universal algebra", 1981 , Definition 13.3. This terminology can be explained – or at least more easily remembered – as follows. Consider the ring of the reals $(\mathbb{R}, +, -, \times, 0, 1)$. A term operation in this ring can only use the constants 0 and 1 which are given in the ring as an algebra, and therefore term operations correspond to polynomials with integer coefficients. If we want to get all polynomials, we need to allow the terms to use arbitrary elements of $\mathbb{R}$ as constants.

[19] A more principled definition, which allows more variables and infinitely many constants, is discussed in Exercise 141. Since we use unary polynomials mainly for finite algebras, the more elementary definition given here is enough.

For the purposes of this proof, we treat a number such as n as a set which has n elements. Define the term u to be the result of applying the term operation

$$t^{\mathsf{T}(1+m+n)} : (\mathsf{T}(1+m+n))^{1+n} \to \mathsf{T}(1+m+n)$$

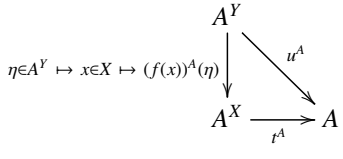
to the valuation f which maps the variable in 1 to the term s (seen as a term over a larger set of variables that does not use the last n variables), and which maps the variables in n to their corresponding units. The equality (4.9) follows from the following claim, in the case where the variables X are $1+n$, the variables Y are $1+m+n$, and the valuation $\eta \in A^Y$ is

$$(a, d_1, \ldots, d_m, c_1, \ldots, c_n) \in A^{1+m+n}.$$

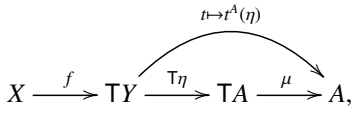
Claim 4.22. *For every $f : X \to \mathsf{T}Y$ and $t \in \mathsf{T}X$, the term*

$$u \stackrel{\text{def}}{=} t^{\mathsf{T}Y}(f)$$

makes the following diagram commute for every algebra A:



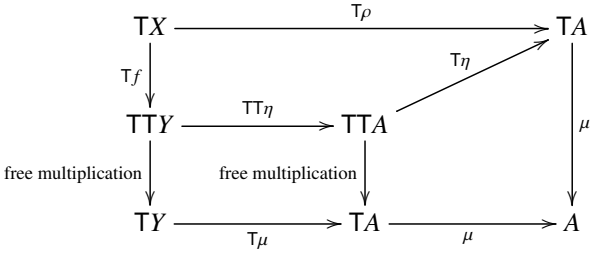
Proof This claim is the same as Exercise 128. Let $\eta \in A^Y$ be a valuation. If we apply the function in the vertical arrow from the diagram to η, then we get the valuation $\rho \in A^X$ that is the composition of the following functions:



where μ is the multiplication operation of the algebra A. The diagram in the statement of the claim says that

$$t^A(\rho) = u^A(\eta).$$

To prove this, consider the following diagram:



The upper-left face commutes by definition of ρ. The lower-left face commutes by naturality of free multiplication, and the lower-right face commutes by associativity of μ. Therefore, the entire diagram commutes. If we apply the top-most path from $\mathsf{T}X$ to A in the diagram to the term $t \in \mathsf{T}X$, then we get the result $t^{\mathbf{A}}(\rho)$, while if we apply the bottom-most path to the same term, then we get the result $u^{\mathbf{A}}(\eta)$. Since the diagram commutes, these results are equal, thus proving the claim. □

□

Another result about unary polynomials that will be used in the proof of the Eilenberg Variety Theorem is the following characterisation of congruences in finite algebras. The finiteness assumption is important, see Exercise 140.

Lemma 4.23. *An equivalence relation $\sim$ in a finite algebra A is a congruence if and only if it commutes with all unary polynomials, in the sense that*

$$a \sim b \quad \textit{implies} \quad f(a) \sim f(b)$$

holds for every $a, b \in A$ and every unary polynomial $f : A \to A$.

Proof The left-to-right implication is immediate, and does not need the assumption on finiteness of the algebra. If $\sim$ is a congruence, then it commutes with all term operations, and so it must also commute with unary polynomials, which are term operations with some arguments fixed.

The right-to-left implication is proved similarly to Lemma 4.14 about joins of congruences. Suppose that $\sim$ commutes with all unary polynomials. By the same argument as in Claim 4.15, where arguments are replaced one by one in finitely many steps, it follows that $\sim$ commutes with all term operations that have finitely many variables. Since the algebra A is finite, all valuations for term operations have finite image, and therefore we can use Claim 4.16 to prove that $\sim$ commutes with all term operations, and therefore it is a congruence. □

Exercises

Exercise 140. Show that the finiteness assumption in Lemma 4.23 is needed. Hint: use Example 24.

Exercise 141. Define a *vectorial polynomial* in an algebra A to be any operation

$$f : A^X \to A^Y,$$

for some sets X and Y, which arises as follows: (a) transform an input valuation A^X to a larger valuation A^{X+Z} by mapping the variables from Z to some fixed constants; and then (b) apply a vectorial term operation $A^{X+Z} \to A^Y$, as described in Exercise 128. Show that vectorial polynomials are closed under composition.

4.4.2 Varieties

The classes with good closure properties will be called varieties, in analogy with the varieties that appear in Birkhoff's theorem from universal algebra. In this section, we define varieties, and give several examples of them. There will be two kinds of varieties: for algebras and for languages. We begin with the algebras. In the following definition, a *quotient* of an algebra is any image of that algebra under a surjective homomorphism. In other words, a quotient is a quotient under some congruence.

Definition 4.24 (Algebra variety)**.** Fix a T 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 26. 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 27. 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 28. 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 X. 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

$$xy = yx$$

in the monad of nonempty finite words. Some algebra varieties do not arise this way. For example, the varieties discussed in Example 26 do not arise from (even possibly infinite sets of) identities. Identities will be discussed in more detail in Section 4.5. □

We now describe language varieties. In Eilenberg's original formulation, 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 polynomials, as described in the following definition.

In the following definition, by recognisable languages we mean recognisable subsets of free algebras.

Definition 4.25 (Language variety)**.** Let T be a monad in the category of sets. A *language variety* is a class $\mathscr{L}$ 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 in a free algebra $\mathsf{T}\Sigma$, then $\mathscr{L}$ is closed under inverse images of f.

Example 29. Consider the monad of finite words, where algebras are monoids. We will that 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, then a strategy copying argument shows that

$$\begin{array}{ccc} \text{Duplicator wins the} \\ k \text{ round game on } w \text{ and } w' \end{array} \quad \text{implies} \quad \begin{array}{c} \text{Duplicator wins the} \\ k \text{ round game on } f(w) \text{ and } f(w'). \end{array}$$

This implies that first-order definable languages are closed under inverse images of homomorphisms and unary polynomials. The same is true for first-order logic on $\circ$-words. $\square$

Example 30. Consider again the monad of finite words, where algebras are monoids. The definite languages from Example 19 are not a variety, because the class of definite languages is not closed under inverse images of the homomorphisms. Indeed, the language

$$\underbrace{a\{a,b\}^* \subseteq \{a,b\}^*}_{\text{words that being with } a}$$

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

$$\underbrace{c^* a\{a,b,c\}^* \subseteq \{a,b,c\}^*}_{\text{words that begin with } a \text{ if } c \text{ is erased}},$$

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. $\square$

Exercises

Exercise 142. 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 polynomials of the form:

$$w \mapsto v_1 w v_2 \qquad \text{for every choice of parameters } v_1, v_2 \in \Sigma^*.$$

Exercise 143. Consider the monad of finite words. Show that there are uncountably many algebra varieties. In particular, for some algebra varieties, the membership problem $A \stackrel{?}{\in} \mathscr{A}$ is undecidable.

Exercise 144. Consider the monad of $\circ$-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 polynomials of the following forms:

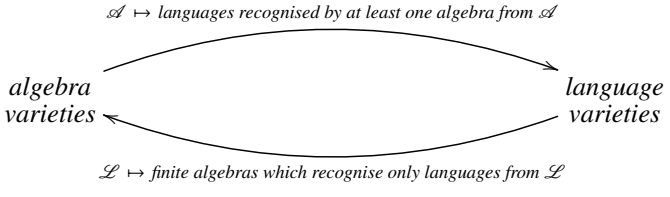
$$\begin{aligned} w &\mapsto v_1 w v_2 && \text{for every choice of parameters } v_1, v_2 \in \Sigma^\circ \\ w &\mapsto w^\omega \\ w &\mapsto w^{\omega *} \\ w &\mapsto \text{shuffle of } \{w, v_1, \ldots, v_n\} && \text{for every choice of parameters } v_1, \ldots, v_n \in \Sigma^\circ. \end{aligned}$$

Exercise 145. Consider the monad T from Example 21, where $\mathsf{T}X$ describes terms over a fixed ranked set Σ with variables X. We view term $t \in \mathsf{T}\Gamma$ as a model, where the elements are the nodes of the corresponding tree, there is a binary ancestor relation $x \leq y$, and for every $\sigma \in \Sigma + \Gamma$ there is a unary relation $\sigma(x)$ which selects nodes with label σ. Show that the class of languages definable in first-order logic is not a variety. Hint: read the exercises in Chapter 5.

4.4.3 Algebra varieties are the same as language varieties

In this section we prove that the two notions of variety are equivalent.

Theorem 4.26 (Eilenberg Variety Theorem)**.** *Let* T *be a monad in the category of sets. Then the maps in the following diagram are mutually inverse bijections.*



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 the input to L satisfies a weaker assumption than being

an algebra variety, namely it is closed under products, then the output $\mathsf{L}\mathscr{A}$ 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 $L \cup K$ 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 a language that is recognised by a homomorphism

$$h : \mathsf{T}\Sigma \to A \in \mathscr{A}.$$

We need to show that $\mathsf{L}\mathscr{A}$ contains all inverse images of L under homomorphisms and unary polynomials. Consider first the homomorphisms: let $g : \mathsf{T}\Gamma \to \mathsf{T}\Sigma$ be a homomorphism, and consider the inverse image of L under g, which can be written as $L \circ g$ if we view L as a function with outputs "yes" and "no". This inverse image is recognised by the homomorphism $h \circ g$, which uses the algebra A, and therefore it belongs to $\mathsf{L}\mathscr{A}$. The same kind argument applies to unary polynomials. Consider a unary polynomial $f : \mathsf{T}\Sigma \to \mathsf{T}\Sigma$. As we have remarked in the proof of Lemma 4.23, congruences commute with unary polynomials, which means that $h \circ f = f \circ h$, and therefore h also recognises the inverse image $L \circ f$.

(2) Similarly to the first step, we show that if the input to A satisfies a weaker condition than being a language variety, namely it is closed under under unions and intersections, then the output is an algebra variety[20]. 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 homomorphism

$$h_A : \mathsf{T}\Sigma \to A,$$

[20] 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 of both kinds are the closed sets, with respect to this Galois connection.

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

$$\mathscr{A} = \mathsf{AL}\mathscr{A},$$

with the dual equality being proved in the next step. The above equality 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 $\mathscr{A}$.

Clearly every algebra $A \in \mathscr{A}$ satisfies (*). We now prove the converse implication. Suppose that an algebra A satisfies (*). The multiplication operation

$$\mu : \mathsf{T}A \to A$$

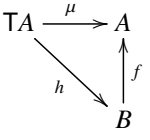
in the algebra A is a homomorphism from the free algebra $\mathsf{T}A$ to A. By the assumption that A satisfies (*), every language recognised by this homomorphism is recognised by some algebra from $\mathscr{A}$. In particular, for every $a \in A$ the language $\mu^{-1}(a)$ is recognised by some homomorphism

$$h_a : \mathsf{T}A \to B_a \in \mathscr{A}.$$

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. The algebra B is a sub-algebra of 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:



By Lemma 4.17, 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 $L \in \mathscr{L}$, and its syntactic homomorphism, which exists by Theorem 4.13, is

$$h : \mathsf{T}\Sigma \to A.$$

To prove (*), we will show that all languages recognised by the syntactic algebra A belong to $\mathscr{L}$.

Claim 4.27. *Let $F : A \to \{$"yes", "no"$\}$ be the accepting set in the syntactic algebra, which means that L is equal to $F \circ h$. Then two elements of A are equal if and only if they have the same values under $F \circ f$ for every unary polynomial $f : A \to A$.*

Proof Let $\sim$ be the equivalence relation on A which identifies two elements that have the same image under $F \circ f$ for every unary polynomial f. Because unary polynomials are closed under composition, it follows that $\sim$ commutes with all unary polynomials, and therefore it is a congruence by Lemma 4.23. Because the identity is a special case of a unary polynomial, elements that are equivalent under $\sim$ have the same value under F. This means that the quotient homomorphism of $\sim$ recognises F, and therefore $\sim$ must be the identity since otherwise A would not be the syntactic algebra of L. □

The following claim shows that unary polynomials in A can be pulled back, along the homomorphism h, to unary polynomials in $\mathsf{T}\Sigma$.

Claim 4.28. *For every unary polynomial $f : A \to A$ there is a unary polynomial $f^h : \mathsf{T}\Sigma \to \mathsf{T}\Sigma$ which makes the following diagram commute:*

$$\begin{array}{ccc} \mathsf{T}\Sigma & \xrightarrow{f^h} & \mathsf{T}\Sigma \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h} \\ A & \xrightarrow[f]{} & A \end{array}$$

Proof Consider a unary polynomial $f : A \to A$ of the form

$$a \in A \quad \mapsto \quad t^A(a, c_1, \ldots, c_n).$$

Because the syntactic homomorphism is surjective, for each $i \in \{1, \ldots, n\}$ there must be some $s_i \in \mathsf{T}\Sigma$ which is mapped to c_i by h. Since term operations commute with homomorphisms by Lemma 4.10, the diagram in the claim commutes if we choose f^h to be

$$s \in \mathsf{T}\Sigma \quad \mapsto \quad t^{\mathsf{T}\Sigma}(s, s_1, \ldots, s_n).$$

□

We are now ready to show that $\mathscr{L}$ contains all languages recognised by the syntactic algebra A.

We first show that $\mathscr{L}$ contains all languages recognised by the syntactic homomorphism $h : \mathsf{T}\Sigma \to A$, and then we generalise this result to other homomorphisms into A. By Claim 4.27 and finiteness of the algebra A, there is a finite set

$$\mathcal{X} \subseteq A \to A$$

of unary polynomials in the algebra A such that two elements are equal if and only if they have the same values for all functions from the set

$$\{F \circ f : f \in \mathcal{X}\}.$$

Putting this together with Claim 4.28, it follows that two elements of $\mathsf{T}\Sigma$ have the same image under h if and only if they belong to the same sets from the finite family

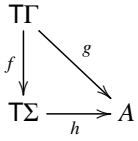
$$\{\underbrace{L \circ f_h}_{\substack{\text{a language that belongs to } \mathscr{L} \\ \text{as the inverse image of } L \\ \text{under the unary polynomial } f_h}} : f \in F\}.$$

In other words, every inverse image $h^{-1}(a)$ is a finite Boolean combination of languages from the above family, and therefore it belongs to $\mathscr{L}$ by closure under Boolean combinations. This in turn means that all languages recognised by h are in $\mathscr{L}$.

We now prove that not only does $\mathscr{L}$ contain every language recognised by the syntactic homomorphism h, as we have already shown, but it also contains every language recognised by a homomorphism

$$g : \mathsf{T}\Gamma \to A$$

which uses the same target algebra of the syntactic 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



By the above diagram, every language recognised by g is an inverse image, under f, of some language recognised by h. Since we have already proved that every language recognised by h is in $\mathscr{L}$, and $\mathscr{L}$ is closed under inverse images of homomorphisms such as f, we see that every language recognised by g is in $\mathscr{L}$.

□

Exercises

Exercise 146. 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 Variety Theorem for monads over this category.

Exercise 147. 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 Variety Theorem holds for varieties understood in this way.

Exercise 148. 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.

Exercise 149. For an algebra colouring $L : A \to U$, define *contextual equivalence* to be the equivalence relation on A which identifies two elements of A if they have the same image under $L \circ f$ for every unary polynomial $f : A \to A$. Show that if L is recognisable, then contextual equivalence is the syntactic congruence of L.

Exercise 150. Show that contextual equivalence, as defined in the previous exercise, need not be a congruence for algebra colourings that are not recognisable.

4.5 Identities and Birkhoff's Theorem

In this section we return to the identities that were described in Example 28. Recall that an *identity* is a pair of terms over a common set of variables. We say that an algebra A satisfies an identity consisting of terms $s, t \in \mathsf{T}X$ if

$$h(s) = h(t) \qquad \text{for every homomorphism } h : \mathsf{T}X \to A.$$

This is equivalent to the definition given in Example 28, which said that an algebra A satisfies the identity if the two term operations s^A and t^A are equal.

Below we present two theorems about classes of algebras that can be defined using identities. There will be two theorems, one for not necessarily finite algebras, and one for finite algebras.

Identities for varieties not necessarily finite algebras

We begin with the first theorem about identities, which is a monad variant of Birkhoff's Theorem from universal algebra[21] . The theorem says that a class of not necessarily finite algebras can be described by identities if and only if it is a *Birkhoff variety*, which means that it is closed under images of surjective homomorphisms, subalgebras and (not necessarily finite) products. Traditionally, Birkhoff varieties are called simply algebra varieties, but the latter name has already been used in this book for classes of finite algebras that are described in Definition 4.24. To avoid confusion, for the purposes of this section where the two kinds of algebra varieties are used, we use the name *Eilenberg variety* for the varieties of finite algebras.

We say that a class of not necessarily finite algebras is *defined by a set of identities* $\mathscr{E}$ if the algebras in the class are exactly those that satisfy all identities from $\mathscr{E}$.

Theorem 4.29 (Birkhoff)**.** *Let* T *be a monad in the category of sets. A class* $\mathscr{A}$ *of not necessarily finite algebras is a Birkhoff variety if and only if it can be defined by some set of identities.*

We will prove a slightly stronger result, which establishes a duality between algebras and identities. Under this duality, Birkhoff varieties will correspond to sets of identities that are closed under consequences, as described below.

A *consequence* of a set of identities $\mathscr{E}$ is defined to be any identity that is satisfied in every algebra that satisfies all identities from $\mathscr{E}$. We say that a set of identities is *closed* if it contains all of its consequences. In symbols, a set of identities $\mathscr{E}$ is closed if it satisfies the following:

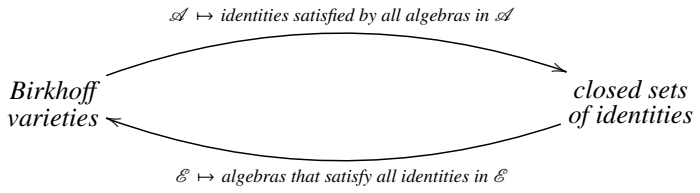
$$\underbrace{\forall e}_{\substack{\text{for every}\\ \text{identity}}} \Big(\underbrace{\forall A}_{\substack{\text{for every}\\ \text{algebra}}} \underbrace{A \models \mathscr{E} \Rightarrow A \models e}_{\substack{\text{if } A \text{ satisfies all}\\ \text{identities in } \mathscr{E},\\ \text{then it satisfies } e}} \Big) \Rightarrow e \in \mathscr{E}.$$

By taking the right and then left arrow in the following theorem, we immediately get the Birkhoff's Theorem.

Theorem 4.30. *Let* T *be a monad in the category of sets. Then the maps in the*

[21] [4] Birkhoff, "On the structure of abstract algebras", 1935 , Theorem 10.

following diagram are mutually inverse bijections.



Proof Let us write E for the left-to-right map in the diagram from the theorem, and A for the right-to-left map. Almost by definition, applying the map E to any set of algebras will produce a set of identities that is closed. It is also not hard to see that the map A produces Birkhoff varieties, because algebras satisfying a given identity are closed under surjective homomorphic images, subalgebras and possibly infinite products. It remains to show that the maps are mutually inverse, which corresponds to the following two equalities:

$$\underbrace{\mathscr{E} = \mathsf{EA}\mathscr{E}}_{\text{for every closed set of identities } \mathscr{E}} \qquad \text{and} \qquad \underbrace{\mathscr{A} = \mathsf{AE}\mathscr{A}}_{\text{for every Birkhoff variety } \mathscr{E}}.$$

The first equality says that an identity belongs to $\mathscr{E}$ if and only if it is satisfied by all algebras that satisfy all identities in $\mathscr{E}$. This equality is simply the definition of a closed set of identities, and the equalities holds for closed sets. We are left with the second equality, which says that an algebra belongs to $\mathscr{A}$ if and only if (*) it satisfies all identities that are true in all algebras from $\mathscr{A}$. Clearly every algebra from $\mathscr{A}$ satisfies (*). The converse implication follows from the following lemma, and the closure properties of Birkhoff varieties.

Lemma 4.31. *Let $\mathscr{A}$ set of algebras. If an algebra B satisfies all identities that are true in all algebras from $\mathscr{A}$, then B is a homomorphic image of a subalgebra of a (possibly infinite) product of algebras from $\mathscr{A}$.*

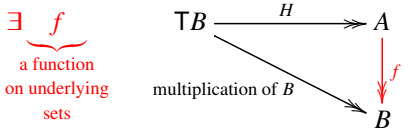
Proof The key observation is that satisfying an identity can be interpreted in terms of homomorphisms, in the following way: an identity consisting of two terms $\mathsf{T}B$ is true in all algebras from $\mathscr{A}$ if and only if the two terms have the same image under every homomorphism

$$h : \mathsf{T}B \to A \in \mathscr{A}.$$

Take the product of all possible homomorphisms h as above, and restrict the resulting homomorphism to its image, yielding a surjective homomorphism

$$H : \mathsf{T}B \to A \in \underbrace{\text{subalgebras}}_{\text{because we restricted to the image}} \text{of products of algebras in } \mathscr{A},$$

such that two terms in $\mathsf{T}B$ are an identity true in all algebras from $\mathscr{A}$ if and only if they have the same image under H. If two terms $s, t \in \mathsf{T}B$ have the same image under H, then they form an identity that is true in all algebras from $\mathscr{A}$, and therefore also an identity that is true in the algebra B, by assumption on B. If an identity is true in B, then the two terms in the identity must have the same result under the multiplication operation of the algebra B, since the latter is an example of a homomorphism of type $\mathsf{T}B \to B$. Summing up, we have shown that if two terms in $\mathsf{T}B$ have the same result under H, then they have the same multiplication. This means that the multiplication operation of the algebra B factors through the surjective homomorphism H.



By Claim 4.17, the function f is actually a homomorphism. Therefore, B is is the image of A under some surjective homomorphism. □

□

Identities for varieties of finite algebras

We now turn to identities that characterise varieties of finite algebras, the same varieties that were use in the Eilenberg Variety Theorem. To avoid confusion with the Birkhoff varieties of possibly infinite algebras that are also discussed in this chapter, we use the name *Eilenberg varieties* for varieties of finite algebras.

Theorem 4.32 (Eilenberg-Shützenberger)**.** [22] *Let* T *be a monad in the category of sets, such that there are countably many finite algebras up to isomorphism. The following conditions are equivalent for every class* $\mathscr{A}$ *of finite algebras:*

(1) *$\mathscr{A}$ is an Eilenberg variety, i.e. it is closed under images of surjective homomorphisms, subalgebras and finite products;*

[22] This theorem is based on
[26] Eilenberg and Schützenberger, *On pseudovarieties*, 1975 , Theorem 1
The theorem cited above differs in two ways from our Theorem 4.32: (a) our theorem works for any monad subject to the assumption on countably many finite algebras; (b) the characterisation in terms of identities from [26] is different, because it gives a sequence of identities (and not sets of identities), and it requires satisfying all but finitely many identities from the sequence. Eliminating difference (b) seems to require some extra assumptions on the monad.

(2) *there is a sequence of sets of identities*

$$\underbrace{\mathscr{E}_1 \supseteq \mathscr{E}_2 \supseteq \cdots}_{\text{each } \mathscr{E}_n \text{ is a set of identities}}$$

such that a finite algebra belongs to $\mathscr{A}$ if and only if for some $n \in \{1, 2, \ldots\}$ it satisfies all identities in $\mathscr{E}_n$.

Proof We begin with the implication (2)⇒(1). If an algebra satisfies all identities in $\mathscr{E}_n$, then the same is true for all of its subalgebras and images under surjective homomorphic images. For binary products the argument is the same: if an algebra satisfies all identities in $\mathscr{E}_n$ and another algebra satisfies all identities in $\mathscr{E}_m$, then their product satisfies all identities in $\mathscr{E}_{\max(m,n)}$. This establishes that every class of algebras satisfying condition (2) has the closure properties required of an Eilenberg variety.

Consider now the converse implication (1)⇒(2). Define $\mathscr{A}_n$ to the first n algebras from $\mathscr{A}$, with respect to the countable enumeration from the assumption in the theorem. Define $\mathscr{E}_n$ to be the identities that are satisfied by all algebras in $\mathscr{A}_n$. We claim that an algebra B belongs to $\mathscr{A}$ if and only if for some $n \in \{1, 2, \ldots\}$ it satisfies all identities in $\mathscr{E}_n$. The left-to-right implication is immediate. For the right-to-left implication, we can apply Lemma 4.31 to conclude that B is the homomorphic image of a subalgebra of a product of algebras from $\mathscr{A}_n$. Furthermore, if we inspect the proof of Lemma 4.31, we will see that the product is finite, because the homomorphisms from $\mathsf{T}B$ to algebras in $\mathscr{A}_n$, as used in (**??**), can be chosen in finitely many ways. □

Exercises

Exercise 151. Show that Theorems 4.29 and 4.32 are also true for monads in categories of sorted sets, even with infinitely many sorts.

Exercise 152. Give an example of a monad which violates the assumption on countably many finite algebras from Theorem 4.32. Hint: see Section 5.2.3.

PART THREE

TREES AND GRAPHS

5

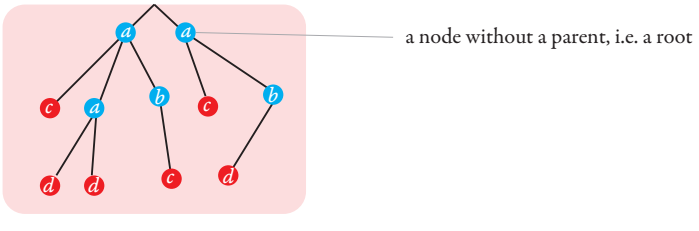
Forest algebra

In this chapter, we present a monad that models trees[1]. The trees are 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 slightly more general objects, namely forests (multisets of trees) and contexts (which are forests with a port that is meant to be replaced by a forest or context). 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 partial *parent* from nodes to nodes, and a *labelling* from nodes to Σ. Here is a picture:



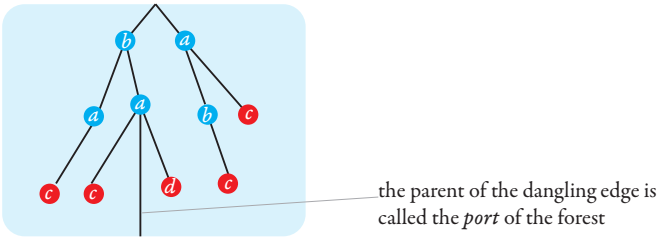
[1] This section is based on
[13] Bojańczyk and Walukiewicz, "Forest Algebras", 2008

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. Note that there is no order on siblings in our definition of forests. The definition

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 (nodes with the same parent). We assume that all roots are siblings.

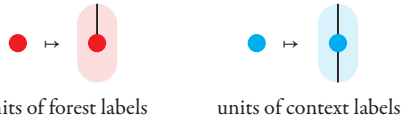
Apart from forests, the forest monad will also talk about *contexts*, which are forests with an extra dangling edge that is attached to a node with a context label, as in the following picture:



The forest monad. We now define a monad structure on forests and contexts.

Definition 5.1. Define the *forest* monad as follows.

- The underlying category is two-sorted sets, where objects are two-sorted sets (with sorts "forest" and "context") and the morphisms are sort-preserving functions between two-sorted sets.
- 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 or context that has one node with label a, as in the following pictures:



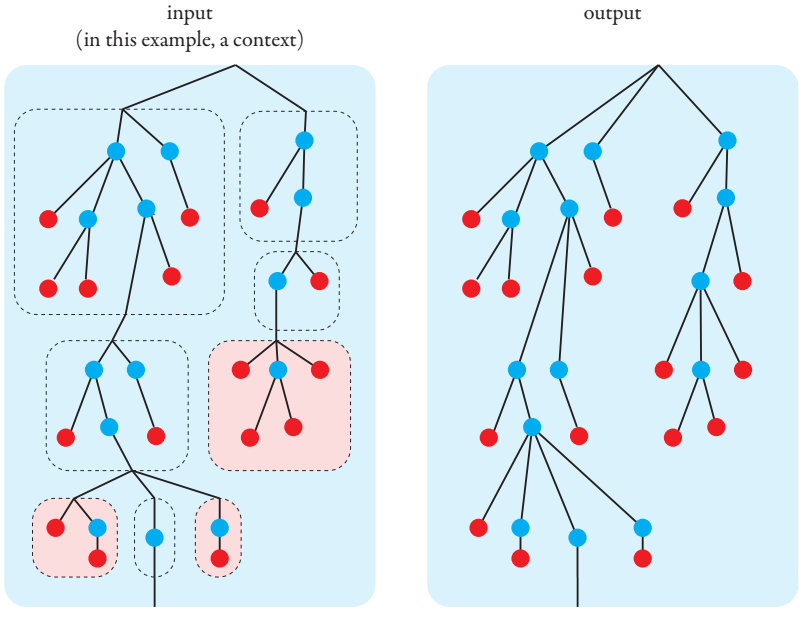


Figure 5.1 Free multiplication in the forest monad

- 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 in the tree or context which is the label of u. The label is inherited from v, while the parent of a node (u, v) 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 algebras* 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.

The notion of compositional function and Lemma 4.7 about compositional functions corresponding to homomorphisms is also true for monads in the category of sorted sets used by forest algebra. Therefore, we will mainly describe homomorphisms using the terminology of compositional functions.

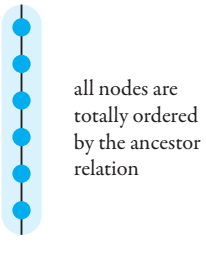
Example 31. Let $\Gamma \subseteq \Sigma$ be two-sorted alphabets. We can view

$$\mathsf{F}\Gamma \subseteq \mathsf{F}\Sigma$$

as a language, which only contains those forests and context over alphabet Σ where all labels are from Γ. Here is a homomorphisms into a finite algebra that recognises this homomorphisms Consider the function which inputs a forest or context in $\mathsf{F}\Sigma$, and outputs the following information: (a) is it a forest or context; (b) are all labels from Γ? This function is easily seen to compositional, and therefore h it can be viewed as a homomorphism of forest algebras. The co-domain of the homomorphism h is a forest algebra with two elements on the forest sort, and two elements on the context sort. □

Example 32. Let Σ be a two-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, and therefore h can be viewed as a homomorphism of forest algebras. This homomorphism recognises the language of forests or contexts where the number of nodes is divisible by n. □

Example 33. 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:



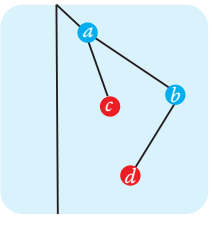
In other words, $\mathsf{F}\Sigma$ is empty on the forest sort, and is isomorphic to the free semigroup Σ^+ on the context sort. Since the monad structure of the free semigroup agrees with the monad structure of the forest monad, it follows that a forest algebra with an empty forest sort is the same thing as a semigroup. □

Exercises

Exercise 153. Show that recognisable languages in the forest monad are closed under images of (not necessarily letter-to-letter) homomorphisms

$$h : \mathsf{F}\Sigma \to \mathsf{F}\Gamma.$$

Exercise 154. Consider a variant of the forest monad, where we allow contexts where the port is a root, like in the following example:



Show that in this variant, recognisable languages are not closed under images of homomorphisms, but are closed under images of letter-to-letter homomorphisms.

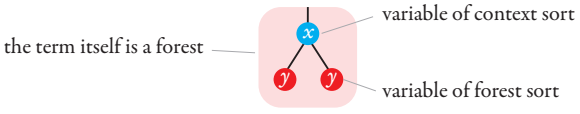
5.2.1 A finite representation

As usual with the monad approach, one needs to explain how 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 multiplication 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.

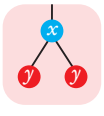
A term basis. Like for any monad, a *term* in the forest monad is defined to be an element of $\mathsf{F}X$ for some two-sorted set of variables X. Here is a picture of a term:



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 a term operation t^A defined by

$$\eta \in A^X \quad \mapsto \quad \text{multiplication in } A \text{ applied to } (\mathsf{F}\eta)(t).$$

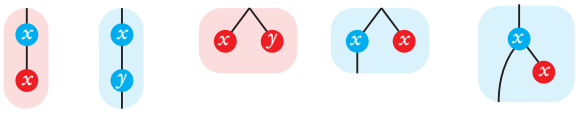
The input to the term operation is a sort-preserving valuation of the variables, while the output is an element of the algebra whose sort is the output sort of the term. For example, the term



induces a term 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:



(These happen to be all terms with exactly two nodes, modulo renaming variables.) 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, listed in the order from the picture above (the

$$
\begin{array}{rcll}
x{+}(y{+}z) & = & (x{+}y){+}z & \text{(F1) forests with } + \text{ are a semigroup} \\
x{+}y & = & y{+}x & \text{(F2) the forest semigroup is commutative} \\
x{\cdot}(yz) & = & (xy){\cdot}z & \text{(F3) contexts with } \cdot \text{ are a semigroup} \\
x{\cdot}(y{\cdot}z) & = & (xy){\cdot}z & \text{(F4) } \cdot \text{ is an action of contexts on forests} \\
x{+}(y{+}z) & = & (x{+}y){+}z & \text{(F5) } + \text{ is an action of forests on contexts} \\
x{\oplus}(y{+}z) & = & (x{\oplus}y){\oplus}z & \text{(F6) } \oplus \text{ is an action of forests on contexts} \\
(x{\cdot}y){+}z & = & (x{+}z){\cdot}y & \text{(F7) compatibility of the actions} \\
(x{\oplus}y){\cdot}z & = & x{\cdot}(y{+}z) & \text{(F8) compatibility of the actions} \\
(x{\oplus}y){\cdot}z & = & x{\cdot}(z{+}y) & \text{(F9) compatibility of the actions}
\end{array}
$$

Figure 5.2 Axioms of forest algebra. 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.

colour of an operator is the colour of the output sort):

$$
\begin{array}{ccccc}
\overbrace{x{\cdot}x}^{\substack{\text{inputs a context } x \\ \text{and forest } x \\ \text{and outputs} \\ \text{a forest}}} & \overbrace{x{\cdot}y}^{\substack{\text{inputs a context } x \\ \text{and context } y \\ \text{and outputs} \\ \text{a context}}} & \overbrace{x{+}y}^{\substack{\text{inputs a forest } x \\ \text{and forest } y \\ \text{and outputs} \\ \text{a forest}}} & \overbrace{x{+}x}^{\substack{\text{inputs a forest } x \\ \text{and context } x \\ \text{and outputs} \\ \text{a context}}} & \overbrace{x{\oplus}x}^{\substack{\text{inputs a forest } x \\ \text{and context } x \\ \text{and outputs} \\ \text{a context}}}
\end{array}
$$

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 it is finite, then it can be finitely represented by giving the multiplication tables for the basic operations. Using this representation, we can talk about algorithms that process finite forest algebras.

We can also give simple list of axioms forest algebra, see Figure 5.2. These axioms are sound (they are satisfied by the basic operations in every forest algebra) and complete (if one gives five operations on a two sorted set A that satisfy the axioms, then these operations can be extended to a forest algebra multiplication $\mu : \mathsf{F}A \to A$). Using this axiomatisation, we can effectively check if a finite representation of a forest algebra is correct, i.e. it comes from some forest algebra.

Exercises

Exercise 155. Show that for every $t \in \mathsf{F}\Sigma$ there is a decomposition

$$t = f(t_1, \ldots, t_n)$$

such that f is a term of size at most 4 (and therefore the number of arguments n is at most 4), and all arguments $t_1, \ldots, t_n$ have at most half the size (number of nodes) of t.

Exercise 156. Fix some language $L \subseteq \mathsf{F}\Sigma$ that is recognised by a finite forest algebra. Suppose that we begin with some forest $t \in \mathsf{F}\Sigma$ and then we receive a stream of updates and queries. Each update changes a label of some node (the set of nodes and the parent function are not changed by updates). Each query asks if the current forest belongs to L. Show that one can compute in linear time a data structure (at the beginning, when the first forest t is given), such that updates can be processed in logarithmic time and queries can be processed in constant time.

Exercise 157. Prove completeness for the axioms (F1)–(F6).

5.2.2 Syntactic algebras

We now discuss syntactic algebras must necessarily exist in the forest monad. This is shown by a minor adaptation of the results from Section 4.3. As mentioned in Section 4.3, syntactic homomorphisms also make sense in other categories, such as the category of two-sorted sets used by the forest monad.

In the forest monad, 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 5.5. *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 4.19, while for item (2) we use the same proof as in Theorem 4.13. The only difference is that the variables in term operations have sorts. In item (2), the assumption on finitely many sort names is used[2] in the extension of Lemma 4.14 to say that there are finitely many equivalence classes of altogether of a congruence that has finitely many equivalence classes on 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 5.5, 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.

Exercises

Exercise 158. 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

[2] This assumption is indeed necessary, which can be proved using ideas from [10] Bojańczyk and Klin, "A non-regular language of infinite trees that is recognizable by a sort-wise finite algebra", 2019

algebra is represented using its basic operations, as in Theorem 5.3, and the homomorphism is represented by its values on the units.

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 countably 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 34. Consider the two-sorted alphabet $\Sigma = \{a, b\}$. Even though there are no forest-sorted letters, it is still possible to construct an infinite forest over this alphabet, because there is no need for leaves. We say that a language of ω-words is *prefix-independent* if it is stable under removing or adding a single letter as a prefix. Consider some prefix-independent language L of ω-words over the alphabet Σ, not necessarily regular. For example

$$L = \{b^{n_1}ab^{n_2}a \cdots : \text{the sequence } n_1, n_2, \ldots \text{ contains infinitely many primes}\}.$$

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 $\mathsf{F}_\infty\Sigma$ 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 L'?
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 L'?

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?

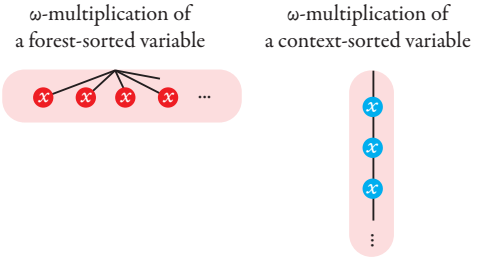
The red questions have at most 4 possible answers, and the blue questions have at most 32 possible answers, hence the number 36. In fact, this number can easily be reduced; for example in case of a "no" answer to question 1, there is no need to store the answers for the remaining questions. We leave 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 159. Consider the monad F_∞. We say that a forest or context in this monad is *thin* if it has countably many branches. Define $\mathsf{F}_{\text{thin}}\Sigma \subseteq \mathsf{F}_\infty\Sigma$ to be thin forests or contexts. Show that this is a monad.

Exercise 160. Show that a countable forest or context is thin, in the sense of Exercise 159, 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 161. Consider the monad F_{thin} from Exercise 159. 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:



5.3 Logics for forest algebra

In Chapter 2, we presented many examples of logics on finite words that could be characterised using structural properties of recognising monoids. In this section, we present some results of this type for forest algebra. Unfortunately, there are fewer interesting examples in the case of forest algebra, since the algebraic theory of forest languages is still not properly understood. A notable gap in our logic is first-order logic on trees, which is not known to have an algebraic characterisation, as will be discussed in Section 5.3.2.

5.3.1 Monadic second-order logic

We begin with monadic second-order logic. The idea is as usual: to each forest or context we associate a model, and then we use monadic second-order logic to describe properties of that model. There is one twist: because siblings in a forest or 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 5.7. Define the *ordered model* of a forest or context as follows: the universe is the nodes, and it is equipped with the following relations:

$$\underbrace{x \leq y}_{\text{ancestor}} \qquad \underbrace{a(x)}_{x \text{ has label } a \in \Sigma} \qquad \underbrace{\mathit{port}(x)}_{x \text{ is the port}}$$

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 5.8. *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.

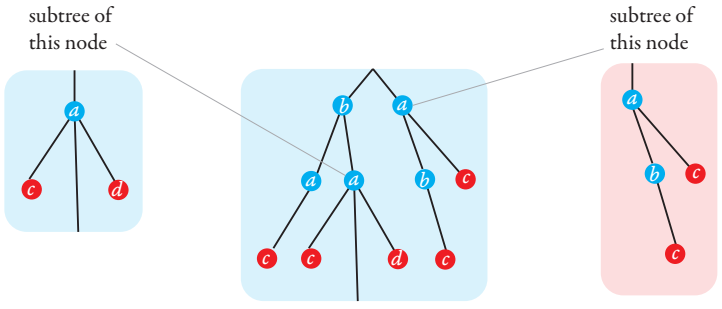
From counting MSO to a finite forest algebra. Same proof as for finite words: we remove the first-order variables (by coding them as singleton sets), and then we show by induction that for every formula of MSO (possibly with free variables), its corresponding language is recognised by a finite forest algebra. In the induction steps we use products and powersets, both of which are finiteness preserving constructions for forest algebras.

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. The idea is the same as for finite words: the defining formula inductively computes the value under h for every subtree in the input forest or context. The induction corresponds to a bottom-up pass through the input[3]. Define the *subtree* of a node in a tree or context as explained in the following picture:

[3] 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 ω-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.



In a forest, all subtrees are trees, while in a context some subtrees are trees and others are contexts. 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 5.9. *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 $n \in \{0, \ldots, |A|\}$.

Proof Let $a_1, \ldots, a_m$ 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

$$a \cdot (a_1 + \cdots + a_m).$$

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. Some of these elments have forest sort and some have context sort. 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 claim. Finally, the image $h(t)$ of a forest or 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). Since the image $h(t)$ determines membership of t in the language, it follows that the language itself is definable in counting MSO. □

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 Section 5.2.1, and the homomorphism is represented by its images for the units.

The exact role of counting is explained in the following theorem.

Theorem 5.10. *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 5.8. The only difference is that in Claim 5.9 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 5.8, 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 are aperiodic. Products clearly preserve aperiodicity of the forest semigroup, and the same is true powersets, as explained in the following lemma.

Lemma 5.11. *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 multiplication, proving $A^{n+1} \subseteq A^n$. □

□

Corollary 5.12. *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 5.10. □

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, in Exercise 27 we showed that every finite semigroup, not necessarily aperiodic, can be obtained by applying products and powersets to an aperiodic semigroup.

5.3.2 First-order logic

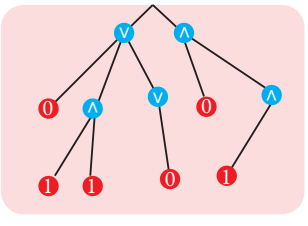
For finite words, the king of algebraic characterisations was the Shützenberger-McNaughton-Papert-Kamp Theorem, which described the languages of finite words that can be defined in first-order logic (using the ordered model). Unfortunately, finding a generalisation of this theorem to forest algebra (or any other algebra modelling trees) remains an open problem[5]. 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 162. 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 35. 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

$$L = \{t \in \mathsf{F}\Sigma : t \text{ is a forest where all roots have value 1}\}$$

[5] This is in contrast to first-order logic on trees with the child relation (and not the descendant ordering), which has an algebraic characterisation, see [3] Benedikt and Segoufin, "Regular tree languages definable in FO and in FO_{mod}", 2009 , Theorem 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 164. □

Exercises

Exercise 162. Prove that first-order logic, as discussed in Section 5.3.2, is a variety in the sense of the Eilenberg variety theorem.

Exercise 163. Show that the language of forests where some leaf has even depth is not definable in first-order logic.

Exercise 164. Show that the language from Example 35 is not definable in first-order logic.

Exercise 165. Consider a two-sorted alphabet

$$\Sigma = \{\text{left, right}, \text{left, right}\}.$$

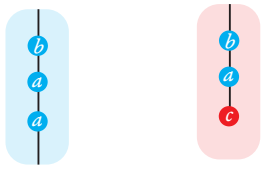
A *binary tree* over this alphabet is a tree where every node is either a leaf, or it has exactly two children, with labels "left" and "right" in the appropriate sort. There are no constraints on the root label. Define $L \subseteq \mathsf{F}\Sigma$ to be the set of binary trees where all leaves are 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 166. 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 165.

Exercise 167. A unary node in a forest or context is defined to be 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.

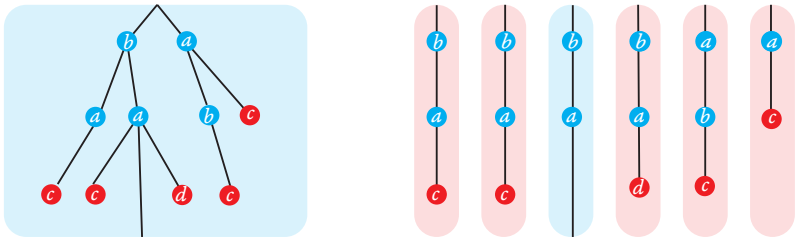
5.3.3 Branch languages

In this section, we discuss languages which are defined only by looking at branches in a forest or context. We say that a forest or 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 or 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 5.13. *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

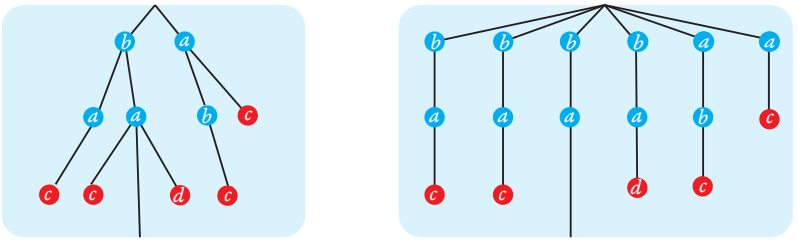
$$\underbrace{a\cdot(b+c)=(a\cdot b)+(a\cdot c)}_{\text{distributivity}} \quad \text{and} \quad \underbrace{b+b=b}_{\substack{\text{idempotence of}\\ \text{the forest semigroup}}} \quad \text{for every } \underbrace{a,b,c}_{\substack{\text{the colour indicates the}\\ \text{sort of the variable}}} \in A.$$

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. Different languages K can be used in different parts of the Boolean combination.

Proof

- (1)⇒(2) In the free algebra $\mathsf{F}\Sigma$, applying the identities from condition (2) does not affect the set of branches.
- (2) ⇒ (1) For a forest or context $t \in \mathsf{F}\Sigma$, define its *branch normal form* to be the forest or 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 or 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[6]. Thanks to the idempotence identity, it is only the set of branches that matters for membership in the language, 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

[6] One could think that the distributivity identity alone (without the identity for idempotence) characterises exactly the languages where membership depends only on the multiset (and not just the set) of branches. This is not true, see Exercise 168.

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 syntactic forest algebra), and therefore it is regular. Finally, $a \in H(t)$ if and only if for some branch the corresponding word is in K_a. Therefore, $H(t)$ can be described 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.

Exercises

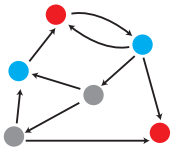
Exercise 168. Give an example of a language $L \subseteq \mathsf{F}\Sigma$ where membership depends only on the multiset of branches, but where the syntactic algebra violates the distributivity identity from Theorem 5.13.

Exercise 169. 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.

5.3.4 Modal logic

We finish this section with a discussion of tree variants for 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:



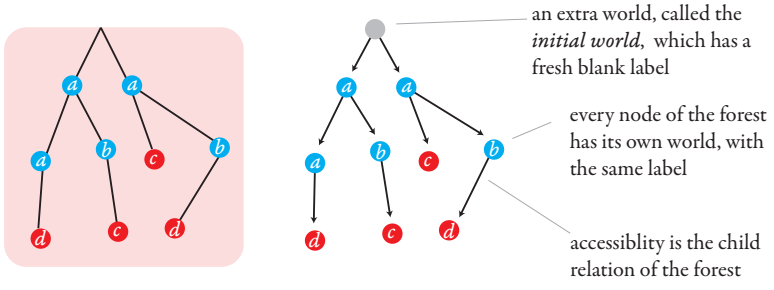
Vertices of the Kripke model are called *worlds*, and the edge relation is called *accessibility*. Accessibility does not need to be transitive. In this section, we only study Kripke models where accessibility is acyclic. To express properties of worlds in Kripke models, we use modal logic, whose formulas are constructed as follows:

$$\underbrace{a}_{\substack{\text{the current}\\ \text{world has}\\ \text{label } a \in \Sigma}} \quad \underbrace{\Diamond\varphi}_{\substack{\text{some}\\ \text{accessible}\\ \text{world}\\ \text{satisfies } \varphi}} \quad \underbrace{\Box\varphi}_{\substack{\text{every}\\ \text{accessible}\\ \text{world}\\ \text{satisfies } \varphi}} \quad \underbrace{\neg\varphi \quad \varphi \wedge \psi \quad \varphi \vee \psi}_{\text{Boolean combinations}}.$$

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 use modal logic to define properties of forests, by assigning a Kripke model to each forest, as explained in the following picture:



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.

Definition 5.15 (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

modal logic, because it suffices to check if the identities hold in the syntactic algebra of a language.

Theorem 5.16. *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*

$$a + a = a \qquad c^! \cdot a = c^! \cdot b \qquad \text{for all } a, b, c,$$

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:

$$\underbrace{a}_{\text{modal rank 0}} \qquad \underbrace{(\Diamond a) \wedge (\Diamond b)}_{\text{modal rank 1}} \qquad \underbrace{(\Diamond(a \wedge \Box b) \wedge (\Diamond b)}_{\text{modal rank 2}}.$$

When the alphabet is finite and fixed, then there are finitely many formulas of given modal rank, up to logical equivalence. This is because, up to logical equivalence, there are finitely many Boolean combinations of formulas from a given set. 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 5.17. *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

[7] For tree-shaped Kripke models this notion coincides with the usual notion of bisimulation for general Kripke models.

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

$$\begin{aligned} c^! a &= && \text{(because } c^! \text{ is idempotent)} \\ c^{!n} a &= && \text{(Claim 5.17)} \\ c^{!n} b &= && \text{(because } c^! \text{ is idempotent)} \\ c^! b. \end{aligned}$$

□

Transitive modal logic. A formula of modal logic as discussed above 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. The formulas stay the same, only the interpretation of forests as Kripke structures changes.

For a forest, define its *transitive Kripke model* in the same way as the Kripke model, except that the accessibility relation now describes the transitive closure of the child relation. In other words, accessibility now represents the proper descendant relation.

Definition 5.18 (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 5.19. *Let $L \subseteq \mathsf{F}\Sigma$ be a language that contains only forests. Then L*

[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. Theorem 5.19 is [13] Bojańczyk and Walukiewicz, "Forest Algebras", 2008 , Theorem 5.3 .

is definable in transitive modal logic if and only if its syntactic forest algebra is finite and satisfies the identities

$$a + a = a \qquad c{\cdot}a{=}(c{\cdot}a){+}a \qquad \text{for all } a, c.$$

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 a reachability ordering on the algebra A defined as follows. We say that $a \in A$ is *reachable* from $b \in A$, denoted by $a \geq b$, 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 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 5.20. *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

$$\underbrace{a = c \cdot b}_{\text{for some context-sorted } c} \quad \text{or} \quad \underbrace{a = c + b}_{\text{for some forest-sorted } c} \quad \text{or} \quad a = b.$$

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 in the reachability ordering. By

Claim 5.20, 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 5.21. *Let $b \in A$ be non-maximal and forest-sorted. Every forest-sorted $c \in A$ that is not reachable from b 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 the ideal I. Because I is an ideal, $\sim$ is a congruence. Because b is non-maximal, the congruence is non-trivial, every forest-sorted equivalence class is definable thanks to the induction assumption on algebra size. Because c is not reachable from b, it does not not belong to the ideal I, and thus its equivalence class consists of c only, and hence c is definable. □

We 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 5.22. *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) *x is not a leaf, its descendant forest has type $d < b$, and $h(\sigma) \cdot d = a$; or*
(2) *x is not a leaf, its descendant forest 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 $\leq b$. This is proved by induction on the number of nodes. 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 statement of 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 $\leq b$, 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, 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 170. Consider transitive modal logic for the monad F_∞ of infinite forests and contexts, as discussed in Section 5. Show that for infinite forests, the equations from Theorem 5.19 are sound (i.e. if a language is definable in transitive modal logic, then the syntactic algebra satisfies the equations) but not complete (i.e. the converse implication to soundness fails).

6

Hypergraphs of bounded treewidth

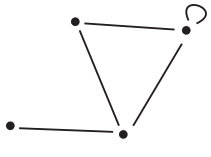
In this chapter, we study algebras for graphs. Although in principle the algebras can describe arbitrary graphs, the more interesting results will assume bounded treewidth.

6.1 Graphs, logic, and treewidth

We begin by discussing graphs, but later we will move to a slightly more general notion, called hypergraphs, which will provide the necessary structure to define a monad. In this chapter, the graphs and hypergraphs are assumed to be finite.

Definition 6.1 (Graph)**.** A *graph* consists of a set of a finite set of vertices, together with a binary symmetric edge relation.

Here is a picture of a graph, with dots representing vertices and lines representing edges:



We use logic, mainly MSO, to define properties of graphs, with graphs represented as models according to the following definition.

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

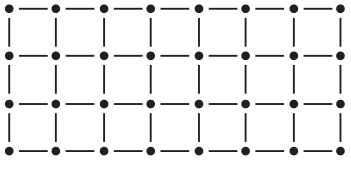
edges, and there is a binary *incidence* relation, which is interpreted as

$$\{(v, e) : \text{vertex } v \text{ is incident with edge } e\}.$$

The two kinds of elements in the universe of the incidence model – vertices and edges – can be distinguished using first-order logic: an edge is an element of the universe that is incident to some vertex, the remaining elements of the universe are vertices.

Monadic second-order logic over the incidence model defined above, which is the main logic of interest in this chapter, is sometimes called MSO_2. A related logic is monadic second-order logic over a representation of graphs where the universe consists only of the vertices, and there is a binary relation for the edges. The related logic is sometimes called MSO_1. The difference is that MSO_2 can quantify over sets of vertices and edges, while MSO_1 can only quantify over sets of vertices. (For first-order logic, the two ways of representing graphs as models does not affect the expressive power, since on first-order quantification over edges can be replaced by two first-order quantifications over vertices.) The difference between MSO_1 and MSO_2 is explained in the following example.

Example 36. A *clique* is a graph where every two vertices are connected by an edge. A *rectangular grid* is a graph that looks like this:



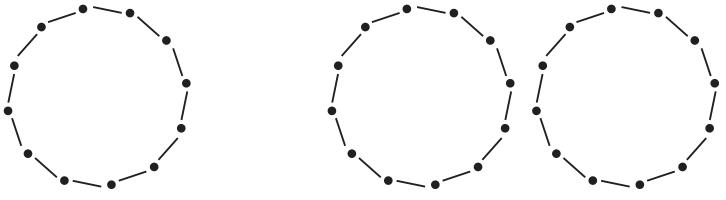
Both cliques and rectangular grids can be defined both in MSO_1 and in MSO_2. Consider now the set of graphs which are cliques of prime size. A clique has prime size if and only if it satisfies the following property: (*) one cannot remove edges so as to get a rectangular grid which has at least two rows and at least two columns. Property (*) can be directly expressed in MSO_2, but it cannot be expressed in MSO_1, see Exercise 172. □

In this chapter, we are mainly interested in monadic second-order logic. First-order logic can only define properties that are local[1], e.g. the existence of a cycle of length three:

$$\exists u\ \exists v\ \exists v \qquad E(u, v) \wedge E(v, w) \wedge E(w, u).$$

[1] The notion of locality is made precise by the Gaifman Theorem, see [34] Heinz-Dieter Ebbinghaus, *Finite Model Theory*, 2006 , Theorem 2.5.1

A classical example of a property that is non-local, and therefore cannot be expressed in first-order logic, is graph connectivity. Using an Ehrenfeucht-Fraïssé argument, one can show that a sentence of first-order logic cannot distinguish between a large cycle and a disjoint union of two large cycles:



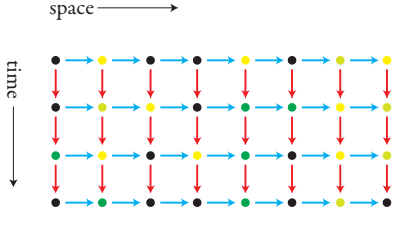
a connected graph a disconnected graph

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

$$\underbrace{\exists X}_{\substack{\text{there is a set}\\ \text{of vertices,}}} \underbrace{(\exists v\ v \in X)\ \wedge (\exists v\ v \notin X)}_{\text{which is neither empty nor full,}} \wedge \underbrace{(\forall v\ \forall w\ E(v,w) \wedge v \in X \Rightarrow w \in X).}_{\text{and which is closed under taking edges}}$$

Already first-order logic is undecidable on graphs, in the following sense: 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 37. Consider directed graphs with coloured vertices and edges. These extra features can be easily encoded, using first-order logic, in the undirected and unlabelled graphs that are discussed in this section, see the exercises. A computation of a Turing machine can be visualised as a coloured rectangular grid, where each vertex represents a tape cell in a given moment of the computation, as in the following picture:



By formalising the definition of a computation of a Turing machine, one can write a sentence of first-order logic, which is true in a connected graph if and only if it represents an accepting computation of a given Turing machine. From

this, one can deduce that the halting problem reduces to satisfiability for first-order logic over finite graphs, see Exercise 174. □

We will no longer discuss first-order logic for graphs. Also, from now on, when talking about MSO, we mean the MSO_2 variant that uses the incidence model from Definition 6.2[2].

Exercises

Exercise 171. Show that for graphs without edges, MSO_1 and MSO_2 has the same expressive power as first-order logic.

Exercise 172. Show that the set of cliques of prime size from Example 36 cannot be defined in MSO_1.

Exercise 173. For the purposes of this exercise, we consider directed graphs with two types of edges, blue and red. For such a graph, the associated model has the vertices as the universe, and two binary predicates for the red and blue edges. Show that rectangular grids, as described in Example 37, can be defined in first-order logic. We assume that the input graph has one connected component.

Exercise 174. Show that the following problem is undecidable: given a sentence of first-order logic, decide if it is true in at least one finite graph.

Exercise 175. Unlike for the rest of this chapter, this exercise and the next one consider possibly infinite graphs. Consider two decision problems: (a) is a sentence of first-order logic true in at least one finite graph; (b) is a sentence of first-order logic true in at least one possibly infinite graph. Show that (a) is recursively enumerable (there is a Turing machine that accepts yes-instances in finite time, and does not halt on no-instances), while (b) is a co-recursively enumerable (there is a Turing machine that does not halt on yes-instances, and rejects no-instances in finite time).

Exercise 176. Show that for every $k \in \{1, 2, \ldots\}$ the following property of

[2] This difference is not so important in the context of this chapter. This because we will be mainly interested in graphs of bounded treewidth, and for bounded treewidth the logics MSO_1 and MSO_2 models are equivalent, see Exercise 186

graphs is definable in MSO using the incidence model: "the graph is connected, infinite, and has degree at most k". Show that "the graph is connected and infinite" is not definable.

Exercise 177. Show that the existence of a Hamiltonian cycle cannot be expressed in MSO, using the MSO_1 representation of graphs as models.

Exercise 178. Show that the existence of an Euler cycle (every edge is visited exactly once) cannot be expressed in MSO, using the MSO_1 representation of graphs as models.

Exercise 179. Consider graphs which allow parallel edges (i.e. multiple edges connecting the same two vertices). The incidence model makes sense for such graphs as well. 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 (with parallel edges) if and only if it is true in its ℓ-reduction.

6.1.1 Treewidth

The undecidability problems described in Example 37 are avoided if we consider graphs that are similar to trees. The notion of similarity that we care about is treewidth, as defined below[4].

Definition 6.3 (Tree decompositions)**.** A *tree decomposition* consists of:

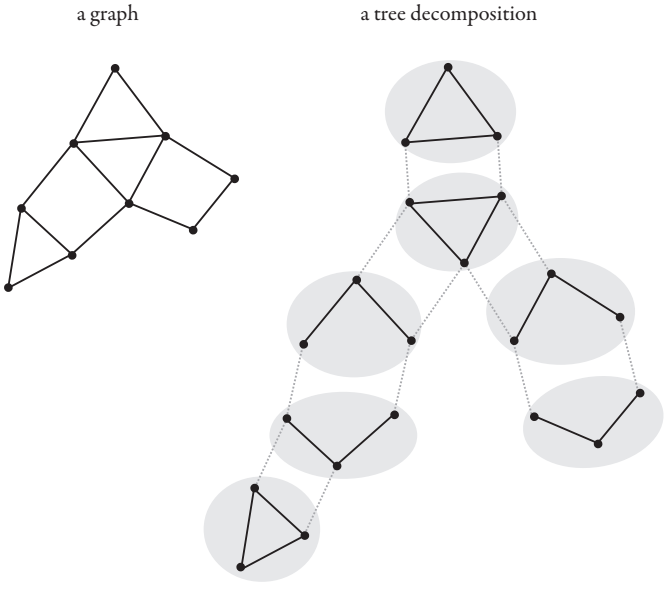
- a graph, called the *underlying graph*;
- a set of *nodes*, equipped with a tree ordering (i.e. there is a least node called the root, and for every node x, the set of nodes $< x$ is totally ordered);
- for each node, an associated nonempty set of vertices called its *bag*.

These should satisfy the following constraints:

(1) every edge in the underlying graph is covered by some bag, i.e. there is some bag that contains both endpoints of the edge;
(2) every vertex v of the underlying graph is *introduced* in exactly one node, which means there is exactly one node x such that v is in the bag of x and either x is the root or v is not in the bag of the parent of x.

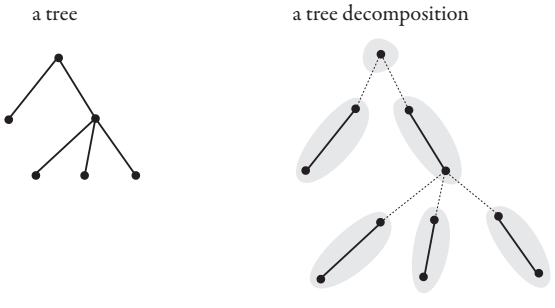
[4] For an introduction to treewidth, including a brief history, see
[23] Diestel, *Graph theory (electronic edition)*, 2006 , Section 12.

Here is a picture of a tree decomposition:



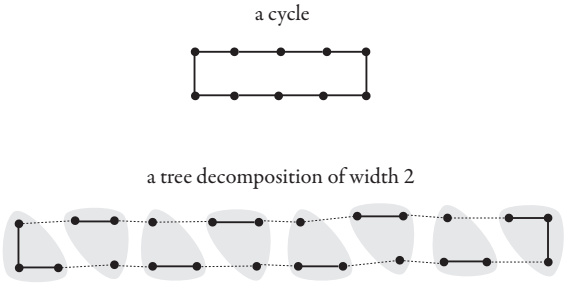
In the picture, the gray circles are bags, and the dotted lines connect appearances of the same vertex in several bags. The *width* of a decomposition is defined to be the maximal size of bags, minus one. For example, the tree decomposition in the above picture has width two, because its maximal bag size is three. The *treewidth* of a graph is the minimal width of a tree decomposition for the graph.

The reason for the minus one in the definition of width is so that trees, where the bags in the natural have tree decomposition have size two, get assigned treewidth one. This is illustrated in the following picture:



Forests (i.e. disjoint unions of trees) are the only graphs of treewidth one.

Cycles have treewidth two, 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. Path decompositions will play an important role in Section 6.3.

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 explained in the following example.

Example 39. We show that for cliques, every tree decomposition must have a bag which contains all vertices. Consider a tree decomposition of a clique. If two vertices in a tree decomposition are connected by an edge, then the nodes which introduce these two vertices must be related by the ancestor relation (if they would be unrelated, then there could be no bag that contains both vertices). Therefore, in a tree decomposition of a clique, the nodes that introduce the clique vertices must be linearly ordered by the ancestor relation. The maximal, i.e. furthest from the root, node in this linear order must have all vertices of the clique in its bag. □

Another example of graphs with unbounded treewidth is rectangular grids, see the exercises. In fact, the Grid Theorem[5], 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 rectangular 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. In the exercises we also discuss a corollary of the Grid Theorem, which says that decidability of MSO for bounded treewidth is optimal: if the MSO theory of a class of graphs is decidable, then the class has bounded treewidth.

[5] For a recent paper about the Grid Theorem, see
[17] Chuzhoy and Tan, "Towards Tight(er) Bounds for the Excluded Grid Theorem", 2019

Exercises

Exercise 180. We say that a graph G is a *minor* of a 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:

$$\underbrace{U \text{ is connected in } G}_{\substack{\text{a subset of vertices is connected if} \\ \text{the induced subgraph is connected}}} \quad \text{implies} \quad (\bigcup_{v \in U} X_v) \text{ is connected in } H.$$

(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 graphs that is definable in MSO using the incidence model (we use the incidence model for the remaining exercises), then the same is true for "some minor satisfies L".

Exercise 181. The Grid Theorem says that if a class of 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 graphs has decidable MSO theory, then it has bounded treewidth.

Exercise 182. Show a class of graphs that has undecidable MSO theory and bounded treewidth.

Exercise 183. Show that the $n \times n$ grid has treewidth at least n.

Exercise 184. Show that for every language $L \subseteq \{a\}^*$ recognised in linear time by a (possibly nondeterministic) Turing machine, the language

$$\{G : G \text{ is an } n \times n \text{ grid such that } a^n \in L\}$$

is definable in MSO.

Exercise 185. Show that if a graph has treewidth at most k, then one can choose an orientation of its edges so that every vertex has at most $k+1$ outgoing edges.

Exercise 186. Show that for every $k \in \{1, 2, \ldots\}$, the logics MSO_1 and MSO_2 have the same expressive power for graphs of bounded treewidth.

6.2 The hypergraph monad and Courcelle's Theorem

In this section, we introduce algebras for graphs. These algebras are defined in terms of a monad that describes graphs[6]. In order to define the monad, we will need to add more structure to graphs, namely labels, directed hyperedges (i.e. edges that connect a number of vertices that is not necessarily two), and distinguished vertices called ports. We use the name *hypergraph* for graphs with such extra 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 Courcelle's Theorem, which says that every graph property definable in MSO is necessarily recognisable. In the next Section 6.3, we prove a converse to Courcelle's Theorem, which says that for bounded treewidth, recognisability implies definability in MSO.

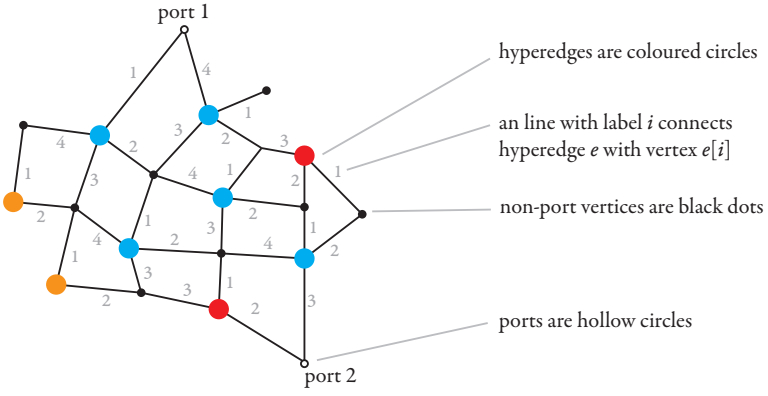
We begin with a formal definition of hypergraphs.

Definition 6.4. A *hypergraph* consists of:

- A set V of *vertices*.
- A set E of *hyperedges*. Each hyperedge has an associated arity in $\{0, 1, \ldots\}$.
- A set Σ of *labels*. Each label has an associated arity in $\{0, 1, \ldots\}$.
- A non-repeating sequence of distinguished vertices called *ports*;
- For each hyperedge e, an associated label in Σ of same arity, and a non-repeating sequence of *incident vertices* whose length is the arity of e.

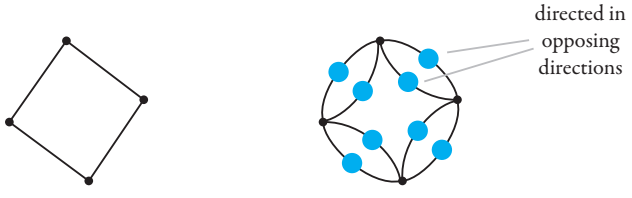
In the end, we care mainly about hypergraphs that have no ports, i.e. the sequence of ports is empty, but the ports will appear when decomposing hypergraphs into parts. We use the name *non-port vertices* for vertices that are not ports. For a hyperedge e of arity n, we write $e[1], \ldots, e[n]$ for the sequence of incident vertices, and use the name *incidence list* for this sequence. In this chapter, all hypergraphs are assumed to be finite, which means that there are finitely many vertices and hyperedges. We draw hypergraphs like this:

[6] This monad is based on the hyperedge replacement algebras of Courcelle. A discussion of these algebras can be found in
[21] Courcelle and Engelfriet, *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, 2012 , Section 2.3.
The presentation of hyperedge replacement that uses monads is based on
[8] Bojańczyk, "Two Monads for Graphs", 2018
.



To avoid clutter in the pictures, we skip the gray numbers on the edges and the numbers of the ports, in situations where they are not important for the picture or implicit from the context.

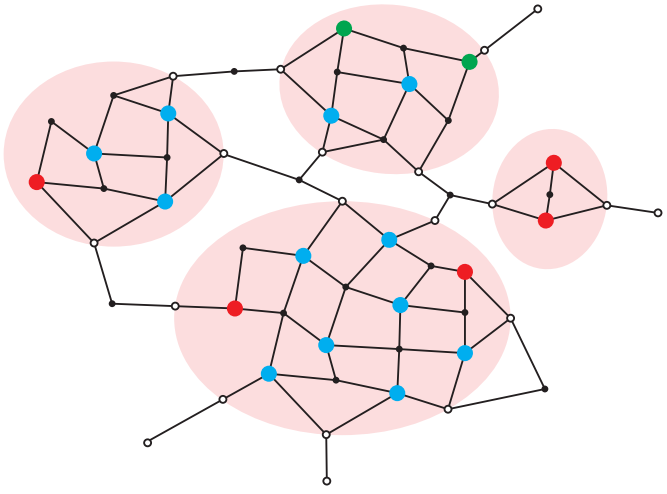
A graph can be represented as a hypergraph. The representing hypergraph has no ports, and the vertices are the same as in the graph. Each edge of the graph is represented by two binary hyperedges (with some fixed label), one in each direction. Here is a picture:



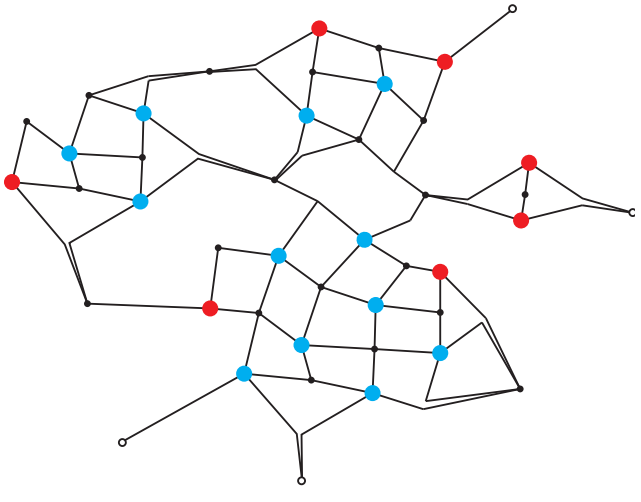
Directed graphs can be represented in the same way, but with the hyperedges not necessarily using both opposing directions.

The hypergraph monad. We now describe the monad structure of hypergraphs. The main idea behind free multiplication is that a hyperedge can be replaced by a hypergraph of matching arity[7]. This replacement, which will be the free multiplication in the monad, is illustrated in the Figure 37.

[7] This is the reason why Courcelle uses the name *hyperedge replacement* for the corresponding algebras.



a hypergraph labelled by hyperegraphs



its free multiplication

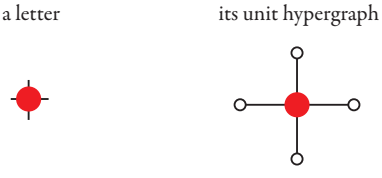
Figure 6.1 Free multiplication in the hypergraph monad.

Definition 6.5 (Hypergraph monad)**.** The hypergraph monad, denoted by H, is defined as follows.

- The underlying category is the *category of ranked sets*

$$\mathsf{Set}^{\{0,1,\ldots\}},$$

  which is the category of sorted sets, where the sort names are natural numbers. Objects in this category 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 finite hypergraphs labelled by Σ, modulo isomorphism. 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 operation in the monad associates to every letter $a \in \Sigma$ of arity n a hypergraph which has n ports, no other vertices, and one hyperedge labelled by a which is incident to all ports (in increasing order). Here is a picture:



- Let $G \in \mathsf{HH}\Sigma$ be a hypergraph labelled by hypergraphs. 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 non-port 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

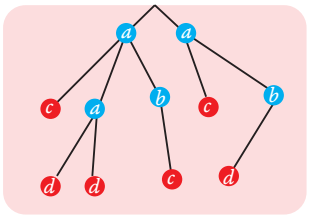
$$(e, f)[i] = \begin{cases} f[i] & \text{if } f[i] \text{ is a non-port vertex} \\ e[j] & \text{if } f[i] \text{ is the } j\text{-th port.} \end{cases}$$

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.

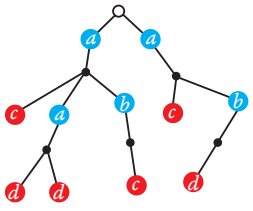
The hypergraph monad generalises the forest monad, as shown in the following example.

Example 40. A forest can be represented as a hypergraph of arity one, as explained in the following picture:

a forest



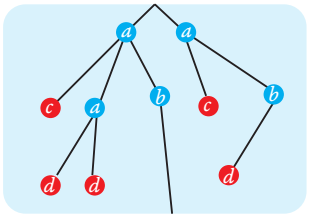
its hypergraph representation



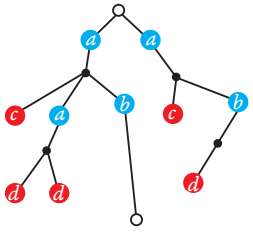
Nodes of forest type in the forest are represented by hyperedges of arity one, while nodes of context type are represented by hyperedges of arity two.

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 the hypergraph monad, such as algebras, homomorphisms, recognisability, and terms.

Exercises

Exercise 187. Show that H satisfies the monad axioms.

Exercise 188. Show that connected hypergraphs are also a monad.

6.2.1 Recognisable languages

In this chapter, we are mainly interested in languages recognised by algebras in the hypergraph monad. We use the name *hypergraph algebra* for such algebras. We are especially interested in languages recognised by hypergraph algebras that are finite in the sense that they are finite on every arity.

Example 41. Let M be a commutative monoid. Define a hypergraph algebra as follows. The underlying ranked set A has a copy of M on each arity, i.e. the underlying set is

$$M \times \underbrace{\{0, 1, \ldots\}}_{\text{the arity}}.$$

The multiplication operation in the hypergraph algebra inputs a hypergraph in $\mathsf{H}A$ and outputs 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 copy of M that corresponds to the arity of the input hypergraph. It is not hard to see that this operation is associative, i.e. it satisfies the axioms of Eilenberg-Moore algebras.

The hypergraph algebra constructed this way can be used to recognise some simple languages of hypergraphs. Apply the above construction to the commutative monoid

$$M = (\{0, 1\}, \vee),$$

yielding a hypergraph algebra A. This algebra recognises the language

$$\{G \in \mathsf{H}\Sigma : \text{some hyperedge has label in } \Gamma\} \qquad \text{for ranked sets } \Gamma \subseteq \Sigma.$$

The homomorphism maps a hypergraph to 1 if it belongs to the language, and to 0 otherwise, 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 hypergraph algebras in the above example are infinite, but finite on every arity. This is the best we can do in the hypergraph monad, because it is impossible for a hypergraph algebra to have an underlying set that is finite altogether. The reason is that the multiplication operation $\mu : \mathsf{H}A \to A$ in a hypergraph algebra is arity-preserving, and $\mathsf{H}A$ is nonempty on every arity (as witnessed by hypergraphs without hyperedges). Therefore, the underlying set of a hypergraph algebra must be nonempty on every arity. In the following

definition, and for the rest of this chapter, we assume that "finite hypergraph 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 a hypergraph 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 44. In fact, no entirely satisfactory definition of "finite algebra" for general hypergraphs is known, and possibly does not exist. However, for hypergraphs of bounded treewidth, hypergraph algebras that are finite on every sort will be a satisfactory definition that is equivalent to MSO, as we will see in Section 6.3.

In Example 41, we already saw some examples of recognisable languages of hypergraphs. Here are some more examples.

Example 42. 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 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. The *source* of the path is the vertex v_0, its *target* is the vertex v_n, and we say that the path *connects* the source with the target. A hypergraph is called *connected* if every vertex can be connected to every other vertex via a path. Define h to be the function which maps a hypergraph to the following information: (a) its arity; (b) is there a pair of non-connected vertices such that at least one of them is not a port; and (c) which pairs of ports are connected. One can check that this function is compositional, and therefore its image can be equipped with the structure of a hypergraph algebra so that h is a homomorphism. The corresponding hypergraph algebra is finite on every arity. Therefore, the language of connected hypergraphs is recognisable. □

Example 43. In this example, we show that the language of k-colourable hypergraphs is recognisable. Define a *k-colouring* of a hypergraph to be a function from vertices to $\{1, \ldots, k\}$ such that no hyperedge has an incidence list that uses some colour twice. (In particular, all hyperedges 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, at least as long as hypergraphs of unbounded treewidth are allowed.

Example 44. We say that a hypergraph is a *clique* if every two vertices are adjacent (i.e. connected by some hyperedge). Let

$$P \subseteq \{0, 1, \ldots\}$$

be any set of natural numbers, possibly undecidable. We will show that the language "cliques with no ports, where the number of vertices is in P" is recognisable. Let h be the function which maps a hypergraph to the following information: (a) its arity; (b) is there a pair of non-adjacent vertices such that at least one of them is not a port; (c) which ports are adjacent. If the arity is zero, then h also stores: (d) 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. □

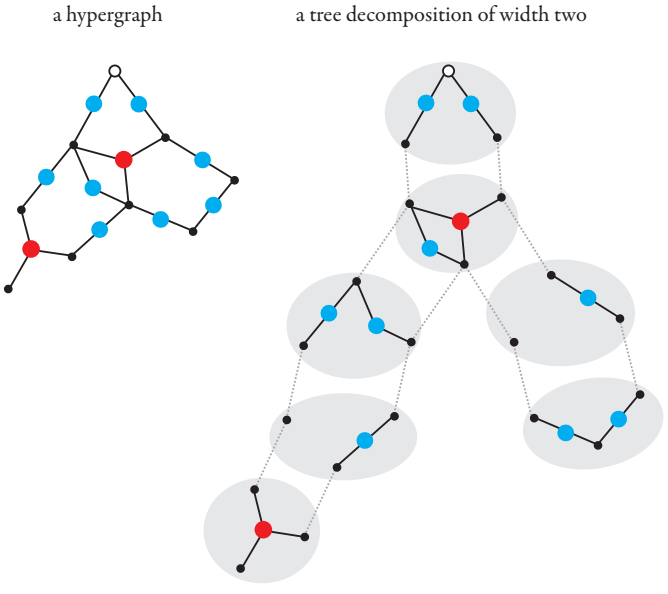
As we will see later on, the problem from the above example will disappear once we restrict attention to hypergraphs of bounded treewidth.

Exercises

Exercise 189. Show that every recognisable language in the hypergraph monad has a syntactic algebra.

6.2.2 Terms and tree decompositions.

Tree decompositions and treewidth can be naturally extended to hypergraphs, as formalised in Definition 6.7 below, and illustrated in the following picture:



In this section we discuss an alternative perspective on treewidth, which is defined using monad terminology.

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) for every hyperedge there must be some bag which contains its entire incidence list (we say that such a bag *covers* the hyperedge); (b) every port of the hypergraph appears in the root bag.

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 cover 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, terms in the hypergraph monad are sorted, which means that each variable used by the term has an arity, and the term itself has an arity. If A is a hypergraph algebra, then a term $t \in \mathsf{H}X$ induces a *term operation* t^A and defined by

$$\underbrace{\eta \in A^X}_{\substack{\text{an arity-preserving}\\ \text{valuation of the variables}}} \quad \mapsto \quad \underbrace{\text{multiplication in } A \text{ applied to } (\mathsf{H}\eta)(t).}_{\substack{\text{an element of the hypergraph algebra } A,\\ \text{whose arity is the same as the arity of } t}}$$

As was the case for forest algebra, term operations are in general not arity-preserving, if only because their inputs do not have a well-defined arity.

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 hypergraph algebra used in the lemma is the free hypergraph 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 of the tree decomposition whose bag contains the incidence list of the hyperedge, remove the hyperedge, and add a child to this node with a tree decomposition of $\eta(x)$. □

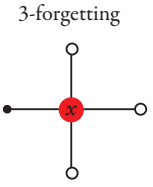
A corollary of the above lemma is that there is a well-defined monad for hypergraphs of treewidth at most k. This monad, call it H_k, uses only hypergraphs with treewidth at most k, with all the monad structure inherited from H. The underlying category is ranked sets with arities at most $k+1$; since hypergraphs with bigger arities will have treewidth at least $k+1$.

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[8] from Figure 6.2. For a hypergraph 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.

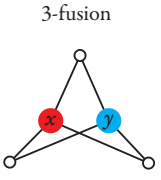
Theorem 6.9. *Let $k \in \{1, 2, \ldots\}$. A hypergraph has treewidth at most k if and*

[8] There is an inconsistency in our use of the words "introduce" and "forget". When we say that a node in a tree decomposition introduces a vertex, we take a top-down perspective on tree decompositions. On the other hand, the name of the "forget" term in Figure 6.2 is based on a bottom-up perspective of the same phenomenon.

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 $(f(1),\ldots,f(k))$.

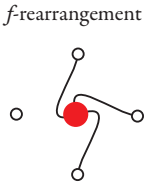


Figure 6.2 The treewidth terms. The parameters k, ℓ are from $\{0,1,\ldots\}$. In the pictures, the ports are ordered clockwise from top, and the same is true for the vertices incident to a hyperedge.

only if it can be generated (in the free hypergraph 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 satisfies: (*) the root bag contains the ports and no other vertices, and if a node has at least two children, then the node and all of its children have the same bag. 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 3.14, we

get the following corollary, which gives a finite representation for algebras in the monad H_k of hypergraphs of treewidth at most k.

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[9] , which says that all languages definable in MSO are recognisable. To define properties of hypergraphs in MSO, we use a hypergraph version of the incidence model, defined as follows.

Definition 6.11 (Incidence model)**.** The incidence model of a hypergraph is defined as follows. The universe is vertices and hyperedges, and it is equipped with the following relations:

$$\underbrace{e[i] = v}_{\substack{v \text{ is the } i\text{-th} \\ \text{vertex incident to } e}} \qquad \underbrace{\mathrm{port}_i(v)}_{\substack{v \text{ is the} \\ i\text{-th port}}} \qquad \underbrace{a(e)}_{\substack{\text{hyperedge } e \\ \text{has label } a}} .$$

The arguments of the relations are e and v, while $i \in \{1, 2, \ldots\}$ and $a \in \Sigma$ are parameters. Each choice of parameters gives a different relation.

As was the case for forest algebra, recognisability holds also for *counting* MSO, which extends MSO by allowing the following form of modulo counting: for every $n \in \{2, 3, \ldots\}$ and every $\ell \in \{0, 1, \ldots, n-1\}$ there is a predicate

$$|X| \equiv \ell \mod n,$$

which inputs a set and says if the size of this set is congruent to ℓ modulo 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, see Exercise 171. As we will show in Section 6.3, counting MSO is enough to describe all recognisable properties of hypergraphs, assuming bounded treewidth.

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 a hypergraph algebra that is finite on every arity.*

[9] [20] Courcelle, "The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs", 1990 , Theorem 4.4

We use the same construction as in previous chapters. The main step of the proof, which deals with set quantification, is presented in the following lemma.

Lemma 6.13. *The recognisable languages images under functions of the form*[10]

$$\underbrace{\mathsf{H}f : \mathsf{H}\Sigma \to \mathsf{H}\Gamma \qquad \text{for } f : \Sigma \to \Gamma}_{\text{such functions are called letter-to-letter homomorphisms.}}$$

Proof We use a powerset construction for hypergraph algebras. Since we have already used powerset constructions before, we take this opportunity to discuss powerset constructions in more detail and generality, so that we can think about the kinds of monads that allow a powerset construction (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[11]. For a ranked set X, 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.

Although trivial in the hypergraph monad, the following claim is not true in general for all monads, e.g. it would be false in the group monad for a naturally defined distribution.

Claim 6.14. *Distribution is a natural transformation, which means that the following diagram commutes for every arity-preserving function* $f : X \to Y$

$$\begin{array}{ccc} \mathsf{HP}X & \xrightarrow{\mathsf{HP}f} & \mathsf{HP}Y \\ {\scriptstyle \text{distribute on } X}\downarrow & & \downarrow{\scriptstyle \text{distribute on } Y} \\ \mathsf{PH}X & \xrightarrow[\mathsf{PH}f]{} & \mathsf{PH}Y \end{array}$$

Proof The right-down path corresponds to the following procedure: for each hyperedge, choose an element of its label, and then apply f. The down-right path corresponds to the following procedure: for each hyperedge, take the image under f of its label, and then choose an element. The two procedures give

[10] In Exercise 192 we show that the assumption on letter-to-letter homomorphisms is important.

[11] In the language of category theory, P is the co-variant powerset functor, as opposed to the contra-variant powerset functor which uses inverse images instead of forward images.

the same result. This is true thanks to the following property of distribution for hypergraphs: if we apply distribution on X to some hypergraph, then every hypergraph in the resulting set will have the same vertices, ports and hyperedges as the original hypergraph. This property would not hold, for example, in the group monad, and the claim would be false in the group monad[12]. □

We 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

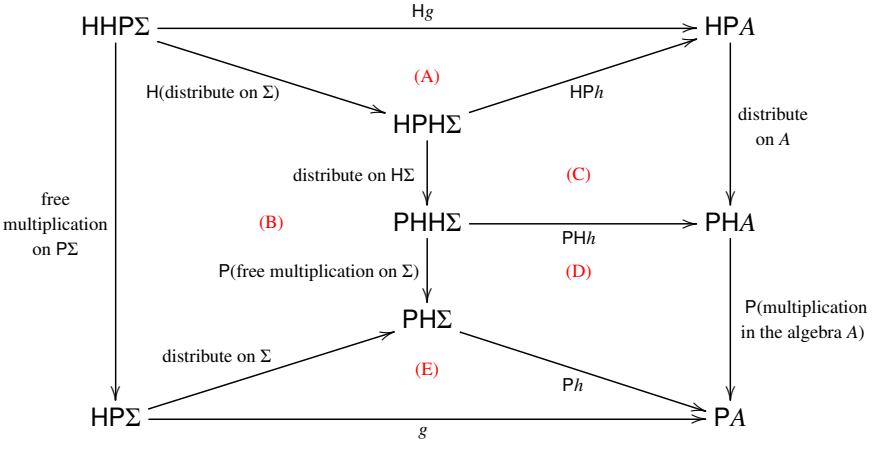
$$\mathsf{H}f : \mathsf{H}\Sigma \to \mathsf{H}\Gamma.$$

Define g to be the composition of the following two functions:

$$\mathsf{HP}\Sigma \xrightarrow{\text{distribute on } \Sigma} \mathsf{PH}\Sigma \xrightarrow{\mathsf{P}h} \mathsf{P}A.$$

Claim 6.15. *The function g is compositional.*

Proof Consider the following diagram, with red letters being labels of faces:

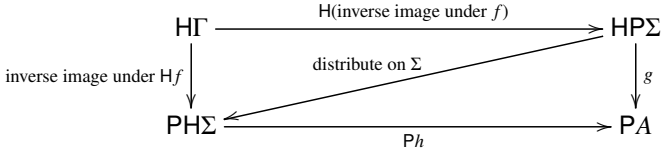


If we prove that the perimeter of the diagram commutes, then we will prove that g is compositional (the composition of the two arrows on the right-most side will be the multiplication operation in the powerset algebra $\mathsf{P}A$). Faces (A) and (E) commute by definition of g. Face (C) commutes by naturality of

[12] 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.

distribution from Claim 6.14, and face (D) commutes because h is a homomorphism. It remains to show that face (B) commutes. This again, is proved via simple check[13], 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. (As mentioned in the proof of the claim above, this multiplication operation is the composition of the two arrows on the right-most side side from the diagram.) We will use the homomorphism g to recognise the image of L under $\mathsf{H}f$. Consider the following diagram:



The top-left face in the diagram commutes by definition of distribution, and the bottom-right face commutes by definition of g. A hypergraph $G \in \mathsf{H}\Gamma$ belongs to the image of the language L under the function $\mathsf{H}f$ 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 of the language L under the function $\mathsf{H}L$. 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. □

The above lemma implies that recognisable languages are closed under quantification of sets of hyperedges (since a subset of the hyperedges can be seen as a colouring of hyperedges with two colours "yes" and "no"). This motivates the following logic.

Definition 6.16. Define *hyperedge counting* MSO to be the following variant of MSO. There is no first-order quantification, and set quantifiers range over sets of hyperedges. The logic allows the following relations on sets of hyperedges:

$\underbrace{X \subseteq Y}_{\text{set inclusion}}$ $\underbrace{X \subseteq a}_{\text{every hyperedge in } X \text{ has label } a \in \Sigma}$ $\underbrace{i \in X[j]}_{\text{there exists a hyperedge } e \in X \text{ such that } e[j] \text{ is the } i\text{-th port}}$ $\underbrace{X[i] \cap Y[j] \neq \emptyset}_{\text{there exist hyperedges } e \in X \text{ and } f \in Y \text{ such that } e[i] = f[j]}$ $\underbrace{|X| \equiv 0 \mod n.}_{\text{the number of hyperedges in } X \text{ is dvisible by } n}$

[13] In the language of category theory, face (B) is the main axiom of a distributive law of a monad over a functor.

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.

Our usual proof of the translation of MSO to algebras shows the following result, which is almost Courcelle's theorem, except that the logic is hyperedge MSO instead of counting MSO. The minor difference between the two logics – which boils down to isolated vertices – will be treated later on.

Lemma 6.17. *If a language $L \subseteq \mathsf{H}\Sigma$ is definable in hyperedge counting* MSO*, then it is recognisable.*

Proof Same proof as for the monads for words and forests. Consider a formula of hyperedge counting MSO

$$\varphi(\underbrace{X_1, \ldots, X_n}_{\substack{\text{the free variables represent} \\ \text{ses of hyperedges}}}),$$

where Σ is the ranked set of labels used by the underlying hypergraphs. Define the *language* of this formula to be the set hypergraphs over an extended alphabet that consists of 2^n disjoint copies of Σ. This language is defined in the same way as for words and forests: for each hyperedge, the bits from 2^n in its label determine which of the sets $X_1, \ldots, X_n$ contain the hyperedge. By induction on formula size, we prove that every formula has a recognisable language. The induction step is proved in the same way as for words and forests: 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 41. 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

$$X[i] \cap Y[j] \neq \emptyset.$$

□

Lemma 6.18. *For every sentence of counting* MSO*, there is a sentence of hypergraph counting* MSO *which gives the same results on hypergraphs without isolated vertices.*

Proof Let n be the maximal arity of letters in the finite alphabet. Every set X of non-isolated vertices can be represented by n sets of hyperedges as

$$X = X_1[1] \cup \cdots \cup X_n[n],$$

where X_i is the set of hyperedges whose i-th incident vertex is in X. Using this representation, we can quantify over sets of non-isolated vertices by using quantification over sets of hyperedges. □

A corollary of Lemmas 6.17 and 6.18 is that for every language definable in counting MSO, there is a recognisable language that agrees with it on hypergraphs without isolated vertices. To finish the proof of Courcelle's Theorem, we need to take into account the isolated vertices, which is a minor inconvenience that is left as an exercise for the reader, see Exercises 194 and 195.

This completes the proof of Courcelle's Theorem.

Exercises

Exercise 190. Consider graphs (not hypergraphs). Show that the existence of an Eulerian cycle can be defined in counting MSO, but not in MSO.

Exercise 191. Show that the existence of a Hamiltonian cycle cannot be defined in counting MSO with set quantification restricted to sets of vertices (and not hyperedges).

Exercise 192. Show that Lemma 6.13 ceases to be true if we allow homomorphisms that are not necessarily letter-to-letter.

Exercise 193. Show that for every MSO formula $\varphi(X)$ with one free set variable, the following problem can be solved in linear time:

- *Input.* A tree decomposition T;
- *Output.* The maximal size of a set of vertices X, such that $\varphi(X)$ is true in the underlying hypergraph.

Exercise 194. For a hypergraph G, define two hypergraphs $\alpha(G)$ and $\beta(G)$ as follows:

- $\alpha(G)$: remove all isolated vertices;
- $\beta(G)$: remove all hyperedges and non-isolated vertices.

The functions α and β are not arity-preserving, since the arity of G is equal to the sum of arities of $\alpha(G)$ and $\beta(G)$. Show that every sentence of counting MSO is equivalent to a finite Boolean combination of sentences of counting MSO, each of which talks about only $\alpha(G)$ or $\beta(G)$.

Exercise 195. Recall the functions α and β from the previous exercise. Show that if L is a language of hypergraphs that is definable in MSO, then the same is true for the languages

$$\{G : \alpha(G) \in L\} \qquad \text{and} \qquad \{G : \beta(G) \in L\}.$$

Together with Exercise 194, this observation 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, assuming bounded treewidth. Recall that already first-order logic on graphs has undecidable satisfiability, and this undecidability carries over to the more general setting of hypergraphs and counting MSO. We recover decidability if we restrict attention to hypergraphs of bounded treewidth.

Theorem 6.19. *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 a 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). The algorithm also says if there are finitely many elements of arity k or not. We say that a hypergraph 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 hypergraph algebras over computable alphabets are computable, all of

the hypergraph algebras that we used as recognisers for the atomic relations in the proof of 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 hypergraph algebra. The hypergraph 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 hypergraph 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 196. Show that the following problem is decidable: given a first-order formula, decide if it is true in some rectangular grid. Here we are talking about unlabelled rectangular grids as in Example 36, and not labelled rectangular grids as in Example 37.

Exercise 197. Show that if a hypergraph language L has bounded treewidth and is definable in counting MSO, then its syntactic algebra is computable.

Exercise 198. Show that the following problem is decidable: given $k \in \{1, 2, \ldots\}$ and an MSO sentence φ, decide if φ is true in infinitely many hypergraphs of treewidth at most k.

6.3 Definable tree decompositions

In this section[14], we show that for hypergraphs of bounded treewidth, tree decompositions can be defined in MSO. One application of this result is going to be a converse of Courcelle's Theorem for bounded treewidth: every recognisable property is definable in counting MSO for hypergraphs of bounded treewidth.

We begin by explaining how a tree decomposition can be defined in MSO. This is split into two ingredients: in Definition 6.20 we represent a tree decomposition using a binary relation on vertices, and then in Definition 6.21 we show how such a binary relation can be defined in MSO.

The introduction ordering. We represent a tree decomposition using the order in which vertices of the underlying hypergraph are introduced.

Definition 6.20 (Introduction ordering)**.** Define the *introduction ordering* of a tree decomposition to be the following binary relation on vertices in the underlying hypergraph:

$$\underbrace{v \text{ is introduced in the same node as, or an ancestor of, the node introducing } w.}_{\text{we say that } v \textit{ is introduced before vertex } w \text{ of the tree decomposition}}$$

The introduction ordering is a pre-order, i.e. it is transitive and reflexive. It need not be anti-symmetric, because several vertices might be introduced in the same node.

Example 45. Consider a hypergraph which has no ports or hyperedges, but only isolated vertices, like in the following picture:

One tree decomposition for this hypergraph has a node for each vertex, with the bag containing only that vertex, and with the nodes ordered left-to-right. Its introduction ordering is in the following picture:

An alternative tree decomposition, has the same nodes and bags. However, this time we have some chosen root, and the remaining nodes are its children. Here is the introduction ordering for the alternative tree decomposition:



[14] The results of this section are based in
[11] Bojańczyk and Pilipczuk, "Definability Equals Recognizability for Graphs of Bounded Treewidth", 2016

In both pictures above, the introduction ordering is anti-symmetric, because each node of the tree decomposition introduces a single vertex. Here is a picture of an introduction ordering which is not anti-symmetric (and has two components):



□

We now explain how a tree decomposition can be recovered from its introduction ordering. To do this, we use two mild assumptions on tree decompositions (in the following, we say that a hyperedge is *introduced* in node x if x is the least node that contains the incidence list of the hyperedge):

(A) every node introduces at least one vertex;

(B) if a vertex v is in the bag of node x, then it is incident to some hyperedge that is introduced in x or its descendants.

Every tree decomposition can be transformed into a tree decomposition that satisfies (A) and (B), without increasing the width. In order to satisfy (A), we merge every node that does not introduce any vertices with its parent. In order to satisfy (B), we remove a vertex v from all bags that violate condition (B).

If a tree decomposition satisfies (A) and (B), then it can be recovered from its introduction ordering as follows. Thanks to condition (A), the nodes of the tree decomposition are equivalence classes of vertices with respect to the equivalence "v is introduced before w and vice versa", and the tree order on nodes is the inherited from the introduction ordering. Thanks to condition (B), a vertex v is present in the bag of a node x if and only if there is hyperedge that is incident to v and a vertex that is introduced in x or its descendants. The way that we recover a tree decomposition from its introduction ordering can be formalised in MSO.

For the rest of Section 6.3, we only consider tree decompositions that satisfy (A) and (B).

Relations definable using set parameters. To represent the introduction ordering of a tree decomposition, we will use a formula of MSO that is equipped with extra set parameters, as described in the following definition. In the definition, when evaluating an MSO formula in a hypergraph, we use the incidence model from Definition 6.2, where the universe is both vertices and hyperedges.

Definition 6.21 (Definable tree decompositions)**.** An MSO *formula with set parameters* is an MSO formula of the form

$$\varphi(\underbrace{Y_1, \ldots, Y_n}_{\substack{\text{set variables}\\ \text{called}\\ \textit{set parameters}}}, \underbrace{x_1, \ldots, x_m}_{\substack{\text{element variables}\\ \text{called}\\ \textit{arguments}}}).$$

We say that an m-ary relation R in a hypergraph is *definable* by φ if

$$\underbrace{\exists Y_1 \cdots \exists Y_n}_{\substack{\text{there is a choice}\\ \text{of set parameters}}} \underbrace{\forall x_1 \ \cdots \forall x_m \quad R(x_1, \ldots, x_m) \Leftrightarrow \varphi(Y_1, \ldots, Y_n, x_1, \ldots, x_m).}_{\substack{\text{such that after fixing these set parameters in } \varphi,\\ \text{we get exactly the relation } R}}$$

We say that a tree decomposition is *definable by* φ if its introduction ordering is definable by φ in the underlying graph. We say that a set of hypergraphs has *definable tree decompositions of bounded width* if there is an MSO formula φ with set parameters, and a width $k \in \{0, 1, \ldots\}$, such that every hypergraph from the set has a tree decomposition that is definable by φ and has width at most k.

Since the above definition uses the incidence model for hypergraph, the set parameters can use hyperedges, even though the introduction ordering itself uses only vertices. Note also that the definition uses MSO, and not counting MSO. We will show that bounded treewidth implies definable tree decompositions of bounded width; not using counting will make the result stronger.

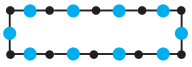
Example 46. Define an *independent set* to be a hypergraph that has only vertices and no ports or hyperedges, as discussed in Example 45. We will show that independent sets have definable tree decompositions of bounded width. There is a minor difficulty, which is that we need to avoid the path decompositions where the introduction ordering looks like this:

The reason is that there is no single MSO formula with set parameters that can define a linear order on every independent set, see Exercise 202. The solution is to consider tree decompositions of depth two, where the introduction ordering looks like this:

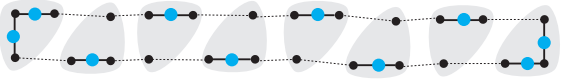


The introduction ordering for such a tree decomposition is definable by an MSO formula, which has one set parameter that describes the root. □

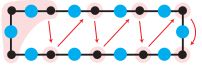
Example 47. In this example, we show that cycles have definable tree decompositions of bounded width. By a cycle, we mean a hypergraph that looks like this:



As in Example 46, we need careful with the choice of decomposition. Consider first a tree decomposition that looks like this:

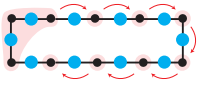


The introduction ordering for the above tree decomposition looks like this:



Note how the successor relation of this introduction ordering connects vertices which are far away in the cycle. For this reason, in order to define this introduction ordering, we would need an MSO with set parameters whose size would depend on the length of the cycle.

To get definable tree decompositions for cycles, we use tree decompositions where the introduction ordering looks like this:



The idea behind such a tree decomposition is that that nodes of the tree decomposition correspond to a clockwise traversal of the cycle, with all bags containing the first vertex (in the above picture, the first vertex is the bottom-left corner). The introduction ordering for this tree decomposition can be defined by an MSO formula with two set parameters, one to indicate the first vertex, and another one to indicate the direction (clockwise or not) of the traversal. □

Exercises

Exercise 199. Suppose that φ_1, φ_2 are two MSO formulas with set parameters, with the same number of arguments. There is a single MSO formula φ with set parameters which defines every relation definable by either φ_1 or φ_2.

Exercise 200. Define MSO_1 to be the variant of MSO where set quantification is restricted to sets of vertices (and not hyperedges). Show that for hypergraphs of bounded treewidth, MSO_1 has the same expressive power as MSO.

Exercise 201. Let $k \in \{0, 1, \ldots\}$. Show that there is an MSO formula φ with set parameters, such that for every hypergraph of treewidth at most k, every unary relation (i.e. a set of vertices and hyperedges) is definable by φ using set parameters that contain only vertices.

Exercise 202. Show that there is no MSO formula φ with set parameters, such that every hypergraph has a linear order definable by φ. Hint: consider independent sets.

Exercise 203. Show that a formula as in Exercise 202 can be found, if we want the linear order only for connected hypergraphs with degree at most k (every vertex is adjacent to at most k hyperedges).

Exercise 204. Show that a formula as in Exercise 202 can be found, if we want a spanning forest instead of a linear order.

Exercise 205. We say that a set of hypergraphs L has *bounded treedepth* if there is some ℓ such that every hypergraph in L has a tree decomposition of width and height at most ℓ (the height is the maximal depth of nodes). Without using Theorem 6.22, show that if L is recognisable and has bounded treedepth, then it is definable in counting MSO.

6.3.1 Bounded treewidth implies definable tree decompositions

We are now ready to state the main result of Section 6.3.

Theorem 6.22. *If $L \subseteq \mathsf{H}\Sigma$ has bounded treewidth, then it has definable tree decompositions of bounded width.*

The width of the tree decompositions in the assumption and in the conclusion of the above theorem will be different. An analysis of the proof would

show that if all hypergraphs in L have treewidth at most k, then the definable tree decompositions from the conclusion of the theorem will have width at most doubly exponential in k. With more care in the proof, we could produce optimal width tree decompositions[15], but the sub-optimal width will be enough for our intended application, which is the converse of Courcelle's Theorem that will be presented in Section 6.3.5.

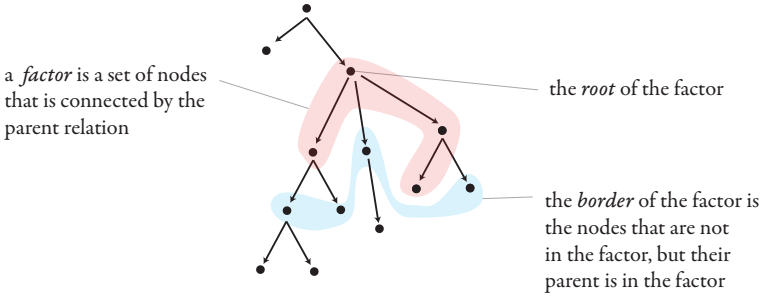
When defining tree decompositions in MSO, we will not care about the labelling relation "hyperedge e has label $a \in \Sigma$". For this reason, we will not specify the alphabet Σ for the rest of this section.

Here is a plan for the rest of Section 6.3:

- In Section 6.3.2, page 220, we state and prove the Merging Lemma, which shows how a definable tree decomposition of definable tree decompositions can be merged into a single definable tree decomposition.
- In Section 6.3.3, page 224, we prove a special case of Theorem 6.22, which says that hypergraphs of bounded pathwidth have definable tree decompositions.
- In Section 6.3.4, page 231, we complete the proof of the theorem.
- In Section 6.3.5, page 240, we apply the theorem to get a converse of Courcelle's Theorem for hypergraph of bounded treewidth.

Before proceeding with the proof, we define torsos. Torsos will be used frequently in the proof.

Torsos. Torsos are used restrict a tree decomposition to the hypergraph corresponding to some subset of the nodes. The subsets that we care about are called *factors*, and are defined in the following picture (where dots indicate nodes of a tree decomposition):



[15] Definability of tree decompositions of optimal width is shown in [12] Bojańczyk and Pilipczuk, "Optimizing Tree Decompositions in MSO", 2017 , Theorem 2.

Suppose that T is a tree decomposition and X is a factor. We define below a hypergraph, called the torso of X in T. The torso will contain vertices and hyperedges of the underlying hypergraph of T that appear in the factor, plus extra hyperedges corresponding to the border of the factor. Before defining torsos formally, we need to overcome one more difficulty. In the torso, we will need a linear ordering for its ports, and for incidence lists in the extra hyperedges (because ports are ordered, and incidence lists are also ordered). To get such linear orders, we will use a local colouring of the underlying hypergraph, as defined below.

Definition 6.23 (Local colouring)**.** Define a *local colouring* of a tree decomposition of width k to be a colouring of vertices in the underlying hypergraph with colours $\{0, \ldots, k\}$ such that in every bag, all vertices have different colours.

Every tree decomposition has a local colouring, which can be obtained in a greedy way by colouring the root bag, then colouring the bags of the children, and so on. If a tree decomposition is equipped with a local colouring, then every bag has an implicit linear order, from the smallest colour to the biggest colour. We also assume that the local colouring is chosen so that the implicit linear order is consistent with the ordering of the ports, i.e. the colours of the ports are increasing.

For a node x in a tree decomposition, define its *adhesion* to be the vertices from the bag of x that are either ports of the underlying hypergraph, or which appear also in parent of x.

Definition 6.24 (Torso)**.** Let T be a tree decomposition, together with a local colouring, and let X be a factor. The *torso of X in T*, denoted by T/X, is defined to be the following hypergraph. The vertices and hyperedges are those which are introduced in nodes from X, plus the adhesion of the root node of the factor. The ports are the adhesion factor's root, ordered according to the local colouring. Furthermore, for every node x in the border of X, we add a hyperedge (called a *border hyperedge*) whose incidence list is the adhesion of x, ordered according to the local colouring.

If a factor is a subtree, i.e. its border is empty, then the torso will have no border hyperedges. For the rest of Section 6.3, we assume that every tree decomposition comes with an implicit local colouring. This way, we can simply talking about torsos in a tree decomposition, without indicating explicitly the local colouring which is needed to define the torsos.

6.3.2 The Merging Lemma

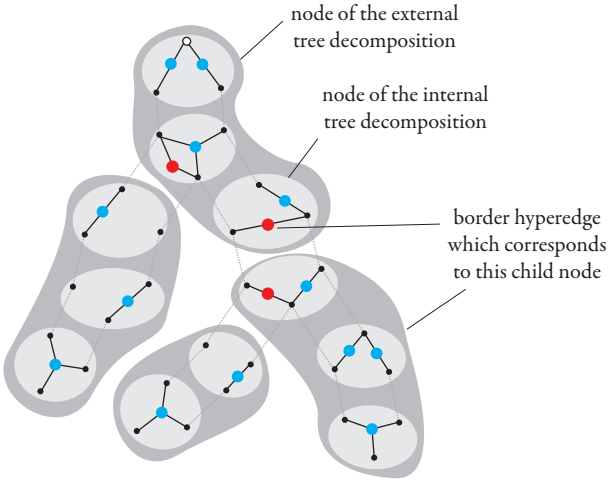
In this section we state and prove the Merging Lemma, which is based on the following simple idea. Suppose that we have an "external" tree decomposition, possibly of unbounded width, where every bag has an accompanying "internal" tree decomposition, of width at most k. We will show that these tree decompositions can be merged, in an MSO definable way, into a single tree decomposition of width at most k. This lemma will be used several times in the proof, with the internal tree decompositions typically obtained by applying some kind of induction assumption.

The internal tree decompositions are formalised using torsos for factors with one node. Torsos for larger factors will be used later, in Section 6.3.4.

Lemma 6.25 (Merging Lemma)**.** *Let $k \in \{1, 2, \ldots\}$. Let T be a tree decomposition (call it external) such that for every every node x, the torso $T/\{x\}$ has a tree decomposition T_x (call it internal) of width at most k. Then:*

(1) *The underlying hypergraph of T has a tree decomposition of width at most k.*
(2) *Suppose that ψ, φ are* MSO *formulas with set parameters, such that the external tree decomposition are definable by ψ and all internal tree decompositions are definable by φ. Then the tree decomposition from* (1) *is definable by an* MSO *formula with set parameters, which depends only on ψ, φ and k, and which does not depend on the external and internal tree decompositions.*

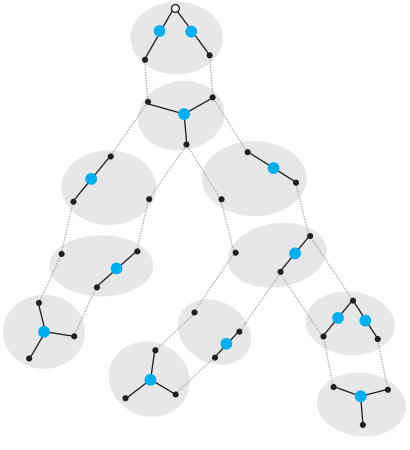
Proof Here is a picture of the external and internal tree decompositions.



In the picture above, red circles are used for border hyperedges of the torsos,

and blue circles are used for the remaining hyperedges. Note that the red border hyperedges are only present in the torsos, and not in the underlying hypergraph of the external tree decomposition.

To prove item (1), we define a *merged tree decomposition* of width k as follows. First, take the disjoint union of the internal tree decompositions, which gives a forest. Next, convert this forest into a tree, by selecting parents for roots in the following way. Suppose that x is a root node in this forest, which corresponds to the root node of some internal tree decomposition T_y. If y has a defined parent z in the external tree decomposition, then the parent of x in the merged tree decomposition is defined to be the node in the internal tree decomposition T_z that introduces the border hyperedge of the torso $T/\{z\}$ which corresponds to node y. Otherwise, if y is the root of the external tree decomposition, then x is the root of the merged tree decomposition. Here is a picture of the merged tree decomposition:



We leave it as an exercise for the reader to check that the merged tree decomposition defined this way is indeed a tree decomposition. Because bags of the merged tree decomposition are inherited from bags of the internal tree decompositions, its width is at most k, thus proving item (1).

We now prove item (2) about definability. Let G be the underlying hypergraph of the external tree decomposition. An inspection of the definition of the merged tree decomposition shows that its introduction ordering can be defined in G using MSO formulas that refer to the following relations: (a) the introduction ordering of the external tree decomposition; and (b) the following ternary relations on vertices of G that uniformly describes all of the introduction or-

derings for the internal tree decompositions:

$$I(u,v,w) \overset{\text{def}}{=} \overbrace{u \text{ is introduced in a node } x}^{\text{in the external tree decomposition}} \text{ such that } \overbrace{v \text{ is introduced before } w}^{\text{in the internal tree decomposition } T_x}.$$

The relations from (a) are definable by the assumption of (2). Therefore to prove definability of the merged tree decomposition, it remains to show definability of the ternary relation I from item (b). This is done in the remainder of this proof.

Let φ be the MSO formula with set parameters, which defines the internal tree decompositions. Suppose that φ has set parameters $Y_1, \dots, Y_\ell$. For every node x of the external tree decomposition, there is a choice of set parameters

$$Y_{1,x}, \dots, Y_{\ell,x} \subseteq \text{vertices and hyperedges in the torso } T/\{x\},$$

such that the introduction ordering in the internal tree decomposition T_x is obtained by fixing this choice of set parameters and evaluating the formula φ in the torso $T/\{x\}$. The main step in the proof will be showing how each family $\{Y_{i,x}\}_x$ can be represented in the hypergraph G using a constant size formula of MSO with set parameters.

An issue is that the set parameters might use border hyperedges, which are not present in the hypergraph G. To solve this issue, we represent border hyperedges using vertices in the following way. We say that a vertex v of G *represents* a border hyperedge e in a torso $T/\{x\}$ if: v is introduced by the external tree decomposition in a node y such that y is the child node of x that corresponds to the border hyperedge e in the torso $T/\{x\}$. Every vertex represents at most one border hyperedge, and every border hyperedge is represented by some vertex.

Let $i \in \{1, \dots, \ell\}$. Using the above representation, we will show in the following claim that the set parameters used to define the internal tree decompositions can be represented in a uniform way in the hypergraph G.

Claim 6.26. *Let $i \in \{1, \dots, \ell\}$. Each of the following relations is definable by an* MSO *formula with set parameters, which depends only on k:*

$$A_i \overbrace{(u,y)}^{\substack{u \text{ is a vertex and} \\ y \text{ is a vertex or} \\ \text{hyperedge of } G}} \overset{\text{def}}{=} \overbrace{\exists x}^{\substack{\text{node of the} \\ \text{external tree} \\ \text{decomposition}}} \quad u \text{ is introduced in } x \text{ and } y \in Y_{i,x}$$

$$B_i \underbrace{(u,v)}_{\substack{u,v \text{ are} \\ \text{vertices} \\ \text{of } G}} \overset{\text{def}}{=} \underbrace{\exists x}_{\substack{\text{node of the} \\ \text{external tree} \\ \text{decomposition}}} \quad \begin{array}{l} u \text{ is introduced in } x \text{ and the border} \\ \text{hyperedge represented by } y \text{ is in } Y_{i,x} \end{array}$$

Before proving the claim, we show how it implies definability of I, and therefore finishes the proof of the Merging Lemma. In order to check $I(u, v, w)$, we run the formula φ in the torso corresponding to the node that introduces vertex u, with calls to the i-th set parameter replaced by calls to its representation in terms of A_i and B_i. Therefore, definability of A_i and B_i implies definability of I. The remainder of the proof of the Merging Lemma is devoted to proving the claim.

Proof We begin with B_i. The key observation is that, when restricted to border hyperedges, the sets in the family $\{Y_{i,x}\}_x$ are disjoint, because each border hyperedge belongs to exactly one torso $T/\{x\}$. For this reason, B_i can be viewed as a subset of the border hyperedges, and is thus definable. More formally, B_i can be defined in MSO using a single extra set parameter, because $B_i(u, v)$ holds if and only if

$$\underbrace{\begin{array}{c}u \text{ is introduced in the node whose torso}\\ \text{contains the border hyperedge represented by } v\end{array}}_{\text{definable in MSO using the external tree decomposition}} \text{ and } v \in \underbrace{\{v : B_i(u, v) \text{ for some } u\}}_{\text{a single extra set parameter}}.$$

We now turn to A_i. Here the difficulty is that a vertex might belong to several sets from the family $\{Y_{i,x}\}_x$. This will happen for vertices from the adhesions in the external tree decomposition. To solve this difficulty, we will use a colouring of the adhesions that uses at most $k + 1$ colours.

Partition the binary relation A_i into two parts as follows:

$$\underbrace{C_i}_{\substack{\text{pairs } (u,v) \in A_i \text{ such that}\\ v \text{ is in the adhesion of}\\ \text{the node represented by } u}} \cup \underbrace{D_i}_{\substack{\text{pairs } (u,y) \in A_i \text{ such that}\\ y \text{ is a vertex or hyperedge}\\ \text{that is introduced in}\\ \text{the node represented by } u}}.$$

The part D_i does not raise any problems, since every vertex or hyperedge is introduced in exactly one node of the external tree decomposition, and therefore we can use the same argument as for B_i to prove that D_i can be defined in MSO using a single extra set parameter.

We are left with C_i. Choose a local colouring for the merged tree decomposition, which maps vertices of G to colours $\{0, \ldots, k\}$ so that no colour is used twice in a bag of the merged tree decomposition. We will show:

(*) for every u, all vertices that v that satisfy $C_i(u, v)$ have different colours.

To see why (*) is true, suppose that in the external tree decomposition, vertex v is in the adhesion of the node x that introduces u. It follows that v is a port of the torso $T/\{x\}$, and therefore it must be in the root bag of the internal tree decomposition T_x. This bag is also a bag of the merged tree decomposition,

and hence all vertices in this bag must have different colours under the local colouring, thus proving (*).

We now use (*) to finish the proof of the claim. For a colour $c \in \{0, \ldots, k\}$ and a vertex u, define $f_c(u)$ to be the vertex of colour c in the adhesion of the node that introduces u. There is at most one such vertex by (*), and hence f_c can be viewed as a partial function. The relation C_i can be defined in MSO using $k+1$ extra set parameters, because $C_i(u, v)$ holds if and only

$$\bigvee_{c \in \{0,\ldots,k\}} \underbrace{v = f_c(u)}_{\substack{\text{can be defined in MSO} \\ \text{using the external} \\ \text{tree decomposition and} \\ \text{its local colouring}}} \text{ and } u \in \underbrace{\{u : C_i(u, f_c(u))\}}_{\substack{\text{an extra set parameter} \\ \text{for each colour } c}}.$$

□

□

Exercises

Exercise 206. A cut hyperedge in a hypergraph is a hyperedge e such that for some two vertices, every path connecting them must pass through e. Let L, K be sets of hypergraphs, such that for every $G \in K$, if all cut hyperedges are removed from G, then the resulting hypergraph is in L. Show that if L has definable tree decompositions, then the same is true for K.

6.3.3 Bounded pathwidth

We now proceed to the second step in the proof of Theorem 6.22. In this step, we prove that bounded pathwidth implies definable tree decompositions of bounded width. In other words, we prove a weaker version of Theorem 6.22, where the assumption is strengthened from bounded treewidth to bounded pathwidth (recall that pathwidth is the minimal width of a path decomposition, i.e. a tree decomposition where all nodes are on a single root-to-leaf path).

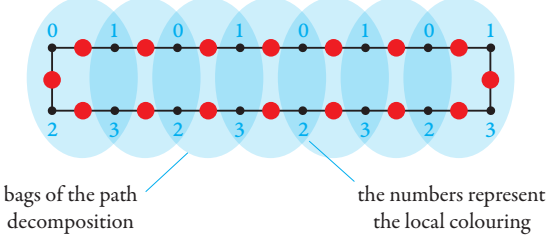
Theorem 6.27. *If a set of hypergraphs has bounded pathwidth, then it has definable tree decompositions of bounded width.*

Note the asymmetry in the above theorem: the assumption uses path decompositions but the conclusion uses tree decompositions. To see the reason

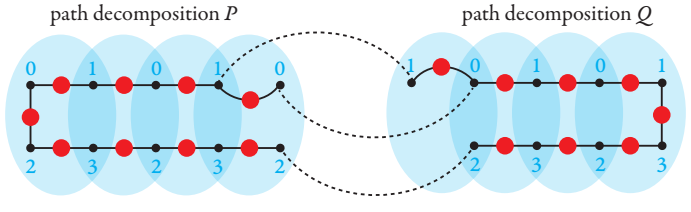
for this asymmetry, recall the independent sets from Example 45. Independent sets have pathwidth zero. Path decompositions of width zero cannot be defined in independent sets using a constant size formula of MSO, in contrast to tree decompositions of width zero.

The rest of Section 6.3.3 is devoted to proving Theorem 6.27. To find definable tree decompositions, we will view path decompositions as semigroup, and use the Factorisation Forest Theorem.

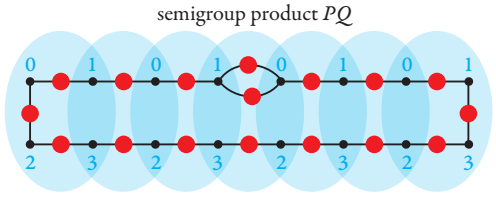
Path decompositions as a semigroup. Fix $k \in \{1, 2, \ldots\}$ for the rest of this section. As was the case for tree decompositions, we assume that every path decomposition comes together with a local colouring that uses colours $\{0, \ldots, k\}$. Here is a picture of a path decomposition together with a local colouring:



For path decompositions defined this way, we define a semigroup product, as explained in the following picture:



when taking the semigroup product, the last bag of P is fused with the first bag of Q, with the vertices being matched according to the local colouring



It is easy to see that the operation described above is associative, and hence path decompositions of width at most k are a semigroup. We call this the "semigroup of path decompositions".

The reachability homomorphism. On the semigroup of path decompositions, we define a semigroup homomorphism, which stores a finite amount of information about paths. Define an *inner path* in a hypergraph to be a path of the form

$$v_0 \xrightarrow{e_1} \underbrace{v_1 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} v_{n-1}}_{\text{these vertices are not ports}} \xrightarrow{e_n} v_n.$$

In other words, an inner path is a path that avoids ports, with the possible exception of the source and target of the path. Define the *reachability homomorphism* to be the function which maps a path decomposition of width at most k to the answers to the following questions, for all $i, j \in \{0, \ldots, k\}$ and $\sigma, \tau \in \{\text{first}, \text{last}\}$:

- does the σ bag contain a vertex with local colour i?
- is there an inner path from a vertex with local colour i in the σ bag to a vertex with local colour j in the τ bag?
- is there are vertex with local colour i that is both in the first and last bag?

The reachability homomorphism is easily seen to be compositional in the semigroup sense, and therefore it can be viewed as a semigroup homomorphism from the semigroup of path decompositions into a finite semigroup (which consists of all possible sets of answers to the questions described above).

Using the Factorisation Forest Theorem. The semigroup of path decompositions is finitely generated, namely by

$$\Delta = \text{path decompositions with at most two nodes.}$$

We view the reachability homomorphism as a homomorphism

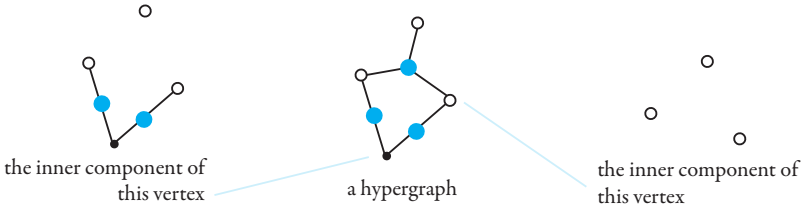
$$h : \Delta^+ \to S.$$

For $\ell \in \{0, 1, \ldots\}$, consider the path decompositions (corresponding to words in) in Δ^+ that have a Simon tree – as defined in the Factorisation Forest Theorem – of height at most ℓ, with respect to the homomorphism h. Let L_ℓ be the underlying hypergraphs of these path decompositions. By the Factorisation Forest Theorem, there is some ℓ such that all hypergraphs of pathwidth at most k belong to L_ℓ. Therefore, to prove that all hypergraphs of pathwidth at most k

have definable tree decompositions of bounded width, it remains to show the following lemma.

Lemma 6.28. *For every $\ell \in \{0, 1, \ldots\}$, L_ℓ has definable tree decompositions of bounded width.*

Proof In the proof, we work with hypergraphs where all non-ports can be connected by inner paths, as described in the following definition. For a vertex or hyperedge x in a hypergraph G, define its *inner component* to be hypergraph obtained from G by restricting to the ports, plus vertices and hyperedges that can be reached from x via an inner path. Here is a picture:



Every hypergraph is equal to the fusion of its inner components, where fusion is the operation on hypergraphs of same arity that was described in Figure 6.2. We will prove that for every $\ell \in \{0, 1, \ldots\}$, the language

$$K_\ell = \{G : G \text{ is an inner component of some hypergraph in } L_\ell\}$$

has definable tree decompositions of bounded width. This will imply the lemma, as explained in the following claim.

Claim 6.29. *Let L be a set of hypergraphs. If*

$$K = \{G : G \text{ is an inner component of some hypergraph in } L\}$$

has definable tree decompositions of bounded width, then so does L.

Proof We use the Merging Lemma to get a definable tree decomposition for every hypergraph $G \in L$. Define an external tree decomposition for G as follows. For every inner component we have a node, and these nodes are connected by a common root, whose bag is the ports. This external tree decomposition is clearly definable in MSO, even without set parameters. The torso of the root node has constant size, while the torsos of the remaining nodes have definable tree decompositions of bounded width thanks to the assumption on L. Therefore, we can apply the Merging Lemma to get a definable tree decomposition for G. □

It remains to prove that, for every $\ell \in \{0, 1, \ldots\}$, the set K_ℓ has definable tree decompositions of bounded width. The proof is by induction on ℓ.

In the induction base of $\ell = 0$, the hypergraphs from K_0 are inner components of the finitely many generators from Δ. Therefore, we can use trivial definable tree decompositions where all vertices are in the same bag.

We are left with the induction step. Consider a hypergraph $G \in K_{\ell+1}$, which by definition is an inner component of some hypergraph $H \in L_{\ell+1}$. Again by definition, H has a path decomposition which can be factorised in the semigroup of path decompositions as $P_1 \cdots P_n$, so that:

(i) $P_1, \ldots, P_n$ have their underlying hypergraphs in L_ℓ;

(ii) either $n = 2$, or all $P_1, \ldots, P_n$ have the same value under the reachability homomorphism, which is furthermore idempotent.

To define a tree decomposition for G, we use the Merging Lemma. The internal decompositions will be definable thanks to (i), and the external decomposition will be definable thanks to (ii).

The external decomposition T, which is a path decomposition[16], is defined as follows. The bags are those $x \in \{1, \ldots, n\}$ such that the path decomposition P_x contains at least one vertex of G, ordered in the natural way. The bag of node $x \in \{1, \ldots, n\}$ consists of those vertices of G that appear in P_x. The following claim takes care of the internal tree decompositions.

Claim 6.30. *For all nodes of T, the corresponding torsos come from a set of hypergraphs with definable tree decompositions of bounded width.*

Proof Let x be a node of T. The torso $T/\{x\}$ is obtained as follows: take the underlying hypergraph H_x of the path decomposition P_x, restrict it to vertices that appear in G, and then add a border hyperedge. Since every inner path of H_x is also an inner path of H, it follows that every inner component of H_x is contained in some inner component of H. Therefore, if we define G_x to be the restriction of H_x to the vertices and hyperedges of the inner component of G, then G_x is a union of inner components of H_x. By assumption, every inner component of H_x belongs to K_ℓ, and therefore has a definable tree decomposition thanks to the induction assumption. Thus G_x has a definable tree decomposition, as a fusion of hypergraphs with definable tree decompositions. Finally, adding a single border hyperedge of constant arity does not affect definable tree decompositions, since we can modify the definable tree decomposition by

[16] We are able to produce a path decompositions, and not just a tree decomposition, thanks to the assumption that G is an inner component. The place in the proof where tree decompositions are needed is the application of Claim6.29 which splits the hypergraph into inner components.

adding to all bags the vertices that are incident with the added border hyperedge. □

Thanks to the above claim and the Merging Lemma, in order to finish the proof of the lemma, it remains to show that the introduction ordering in the external tree decomposition T is definable.

If $n = 2$ then there is not much to do: the external path decomposition has two nodes, and therefore it is definable by an MSO formula with two set parameters that say which vertices are in which bags. We are left with the case where $n > 2$. The key to defining the external tree decomposition is given in the following claim. In the claim, the *profile* of a path in the hypergraph G is defined to be the nodes of the external tree decomposition that introduce at least one vertex used by the path.

Claim 6.31. *Let v and w be non-port vertices of G that are introduced, respectively, in nodes $x \leq y$ of the external tree decomposition T. If Π is the set of inner paths in G with source v and target w, then:*

(1) *every inner path in Π has profile that contains $\{x, x+1, \ldots, y-1, y\}$;*
(2) *some inner path in Π has profile contained in $\{x-1, x, \ldots, y, y+1\}$.*

Proof The proof crucially depends on the assumptions that: (a) all path decompositions $P_1, \ldots, P_n$ have the same idempotent image under the reachability homomorphism; and (b) the hypergraph G is an inner component.

(1) To prove item (1), we will show:

(*) every non-port vertex u appears in at most two bags of T.

Suppose that we have proved (*). Because bags containing u must be consecutive by definition of path decompositions, it follows that consecutive vertices in an inner path must be introduced in consecutive nodes of T. This implies that the profile of an inner path is an interval contained in $\{1, \ldots, n\}$. Such an interval must contain all numbers between x and y, if the inner path has source introduced in x and target introduced in y, thus proving (1).

It remains to prove (*). Toward a contradiction, suppose that u appears in at least three bags of the external tree decomposition T, which means that it appears in at least three nodes of the path decompositions $P_1, \ldots, P_n$. If we take $x \in \{1, \ldots, n\}$ to be the second node where u appears, then u appears in the path decompositions both before and after P_x. From the way that path decompositions are composed in the semigroup of path decompositions, it follows that u must appear in both the first and the last bags of the path decomposition P_x. Therefore, the reachability homomorphism gives a positive answer to the question "is there are vertex with local colour c that

is both in the first and last bag?", where c is the colour of u under the local colouring corresponding to the path decomposition P. Since all of the path decompositions $P_1, \ldots, P_n$ have the same abstraction under the reachability homomorphisms, a positive answer to this question is given for all of these path decompositions, and therefore u appears in all of them. This means that u is a port, contradicting the assumption of (*).

(2) Consider the following partition of the path decomposition $P_1 \cdots P_n$:

$$\underbrace{P_1 \cdots P_{x-1}}_{\text{left part}} \underbrace{P_x \cdots P_y}_{\text{middle part}} \underbrace{P_{x+1} \cdots P_n}_{\text{right part}}$$

By the assumption, all of the path decompositions $P_1, \ldots, P_n$ have the same image under the reachability homomorphism, and this is also the same as the image of the left, right and middle parts described above. Consider now an inner path in G that connects vertices v and w, which must exist because G is a single inner component. By the previous item, this path can be split into segments of three kinds: (a) paths in the left part that begin and end in its last bag; (b) paths in middle part; (c) paths in the right part that begin and end its first bag. Each of these segments can be modified, without changing its source or target, so that its profile is contained in $\{x-1, x \ldots, y, y+1\}$. For segments of the kind (b), there is nothing to do. For segments of kind (a), we use the fact that P_{x-1} has the same image under the reachability homomorphisms as the left part. A similar argument holds for segments of kind (c).

□

Using the above claim, we define in MSO the introduction ordering of the external tree decomposition, thus completing the proof of the lemma. For a vertex of G that is introduced in node $x \in \{1, \ldots, n\}$ of the external tree decomposition, define its *colour* to be x modulo 5. The colours of vertices can be represented using 5 set parameters. We begin with the following observation, which implies that the relation "introduced in the same node of the external tree decomposition" is definable by a constant MSO formula with set parameters:

(0) two vertices v and w are introduced in the same node of the external tree decomposition if and only if they have the same colour, and they can be connected by an inner path that does not use all colours.

The right-to-left implication in (0) follows from the first item in Claim 6.31, since vertices introduced in the same node can be connected by a path that uses at most three colours. The left-to-right implication of (0) follows from the second item.

A similar argument proves the following characterisation of the successor relation on nodes in the external tree decomposition:

(1) the node introducing vertex w is the successor of the node introducing v if and only if the colour of w is one plus the colour of v (modulo 5), and there is a path connecting them that does not use all colours.

Again, the left-to-right implication uses the first item of Claim 6.31, because vertices introduced in nodes x and $x+1$ can be connected using a path with at most 4 colours. For the right-to-left implication, we use the second item of Claim 6.31, and the following observation, which explains the need for counting modulo 5:

$$\underbrace{y \equiv x+1}_{\text{modulo 5}} \qquad \text{implies} \qquad y = x+1 \quad \text{or} \quad \underbrace{y \leq x-4 \quad \text{or} \quad y \geq x+6}_{\text{any connecting path must use all 5 colours}}.$$

The introduction ordering of the external tree decomposition is the transitive closure of the union of the two relations defined in (0) and (1), and therefore it is definable. □

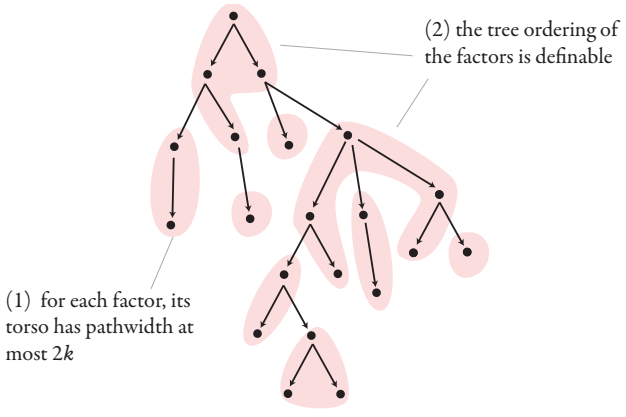
Exercises

Exercise 207. We say that a hypergraph is a tree if removing any hyperedge increases the number of connected components. Show that trees have unbounded pathwidth.

Exercise 208. Give an algorithm which inputs a tree hypergraph, and computes its pathwidth.

6.3.4 Unbounded pathwidth

In this section, we complete the proof of Theorem 6.22, using the Merging Lemma and the case of bounded pathwidth shown above. The idea is to show that for every tree decomposition of width at most k, its nodes can be partitioned into factors in a way that is depicted in the following picture:



Given such a factorisation, we will apply the Merging Lemma, with the external tree decomposition having the factors as nodes, and with the internal tree decompositions being definable thanks to the results on bounded pathwidth.

To define the tree ordering on the factors from item (2) in the picture, we will use paths with bounded overlap that connect the factors. Paths with bounded overlap, and their application to MSO definability, are explained in the following lemma.

Lemma 6.32 (Bounded Overlap Lemma)**.** *Let $\mathcal{P}$ be a family of paths in a hypergraph G such that every vertex is used by at most $\ell \in \{1, 2, \ldots\}$ paths. Then the binary relation*

$$\{(s,t) : s,t \text{ are vertices such that some path in } \mathcal{P} \text{ has source } s \text{ and target } t\}$$

is definable by an MSO *formula with set parameters that depends only on ℓ and the maximal arity of hyperedges, and not on the hypergraph G or the family of paths $\mathcal{P}$.*

Proof Choose a family of colourings

$$\underbrace{\text{vertices used by } P \quad \to \quad \{1, \ldots, \ell\}}_{\text{one colouring for each path } P \in \mathcal{P}},$$

so that if a vertex appears in two different paths from $\mathcal{P}$, then it has different colours under the corresponding colourings. Such a family of colourings can easily be obtained using a greedy algorithm, thanks to the assumption that every vertex is used by at most ℓ paths. Define $\to$ to be the binary relation on pairs of the form (vertex of G, number in $\{1, \ldots, \ell\}$), such that

$$(v,i) \to (w,j) \tag{6.1}$$

holds if there is some path $P \in \mathcal{P}$ where v and w are consecutive vertices which have have colours i and j under the colouring corresponding to P. A vertex pair (s,t) belongs to the relation in the statement of the lemma if and only if there exist colours $i, j \in \{1, \ldots, \ell\}$ such that: (s,i) can reach (t,j) using finitely many steps of $\to$, (s,i) has no incoming edges with respect to $\to$, and (t,j) has no outgoing edges with respect to $\to$. Therefore, to prove the lemma, it will be enough to show that both $\to$ and its transitive closure can be defined in MSO, as described below.

For every fixed choice of colours i and j, we can view (6.1) as a binary relation on vertices. This relation can be described in MSO using set parameters that range over sets of hyperedges, as described below:

$$\bigvee_{n,m} \quad \exists e \; e \in \underbrace{E_{i,j,n,m}}_{\substack{\text{set of hyperedges } e \text{ such that} \\ \text{for some path } P \in \mathcal{P}, \\ P \text{ uses a step of the form} \\ e[n] \xrightarrow{e} e[m] \\ \text{and the colouring of } P \text{ satisfies} \\ e[n] \mapsto i \text{ and } e[m] \mapsto j.}} \wedge \, e[n] = v \; \wedge \; e[m] = w.$$

In the above, n and m range over positions in incidence lists, and therefore these numbers are bounded by the maximal arity of hyperedges. Consider now the transitive reflexive closure

$$(v,i) \to^* (w,j) \tag{6.2}$$

of the relation $\to$. To define this transitive closure, we use the usual MSO formalisation of least fix-points. For every choice of colours i and j, the relation (6.2) can be defined using MSO in terms of the relations from (6.1), as follows:

$$\forall \overbrace{V_1, \ldots, V_\ell}^{\substack{\text{sets of} \\ \text{vertices}}} \; \overbrace{(v \in V_i \wedge \bigwedge_{n,m} \forall x \, (x \in V_n \wedge (x,n) \to (y,m)) \Rightarrow y \in V_m) \Rightarrow v \in V_j}^{\text{if the set } \{(x,i) : x \in V_i\} \text{ contains } (v,i) \text{ and is closed under } \to, \text{ then it contains } (w,j)}$$

It follows that the transitive closure in (6.2) can be defined using a constant formula of MSO with set parameters, for every choice of $i, j \in \{1, \ldots, \ell\}$. □

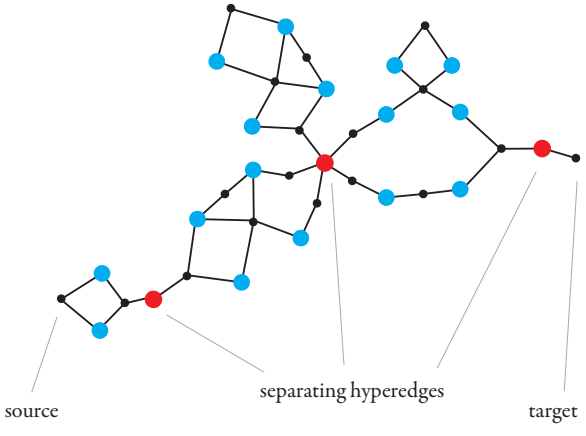
The Two Path Lemma. The next step in the proof is the Two Path Lemma, which will provide the paths with small overlap that are needed to connect factors in a tree decomposition. The Two Path Lemma says that for every tree decomposition of width at most k, and every choice of source and target vertices from the same bag in the tree decomposition, one can find a factor of the tree decomposition, such that the corresponding torso has bounded pathwidth and contains two roughly disjoint paths connecting the source and target.

The Two Path Lemma uses a mild assumption on tree decompositions, defined as follows. We say that a tree decomposition is *sane* if for every subtree, the corresponding torso is inner connected, which means that it consists of a single inner component. If a hypergraph is inner connected, then it has a sane tree decomposition of optimal width. Indeed, if we take any tree decomposition, and we find a subtree whose torso is not inner connected, then we can distribute vertices introduced by the subtree into separate subtrees, one for each inner component.

Lemma 6.33 (Two Path Lemma)**.** *Let T be a sane tree decomposition of width k, with two distinguished vertices (call them source and target) that belong to the bag of some node x. Then there exists a factor X with root x such that:*

(1) *the torso T/X has pathwidth at most $2k$;*
(2) *the source and target can be connected by two inner paths in the torso T/X, such that every border hyperedge of T/X is used by at most one of the paths.*

Proof To create the two paths from the conclusion of the lemma, we use (a hypergraph version of) the Menger's Theorem about cuts and disjoint paths, as described below. For a hypergraph with distinguished source and target vertices, define a *separating hyperedge* to be any hyperedge that must appear on every path from the source to the target. One could also talk about separating vertices, but we only need separating hyperedges here. Here is a picture:



Menger's Theorem[17] says that if the source and target in a hypergraph can be

[17] The hypergraph version of Menger's Theorem that we use here can be easily inferred from the classical statement of undirected graphs, which can be found here:
[23] Diestel, *Graph theory (electronic edition)*, 2006 , Theorem 3.3.3.
For a self-contained proof, see the exercises.

connected by some path, then they can be connected by two paths such that every non-separating hyperedge is used by at most one of the two paths.

To prove the lemma, we iterate the following claim starting with $X = \{x\}$. In the claim, use an operation "forget all ports" which inputs a hypergraph, and outputs the same hypergraph, except that no vertices are ports any more. If we forget all ports in a torso, then the accompanying path decompositions do not need to have the adhesion of the torso in the first bag.

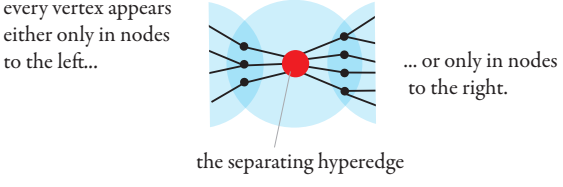
Claim 6.34. *Let X be a factor of the tree decomposition T such that:*

(*) *the root of X is x and there is a path decomposition of*

$$\textit{forget all ports in } T/X$$

which has width at most k and satisfies the following property:

(**) *the source and target vertices from the assumption lemma are, respectively, in the first and last bags, and every separating hyperedge is covered by some node such that:*



Then either T/X has no border hyperedges which separate the source from the target, or otherwise one can add a new node to X so that it still satisfies (*).

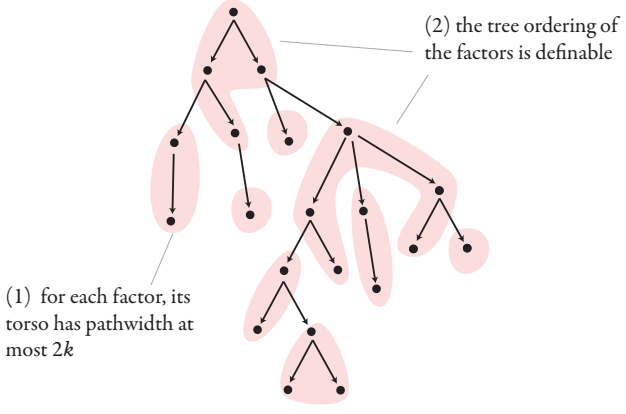
Proof Suppose that there is a border hyperedge e in the torso T/X which separates the source from the target. This border hyperedge corresponds to some node y of the tree decomposition that is in the border of X. We add y to the factor X, while preserving the invariant (*). Take the path decomposition from the assumption that X satisfies (*). By (**), there is a node z in this path decomposition that covers the separating hyperedge e, and such that every vertex of the torso T/X appears either only to the left, or only to the right of z. Replace node z by the torso $T/\{y\}$, with the separating hyperedges put into separate bags so that (**) is still satisfied. □

Start with $\{x\}$ and keep iterating the above claim, until a factor X is reached such that T/X has no border hyperedge that separates the source from the target. Take the path decomposition from the claim and, if necessary, add the ports of the torso to a prefix the path decomposition so that they are also present in

the first bag (as required for path decompositions). This modification can make the width of the path decomposition go up from k to $2k$.

Because the tree decomposition is sane, the torso T/X contains at least one path that connects the source with the target. Therefore, thanks to Menger's Theorem there are two paths from the source to the target in T/X which are disjoint on border hyperedges. □

Partitioning a tree decomposition into factors of bounded pathwidth. To finish the proof of Theorem 6.22, we will iterate the Two Path Lemma to show that every sane tree decomposition can be partitioned into factors, so that each factor has a torso with bounded pathwidth, and the partition into factors can be defined by a constant MSO formula with set parameters. The picture is the same as in the beginning of the section, but we repeat it for the reader's convenience:



Lemma 6.35. *Let T be a sane tree decomposition of width k. There is a family of factors $\mathcal{X}$ which partitions the nodes of T such that:*

(1) *For every factor $X \in \mathcal{X}$, the torso T/X has pathwidth at most $2k$.*
(2) *There is an* MSO *formula with set parameters, which only depends on k, and which defines the following binary relation on vertices:*

$$\text{factor from } \mathcal{X} \text{ that introduces } v \quad \underbrace{\leq}_{\text{tree ordering on } \mathcal{X} \text{ inherited from } T} \quad \text{factor from } \mathcal{X} \text{ that introduces } w.$$

Before proving the lemma, we use it to complete the proof of Theorem 6.22. By Claim 6.29, it is enough to show that for every $k \in \{0, 1, \ldots\}$, the inner connected hypergraphs of treewidth at most k have definable tree decompositions of bounded width. If a hypergraph is inner connected, then it has a sane tree decomposition of optimal width. Apply Lemma 6.35 to this tree decomposition, yielding a family of factors $\mathcal{X}$. Define a new tree decomposition, where

the nodes are the factors from $\mathcal{X}$, with the tree ordering inherited from the original tree decomposition. By item (2) of the lemma, the new tree decomposition is definable, and by item (1) of the lemma, each node has a torso of pathwidth at most $2k$. Therefore, we can apply the Merging Lemma to get a definable tree decomposition of bounded width, with the internal tree decompositions coming from Theorem 6.27 about bounded pathwidth.

It remains to prove the lemma.

Proof We iterate the Two Path Lemma, to get the family of factors that satisfy item (1), together with a family of paths with bounded overlap that will be used to ensure item (2). One step of the iteration is described in the following claim.

Claim 6.36. *Suppose that $\mathcal{X}$ is a family of disjoint factors in T which satisfies:*

(*) *the union $\bigcup \mathcal{X}$ is a prefix of the tree decomposition, i.e. it is closed under ancestors, and:*

(1) *for every factor $X \in \mathcal{X}$, the torso T/X has pathwidth at most $2k$;*
(2) *there is a family of paths $\mathcal{P}$ in the torso $T/(\bigcup \mathcal{X})$ such that:*
   (i) *for every factor $X \in \mathcal{X}$ there is some vertex u introduced in X such that for every vertex v in the root bag of X there is a path from u to v in $\mathcal{P}$;*
   (ii) *every vertex of $T/(\bigcup \mathcal{X})$ is used by at most $2k^3 + k$ paths from $\mathcal{P}$;*
   (iii) *every border hyperedge of $T/(\bigcup \mathcal{X})$ is used by at most $2k^3$ paths from $\mathcal{P}$.*

Then either $\bigcup \mathcal{X}$ is all nodes of the tree decomposition, or otherwise one can add a new factor to $\mathcal{X}$ so that the resulting family still satisfies (*).

Proof Suppose that the prefix $\bigcup \mathcal{X}$ is not all nodes in the tree decomposition. Choose some minimal node x outside this prefix; this node corresponds to a border hyperedge e in the torso $T/(\bigcup \mathcal{X})$. By definition of torsos, the vertices incident to this border hyperedge are the adhesion of x. The adhesion has size at most k, since bags have size at most $k + 1$ and the adhesion of a non-root node is a proper subset of its bag (otherwise a node would have the same bag as its parent). For two vertices v, w from the adhesion of x, we say that a path $P \in \mathcal{P}$ has *profile* (v, w) if it contains an infix of the form

$$v \xrightarrow{e} w.$$

There are less than k^2 choices for v, w. In general a path might have several profiles if it uses the hyperedge e several times. However, by eliminating loops, we can assume without affecting the invariant (*) that every path from $\mathcal{P}$ uses each hyperedge at most once, and therefore every path has at most one profile. Choose a pair of vertices (s, t) in the adhesion of y so that the number of paths with this profile is maximal; let ℓ be the number of such paths. Apply the Two

Path Lemma with s being the source and t being the target, yielding a factor X in T with root node x. To prove the claim, we will show that the invariant (*) is still satisfied after adding X to the family $\mathcal{X}$.

Clearly item (1) in the invariant is satisfied, because T/X has pathwidth at most $2k$ thanks to the Two Path Lemma. It remains to find a family of paths, call it $\mathcal{R}$, which witnesses item (2) of the invariant. This family is constructed as follows:

(a) Choose a vertex u that is introduced in x. For every vertex in the bag of x, choose an inner path in the torso T/X which goes from that vertex to u, and add this path to $\mathcal{R}$. Such a path must exist because the tree decomposition T is sane. The paths added in this step ensure that item (i) of the invariant will hold. In the remaining steps, we need to take care of the border hyperedge e which is used by the paths from $\mathcal{P}$. This hyperedge will not be allowed in $\mathcal{R}$ since it is removed from the torso once we add the factor X to $\mathcal{X}$.

(b) Consider a path in $\mathcal{P}$ which either does not use the border hyperedge e, or uses it with a profile (v, w) that is different from (s, t). If e is used by this path, then replace the corresponding segment

$$v \xrightarrow{e} w$$

with some inner path in T/X that goes from v to w (which exists because the tree decomposition is sane). Then, add the resulting path to $\mathcal{R}$.

(c) Consider a path in $\mathcal{P}$ which uses the hyperedge e with profile (s, t). We do a similar procedure as in the previous item, except that we use two inner paths instead of one. Let P_1 and P_2 be the two inner paths in the torso T/X from the conclusion of the Two Path Lemma. Partition the ℓ paths in $\mathcal{P}$ which have profile (s, t) into two groups, of sizes $\lceil \ell/2 \rceil$ and $\lfloor \ell/2 \rfloor$. For every path in the first group, replace the segment

$$s \xrightarrow{e} t$$

with the inner path P_1, and for every path in the second group replace this segment with the inner path P_2. Add the resulting paths to $\mathcal{R}$. By construction, if we look at the paths added in this step, then every border hyperedge of T/X is visited by paths from at most one of the two groups, and therefore by at most $\lceil \ell/2 \rceil$ paths.

We now argue that the family $\mathcal{X} \cup \{X\}$ satisfies item (2) of the invariant, with respect to the family $\mathcal{R}$ defined above.

(i) This part of the invariant is satisfied thanks to the paths that are described in item (a) of the definition of $\mathcal{R}$.

(ii) This part of the invariant says that every vertex from the torso of $\bigcup(\mathcal{X} \cup \{X\})$ is used by at most $2k^3 + k$ paths from $\mathcal{R}$. Consider first a vertex v that appears already in $\bigcup \mathcal{X}$. Because, in the definition of $\mathcal{R}$, we used inner paths from T/X to replace traversals of the border hyperedge e, it follows that

$$\text{number of paths in } \mathcal{R} \text{ that uses } v = \underbrace{\text{number of paths in } \mathcal{P} \text{ that uses } v}_{\text{at most } 2k^3 + k \text{ by assumption}}.$$

Consider now a vertex that is introduced in X. This vertex is used by at most

$$\underbrace{2k^3}_{\substack{\text{paths from } \mathcal{P} \\ \text{that used } e}} + \underbrace{k}_{\substack{\text{paths added} \\ \text{in item (a)}}}$$

paths and therefore the invariant is also satisfied.

(iii) This part of the invariant says that every border hyperedge in the torso of $\bigcup(\mathcal{X} \cup \{X\})$ is used by at most $2k^3$ paths from $\mathcal{R}$. If this hyperedge is not a border hyperedge of T/X, then we use the same argument as for vertices. Otherwise, the number of paths in $\mathcal{R}$ that use this border hyperedge is at most

$$\underbrace{\text{number of paths in } \mathcal{P} \text{ that do not have profile } (s,t)}_{\substack{\text{paths added} \\ \text{in item (b)}}} + \underbrace{\lceil \ell/2 \rceil}_{\substack{\text{paths added} \\ \text{in item (c)}}} + \underbrace{k.}_{\substack{\text{paths added} \\ \text{in item (a)}}}$$

The above sum is at most $2k^3$, which is argued by considering two cases. If $\ell < 2k$, then the summands for items (b) and (c) are at most $k^2(2k-1)$, and adding k will not exceed the threshold $2k^3$. Otherwise, if $\ell \geq 2k$, then

$$\lceil \ell/2 \rceil + k \leq \ell$$

and therefore the sum of all items at most the number of paths in $\mathcal{P}$ which uses the border hyperedge e.

□

Iterate the above claim, starting with the empty family, until reaching a family of factors $\mathcal{X}$ which satisfies the invariant (*), and which partitions all nodes of the tree decomposition. We will show that $\mathcal{X}$ satisfies the conclusion of the lemma. Item (1) of the lemma, about bounded pathwidth of the torsos, follows immediately from the corresponding item in the claim. We are left with proving item (2) from the lemma, about definability of the binary relation

$$\text{factor from } \mathcal{X} \text{ that introduces } v \quad \underbrace{\leq}_{\text{tree ordering on } X \text{ inherited from } T} \quad \text{factor from } \mathcal{X} \text{ that introduces } w.$$

Here we will use the family of paths $\mathcal{P}$ from item (2) in the invariant (*). By

item (2)(i) of the invariant, for every factor $X \in \mathcal{X}$ we can choose a vertex v_X that is introduced in X, such that using paths from $\mathcal{P}$ one can connect v_X with every vertex in the root bag of X. By the bounded overlap condition for vertices from from item (2)(ii) of the invariant, we can use the Bounded Overlap Lemma to show that the binary relation

$$\{(v_X, w) : X \in \mathcal{X} \text{ and } w \text{ is in the adhesion of the root node of } X\} \tag{6.3}$$

can be defined by an MSO formula with set parameters that depends only on k. (When applying the Bounded Overlap Lemma, we observe that in a hypergraph with treewidth at most k, all hyperedges have arity at most $k + 1$.) Let X be a factor in $\mathcal{X}$, with root node x. By definition of sane tree decompositions, the torso of the subtree of x in T is inner connected, which means that every two vertices from this torso that are not ports (i.e. they are not in the adhesion of x) can be connected by an inner path. Since v_X is not in the adhesion of x, it follows that a vertex w is introduced in the subtree of x if and only if there is a path from w to v_X which does not use vertices from the adhesion of x. Since this can be formalised in MSO using the relation (6.3), we can define in MSO the following binary relation:

$$\{(v_X, w) : X \in \mathcal{X} \text{ and } w \text{ is introduced in root node of } X \text{ or its descendants}\}.$$

Using the above relation, we can easily define in MSO the binary relation from item (2) in the statement of the lemma, thus finishing the proof. □

Exercises

Exercise 209. Prove Menger's Theorem, in the hypergraph version that is used in the proof of the Two Path Lemma.

6.3.5 Application to recognisability

A corollary of Theorem 6.22 is that the converse of Courcelle's Theorem holds for hypergraphs of bounded treewidth, as stated below.

Corollary 6.37. *Let $L \subseteq \mathsf{H}\Sigma$ be a set of hypergraphs of bounded treewidth. Then L is recognisable if and only if it is definable in counting* MSO.

Proof The right-to-left implication, even without the assumption on bounded treewidth, is Courcelle's Theorem.

The left-to-right implication is proved using the definable tree decompositions from Theorem 6.22. The formula defining the language L guesses the tree decomposition, by existentially quantifying over the set parameters needed to define its introduction order adjacent bag relation, and then does a bottom-up pass through the decomposition to compute the value of the hypergraph with respect to the recognising homomorphism. Suppose that $L \subseteq \mathsf{H}\Sigma$ has bounded treewidth and is recognised by a homomorphism

$$h : \mathsf{H}\Sigma \to A$$

into a hypergraph algebra that is finite on every arity. We need to show that L is definable in counting MSO. By Theorem 6.22 there is some $k \in \{0, 1, \ldots\}$ and an MSO formula φ with set parameters, such that every hypergraph in L has a tree decomposition T of width at most k that can be defined by φ.

The formula of counting MSO that defines the language L works as follows. Let $G \in L$ and let T be a tree tree decomposition which can be defined by some choice of set parameters in the formula φ. For a node x of this tree decomposition, define G_x to be the torso of the subtree of x, and define the *type* of x to be the image of G_x under the recognising homomorphism h. The formula defining L uses existential set quantification to guess the following sets:

(a) set parameters for the formula φ which define the introduction order of T;
(b) set parameters for the formula φ which define the bag relation of T;
(c) $k + 1$ sets which represent a local colouring;
(d) sets which represent the types of the nodes in the tree decomposition.

The types from item (d) are represented by storing the type of node x in every vertex v of the underlying hypergraph that is introduced in node x. Note that the arity of a type is at most $k + 1$, since each torso has at most $k + 1$ ports, and therefore there are finitely many possibilities for the types, which ensures that they can be represented a bounded number of sets (number of elements in the algebra that have arity at most $k + 1$). Next, the formula checks that the sets guessed in (a, b, c) indeed describe a tree decomposition of width at most k together with a local colouring; this can be easily done by expressing the appropriate definitions in MSO. The rest of this proof is devoted to explaining how a formula of counting MSO can check that the types from item (d) were guessed correctly. Once we know this, membership in the language boils down to checking that the type of the root node in the tree decomposition belongs to the accepting set for the homomorphism.

To check that the types from item (d) are guessed correctly, we examine every node x of the tree decomposition, and check if its guessed type is consistent

with the guessed types of its children. This is done as follows. For a node x in the tree decomposition, decompose the torso G_x into the following parts:

- For a hyperedge e that is introduced in x, define E_e to be the hypergraph obtained from G_x by restricting it so that (a) the vertices are the bag of x; (b) the only hyperedge is e; (c) all vertices are ports.
- For a child y node of x, define H_y to be the hypergraph obtained from G_x by restricting it so that: (a) the vertices are the bag of x plus vertices introduced in y and its descendants of y; (b) the hyperedges are those that are introduced in y and its descendants; (c) the ports are the bag of x.

Recall the fusion, forget and rearrangement operations from Figure 6.2. Apart from the ports, the parts E_e and H_y defined above are disjoint, and together they represent all vertices and hyperedges in the torso G_x. Therefore, G_x can be recovered from these hypergraphs by fusing these and then forgetting the vertices from the bag of x which are not in the adhesion of the subtree of x, as expressed in the following equality:

$$G_x = \overbrace{f(\underbrace{\sum_y H_y}_{\substack{\text{fusion ranging}\\ \text{over children of } x}} \underbrace{+}_{\text{fusion}} \underbrace{\sum_e E_e}_{\substack{\text{fusion ranging}\\ \text{over hyperedges}\\ \text{introduced in } x}})}^{\substack{\text{the operation } f \text{ restricts the ports}\\ \text{to the adhesion of the subtree of } x}}.$$

Although fusion is in principle a binary operation, it is associative and commutative when the number of ports is fixed, and therefore unordered sums in the above expression are meaningful. Both fusion and f are special cases of term operations in hypergraph algebra, which means that they commute with the homomorphism h.

The hypergraph H_y is obtained from G_y by forgetting some ports (namely the ports that are not in the bag of x), and then adding some new vertices as ports (namely the vertices that are in the bag of x but not in the bag of y). This corresponds to a term operation, call it f_y, which transforms H_y into G_y is a term operation. Summing up, we have the following equation for every node x in the tree decomposition:

$$G_x = f(\sum_y \underbrace{f_y(G_y)}_{H_y} + \sum_e E_e).$$

The above equation is in the free algebra $\mathsf{H}\Sigma$. Since h is a homomorphism, and homomorphisms commute with term operations such as fusion and f_y, we have

the following equation in the algebra A:

$$\underbrace{h(G_x)}_{\text{type of } x} = f(\sum_y f_y(\underbrace{h(G_y)}_{\text{type of } y}) + \sum_e h(H_e)), \tag{6.4}$$

where both fusion and the term operations f and f_y are now interpreted in the algebra A. If the guessed types satisfy the equality in (6.4) for every node x, then they are equal to the actual types. Therefore, it remains to show that a formula of counting MSO can check if the guessed types satisfy the equality (6.4) for every node x.

Define A^x and A_x to be the elements of the algebra A whose arity is equal to sizes of, respectively, the adhesion and the bag of x. The term operations in (6.4) have the following types:

$$\underbrace{A_x \to A^x}_{f} \qquad \underbrace{A_x \times A_x \to A_x}_{+} \qquad \underbrace{A^y \to A_x}_{f_y}.$$

Since fusion + is associative and commutative, it follows that the fusion from (6.4) corresponds to multiplication in a finite commutative semigroup, and such multiplication can be computed in counting MSO. The result follows, since the term operation f_y can be determined in MSO based on the node y (given by a vertex introduced in it), and the same is true for $h(H_e)$. □

Exercises

Exercise 210. We say that a hypergraph algebra A is *aperiodic* if for every $n \in \{0, 1, \ldots\}$ the semigroup

$$(\text{elements of } A \text{ with arity } n, \text{fusion})$$

is aperiodic. Show that if L is a set of hypergraphs of bounded treewidth, then L is definable in MSO without counting if and only if it is recognised by a finite aperiodic hypergraph algebra.

Exercise 211. Suppose that L has bounded treedepth, as described in Exercise 205. Show that if L is recognised by an aperiodic hypergraph algebra, then it is definable in first-order logic.

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] Michael Benedikt and Luc Segoufin. "Regular tree languages definable in FO and in FO_{mod}". In: *ACM Trans. Comput. Log.* 11.1 (2009), 4:1–4:32.

[4] Garrett Birkhoff. "On the structure of abstract algebras". In: *Proc. Cambridge Phil. Soc.* 31 (1935), pp. 433–454.

[5] Stephen L. Bloom and Zoltán Ésik. "The equational theory of regular words". In: *Information and Computation* 197.1 (2005), pp. 55 –89.

[6] Achim Blumensath. "Regular Tree Algebras". In: *CoRR* abs/1808.03559 (2018).

[7] Mikołaj Bojańczyk. "Recognisable languages over monads". In: *CoRR* abs/1502.04898 (2015).

[8] Mikołaj Bojańczyk. "Two Monads for Graphs". In: *CoRR* abs/1804.09408 (2018).

[9] Mikołaj Bojańczyk and Wojciech Czerwiński. *An Automata Toolbox*. 2018. URL: `https://www.mimuw.edu.pl/~bojan/upload/reduced-may-25.pdf`.

[10] 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).

[11] Mikołaj Bojańczyk and Michal Pilipczuk. "Definability Equals Recognizability for Graphs of Bounded Treewidth". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*

'16, New York, NY, USA, July 5-8, 2016. Ed. by Martin Grohe, Eric Koskinen, and Natarajan Shankar. ACM, 2016, pp. 407–416.

[12] Mikołaj Bojańczyk and Michal Pilipczuk. "Optimizing Tree Decompositions in MSO". In: *Symposium on Theoretical Aspects of Computer Science, STACS, Hannover, Germany*. Ed. by Heribert Vollmer and Brigitte Vallée. Vol. 66. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 15:1–15:13.

[13] Mikołaj Bojańczyk and Igor Walukiewicz. "Forest Algebras". In: *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*. Ed. by Jörg Flum, Erich Grädel, and Thomas Wilke. Vol. 2. Texts in Logic and Games. Amsterdam University Press, 2008, pp. 107–132.

[14] 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.

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

[16] 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.

[17] Julia Chuzhoy and Zihan Tan. "Towards Tight(er) Bounds for the Excluded Grid Theorem". In: *Symposium on Discrete Algorithms, (SODA), San Diego, USA*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 1445–1464.

[18] Joëlle Cohen, Dominique Perrin, and Jean-Eric Pin. "On the expressive power of temporal logic". In: *J. Comput. System Sci.* 46.3 (1993), pp. 271–294.

[19] 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.

[20] Bruno Courcelle. "The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs". In: *Information and Computation* 85.1 (Mar. 1990), pp. 12–75.

[21] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Vol. 138. Encyclopedia of Mathematics and Its Applications. Cambridge University Press, 2012.

[22] Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. "Efficient Separability of Regular Languages by Subsequences and Suffixes". In: *International Colloquium on Automata, Languages and Programming, ICALP, Riga, Latvia*. Ed. by Fedor V. Fomin et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 150–161.

[23] Reinhard Diestel. *Graph theory (electronic edition)*. Vol. 173. Graduate texts in mathematics. Springer-Verlag, 2006.

[24] Paul Dubreil. "Contribution à la théorie des demi-groupes. I". In: *Mém. Acad. Sci. Inst. France* 61 (1941), pp. 1–52.

[25] Samuel Eilenberg. *Automata, languages, and machines. Vol. B*. New York: Academic Press [Harcourt Brace Jovanovich Publishers], 1976.

[26] Samuel Eilenberg and Marcel-Paul Schützenberger. *On pseudovarieties*. IRIA. Laboratoire de Recherche en Informatique et Automatique, 1975.

[27] Samuel Eilenberg and Jesse Wright. "Automata in General Algebras". In: *Information and Control* 11 (1967), pp. 452–470.

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

[29] Kousha Etessami and Thomas Wilke. "An Until Hierarchy and Other Applications of an Ehrenfeucht-Fraïssé Game for Temporal Logic". In: *Inf. Comput.* 160.1-2 (2000), pp. 88–108.

[30] 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.

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

[32] J. A. Green. "On the Structure of Semigroups". In: *Annals of Mathematics* 54.1 (1951), pp. 163–172.

[33] 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.

[34] Jörg Flum Heinz-Dieter Ebbinghaus. *Finite Model Theory*. 2nd. Springer Monographs in Mathematics. Springer, 2006.

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

[36] 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.

[37] H Läuchli and J Leonard. “On the elementary theory of linear order”. In: *Fundamenta Mathematicae* 59.1 (1966), pp. 109–116.

[38] Robert McNaughton. “Testing and generating infinite sequences by a finite automaton”. In: *Information and Control* 9 (1966), pp. 521–530.

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

[40] Edward F Moore. “Gedanken-experiments on sequential machines”. In: *Automata studies* 34 (1956), pp. 129–153.

[41] Dominique Perrin and Jean-Eric Pin. “Semigroups and automata on infinite words”. In: *Semigroups, formal languages and groups (York, 1993)*. Dordrecht: Kluwer Acad. Publ., 1995, pp. 49–72.

[42] J.-E. Pin and P. Weil. “Polynomial closure and unambiguous product”. In: *Theory Comput. Syst.* 30.4 (1997), pp. 383–422.

[43] 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.

[44] Frank D. Ramsey. “On a problem of formal logic”. In: *Proc. of the London Math. Soc.* 30 (1929), pp. 338–384.

[45] 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.

[46] Shmuel Safra. “On the complexity of the ω-automata”. In: *Proc. 29th Ann. IEEE Symp. on Foundations of Computer Science*. IEEE, 1988, pp. 319–327.

[47] Hanamantagouda P Sankappanavar and Stanley Burris. “A course in universal algebra”. In: *Graduate Texts Math* 78 (1981).

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

[49] Marcel-Paul Schützenberger. “Sur Le Produit De Concatenation Non Ambigu”. In: *Semigroup Forum* 13 (1976), pp. 47–75.

[50] Marcel-Paul Schützenberger. “Une théorie algébrique du codage”. In: *Séminaire Dubreil. Algèbre et théorie des nombres* 9 (1955-1956).

[51] Saharon Shelah. “The Monadic Theory of Order”. In: *Annals of Mathematics* (1975), pp. 379–419.

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

[53] 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.

[54] 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.

[55] Denis Thérien and Thomas Wilke. “Over words, two variables are as powerful as one quantifier alternation: $FO^2 = \Sigma^2 \cap \Pi^2$”. In: *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*. 1998, pp. 41–47.

[56] 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.

[57] 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