Slightly Infinite Sets

Mikołaj Bojańczyk
July 8, 2025
The latest version can be downloaded from:
mimuw.edu.pl/~bojan/paper/atom-book

July 8, 2025

Contents

Preface

This book is about algorithms that run on objects that are infinite, but finite up to certain symmetries. Under a suitably chosen notion of symmetry, such objects – called *orbit-finite sets* – can be represented, searched and processed just like finite sets. The goal of the book is to explain orbit-finiteness and demonstrate its usefulness. Most of the examples of orbit-finite sets are taken from automata theory, since this is where orbit-finite sets began.

Chapter 1

Polynomial orbit-finite sets

The general idea in this book is to discuss sets which are built from some basic infinite set $\mathbb{A}$, and which are simple enough to be represented finitely and manipulated algorithmically. These sets will be called *orbit-finite sets*. The fully general notion will be described in Chapters 4 and 5, but we begin the book with a special case, called *polynomial orbit-finite sets*, which is simpler to formalize, and yet general enough to describe most interesting examples.

For the first few chapters of this book, the basic infinite set $\mathbb{A}$ that will be used to build the other sets will have no structure except equality. The idea of having "no structure except equality" will be formalized later on, by using invariance under atom permutations. For the moment, this idea will be apparent in the examples, and our convention that elements of $\mathbb{A}$ – which will be called *atoms* – are names such as John or Eve. Everybody knows that names have no structure beyond equality.

Before formally defining polynomial orbit-finite sets, we begin with several examples. These examples are based on automata theory, which was the original motivation for these notions.

Example 1. Consider the language

$$\{ w \in \mathbb{A}^* \mid \text{the first and last letters of } w \text{ are the same } \}.$$

To recognise this language, we use a deterministic automaton that remembers the first letter seen in its state, plus one extra bit of information that tells us whether the last letter seen is the same as the first. This state space consists of an initial state, and two disjoint copies of the atoms. We write this state space as follows, where + is used to denote disjoint union:

$$\{\text{initial}\} \quad + \quad \underbrace{\mathbb{A}}_{\text{equal}} \quad + \quad \underbrace{\mathbb{A}}_{\text{nonequal}}.$$

There are two copies of the atoms: the "equal" copy and the "nonequal" copy. For an atom $a \in \mathbb{A}$, we write equal(a) for its first copy, and nonequal(a) for its second copy. The choice of copy corresponds to storing one bit of information. The transition

function of the automaton, which is a function because the automaton is deterministic, consists of the following transitions, where a and b range over $\mathbb{A}$:

$$\begin{aligned}
\text{initial} &\xrightarrow{a} \text{equal}(a) \\
\text{equal}(a) &\xrightarrow{b} \begin{cases} \text{equal}(a) & \text{if } a = b \\ \text{nonequal}(a) & \text{if } a \neq b \end{cases} \\
\text{nonequal}(a) &\xrightarrow{b} \begin{cases} \text{equal}(a) & \text{if } a = b \\ \text{nonequal}(a) & \text{if } a \neq b. \end{cases}
\end{aligned}$$

The accepting states are those from the first copy of $\mathbb{A}$. □

The automaton in the above example was deterministic. Here is an example of an automaton that is nondeterministic.

Example 2. Consider the language

$$L = \{ w \in \mathbb{A}^* \mid \text{ some letter appears at least twice } \}.$$

When it reads an input letter, the recognizing automaton uses nondeterminism to guess if this letter will appear a second time. It then loads that letter into its state, and waits for a second appearance, upon which it enters an accepting sink state. The state space is

$$\{\text{initial}, \text{accept}\} \quad + \quad \mathbb{A}.$$

The first two states are the initial and accepting states, respectively. The transitions of this automaton are listed below, where a and b range over $\mathbb{A}$:

$$\begin{aligned}
\text{initial} &\xrightarrow{a} \text{initial} \\
\text{initial} &\xrightarrow{a} a \\
a &\xrightarrow{b} \begin{cases} \text{accept} & \text{if } a = b \\ a & \text{if } a \neq b \end{cases} \\
\text{accept} &\xrightarrow{a} \text{accept}.
\end{aligned}$$

The nondeterminism is in the first two kinds of transitions. When the automaton is in the initial state, and it sees a letter a, then it can choose to either remain in the initial state, to go to state a. The first choice is made if the automaton does not expect a to appear again, otherwise the second choice is made. We will later show that this language cannot be recognised by a deterministic automaton, but this will require a formal definition of the model. □

In the automata from the above examples, the state space could be infinite, but it had a very special form: each state would store some finite information (for example, is the state accepting or rejecting), and some atoms. So far, each state would store zero or one atom, but one could of course imagine that more atoms are stored, e.g. we could have a state space of the form

$$\mathbb{A}^0 + \mathbb{A}^0 + \mathbb{A}^1 + \mathbb{A}^7.$$

As before, we write + for disjoint union of sets. In the disjoint union above, the components of the form $\mathbb{A}^0$ represent states where no atoms are stored, such as the initial states in the two examples. This leads us to the following definition.

Definition 1.1 (Pof set)**.** A *polynomial orbit-finite set*, pof set for short, is any finite disjoint union

$$\mathbb{A}^{d_1} + \cdots + \mathbb{A}^{d_k}$$

for some $k, d_1, \ldots, d_k \in \{0, 1, \ldots\}$.

It should be clear why we use the word "polynomial" in the name – syntactically a pof set is the same thing as a univariate polynomial with coefficients in the natural numbers. The meaning of the words "orbit-finite" will become apparent later in this section, when we discuss orbits under the action of atom permutations. In a pof set, we use the name *component* for summands in the disjoint union. The pof set in the above definition has k component, and the i-th component is $\mathbb{A}^{d_i}$. The *dimension* of a component is the exponent d, the dimension of a pof set is the maximal dimension of its components.

In the first chapters of this book, we will be interested in computational models, such as automata or Turing machines, where instead of finite sets, we use pof sets. We already saw this in Examples 1 and 2, which used automata where the state spaces and input alphabets were pof sets. This resulting theory will generalize the standard theory of finite objects, because a finite set can be seen as a pof set of dimension zero. For example, a set with three elements can be seen as a pof set

$$\mathbb{A}^0 + \mathbb{A}^0 + \mathbb{A}^0$$

that has three components of dimension zero. We use the notational convention where 1 is the set $\mathbb{A}^0$, and therefore the above set can also be denoted as

$$1 + 1 + 1.$$

One can further streamline the notion, and write 3 for this set, which is something that we will also sometimes do.

In order to get a meaningful theory, we need to make some restrictions on the way that elements of pof sets are manipulated. Otherwise, we would be working with models that use countable sets instead of finite ones, and without further restrictions there is nothing interesting that can be said at this level of generality, at least as long as we care about computability.

The restriction that we make formalizes the notion that atoms have no structure beyond equality. The idea is that if atoms are renamed in a way that preserves equality, then all properties should be preserved. For example, if an automaton has a transition of the form

$$(\text{John}, \text{Eve}) \xrightarrow{\text{Adam}} (\text{Adam}, \text{John})$$

then the same automaton should also have a transition of the form

$$(\text{Tom}, \text{Adam}) \xrightarrow{\text{John}} (\text{John}, \text{Tom}),$$

because the equality patterns are the same in both transitions. This notion is formalized in the following definition, by using *atom permutations*, which are defined to be bijective functions of type $\mathbb{A} \to \mathbb{A}$. We use the convention that atom permutations are denoted by π or σ.

Definition 1.2 (Equivariant subset)**.** A subset $X \subseteq \mathbb{A}^d$ is called *equivariant* if it is stable under applying atom permutations, i.e.

$$(a_1, \ldots, a_d) \in X \quad \Leftrightarrow \quad (\pi(a_1), \ldots, \pi(a_d)) \in X$$

holds for every atom permutation π. A subset of a pof set is called equivariant if its intersection with each component is equivariant.

Example 3. [Orbits in $\mathbb{A}^3$] Consider the set $\mathbb{A}^3$. Up to atom permutations, this set contains five kinds of elements, namely a non-repeating triple

$$(\text{John}, \text{Eve}, \text{Tom}),$$

three kinds of triples that use two atoms

$$(\text{John}, \text{Eve}, \text{Eve}), \quad (\text{Eve}, \text{John}, \text{Eve}), \quad (\text{Eve}, \text{Eve}, \text{John}),$$

and a triple that uses the same atom three times

$$(\text{John}, \text{John}, \text{John}).$$

These five elements represent all possible equality types that can arise in triples of atoms. Depending on its equality type, every other element of $\mathbb{A}^3$ can be mapped to one of the above five example using an atom permutation, and the five kinds are all different, i.e. none of them can be mapped to another by an atom permutation. If we want to choose an equivariant subset of $\mathbb{A}^3$, we need to decide for each of the five kinds whether we want to include it or not. The five decisions are independent, and therefore there are 2^5 possibilities of choosing an equivariant subset. □

The kinds of elements, as described in the above example, will be called *orbits*. This is because they are the special case of the general notion of orbits under a group action, in the case where the group is the group of all atom permutations.

Definition 1.3 (Orbit)**.** The *orbit* of an element x in a pof set X is the set

$$\{\, \pi(x) \mid \pi \text{ is an atom permutation} \,\}.$$

Example 4. [Orbits in $\mathbb{A}^d$] In Example 3, we showed that the set $\mathbb{A}^3$ has five orbits. More generally, the number of orbits in $\mathbb{A}^d$ is the number of equivalence relations on the set $\{1, \ldots, d\}$. This is because an orbit describes an equality type, i.e. information about which coordinates in the tuple are equal to each other. Therefore, counting orbits in $\mathbb{A}^d$ is the same as counting equivalence relations on $\{1, \ldots, d\}$. The number of such equivalence relations is called the Bell number, and it grows exponentially with d. For example, the 4-th Bell number is 15, and the 5-th Bell number is 52. □

In the above example, we have argued that for sets of the form $\mathbb{A}^d$, the number of orbits is finite, albeit exponential. Since every equivariant set is a union of orbits, it follows that the number of equivariant subsets in $\mathbb{A}^d$ is also finite, albeit doubly exponential in the dimension d. These results extend immediately to pof sets, which are finite disjoint unions of such sets. This is because the number of orbits in a disjoint union $X + Y$ is the sum of the numbers of orbits in the summands X and Y. Hence, we get the following result, which explains the expression "orbit-finite" in the name "polynomially orbit-finite".

Lemma 1.4. *Every pof set has finitely many orbits, and finitely many equivariant subsets.*

The orbit count for a product $X \times Y$ is more subtle, and will be discussed later on. For example $\mathbb{A}^1$ has one orbit and $\mathbb{A}^2$ has two orbits, while their product $\mathbb{A}^3$ has five orbits, which shows that the formula cannot be completely trivial.

In Definition 1.2, we defined equivariant subsets of one pof set. This extends naturally to relations on pof sets, e.g. binary relations

$$R \subseteq X \times Y,$$

where X and Y are pof sets. This is because the product of two pof sets can itself be seen as a new pof set, by distributing products across disjoint unions:

$$(\sum_{i \in I} \mathbb{A}^{d_i}) \times (\sum_{j \in J} \mathbb{A}^{e_j}) \quad \equiv \quad \sum_{\substack{i \in I \\ j \in J}} \mathbb{A}^{d_i + e_j}.$$

Equivariant relations can also be described directly: a binary relation is equivariant if and only if membership in it is stable under applying the same atom permutation to both coordinates:

$$(x, y) \in R \quad \Leftrightarrow \quad (\pi(x), \pi(y)) \in R \qquad \text{for every atom permutation } \pi.$$

The above discussion was for binary relations, but the same idea extends to relations of any finite arity. Similarly, we can also talk about equivariant functions $f : X \to Y$. These are the same as binary relations that are both equivariant and functional, i.e. for every input $x \in X$ there exactly one output $y \in Y$ such that (x, y) belongs to the relation.

Example 5. [Functions with Boolean outputs] To represent booleans, we can use the set 2, which is the disjoint union $1 + 1$, where 1 is defined to be the set $\mathbb{A}^0$. An element of the set 2 consists of one bit of information, and no atoms. Sets which are disjoint union of several copies of 1 will be called *atomless*, and they will correspond to the usual finite sets. For an atomless set, the action of atom permutations is trivial, i.e. $\pi(x) = x$, since there are no atoms to change. For a pof set X, an equivariant function of type $X \to 2$ is the same thing as an equivariant subset of X. This is because, by triviality of the action on the output set, we have

$$f(x) = y \quad \Leftrightarrow \quad f(\pi(x)) = y \qquad \text{for every } y \in 2,$$

which means that the function has the same outputs for every two inputs in the same orbit. □

Example 6. [Equivariant functions of type $\mathbb{A} \to \mathbb{A}$] In this example, we show that there is only one equivariant function of type $\mathbb{A} \to \mathbb{A}$, namely the identity. Clearly the identity is equivariant, since the corresponding set of pairs is the diagonal

$$\{ (a, a) \mid \ a \in \mathbb{A} \ \},$$

and this set is equivariant. (It happens to be exactly one orbit.) Let us now prove that there is no other equivariant function of this type. Toward a contradiction, suppose that an equivariant function would map some input atom a to an output atom $b \neq a$. From the pair (a, b) we can go to any pair (a, c) with $a \neq c$ by applying an atom permutation. This would yield a violation – in fact infinitely many violations – of the functionality condition, which says that each input has only one output. □

Example 7. [Equivariant functions of type $\mathbb{A}^2 \to \mathbb{A}$] Let us list all equivariant functions of type $f : \mathbb{A}^2 \to \mathbb{A}$. If the input to such a function is a repeating pair $(a, a) \in \mathbb{A}^2$, then the output has to be a, by the same argument as in the previous example. If the input is a non-repeating pair (a, b) with $a \neq b$, then the output must be either the first argument a or the second argument b, and it cannot be a fresh atom, again by the same argument as in the previous example. Furthermore, this choice must be uniform: if for some input that is a non-repeating pair the output is the first coordinate, then this is true for all other inputs that are non-repeating pairs. This is because every non-repeating pair can be mapped to every other non-repeating pair by an atom permutation. Therefore, there are two possibilities for f: it is either the projection to the first coordinate, or the projection to the second coordinate. □

Example 8. An example of an equivariant function of type $\mathbb{A}^3 \to \mathbb{A}$ is the following function, which projects onto the second or third coordinate, depending on whether the first two coordinates are equal or not:

$$(a, b, c) \quad \mapsto \quad \begin{cases} c & \text{if } a \neq b \\ b & \text{if } a = b. \end{cases}$$

Generally speaking, an equivariant function of type $\mathbb{A}^d \to \mathbb{A}$ will look at the equality type (i.e. the orbit) of the input, and based on that orbit it will choose one of the input coordinates to be sent to the output. Therefore, the number of such functions is the product

$$\prod_X (\text{number of distinct atoms in the orbit } X),$$

where X ranges over orbits in the set $\mathbb{A}^d$. □

As shown in Lemma 1.4, a pof set will have finitely many equivariant subsets. If X and Y are pof sets, then the same will be true for $X \times Y$, and therefore there will be

finitely many equivariant relations $R \subset X \times Y$. Only some of these relations will be functions. Summing up, for every pair of pof sets X and Y, there will be finitely many equivariant functions of type $X \to Y$.

Exercises

Exercise 1. In the definition of an equivariant subset from Definition 1.2, we have an equivalence $\Leftrightarrow$, and we quantify over atom permutations, which can be briefly written as

$$0. \quad \bar{a} \in X \quad \Leftrightarrow \quad \pi(\bar{a}) \in X \quad \text{for all permutations } \pi : \mathbb{A} \to \mathbb{A}.$$

Instead of a two-way implication, we can have a one-way implication in either of the two directions, and we can quantify over functions that are not necessarily permutations, as in the following variants:

$$\begin{array}{lllll}
1. & \bar{a} \in X & \Rightarrow & \pi(\bar{a}) \in X & \text{for all permutations } \pi : \mathbb{A} \to \mathbb{A} \\
2. & \bar{a} \in X & \Leftarrow & \pi(\bar{a}) \in X & \text{for all permutations } \pi : \mathbb{A} \to \mathbb{A} \\
3. & \bar{a} \in X & \Leftrightarrow & \pi(\bar{a}) \in X & \text{for all functions } \pi : \mathbb{A} \to \mathbb{A} \\
4. & \bar{a} \in X & \Rightarrow & \pi(\bar{a}) \in X & \text{for all functions } \pi : \mathbb{A} \to \mathbb{A} \\
5. & \bar{a} \in X & \Leftarrow & \pi(\bar{a}) \in X & \text{for all functions } \pi : \mathbb{A} \to \mathbb{A}
\end{array}$$

Which variants are equivalent to the original definition, as in variant 0?

Exercise 2. Show that there is no equivariant function of type $\mathbb{A}^0 \to \mathbb{A}$.

Exercise 3. Show that the number of equivariant subsets of $\mathbb{A}^d$ is doubly exponential in d.

Exercise 4. Consider a pof set X and an equivariant binary relation $R \subseteq X \times X$. Show that the transitive closure of R is also equivariant.

1.1 Representation of equivariant subsets

The central idea of this book is that sets such as pof sets can be used instead of finite sets, and the resulting computational problems can be studied. If we want to reap the benefits of finiteness, we must use functions and subsets that respect the structure, which means that they are equivariant. For example, in pof graph, the set of vertices is a pof set, and the edge relation is required to be equivariant. In a pof automaton, the state space and input alphabet are pof sets, while the initial and final subsets, as well as the transition relation, are all required to be equivariant. (This was the case for the automata from Examples 1 and 2.)

We will be interested in decision problems, such as reachability for pof graphs or emptiness for pof automata. In order to meaningfully discuss these decision problems, we need some finite representation of their inputs. An input will typically consider of one or more pof sets (such as the input alphabet and state space of an automaton) and some equivariant relations that relate these sets (such as the transition relation in an automaton). Therefore, we need some finite representations of pof sets and equivariant relations on them.

For pof sets, there is little doubt: a pof set

$$\mathbb{A}^{d_1} + \cdots + \mathbb{A}^{d_k},$$

is represented by the list of natural numbers $d_1, \ldots, d_k$, which describe the dimensions of the various components. The relevant question is about representation of equivariant subsets. We think of an equivariant subset in a pof set as being a family of equivariant subsets, one for each component $\mathbb{A}^{d_i}$, and therefore we focus on representing equivariant subsets of a single component. There will be two representations: one will use generating sets, and the other one will use formulas.

Generating sets. The first representation of an equivariant subset is based on giving examples of elements in the set, which are then assumed to be generalised by using atom permutations. For example, the subset of $\mathbb{A}^2$ that consists of non-repeating pairs is generated by one example, namely (John, Eve), and all other elements in this subset are the same, up to choosing different names. This leads to the following definition.

Definition 1.5 (Generated subset)**.** For a pof set X, the subset *generated* by $Y \subseteq X$ is defined to be

$$\{\, \pi(y) \mid \ y \in Y \text{ and } \pi \text{ is an atom permutation } \}.$$

In other words, this is the union of orbits of the elements from Y. The idea behind the terminology in the above example is that a pof set can be seen as a set equipped with infinitely many unary operations, one for each atom permutation. The subset generated by Y is then the least set that contains Y and is closed under applying the operations. This perspective will also be used in Chapter 8, where the sets will have additional structure, namely that of a vector space, and subsets will be generated by both atom permutations and linear combinations.

Example 9. The full set $\mathbb{A}^2$ is generated by the two pairs

$$(\text{Eve}, \text{Eve}), (\text{John}, \text{Eve}).$$

As explained in Example 4, the set $\mathbb{A}^d$ is generated by a finite subset, whose size is the d-th Bell number. In particular, in order to generate the full set $\mathbb{A}^d$ we need a number of generators that is exponential in the dimension d. □

Example 10. An equivariant function $f : X \to Y$ is seen as a special case of an equivariant subset of $X \times Y$. Therefore, we can use generating sets to describe such functions. For example, the identity function of type $\mathbb{A}^2 \to \mathbb{A}^2$ is generated by

$$(\text{John}, \text{John}) \mapsto (\text{John}, \text{John}) \qquad (\text{John}, \text{Eve}) \mapsto (\text{John}, \text{Eve}).$$

In the above, we write $a \mapsto b$ instead of (a, b) when describing input/output pairs that belong to the graph of a function □

We use finite generating subsets as a representation of equivariant subsets. This assumes that we can represent individual atoms; for the moment we simply assume that atoms are strings over some finite alphabet, but the issue of representations will be discussed in more detail in Section 3.5. The representation by generating subsets is general enough to cover all equivariant subsets, as shown in the following lemma.

Lemma 1.6. *Every equivariant subset of a pof set is generated by finitely many elements.*

Proof. There are finitely many orbits, and an equivariant subset is a union of some of these orbits. For each orbit, we need only one generator. □

The above lemma shows that finite generating sets can be used as a way of representing equivariant subsets. The representation has several advantages, but conciseness is not one of them. (Non-conciseness can also be framed as an advantage, since making the inputs longer for an algorithm can give a better bound on its running time, as we will see in the next section.) For example, to represent the full subset of $\mathbb{A}^d$ we need a number of generators that is exponential in the dimension d. Another disadvantage is that this representation is not well suited to basic operations on sets. For example, the empty set has a very small representation, but its complement does not. Another example is taking pairs, as we explain below.

Example 11. Consider the subset of $\mathbb{A}^d$ that contains only non-repeating pairs. We write $\mathbb{A}^{(d)}$ for this subset. This subset is generated by one element, e.g. if $d = 3$ then a generator is

$$(\text{John}, \text{Adam}, \text{Tom}).$$

However, if we want to take the product of this subset with itself, which is an equivariant subset of $\mathbb{A}^{2d}$, then we will need a number of generators that is exponential in d. This is because $\mathbb{A}^{(d)} \times \mathbb{A}^{(d)}$ consists of tuples of length $2d$ where the first half is non-repeating and the second half is also non-repeating, but there is no further restriction on the equalities between the first half and the second half. In particular, this set will contain any tuple where the second half is a permutation of the first half, such as

$$((\text{John}, \text{Adam}, \text{Tom}), (\text{Tom}, \text{John}, \text{Adam})).$$

Each permutation will need a new generator, and therefore we will need at least $d!$ generators. The set will also contain tuples where some atoms are shared between the first and second half, and some atoms are not, such as

$$((\text{John}, \text{Adam}, \text{Tom}), (\text{Tom}, \text{John}, \text{Eve})).$$

Different kinds of sharing will also need different generators, which also gives an exponential number of generators, in terms of the dimension d. □

The disadvantage described above will be rectified by a second representation, using formulas, which is described below.

Formulas. As an alternative to generating sets, we can use formulas to represent equivariant subsets. For example, the set of non-repeating tuples in $\mathbb{A}^3$ can be described by the formula

$$x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3.$$

The variables of the formula refer to the coordinates in the tuple, and the formula is true for exactly those tuples which satisfy the desired condition (in this case, being non-repeating). The formulas that we use have no quantifiers, and are only Boolean combinations of equalities on the coordinates (quantifiers will appear later in the book).

The formula representation can be exponentially more concise than the generating set representation. For example, the full set $\mathbb{A}^d$ can be represented by the short formula "true", while the number of generators is exponential in d. Also, the representation efficiently and trivially supports such operations as complementation, which is implemented by adding $\neg$ to the formula, or intersection, which is implemented by combining to formulas with the logical connective $\wedge$. The following lemma shows that the formula representation is equivalent to the generating set representation, in the sense that both define the same subsets, namely the equivariant subsets.

Lemma 1.7. *A subset $X \subseteq \mathbb{A}^d$ is equivariant if and only if it can be defined by a formula $\varphi(x_1, \ldots, x_d)$ that is constructed using equality comparisons $x_i = x_j$ and Boolean operations $\wedge, \vee, \neg$.*

Proof. For the implication $\Leftarrow$, we observe that if we apply an atom permutation to a tuple in $\mathbb{A}^d$, then this will not change the pattern of equalities between coordinates. Therefore, the truth value of a formula that uses only equality will be preserved.

For the implication $\Rightarrow$, consider an equivariant subset $X \subseteq \mathbb{A}^d$. This subset is generated by a finite set $Y \subseteq X$, thanks to Lemma 1.6. The orbit of each generator $y \in Y$ is described by a formula, which asserts the pattern of equalities in this generator:

$$\underbrace{\Big(\bigwedge_{i,j} x_i = x_j\Big)}_{\substack{\text{conjunction ranges over}\\ \text{those coordinates}\\ i,j \in \{1,\ldots,d\}\\ \text{such that } y[i] = y[j]}} \quad \wedge \quad \underbrace{\Big(\bigwedge_{i,j} x_i \neq x_j\Big)}_{\substack{\text{conjunction ranges over}\\ \text{those coordinates}\\ i,j \in \{1,\ldots,d\}\\ \text{such that } y[i] \neq y[j]}}.$$

Since there are finitely many generators, to define X we can take the finite disjunction of these formulas, ranging over the generators. The size of the formula is the number of generators, times a factor that is polynomial in the dimension d. □

The formula representation is not without its disadvantages. For example, if we want to check if a set X is nonempty, under the formula representation, then we need to check if the corresponding formula is satisfiable. It is an easy exercise, see Exercise 5, to show that nonemptiness is an NP-complete problem under the formula representation. This is in contrast to the generating set representation, where nonemptiness is trivial: if there is at least one generator, then the generated set is nonempty. Nevertheless, in this book we will typically use the formula representation, because of how it supports basic operations on subsets.

Exercises

Exercise 5. Consider the following problem: given two subsets of a pof set decide if they are equal. Show that this problem is: (a) in deterministic logarithmic space under the generating set representation; and (b) complete for coNP under the formula representation.

Exercise 6. To specify a subset of $X \subseteq \mathbb{A}^d$, we can also use a formula with quantifiers (which range over atoms). Show that for every such formula, there is an equivalent formula that is quantifier-free. For example, the formula

$$\varphi(x_1, x_2) = \exists y\ (x_1 \neq y) \wedge (x_2 \neq y),$$

is equivalent to "true".

1.2 Graph reachability

As we mentioned before, the purpose of pof sets is to consider decision problems where the instances use pof sets instead of finite sets. We begin a simple problem of this kind, which will be used frequently later in the book, namely reachability in directed graphs.

Definition 1.8 (Pof graph)**.** A directed *pof graph* consists of a set of vertices V, which is a pof set, and an edge relation $E \subseteq V^2$ that is equivariant.

In this section, we will show that graph reachability is decidable, i.e. given a pof graph with designated source and target vertices, we can decide whether there is a directed source-to-target path. Before presenting the algorithm, and discussing its complexity, we will give some examples of pof graphs.

Example 12. [Cliques] Consider the clique on the atoms: the set of vertices is $\mathbb{A}$, and all edges are allowed, i.e.

$$E = \{\ a \to b \mid\ \ a, b \in \mathbb{A}\ \ \}.$$

This is clearly a pof graph, since the edge set is equivariant. Similarly, we could consider a clique on any set of vertices that is a pof graph. As long as we take an infinite pof sets for the vertices, these cliques will be isomorphic as graphs, because they will be countably infinite cliques. However, they will not admit any equivariant isomorphism. For example, the cliques on $\mathbb{A}^2$ and $\mathbb{A}$ are not isomorphic, since there is no equivariant bijection between the two sets. (As explained in Example 7, the equivariant functions of type $\mathbb{A}^2 \to \mathbb{A}$ are the projections, and these are not bijections.) $\square$

Example 13. The cliques from the previous example had a symmetric edge relation. Here is a non-symmetric example. The vertices are pairs of atoms, i.e. $\mathbb{A}^2$, and the edges are

$$E = \{\ (a, b) \to (b, c) \mid\ \ a, b, c \in \mathbb{A}\ \ \}$$

This graph is strongly connected, i.e. one can go from any vertex to any other vertex via a finite path. In fact, any two vertices can be connected by a path of length two:

$$(a, b) \to (b, c) \to (c, d).$$

This is not a coincidence: for every directed pof graph, if two vertices can be connected by a path, then they can be connected by a path whose length is bounded by a constant that depends only on the graph, and not the vertices, see Exercise 10. □

A directed pof graph can be represented in a finite way, by giving the pof set for the vertices, and a representation (generating set or formula) for the edge relation. Therefore, it is meaningful to discuss decision problems for pof graphs, such as reachability.

Theorem 1.9. *The following problem is decidable:*

- ***Input:*** *A pof graph, and two equivariant subsets of vertices $S, T \subseteq V$.*
- ***Question:*** *Is there a path from some vertex in S to some vertex in T?*

The complexity depends on the representation of equivariant subsets:

- *PSPACE-complete under the formula representation;*
- *NL-complete under the generating set representation.*

Proof. We give three variants of the algorithm. The first variant is a deterministic algorithm, and it illustrates the essential concept of this book, which is that exhaustive search for infinite sets is possible, if we assume equivariance and orbit-finiteness. This variant will also be the basis for extensions that will be discussed later in the book, such as the nonemptiness algorithm for pushdown automata that will be discussed in Chapter 3, and for general frameworks, such as the programming language that will be discussed in Chapter 11. The other two variants of the algorithm, which witness the complexity bounds from the statement of the theorem, are nondeterministic algorithms that are more directly tailored to the graph reachability problem.

In all algorithms, the crucial observation is that the set of reachable vertices is equivariant, and remains so if we fix a bound on the number of steps. This is because if we take any path

$$S \ni v_0 \to v_1 \to \cdots \to v_n$$

that begins in the source set, and we apply the same atom permutation π to all vertices in the path, then the new sequence of vertices will also be a path

$$S \ni \pi(v_0) \to \pi(v_1) \to \cdots \to \pi(v_n),$$

thanks to equivariance of the source set and the edge relation. By varying the atom permutation π, we can reach all vertices in the orbit of the last vertex v_n. This shows, that if a vertex can be reached in n steps, then the same is true for every vertex in the orbit of v_n. In other words, the set of vertices reachable in n steps is equivariant, and therefore it can be represented, using either generating sets or formulas.

We now describe the first variant of the reachability algorithm. This is a deterministic algorithm, which computes for each n a representation of all vertices reachable in at most n steps. (We use generators for the representation in this algorithm.) If the number of steps n exceeds the number of orbits of reachable vertices in the graph, then

no further orbits will be added, and therefore the algorithm will terminate. Initially, for $n = 0$, we use the generating set for the source vertices. Suppose that we have a generating set Γ_n for the vertices reachable in n steps. The new set of generators for step $n + 1$ is computed using the following code:

```
1    Γ_{n+1} = Γ_n
2    for (v, w) ∈ generators of edges:
3      for v ∈ set generated by Γ_n:
4        if w ∉ set generated by Γ_n:
5          Γ_{n+1} = Γ_{n+1} ∪ {w}
```

The test in line 3 is implemented by enumerating through all generators in Γ_n, and checking if there is one that has the same component and equality type as v. It is now easy to show the invariant of the program, which is that Γ_n is a generating set of the vertices reachable in n steps. In each step, we add some orbits of vertices. Since there are finitely many such orbits, the algorithm is guaranteed to stabilise, i.e. no new vertices will be added at some point. At this point, we can check if the set Γ_n contains some vertex in the same orbit as some generator of the target set, and this will tell us whether there is a path from the source set to the target set. A simple analysis of this code shows that it runs in time that is polynomial in the number of generators in the vertex set, and the number of generators in the edge set.

The rest of the proof, with proves the exact complexity bounds for the two representations, is mere bookkeeping.

Generating set representation. We begin with complexity of the problem under the generating set representation. In order to formally speak of this representation, we need to discuss how individual atoms are represented. We assume that atoms are bit strings, i.e. $\mathbb{A} = 2^*$. (This is a bit inconsistent with our convention of representing atoms as names, but of course names can be encoded in bit strings.) We will show that, under the generating set representation, the reachability problem is complete for the complexity class of nondeterministic logarithmic space (NL). When talking about logarithmic space, we use a two-tape model for Turing machines: a read-only input tape, and a read-write work tape of logarithmic size.

- **Lower bound.** A special case of our problem is reachability for finite graphs, since pof sets subsume finite sets. The reachability problem is hard for NL in the case of finite graphs, and therefore this lower bound carries over to the more general atom version of the problem.
- **Upper bound.** The algorithm for the upper bound is similar to the deterministic algorithm at the beginning of this proof, except that instead of deterministically computing all reachable orbits, we nondeterministically guess a source-to-target path, which requires storing only a single orbit at a given moment. Furthermore, the input representation contains an explicit list of all possible orbits (by looking at the generators of the edges), and therefore an orbit can be stored in logarithmic space, by pointing to the input tape.

  More formally, we reduce the problem to the special case of finite graphs. Reachability in the latter case can be solved in NL, using a naive algorithm that nonde-

terministically guesses a path, and stores the current vertex by using a pointer to the input tape (logarithmic space suffices for that). Suppose that the original instance, for reachability on pof graphs, has a list of generators

$$v_1 \to w_1, \ldots, v_n \to w_n$$

for the edges. Based on the original instance (which uses a pof set for the vertices), the reduction produces a new instance (which uses a finite set for the vertices):

- The vertices of the new instance are $v_1, w_1, v_2, w_2, \ldots, v_n, w_n$, i.e. the vertices that appear in the edge generators of the original instance. This is a finite set of vertices.
- In the new instance, there is an edge $v \to w$ if and only if there is some generator $v_i \to w_i$ which is in the same orbit. (Here, we talk about orbits of pairs of vertices in the original instance.) This means that the components are the same for v and v_i, the components are the same for w and w_i, and furthermore the equality types are the same. When talking about equality types, we also refer to comparisons between the source and target vertices in the edge. For example, if the first atom used by v is equal to the second atom used by w, then the same is true for v_i and w_i.
- In the new instance, the source vertices are those which are in the same orbit as some generator of the source set S in the original instance.
- In the new instance, the target vertices are those which are in the same orbit as some generator of the target set T in the original instance.

The correctness of the reduction is given in the following claim.

Claim 1.10. *The original instance has a source-to-target path if and only if the same is true in the new instance.*

Proof. Using equivariance of the edge relation, one shows that for every vertex in the original instance, it is reachable from a source if and only if some vertex in the same orbit is reachable in the new instance. □

The reduction can be computed in logarithmic space, even deterministically, and therefore the reachability problem is in NL.

Formula representation. We now discuss the reachability problem under the formula representation. Here, the complexity will be exponentially higher, namely polynomial space instead of logarithmic space.

- **Upper bound.** We use the same kind of nondeterministic guessing algorithm that was used for the generating set representation. We are allowed to use nondeterminism, since PSpace is equal to NPSpace by Savich's Theorem. This time, we will store on the tape a reachable vertex. (In the previous item, for

the generating set representation, we could store a pointer to the input tape, but this is no longer available, which is the entire reason for the exponentially larger complexity.) At each step, the algorithm guesses a new vertex, with atoms represented as strings, and it then checks if the formula for the edge relation allows a connection. The space used by this algorithm is polynomial in:

1. the representation of the graph;
2. the space used to represent atoms.

We will now justify why the space used by atoms is small, in fact logarithmic in the graph (polynomial would be enough for our purposes). In every transition, there are at most

$$d = 2 \cdot (\text{dimension of } V)$$

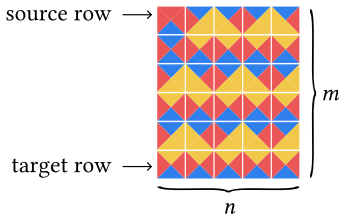
atoms that are used. When we are guessing a new vertex, we might need to get some new atoms that were not seen in the previous vertex. We can always take the shortest unused atoms, and so we will always be using the first d atoms, which can be stored using $\log d$ bits.

- **Lower bound.** This is a routine reduction from the corridor tiling problem. Let us begin by recalling what this problem is. In the corridor tiling problem, we have a finite set of square tiles, with each tile having a colour on each of its four sides. This is formalized as a finite set of colours C, together with a set of functions of type

$$\underbrace{\{\mathrm{N}, \mathrm{S}, \mathrm{E}, \mathrm{W}\} \to C}_{\text{a tile has colours on the four directions of the compass}} .$$

Each such function (which is a 4-tuple of colours) will be called a tile. Here is a picture of a set of tiles that uses three colours:

Apart from the colours and tiles, in the instance of the problem we are also given source and target rows s, t, which are sequences of tiles of the same length, say n. This length will be the width of the corridor. A solution to the corridor tiling problem is an $n \times m$ rectangle labelled by the tiles, such that the first row is s, the last row is t, and every two adjacent tiles have the same colour on their connecting side. Here is a picture of a solution:



The corridor tiling problem, i.e. deciding if there exists a solution, is known to be PSPACE-complete. We will show that the corridor tiling problem reduces to the graph reachability problem under the formula representation, thus proving PSPACE-hardness for the latter problem.

In the reduction, a vertex of the graph will store the representation of a row in the solution. Assuming that there are $k = |C|$ colours, one row will be represented by $k + 4n$ atoms

$$(\underbrace{a_1, \ldots, a_k}_{\substack{\text{distinct atoms}\\ \text{that represent}\\ \text{the tile colours}}}, \underbrace{n_1, s_1, e_1, w_1, \ldots, n_n, s_n, e_n, w_n}_{\substack{\text{four atoms for each tile in the row,}\\ \text{corresponding to the colours on the sides}\\ \text{north, south, east, west}}}).$$

Not every tuple of length $k + 4n$ will represent a row, but the tuples that do so can be specified by a formula that has size polynomial in k and d, as follows:

$$\underbrace{\bigwedge_{i \neq j \in \{1,\ldots,n\}} a_i \neq a_j}_{\text{atoms for colours are distinct}}$$

$$\wedge \underbrace{\bigwedge_{i \in \{1,\ldots,n-1\}} e_i = w_{i+1}}_{\text{colours match horizontally}}$$

$$\wedge \underbrace{\bigwedge_{i \in \{1,\ldots,n\}} \bigvee_{t \in T} n_i = a_{t(\text{north})} \wedge s_i = a_{t(\text{south})} \wedge e_i = a_{t(\text{east})} \wedge w_i = a_{t(\text{west})}}_{\text{each position is occupied by a legitimate tile}} .$$

We can further refine the formula to say that the row represents the source row, or the target row, by restricting the tile t from the last condition to be the one that should be used. This way, we get formulas for the source vertices S and the target vertices T in the instance of graph reachability that is produced by the reduction. Finally, we need to specify the formula for the edge relation. This formula has

$$\underbrace{k + 4n}_{\substack{\text{old}\\ \text{row}}} + \underbrace{k + 4n}_{\substack{\text{new}\\ \text{row}}},$$

variables. It says that both the old and new rows are valid, in the sense described above, and furthermore the south atoms of the old row match the north atoms of the new row. This, again, can be described by a formula polynomial in k and n. It is now easy to see that accepting runs of the automaton correspond to solutions of the corridor tiling problem, and therefore the nonemptiness problem is PSPACE-hard.

□

Exercises

Exercise 7. Show that the reachability problem remains PSPACE-complete when we restrict it to symmetric graphs, i.e. graphs where the edge relation is symmetric[1].

Exercise 8. Consider an undirected pof graph, i.e. a graph where the edge relation is symmetric. Does it necessarily have a spanning tree that is equivariant?

Exercise 9. Consider two undirected pof graphs, which are isomorphic. Is there necessarily an isomorphism that is equivariant?

Exercise 10. Show that given a directed pof graph, one can compute a number $k \in \{0, 1, \ldots\}$ such that for every two vertices s and t, if there is a path from s to t, then there is a path of length at most k.

Exercise 11. Consider a directed pof graph. Show that there is an infinite path if and only if there is a cycle.

Exercise 12. Consider a directed pof graph. Show that if the graph is acyclic, then there is a finite upper bound k on the length of paths.

Exercise 13. Show that the following problem is decidable: given a directed pof graph, decide if it has finite outdegree, i.e. for every vertex v, there are finitely many vertices w with an edge $v \to w$.

Exercise 14. Assume the equality atoms. Show a graph which has an infinite path, but does not have any infinite finitely supported path.

[1] Note that in the case of finite graphs, the complexity drops from NL to L when restricting to symmetric graphs, as shown in Reingold (2008).

Chapter 2

Automata for polynomial orbit-finite sets

This chapter is devoted to automata for polynomial orbit-finite sets, which we call pof automata. We show that, despite being formally infinite, these automata can be treated algorithmically, and some of their decision problems can be decided. However, this comes at a certain cost – not all constructions are allowed, and the model is less robust than for finite sets. For example, determinisation fails, because the powerset construction does not work.

2.1 Automata and their emptiness problem

We begin by formally defining the models, namely deterministic and nondeterministic automata.

Definition 2.1 (Pof automaton)**.** A nondeterministic *pof automaton* consists of:

1. an input alphabet Σ, which is a pof set;
2. a state space Q, which is a pof set;
3. initial and accepting subsets $I, F \subseteq Q$, which are equivariant;
4. a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, which is equivariant.

A deterministic pof automaton is the special case where there is exactly one initial state, and where the transition relation is a function.

As was the case for graphs, the above definition is simply the usual definition, except that finite sets are replaced by pof sets, and all subsets, relations and functions are required to be equivariant. Using the same principle, one can consider pof variants of other models, such as pushdown automata, context-free grammars, Turing machines, etc. This will be the content of Chapter 3.

Using the result about graph reachability, we obtain decidability of emptiness for pof automata, deterministic or not.

Theorem 2.2. *The emptiness problem is decidable for nondeterministic pof automata. The complexity is the same as for the reachability problem in pof graphs.*

Proof. The lower bounds for graph reachability transfer directly to the emptiness problem, by considering automata over a one-letter alphabet, which are the same as instances of graph reachability. For the upper bound under the formula representation, we can use the same straightforward nondeterministic algorithm as in graph reachability. For the upper bound under the generating set representation, we simply delete the input letters from the generators of the transitions, and then we solve the corresponding instance of graph reachability. □

A corollary of the above theorem is decidability of language equivalence for deterministic pof automata.

Corollary 2.3. *The following problem is decidable:*

- ***Input:*** *Two deterministic pof automata.*
- ***Question:*** *Do they recognise the same language?*

The complexity is the same as for the emptiness problem.

Proof. We can use the product construction. A product $Q_1 \times Q_2$ of two pof sets is also a pof sets, and the corresponding operations on subsets and transitions preserve equivariance. We then check if the product automaton can reach states that are accepting in one automaton, but rejecting in the other. □

In the equivalence algorithm from the above corollary, we use determinism. This is because we complement an automaton by complementing its accepting states, and this only works for deterministic automata. As we will see in the next section, the equivalence problem becomes undecidable for nondeterministic automata. Let us first show that we cannot solve this problem by determinising. The natural idea would be to use the powerset construction. Unfortunately, this fails. Already for the simplest pof set that is not finite, namely $\mathbb{A}$, the powerset $\mathsf{P}\mathbb{A}$ is not a pof set. In fact, not only this construction fails, but there is no successful construction at all, as shown in the following theorem.

Theorem 2.4. *Languages recognised by nondeterministic pof automata are not closed under complementation. Also, deterministic pof automata are strictly less expressive than nondeterministic ones.*

Proof. The first part of the theorem directly implies the second part, since deterministic automata are closed under complementation. To prove the first part, we use the language

$$L = \{\, w \in \mathbb{A}^* \mid \text{ some letter appears at least twice } \}.$$

In Example 2, we showed that this language is recognised by a nondeterministic pof automaton. It remains to show that its complement is not recognised by any nondeterministic pof automaton.

The complement of L consists of words where all letters are pairwise different. Suppose, toward a contradiction, that this complement is recognised by a nondeterministic pof automaton. Let d be the dimension of the state space, i.e. the maximal number of atoms that can be stored in a state. Choose n so that it is strictly larger than this dimension, and consider a word w with $2n$ pairwise distinct atoms. This belongs to the complement of L, and thus it must have an accepting run in the hypothetical automaton for the complement. Consider the states at the beginning, middle and end of the accepting run, i.e. the states

$$I \ni p \xrightarrow{w_1} q \xrightarrow{w_2} r \in F,$$

where w_1 and w_2 are the two halves of w that have length n. Because n was chosen to be strictly bigger than the dimension of the state space, there must be some atom a that appears in the first half w_1 but not in the state q. Similarly, there must be some atom b that appears in the second half w_2 but not in the state q. Let π be the atom permutation that swaps a and b. By equivariance, we have

$$\pi(q) \xrightarrow{\pi(w_2)} \pi(r) \in F.$$

Since neither a nor b appear in the state q, the permutation π does not move this state, i.e. $\pi(q) = q$, and therefore we can stitch the two runs above to get an accepting run over the concatenation of w_1 and $\pi(w_2)$. This concatenation has an atom repetition, and therefore should be rejected, thus leading to a contradiction. □

Exercises

Exercise 15. Find a deterministic pof automaton for the following language:

$$\{ w \in \mathbb{A}^* \mid \text{ the first and last letters are different } \}.$$

Exercise 16. Find a deterministic pof automaton for the following language:

$$\{ w \in \mathbb{A}^* \mid \text{no two consecutive letters are the same} \}.$$

Exercise 17. Find a deterministic pof automaton for the language

$$\{ w \in \mathbb{A}^* \mid \text{there are at least three different letters} \}.$$

Exercise 18. Consider a deterministic pof automaton. Show that after reading an input string w, all atoms that appear in the state must also appear in w.

Exercise 19. Show that if the input alphabet is finite (i.e. a pof set of dimension zero), then deterministic pof automata recognise exactly the regular languages.

Exercise 20. Consider a nondeterministic pof automaton, and let k be the maximal number of atoms that can appear in a single transition. Let A be a finite set of k atoms. Show that if the automaton is nonempty, then it accepts some word that uses only atoms from A.

Exercise 21. Define a left derivative of a language $L \subseteq \Sigma^*$ to be a language of the form

$$v^{-1}w \stackrel{\text{def}}{=} \{ w \in \Sigma^* \mid \ vw \in L \ \}$$

for some word $v \in \Sigma^*$. Are languages recognised by deterministic pof automata closed under left derivatives?

Exercise 22. We say that two states p and q in a deterministic pof automaton are *behaviourally equivalent* if for every input string w, the states pw and qw are both accepting or both rejecting. Show that behavioural equivalence is equivariant.

Exercise 23. A deterministic pof automaton is called minimal if one cannot find reachable states $p \neq q$ that are behaviourally equivalent. Show a language that is recognised by some deterministic pof automaton but not by any minimal deterministic pof automaton.

Exercise 24. Show that the expressive power of nondeterministic pof automata does not change if we allow ε-transitions.

Exercise 25. Show that for every nondeterministic pof automaton, the set

$$\{ n \in \{0, 1, \ldots\} \mid \text{ the automaton accepts some word of length } n \ \}$$

is ultimately periodic.

Exercise 26. Show a family of deterministic pof automata, such that in the n-th automaton the input alphabet is $\mathbb{A}$, the state space is $\mathbb{A}^0 + \mathbb{A}^n$, but the shortest accepted word is exponential in n.

Exercise 27. Show that for every deterministic pof automaton one can compute some bound k such that if two states p and q are not behaviourally equivalent, then they can be distinguished by some word of length at most k. Show that this k can be exponential in the dimension of the state space.

Exercise 28. Show that the following problem is decidable: given a deteriminstic pof automaton, decide if its language is commutative.

Exercise 29. A language is called *positively equivariant* if for every function $\pi : \mathbb{A} \to \mathbb{A}$ which is not necessarily a bijection, we have

$$w \in L \quad \Rightarrow \quad \pi(w) \in L.$$

Show that the following problem is decidable: given a determinstic pof automaton, decide if its language is positively.

Exercise 30. Let $L \subseteq \Sigma^*$ be a language recognised by a deterministic pof automaton. Show that there is some k with the following property: for every $w \in \Sigma^*$ there are atoms $a_1, \ldots, a_k \in \mathbb{A}$ such that for every atom permutation π that fixes all atoms from $a_1, \ldots, a_k$, and every $v \in \Sigma^*$ we have

$$wv \in L \quad \Leftrightarrow \quad \pi(w)v \in L.$$

Exercise 31. Suppose that $L \subseteq \Sigma^*$ is recognised by a nondeterministic pof automaton. Show that there is some k such that for every $w \in \Sigma^*$ longer than k one can find

1. a decomposition $w = xyz$ with y nonempty; and
2. an atom permutation π that moves finitely many atoms

such that for every $n \in \{0, 1, \ldots\}$ we have

$$xy\pi^1(y)\pi^2(y) \cdots \pi^n(y)\pi^n(z) \in L.$$

Exercise 32. Is there a deterministic pof automaton for the following language?

$$\{ w \in \mathbb{A}^* \mid \text{ all letters are different } \}$$

Exercise 33. Which of the following closure properties are true for the class of languages recognised by deterministic pof automata?

1. Complementation;
2. union;
3. intersection;
4. reverse;
5. concatenation;
6. Kleene star.

Exercise 34. Which of the closure properties from Problem 33 hold for nondeterministic pof automata?

Exercise 35. Show that the complement of the language

$$\{ ww \mid \ w \in \mathbb{A}^* \text{ is non-repeating } \}$$

is recognised by a nondeterministic pof automaton.

Exercise 36. In this exercise, we consider a variant of regular expressions. Consider the least class of languages that:

1. contains every equivariant set of words that has bounded length;
2. is closed under union and concatenation;
3. is closed under Kleene star L^*.

Show that these languages are a strict subset of nondeterministic, pof automata.

Exercise 37. Show that the regular expressions from the previous exercise are not closed under intersection.

2.2 Undecidable universality

In Corollary 2.3, we showed that language equivalence is decidable for deterministic pof automata. For finite automata, this result extends to nondeterministic automata using determinisation, but determinisation fails for pof automata, as shown in Theorem 2.4. We will now show that equivalence is in fact undecidable for nondeterministic automata. Already a special case of the language equivalence problem will be undecidable, namely universality: given one nondeterministic pof automaton, we want to decide if it accepts all words, i.e. it is equivalent to the automaton that accepts all words.

Theorem 2.5. *The following problem is undecidable:*

- ***Input:*** *A nondeterministic pof automaton.*

- ***Question:*** *Does it accept all words?*

Proof. We reduce from the halting problem for counter machines.

Let us begin by describing counter machines, which are also known as Minsky machines. This is a computational model that uses counters as its data structure. Each counter stores a natural number, and counters are updated via increments, decrements and zero tests. The syntax of the machine is given by a finite set of states Q, a finite set C of counters, and a finite set of transitions

$$\Delta \subseteq \underbrace{Q}_{\substack{\text{source}\\ \text{state}}} \times \underbrace{\{\text{inc}, \text{dec}, \text{zero}\}}_{\text{counter operations}} \times \underbrace{C}_{\substack{\text{which}\\ \text{counter is}\\ \text{affected}}} \times \underbrace{Q}_{\substack{\text{target}\\ \text{state}}}.$$

The increment operation adds 1 to the affected counter and leaves the other counters unchanged, the decrement operation subtracts 1, and the zero test transition does not change the counters but is only enabled if the corresponding counter is equal to zero. A decrement is not enabled if the corresponding counter is zero, since we require counters to be natural numbers. The semantics of the machine is its *configuration graph*, which is a directed graph where the vertices are pairs of the form

$$\underbrace{Q}_{\text{state}} \times \underbrace{\mathbb{N}^C}_{\substack{\text{counter}\\ \text{valuation}}}$$

and where the edges are given by the transitions in the expected way. The following problem, which we call the *halting problem for counter machines*, is a classic undecidable problem:

- **Input:** a counter machine, and two states p and q.
- **Question:** is there a path from $(p, \bar{0})$ to $(q, \bar{0})$ in the configuration graph?

We will show that the above problem reduces to universality of nondeterministic pof automata, and therefore the latter problem is also undecidable. (The halting problem for counter machines is known to be undecidable even for two counters. However, we do not need this in our reduction, since it will also work with more than two counters.)

In the reduction, we define a language L, whose input alphabet is $\mathbb{A} \times \Delta$, i.e. an input letter consists of an atom and a transition of the counter machine. The intuition is that this language represents accepting computations of the counter machine, and the atoms are used to describe matching pairs of increments and decrements. Our goal will be to give a nondeterministic pof automaton that recognises the complement of this language. (Therefore, if the pof automaton is universal, then the language is empty, i.e. there is no accepting computation of the counter machine.) Formally, the language L is defined to be the words that satisfy the following two conditions:

1. The first letter uses a transition whose source state is the initial state p, the last letter uses a transition whose target state is q, and transitions in consecutive letters agree on the states that connect them (i.e. the target state of the previous transition is equal to the source state of the next transition).

2. An atom can appear at most twice. Also:

   (i) if an atom appears once, then its only appearance labels a zero test;

   (ii) if an atom appears twice, then the first appearance labels an increment, the second appearance labels a decrement on the same counter, and there are no zero tests on this counter between them.

It is not hard to see that L is nonempty if and only if there is an accepting computation of the counter machine. This is because the conditions in the language ensure that increments and decrements match, and zero tests cannot be properly enclosed by a matching pair of increments and decrements. Therefore, if we could decide emptiness of L, then we could decide the halting problem. Emptiness of L is the same as universality for its complement. The following lemma shows that the complement is recognised by a pof automaton, thus completing the reduction.

Lemma 2.6. *The complement of L is recognised by a nondeterministic pof automaton.*

Proof. A word belongs to the complement of L if and only if it violates one of the two conditions 1 or 2 in the definition of L. Since nondeterministic pof automata are closed under unions, it is enough to give separate automata for the two conditions. Condition 1 does not refer to atoms, and therefore its complement can be recognised by a finite automaton, because regular languages on finite (i.e. atom-less) alphabets are closed under complementation. The interesting part is violations of condition 2. This condition is violated if and only if at least one of the following holds:

(a) some atom appears at least three times; or

(b) some atom appears at least twice, but in a way that violates condition 2(ii), i.e. either the two appearances are not matching increment/decrement pairs, or there is a zero test between them; or

(c) some atom appears exactly once, but its label is not a zero test.

It is enough to give a separate automaton for each of the three kinds of violations.

(a) Violations of this kind, i.e. words where some atom appears at least three times, can be recognised by an automaton which stores the repeated atom in its state, using state space

$$\underbrace{\mathbb{A}^0}_{\text{initial}} + \underbrace{\mathbb{A}^1}_{\text{after first}} + \underbrace{\mathbb{A}^1}_{\text{after second}} + \underbrace{\mathbb{A}^0}_{\substack{\text{after third,}\\ \text{thus accepting}}}.$$

(b) In this kind of violation, some atom appears at least twice and either: (1) the labels are not matching increment/decrement pairs; or (2) the labels are matching increment/decrement pairs, but they are separated by a zero test on the corresponding counter. We explain the second case (2), and leave (1) to the reader. For case (2), we have a union of finitely many automata, each of which checks

case (2) for one of the finitely many counters. Once we fix the counter c, the state space is:

$$\underbrace{\mathbb{A}^0}_{\text{initial}} + \underbrace{\mathbb{A}^1}_{\text{after increment}} + \underbrace{\mathbb{A}^1}_{\text{after zero test}} + \underbrace{\mathbb{A}^0}_{\text{accepting}}.$$

The automaton starts reading the word in the first component, which does not store any atoms. When it sees an increment on counter c, the automaton can nondeterministically choose to move to the second component "after increment", and store the corresponding atom a in its state. Then, the automaton waits until it sees a zero test on counter c, with any atom used as a label, upon which it moves to the third component "after zero test", while keeping in its state the atom a that it loaded upon seeing an increment. Finally, when the automaton sees a decrement on counter c with the atom a from the original increment, it moves to the last component, which is an accepting sink state.

(c) The most interesting violations are of this kind, where some atom appears exactly one time but not in a zero test. The challenge is that the automaton needs to check that the atom appears exactly once. For this reason, it must guess the atom at the beginning of the input word, before having seen it, to check that it does not appear earlier in the word. This is done by an automaton with a state space

$$\underbrace{\mathbb{A}^1}_{\text{initial}} + \underbrace{\mathbb{A}^1}_{\text{after first}} + \underbrace{\mathbb{A}^0}_{\text{accepting}},$$

where each initial state stores an atom that is nondeterministically guessed. The only nondeterminism in this automaton is the choice of initial state, but this is an infinite kind of nondeterminism, since there are infinitely many possible atoms to start with. In the next section, we will discuss this kind of nondeterminism in more detail.

□

Observe that the automaton in the proof of the lemma above had dimension one, and therefore the universality problem is undecidable already in dimension one. □

Exercises

Exercise 38. To express properties of words in $\mathbb{A}^*$, we can use first-order logic, where the quantifiers range over positions, and there are predicates for the order on positions, and equality of data values. For example, the following formula says that the first position has the same atom as some later position:

$$\forall x \quad \underbrace{(\forall y\ y \geq x)}_{x \text{ is the first position}} \quad \Rightarrow \quad \underbrace{(\exists y\ y > x \wedge y \sim x)}_{\substack{x \text{ has the same atom} \\ \text{as some later position}}}.$$

Show that satisfiability is undecidable for this logic, i.e. one cannot decide if a given formula is true in some word from $\mathbb{A}^*$.

2.3 A decidable case of universality

In this section, we show that under extra assumptions, we can recover decidability of universality for nondeterministic pof automata. There is not much space here, since the undecidability argument used automata with atom dimension one, i.e. a state would store at most one register. Of course atom dimension zero would be sufficient for decidability, since such automata determinise (even if the input alphabet is infinite), but we are looking for something more exciting. It turns out that the crucial distinction is the nondeterministic guessing of atoms that was used in the reduction in the previous section. We formalize this in the following definition.

Definition 2.7 (Guessing automaton)**.** A nondeterministic pof automaton is called *guessing* if there is some initial state that contains an atom, or some transition

$$p \xrightarrow{a} q,$$

where q contains an atom that appears neither in p nor a.

The idea behind a guessing automaton is that a state of the automaton can store an atom that was not seen in the input word. The contrapositive is that if an automaton is non-guessing, then every atom in the state must have been seen in the input word.

The automaton for condition (c) in the proof of Lemma 2.6 used guessing, since its initial states contained atoms. We will show that if the automaton is non-guessing, and its state space has dimension at most one, then the universality problem is decidable. Although this result covers a very restricted model, we include it in the book because it uses an interesting technique, namely well-quasi-orders. The bound is tight – if we allow guessing then we can use the undecidability proof from Theorem 2.5, and if we allow dimension two even without guessing, then we also get undecidability, which is left as an exercise for the reader.

Theorem 2.8. *The following problem is decidable:*

- ***Input:*** *A nondeterministic pof automaton, which is non-guessing and has a state space of dimension at most one.*
- ***Question:*** *Does it accept all words?*

We do not give any complexity bounds. The algorithm that we provide is highly inefficient, and is based on a brute-force procedure that searches through all possible witnesses of some kind, with no explicit bound on the size of these witnesses. This is not far from optimal, since one can give non-elementary lower bounds for the complexity of the problem.

The rest of Section 2.3 is devoted to proving the above theorem. As mentioned in Theorem 2.2, automata cannot be determinised, and therefore the powerset construction does not work for pof sets. However, as we will see in this proof, the two extra assumptions – dimension at most one and non-guessing – will enable us to exhaustively search the state space in the powerset automaton.

For the rest of this proof, fix a nondeterministic pof automaton

$$\mathcal{A} = (Q, \Sigma, \Delta, I, F)$$

that we want to check for universality. We will discuss the usual powerset automaton, which we denote by $\mathsf{P}\mathcal{A}$, despite this automaton not being a pof automaton. Let us recall the construction of the powerset automaton. The input alphabet is the same. States in the powerset automaton are sets of states in the original automaton, with the initial state being I, and the final states being those that intersect F. The point of the powerset automaton is that it is deterministic: when it is in a state $P \subseteq Q$, and it reads an input letter a, then it deterministically goes to the state

$$\{\, q \in Q \mid \ p \xrightarrow{a} q \text{ for some } p \in P \ \}.$$

The non-guessing assumption ensures that only finite sets of states appear in the powerset automaton.

Lemma 2.9. *If the automaton $\mathcal{A}$ is non-guessing, then every reachable state in the powerset automaton is a finite subset of Q.*

Proof. Since the automaton $\mathcal{A}$ is non-guessing, every atom in a reachable state must have appeared previously in the input word. If we fix the input word, then there are finitely many possible states which use atoms from it, since a pof set can only have finitely many elements that use a given finite set of atoms. Therefore, reachable states of the powerset automaton will be finite subsets of Q. □

Thanks to the above lemma, from now on, we will be working in the *finite powerset automaton*, which is obtained from the powerset automaton by restricting its state space to finite sets. The general idea behind our universality algorithm is that it will exhaustively search for two kinds of witnesses:

- a finite witness that the automaton rejects some word; or
- a finite witness that the automaton accepts all words.

The witnesses of the first kind are straightforward: they are rejected words. The witnesses of the second kind are described in the following lemma (we will later explain why these witnesses can be viewed as finite).

Lemma 2.10. *The automaton $\mathcal{A}$ accepts all words if and only if there is a set*

$$\mathcal{R} \subseteq \textit{states of the finite powerset automaton} = \mathsf{P}_{\textit{fin}} Q$$

with the following properties:

1. *$\mathcal{R} \ni I$, i.e. $\mathcal{R}$ contains the initial state of the finite powerset automaton;*
2. *$\mathcal{R}$ is closed under applying transitions of the finite powerset automaton;*
3. *$\mathcal{R}$ contains only sets that intersect F;*
4. *$\mathcal{R}$ is equivariant, i.e. closed under applying atom permutations;*
5. *$\mathcal{R}$ is upward closed with respect to inclusion, when restricted to finite sets: if a finite set $P \subseteq Q$ contains some set from $\mathcal{R}$, then also $P \in \mathcal{R}$.*

Proof. The implication $\Leftarrow$ is immediate; in fact it holds already if we only keep the first three conditions 1–3. Conditions 1–2 ensure that $\mathcal{R}$ contains all reachable states of the finite powerset automaton, and condition 3 ensures that these reachable states witness acceptance of $\mathcal{A}$.

Let us now prove the implication $\Rightarrow$. Define $\mathcal{R}$ to be the upward closure of the reachable states of the powerset automaton, i.e. $\mathcal{R}$ is the finite sets that contain some reachable state of the powerset automaton. Conditions 1, 3 and 5 follow from the definition of $\mathcal{R}$. By equivariance of the original automaton, the set of reachable states in the powerset automaton is also equivariant, and therefore so is its upward closure, thus proving condition 4. Finally, condition 2 follows from monotonicity of the transition function of the powerset automaton: if we increase the source state, then we also increase the target state. □

As mentioned in the proof, the equivalence in the above lemma would continue to hold without the last two conditions 4 and 5 about equivariance and upward closure. The purpose of these conditions, and of the assumption that Q has dimension at most one, is to ensure that $\mathcal{R}$ can be represented in a finite way. This representation will be based on the following order on finite subsets of Q:

$$R_1 \sqsubseteq R_2 \qquad \text{iff} \qquad \pi(R_1) \subseteq R_2 \text{ for some atom permutation } \pi. \tag{2.1}$$

It is not hard to see that two subsets are equivalent under the above order, i.e. they are compared in both directions, if and only if they are in the same orbit (of the finite powerset) under atom permutations. It is also not hard to see that a subset $\mathcal{R}$ is upward closed with respect to this order if and only if it satisfies conditions 4 and 5. The following lemma shows that every upward closed set – and therefore also the set $\mathcal{R}$ from Lemma 2.10 – is the upward closure of a finite set, and thus it can be represented in a finite way. The lemma crucially uses the assumption on dimension one, and it would fail for atom dimension two or more.

Lemma 2.11. *Let Q be a pof set of dimension one. If $\mathcal{R} \subseteq \mathsf{P}_{fin}Q$ is upward closed with respect to the order $\sqsubseteq$, then there is a finite set $\mathcal{R}_0 \subseteq \mathcal{R}$ such that*

$$R \in \mathcal{R} \qquad \textit{iff} \qquad R_0 \sqsubseteq R \textit{ for some } R_0 \in \mathcal{R}_0.$$

Proof. To prove the lemma, we will use a similar observation about vectors of natural numbers, equipped with the coordinate-wise ordering

$$(x_1, \ldots, x_d) \leq (y_1, \ldots, y_d) \quad \stackrel{\text{def}}{=} \quad x_1 \leq y_1 \wedge \cdots \wedge x_d \leq y_d.$$

This observation is called Dickson's Lemma, and is stated below.

Dickson's Lemma *Let $X \subseteq \mathbb{N}^d$ be a set that is upward closed with respect to the coordinate-wise ordering. Then there is some finite set $X_0 \subseteq X$ such that*

$$x \in X \qquad \textit{iff} \qquad x_0 \leq x \textit{ for some } x_0 \in X_0.$$

We will reduce the present lemma to Dickson's Lemma, using the assumption that Q has atom dimension at most one. Let us decompose Q into components of dimension zero and one:

$$Q \quad = \quad \underbrace{\mathbb{A}^0 + \cdots + \mathbb{A}^0}_{\substack{k \text{ components of} \\ \text{dimension zero}}} \quad + \quad \underbrace{\mathbb{A}^1 + \cdots + \mathbb{A}^1}_{\substack{\ell \text{ components of} \\ \text{dimension zero}}}$$

For a finite set $R \subseteq Q$, define its *profile* to be the following information:

i. which elements from components of dimension zero belong to R;

ii. for each nonempty subset $I \subseteq \{1, \ldots, \ell\}$, how many atoms a satisfy

$$i \in I \qquad \text{iff} \qquad \text{the } i\text{-th copy of } a \text{ belongs to } R.$$

The point of profiles is that they are essentially vectors of natural numbers. More precisely, we can view the profile as a function

$$\mathsf{P}_{\text{fin}} Q \to \underbrace{\{0,1\}^{\{1,\ldots,k\}}}_{\substack{\text{answers to} \\ \text{question (i)}}} \times \underbrace{\mathbb{N}^{\text{nonempty subsets of } \{1,\ldots,\ell\}}}_{\substack{\text{answers to} \\ \text{question (ii)}}}$$

This allows us to view finite sets of states as vectors of natural numbers of fixed dimension, which will enable us to use Dickson's Lemma. (Observe that this technique would no longer work for dimension two, since we would need to describe how pairs of atoms interact).

A finite set is identified by its profile up to atom permutations: two finite subsets of Q have the same profile if and only if they are in the same orbit. Together with equivariance of $\mathcal{R}$, this implies that membership in $\mathcal{R}$ can be seen as a question about profiles, i.e. we have

$$R \in \mathcal{R} \qquad \Leftrightarrow \qquad \text{profile}(R) \in \mathcal{P}, \tag{2.2}$$

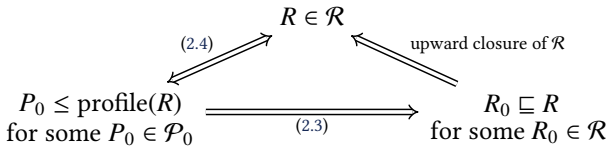
where $\mathcal{P}$ is defined to be the image of $\mathcal{R}$ under the profile map. Furthermore, the profile map has the following monotonicity property, where profiles are ordered coordinate-wise:

$$R_1 \sqsubseteq R_2 \qquad \Leftarrow \qquad \text{profile}(R_1) \leq \text{profile}(R_2). \tag{2.3}$$

This is because increasing the profile corresponds to adding states to the set. (The converse implication is not true. If we add a new state to R_1, and the atom of this state is already present in R_1, then the effect on the profile will not be a coordinate-wise increase, since one coordinate will be incremented, and another coordinate will be decremented. If we changed the coordinate-wise ordering to a slightly more subtle ordering, then we could recover the converse implication.) Thanks to the monotonicity property (2.3) and upward closure of $\mathcal{R}$, the set of profiles $\mathcal{P}$ is upward closed under $\leq$. Therefore, we can apply Dickson's Lemma to conclude that the set of profiles $\mathcal{P}$ is generated by some finite set of profiles $\mathcal{P}_0$. Together with (2.2), this gives us

$$R \in \mathcal{R} \quad \Leftrightarrow \quad P_0 \leq \text{profile}(R) \text{ for some } P_0 \in \mathcal{P}. \tag{2.4}$$

Choose $\mathcal{R}_0$ so that its profiles are exactly those from $\mathcal{P}_0$. The conclusion of the lemma is proved in the following diagram:



□

We are now ready to complete the proof of the theorem. Define a witness for universality to be a set $\mathcal{R}$ as in Lemma 2.10, which is represented by a finite set $\mathcal{R}_0$ as in Lemma 2.11. Define a witness for non-universality to be a rejected word. The algorithm exhaustively searches for witnesses of both kinds. It is guaranteed to find one in finite time, since both kinds of witnesses are complete characterisations. (As mentioned before, we have no explicit bounds on the running time of the algorithm, beyond saying that it runs in finite time.) It remains to show that one can verify a witness for universality: given a finite set $\mathcal{R}_0$, one can check if its upward closure $\mathcal{R}$ satisfies the conditions of Lemma 2.10. The only interesting condition here is 2, which says that $\mathcal{R}$ is closed under applying transitions. By monotonicity of the transition function in the powerset automaton, this reduces to checking if for every R in the finite set $\mathcal{R}_0$ and every input letter $a \in \Sigma$, the resulting state is in $\mathcal{R}$. Although there are infinitely many possible letters a, we only need to check this for finitely many choices, since we only need to use at most d fresh atoms, where d is the atom dimension of the input alphabet and fresh atoms are those that do not appear in R. This completes the proof of Theorem 2.8.

Exercises

Exercise 39. Show that languages recognised by one way non-guessing alternating automata are not closed under reversals.

Exercise 40. Show that the order defined in (2.1) is no longer a well-quasi ordering if we use infinite subsets of $\mathbb{A}$.

Exercise 41. Show that the order defined in (2.1) is no longer a well-quasi ordering if we use finite subsets of $\mathbb{A}^2$ instead of $\mathbb{A}$. For example, we have

$$\{(\text{John}, \text{Eve}), (\text{John}, \text{John})\} \quad \leq \quad \{\underbrace{(\text{Eve}, \text{Tom})}_{\substack{\text{this pair} \\ \text{is deleted}}}, \underbrace{(\text{Mark}, \text{John}), (\text{Mark}, \text{Mark})}_{\substack{\text{to the remaining elements, we} \\ \text{apply an atom permutation with} \\ \text{Mark} \mapsto \text{John and John} \mapsto \text{Eve}}} .\}$$

Exercise 42. Consider the following ordering on $\mathbb{A}^*$. We say that $w \leq v$ if one can obtain w from v as follows: (a) first delete some letters from v; then (b) apply some atom permutation. Is this a well-quasi-ordering?

Exercise 43. We say that a language $L \subseteq \Sigma^*$ is *upward closed* if it is closed under inserting letters. In other words,

$$wv \in L \quad \Rightarrow \quad wav \in L \quad \text{for every } w, v \in \Sigma^* \text{ and } a \in \Sigma.$$

Is it true that every language that is both equivariant and upward closed is necessarily recognised by a nondeterministic pof automaton?

Chapter 3

More computational models with atoms

In the previous sections, we discussed pof variants of deterministic and nondeterministic automata. In this chapter, we give a sample of other models of computation, namely alternating automata, two-way automata, circuits, context-free grammars, and Turing machines. This material is nothing but a collection of exercises, each one preceded by a brief description of the relevant model of computation.

We begin with some more variants of finite automata. In the previous chapter, we showed that in the pof setting, nondeterministic automata are no longer equivalent to deterministic. There are two more examples of this phenomenon, namely two other variants of automata that are equivalent to the usual automata in the atom-free case, but are no longer equivalent in the presence of atoms. These are alternating automata and two-way automata, which are discussed in Sections 3.1 and 3.2.

3.1 Alternating automata

Alternating pof automaton are a generalization of nondeterministic automata, which is self-dual in the sense that it does not privilege existential choice over universal choice. Let us describe this model.

Definition 3.1 (Pof alternating automaton)**.** The syntax of a pof alternating automaton consists of two pof sets Q and Σ for the states and input alphabet, as well as:

1. an initial state $q_0 \in Q$, which is equivariant (i.e. contains no atoms);
2. a partition of Q into two equivariant subsets, called *existential* and *universal*;
3. an equivariant set of transitions $\delta \subseteq Q \times (\Sigma + \{\epsilon\}) \times Q$,

Note that the automaton does not have an accepting set of states. The language recognised by such an automaton is defined in terms of a game, which is played by

two players, called the existential and universal player. The existential player represents acceptance, i.e. the word is accepted when the existential player wins. Intuitively speaking, the game is designed so that the two players construct a run of the automaton by choosing transitions, with the choice being made by the player who owns the current state. The goal of each player is to force the opponent into a situation where they need to choose a transition, but there is no transition that is available. Once such a situation arises, the player who cannot choose a transition loses, and the game is terminated immediately. Since the automaton has ε-transitions, the game might last forever, in which case we assume that the existential player wins (unfortunately, this breaks the symmetry between the players in the model, which is an issue that we will discuss below).

Here is a more detailed description of the game. A position of the game is a pair (q, w) consisting of a state q and a possibly empty input word w. In such a position, the player who owns the state q according to the partition on states picks a transition

$$(q, a, p) \in \delta$$

such that: (1) the source state q of the transition is the same as in the current position; and (2) the letter a of the transition is either ϵ or the first letter of the word w in the current position. If there is no transition which satisfies the two criteria (for example, the word w is empty and there are no ε-transitions), then the game is terminated immediately, and the player making the choice loses. Otherwise, the position is updated by using the target state p of the chosen transition, and updating the input word w by removing the letter a from the beginning (which has no effect if $a = \varepsilon$). Then the game continues from the new position. The game might last forever, by repeatedly using ε-transitions. If the game lasts forever then we assume that the existential player wins. An input word w is accepted by the automaton if the existential player has a strategy to win the game, starting from the position (q_0, w).

In the atomless case, the alternating model is equivalent to usual automata, i.e. it recognises exactly the regular languages over finite alphabets, see Exercise 44. In fact, alternating automata are an important model, especially for infinite objects, such as infinite words or trees. In the presence of atoms, this model is more powerful than nondeterministic automata, as we will see in Example 15. We begin with a very simple example of an alternating automaton, which explains how accepting states can be simulated in the model.

Example 14. [A letter appears at least twice] Recall the nondeterministic automaton from Example 2, which accepted words in $w \in \mathbb{A}^*$ where some atom appeared at least twice. This automaton had a state space of the form

$$\underbrace{\{\text{initial, accept}\}}_{\text{formally speaking, two copies of } \mathbb{A}^0} + \mathbb{A}.$$

We can view this automaton as an alternating automaton, which recognises the same language, as follows. All states are made existential, except the accepting state, which is universal. The automaton has the same transitions as in Example 2, except that we

remove the outgoing transitions from the accepting state:

$$\begin{aligned} &\text{initial} \xrightarrow{a} \text{initial} \\ &\text{initial} \xrightarrow{a} a \\ &a \xrightarrow{b} \begin{cases} \text{accept} & \text{if } a = b \\ a & \text{if } a \neq b. \end{cases} \end{aligned}$$

Since the accepting state is universal, and it has no outgoing transitions, the universal player immediately loses upon reaching this state, and therefore this state accepts all words. The remaining states of the automaton are existential, and therefore in order to win the game, the existential player must ensure that the accepting state is reached. This can only be done by finding a second appearance of some atom, as was the case in Example 2. □

Example 15. [All letters distinct] We now complement the alternating automaton from the previous example. This automaton has no ε-transitions, which means that the corresponding game is played in a finite number of rounds, with each round consuming an input letter. For such automata, the role of the two players is completely symmetric (the only asymmetry in the model would arise for infinite plays, which were arbitrarily chosen to be winning for the existential player). Thanks to this symmetry, we can swap the two players, which means that for each input word, the winner turns into the loser and vice versa. After this swap, the automaton recognises the complement of the language. In the case of the automaton from Example 15, the complement language is the set of words where all letters are distinct. □

Example 16. [The empty word] Here is an alternating automaton that accepts only the empty word. There are two equivariant states: which are called "empty" and $\bot$. Formally the state space is $1 + 1$, with the two components corresponding to the two states. The initial state is "empty". The state $\bot$ is owned by the existential player, and has no outgoing transitions. This means that it rejects all words (it is dual to the accepting state from Example 14). The state "empty" is owned by the universal player, and it enables all transitions

$$\text{empty} \xrightarrow{a} \bot \qquad \text{with } a \in \Sigma.$$

If we run the automaton on an empty input word, then the universal player cannot pick a transition from the initial state, and thus the existential player wins immediately, witnessing acceptance. Otherwise, if the input word is nonempty, then the universal player can pick a transition which goes to the state $\bot$, where the existential player has no choice, and therefore loses, witnessing rejection. □

Theorem 3.2. *Alternating pof automata generalise nondeterministic pof automata.*

Proof. Take a nondeterministic pof automaton (without ε-transitions), and interpret it as an alternating automaton, by making all states existential. For the moment, this construction is incorrect. This is because when the input word ends, the existential

player will lose, for a lack of possible transitions. Therefore, this automaton will reject all words. Another problem with this construction is that it ignores the accepting states of the original nondeterministic automaton.

Let us fix this construction, by taking into account the accepting states. We add a copy of the automaton for the empty word from Example 16, which has two states "empty" and $\bot$. For each transition (p, a, q), in the original nondeterministic automaton, if q is an accepting state, then we add a new transition (p, a, empty). The idea is that the existential player guess that the input word will end in a moment, leading to acceptance. Therefore, the automaton can enter state that will only accept the empty word. □

One of the principal motivations behind alternating automata is that Boolean operations have natural constructions. For intersection and union, we take the disjoint union of two automata, and combine them by adding a new initial state. In the new initial state, the owner of the state chooses the initial state of the two original automata, and goes there via an ε-transition. If the new initial state is existential, then the construction gives us the union, and if it is universal, then we get the intersection. The more interesting construction concerns complementation. For this construction, we make another assumption on the model, which disallows infinite sequences of ε-transitions.

Definition 3.3 (Well-founded)**.** An alternating automaton is called well-founded if there is no infinite sequence of ε-transitions

$$q_1 \xrightarrow{\varepsilon} q_2 \xrightarrow{\varepsilon} q_3 \xrightarrow{\varepsilon} \dots$$

Theorem 3.4. *Well-founded alternating pof automata are closed under complementation.*

Proof. The well-foundedness assumption ensures that the model is symmetric, since every play in the game is finite. Therefore, the automaton is complemented by swapping the two players, as we did in Example 20. □

One could extend the complementation construction to general alternating automata, if we extended the model to be more self-symmetric. This could be achieved, for example, by using a parity condition for infinite runs. However, we do not discuss such extensions in more detail, since already without such extensions the model is undecidable for essentially any problem, such as emptiness or universality. In light of this undecidability, there seems to be little motivation for seriously studying the model, as opposed to studying general Turing machines.

Exercises

Exercise 44. Show that in the atom-less case, i.e. when the states and input alphabet have dimension zero, alternating automata recognise exactly the regular languages.

Exercise 45. Show that emptiness is undecidable for alternating pof automata.

Exercise 46. Show that emptiness continues to be undecidable for alternating pof automata even if we require the state space to have dimension one, i.e. each state stores at most one atom.

Exercise 47. Show that emptiness becomes decidable for alternating pof automata if we require the state space to have dimension one, and the automaton must be non-guessing.

Exercise 48. Show that the non-guessing alternating pof automata are strictly weaker than the general model.

3.2 Two-way automata

We now describe a second extension of finite automata, which is equivalent to the usual automata in the atom-free case, but not in the presence of atoms. This is a two-way automaton, where the head can move both left and right. This model is the same as Turing machines that have a read-only input tape and no work tape. We will consider the pof variant of this model, in the deterministic case. A *deterministic two-way pof automaton* is defined like a deterministic pof automaton, except that the transition function is of type

$$Q \times \underbrace{(\Sigma + \{\vdash, \dashv\})}_{\text{input letters or endmarkers}} \to \{\text{accept}, \text{reject}\} + (Q \times \underbrace{\{\text{left}, \text{stay}, \text{right}\}}_{\text{head movement}}).$$

The automaton can also reject by entering an infinite loop.

Exercises

Exercise 49. Show that in the atom-less case, i.e. when the states and input alphabet have atom dimension zero, this model recognises exactly the regular languages.

Exercise 50. Suppose that atoms are names, which can be written using the Latin alphabet. The *atomless representation* of an element in a pof set is the string over the finite alphabet, which is obtained from the Latin alphabet by adding letters for brackets and commas, that is obtained by writing out each atom as a string. For example the triple

$$(\text{John}, \text{Eve}, \text{John}) \in \mathbb{A}^3$$

has an atomless representation of 15 letters, where the first letter is an opening bracket and the second letter is J. The atomless representation extends to words over a pof alphabet. Show that for every two-way pof automaton, the set

$$\{ \text{ atomless representation of } w \mid \text{ the automaton accepts } w \ \}$$

is in the complexity class L, i.e. deterministic logarithmic space.

Exercise 51. Consider the nondeterministic variant of the previous exercise. Show that the language of atomless representations is in NL, i.e. nondeterministic logarithmic space, and it can be complete for that class[1].

Exercise 52. Find a deterministic two-way register automaton which recognises the language

$$\{a_1 \cdots a_n : a_1, \ldots, a_n \text{ are distinct and } n \text{ is a prime number}\}.$$

[1] A corollary of Exercises 50 and 51 is that

$$\text{pof two-way automata determinise} \quad \Rightarrow \quad \text{NL} = \text{L}.$$

It is likely that the assumption is false, but no proof is known as of this time.

Exercise 53. Consider nondeterministic two-way register automaton $\mathcal{A}$ with one register and labels Σ. Show that the following language is regular (in the usual sense, without data values):

$$\{b_1 \cdots b_n \in \Sigma^* \quad : \quad \mathcal{A} \text{ accepts } (b_1, a_1) \cdots (b_n, a_n) \text{ for some distinct atoms } a_1, \ldots, a_n \in \mathbb{A}\}.$$

Exercise 54. Show that for every nondeterministic two-way pof automaton, there is an equivalent (one-way) alternating pof automaton with ε-transitions.

Exercise 55. Show a language that is two-way deterministic, also one-way nondeterministic, but not one-way alternating without guessing.

Exercise 56. Show a language that is one-way nondeterministic and one-way alternating without guessing, but not two-way deterministic, possibly assuming conjectures about complexity classes being distinct.

3.3 Circuits

In this group of problems, we consider the pof version of circuits. A pof circuit consists of:

1. a pof set X of variables;
2. a pof directed acyclic graph whose vertices are called *gates*;
3. a distinguished output gate;
4. an equivariant labelling from gates to $X + \{\vee, \wedge\}$.

Given a valuation of the variables $X \to \{\text{true}, \text{false}\}$, the circuit computes a value in the natural way, which is the value of the output gate.

Exercises

Exercise 57. Show that satisfiability is undecidable for pof circuits.

Exercise 58. A circuit is called a *formula* if the directed acyclic graph is a tree. Show that every pof circuit can be transformed into an equivalent formula.

Exercise 59. A pof circuit is said to be in DNF form if the root gate is a disjunction, its children are conjunctions, and their children are variables or their negations. Show that satisfiability is decidable for circuits in DNF form.

Exercise 60. CNF normal form is defined dually to DNF normal form. Show that not every pof DNF circuit can be transformed into an equivalent pof CNF circuit.

3.4 Pushdown automata and context-free grammars

In this section, we discuss pof variants of pushdown automata[2] and context-free grammars. We show that basic results, such as equivalence of pushdown automata and

[2] Context-free languages for infinite alphabets were originally introduced by Cheng and Kaminski (1998), who proved equivalence for register extensions of context-free grammars and pushdown automata. The generalisation to orbit-finite pushdown automata and context-free grammars is from Bojańczyk et al. (2014). See also Murawski et al. (2014); Clemente and Lasota (2015a,b).

context-free grammars, or decidability of emptiness, transfer easily to the pof setting. We also motivate the models by giving examples of automata and grammars that use atoms.

Definition 3.5. A *pof pushdown automaton* consists of

$$\underbrace{Q}_{\text{states}} \quad \underbrace{\Sigma}_{\substack{\text{input}\\\text{alphabet}}} \quad \underbrace{\Gamma}_{\substack{\text{stack}\\\text{alphabet}}} \quad \underbrace{q_0 \in Q}_{\text{initial state}} \quad \underbrace{\gamma_0 \in \Gamma}_{\substack{\text{initial stack}\\\text{symbol}}},$$

such that the initial state and initial stack symbol are equivariant, together with an equivariant transition relation

$$\delta \quad \subseteq \quad Q \times \overbrace{\Gamma^*}^{\text{popped}} \times \overbrace{(\Sigma \cup \epsilon)}^{\text{input}} \times Q \times \overbrace{\Gamma^*}^{\text{pushed}}$$

such that the popped and pushed strings have bounded length.

The language recognised by such an automaton is defined in the usual way. We assume that the automaton accepts via empty stack, i.e. a run is accepting if the last configuration (state, stack contents) has an empty stack.

Similarly, we can define a pof pushdown grammar.

Definition 3.6. A *pof context-free grammar* consists of

$$\underbrace{N}_{\text{nonterminals}} \quad \underbrace{\Sigma}_{\substack{\text{input}\\\text{alphabet}}} \quad \underbrace{R \subseteq N \times (N + \Sigma)^*}_{\text{rules}} \quad \underbrace{S \in N}_{\substack{\text{initial}\\\text{nonterminal}}}$$

where the nonterminals and input alphabet are pof sets, the set of rules is equivariant and has bounded length, and the initial nonterminal is equivariant.

The language generated by a grammar is defined in the usual way.

Example 17. [Pushdown automaton for palindromes.] For a pof alphabet Σ, consider the language of palindromes, i.e. words which are equal to their reverse. This language is recognised by a pof pushdown automaton which works exactly the same way as the usual automaton for palindromes, with the only difference being that the stack alphabet Γ is now a pof set, namely Σ. For instance, in the case when $\Sigma = \mathbb{A}$, the automaton keeps a stack of atoms during its computation. The automaton has two control states: one for the first half of the input word, and one for the second half of the input word. As in the standard automaton for palindromes, this automaton uses nondeterminism to guess the middle of the word. □

Example 18. [Pushdown automaton for modified palindromes.] The automaton in Example 17 had two control states, which did not store any atoms. In some cases, it might be useful to have a set Q of control states that uses atoms. Consider the set of odd-length palindromes where the middle letter is equal to the first letter. A natural automaton recognising this language would be similar to the automaton for palindromes, except that it would store the first letter a_1 in its control state.

Another solution would be an automaton which keeps the first letter in every token on the stack. This automaton has a stack alphabet of $\Gamma = \Sigma \times \Sigma$, and after reading letters $a_1 \cdots a_n$ its stack is

$$(a_1, a_1), (a_1, a_2), \ldots, (a_1, a_n).$$

This automaton needs only two control states. Actually, using the standard construction, one can show that every orbit-finite pushdown automaton can be converted into one that has one control state, but a larger stack alphabet. □

The following example gives some motivation for studying orbit-finite pushdown automata.

Example 19. [Modelling recursive programs] Pushdown automata without atoms are sometimes used to model the behaviour of recursive programs with Boolean variables. By adding atoms, we can also model programs that have variables ranging over atoms. Consider a recursive function such as the following one (this program does not do anything smart):

```
function f(a: atom)
begin
    b:=read() // read an atom from the input
    if b != a then
        f(b)
        if b != read() then
            fail() // terminate the computation
end
```

The behaviour of this program can be modelled by a pof pushdown automaton. The input tape corresponds to the `read()` functions. The stack corresponds to the call stack of the recursive functions; the stack stores atoms since the functions take atoms as parameters. Since the only variables are atoms, the set of possible call frames (i.e. the stack alphabet) is a pof set. Pof pushdown automata could also be used to model more sophisticated behaviour, including mutually recursive functions and boolean variables. □

Theorem 3.7. *Pushdown automata recognise the same languages as context-free grammars. Furthermore, emptiness is decidable.*

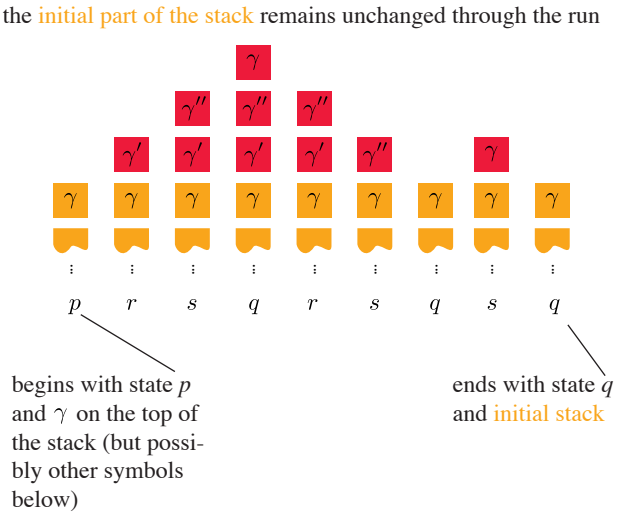
Proof. We just redo the classical constructions, which are so natural that they easily go through in the pof extension.

- *From a pushdown automaton to a context-free grammar.* Without loss of generality, we assume that each transition either: pops nothing and pushes one symbol; or pops one symbol and pushes nothing. We also assume that in every accepting run, the stack is nonempty until the last configuration. Every pushdown automaton can be transformed into one of this form, without changing the recognised language, by using additional states and ε-transitions. The transformation can be done in polynomial time, assuming that equivariant subsets are represented using formulas.

Assuming that the pushdown automaton has the form discussed above, the corresponding grammar is defined as follows. The nonterminals are

$$N \quad = \quad \underbrace{\{S\}}_{\text{an initial nonterminal}} \quad + \quad Q \times \Gamma \times Q.$$

The language generated by a nonterminal (p, γ, q) is going to be the set of words which label runs of the following form:



To describe these runs, we use the following grammar rules. All the sets below are equivariant and have bounded length:

1. *Transitive closure.* For every $p, q, r \in Q$ and $\gamma \in \Gamma$, there is a rule

$$(p, \gamma, q) \to (p, \gamma, r)(r, \gamma, q).$$

2. *Push-pop.* For every pair of transitions

$$\underbrace{(p, \epsilon, a, p', \gamma')}_{\text{push}} \quad \text{and} \quad \underbrace{(q', \gamma', b, q, \epsilon)}_{\text{pop}}$$

there is a rule

$$(p, \gamma, q) \to a(p', \gamma', q')b.$$

3. *Starting.* For every transition that pops the initial stack symbol γ_0

$$\underbrace{(p, \gamma_0, a, q, \epsilon)}_{\text{pop}}$$

there is a rule

$$S \to (q_0, \gamma_0, p)a.$$

- *From a context-free grammar to a pushdown automaton.* The automaton keeps a stack of nonterminals. It begins with just the starting nonterminal, and accepts when all nonterminals have been used up. In a single transition, it replaces the nonterminal on top of the stack by the result of applying a rule. This automaton has one state (if we disregard the restriction that all transitions have to be either push or pop).
- *Emptiness is decidable.* We now show that emptiness is decidable. We use the context-free grammars, and the usual algorithm. This algorithm stores an equivariant subset of the nonterminals that are known to be nonempty (also known as productive nonterminals). Initially, the subset is empty. In each step, we add a nonterminal X to the subset if there is some rule in the grammar, where the left-hand side has X, and the right-hand side has only terminals and nonterminals that are already in the subsets. Because the set of rules is equivariant, in each step the subset is equivariant. Therefore, the subset can grow only in finitely many steps before stabilizing. The number of steps is at most the number of orbits in the set of nonterminals, which is at most exponential in the representation of the grammar.

□

Exercises

Exercise 61. Consider the following extension[3] of pof pushdown automata, where a new kind of transition is allowed:

$$q \xrightarrow{\text{fresh}(a)} p \qquad \text{for states } p, q \text{ and an input letter } a \in \mathbb{A}.$$

When executing this transition, the automaton reads letter a and changes state from q to p, but only under the condition that all atoms from the input letter a are fresh (i.e. do not appear in) with respect to every letter on the stack and the current state q. Show that emptiness is decidable.

Exercise 62. Consider the following higher-order variant of orbit-finite pushdown automata[4]. The automaton has a stack of stacks (one could also consider stacks of stacks of stacks, etc., but this exercise is about stacks of stacks). There are operations as in a usual pushdown automaton, which apply to the topmost stack. There is also an operation "duplicate the topmost stack" and an operation "delete the topmost stack". Show that emptiness is undecidable.

Exercise 63. Show a language that is generated by a pof context-free grammar, but not by any pof context-free grammar with a finite (not just pof) set of nonterminals.

Exercise 64. Show that emptiness for pof context-free grammars is ExpTime-complete.

Exercise 65. Show that if the set of terminals (i.e. the input alphabet), is finite (i.e. pof of dimension zero), then pof context-free grammars are the same as usual context-free grammars.

Exercise 66. Show that pof context-free grammars can be converted into Chomsky normal form, where all rules are of the form $X \to YZ$ with X, Y, Z nonterminals, or $X \to a$ with X a nonterminal and a a terminal.

[3] This extension is based on Murawski et al. (2014).

[4] This exercise is based on (Murawski et al., 2014, Section 6).

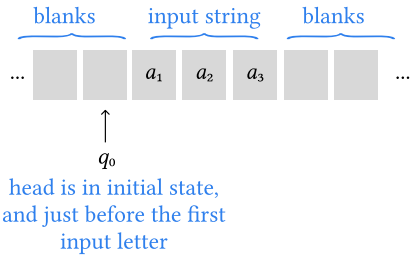
3.5 Turing machines

In this section, we discuss the pof version of Turing machines. A pof Turing machine is defined like a Turing machine, except that the set of states, and the alphabets are pof sets, and the transition function is equivariant.

We assume that the reader is familiar with Turing machines, but we give a more detailed description of our modal to fix notation. The input alphabet Σ, the work alphabet Γ, and the set of states Q are all pof sets. We assume that the work alphabet contains the input alphabet, and there is some designated blank symbol

$$\text{blank} \in \Gamma \setminus \Sigma$$

that is equivariant. One could have a two tape model, but since we will not be interested in machines with sublinear space (e.g. logarithmic space), we use the one tape model for simplicity. In this model, there is one tape that is read-write, which initially contains the input string, and which is also used for storing intermediate computations. The tape is infinite in both directions. A configuration of the Turing machine consists of the tape contents (i.e. each cell has some letter from the work alphabet), a head position (which points to some cell), and a state from Q. The initial configuration looks like this:



The behaviour of the Turing machine is specified by its transition function, which is an equivariant function of type

$$\underbrace{Q \times \Gamma}_{\substack{\text{current state and}\\ \text{letter under}\\ \text{the head}}} \quad \to \quad \{\text{accept, reject}\} + (Q \times \underbrace{\{\text{left, stay, right}\}}_{\text{head movement}} \times \underbrace{\Gamma}_{\substack{\text{what is}\\ \text{written on}\\ \text{the tape}}}).$$

Using the transition function, the machine computes a new configuration in the expected way, or it accepts/rejects. This leads to a computation (a sequence of configurations), which is either finite – when an accept/reject instruction is executed – or infinite. In a nondeterministic machine, instead of a function we have a binary relation, and an input string might have more than one computation. The language *recognised by* a (possibly nondeterministic) Turing machine is the set of words that have at least one accepting computation.

Example 20. [A Turing machine checking that all letters are different] Consider the equality atoms. Assume that the input alphabet is $\mathbb{A}$. We show a deterministic

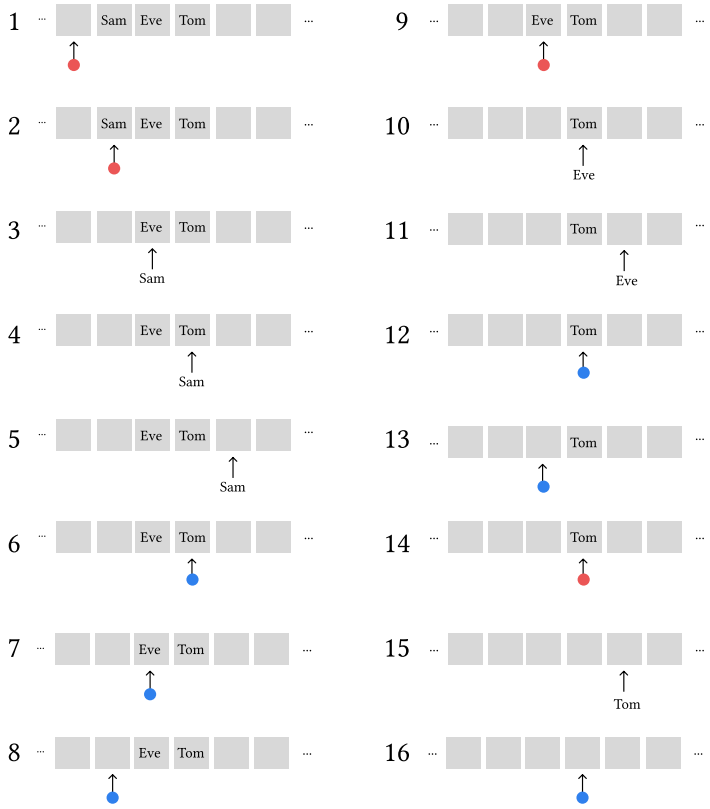


Figure 3.1: An accepting run of the Turing machine from Example 20.

Turing machine which accepts words where all letters are distinct. The idea is that the machine iterates the following procedure until the tape contains only blank symbols: if the first non-blank letter on the tape is a, replace it by a blank and load a into the state, scan the word to check that a does not appear again (if it does appear again, then reject immediately), and after reading the entire word go back to the beginning of the tape. If the tape is entirely erased, then accept. The sets of states is $\mathbb{A}$, plus two extra states for the scanning, which are depicted using red and blue in Figure 3.1. □

Having defined Turing machines, we get the usual notions of semi-decidability (the language of some Turing machine) and decidability (the language of some Turing machine that always halts). The Church-Turing Thesis states that there is only one notion of decidable language, which is captured by Turing machines. Does introducing atoms give a violation of this thesis? What does that even mean? One way of answering this question is to relate computation with atoms to the classical notion of computation without atoms. A word with atoms can be represented by a word

without atoms, by writing down the atoms, such as "John" or "Mary" using a finite alphabet. Under such a representation, we get a usual word over a finite alphabet, which can be used as an input for the classical atom-free models of computation. We will show later in this section that Turing machines with atoms can be simulated by machines without atoms, and vice versa, and thus the two models of computation are essentially equivalent. Using this equivalence, we can carry over to the atom world classical results, such as equivalence of deterministic and nondeterministic machines in the presence of unbounded computation time. However, in Chapter 9 we will discover a twist in the story – if we use a more general notion of pof sets, namely (not necessarily polynomial) orbit-finite sets, then some equivalences break down, for example nondeterministic machines are not equivalent to deterministic ones. Before we get to the twist, let us tell the untwisted story, which involves pof sets.

We begin by formalizing what it means to "write down" an atom.

Definition 3.8. A *representation* of the atoms is any function

$$r : 2^* \to \mathbb{A}$$

which is surjective (every atom has at least one representation) and such that one can decide if two strings represent the same atom.

An alternative choice of definition would require the function to be bijective, which would also give a simpler algorithm for deciding if two strings represent the same atom. We choose to use the above definition because it will more naturally extend to atoms with more structure.

Suppose that we have a representation of the atoms. We can extend it to represent elements of a pof set: an element of such a set is described by indicating which component $\mathbb{A}^{d_i}$ is used, followed by a representation of the d_i atoms in the tuple. We can also extend the representation to describes words over a pof set, by using separator symbols between the letters. Summing up, once we know how to represent atoms with atom-less strings, we can do the same for words over a pof alphabet. In the following theorem, we show that the atom version of Turing machines correspond to the usual Turing machines without atoms, via the representation. Furthermore, the choice of representation is not important.

Theorem 3.9. *The following conditions are equivalent for every language $L \subseteq \Sigma^*$ over a pof alphabet:*

1. *L is recognised by a deterministic pof Turing machine;*
2. *L is recognised by a nondeterministic pof Turing machine;*
3. *L is equivariant and for every representation r,*

$$\{ w \mid \text{ } w \text{ represents, under } r, \text{ some word in } L \text{ } \}$$

   is recognised by a nondeterministic Turing machine;

4. *as in the previous item, but the machine is deterministic;*

5. *as in the previous item, but the representation r is quantified existentially.*

Proof. The implications 1 ⇒ 2 and 4 ⇒ 5 are trivial. For the implication 2 ⇒ 3, we use a straightforward simulation, where the simulating machine stores representations of the simulated Turing machine. Implication 3 ⇒ 4 is the classical fact that, without atoms, deterministic and nondeterministic Turing machines compute the same languages. The interesting implication is 5 ⇒ 1, which is proved below.

Let r be a representation as in the assumption 5, and let us write $s : 2^* \to \Sigma^*$ for the extension of this representation to words over the alphabet Σ. The main idea is that this representation can be inverted, up to atom permutations, by a deterministic pof Turing machine. This is proved in the following lemma, which we call the deatomisation lemma, because it transforms a word with atoms into a representation without atoms. (We use the standard notion of Turing machines for computing a function – there is an output tape, the machine always halts, and the contents of the output tape is the output of the function.)

Lemma 3.10 (Deatomisation)**.** *There is a function $f : \Sigma^* \to 2^*$, computed by a deterministic pof Turing machine, such that every word $w \in \Sigma^*$ is in the same orbit as $s(f(w))$.*

Before proving the above lemma, we use it to prove the implication 5 ⇒ 1. Using the atom-less Turing machine from the assumption, we know that there is a Turing machine that in puts $w \in \Sigma^*$, and checks if $s(f(w))$ belongs to the language. By the assumption that the language is equivariant, this is the same as checking if w belongs to the language. It remains to prove the Deatomisation Lemma.

Proof. Consider some computable enumeration of representations of the atoms, i.e. an infinite list of strings in 2^* which is computed by a Turing machine, and such that every atom is represented by exactly one string on the list. Such an enumeration can be found for every representation.

Using this enumeration, we define the deatomisation function f from the statement of the lemma. Consider an input string $w \in \Sigma^*$. The string w contains some atoms, and these atoms can be listed in the order of their first appearance in the string. For each of these atoms, we choose a string representation according to the enumeration in the previous paragraph, i.e. the atom with the leftmost appearance gets the first representation, the atom with the second leftmost appearance gets the second representation, and so on. We then apply this choice consistently to the entire string. All of this can be implemented by a deterministic pof Turing machine. □

This completes the proof of the implication 5 ⇒ 1, and therefore also of the theorem. We would like to remark that the proof of the Deatomisation Lemma given above will fail for more general input alphabets which will be considered later in the book. The issue is that the proof above refers to the order of appearance of atoms in the input string, and this will no longer be meaningful for some input alphabets, such as unordered pairs of atoms, which will be legitimate alphabets in the more general settings. □

Exercises

Exercise 67. Give a deterministic pof Turing machine for the language

$$\{\, w\#v \mid \ w, v \in \mathbb{A}^* \text{ are in the same orbit } \}.$$

Exercise 68. Consider a two-tape model, which has a work tape with a separate head. Show that for every pof Turing machine, deterministic or not, there is an equivalent one (in the two-tape model) where the state space has atom dimension zero. (Attention: this will no longer be true for orbit-finite sets, which are not polynomial orbit-finite.)

Exercise 69. Show that the answer to the previous problem is negative in the one-tape model.

Chapter 4

Orbit-finite sets

So far, we have worked with polynomial orbit-finite sets, which had an elementary and concrete definition, but were enough to describe many interesting examples of automata, or other computational models. However, there are certain limitations of pof sets.

One limitation is that pof sets are not closed under taking equivariant subsets. An example is the set of non-repeating tuples

$$\mathbb{A}^{(d)} \quad \stackrel{\text{def}}{=} \quad \{\, (a_1, \ldots, a_d) \in \mathbb{A}^d \mid \ a_1, \ldots, a_d \text{ are pairwise different } \},$$

which is not a pof set. (This is an important example, since every orbit in a pof set is of this kind, up to an equivariant bijection.) Subsets will arise naturally in various contexts. For example, we might want to restrict the state space of an automaton the reachable states, and this will be impossible with pof sets.

Another limitation of pof sets is that they are not closed under taking quotients. An example is the set of unordered tuples

$$\binom{\mathbb{A}}{d} \quad \stackrel{\text{def}}{=} \quad \{\, \{a_1, \ldots, a_d\} \mid \ a_1, \ldots, a_d \text{ are pairwise different } \},$$

which can be seen as the quotient of the set $\mathbb{A}^{(d)}$ under the equivalence relation that identifies two tuples if they are the same up to a permutation of coordinates. Quotients will arise naturally in various contexts. For example, when minimising an automaton, we will want to identify two states if they accept the same words, and this will be impossible with pof sets.

In this chapter, we overcome these limitations, by introducing the notion of (not necessarily polynomial) orbit-finite sets. This notion is defined using the abstract language of group actions, which may seem at first glance to be excessively formalistic. However, as we will see later in this book, the added generality has many advantages. Also, the same notion can be described in more concrete terms, such as the closure of pof sets under equivariant subsets and quotients.

Minimization of automata. The motivating example is minimization of deterministic automata. Suppose that we want to minimise a deterministic pof automaton. The classical construction restricts the state space to the subset of reachable states, and then quotient this subset under the Myhill-Nerode equivalence relation

$$q \sim p \qquad \stackrel{\text{def}}{=} \qquad qw \in F \Leftrightarrow pw \in F \text{ for every word } w \in \Sigma^*.$$

As we have explained at the beginning of this chapter, we cannot apply this construction for pof automata, since pof sets are not closed under taking subsets, or quotients. This is illustrated in the following example.

Example 21. Consider the language

$$L = \{ w \in \mathbb{A}^* \mid \ w \text{ uses at most two different atoms } \}.$$

This language is recognised by a pof automaton, which has a state space

$$\underbrace{\mathbb{A}^0 + \mathbb{A}^1 + \mathbb{A}^2}_{\text{atoms seen so far}} \quad + \quad \underbrace{\mathbb{A}^0}_{\text{reject}}.$$

This automaton is not minimal, because the states from $\mathbb{A}^2$ store the order in a pair (a, b), which is not needed for the language. To make this automaton minimal, we should use a state space of the form

$$\mathbb{A}^0 + \mathbb{A}^1 + \underbrace{\binom{\mathbb{A}}{2}}_{\substack{\text{sets of exactly two atoms, as described} \\ \text{at the beginning of this chapter}}} + \mathbb{A}^0$$

This kind of feature is not available in pof automata, and therefore the minimal automaton for this language cannot be a pof automaton. More formally, we will show that for every deterministic pof automaton for this language, the states after reading the input words ab and ba must be different (while they should be equal in a minimal automaton). Indeed, if the same state q would be reached after reading both ab and ba, then this state would satisfy

$$\pi(q) = q \qquad \text{where } \pi \text{ is the atom permutation that swaps } a \text{ and } b.$$

If an element in a pof set satisfies the above condition, then it cannot use the atoms a and b. This cannot happen in an automaton that recognises the language. □

If we would like our automata to be closed under minimization, then they should support taking subsets and quotients. A quick and dirty solution is to simply add these two features. We will describe this solution in Section 4.1. Later in this chapter, we will show that the solution is not so dirty after all, and it can be equivalently described in a more abstract way, using the language of group actions. Later in the book, other ways of describing the notion will also appear, namely the representable sets from Chapter 10, thus inspiring confidence that the notion is well motivated.

Exercises

Exercise 70. Define the *orbit count* of a deterministic pof automaton to be the number of orbits of reachable states. For a language, we can consider the set of deterministic pof automata that recognise it, and have minimal orbit count for this property. Show that this set can contain automata that are non-isomorphic. Here, an isomorphism between two automata is an equivariant bijection

$$\text{reachable states of } \mathcal{A} \quad \xrightarrow{f} \quad \text{reachable states of } \mathcal{B}$$

such that for every input word, applying f to the state of $\mathcal{A}$ after reading a word gives the state of $\mathcal{B}$ after reading the same word.

4.1 Subquotiented pof sets

To formalize subsets and quotients, we use partial equivalence relations. A *partial equivalence relation* on a set X is defined to be a binary relation that satisfies

$$\underbrace{x \sim y \quad \Rightarrow \quad y \sim x}_{\text{symmetric}} \quad \text{and} \quad \underbrace{x \sim y \text{ and } y \sim z \quad \Rightarrow \quad x \sim z}_{\text{transitive}}.$$

This is like an equivalence relation, but the missing axiom is reflexivity $x \sim x$. Defining a partial equivalence relation is the same as indicating some subset (the elements that are equivalent to themselves), and then defining a (total) equivalence relation on the subset. Therefore, partial equivalence relations subsume both subsets and quotients. We write $X_{/\sim}$ for the set of equivalence classes of $\sim$, and we call this a *subquotient* of the original set.

Definition 4.1. A *subquotiented pof set*, spof for short, is any subquotient $X_{/\sim}$ of a pof set X by a partial equivalence relation $\sim$ that is equivariant.

Example 22. Subquotiented pof sets subsume both subsets and quotients. Let us begin by illustrating subsets. Every equivariant subset $Y \subseteq X$, can be seen as a subquotiented pof set, which corresponds to the partial equivalence relation

$$x \sim y \quad \text{if} \quad x = y \text{ and } x \in Y.$$

One example of such a set is the set

$$\mathbb{A}^{(d)} \stackrel{\text{def}}{=} \{\, (a_1, \ldots, a_d) \in \mathbb{A}^d \mid \ a_1, \ldots, a_d \text{ are pairwise different } \,\},$$

which we call the *set of non-repeating tuples*. We will use these sets a lot. □

Example 23. In Example 21, we discussed the set

$$\binom{\mathbb{A}}{2}.$$

This is an example of a subquotiented pof set. It is the subquotient of $\mathbb{A}^2$ under the partial equivalence relation defined by

$$(a, b) \sim (a', b') \quad \text{if} \quad \{a, b\} = \{a', b'\} \wedge a \neq b.$$

□

Similarly to the case of pof sets, all structure on subquotiented pof sets will be required to be equivariant. Equivariance is defined in the same way as for subsets of pof sets: membership in the set must be stable under applying atom permutations. Let us define this more formally.

Definition 4.2. [Equivariant subset of a spof set] A subset

$$Y \subseteq X_{/\sim}$$

of a subquotiented pof set is called *equivariant* if for every element of $X_{/\sim}$, which is an equivalence class $[x]_\sim$ of some element $x \in X$, we have

$$[x]_\sim \in Y \quad \Leftrightarrow \quad \underbrace{\pi([x]_\sim)}_{\substack{\text{image of } \pi, \text{ when applied to the set} \\ \text{of elements that are in the equivalence class } [x]}} \in Y \quad \text{for every atom permutation } \pi.$$

In the above definition, when we apply an atom permutation to an element of an equivalence class, we take the image of all elements in the equivalence class. An alternative approach would be to apply the atom permutation to a representative of the equivalence class, and then take the equivalence class of the image. This would give the same effect, since applying atom permutations commutes with taking equivalence classes, as long as $\sim$ is equivariant:

$$\underbrace{\pi([x]_\sim)}_{\substack{\text{first take the equivalence class,} \\ \text{then apply the atom permutation}}} = \underbrace{[\pi(x)]_\sim}_{\substack{\text{first apply the atom permutation,} \\ \text{then take the equivalence class}}}.$$

Example 24. Consider the subquotiented pof set

$$\binom{\mathbb{A}}{2}$$

from Example 23, which is obtained by a quotient of $X = \mathbb{A}^2$ with respect to a certain partial equivalence relation $\sim$. An element of the subquotient is an equivalence class of pairs, as in the following example:

$$\underbrace{(\text{John}, \text{Mary})}_{\text{element } x \in X} \qquad \underbrace{\{(\text{John}, \text{Mary}), (\text{Mary}, \text{John})\}}_{\text{its equivalence class in } X_{/\sim}}. \tag{4.1}$$

If we apply to the above equivalence class the atom permutation that swaps John and Mary, then we get the same equivalence class. □

In Definition 4.2, we defined equivariance for subsets of subquotiented pof sets. Since subquotient pof sets are closed under taking products $X \times Y$, we can also talk about equivariant relations on subquotient pof sets, by considering equivariant subsets of the product $X \times Y$. As a special case of relations, we can discuss equivariant functions between subquotient pof sets.

Example 25. We will show that there is no equivariant function

$$f : \binom{\mathbb{A}}{2} \to \mathbb{A}.$$

In this proof, we treat elements of $\binom{\mathbb{A}}{2}$ as sets $\{a, b\}$ of size two, although formally they are defined to be equivalence classes of ordered pairs. Consider some hypothetical function f, and some input-output pair

$$f(\{a, b\}) = c.$$

We first rule out the case that $c \notin \{a, b\}$. If this were the case, then we could apply an atom permutation to the input-output pair that moves c without moving a and b, and get a violation of functionality. Let us now rule out the case $c \in \{a, b\}$. If this were the case, then we could apply an atom permutation that swaps a and b; this atom permutation would not change the input, but it would change the output, and so it would also be a violation of functionality. □

Subquotiented pof sets are a solution to the problem of minimization of deterministic pof automata. Indeed, if we have some pof automaton, then we can define a partial equivalence relation on its states by

$$p \sim q \quad \stackrel{\text{def}}{=} \quad p \text{ and } q \text{ are both reachable, and accept the same words.}$$

By equivariance of the automaton, this is an equivariant partial equivalence relation. Therefore, the quotient $Q_{/\sim}$ is a subquotiented pof set. As usual, one can define a quotient automaton $\mathcal{A}_{/\sim}$ which recognises the same language as the original automaton $\mathcal{A}$. This automaton is well-defined (i.e. its structure is equivariant), and it is minimal in the appropriate sense. More details will be provided later in this chapter, when we prove Myhill-Nerode Theorem.

Exercises

Exercise 71. How many equivariant functions are there of type

$$\binom{\mathbb{A}}{d} \to \binom{\mathbb{A}}{e}$$

for $d, e \in \{0, 1, \ldots\}$?

Exercise 72. Find a deterministic spof automaton for the language

$$\{ w \in \binom{\mathbb{A}}{3}^* \mid \text{ some atom } a \text{ appears in all letters } \}.$$

Exercise 73. Find a deterministic spof automaton for the language

$$\{ w \in \binom{\mathbb{A}}{3}^* \mid \text{there are at most distinct 5 atoms used in the word} \}.$$

Exercise 74. Find a deterministic spof automaton for the language

$$\{ w \in \binom{\mathbb{A}}{3}^* \mid \begin{array}{l}\text{some atom from the set in the first letter appears} \\ \text{an even number of times in the remaining letters}\end{array} \}.$$

Exercise 75. Find a deterministic spof automaton for the language:

$$\{ w \in \binom{\mathbb{A}}{2}^* \mid \text{there exist } a, b \in \mathbb{A} \text{ such that every letter in } w \text{ intersects } \{a, b\} \}.$$

Exercise 76. Consider a spof group, i.e. the underlying set is a spof, and the group operation is equivariant. Show that such a group must be finite.

Exercise 77. Let X be a spof. Show that if $f : X \to X$ is an equivariant surjective function, then f is a bijection.

Exercise 78. Consider a chain

$$X_0 \overset{f_1}{\twoheadrightarrow} X_1 \overset{f_2}{\twoheadrightarrow} X_2 \overset{f_3}{\twoheadrightarrow} \dots \overset{f_n}{\twoheadrightarrow} X_n$$

of equivariant surjective function between spof sets. Show that the length of this chain is bounded by a polynomial of the following two parameters of the first set X_0: the orbit count, and the atom dimension.

4.2 Orbit-finiteness

One could worry that subquotient pof sets are a hack that fixes the minimization issue for automata, but does not have other applications, or a proper theoretical justification. In this section, we ease such worries, by giving a more semantic concept, namely orbit-finite sets, and showing that they are exactly the same as subquotient pof sets.

Group actions. The semantic characterization will use two of the central notions in this book: finite supports, and orbit-finiteness. These notions are defined in terms of group actions, so begin by defining those.

Definition 4.3 (Group action)**.** An action of a *group* G on a set X is defined to be a function

$$G \times X \to X,$$

which we denote by

$$(\pi, x) \mapsto \pi(x),$$

that satisfies the following two axioms:

$$\underbrace{(\pi \circ \sigma)(x)}_{\substack{\text{compose in the group} \\ \text{and then apply the action}}} = \underbrace{\pi(\sigma(x))}_{\substack{\text{apply the action} \\ \text{two times}}} \quad \text{and} \quad \underbrace{1(x) = x}_{\substack{\text{the group identity} \\ \text{does not move anything}}}.$$

In this book, the group G will always consist of permutations of the atoms, although in later chapters will restrict the permutations to those that respect some extra structure on the atoms, such as a linear order.

Example 26. Sets constructed using atoms are naturally equipped with an action of the group of atom permutations. For example, the set $\mathbb{A}^d$ is equipped with the action defined by

$$\pi(a_1, \ldots, a_d) = (\pi(a_1), \ldots, \pi(a_d)).$$

We have been using this group action implicitly in the previous chapters. The action also extends to other sets, such as $\mathbb{A}^*$ or the powerset $\mathsf{P}\mathbb{A}$. In the case of the powerset, we use the image, i.e.

$$\pi(X) = \{\, \pi(a) \mid \ a \in X \ \}.$$

□

Finite supports. We now introduce the fundamental notion of supports. The general idea is that the support of an element x in a set X consists of the atoms that are needed to describe it. Since the notion is defined in abstract terms of group actions, it will be useful to keep the following examples in mind while reading the formal definition.

Set X	Element $x \in X$	Support of x
$\mathbb{A}^2$	(John, Mary)	John, Mary
$\mathsf{P}\mathbb{A}$	$\{\, a \in \mathbb{A} \mid \ a \neq \text{John} \ \}$	John
$\mathsf{P}\mathbb{A}$	$\mathbb{A}$	$\emptyset$

Definition 4.4 (Supports)**.** Consider a set X that is equipped with an action of the group of all atom permutations. An element $x \in X$ is said to be *supported* by a list of atoms $a_1, \ldots, a_n$ if

$$\underbrace{\pi(a_1) = a_1 \wedge \cdots \wedge \pi(a_n) = a_n}_{\text{we write this as } \pi(\bar{a}) = \bar{a}, \text{ where } \bar{a} \text{ is the list } a_1, \ldots, a_n} \quad \Rightarrow \quad \pi(x) = x \tag{4.2}$$

holds for every atom permutation π. We say that x is *finitely supported* if it is supported by some finite list of atoms.

Observe that in the above definition, the group of atom permutation acts on two sets. The first action is on atoms: an atom permutation can be applied to an atom, and the result is another atom. This is the action that is used in the assumption of

implication (4.2). The second action is the action on the set X, which is used in the conclusion of the implication. The second action is a parameter of the definition, i.e. in order for the definition to be meaningful, the set has to come with an action of the group of atom permutations. In most cases of interest, the second action will be clear from the context, e.g. if we take X to be $\mathbb{A}^d$, or the powerset $\mathsf{P}\mathbb{A}$, or any other set that is constructed using atoms in some natural way. However, in principle when talking about finite supports, we must remember that the notion depends on the action of the group of atom permutations on the set X.

Before continuing, let us remark on the notation. In the above definition, we use finite lists for supports. The order of atoms in this list, or their repetitions, are not relevant for the notion of support, since they do not affect the assumption of the implication (4.2). Therefore, the only relevant information is the set of atoms that appears on this list, which is why many authors present the support as a finite set of atoms, and not a list. If one uses sets $\{a_1, \ldots, a_n\}$ for supports, then one should remember that the assumption in implication (4.2) is not

$$\pi(\{a_1, \ldots, a_n\}) = \{a_1, \ldots, a_n\},$$

which is a weaker assumption, because it allows π to swap atoms inside the set.

Example 27. Let us discuss which elements of the powerset $\mathsf{P}\mathbb{A}$ are finitely supported. If $x \in \mathsf{P}\mathbb{A}$ is finite, then it is finitely supported, namely it is supported by any list that contains all atoms in this set. A similar result holds for co-finite sets, i.e. sets obtained by removing finitely many atoms. For example, if we take

$$x = \mathbb{A} \setminus \{\text{John}, \text{Mary}\},$$

then x is supported by the atoms John, Mary, because any atom permutation that fixes both John and Mary will map the set x to itself, even if it permutes the other atoms. Therefore, all finite and co-finite elements in $\mathsf{P}\mathbb{A}$ are finitely supported.

The remaining elements of the powerset are not finitely supported. Let us prove this formally. Suppose that $x \in \mathsf{P}\mathbb{A}$ is neither finite nor cofinite, and take some hypothetical finite support $\bar{a}$. There must be some atoms b, c that are not in this finite support, and such that $b \in x$ and $c \notin x$. Take the atom permutation that swaps b and c, and leaves all other atoms fixed. This permutation fixes the support, but moves x, which contradicts the definition of finite support. □

Orbits and orbit-finiteness. We now introduce the second fundamental notion of this book, which is orbit-finiteness. This idea was already discussed informally before, but we now give a formal definition in terms of group actions. Consider a set X equipped with an action of atom permutations. An *orbit* of X is defined to be any subset of the form

$$\{\, \pi(x) \mid \ \pi \text{ is an atom permutation} \ \},$$

for some $x \in X$. We will be interested in sets that have finitely many orbits, and where all elements are finitely supported. (In the exercises, we explain why these two conditions are necessary for the theory to work.)

Definition 4.5 (Orbit-finite set)**.** An *orbit-finite set* is defined to be a set X, together with an action of the group of atom permutations, such that every element $x \in X$ has finite support, and there are finitely many orbits.

In the above definition, we require two things: there are finitely many orbits, and every element has finite support. The name "orbit-finite" would seem to imply only the first condition. At the end of this section we discuss why the second condition is also included.

Example 28. [Pof sets are orbit-finite] Recall the set $\mathbb{A}^{(d)}$, which consists of non-repeating d-tuples of atoms. This set has one orbit, and every element has a support of size d, and therefore it is orbit-finite. Another example of an orbit-finite set, this time with more than one orbit, is $\mathbb{A}^3$. Like any orbit-finite set, this set decomposes into a disjoint union of one-orbit sets, which in this case uses five orbits:

$$\underbrace{\mathbb{A}^1}_{\substack{\text{all atoms}\\ \text{are equal}}} + \underbrace{\mathbb{A}^{(2)} + \mathbb{A}^{(2)} + \mathbb{A}^{(2)}}_{\substack{\text{two atoms are equal, and}\\ \text{one is diffferent, which}\\ \text{can happen in three ways}}} + \underbrace{\mathbb{A}^{(3)}}_{\substack{\text{all atoms}\\ \text{are different}}}.$$

More generally, any polynomially orbit-finite set is orbit-finite, which explains the name. □

Example 29. [Powerset is not orbit-finite] The powerset $\mathsf{P}\mathbb{A}$ is not orbit-finite, for two reasons. The first reason is that it contains elements (i.e. subsets of $\mathbb{A}$) which are not finitely supported. The second reason is that, even if we restricted the powerset to finitely supported element (the resulting variant of the powerset is called the *finitely supported powerset*), then it would still not be orbit-finite, because it would have infinitely many orbits. This is because sets of different finite size are in different orbits. □

Example 30. [Triples modulo cyclic shift] Consider the set $\mathbb{A}^{(3)}$ of non-repeating triples. On this set, consider the equivalence relation that identifies triples modulo cyclic shift:

$$(a, b, c) \sim (b, c, a) \sim (c, a, b) \quad \text{for all } a, b, c \in \mathbb{A}.$$

Consider the quotient of under this equivalence relation. This is a one-orbit set. It is also an example of a subquotiented pof set, since we can view $\sim$ as a partial equivalence relation on (possibly repeating) triples that removes the repeating triples. □

The positive examples we have seen above were examples of subquotient pof sets. This is not a coincidence, since as the following theorem asserts, these are the only possible orbit-finite sets.

Theorem 4.6. *Let X be a set equipped with an action of atom permutations. Then X is orbit-finite if and only if it admits an equivariant bijection with a subquotient pof set.*

Proof. The bottom-up implication is easy, so we only prove the top-down implication. If a set has finitely many orbits, then it is a disjoint union of finitely many one-orbit sets. Since subquotient pof sets are closed under finite disjoint union, it is enough to prove the top-down implication for a one-orbit set X.

Let then X be a one-orbit set. Choose some $x \in X$. By assumption on finite supports, this element is supported by finitely many atoms $a_1, \ldots, a_d$. Consider the orbit of the pair $((a_1, \ldots, a_d), x)$, which is the set

$$\{ (\pi(a_1, \ldots, a_d), \pi(x)) \mid \pi \text{ is an atom permutation } \}.$$

This set can be seen as a binary relation between $\mathbb{A}^d$ and X. It is equivariant by definition, since it is the orbit of some pair. The binary relation is in fact a partial function, i.e. for every input from $\mathbb{A}^d$ there is at most one output from X that is related to it; this is by definition of supports. The function is partial, because its domain is a single orbit inside $\mathbb{A}^d$, namely the orbit of $(a_1, \ldots, a_d)$. (We could make the function total by restricting it to its domain, but then the domain would cease to be a pof set, while the theorem asks for a subquotient pof set.) The range of the function is the orbit of x, and therefore the function is surjective, since we have assumed that X is a one-orbit set.

So far, we have found a partial function of type $\mathbb{A}^d \to X$ that is equivariant and surjective. We will now improve by quotienting the input with respect to the partial equivalence relation on $\mathbb{A}^d$ that is defined by

$$\bar{a} \sim \bar{b} \quad \text{if} \quad f(\bar{a}) \text{ and } f(\bar{b}) \text{ are both defined an equal to each other.}$$

After this subquotient, the function becomes total (because we have removed the inputs that have undefined outputs) and a bijective (because we have identified inputs that have the same output), as required by the theorem. □

Why finite supports? We finish this section by explaining why Definition 4.5 requires every element in an orbit-finite set to have finite support. One answer is that in the proof of Theorem 4.6, we used finite supports, to choose the atoms $a_1, \ldots, a_d$. Let us now give a more thorough explanation, by showing that without the finite support, not only does the representation from Theorem 4.6 fail, but also any other finite representation will fail. Therefore, without finite supports, the definition of orbit-finiteness becomes useless.

We begin our discussion with a study of the orbits in $\mathbb{A}^\omega$, which is a natural example of a set without finite supports. As we will see, the orbits in this set are relatively tame, and more refined examples are needed to understand the finite support requirement.

Example 31. [Orbits in $\mathbb{A}^\omega$] Consider the set $\mathbb{A}^\omega$ of infinite sequences of atoms. This set uncountably many orbits. Even if we consider sequences that use only two atoms, such as

$$(\text{John}, \text{Eve}, \text{John}, \text{Eve}, \text{John}, \text{Eve}, \ldots),$$

then the pattern in which the two atoms are distributed in the sequence can be chosen in uncountably many ways, and each such pattern will be a different orbit. However, as we will see, there will be only countably many kinds of orbits, if we identify two orbits that admit an equivariant bijection with each other.

For example, if we take any sequence that uses exactly d atoms, then the orbit of this sequence will admit an equivariant bijection with the set $\mathbb{A}^{(d)}$. (Note that a sequence which uses d atoms is finitely supported, namely by the atoms that appear in it, despite the fact that the sequence has infinite length, and therefore some atoms are repeated infinitely often in it.) In particular, all orbits that use exactly d atoms will be the same, up to equivariant bijections. Since the number d can be chosen in countably many ways, it follows that, up to equivariant bijections, there are countably many orbits that use finitely many atoms. These orbits will be the ones that have finitely supported elements.

Let us now consider the orbits in which the sequences use infinitely many atoms, i.e. orbits whose elements are not finitely supported. For a sequence define its profile to be the following information: (a) how many distinct atoms appear in the sequence, and (b) how many distinct atoms do not appear in the sequence. The profile can be chosen in countably many ways. Furthermore, one can show that if two sequences have the same profile, then their orbits will admit an equivariant bijection with each other. It follows that, up to equivariant bijections, there are countably many orbits in $\mathbb{A}^{\omega}$. □

In the above example, we have only succeeded to find countably many one-orbit sets, up to equivariant bijections. This is rectified in the following example, which will have uncountably many such sets.

Example 32. [Orbits of equivalence relations] Consider the set

$$\text{all equivalence relations on } \mathbb{A}.$$

The equivalence relations are not required to be finitely supported. (An equivalence relation is finitely supported if and only if it has one equivalence class that is co-finite.) An equivalence relation is a set of pairs of atoms, and therefore one can apply an atom permutation to an equivalence relation, yielding a new equivalence relation. Therefore, it is meaningful to talk about orbits of equivalence relations. There are uncountably many different orbits. For example, if two equivalence relations are in the same orbit, then they must agree on the following information: what is the set of possible sizes of equivalence classes. Since this information can be chosen in uncountably many ways, it follows that there are uncountably many orbits.

We will now show that for every two different orbits, there cannot be an equivariant bijection between them. This will show that, even up to equivariant bijections, there are uncountably many different orbits in our set. The key observation is that an equivalence relation $\sim$ can be identified by looking at its stabiliser, i.e. the set of atom permutations that fix it. This is because

$$a \sim b \quad \Leftrightarrow \quad \underbrace{\pi(\sim) = \sim \text{ for the atom permutation that swaps } a \text{ and } b.}_{\text{the atom permutation that swaps } a \text{ with } b \text{ belongs to the stabiliser of } \sim} \tag{4.3}$$

Stabilisers behave in a very predictable way under equivariant functions, namely they can only increase, as the following claim asserts.

Claim 4.7. *Let $f : X \to Y$ be an equivariant function, not necessarily a bijection. Then*

$$\textit{stabiliser of } x \subseteq \textit{stabiliser of } f(x) \qquad \textit{for every } x \in X.$$

Proof. If π is in the stabiliser of x, i.e. $\pi(x) = x$, then $\pi(f(x)) = f(\pi(x)) = f(x)$ by equivariance, and therefore π is also in the stabiliser of $f(x)$. □

In particular, if the function f is a bijection, then both f and its inverse can only increase the stabilisers, which means that the stabilisers do not change along f. This, together with the observation from (4.3), which says that stabilisers uniquely determine equivalence relations, we infer that an equivariant bijection between two orbits of equivalence relations can only be the identity. □

As shown in the above example, if we allow elements without finite supports, then a one-orbit set can be defined in uncountably many different ways, even up to equivariant bijections. In particular, there cannot be any finite representation of one-orbit sets without finite supports, unlike the representation from Theorem 4.6 that we have seen above. This is why the definition of orbit-finiteness requires finite supports.

4.3 A Myhill-Nerode Theorem

We now use the theory developed above to prove an orbit-finite version of the Myhill-Nerode Theorem. In the classical, finite version, the theorem says that a language is regular if and only if its syntactic congruence has finite index (i.e. finitely many equivalence classes), with the syntactic congruence defined as follows.

Definition 4.8 (Syntactic congruence)**.** The *syntactic congruence* of a language $L \subseteq \Sigma^*$ is the equivalence relation on Σ^* that identifies two words w and w' if they cannot be distinguished by any future, i.e.

$$wv \in L \Leftrightarrow w'v \in L \qquad \text{for every } v \in \Sigma^*.$$

The syntactic congruence can be applied also for infinite alphabets, such as subquotient pof sets. We will show that in this case, orbit-finite index will correspond to being recognised by a deterministic subquotient pof automaton.

We begin by explaining what it means for the syntactic congruence to have orbit-finite index. The first observation is that the syntactic congruence is equivariant, as long as the language itself is equivariant.

Lemma 4.9. *Let Σ be a subquotient pof set. If a language $L \subseteq \Sigma^*$ is equivariant, then the same is true for its syntactic congruence, i.e.*

$$w \sim w' \quad \Leftrightarrow \quad \pi(w) \sim \pi(w')$$

for every pair of words $w, w' \in \Sigma^*$ and every atom permutation π.

Proof. If the words w and w' can be distinguished by some future v, then the words $\pi(w)$ and $\pi(w')$ can be distinguished by the future $\pi(v)$, thanks to equivariance of concatenation and of the language L. □

We now explain why the notion of orbits is applicable to the quotient of Σ^* under the syntactic congruence. Consider some set X with an action of atom permutations (we care about $X = \Sigma^*$ in this example). Let $\sim$ be an equivalence relation on this set that is equivariant (we care about the syntactic congruence). The quotient $X_{/\sim}$ is also equipped with an action of atom permutations, with the action defined by

$$\text{equivalence class of } x \quad \overset{\pi}{\mapsto} \quad \text{equivalence class of } \pi(x). \tag{4.4}$$

By equivariance of $\sim$, it is easy to check that this action is well-defined, i.e. it does not depend on the choice of representative x in the equivalence class. Thanks to the above observations, if Σ is a subquotient pof set, and $L \subseteq \Sigma^*$ is an equivariant language, then we can equip the quotient

$$\Sigma^*_{/\text{syntactic congruence of } L}$$

with an action of atom permutations, and therefore we can ask if this quotient is orbit-finite. As the following theorem shows, orbit-finiteness is equivalent to recognizability by a deterministic spof automaton.

Theorem 4.10. *The following conditions are equivalent for an equivariant language $L \subseteq \Sigma^*$ over a spof alphabet Σ:*

1. *L is recognised by a deterministic spof automaton;*
2. *the quotient of Σ^* under the syntactic congruence of L is orbit-finite.*

Proof. We use essentially the same proof as in the classical Myhill-Nerode Theorem, except that we use "orbit-finite" instead of "finite".

1 ⇒ 2 Let us first show that if L is recognised by a deterministic spof automaton, then the quotient from item 2 is orbit-finite. Let Q be the reachable states of the automaton. If two words give the same state of the automaton, then they must be equivalent under the syntactic congruence. This gives us a function from Q to the equivalence classes of the syntactic congruence. This function is surjective, since every word gives some state, and it is easily seen to be equivariant. Therefore, we can deduce orbit-finiteness of the syntactic congruence by applying the following straightforward lemma.

Lemma 4.11. *Let $f : X \to Y$ be a surjective equivariant function between two sets equipped with actions of atom permutations. If X is orbit-finite, then so is Y.*

Proof. Every orbit of X is mapped to an orbit of Y. □

2 ⇒ 1 We use the standard syntactic automaton whose state space is the quotient of Σ^* under syntactic congruence, and whose transition function is given by

$$\text{equivalence class of } w \quad \xrightarrow{a} \quad \text{equivalence class of } wa.$$

We will justify that this is indeed a spof automaton. Directly from the definition of the action on the subquotient described in (4.4), we deduce that quotienting preserves finite supports: if a tuple of atoms supports a word $w \in \Sigma^*$, then the same tuple supports its equivalence class under syntactic congruence. Therefore, every element in the quotient has a finite support. By Theorem 4.6, the quotient is isomorphic to a spof set.

□

In the theorem above, we use spof sets. What about pof sets (without subquotients), as discussed at the beginning of this book? If the input alphabet is non-trivially quotiented, then we will also need quotients for the state space of the automaton, as explained in the following example.

Example 33. Consider the input alphabet

$$\Sigma = \binom{\mathbb{A}}{2},$$

and a deterministic pof automaton (without subquotients). We claim that in this automaton, all reachable states will have atom dimension zero, i.e. they will come from atom-free components $\mathbb{A}^0$. To see why this is true, we use the following observation.

Lemma 4.12. *Let $d > 0$. There is no equivariant function*

$$f : \binom{\mathbb{A}}{2} \to \mathbb{A}^d.$$

Thanks to the observation in the above lemma, if the current state is of atom dimension zero, then the next state also has this property. Therefore, the reachable states of the automaton have atom dimension zero. This will preclude recognizing any language that depends on atoms in any way. □

The above example shows that we may need quotients if the input alphabet has quotients. But what if the input alphabet is a pof set without any quotients? As we will see later in this chapter, the Myhill-Nerode Theorem does hold in this case, because we have an implication

$$\begin{array}{c}\text{recognised by a subquotient pof automaton and input alphabet is a pof set}\\ \Downarrow\\ \text{recognised by a pof automaton}\cdot\end{array}$$

However, proving this implication will require developing some extra theory, namely the existence of least supports.

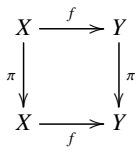
Exercises

Exercise 79. Show that a tuple $\bar{a}$ supports x if and only if

$$\pi(\bar{a}) = \sigma(\bar{a}) \quad \text{implies} \quad \pi(x) = \sigma(x) \qquad \text{for every atom automorphisms } \pi, \sigma.$$

Exercise 80. Find all equivariant binary relations on $\mathbb{A}$.

Exercise 81. Show that a function $f : X \to Y$ is equivariant if and only if the following diagram commutes for atom permutation π:



Exercise 82. Consider an enumeration $a_1, a_2, \ldots$ of some countably infinite set A. Define the distance between two permutations of A to be $1/n$ where a_n is the first argument where the permutations disagree. Let X be a countably infinite set equipped with an action of permutations of the equality atoms. Show that all elements of X are finitely supported if and only if

$$\underbrace{\pi}_{\text{permutation of } \mathbb{A}} \quad \mapsto \quad \underbrace{(x \mapsto \pi(x))}_{\text{permutation of } X}$$

is a continuous mapping, and that this continuity does not depend on the choice of enumerations of $\mathbb{A}$ or X.

Exercise 83. Show a counterexample, in the equality atoms, to the converse implication from Exercise 105. In other words, show a set which is not orbit-finite, but where every tuple of atoms supports finitely many elements.

Exercise 84. Assume the equality atoms. Let $R \subseteq \mathbb{A}^{n+k}$ be a finitely supported relation which is total in the following sense: for every $\bar{a} \in \mathbb{A}^n$ there is some $\bar{b} \in \mathbb{A}^k$ such that $R(\bar{a}\bar{b})$. Show that there is a finitely supported function $f : \mathbb{A}^n \to \mathbb{A}^k$ whose graph is contained in R.

Exercise 85. Show that in the equality atoms (actually, under any oligomorphic atoms), every orbit-finite is Dedekind finite[1], i.e. does not admit a finitely supported bijection with a proper subset of itself.

Exercise 86. Show that in the equality atoms, there is a set that is not orbit-finite, but Dedekind finite in the sense from Exercise 85.

Exercise 87. Call a family of sets *directed* if every two sets from the family are included in some set from the family. Consider the equality atoms. Show that a set with atoms X is finite (in the usual sense) if and only if it satisfies: for every set with atoms $\mathcal{X} \subseteq \mathsf{P}X$ which is directed, there is a maximal element in $\mathcal{X}$.

Exercise 88. Call a family $\mathcal{X}$ of sets *uniformly supported*[2] if there is some tuple of atoms which supports all elements of $\mathcal{X}$. Assume that the atoms are oligomorphic. Show that a set X is

[1] This exercise is inspired by Blass (2013).

[2] This exercise is inspired by (Pitts, 2013, Section 5.5).

orbit-finite if and only if: (*) there is a maximal element in every set of atoms $\mathcal{X} \subseteq \mathsf{P}X$ which is directed and uniformly supported.

Exercise 89. Show the following variant of König's lemma. If a tree has orbit-finite branching and arbitrarily long branches, then it has an infinite branch.

4.4 Least supports

An element of a set might have different supports. For example, we can add atoms to a support, and it will still be a support. In this section, we show that adding useless atoms to the support is the only phenomenon that can arise, because there will always exist a least support[3].

Theorem 4.13 (Least Support Theorem)**.** *Let X be a set with an action of atom permutation. If $x \in X$ has some finite support, then one can find atoms $a_1, \ldots, a_d$ that support x, and such that every finite support of x contains all atoms $a_1, \ldots, a_d$.*

Another way of stating the above theorem is that finite supports are closed under intersection, if they are viewed as sets (and not lists). It is important that we consider finite supports. For example, any atom a is supported by the cofinite set $\mathbb{A} - \{a\}$, since fixing the cofinite set is the same as fixing a. The intersection of the two supports $\{a\}$ and $\mathbb{A} - \{a\}$ is empty, but a does not have empty support.

Proof of the Least Support Theorem. It is enough to prove the theorem in the case when X has one orbit. This is because every other set is a disjoint union of (possibly infinitely many) one-orbit sets. Recall the set $\mathbb{A}^{(d)}$ of non-repeating tuples that was described in Example 22. This is an equivariant one-orbit set. The key observation is the following lemma.

Lemma 4.14. *Assume that X has one orbit. There is some $d \in \{0, 1, \ldots\}$ and an equivariant surjective function*

$$f : \mathbb{A}^{(d)} \to X$$

such that tuples with the same value under f are equal as sets:

$$f(a_1, \ldots, a_d) = f(b_1, \ldots, b_d) \qquad \textit{implies} \qquad \{a_1, \ldots, a_d\} = \{b_1, \ldots, b_d\}.$$

Proof. By Theorem 4.6, X admits an equivariant bijection with a spof set. This means that there is an equivariant surjective function from a pof set Y (without subquotients) to X. Take some equivariant orbit of this function, with the function viewed as a subset of $Y \times X$. This orbit is still an equivariant function whose image is also X. This way, we can assume without loss of generality that Y is a single equivariant orbit in a set of the form $\mathbb{A}^d$. Such an orbit is an equality type. By projecting away the duplicated

[3] The Least Support Theorem was first proved in (Gabbay and Pitts, 2002, Proposition 3.4). A generalisation of this theorem, for other kinds of atoms, can be found in (Bojańczyk et al., 2014, Section 10).

coordinates in the equality type, we can assume that Y contains only non-repeating tuples. Summing up, we know that there is a surjective equivariant function

$$f : \mathbb{A}^{(d)} \to X.$$

We show below that the function either satisfies the condition in the statement of the lemma, or the dimension d can be made smaller. If the condition in the statement of the lemma is not satisfied, then

$$f(a_1, \ldots, a_d) = f(b_1, \ldots, b_d) \tag{4.5}$$

holds for some tuples $\bar{a}, \bar{b}$ which are not equal as sets. Without loss of generality, we assume that the last atom a_d in $\bar{a}$ does not appear in the tuple $\bar{b}$. Choose some atom permutation π which fixes the first $d-1$ atoms in $\bar{a}$ and all atoms in $\bar{b}$, but does not fix the last atom a_d in $\bar{a}$. We have

$$f(\bar{a}) \overset{(4.5)}{=} f(\bar{b}) \overset{\pi \text{ fixes } \bar{b}}{=} f(\pi(\bar{b})) \overset{\text{equivariance}}{=} \pi(f(\bar{b})) \overset{(4.5)}{=} \pi(f(\bar{a})) \overset{\text{equivariance}}{=} f(\pi(\bar{a})),$$

which proves that

$$f(a_1, \ldots, a_{d-1}, a_d) = f(a_1, \ldots, a_{d-1}, a) \qquad \text{for some distinct } a, a_1, \ldots, a_d.$$

The set of tuples $a, a_1, \ldots, a_d$ which satisfies the condition above is an equivariant subset of $\mathbb{A}^{(d+1)}$, by equivariance of f. Therefore, if some tuple satisfies the condition, then all tuples in $\mathbb{A}^{(d+1)}$ satisfy it as well, i.e. we could also write "for all distinct" in the above condition. In other words, the value of f depends only on the first $d-1$ coordinates. Therefore, we can use the induction assumption. □

Using the above lemma, we complete the proof of the Least Support Theorem. Apply Lemma 4.14 yielding some equivariant function

$$f : \mathbb{A}^{(n)} \to X.$$

Let $x \in X$, and choose some tuple $(a_1, \ldots, a_n)$ which is mapped by f to x. To prove the Least Support Theorem, we will show that the atoms $a_1, \ldots, a_n$ appear in every support of x. Let then $\bar{b}$ be some atom tuple which supports x. Toward a contradiction, suppose that $\bar{b}$ is not a permutation of $a_1, \ldots, a_d$, and therefore one can choose atom permutation π such that

$$\begin{aligned} &\pi(b_i) = b_i \quad \text{for every } i \in \{1, \ldots, d\} \\ &\pi(a_i) \notin \{a_1, \ldots, a_d\} \quad \text{for some } i \in \{1, \ldots, d\}. \end{aligned}$$

We have

$$\begin{aligned} x &= && (\pi \text{ fixes the support of } x) \\ \pi(x) &= && (\text{choice of } a_1, \ldots, a_d) \\ \pi(f(a_1, \ldots, a_d)) &= && (\text{equivariance of } f) \\ f(\pi(a_1, \ldots, a_d)). & && \end{aligned}$$

Since the tuple $\pi(a_1, \ldots, a_n)$ is not equal to $(a_1, \ldots, a_n)$ as a set, it must have a different value than x, by assumption on the function f. □

A representation theorem

Apart from the Least Support Theorem, another application of Lemma 4.14 is the following representation theorem for equivariant orbit-finite sets. Let X be a one-orbit set. Apply Lemma 4.14, yielding an equivariant function

$$f : \mathbb{A}^{(d)} \to X.$$

Because f is equivariant and permutations of coordinates commute with atom automorphisms, the following conditions are equivalent for every permutation g of the coordinates $\{1, \ldots, d\}$:

$$f(a_1, \ldots, a_d) = f(a_{g(1)}, \ldots, a_{g(d)}) \qquad \text{for some } (a_1, \ldots, a_d) \in \mathbb{A}^{(d)} \tag{4.6}$$

$$f(a_1, \ldots, a_d) = f(a_{g(1)}, \ldots, a_{g(d)}) \qquad \text{for every } (a_1, \ldots, a_d) \in \mathbb{A}^{(d)}. \tag{4.7}$$

Permutations g which satisfy condition (4.7) form a group, call it G. This is a subgroup of the group of permutations of the coordinates $\{1, \ldots, d\}$. We claim:

$$\begin{array}{rcl} f(a_1, \ldots, a_d) & = & f(b_1, \ldots, b_d) \\ & \text{iff} & \\ \exists g \in G\ (a_1, \ldots, a_d) & = & (b_{g(1)}, \ldots, b_{g(d)}). \end{array}$$

The bottom-up implication is by definition. For the top-down implication, recall that Lemma 4.14 asserted that tuples with the image under f must contain the same atoms, and therefore some $g \in G$ must take one tuple to the other. Let us write

$$\mathbb{A}^{(d)}/_G$$

to be $\mathbb{A}^{(d)}$ for the set of non-repeating atom tuples modulo coordinate permutations from the group G. Since quotienting by G is exactly the kernel of the function f, we have just proved the following theorem[4]:

Theorem 4.15. *Let X be an orbit-finite set that has one orbit. Then X admits an equivariant bijection to a set of the form*

$$\mathbb{A}^{(d)}/_G$$

for some $d \in \mathbb{N}$ and some subgroup G of the group of permutations of the set $\{1, \ldots, d\}$.

Example 34. Let $d \in \{1, 2, \ldots\}$ and let G be the group of all permutations of $\{1, \ldots, d\}$. In this case, $\mathbb{A}^{(d)}/_G$ is the same as

$$\binom{\mathbb{A}}{d},$$

i.e. unordered sets of atoms with exactly d elements. □

[4] This result is from (Bojańczyk et al., 2014, Theorem 10.17), although a similar construction can already be found in (Ferrari et al., 2002, Definition 2).

Myhill-Nerode for pof sets

As we have mentioned after the proof of the Myhill-Nerode characterization in Theorem 4.10, in the case of pof sets without subquotients, deterministic pof automata are equivalent to deterministic spof automata. This is proved below.

Theorem 4.16. *Assume that the input alphabet is a pof set Σ (without subquotients). Then the two equivalent conditions in Theorem 4.10 are also equivalent to*

(3) L is recognised by a deterministic pof automaton.

Proof. Since pof sets are a special case of subquotient pof sets, it is enough to show that if the input alphabet is a pof set (without subquotients), then the subquotients can be removed from deterministic automata: for every deterministic spof automaton, there is an equivalent deterministic pof automaton. Consider a deterministic subquotient pof automaton. Let Q be its state space. By Theorem 4.15, we can assume that the state space is

$$Q = \sum_{i \in I} \mathbb{A}^{(d_i)}/G_i$$

Consider the pof set

$$P = \sum_{i \in I} \mathbb{A}^d.$$

There is a natural projection from P to Q, which is a partial function

$$\Pi : P \to Q.$$

This projection is defined on elements with non-repeating tuples of atoms, and it returns the corresponding equivalence class. An important property of this projection is:

(*) every output element arises from finitely many input elements.

Let us pull back the transition function of the original automaton to a transition relation on states P. In other words, define

$$\Delta \subseteq P \times \Sigma \times P$$

to be the inverse image, under the projection Π, of the transition function of the original automaton. We view Δ as a binary relation between two pof sets, namely $P \times \Sigma$ and P. Because the original automaton was deterministic, and thanks to the finiteness condition (*), we know that for every input in $P \times \Sigma$, the transition relation Δ has finitely many outputs in P. Therefore, we can apply the following lemma to extract an equivariant function contained in Δ.

Lemma 4.17. *Let X and Y be pof sets, and let $\Delta \subseteq X \times Y$ be an equivariant binary relation such that for every $x \in X$ the set*

$$\{ y \in Y \mid (x, y) \in \Delta \}$$

is nonempty and finite. Then there is an equivariant function $\delta : X \to Y$ whose graph is contained in Δ.

Proof. We first observe that for every pair $(x, y) \in \Delta$, every atom that appears in the output y must also appear in the input x. This is because if the output would contain some fresh atom, i.e. some atom that does not appear in the input, then by equivariance of the relation we could replace this fresh atom by infinitely many other choices. This would contradict the assumption that there are finitely many outputs for every input.

Decompose the set Δ into orbits:

$$\Delta = \Delta_1 \cup \cdots \cup \Delta_n.$$

We will now show that each of these orbits is an equivariant partial function from X to Y. Clearly each orbit is equivariant. To show that the orbits are partial functions, we must show that if we take two pairs

$$(x_1, y_1), (x_2, y_2)$$

that are in the same orbit Δ_i, and their inputs x_1 and x_2 are equal, then their outputs y_1 and y_2 must also be equal. Indeed, by the observations from the previous paragraph, all atoms appear in the output must appear also in the input. By the assumption that the two pairs are in the same orbit, it follows that the component must be the same in the outputs y_1 and y_2, and also the way that output coordinates are filled with input atoms must be the same. Since all atoms are copied from the input, and the inputs are the same, it follows that the two outputs are the same.

We will now select some orbits of Δ so that they sum up to a total equivariant function from X to Y. If we project an orbit of Δ to its first (input) coordinate, then we get an orbit of X. By the assumption that every input has at least one output, we know that each orbit of X arises as the projection of at least one orbit of Δ. Choose for each orbit of X exactly one orbit of Δ that projects to it. As we have argued in the previous paragraph, this chosen orbit is a partial function from X to Y. Since we have chosen the orbits so that every input is covered exactly once, by summing up these orbits we obtain a total equivariant function from X to Y. □

□

Chapter 5

Atoms beyond equality

So far, we have worked with atoms that have equality only. It turns out that the theory developed in this book is also meaningful when the atoms have extra structure, like an order. We take a logical approach, where the notion of atoms is specified by a relational structure, i.e. a set with some relations on it. Here are some examples:

$$\underbrace{(\mathbb{N})}_{\substack{\text{the natural numbers}\\ \text{with no relations}}} \qquad \underbrace{(\mathbb{N}, <)}_{\substack{\text{the natural numbers}\\ \text{with order}}} \qquad \underbrace{(\mathbb{Z}, <)}_{\substack{\text{the integers}\\ \text{with order}}} \qquad \underbrace{(\mathbb{Q}, +)}_{\substack{\text{the rational numbers}\\ \text{with a ternary relation}\\ \text{for addition } x+y=z}}.$$

All of these structures will be candidates for atoms, however only the first and last one will turn out to be appropriate. This chapter explains when a structure is appropriate, and how the theory works when that happens.

5.1 Oligomorphic structures

The choice of atoms will be formalized by a relational structure, as in model theory.

Definition 5.1 (Relational structure)**.** A *relational structure* consists of:

1. an underlying set A, called the *universe* of the structure;
2. a family of relations on this set, each one of the form $R \subseteq A^d$ for some d.

We use letters like $\mathbb{A}$ or $\mathbb{B}$ to describe the atoms. A candidate for the atoms is a relational structure. Not all candidates are appropriate, however. Here is an example of an inappropriate one.

Example 35. [Presburger Arithmetic] Suppose that for the atoms we would like to use the natural numbers with addition, i.e. the relational structure

$$\mathbb{A} = (\{0, 1, 2, \ldots\}, \underbrace{x + y = z}_{\text{a ternary relation for the addition}}).$$

This structure is also known as Presburger Arithmetic. We could consider polynomial orbit-finite sets for this structure, and subsets of them that are definable using formulas. This time, the formulas could use the relations given in the structure, namely the successor relation and a zero test. In this setting, many of the problems that were decidable previously, will become undecidable for the new choice of atoms. An example is graph reachability – one can easily encode the halting problem for counter machines. To implement a zero test on variable x, we check if $x + x = x$. □

As we see from the above example, some structures, such as Presburger Arithmetic, will not be a good choice for the atoms. This is despite Presburger Arithmetic being a very tame structure, in particular it has a decidable first-order theory, as formalized in the following definition. (We assume that the reader is familiar with the basics of first-order logic, such as what it means for a formula to be true in a structure, or what free variables are. For a more detailed introduction, see Hodges (1993).)

Definition 5.2 (Decidable first-order theory)**.** The *first-order theory* of a structure is the set of first-order sentences that are true in it. Here, a first-order sentence is a formula that is built using the following constructors

$$\underbrace{\forall x \quad \exists x}_{\text{quantifiers}} \qquad \underbrace{\vee \quad \wedge \quad \neg}_{\text{Boolean combinations}} \qquad \underbrace{x = y}_{\text{equality}} \qquad \underbrace{R(x_1, \ldots, x_d)}_{\text{relations from the structure}},$$

and which has no free variables. A structure has a decidable first-order theory if there is an algorithm that decides if first-order sentence belongs to the theory.

We will typically be interested in structures with a decidable first-order theory. However, as we saw in Example 35, having a decidable first-theory will not – on its own – be sufficient for our theory. It will be equally important that the notion of equivariant subset is well-behaved. So far, equivariance, was defined in terms of atom permutations. When the atoms are a structure with relations beyond equality, the role of permutations is played by automorphisms, as described in the following definition.

Definition 5.3 (Automorphisms and equivariance)**.** Let $\mathbb{A}$ be a relational structure.

- An *automorphism* of $\mathbb{A}$ is a bijection π of its universe with itself, which preserves all relations, i.e.

  $$\bar{a} \in R \qquad \Leftrightarrow \qquad \pi(\bar{a}) \in R$$

  holds for every relation R of arity d in the structure and every tuple $\bar{a} \in \mathbb{A}^d$. The automorphisms form a group, since they can be composed and inverted.

- Consider a set X that is equipped with an action of the group of automorphisms of $\mathbb{A}$. A subset $Y \subseteq X$ is called *equivariant* if it is invariant under the action, i.e.

  $$y \in Y \quad \Leftrightarrow \quad \pi(y) \in Y \qquad \text{for every automorphism } \pi \text{ of } \mathbb{A}.$$

Example 36. [Presburger arithmetic is rigid] Suppose that we define the atoms $\mathbb{A}$ to be Presburger Arithmetic. The problem with this choice is that there are no non-trivial

automorphisms (such structures are called rigid). Indeed, an automorphism must map 0 to 0, and then it must map 1 to 1, and so on. Therefore, the only automorphism is the identity. This means that every subset of $\mathbb{A}$, or more generally $\mathbb{A}^d$, is going to be equivariant. In particular, this precludes any finite representation or algorithms that would deal with equivariant subsets. □

Example 37. [Equality atoms] Suppose that we define the atoms $\mathbb{A}$ to be a structure where the universe is some countably infinite set, and which has no relations. (Equality is assumed to be a given for first-order logic.) An automorphism in this case is the same as a bijection of the universe with itself, i.e. a permutation of the universe, which corresponds to the atoms that were studied in the previous chapters of this book. For this reason, we call this structure the *equality atoms*. These were the atoms that were studied in the first chapters of this book. □

Example 38. [Integers with order] In this example, we present a candidate for the atoms which will turn out to be inadequate, since it will not satisfy the oligomorphism condition that will be defined below. Suppose that we define the atoms $\mathbb{A}$ to be $(\mathbb{Z}, <)$, i.e. the integers with order. Automorphisms of this structure must preserve the order. Therefore, they must also preserve the successor relation. Indeed, if two consecutive elements x and $x+1$ would be mapped to non-consecutive elements, then the resulting gap would be a violation of bijectivity. Therefore, automorphisms of this structure are translations, i.e. functions of the form $x \mapsto x+c$ for some $c \in \mathbb{Z}$. In this structure, $\mathbb{A}$ has one orbit, because one can go from every integer to every other integer by applying some translation. However, $\mathbb{A}^2$ has infinitely many orbits, because the difference $x_1 - x_2$ between the two coordinates is preserved by translations. Therefore, there are uncountably many equivariant subsets of $\mathbb{A}^2$. □

As illustrated in the above examples, we want the structure to have finitely many orbits under its automorphisms. This should not only hold for the structure $\mathbb{A}$ itself, but also for its powers $\mathbb{A}^d$, since such powers will arise in our constructions (such as pof sets). Hence the following definition.

Definition 5.4 (Oligomorphic structure)**.** A structure $\mathbb{A}$ is called *oligomorphic*[1] if for every $d \in \{0, 1, \ldots\}$, the set $\mathbb{A}^d$ has finitely many elements up to automorphisms of $\mathbb{A}$. More precisely, for every d, the equivalence relation on $\mathbb{A}^d$ defined by

$$\bar{a} \sim \bar{b} \qquad \text{if } \pi(\bar{a}) = \bar{b} \text{ for some automorphism } \pi \text{ of } \mathbb{A}$$

has finitely many equivalence classes.

Example 39. The equality atoms from Example 37, i.e. an infinite set without any structure except equality, are oligomorphic. These are the atoms that we have studied

[1] The notion of oligomorphic structures comes from Ryll-Nardzewski (1959), Engeler (1959) and Svenonius (1959), who proved that countable oligomorphic structures are exactly those which are ω-categorical, i.e. are the unique countable models of their first-order theory. This connection with first-order logic will be important later in this chapter, when discussing how orbit-finite sets can be represented using formulas of first-order logic.

so far: automorphisms are permutations, and the number of orbits in $\mathbb{A}^d$ is the d-th Bell number. The other structures

$$(\mathbb{N}, +) \qquad (\mathbb{Z}, <)$$

discussed in Examples 35 and 38 are not oligomorphic. For the second one, we need to go to the second power to get infinitely many orbits. □

Example 40. Every structure with a finite universe is oligomorphic. □

Example 41. [Ordered atoms] Consider the structure $(\mathbb{Q}, <)$ of ordered rational numbers. We will call this structure the *ordered atoms*. This is because it will turn out that this structure is the canonical way of modelling a total order in our theory, as we will describe in Chapter 6. We will show that this structure is oligomorphic. An automorphism of this structure is any order-preserving permutation. For example, the affine function

$$x \mapsto \frac{x}{3} - 2$$

is an automorphism. On the other hand, x^3 is not an automorphism, despite preserving the order. The reason is that cubing is not invertible on the rationals. To show that this structure is oligomorphic, we will prove that two tuples

$$(a_1, \ldots, a_d), (b_1, \ldots, b_d) \in \mathbb{Q}^d$$

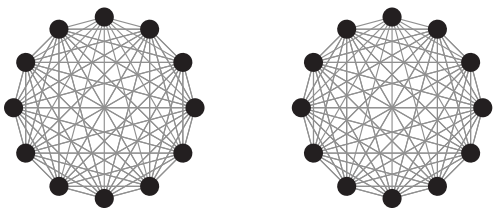
are in the same orbit, with respect to atom automorphisms, if and only if they have the same order type, i.e.

$$a_i \leq a_j \quad \Leftrightarrow \quad b_i \leq b_j \qquad \text{for all } i, j \in \{1, \ldots, d\}.$$

This will imply oligomorphism, since there are finitely many different order types for each dimension d. Clearly if the tuples are in the same orbit, then they must have the same order type, by definition of automorphisms. The converse implication is also not hard to see, for example it is sufficient to consider piecewise affine maps. □

Example 42. The real numbers with order $(\mathbb{R}, <)$ are also oligomorphic. The same argument as for the rationals works. However, we will not study this structure since we care about countable ones only. □

Example 43. Consider an undirected graph with two countably infinite cliques (without self-loops). Here is a picture, with only 12 vertices shown for each of the two cliques:



The graph, like any graph, can be viewed as a logical structure, where the universe is the vertices, and there is one binary relation for edges, which is symmetric and irreflexive. The automorphisms of this structure (which are the same as graph automorphisms in the usual sense) are generated by: permutations of the first clique, permutations of the second clique, and swapping the two cliques. In particular, the tuples

$$(a_1, \ldots, a_d) \quad \text{and} \quad (b_1, \ldots, b_d)$$

are equal up to atom automorphisms if and only if they have the same equality types and the same equivalence types with respect to the equivalence relation "in the same clique". Since there are finitely many possibilities for every choice of n, it follows that these atoms are oligomorphic. □

Polynomial orbit-finite sets. Many of the notions that we have described so far make sense for other atoms, and not just the equality atoms. The only difference is that instead of atom permutations, we use the more general notion of atom automorphism. In the special case of the equality atoms, this will be the same as atom permutations.

We begin with the generalization of pof sets and their equivariant subsets.

Definition 5.5 (Polynomial orbit-finite sets for general atoms)**.** Let $\mathbb{A}$ be an oligomorphic structure. A *pof set* over this structure is any set of the form

$$\mathbb{A}^{d_1} + \cdots + \mathbb{A}^{d_k}.$$

A subset X of a pof set is called *equivariant* if membership in the subset is invariant under atom automorphisms, i.e.

$$x \in X \quad \Leftrightarrow \quad \pi(x) \in X \qquad \text{for every automorphism } \pi \text{ of } \mathbb{A},$$

with the expected action of automorphisms on elements of the pof set.

The first part of the above definition, i.e. applying a polynomial to some structure, makes sense for structures that are not necessarily oligomorphic. However, we intend to study pof sets equipped with equivariant subsets, and equivariant subsets are well-behaved only for oligomorphic structures.

As was the case for the equality atoms, we can consider pof automata, now for a general structure.

Example 44. Consider the ordered atoms $\mathbb{A} = (\mathbb{Q}, <)$ from Example 41, and the language

$$\{ w \in \mathbb{A}^* \mid \text{ the letters in } w \text{ are strictly growing } \}.$$

This language is recognised by a deterministic pof automaton. The automaton stores the most recent letter, and enters a rejecting sink state if it sees a decrease. The state space is

$$\underbrace{\mathbb{A}^0}_{\text{initial}} + \underbrace{\mathbb{A}^1}_{\substack{\text{last}\\ \text{atom}}} + \underbrace{\mathbb{A}^0}_{\text{error}}$$

and the transition function is defined as expected. □

Exercises

Exercise 90. Show that the structure $(\mathbb{Z}, <)$ is not oligomorphic.

Exercise 91. For the atoms $(\mathbb{Q}, <)$, find all equivariant binary relations on $\mathbb{A}$.

Exercise 92. Consider a structure $\mathbb{A}$ that is oligomorphic. Let $\mathbb{B}$ be a new structure, whose universe is a pof set over $\mathbb{A}$, and whose relations are equivariant (under automorphism of $\mathbb{A}$). Show that $\mathbb{B}$ is also an oligomorphic structure.

5.2 Representation of equivariant subsets

The purpose of the theory that is developed in this book is to have a generalization of finiteness that is amenable to algorithms. In particular, equivariant sets should allow for finite representations, and should have other good properties of finite sets. From the very definition of oligomorphism we see that an equivariant subset can be chosen in finitely many ways, as explained in the following lemma.

Lemma 5.6. *Let $\mathbb{A}$ be a relational structure that is oligomorphic. Then every pof set has finitely many equivariant subsets.*

Proof. It is enough to show that pof sets of the form $\mathbb{A}^d$ have finitely many equivariant subsets, and the result will transfer to general pof sets, which are disjoint unions of such sets. By definition of oligomorphism, there are finitely many orbits in $\mathbb{A}^d$, and each equivariant subset is a union of such orbits. Therefore, there are finitely many choices. □

A corollary of the above lemma is that certain fixpoint algorithms will be guaranteed to terminate in finite time. A typical example is graph reachability, as illustrated below.

Example 45. In Theorem 1.9, we showed that graph reachability is decidable for pof graphs under the equality atoms. Suppose that we want to generalise this to any oligomorphic atoms. A natural idea is to consider the chain

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq \cdots \tag{5.1}$$

where V_n is the set of vertices that can be reached from some source vertex via a path of length at most n. Assuming that set of source vertices is equivariant, and the edge relation is also equivariant, the set V_n will also be equivariant for every n. It follows from Lemma 5.6 that the chain (5.1) will stabilize after finitely many steps, and therefore the set of reachable vertices can be obtained in finitely many steps. □

In the above example, we showed an "algorithm" that decides graph reachability by computing a finite increasing chain of equivariant subsets of vertices. However, for this to be an algorithm, we need to be to represent subsets V_n from the chain in a finite way; compute the new subsets based on the previous ones, and test equality between such subsets. This leads us to the question:

How can we represent equivariant subsets of a pof set?

Of course, we want the representation to support certain basic operations, such as checking if two subsets are the same (because the same subset might have several representations), or Boolean combinations on subsets. Since a pof set is a finite union of sets of the form $\mathbb{A}^d$, the question reduces to

How can we represent equivariant subsets of $\mathbb{A}^d$?

In the case of the equality atoms, in Section 1.1 we proposed two representations, namely generating subsets, and formulas. As it turns out, these two representations carry over to general oligomorphic structures.

By definition of oligomorphic atoms, see Lemma 5.6, a pof set will have finitely many orbits, and therefore every equivariant subset can be represented by giving one element for each orbit. Therefore, we can use finite sets of generators to describe equivariant subsets, at least as long as we can write down individual elements of the structure. There are, however, some unresolved questions about this representation of subsets. For example: how do we test equality of two subsets given by generators? Or: how do we compute the complement? We will return to these questions in Section 7.1; for the moment we will stick to the formula representation. As we will see below, the formula representation is very well suited to the oligomorphic case, since oligomorphic structures are exactly those where equivariant subsets can be defined by first-order formulas.

Under the equality atoms, we represented an equivariant subset of $\mathbb{A}^d$ by a formula

$$\varphi(x_1, \ldots, x_d)$$

that used Boolean combinations and equality. In the general oligomorphic case, we will also use such formulas, but we will allow quantifiers, and other relations – beyond equality – that are present in the structure. Subset of $\mathbb{A}^d$ that can be defined this way are called *first-order definable*. For some structures, such as the equality atoms, we can avoid quantifiers, but for others the quantifiers will be necessary, as illustrated in the following example.

Example 46. Consider the following three-vertex graph:

As mentioned in Example 43, this graph can be seen as a relational structure with one binary relation. Like any finite structure, this structure is oligomorphic. There is no quantifier-free formula that distinguished the isolated vertex from a non-isolated vertex, despite the two vertices being in different orbits. □

The following theorem shows that for oligomorphic structures which are countable (i.e. have a countable universe), equivariant subsets are exactly the first-order definable ones.

Theorem 5.7. *Let $\mathbb{A}$ be a countable oligomorphic structure. A subset $X \subseteq \mathbb{A}^d$ is equivariant if and only if it is first-order definable.*

Proof. In this proof, we use the name *atom* for elements of the universe. Consider the following game (known as the Ehrenfeucht-Fraïssé game), which is parametrised by two tuples $\bar{a}, \bar{b} \in \mathbb{A}^d$ and a number of rounds $k \in \{0, 1, 2, \ldots, \omega\}$. The game is played by two players, called Spoiler and Duplicator. In each round:

- Spoiler chooses one of the tuples and extends it with one atom.
- Duplicator responds by extending the other tuple with one atom.

Spoiler wins the game if, for some finite $i \leq k$, the (extended) tuples after playing i rounds can be distinguished by some quantifier-free formula (using the relations from the structure), otherwise Duplicator wins. The theorem follows immediately from the equivalence of items 1 and 4 in the following lemma.

Lemma 5.8. *The following conditions are equivalent for every tuples $\bar{a}, \bar{b} \in \mathbb{A}^d$:*

1. *the tuples belong to the same first-order definable subsets;*
2. *Duplicator has a winning strategy in the k-round game for every $k < \omega$;*
3. *Duplicator has a winning strategy in the ω-round game;*
4. *the tuples are in the same orbit.*

Proof.

- *1 implies 2.* This is (half of) the classical Ehrenfeucht-Fraïssé theorem[2], which says that if two tuples satisfy the same formulas of quantifier rank at most k, then Duplicator has a winning strategy in the k-round game. Recall that the quantifier rank of a formula is the maximal nesting of quantifiers.
- *2 implies 3.* In this step, we use oligomorphism. The key observation is in the following claim, which shows that Duplicator has a strategy that ensures staying in positions that satisfy 2.

  Claim 5.9. *Consider one round of the Ehrenfeucht-Fraïssé game, which begins in a position (i.e. a pair of atom tuples of same finite length) that satisfies condition 2. For every move of player Spoiler, there is a response of player Duplicator which ensures that the resulting position also satisfies condition 2.*

  Proof. Suppose that the round begins in a position $(\bar{a}, \bar{b})$. By symmetry, we only consider the case when Spoiler extends the tuple $\bar{a}$ with some atom $a \in \mathbb{A}$. By condition 2, we know that for every k there is some response $b_k \in \mathbb{A}$ of player Duplicator, which guarantees that

  $$\text{Duplicator can win the } k\text{-round game from position } (\bar{a}a, \bar{b}b_k). \tag{5.2}$$

  Observe that the above condition, which describes a property of tuples of some fixed length, is equivariant. This is because the dynamics of the game would

[2] See (Hodges, 1993, Section 3.2)

not be affected if we applied an atom automorphism to all choices. By oligomorphism, we know that there are finitely many orbits of tuples

$$(\bar{a}a, \bar{b}b_k)$$

that can be realized. Therefore, some orbit is hit by infinitely many choices of b_k. By equivariance of (5.2), we can pick some b_k that witnesses an orbit that is hit infinitely often, and this b_k will guarantee winning the k-round game for infinitely many k, and therefore for all k. □

Thanks to the above claim, if we play the ω-round game starting in a position that satisfies 2, then Duplicator can play in a way that guarantees always staying in positions that satisfy 2. In particular, Duplicator can win ω-rounds, thus witnessing 3.

- *3 implies 4.* In this step, we use countability. We need to show that if Duplicator has a winning strategy in the ω-round game for tuples $\bar{a}$ and $\bar{b}$, then there is an automorphism that maps one tuple to the other. This is proved using a back-and-forth argument. Fix some enumeration of the model $\mathbb{A}$, which exists by assumption on countability. Consider a play in the ω-round game, where Spoiler uses the following strategy:
  - in even-numbered rounds, extend the $\bar{a}$ tuple with the least (according to the enumeration) atom that does not appear in it;
  - in odd-numbered rounds, do the same for the $\bar{b}$ tuple.

  Suppose that Duplicator responds to the above strategy with a winning strategy. In the resulting play, we get two infinite sequences

  $$a_1, a_2, \dots \qquad b_1, b_2, \dots$$

  of atoms that extend the tuples $\bar{a}$ and $\bar{b}$, respectively. By the choice of Spoiler's strategy, every atom appears in the first infinite sequence, and also every atom appears in the second infinite sequence. Therefore, the function $a_i \mapsto b_i$ is permutation of the atoms. Furthermore, this permutation is an automorphism, since at every step in the game, the same quantifier-free formulas must be satisfied on both sides.

- *4 implies 1.* By induction on the quantifier rank k, one shows that tuples in the same equivariant orbit must satisfy the same first-order formulas of quantifier rank k.

This completes the proof of the lemma, and therefore also of the theorem. □

□

Graph reachability

In the previous chapters, we showed that some decision problems – such as graph reachability or emptiness for nondeterministic automata – can be decided. We now show that these results carry over to other structures, under the suitable assumptions. The first of these assumptions is that the structure is countable and oligomorphic, and so we can use Theorem 5.7 to conclude that equivariant subsets can be represented in a finite way, namely by first-order formulas. This assumption makes the decision problems well-posed, because the inputs (such as graphs or automata) can be represented in a finite way. However, we also need to be able to operate on equivariant subsets. For example, the same equivariant subset might have several representations, and we need to be able to test equality between them. This boils down to the question: given two first-order formulas

$$\varphi(x_1, \ldots, x_d) \qquad \text{and} \qquad \psi(x_1, \ldots, x_d),$$

decide if they define the same subset of $\mathbb{A}^d$. Already in the special case when $d = 0$, i.e. when the formulas are sentences, this problem is the same as checking which sentences are true in the structure. Therefore, in order to manipulate equivariant subsets represented by formulas, we will want the first-order theory to be decidable; this will be our second assumption. These two assumptions will be enough for many algorithms. An example is graph reachability – the following theorem shows that the decidability result from Section 1.2 transfers over from the equality atoms to general oligomorphic structures.

Theorem 5.10. *Assume that the atoms $\mathbb{A}$ are a countable oligomorphic structure with a decidable first-order theory. Then reachability for pof graphs is decidable.*

Proof. Although we have essentially described the algorithm in Example 45, we spell out the details about the representation in this proof, to explain how exactly we manipulate equivariant subsets represented by formulas. The input to the problem consists of a pof set

$$V = \sum_{i \in I} \mathbb{A}^{d_i},$$

together with three equivariant relations:

$$\underbrace{E \subseteq V^2}_{\text{edges}} \qquad \underbrace{S, T \subseteq V}_{\substack{\text{source and target} \\ \text{vertices}}}$$

An equivariant subset of V – such as the source and target sets – is represented by a family of first-order formulas, with one formula for each component $i \in I$ of the disjoint union in the set V. The formula for component i has d_i free variables, and tells us when a tuple of atoms belongs to the i-th component. A similar representation is used for the binary relation E – for each pair of components $i, j \in I$, there is a formula with $d_i + d_j$ free variables, which tells us when a tuple of atoms from the i-th

component is related to a tuple of atoms from the j-th component. We will use these representations to implement a reachability algorithm.

We intend to compute the chain

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq \cdots$$

of sets, such that V_n is the vertices that can be reached by a path of length at most n. Each set V_n will be represented by a family of formulas, call these formulas $\{\varphi_i^n\}_{i \in I}$. For $n = 0$, we use the formulas for the source set. Let us now show how to compute the formulas for V_{n+1} based on the formulas for V_n:

$$\overbrace{\varphi_i^{n+1}(\bar{x})}^{\substack{\text{the formula for} \\ \text{component } i \text{ in } V_{n+1}}} = \overbrace{\varphi_i^n(\bar{x})}^{\substack{\text{the formula for} \\ \text{component } i \text{ in } V_n}} \vee \underbrace{\bigvee_{j \in I} \exists \bar{y}}_{\substack{\text{choosing a component} \\ j \in I \text{ and a tuple } \bar{y} \\ \text{of } d_j \text{ atoms is the same as} \\ \text{choosing an element of } V}} \Big(\overbrace{\varphi_j^n(\bar{y})}^{\substack{\text{the formula for} \\ \text{component } j \text{ in } V_n}} \wedge \underbrace{\varphi_{ji}^E(\bar{y}, \bar{x})}_{\substack{\text{the formula for} \\ \text{components } i \text{ and } j \text{ in} \\ \text{the edge relation } E}} \Big).$$

As explained in Example 45, this chain cannot grow infinitely often, because the set of vertices has finitely many orbits, and each set in the chain is a union of these orbits. Also, a new set in the chain is defined in terms of the previous one, and therefore once we have $V_{n+1} = V_n$ for some n, then the chain stabilizes forever. We can check when the chain stabilizes by asking if the following first-order formula – which says that no new elements have been added – is true in the atoms:

$$\bigwedge_{i \in I} \forall \bar{x}\, \varphi_i^n(\bar{x}) \;\Rightarrow\; \varphi^{n+1}(\bar{x}).$$

We can get an answer to this question, by the assumption that the atoms have a decidable first-order theory. □

In the proof above, we did not give a more precise estimate on the computational complexity of the problem, beyond saying that it is decidable. Later on in this book, we will see that the algorithm is in PSPACE for most choices of atoms that we consider, including the equality atoms (this was already shown in Section 1.2), and the ordered atoms.

Exercises

Exercise 93. Consider a structure with a countable vocabulary. Show that if it is not oligomorphic, then there is some subset of $\mathbb{A}^d$ that is equivariant, but not first-order definable.

Exercise 94. Consider the following two conditions for an orbit-finite graph:

1. there is an infinite directed path;
2. there is a cycle.

Find an atom structure where the two conditions are equivalent, and also an atom structure where only the implication $1 \Leftarrow 2$ is true.

Exercise 95. Show that under the assumptions of Theorem 5.10, there is an algorithm that checks if condition 1 of Exercise 94 is satisfied for a pof graph. Likewise for condition 2.

Exercise 96. An instance of *alternating reachability* is defined in the same way as an instance of graph reachability, i.e. there is a directed graph with distinguished source and target vertices. The difference is in the semantics: we play a game between players Odd and Even, with Odd choosing the next edge in odd rounds, and Even choosing the next edge in even rounds. We want to decide if player Odd has a strategy that guarantees seeing a target vertex in a finite number of rounds, regardless of the choice of initial vertex in the source set[3]. Show that this problem is decidable under the assumptions of Theorem 5.10.

Exercise 97. Assume the equality atoms. A *Büchi game* has the same syntax as alternating reachability from Exercise 96. The game is played similarly, except that the objective of player 0 is to see vertices from T infinitely often. Give an algorithm that decides the winner in a Büchi game represented by a set builder expression. Hint: use memoryless determinacy of Büchi games without atoms, see (Thomas, 1990, Theorem 6.4).

Exercise 98. Consider the graph which is obtained by taking a disjoint union of all cliques, one for each size $n \in \{1, 2, \ldots\}$. This structure is not oligomorphic, but we can still consider pof sets with first-order definable subsets. Show that graph reachability is decidable.

5.3 Orbit-finite sets

In Section 4.2, we gave a more semantic notion of finiteness for the equality atoms, called orbit-finiteness. This notion, which is the central one for this book, extends to other structures by using automorphisms instead of permutations.

Definition 5.11 (Finite supports and orbit-finiteness)**.** Let $\mathbb{A}$ be a relational structure, and consider a set X that is equipped with an action of atom automorphisms, i.e. automorphisms of the structure $\mathbb{A}$.

- *Supports.* An element $x \in X$ is *supported* by a list of atoms $a_1, \ldots, a_n$ if

$$\pi(\bar{a}) = \bar{a} \quad \Rightarrow \quad \pi(x) = x$$

  holds for every automorphism π of the structure $\mathbb{A}$. We say that x is *finitely supported* if it is supported by some finite list of atoms.

- *Orbit-finite set.* The set X is called *orbit-finite* if every element $x \in X$ has finite support, and there are finitely many orbits under the group action.

We will only be interested in orbit-finite sets for atoms that are oligomorphic. The oligomorphic assumption will guarantee that basic operations, such as product $X \times Y$, can be implemented on orbit-finite sets.

Example 47. [Finitely supported subsets in the ordered atoms] Consider the ordered atoms $\mathbb{A} = (\mathbb{Q}, <)$. In this case, the automorphisms are order-preserving bijections.

[3] This type of game is called a *reachability game*. More general games, namely parity games, are studied in (Klin and Lelyk, 2017, Section 5.2)

Let us discuss the finitely supported elements of the powerset, i.e. the finitely supported subsets of $\mathbb{A}$. Consider a subset $X \subseteq \mathbb{A}$ which is supported by a tuple of atoms $\bar{a}$. We claim that X is a union of intervals (open, closed, open-closed or closed-open) whose endpoints are either $-\infty, \infty$, or appear in $\bar{a}$. Indeed, consider atoms b, c that are not in $\bar{a}$ and are not separated by an atom in $\bar{a}$ in terms of the order. There is an automorphism that fixes $\bar{a}$, and which maps b to c. Since the set X is supported by $\bar{a}$, it follows that $b \in X$ if and only if $c \in X$. □

In Chapter 4, we defined spof sets under the equality atoms, and we showed that they were the same as orbit-finite sets, up to equivariant bijections. The notion of *spof set* extends to oligomorphic structures (a pof set quotiented by an equivariant partial equivalence relation). Also, the characterization carries over, as stated in the following theorem.

Theorem 5.12. *Let $\mathbb{A}$ be an oligomorphic structure, and let X be a set that is equipped with an action of atom automorphisms. Then X is orbit-finite if and only if it admits an equivariant bijection with a spof set.*

Proof. Same proof as the special case for equality atoms from Theorem 4.6. Oligomorphism is used in the easier right-to-left implication: every spof set is orbit-finite. This is because pof sets without subquotients are orbit-finite by definition of oligomorphism, and taking a subquotient does not increase the number of orbits. □

A corollary of the above theorem is that orbit-finite sets enjoy the same closure properties as spof sets. For example, they are closed under Cartesian products $X \times Y$, since spof sets have this property. Not all results carry over to oligomorphic structures. For example, least supports can fail, as explained below.

Example 48. Consider an atom structure $\mathbb{A}$ which is the following graph:

Consider the spof set $\mathbb{A}/\sim$, where $\sim$ is the equivalence relation "in the same connected component". An element of this set, i.e. a connected component, is supported by any of the two atoms in it, but none of these supports is the least support. A similar phenomenon can be observed in the structure of two cliques from Example 43. In this case, each of the two cliques – when seen as an element of the finitely supported powerset – is supported by any atom that appears in it. □

Exercises

Exercise 99. Assume that the atoms are oligomorphic. Show that for every orbit-finite set X, there is some $d \in \{0, 1, \ldots\}$ and a surjective equivariant function $f : \mathbb{A}^d \to X$.

Exercise 100. Show that the atoms $(\mathbb{Q}, <)$ also have least supports.

Exercise 101. Show an example of oligomorphic atoms without least supports.

Exercise 102. Assume that the atoms are oligomorphic. Let X be a set with an action of group automorphisms, which is not known to be orbit-finite. Let $R \subseteq X \times X$ be an equivariant binary relation which is orbit-finite. Show that the transitive closure of R is also orbit-finite.

Exercise 103. Assume that the atoms are oligomorphic, and there are infinitely many atoms. Show that orbit-finite sets are not closed under taking finitely supported function spaces:

$$X \xrightarrow{\text{fs}} Y \quad \stackrel{\text{def}}{=} \quad \{ f : X \to Y \mid f \text{ is finitely supported } \}.$$

Exercise 104. Assume oligomorphic atoms. Let X, Y be orbit-finite sets and let F be an equivariant subset of the finitely supported function space from the previous exercise. Show that F is orbit-finite if and only if there is some $n \in \{0, 1, 2, \ldots\}$ such that every function $f \in F$ has a support of size at most n.

Exercise 105. Assume oligomorphic atoms. Show that in an orbit-finite set, for every atom tuple $\bar{a}$ there are finitely many elements supported by $\bar{a}$.

Exercise 106. Show that Exercise 84 fails in $(\mathbb{Q}, <)$.

Exercise 107. Show that Exercise 84 fails in some atoms, even for a relation R such that for every first argument, there are finitely many second arguments related by the relation.

Exercise 108. Assume that the atoms are oligomorphic. Let X be an orbit-finite set and let $\bar{a}$ be a tuple of atoms. Consider the family of equivalence relations on X which are supported by $\bar{a}$ and where every equivalence class is finite. Show that this family has a greatest element with respect to inclusion (i.e. a coarsest equivalence relation).

z

Exercise 109. Show that the following statement is true in the equality atoms but not in $(\mathbb{Q}, <)$. Let X be a set equipped with an action of atom automorphisms, where every element is finitely supported. Then X is orbit-finite if and only if: (***) for every equivariant family of finitely supported subsets of X which is totally ordered by inclusion, there is a maximal element.

Chapter 6

Homogeneous atoms

To define orbit-finiteness, we need atoms that are oligomorphic. How does one get oligomorphic structures?

This chapter is devoted to a method of producing oligomorphic structures, which is called the Fraïssé limit[1]. The idea behind the Fraïssé limit is that it inputs a class of finite structures, sufficiently well-behaved, and outputs a single countably infinite structure which contains all the finite structures, and does so in a certain homogeneous way. The Fraïssé limit can be applied to classes of finite structures such as all finite total orders, all finite directed graphs, all equivalence relations on finite sets, etc.

6.1 Homogeneous structures

Before defining homogeneous structures, we begin by recalling some terminology from logic. In the previous chapter, there was one structure, and we used logic to define relations on this structure. In this chapter, there will be many structures, but we will still want to compare them using a single formula. To do this, we use the notion of a *vocabulary*: this is a set of names for relations, each one with an associated arity in $\{0, 1, \ldots\}$. Here are some examples of vocabularies:

$$\underbrace{x \leq y}_{\substack{\text{the vocabulary for}\\ \text{ordered structures}\\ \text{has one binary relation}}} \qquad \underbrace{\text{edge}(x, y)}_{\substack{\text{the vocabulary for graphs}\\ \text{has one binary relation}}} \qquad \underbrace{x + y = z \quad x \times y = z}_{\substack{\text{the vocabulary for rings}\\ \text{has two ternary relations}}}.$$

Note that the first two vocabularies are essentially the same, since they both have one relation of arity two. However, is useful to give different names to convey different intentions. In the third vocabulary, we use ternary relations instead of binary functions – this is because we want to stick to relational vocabularies for simplicity. A structure over a given vocabulary is a structure in the sense of Definition 5.1, together with a function (called the *interpretation* of the vocabulary) that assigns each relation

[1] This is a basic notion in model theory. For further information, see e.g. (Hodges, 1993, Section 7).

name from the vocabulary to relation in the structure of the same arity. Thanks to the interpretation, we can evaluate a formula over the vocabulary in any structure over this vocabulary.

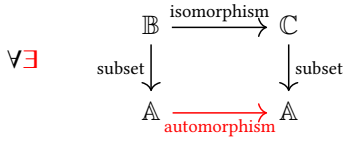
Consider two structures $\mathbb{A}, \mathbb{B}$ over the same vocabulary. An *embedding* $f : \mathbb{A} \to \mathbb{B}$ is any injective function from the universe of $\mathbb{A}$ to the universe of $\mathbb{B}$ which preserves and reflects the relations in the following sense

$$\underbrace{R(a_1, \ldots, a_n)}_{\text{in } \mathbb{A}} \quad \Leftrightarrow \quad \underbrace{R(f(a_1), \ldots, f(a_n))}_{\text{in } \mathbb{B}}.$$

An *isomorphism* is the special case of an embedding where the function is a bijection. An *embedded substructure* of a structure is defined to be any structure that embeds into it. A *substructure* is the special case where the embedding is simply an inclusion map. We will be mainly interested in (embedded or not) substructures that are finite, i.e. have finite universes. Here is the fundamental definition for this chapter.

Definition 6.1 (Homogeneous structure)**.** A structure is called *homogeneous* if every isomorphism between finite substructures extends to a full automorphism of the entire structure.

Here is a diagram that describes the above definition.



Example 49. The equality atoms and the ordered atoms $(\mathbb{Q}, <)$ are homogeneous. Let us do the proof for the ordered atoms. A finite substructure is the same as a choice of d rational numbers $x_1 < \cdots < x_d$. Any two such choices will be isomorphic, assuming the same dimension d. If we take two such choices, they will be in the same orbit, i.e. the isomorphism will extend to an automorphism. □

Example 50. Consider the structure which consists of finite subsets of natural numbers, equipped with a binary relation for subset inclusion:

$$\mathbb{A} = (\mathsf{P}_{\text{fin}}(\mathbb{N}), \subseteq).$$

We will show that this structure is not homogeneous. Consider a finite substructure $\mathbb{B}$ that has only one element, namely the empty set, and another finite substructure $\mathbb{C}$ that also has only one element, namely the singleton set $\{1\}$. As finite structures, they are isomorphic – the subset relation connects the unique element with itself in both of them. However, there is no automorphism of $\mathbb{A}$ that maps the empty set to a nonempty set. This structure is also not oligomorphic, because it has infinitely many orbits already in $\mathbb{A}^1$, namely sets of different finite sizes will be in different orbits. □

In principle, oligomorphism and homogeneity are incomparable notions, as explained in the following two examples.

Example 51. [Oligomorphic $\not\Rightarrow$ homogeneous] Every finite structure is oligomorphic, but not every finite structure is a homogeneous. For example, consider the three element path

Let the vertices be $1, 2, 3$. The substructures $\{1\}$ and $\{2\}$ are isomorphic, but there is no automorphism that maps 1 to 2. □

Example 52. [Homogeneous $\not\Rightarrow$ oligomorphic] Consider an infinite structure which has one unary relation for every possible singleton. This structure has no automorphism, and therefore it has infinitely many orbits. However, it is homogeneous for the trivial reason that the only isomorphism between finite substructures are between identical subsets. □

However, the differences exhibited in the above examples are rather superficial. Every oligomorphic structure can be trivially made into a homogeneous one by adding (infinitely many) relations: we can simply add a d-ary relation for every orbit in $\mathbb{A}^d$. An infinite vocabulary may indeed be needed, as witnessed by the bit-vector structure that will be discussed in Section 6.3.2. For the converse implication, the following theorem shows that most reasonable homogeneous structures are in fact oligomorphic.

Theorem 6.2. *If a structure is homogeneous, then it is oligomorphic if and only if*

(*) for every $d \in \{0, 1, \ldots\}$, it has finitely many substructures of size d, up to isomorphism.

Proof. If we take tuples $(a_1, \ldots, a_d)$ and $(b_1, \ldots, b_d)$ that are in the same orbit, then the function $a_i \mapsto b_i$ is an isomorphism between the substructures generated by the tuples. Therefore, if there are finitely many orbits in $\mathbb{A}^d$, then there are finitely many kinds of substructures of size d, up to isomorphism, which proves the left-to-right implication. For the converse implication, we use homogeneity: the orbit of a tuple is uniquely determined by the isomorphism type of the induces substructure, and the order and repetitions of the elements in the tuple, which can be chosen in finitely many ways. □

A corollary of the above theorem is that for finite vocabularies, homogeneity implies oligomorphism. This is because condition (*) will automatically hold in the presence of a finite vocabulary.

One of the fundamental properties of oligomorphic structures was that equivariant relations were exactly those that could be defined in first-order logic, see Theorem 5.7. For homogeneous structures, quantifier-free formulas are enough.

Theorem 6.3. *Consider a homogeneous structure $\mathbb{A}$.*

1. *Two tuples from $\mathbb{A}^d$ are in the same orbit if and only if they satisfy the same quantifier-free formulas.*
2. *If $\mathbb{A}$ additionally satisfies condition (*) from Theorem 6.2, then the equivariant subsets of $\mathbb{A}^d$ are exactly those that are definable by quantifier-free formulas.*

Proof. We begin with the equivalence in the first item. The left-to-right implication is immediate: the truth-value of quantifier-free formula does not change when an automorphism is applied. Conversely, if two tuples of atoms satisfy the same quantifier-free formulas, then one can build an isomorphism between the substructures generated by them, which will extend to an automorphism of the entire structure by homogeneity.

The second item follows from the first item, and the observation that under assumption (*), there are finitely many possible quantifier-free formulas with d variables, up to logical equivalence. □

Exercises

Exercise 110. Which finite graphs are homogeneous?

6.2 The Fraïssé limit

In this section, we describe the Fraïssé limit, which is a way – in fact the only way – of constructing countable homogeneous structures. Before defining the Fraïssé limit, consider the following problem: for a class $\mathscr{A}$ of finite structures, find some (possibly infinite) structure $\mathbb{A}$ such that

$$\mathscr{A} = \underbrace{\{\, \mathbb{B} \mid \ \mathbb{B} \text{ is a finite structure that embeds into } \mathbb{A} \ \}}_{\text{this is called the } \textit{age} \text{ of the structure } \mathbb{A}}.$$

For example, if $\mathscr{A}$ is the class of all finite structures over an empty vocabulary, then it is the age of any infinite structure over the empty vocabulary. If $\mathscr{A}$ is the class of finite total orders, then it is the age of any infinite total order, such as

$$(\mathbb{N}, <) \quad (\mathbb{Z}, <) \quad (\mathbb{Q}, <) \quad (\mathbb{R}, <).$$

However, if we want the total order to be countable and homogeneous, then the only choice is the rational numbers. Finally, not every class arises as the age of some structure. A necessary condition is that every two structures from $\mathscr{A}$ can be embedded into a single structure from $\mathscr{A}$, since this is a property that will hold for the age of a single structure $\mathbb{A}$. For this reason, the class

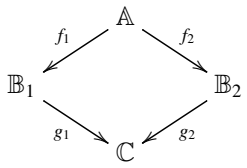
$$\mathscr{A} = \{\, G \mid \ G \text{ is a graph with at most 10 edges} \ \}$$

is not the age of any structure. The purpose of this chapter is to identify conditions which guarantee that $\mathscr{A}$ can be obtained as the age of some structure, and furthermore we want this structure to be homogeneous. These conditions are described in terms of amalgamations, so we begin by defining amalgamation.

Definition 6.4 (Amalgamation)**.** An *instance of amalgamation* is two embeddings with a common source:

$$\begin{array}{ccccc} & & \mathbb{A} & & \\ & \overset{f_1}{\swarrow} & & \overset{f_2}{\searrow} & \\ \mathbb{B}_1 & & & & \mathbb{B}_2 \end{array} \tag{6.1}$$

A *solution* of the instance is a structure $\mathbb{C}$ and two embeddings g_1, g_2 such that the following diagram commutes:



(6.2)

Definition 6.5 (Fraïssé class)**.** A *Fraïssé class* is a class of finite structures over a common vocabulary which is closed under isomorphism, substructures, and also:

- it is *closed under amalgamation*, which means that for every instance of amalgamation which uses structures from the class, there is a solution which also uses a structure from the class.

A Fraïssé class is called *countable* if it has countably many structures, up to isomorphism. We are now ready to state the Fraïssé theorem, which says that Fraïssé classes are in one-to-one correspondence with countable homogeneous structures.

Theorem 6.6 (Fraïssé Theorem)**.** *The map*

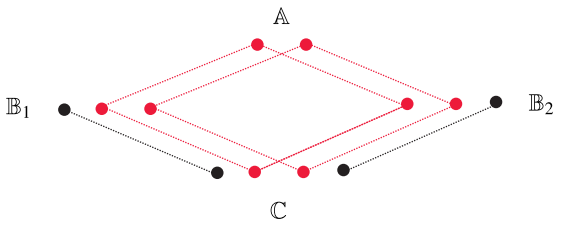
$$\mathbb{A} \qquad \mapsto \qquad \textit{age of } \mathbb{A}$$

is a bijection between countable homogeneous structures (modulo isomorphism) and countable Fraïssé classes. In other words:

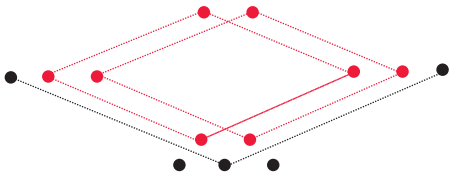
1. *the age of every countable homogeneous structure is a countable Fraïssé class; and*
2. *every countable Fraïssé class is obtained this way; and*
3. *if two countable homogeneous structures have the same age, then they are isomorphic.*

The inverse of the age operation, i.e. the map which inputs a Fraïssé class and outputs the corresponding countable homogeneous structure (which is unique up to isomorphism thanks to the above theorem), is called the *Fraïssé limit*. Before proving Theorem 6.6, we give some examples and non-examples of Fraïssé classes. In all these examples, closure under substructures and isomorphism is immediate, and only amalgamation need be discussed.

Example 53. Consider the class of all finite structures over an empty vocabulary (in which case formulas can talk only about equality). This class is closed under amalgamation, by taking the disjoint union of two sets with a common subset. Here is an example of an instance of amalgamation and its solution:

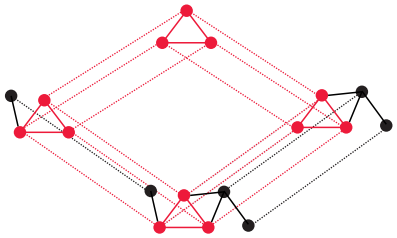


When drawing amalgamation diagrams, we use the red colour for elements of $\mathbb{A}$. In general, the same instance might have several solutions. Here is an example of a different solution to the instance above:



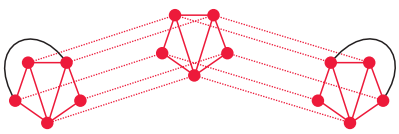
In fact, the above instance has infinitely many solutions (because the solution can be arbitrarily large). Note how the second solution uses the same black element as the target of both black nodes in the second row. □

Example 54. Consider the class of finite undirected graphs. In other words, this is the class of all finite structures over a vocabulary which has one binary relation that is required to be symmetric and irreflexive. This class is closed under amalgamation (the same argument works for directed graphs), by taking the disjoint union of two directed graphs with a common induced subgraph. Here is an example:



As in Example 53, there are other solutions to the above instance. More generally, for every relational vocabulary, the class of all finite structures over this vocabulary is closed under amalgamation. In particular, by Theorem 6.6, each of these classes has a Fraïssé limit. The limit for undirected graphs will be discussed in more detail in Section 6.3.1. □

Example 55. Consider the class of finite planar graphs. To simplify this example, we assume that a graph is represented (unlike in Example 54) as a structure where the universe is the vertices and edges of the graph, and there is a binary relation for incidence between edges and vertices. (This representation of graphs means that an embedding can add edges without adding vertices, i.e. embedding corresponds to subgraphs, and not induced subgraphs.) Under this representation, the class of planer graphs is not closed under amalgamation. Here is an instance without a solution:

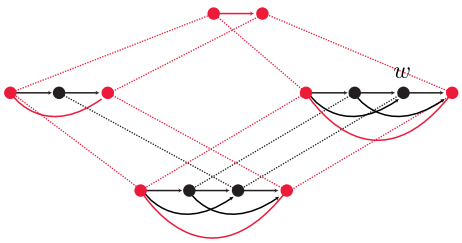


Any hypothetical solution to the above instance would have the 5-clique as a minor, and would therefore not be planar. A similar but more elaborate example would show failure of amalgamation for planar graphs under the modelling of graphs used by Example 54, where the universe of the structure is the vertices and there is a binary relation for the edges. □

Example 56. Consider directed graphs where the edge relation is a partial successor, i.e. vertices have out-degree and in-degree at most one, and no loops. The class is not closed under amalgamation, here is an instance without a solution:

□

Example 57. Consider the class of finite total orders. This class is closed under amalgamation. Here is an example of an instance of amalgamation and its solution:

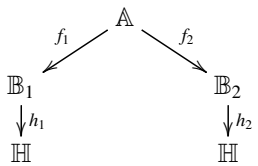


□

We now begin the proof of the Fraïssé Theorem. We first show item 1, which says that the age of a countable homogeneous structure is a Fraïssé class. Next, we prove a slightly stronger result, which does not assume countability.

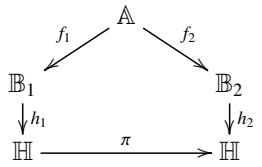
Lemma 6.7. *For every homogeneous structure, not necessarily countable, its age is a Fraïssé class.*

Proof. The only nontrivial part is amalgamation. Let $\mathbb{H}$ be a homogeneous structure. Consider an instance of amalgamation which uses structures that embed into $\mathbb{H}$, as in the following diagram (all arrows are embeddings):



The diagram distinguishes the targets of h_1 and h_2 because the embeddings $h_1 \circ f_1$ and $h_2 \circ f_2$ need not be the same embedding of $\mathbb{A}$ in $\mathbb{H}$. However, the images of

both of these embeddings are isomorphic finite substructures of $\mathbb{H}$. Therefore, by homogeneity there is an automorphism π which extends this partial automorphism. In other words, the following diagram commutes:

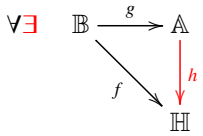


If we restrict the right copy of $\mathbb{H}$ to the union of the images of the maps h_2 and $\pi \circ h_1$, then we get a solution of amalgamation. □

If a homogeneous structure is countable, then it has countably many embedded finite substructures. Therefore, by the above lemma, the age of a countable homogeneous structures is a countable Fraïssé class. We now establish item 3 in the theorem, which says that the age uniquely identifies a countable homogeneous structure.

Lemma 6.8. *A countable structure $\mathbb{H}$ is homogeneous if and only if:*

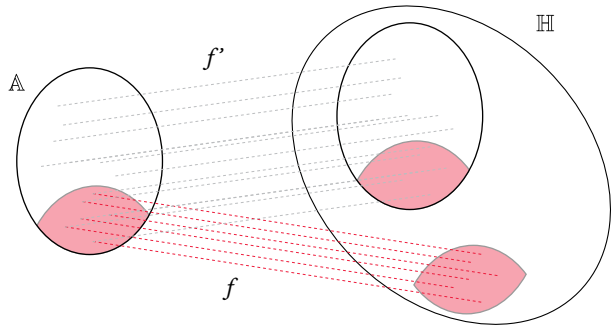
(*) If $\mathbb{A}, \mathbb{B}$ are finitely generated substructures of $\mathbb{H}$ then



Furthermore, countable homogeneous structures with the same age are isomorphic.

Proof.

- *Homogeneous structures satisfy (*).* Let g, f be as in (*). We assume without loss of generality that g is an inclusion. Let f' be an embedding of $\mathbb{A}$ into $\mathbb{H}$, which exists by the assumption that $\mathbb{A}$ is a substructure. Here is a picture:

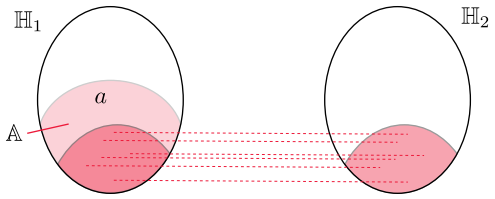


By following the inverse of f and then f', we get a partial automorphism between two finitely generated substructures of $\mathbb{H}$, namely the two red parts on the right. By homogeneity, this partial automorphism extends to a full automorphism. The function f' composed with the inverse of that automorphism is the desired embedding.

- *Structures satisfying (*) are homogeneous.* Here we use countability. The following claim, in the special case of $\mathbb{H} = \mathbb{H}_1 = \mathbb{H}_2$, shows that $\mathbb{H}$ is homogeneous.

  Claim 6.9. *Let $\mathbb{H}_1, \mathbb{H}_2$ be countable structures with the same age. If both satisfy (*), then every partial isomorphism between finite substructures of $\mathbb{H}_1$ and $\mathbb{H}_2$ extends to a full isomorphism.*

  Proof. Let f be an isomorphism between structures in the ages of $\mathbb{H}_1$ and $\mathbb{H}_2$, respectively, and let a be an element of $\mathbb{H}_1$. Let $\mathbb{A}$ be the substructure of $\mathbb{H}_1$ whose universe is a plus the domain of f. Here is a picture:

  The structure $\mathbb{A}$ is in the age of $\mathbb{H}_1$, and therefore by the assumption of the claim it embeds into $\mathbb{H}_2$. By (*), f extends to an embedding of $\mathbb{A}$ into $\mathbb{H}_2$. This argument and a symmetric one where a is in $\mathbb{H}_2$ establishes that:

  (**) For every isomorphism between structures in the ages of $\mathbb{H}_1$ and $\mathbb{H}_2$, respectively, and every element a of either $\mathbb{H}_1$ or $\mathbb{H}_2$, the partial isomorphism can be extended to be defined also on a.

  The conclusion of the claim follows from (**) using a back-and-forth construction. Define inductively a sequence of partial isomorphisms between finitely generated substructures of $\mathbb{H}_1$ and $\mathbb{H}_2$, such that the next one extends the previous one, and every element of both structures appears eventually in the source or target of a partial isomorphism from the sequence. The full isomorphism is then the limit of these partial isomorphisms. □

- *Homogeneous structures are uniquely determined by their finitely generated substructures.* By Claim 6.9 applied to the empty partial isomorphism between $\mathbb{H}_1$ and $\mathbb{H}_2$, we see that countable homogeneous structures are uniquely determined – up to isomorphism – by their age.

□

To finish the proof of Fraïssé Theorem, we need to show item 2.

Lemma 6.10. *Every countable Fraïssé class $\mathscr{A}$ arises as the age of some countable homogeneous structure.*

Proof. Choose some enumeration

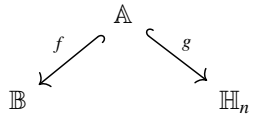
$$\mathbb{A}_1, \mathbb{A}_2, \dots \tag{6.3}$$

of the structures in $\mathscr{A}$, which represents every structure up to isomorphism. We define a sequence

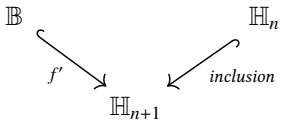
$$\mathbb{H}_0 \subseteq \mathbb{H}_1 \subseteq \cdots \tag{6.4}$$

of structures in $\mathscr{A}$ as follows. Choose the first structure $\mathbb{H}_0$ arbitrarily, say the empty structure. A new structure is obtained by applying the following claim.

Claim 6.11. *Suppose that $\mathbb{H}_n$ is already defined. There is a structure $\mathbb{H}_{n+1} \supseteq \mathbb{H}_n$ in $\mathscr{A}$ such that for every instance of amalgamation*



where both $\mathbb{A}, \mathbb{B}$ are among the first n structures in the enumeration of $\mathscr{A}$, there is a solution of the form



Proof. There are finitely many possible instances of amalgamation as in the claim, because $\mathbb{A}$ and $\mathbb{B}$ can be chosen in finitely many ways, and there are finitely many possible embeddings between two finite structures. Let m be the number of instances. By induction on $i \in \{1, \dots, m\}$, we create a structure $\mathbb{H}^i_{n+1}$ that solves the first i instances; once we have done this we can use $\mathbb{H}^m_{n+1}$ as the solution for all instances. The induction step is proved by applying amalgamation to the previous solution. □

Define $\mathbb{H}$ to be the limit (i.e. union) of the sequence $\mathbb{H}_1, \mathbb{H}_2, \dots$. By construction, $\mathbb{H}$ satisfies condition (*) from Lemma 6.8, and is therefore homogeneous.

To complete the proof, we justify that the age of $\mathbb{H}$ is exactly $\mathscr{A}$. Every finite structure that embeds into the limit $\mathbb{H}$ must embed into some $\mathbb{H}_n$, and is therefore in $\mathscr{A}$, because $\mathbb{H}_n \in \mathscr{A}$ and the class is closed under substructures. Therefore, the age of $\mathbb{H}$ is contained in $\mathscr{A}$. Let us prove the converse inclusion. Suppose that $\mathbb{A} \in \mathscr{A}$. At some point n in the enumeration, we have seen both $\mathbb{A}$ and the empty structure. Therefore, $\mathbb{H}_{n+1}$ will contain a solution to an instance of amalgamation where the empty structure is embedded into both $\mathbb{A}$ and $\mathbb{H}_{n+1}$. This means that $\mathbb{H}_{n+1}$ contains an isomorphic copy of $\mathbb{A}$. □

This completes the proof of Fraïssé Theorem.

Computability. The Fraïssé limit not only exists, but under mild assumptions on the Fraïssé class, it can be computed. What does it mean to compute an infinite structure? This is formalized in the following theorem, which shows that one perform basic computational operations, such as counting orbits, deciding the first-order theory, and representing elements.

Theorem 6.12. *Let $\mathscr{A}$ be a Fraïssé class such that:*

(*) there is an algorithm that inputs d and returns a finite list of all structures that represents all structures of size d in $\mathscr{A}$, up to isomorphism.

Then its Fraïssé limit, call it $\mathbb{A}$, has the following properties:

1. *it is oligomorphic, and given d one can compute the number of orbits in $\mathbb{A}^d$;*
2. *it has effective quantifier elimination, i.e. for every first-order formula, one can compute an equivalent one that is quantifier-free;*
3. *there is a function $\rho : 2^* \to \mathbb{A}$, called a* representation*, which has the following properties (when atoms are used in algorithms, they are represented as strings using the representation):*
   (a) *every atom is represented by at least one string;*
   (b) *given a first-order formula $\varphi(x_1, \ldots, x_d)$ and $a_1, \ldots, a_d \in \mathbb{A}$, one can decide if*
   $$\mathbb{A} \models \varphi(a_1, \ldots, a_d);$$
   (c) *given two tuples in $\mathbb{A}^d$, decide if they are in the same orbit.*

Proof. We begin with item 1. By assumption (*), there are finitely many substructures of size d in $\mathscr{A}$, up to isomorphism. Therefore, $\mathbb{A}$ is oligomorphic by Theorem 6.2. An orbit in $\mathbb{A}^d$ is the same thing as a substructure with at most d elements, together with a list of length d that covers all of its elements, possibly with repetitions. Such objects can be counted, up to isomorphism, using assumption (*).

We now show item 2, about quantifier elimination. We assume that "true" and "false" are quantifier-free formulas; these will be the only possible formulas when we apply the quantifier elimination to a sentence, i.e. a formula without free variables. The proof is by induction on the size of the formula. The only non-trivial case is eliminating a single quantifier, say an existential one (because eliminating a universal quantifier reduces to this case by De Morgan's laws):

$$\exists x \underbrace{\varphi(x_1, \ldots, x_n, x)}_{\text{quantifier free}}.$$

The inner formula φ can be seen as describing structures with $n+1$ distinguished elements; with the distinguished elements not being necessarily pairwise distinct. Let us write $\mathscr{A}_\varphi$ for the corresponding structures, i.e. this is the class

$$\mathscr{A}_\varphi = \{(\mathbb{A}, \overbrace{a_1, \ldots, a_n}^{\bar{a}}, a) : \mathbb{A} \in \mathscr{A} \text{ and } a_1, \ldots, a_n, a \text{ are elements that satisfy } \varphi\}.$$

Up to isomorphism, the above class is finite and can be computed thanks to assumption (*). Because the Fraïssé limit is homogeneous, a tuple $\bar{a}a$ in the Fraïssé limit satisfies φ if and only if $\mathscr{A}_\varphi$ contains the substructure generated by $\bar{a}$ (together with the distinguished $\bar{a}$). Define $\mathscr{A}_{\exists x\varphi}$ to be the following projection of $\mathscr{A}_\varphi$: for each $(\mathbb{A}, \bar{a}a) \in \mathscr{A}_\varphi$, remove the last element a from the list of distinguished elements. A tuple $\bar{a}$ in the Fraïssé limit satisfies the quantified formula $\exists x\varphi$ if and only if $\mathscr{A}_{\exists x\varphi}$ contains the substructure generated by $\bar{a}$ (together with the distinguished $\bar{a}$). This property can be expressed using a quantifier-free formula.

We now prove the last item 3, about the representation. Here, we revisit the construction of the Fraïssé limit in the proof of Lemma 6.10. In that proof, we started off with an enumeration, see (6.3), which represents all structures in $\mathscr{A}$ up to isomorphism. Thanks to assumption (*), we can assume that this enumeration is effective in the following sense: there is an algorithm that inputs n and returns the n-th structure $\mathbb{A}_n$ in the enumeration. The construction in Claim 6.11 preserves this notion of effectiveness, and therefore also the sequence $\mathbb{H}_n$ is effective. Since the Fraïssé limit is defined to be the union of the latter enumeration, it follows that the Fraïssé limit is effective in the sense that one can define a surjective representation $\rho : 2^* \to \mathbb{H}$ that allows us to test if a given tuple of elements satisfies a given relation from the vocabulary. (The string representing an element from $\mathbb{H}$ stores the following information: at which stage n did the element appear in $\mathbb{H}_n$, and which element of $\mathbb{H}_n$ it is.) If we can decide the relations from the vocabulary, then we can decide quantifier-free formulas, and so we can also decide first-order formulas, as required by item 3b, thanks to the previous item about quantifier elimination. The last part of the theorem, in item 3c, is about deciding if two tuples are in the same orbit. By Theorem 6.3, we know that two tuples are in the same orbit if and only if they satisfy the same quantifier-free formulas. Although the vocabulary is in principle infinite, we can use assumption (*) to show that for every d, there is some finite part of the vocabulary such that quantifier-free formulas using only that part are enough to distinguish different orbits in $\mathbb{A}^d$. In combination with the previous observations about deciding quantifier-free formulas, we can get an effective criterion for checking if two tuples from $\mathbb{A}^d$ are in the same orbit. □

All Fraïssé classes discussed in this chapter satisfy the assumptions of the above theorem. In particular, the corresponding Fraïssé limit will have a decidable first-order theory, thanks to the special case of item 3b for formulas without free variables. Therefore, we can apply Theorem 5.10 to decide graph reachability. In the Chapter 7, we will see many other examples of algorithms, beyond graph reachability, which can be used for atoms that arise a Fraïssé limit. Before we do that, however, we present several interesting examples of Fraïssé limits, which will illustrated the scope of applicability for the algorithms that will be presented in Chapter 7.

Exercises

Exercise 111. Consider the class of all finite partial orders, i.e. binary relations that are reflexive and transitive. Show that this class is closed under amalgamation.

Exercise 112. Are series parallel graphs closed under amalgamation?

Exercise 113. Show a Fraïssé class where solutions to amalgamation necessarily violate the following condition:

(*) the intersection of the images of g_1 and g_2, as per diagram (6.2), is exactly the image of $\mathbb{A}$.

Exercise 114. Assume a finite relational vocabulary. Suppose that $\mathscr{A}$ is a class of structures that satisfies the assumptions of Theorem 6.6, and let $\mathbb{A}$ be its Fraïssé limit. Show that if membership in $\mathscr{A}$ is decidable, $\mathbb{A}$ is an effective structure.

Exercise 115. Let $\mathscr{A}$ be a class of structures over a finite vocabulary, possibly including functions, which:

1. has decidable membership;
2. is closed under substructures, isomorphism and amalgamation;
3. given $k \in \mathbb{N}$ one can compute some $n \in \mathbb{N}$ such that structures in $\mathscr{A}$ with k generators have size at most n.

Show that the Fraïssé limit of $\mathscr{A}$ has a decidable first-order theory with constants and a computable Ryll-Nardzewski function.

Exercise 116. Define *monadic second-order logic* (MSO) to be the extension of first-order logic where one can also quantify over sets of vertices. A famous result on MSO is Rabin's Theorem[2], which says that the structure $\{0,1\}^*$ equipped with functions $x \mapsto x0$ and $x \mapsto x1$ has decidable MSO theory, i.e. one can decide if a sentence of MSO is true in it. Show that $(\mathbb{Q}, <)$ has decidable MSO theory.

Exercise 117. If Σ is a finite alphabet. We model a word $w \in \Sigma^*$ as a structure, where the universe is positions in w, there is a binary predicate $<$ for the order relation, and for every label $a \in \Sigma$ there is a unary predicate $a(x)$. We denote the vocabulary used for this structure by $\Sigma_<$. Show that for every regular language $L \subseteq \Sigma^*$ there is a homogeneous structure $\mathbb{A}$ over a vocabulary containing $\Sigma_<$ such that the age of $\mathbb{A}$ after restricting to $\Sigma_<$ is exactly the structures corresponding to L.

6.3 Examples of homogeneous atoms

We end this chapter with three extended examples of homogeneous structures.

6.3.1 The random graph

We begin with the Fraïssé limit of all finite undirected graphs. As shown in Example 54, this is a Fraïssé class, and therefore it has a Fraïssé limit. Call this limit the *random graph*. The name is justified by the following observation.

Theorem 6.13. *Consider a countably infinite undirected graph, where each the presence/absence of an edge is chosen independently with equal probability one half*[3]*. Almost surely (i.e. with probability one) this graph is isomorphic to the random graph.*

[2] For an introduction to MSO and Rabin's Theorem, see (Thomas, 1990, Theorem 6.8).

[3] The conclusion of the theorem would not change if we used a different distribution, e.g. there would be an edge with probability 0.99.

Proof. Let us write $\mathbb{H}$ for the graph that is chosen randomly. For a finite graph G, and a function h from vertices of an induced subgraph $F \subseteq G$ to vertices of $\mathbb{H}$, consider the event: "either h is not an embedding, or it can be extended to an embedding of G". This event happens almost surely because failing the event would require infinitely many independent random events that go wrong. Since there are countably many choices of $F \subseteq G$ and functions h, up to isomorphism, it follows that almost surely the graph $\mathbb{H}$ satisfies condition (*) of Lemma 6.8, and therefore it is isomorphic to the random graph. □

Since the class of finite undirected graphs is clearly countable, its Fraïssé limit is oligomorphic and has all the computability properties in the conclusion of Theorem 6.12. It follows that problems such as graph reachability or automaton emptiness are decidable, assuming that the inputs are orbit-finite automata, with orbit-finite sets represented as spof sets.

Exercises

Exercise 118. Assume that the atoms are the random graph. Is the language

$$\{a_1 \cdots a_n \in \mathbb{A} : \text{the subgraph induced by } a_1, \ldots, a_n \text{ is connected}\}$$

recognised by a nondeterministic orbit-finite automaton?

Exercise 119. Assume that the atoms are the random graph. Give examples and non-examples of graph properties X such that the following language is recognised by a nondeterministic orbit-finite automaton:

$$L_X = \{a_1 \cdots a_n : \text{ the subgraph induced by } a_1, \ldots, a_n \text{ satisfies } X\}.$$

To recognise L_X, the automaton should be prepared for an arbitrary enumeration of the vertices of the graph, possibly with repetitions.

Exercise 120. Assume that the atoms are the random graph. Show that there is no finitely supported total order on the random graph.

Exercise 121. Show that there is no orbit-finite automaton, even nondeterministic, which recognises the language of width k path decompositions.

Exercise 122. Assume that the atoms are the random graph. Show that for every MSO formula $\varphi(x_1, \ldots, x_n)$ with free variables that represent vertices (not sets of vertices) there is formula of first-order logic which is equivalent on the random graph. Nevertheless, there is no algorithm which computes such equivalent formulas.

Exercise 123. Assume that the atoms are the random graph. Show that solving equations, as discussed in Section 7.4, is undecidable.

6.3.2 Bit vectors

This section is about the Fraïssé limit of finite vector spaces over the two element field. These atoms will also be discussed in Chapter 9, where we will show that, over these atoms, deterministic polynomial time orbit-finite Turing machines are weaker than the nondeterministic ones.

For the rest of this section, we only study vector spaces over the two element field, so we say vector space with the implicit assumption that the underlying field is the two element field. Every vector space of finite dimension (which is equivalent to having finitely many vectors) is isomorphic to

$$(\{0,1\}^d, +) \qquad \text{for some } d \in \{1,2,\ldots\}$$

where addition is modulo two. We model a – possibly infinite – vector space V as a structure over the following infinite vocabulary: for every $d \in \{0,1,\ldots\}$ there is a relation which selects d-tuples of vectors that are linearly independent. Here, a d-tuple $\bar{v} \in V^d$ is called linearly independent if it does not satisfy any non-trivial dependency

$$\alpha_1 v_1 + \cdots + \alpha_d v_d = 0,$$

where non-trivial means that at least one of the coefficients α_i is nonzero. In particular, if the tuple contains a repetition, then it is linearly dependent. If the vector space has finite dimension, then the relation will not select any tuple, once d exceeds the dimension.

It is not hard to see that finite vector spaces are a Fraïssé class. Embeddings are the same thing as injective linear maps. To amalgamate two vector spaces, of dimensions say d_1 and d_2, one needs a vector space of dimension $\max(d_1, d_2)$. Therefore, there is a Fraïssé limit of the finite vector spaces; and thanks to Theorem 6.12 this limit is a countably oligomorphic structure.

One can also construct the Fraïssé limit explicitly. The Fraïssé limit must be a vector space, since any violation of the vector space axioms would need to happen already in a finitely generated substructure. Since the Fraïssé limit is countable, its dimension must be countable, and since the Fraïssé limit embeds all finite vector spaces, its dimension must be infinite. Therefore, the Fraïssé limit is a vector space of countably infinite dimension. Up to isomorphism, there is a unique vector space like this. One way of representing this unique vector space is as follows. The elements are *bit vectors*, which are defined to be ω-sequences of zeroes and ones which have finitely many ones (if we allowed infinitely many ones, the resulting vector spaces would have uncountable dimension). By ignoring trailing zeroes, a bit vector can be represented as a finite sequence, such as 00101001. Define the *bit vector atoms* to be the bit vectors equipped with a function for coordinate-wise addition modulo two:

$$01011 + 11001 = 10010 = 1001.$$

An example basis consists of bit vectors which have a 1 on the n-th coordinate:

$$1, 01, 001, 0001, \ldots.$$

Another example of a basis is

$$1, 11, 111, 1111, \ldots.$$

Least supports. We prove below that for the bit vector atoms, a version of the Least Support Theorem is true. For bit vectors, least supports are not unique as sets, but as spanned subspaces. For example, the pair of atoms $(01, 10)$ is supported by itself, but it is also supported by $(11, 01)$. More generally, the following lemma shows that supporting and spanning are the same concepts, when talking about tuples of atoms.

Lemma 6.14. *Assume the bit vector atoms. An atom tuple $\bar{a}$ supports an atom tuple $\bar{b}$ if and only if all atoms in $\bar{b}$ are spanned by $\bar{a}$.*

Proof. The right-to-left implication is immediate. For the converse implication, suppose that some atom in $\bar{b}$ is not spanned by $\bar{a}$. By the Steinitz exchange lemma, this atom can be mapped to some other atom by a $\bar{a}$-automorphism. □

We are now ready to state the Least Support Theorem for bit vector atoms.

Theorem 6.15 (Least Support Theorem)**.** *Assume the bit vector atoms. Let X be a set equipped with an action of atom automorphisms. If $x \in X$ has finite support, then there exists a tuple $\bar{a}$ of atoms which supports x, and which is least in the sense that if $\bar{b}$ supports x, then $\bar{a}$ supports $\bar{b}$.*

Proof. Without loss of generality, we assume that X has one orbit. The proof follows the same lines as the proof for the equality atoms, except that vector independence plays the role of equality. Let us write $\mathbb{A}^{\langle d \rangle}$ for the set of d-tuples of atoms which are linearly independent. This is a one orbit set.

Lemma 6.16. *There is an equivariant function*

$$f : \mathbb{A}^{\langle d \rangle} \to X$$

which satisfies the following condition[4] *for every $\bar{a}, \bar{b} \in \mathbb{A}^{\langle d \rangle}$:*

$$f(\bar{a}) = f(\bar{b}) \quad \Rightarrow \quad \text{every atom in } \bar{a} \text{ is spanned by } \bar{b} \text{ and vice versa.}$$

Proof. We start with some function $f : \mathbb{A}^{\langle d \rangle} \to X$ that is equivariant, but which does not necessarily satisfy the condition in the lemma. Such a function can be found, by taking some tuple $\bar{a}$ of independent atoms that supports some element $x \in X$, and extending it to an equivariant function. We will now show that either f satisfies the condition, or the dimension d can be made smaller. By iterating this argument at most d times, we get the conclusion of the lemma.

Suppose that f violates the condition in the lemma, as witnessed by tuples $\bar{a}$ and $\bar{b}$, which have the same image under f but do not span each other. Some coordinates in $\bar{a}$ are spanned by $\bar{b}$, but at least one coordinate is not. Without loss of generality, we assume that the first i coordinates in the tuple $\bar{a}$ are not spanned by $\bar{b}$, and the remaining coordinates are spanned by $\bar{b}$. In other words, the tuple

$$(\underbrace{a_1, \ldots, a_i}_{\substack{\text{first } i \text{ atoms} \\ \text{in the tuple } \bar{a}}}, \underbrace{b_1, \ldots, b_d}_{\substack{\text{all atoms in} \\ \text{the tuple } \bar{b}}})$$

[4] The conclusion of the implication in the lemma is equivalent to saying that $\bar{a}$ and $\bar{b}$ have the same algebraic closure, in the model theory sense, see (Hodges, 1993, Chapter 4).

is linearly independent. Since the vector space $\mathbb{A}$ has infinite dimension, one can choose $a'_1, \ldots, a'_i \in \mathbb{A}$ which are linearly independent, and which are not spanned by $\bar{a}\bar{b}$. It follows that

$$(a_1, \ldots, a_i) \quad \overset{\pi}{\mapsto} \quad (a'_1, \ldots, a'_i)$$

holds for some atom automorphism π that fixes $\bar{b}$. Because $\bar{b}$ supports $f(\bar{b})$, which is the same as $f(\bar{b})$, we have

$$f(\bar{a}) = f(\pi(\bar{a})).$$

Since we have assumed that the last $d - i$ coordinates of $\bar{a}$ are supported by $\bar{b}$, it follows that the last $d - i$ coordinates in $\pi(\bar{a})$ are the same as in $\bar{a}$. Summing up, we have found two inputs for the function f, namely $\bar{a}$ and $\pi(\bar{a})$, which agree on the last $d - i$ coordinates, but which have independent atoms on the first i coordinates. By equivariance of f, this means that the first i coordinates in a tuple from $\mathbb{A}^{\langle d \rangle}$ can be replaced by fresh independent atoms without affecting the value of f. It follows that f does not depend on the first i coordinates, and hence we can lower the dimension d. □

Take the function f from the above lemma. This function is surjective, since an input orbit is mapped to an output orbit, and X is assumed to be a one-orbit set. We will show that if $\bar{a}$ is the least support, in the sense of the theorem, for $f(\bar{a})$. Indeed, suppose that $f(\bar{a})$ would be supported by some tuple $\bar{b}$ which does not span $\bar{a}$. Then there would be an atom automorphism π that would fix $\bar{b}$ – and therefore also the output of the function – but would map $\bar{a}$ to some tuple not spanned by $\bar{a}$. In this case, the inputs $\bar{a}$ and $\pi(\bar{a})$ would be a violation of the above lemma. □

Exercises

Exercise 124. Let $\mathbb{B}$ be the structure where the universe is the same as in the bit vector atoms, but we only have the independence predicate for dimension 3, i.e. there is a ternary predicate "the atoms a, b, c are linearly independent". Show that $\mathbb{B}$ has the same automorphisms as the bit vector atoms.

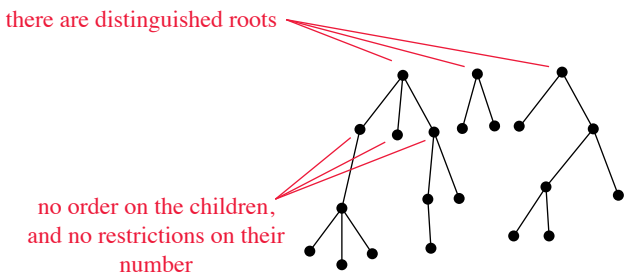
Exercise 125. Show that the structure $\mathbb{B}$ from Exercise 124 is not homogeneous.

Exercise 126. Consider vector spaces over the three element field, with the independence relations. Is the class of finite-dimensional vector spaces a Fraïssé class?

6.3.3 Trees and forests

In this section, we study the Fraïssé limit of trees and forests[5]. The trees and forests we study are rooted, unlabelled, and unordered, as explained in the following picture:

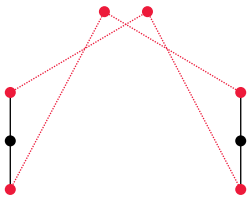
[5] This section is based on Bojańczyk et al. (2013b).



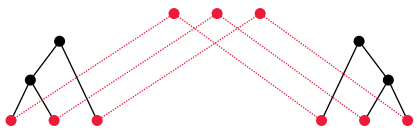
A tree is the special case of a forest when there is exactly one root.

The purpose of this section is to show that care is needed when choosing predicates and functions to model a combinatorial object, like a tree or forest, if we want to have a Fraïssé limit. The following list shows three ways of modelling trees as logical structures; only the third way will admit a Fraïssé limit. In all cases, the universe of the structure is the nodes of the tree.

1. There is a binary predicate for the parent relation. A finite forest is characterised by the requirement that each node has at most one parent. This way of modelling forests leads to a class that is not closed under amalgamation. Here is an instance of amalgamation that has no solution:



2. There is a binary predicate for the ancestor relation. A finite forest is characterised by the requirement that for every node, its ancestors are totally ordered. This way of modelling forests also leads to a class that is not closed under amalgamation. Here is an instance of amalgamation that has no solution:



3. We have a ternary relation

$$z = \text{closest common ancestor of } x \text{ and } y.$$

The class of trees modelled this way is closed under amalgamation, as illustrated in Figure 6.1. Therefore, it has a Fraïssé limit, which we call the *universal forest*.

(This forest is connected, because by amalgamation we can connect any two forests.)

Exercises

Exercise 127. Assume the universal forest atoms. Find a finitely supported equivalence relation on the atoms which has infinitely many infinite equivalence classes.

Exercise 128. Assume the universal forest atoms. Show that one cannot find an infinite equivariant set X and an equivariant relation on it which is a total dense order. Equivariance is important here, since if we only want a finitely supported one then this is easily accomplished by taking the path connecting some two atoms $a < b$, and using the order inherited from the atoms.

Exercise 129. Show that the universal forest has decidable MSO theory.

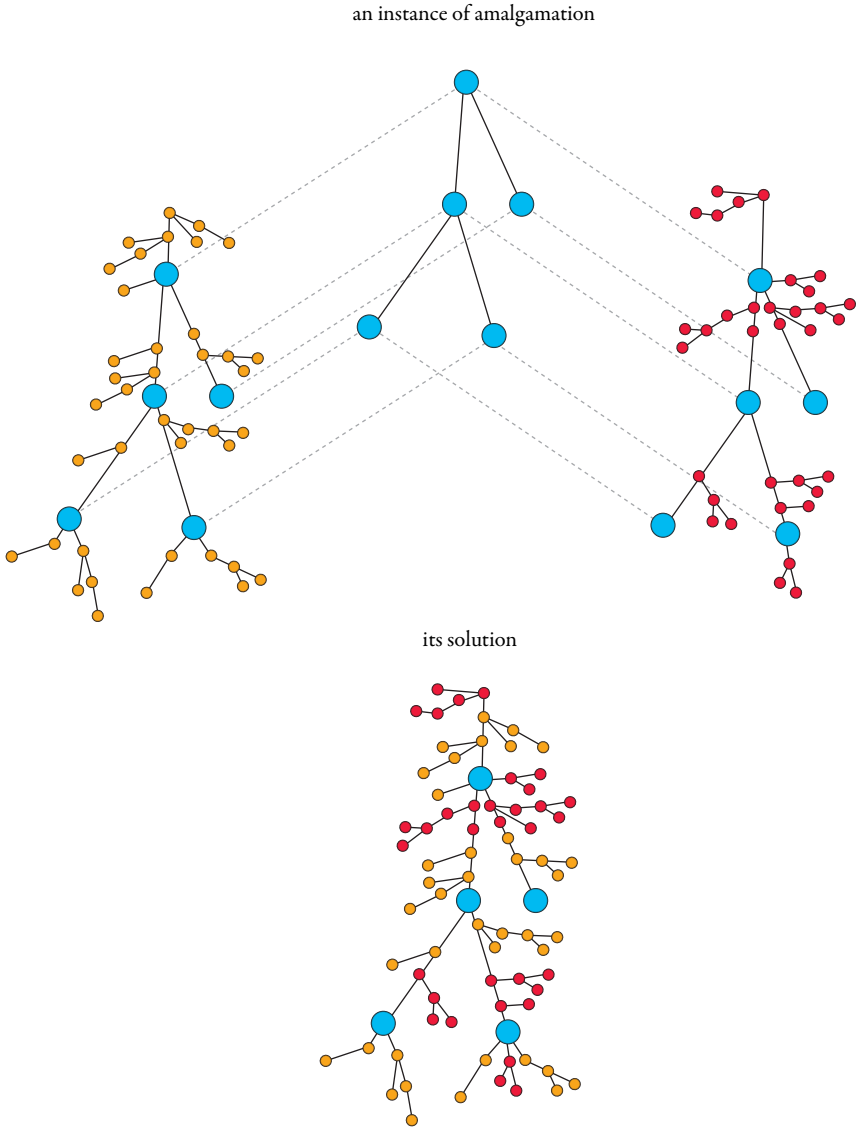


Figure 6.1: Amalgamation for forests with a relation for closest common ancestor.

Chapter 7

Algorithms on orbit-finite sets

In this chapter, we give examples that illustrate how algorithms can be generalized from finite sets to orbit-finite sets. For all algorithms in this chapter, we make the following assumptions about the atom structure.

Definition 7.1. A structure $\mathbb{A}$ is called *effectively oligomorphic* if:

1. it is oligomorphic and countable;
2. it has a decidable first-order theory;
3. given $d \in \{0, 1, \ldots\}$, the number of orbits in $\mathbb{A}^d$ can be computed.

These assumptions are satisfied by all atom structures that have been discussed so far, such as the equality atoms, the order atoms, the graph atoms, or the bit vector atoms.

7.1 Representing orbit-finite sets

To discuss algorithms, we need a finite representation of orbit-finite sets and their equivariant subsets. In the special case of polynomial orbit-finite sets, we already discussed such a representation in Section 5.2, when deciding graph reachability. This representation was defined for polynomial orbit-finite sets, but it extends naturally to (not necessarily polynomial) orbit-finite sets, as described below.

- *How do we represent an orbit-finite set?* By Theorem 5.12, every orbit-finite set admits an equivariant bijection with a spof set, and so we will use spof sets as our representation of orbit-finite sets. A spof set is of the form

$$\mathbb{A}^{d_1}_{/\sim_1} + \cdots + \mathbb{A}^{d_k}_{/\sim_k},$$

  where each $\sim_i$ is an equivariant partial equivalence relation on $\mathbb{A}^{d_i}$. By Theorem 5.7, each of the equivalence relations $\sim_i$ is necessarily first-order definable, and therefore it can be represented by a first-order formula. This formula has

$2d_i$ variables, because it is a binary relation on $\mathbb{A}^{d_i}$. Summing up, an orbit-finite set is represented by a list of dimensions $d_1, \ldots, d_k \in \{0, 1, \ldots\}$, one for each component, together with a list of first-order formulas $\varphi_1, \ldots, \varphi_k$ that describe partial equivalence relations on these components. We would like the syntax to be decidable, which in this case means checking if the first-order formulas do indeed define partial equivalence relations. This can be formalized by writing a first-order sentence:

$$\bigwedge_{i \in \{1,\ldots,k\}} \forall \bar{x}, \bar{y}, \bar{z} \in \mathbb{A}^{d_i} \quad \underbrace{\varphi_i(\bar{x}, \bar{y}) \Leftrightarrow \varphi_i(\bar{y}, \bar{x})}_{\text{symmetry}} \quad \wedge \quad \underbrace{\varphi_i(\bar{x}, \bar{y}) \wedge \varphi_i(\bar{y}, \bar{z}) \Rightarrow \varphi_i(\bar{x}, \bar{z})}_{\text{transitivity}}.$$

Since the first-order theory is decidable for an effectively oligomorphic structure, we can check if the above formula is true, and so the syntax is decidable.

- *How do we represent an equivariant subset of an orbit-finite set?* Apart from orbit-finite sets, we also need to represent their equivariant subsets (which themselves can be seen as new orbit-finite sets.) Suppose that we have an orbit-finite set as in the previous item. To describe an equivariant subset, we need to specify for each component $\mathbb{A}^{d_i}$ an equivariant subset, which is required to be stable under the equivalence relation $\sim_i$. This subset can be described by a first-order formula ψ_i, and stability can be formalized by a first-order sentence

$$\bigwedge_{i \in \{1,\ldots,k\}} \forall \bar{x} \in \mathbb{A}^{d_i} \quad \psi_i(\bar{x}) \quad \Rightarrow \quad \underbrace{\varphi_i(\bar{x}, \bar{x})}_{\substack{\text{each element} \\ \text{is in some} \\ \text{equivalence class}}} \quad \wedge \quad \underbrace{\forall \bar{y} \in \mathbb{A}^{d_i}\ \varphi_i(\bar{x}, \bar{y}) \Rightarrow \psi_i(\bar{y})}_{\substack{\text{subset is closed under replacing} \\ \text{elements with equivalent ones}}}.$$

In the rest of this chapter, we will present algorithms that operate on orbit-finite sets and their equivariant subsets, using the representation described above. For the moment, we need to care about the representation, and we will need to justify how various operations, such as Boolean operations or certain kinds of loops, can be implemented on this representation. Later on in this book, we will present a more principled approach, namely a programming language that takes care of all of these operations.

7.2 Representing elements of orbit-finite sets

In the previous section, we explained how we can represent orbit-finite sets and their subsets. It would also be desirable to represent individual atoms; for example this would be needed to use a representation of sets in terms of generators, as we did in Section 1.1 for the equality atoms. Before moving on to the algorithms, we discuss how individual elements can be represented. Of course such a representation should support certain basic operations, such as testing equality. This is formalized in the following definition, which uses the same three conditions as in item 3 of Theorem 6.12.

Definition 7.2 (Atom representation)**.** An *atom representation* of a structure $\mathbb{A}$ is a function $r : 2^* \to \mathbb{A}$ which has the following properties (when atoms are used in algorithms, they are represented as strings using the representation):

(a) every atom is represented by at least one string;

(b) given a first-order formula $\varphi(x_1, \ldots, x_d)$ and $a_1, \ldots, a_d \in \mathbb{A}$, one can decide if

$$\mathbb{A} \models \varphi(a_1, \ldots, a_d);$$

(c) given two tuples in $\mathbb{A}^d$, decide if they are in the same orbit.

In Exercise 130 we show that if an atom representation exists, then it is essentially unique, since there are computable translations between any two atom representations. The following theorem shows that atom representations exist, under our usual effectivity assumption.

Theorem 7.3. *Every effectively oligomorphic structure has an atom representation.*

Proof. We will use Theorem 6.12, whose conclusion says that there is an atom representation. An oligomorphic structure $\mathbb{A}$ can be seen as a homogeneous structure $\mathbb{H}$, which has the same elements, but its vocabulary is extended so that it has one relation for every first order formula. (The vocabulary is infinite.) Let $\mathscr{H}$ be the age of $\mathbb{H}$. By the assumption that $\mathbb{A}$ is effectively oligomorphic, there is an algorithm that inputs d and returns all structures in $\mathscr{H}$ up to isomorphism[1]. In other words, $\mathscr{A}$ satisfies assumption (*) of Theorem 6.12. Therefore, we can apply that theorem, which yields an atom representation of the Fraïssé limit $\mathbb{H}$, because item 3 of Theorem 6.12 is the same as the definition of an atom representation. This in turn yields a representation of $\mathbb{A}$. □

The atom representation which arises from the above theorem will be very inefficient. In cases of interest, such as the equality atoms or the order atoms, it will be better to manually prepare more efficient atom representations.

One application of atom representations will be discussed in Chapter 9, where we will study what it means for a language $L \subseteq \mathbb{A}^*$ to be decidable. In the presence of atom representations, we can simply require that this language is decidable in the usual sense, assuming that atoms are given as strings that represent them. This requirement will be a baseline to which we will compare other notions, such as orbit-finite Turing machines.

Another application is to represent equivariant subsets of $\mathbb{A}^d$ by finite generating sets. This is the same method as in Section 1.1: a *generating set* for a subset $X \subseteq \mathbb{A}^d$ is any set that contains at least one tuple per orbit. The assumptions on an atom representation will enable us to perform basic operations on subsets represented this way, such as Boolean combinations. For union, we can simply combine the two sets of generators (which might result in some orbits being represented by more than one generator). For intersection, we can use the "same orbit" test to check which orbits are represented in both sets. The most interesting operation is complementation, where we need to be able to describe orbits that are *not* represented. For this, we use the following lemma.

[1] It is worth explaining how one "returns" a structure over an infinite vocabulary. This means that we return a list of its elements, together with an algorithm which inputs a relation name, and tells us which tuples are selected by the relation.

Lemma 7.4. *For an atom representation of an effectively oligomorphic structure, there is an algorithm which does the following:*

(d) given d, compute a list of tuples that represents every orbit in $\mathbb{A}^d$.

Proof. By assumption on being effectively oligomorphic, we can compute the number of orbits. We can then start enumerating all of $\mathbb{A}^d$, until we have represented all orbits, which can be checked using the "same orbit" test from item (c). □

Of course, in cases of interest we will want to use more efficient algorithms than the one described in the proof of the above lemma. For example, the equality atoms the number of orbits in $\mathbb{A}^d$ is the corresponding Bell number, and a similar formula can also be computed for the ordered atoms.

Exercises

Exercise 130. Consider an effectively oligomorphic structure and two atom representations r_1, r_2. Show that there is some computable function $f : 2^* \to 2^*$ which inputs a representation of some atom under r_1, and returns a representation of the same atom under r_2.

Exercise 131. Consider the three conditions in Definition 7.2. Show that in the presence of the first two conditions (a) and (b), the third condition (c) is equivalent to any of the following conditions:

(c1) the following function, called the *Ryll-Nardzewski function*, is computable:

$$d \in \{0, 1, \ldots\} \qquad \mapsto \qquad \text{number of orbits in } \mathbb{A}^d;$$

(c2) there is an algorithm that inputs $d \in \{0, 1, \ldots\}$ and returns a list of tuples that generate $\mathbb{A}^d$, with one tuple for each orbit (i.e. no orbit is represented twice);

(c3) there is an algorithm that inputs $d \in \{0, 1, \ldots\}$ and returns a formula with $2d$ free variables that defines the "same orbit" relation on $\mathbb{A}^d$.

7.3 Orbit-finite graphs and automata

Having discussed representations of orbit-finite sets and their elements, we now start to present algorithms that operate on them. The first group of results, presented in this section, is about orbit-finite automata. These results are mainly a consequence of Theorem 5.10, which showed that reachability in pof graphs is decidable for effectively oligomorphic atoms (in fact, the proof did not use the full power of the assumption, since it did not require that we can compute the number of orbits in $\mathbb{A}^d$). Taking subquotients does not affect the algorithm, and so it extends to orbit-finite graphs, as stated in the following theorem.

Theorem 7.5. *Assume that the atoms are effectively oligomorphic. Then the reachability problem for orbit-finite graphs (represented as in Section 7.1) is decidable.*

Proof. Same as for Theorem 5.10. □

The emptiness problem for automata is the same as the reachability problem for graphs, and therefore we can use the above theorem to decide emptiness for orbit-finite automata. Let us begin by formally defining the model. Similarly to graphs, the definition of an orbit-finite automaton is the same as in the finite case, except that the word "finite" is replaced by "orbit-finite", and all subsets must be equivariant.

Definition 7.6 (Nondeterministic orbit-finite automaton)**.** Let $\mathbb{A}$ be an oligomorphic structure. A *nondeterministic orbit-finite automaton* over $\mathbb{A}$ is a tuple

$$\mathcal{A} = (\underbrace{Q,}_{\text{states}} \quad \underbrace{\Sigma,}_{\text{input alphabet}} \quad \underbrace{I \subseteq Q,}_{\text{initial states}} \quad \underbrace{F \subseteq Q,}_{\text{accepting states}} \quad \underbrace{\delta \subseteq Q \times \Sigma \times Q}_{\text{transitions}}),$$

where Q and Σ are orbit-finite sets, and the subsets I, F, δ are equivariant.

The semantics of the automaton are defined as for nondeterministic finite automata. The language recognised by such an automaton is equivariant, since the set of accepting runs is equivariant. An automaton is called *deterministic* if it has one initial state, and δ is a function from $Q \times \Sigma$ to Q.

Theorem 7.7. *Assume that the atoms are effectively oligomorphic. Then the emptiness problem for nondeterministic orbit-finite automata (represented as in Section 7.1) is decidable.*

Proof. An immediate corollary of Theorem 7.5. □

Other positive results that generalise easily to orbit-finite automata include:

- ϵ-transitions do not add to the power of nondeterministic automata (Exercise 132);
- one can minimize deterministic automata (Theorem 7.9);
- one can decide if a nondeterministic automaton is unambiguous (Exercise 133).

Negative results for the equality atoms generalise to other oligomorphic atoms, when the atoms are infinite:

- nondeterministic automata are not closed under complement;
- the universality problem is undecidable;
- deterministic automata are strictly weaker than nondeterministic ones.

To illustrate the scope of the above results, we give several examples of deterministic or nondeterministic orbit-finite automata in various oligomorphic atoms.

Example 58. Assume that the atoms are the random graph from Section 6.3.1. This structure is effectively oligomorphic, because it is the Fraïssé limit of a Fraïssé class that can be enumerated as in the assumptions of Theorem 6.12. The set of paths in the random graph can be viewed as a language

$$\{a_1 \cdots a_n \in \mathbb{A} : \text{for every } i < n \text{ there is an edge from } a_i \text{ to } a_{i+1}\} \subseteq \mathbb{A}^*.$$

This language is recognised by a deterministic orbit-finite automaton, which uses its state to store the last vertex that has been seen so far. The set of cycles is also recognised by a deterministic automaton, this automaton also needs to remember the first vertex to check if it is connected to the last one. □

Example 59. Assume that the atoms are the random graph, and consider the language

$$\{ w \in \mathbb{A}^* \mid \text{ the subgraph of } \mathbb{A} \text{ induced by atoms from } w \text{ is a clique } \}$$

This language cannot be recognised by an orbit-finite automaton, even nondeterministic, as we show in Exercise 134. □

Example 60. Assume that the atoms are the random graph. The graph atoms are a natural setting to talk about path and tree decompositions of graphs, as used in the graph minor project of Robertson and Seymour. To make notation lighter, we only discuss path decompositions. A *path decomposition* for a finite subset $V \subseteq \mathbb{A}$ is defined to be a list of (not necessarily disjoint) subsets $V_1, \ldots, V_n \subseteq V$ such that: (a) every vertex from V appears in at least one bag; and (b) if two vertices from V are connected by an edge, then they appear together in at least one bag; and (c) if a vertex appears in some two bags, then it also appears in all other bags between them. The *width* of such a path decomposition is the maximal size of its bags, minus one (the minus one is used to ensure that paths have path decompositions of width one).

If k is fixed, then such a path decomposition can be seen as a word over an orbit-finite alphabet, namely the sets of at most $k+1$ atoms. (The number of orbits in this alphabet is the number of isomorphism types of graphs with at most $k+1$ vertices.) Therefore, a path decomposition can be used as the input to an orbit-finite automaton. We now show that interesting properties of the underlying graph can be recognised by such automata.

Claim 7.8. *There is a deterministic orbit-finite automaton $\mathcal{A}$ such that*

$$\mathcal{A} \text{ accepts } V_1 \cdots V_n \quad \text{iff} \quad \text{the graph } V_1 \cup \cdots \cup V_n \text{ is connected}$$

holds for input which is a width k path decomposition[2].

Proof. After reading a path decomposition $V_1, \ldots, V_n$, the automaton stores in its state the last bag V_n, together with the equivalence relation $\sim_n$ on it. The equivalence relation identifies vertices from the last bag if they are in the same connected component of the underlying graph $V_1 \cup \cdots \cup V_n$. The states of the automaton are pairs (set of at most k atoms, an equivalence relation on this set); this state space is orbit-finite. The initial state is the empty set equipped with an empty equivalence relation, and the accepting states are those where the equivalence relation has one equivalence class. The definition of the transition function is left to the reader. □

[2]The automaton does not check if the input is a path decomposition. In fact, this cannot be done, see Exercise 121.

Constructions similar to the above claim can be done for any property of graphs of bounded pathwidth that is recognisable in the sense of Courcelle, which covers all graph properties that can be defined in monadic second-order logic[3]. Using tree automata instead of word automata, one can also cover tree decompositions. □

Example 61. Consider the bit-vector atoms and the language

$$\{\ w \in \mathbb{A}^* \mid \ w \text{ is linearly dependent } \ \}.$$

The linear dependence in the above language is the same as the one discussed when defining the bit-vector atoms, i.e. w is viewed as a list and not as a set. This means that any repetition in the list will immediately be a dependence. This language is recognised by a nondeterministic orbit-finite automaton. The state space is $\mathbb{A}$, the initial subset is the singleton of the zero vector $\{0\}$, and the accepting subset is the set of non-zero vectors. The transition relation is

$$\{\ p \xrightarrow{a} q \mid \ p + a = q \text{ or } p = q \ \}.$$

One can show that the nondeterminism in the above automaton is unavoidable – the language is not recognised by a deterministic orbit-finite automaton. In fact, we will show an even stronger result later in this book, namely that the language is not recognised by any deterministic Turing machine running in polynomial time. □

Minimization of deterministic automata. In Chapter 5, one of our motivations for introducing orbit-finite sets as a generalization of pof sets was to minimize deterministic automata. In Theorem 4.10, we showed a version of the Myhill-Nerode Theorem for the equality atoms, which gave a machine independent characterization of deterministic orbit-finite automata, in terms of an orbit-finite syntactic congruence. The same result carries over to general oligomorphic atoms.

Theorem 7.9. *Assume that the atoms are oligomorphic. The following conditions are equivalent for an equivariant language $L \subseteq \Sigma^*$ over an orbit-finite alphabet Σ:*

1. *L is recognised by a deterministic orbit-finite automaton;*
2. *the quotient of Σ^* under the syntactic congruence of L is orbit-finite.*

Proof. The same proof as for Theorem 4.10, and also the original Myhill-Nerode Theorem for finite sets. We simply construct a deterministic automaton on the equivalence classes of syntactic congruence. The assumption that the language is equivariant guarantees that the structure of the automaton – the transition function and the accepting states – is also equivariant. □

If the atoms are not only oligomorphic, but they are also effectively oligomorphic, then the syntactic automaton (i.e. the automaton that arises from the above theorem, also known as the minimal automaton) can be computed based on any other deterministic automaton.

[3]For more on recognisability, pathwidth, and monadic second-order logic, see (Courcelle and Engelfriet, 2012, Chapter 5.3).

Theorem 7.10. *If the atoms are effectively oligomorphic, then the syntactic automaton can be computed based on any deterministic orbit-finite automaton.*

Proof. We use what is called the *Moore algorithm*, i.e. a fixpoint procedure that computes equivalence on states[4]. Suppose that we are given a deterministic orbit-finite automaton, whose states are Q. We first use the graph reachability algorithm from Theorem 7.5 to restrict the state space to reachable ones. Next, we quotient the state space with respect to syntactic equivalence, i.e. recognizing the same language, as described below.

For $n \in \{0, 1, \ldots\}$, define $\sim_n$ to be the equivalence relation on states that identifies two states if they accept the same words of length at most n. It is easy to see that this equivalence relation is equivariant, and each $\sim_n$ can be computed using the formula representation of equivariant subsets. The chain

$$\sim_1 \quad \sim_2 \quad \sim_3 \quad \cdots$$

is a decreasing sequence of equivariant subsets of $Q \times Q$, and therefore it must stabilize after finitely many steps. The stable value of this sequence is the syntactic equivalence relation, and the minimal automaton is obtained by quotienting its state space under this relation. □

Exercises

Exercise 132. Show that adding ϵ-transitions does not change the expressive power of nondeterministic orbit-finite automata.

Exercise 133. Show that, under the assumptions of Theorem 7.5, one can check if a nondeterministic orbit-finite automaton is deterministic. Likewise for unambiguous (each input admits at most one accepting run).

Exercise 134. Consider the graph atoms. Show that the language of cliques, i.e. words in $\mathbb{A}^*$ where every two letters are connected by an edge, is not recognised by a nondeterministic orbit-finite automaton.

Exercise 135. Consider the equality atoms. For a language $L \subseteq \Sigma^*$, consider the two-sided Myhill-Nerode equivalence relation which identifies words $w, w' \in \Sigma^*$ if

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

The quotient of Σ^* under this equivalence relation is called the *syntactic monoid* of L. Show that if the syntactic monoid is orbit-finite, then the syntactic automaton is orbit-finite, but the converse implication fails.

Exercise 136. Let $L \subseteq \Sigma^*$ be a language, and let Q be the states of its syntactic automaton. Show that the syntactic monoid defined in the previous exercise is isomorphic to the submonoid of functions $Q \to Q$ which is generated by the state transition functions $\{q \mapsto qa\}_{a \in \Sigma}$ of the syntactic automaton.

Exercise 137. Let $L \subseteq \Sigma^*$, and let $h : \Sigma^* \to M$ be its syntactic homomorphism, i.e. the function which maps a word to its equivalence class under two-sided Myhill-Nerode equivalence. Show

[4] The computational complexity of automata minimisation is studied in Murawski et al. (2015), using the equality atoms and a more concrete model with registers and control states.

that M is orbit-finite if and only if the syntactic automaton of L is orbit-finite and there is some $k \in \{0, 1, \ldots\}$ such that all elements of M have support of size at most k.

Exercise 138. We say that a monoid M is aperiodic if for every $m \in M$ there is some $k \in \{0, 1, \ldots\}$ such that $m^k = m^{k+1}$. Let L be a language with an orbit-finite syntactic automaton. Show that the syntactic monoid of L is aperiodic if and only if for every state q of the syntactic automaton and every $w \in \Sigma^*$ there is some $k \in \{0, 1, \ldots\}$ such that $qw^k = qw^{k+1}$.

Exercise 139. Suppose that M is an orbit-finite monoid. Can one find an infinite sequence

$$M \supsetneq M_1 \supsetneq M_2 \supsetneq M_3 \supsetneq \cdots$$

such that each M_i is a submonoid?

Exercise 140. Consider an orbit-finite monoid M. We define the prefix relation on this monoid as follows: a is a prefix of b if $b = ax$ for some $x \in M$. Show that under the equality atoms, the prefix relation is well-founded, but this is no longer true under the order atoms.

7.4 Systems of equations

In the previous section, we discussed automata problems, which were based on graph reachability. Using a similar approach, the results on context-free grammars from Section 3.4 can be extended from the equality atoms to effectively oligomorphic atoms. Let us now give a new algorithm, which is based on a different approach[5]. In this algorithm, we use only two kinds of atoms, namely the equality atoms and the ordered atoms, but curiously enough, the ordered atoms are needed to analyse the equality atoms.

Consider a system of equations in the two element field $\mathbb{Z}_2$, like this one:

$$\begin{aligned} x + y &= 1 \\ x + z &= 1 \\ y + z &= 1 \end{aligned}$$

The system above does not have a solution, because some two variables need to get the same value, violating the equations. The system has finitely many equations. In this section, we consider systems where the set of equations is orbit-finite, but each individual equation is finite.

Example 62. Consider the equality atoms. The variables are pairs of distinct atoms, and the set of equations is

$$\underbrace{(a,b)}_{\text{one variable}} + \underbrace{(b,a)}_{\text{one variable}} = 1 \qquad \text{for all } a \neq b \in \mathbb{A}.$$

A solution in $\mathbb{Z}_2$ to this system amounts to a choice function, which chooses for every two atoms $a \neq b \in \mathbb{A}$ exactly one of the pairs (a, b) or (b, a). It follows that the above system has a solution, but no equivariant supported solution. □

[5] This section is based on Klin et al. (2015)

The above example shows that, under the equality atoms, an equivariant system of equations might have a solution, but it might not have an equivariant solution. If we use the ordered atoms, then the problem goes away, as shown in the following theorem.

Theorem 7.11. *Assume the atoms $(\mathbb{Q}, <)$. Let $\mathcal{E}$ be an equivariant orbit-finite set of equations. If $\mathcal{E}$ has any solution in $\mathbb{Z}_2$, then it has a solution in $\mathbb{Z}_2$ that is equivariant.*

Proof.

1. In the first step, we show that without loss of generality we can assume that the variables are tuples of atoms. Let X be the orbit-finite set of variables that appear in the equations $\mathcal{E}$. By the representation result from Theorem 5.12, see also Exercise 99, there is some $k \in \{0, 1, 2 \ldots\}$ and an equivariant surjective function

$$f : \mathbb{A}^k \to X.$$

   Define $\mathcal{F}$ to be the following set of equations over variables $\mathbb{A}^k$:

$$\underbrace{x = y}_{\text{when } f(x) = f(y)} \qquad \underbrace{y_1 + \cdots + y_n = i.}_{\substack{\text{when } \mathcal{E} \text{ contains an equation} \\ x_1 + \cdots + x_n = i \\ \text{where } f(y_1) = x_1, \ldots, f(y_n) = x_n}}$$

   It is easy to see that $\mathcal{E}$ has a solution if and only if $\mathcal{F}$ has a solution. Likewise for equivariant solutions.

2. Let $\mathcal{F}$ be the system of equations produced in the previous item. To prove the theorem, it remains to show that if $\mathcal{F}$ has a solution

$$s : \mathbb{A}^k \to \mathbb{Z}_2$$

   then it also has an equivariant one. We prove this using the Ramsey Theorem. By the Ramsey Theorem, there is an infinite set $A \subseteq \mathbb{A}$ such that

$$s(a_1, \ldots, a_n) = s(b_1, \ldots, b_n)$$

   holds for all $\bar{a}$ and $\bar{b}$ which are strictly growing tuples from A. Again by the Ramsey Theorem, there is an infinite set $B \subseteq A$ such that

$$s(a_1, \ldots, a_n) = s(b_1, \ldots, b_n)$$

   holds for all $\bar{a}$ and $\bar{b}$ which are strictly decreasing tuples from B. Repeating this argument for all finitely many order types, i.e. for all orbits in $\mathbb{A}^k$, we get an infinite set $Z \subseteq \mathbb{A}$ such that

$$s(a_1, \ldots, a_n) = s(b_1, \ldots, b_n)$$

   holds whenever $\bar{a}$ and $\bar{b}$ are tuples from Z^k with the same order type (in other words, in the same equivariant orbit of $\mathbb{A}^k$). Define

$$s' : \mathbb{A}^k \to \mathbb{Z}_2$$

to be the function that maps $\bar{a}$ to $s(\bar{b})$ where $\bar{b}$ is some tuple from Z^k in the same equivariant orbit as $\bar{a}$. Such a tuple $\bar{b}$ exists, and furthermore $s(\bar{b})$ does not depend on the choice of $\bar{b}$ by construction. Because $s'(\bar{a})$ depends only on the equivariant orbit of $\bar{a}$, the function s' is equivariant. It is also a solution to $\mathcal{F}$. This is because every equation from $\mathcal{F}$ can be mapped to some equation in $\mathcal{F}$ which uses only variables from Z, and s' satisfies those equations.

□

Corollary 7.12. *Assume that the atoms are* $(\mathbb{Q}, <)$*. Given an equivariant orbit-finite system of equations, one can decide if the system has a solution in* $\mathbb{Z}_2$*. Likewise for the equality atoms.*

Proof. Assume the atoms are $(\mathbb{Q}, <)$. By Theorem 7.11, it is enough to check if the system has an equivariant solution. We can compute all equivariant orbits of the variables, and therefore we can check all equivariant functions from the variables to $\mathbb{Z}_2$, to see if there is any solution.

Consider now the equality atoms. We reduce to $(\mathbb{Q}, <)$. Every equivariant orbit-finite set over the equality atoms can be viewed as an equivariant orbit-finite set over $(\mathbb{Q}, <)$, by using the same set builder expressions. This transformation does not affect the existence of solutions, and for systems of equations over atoms $(\mathbb{Q}, <)$ we already know how to answer the question. □

Exercises

Exercise 141. Assume that the atoms are Presburger arithmetic $(\mathbb{N}, +)$. Consider sets of equations over the field $\mathbb{Z}_2$, where both the variables and the set of equations are represented by set builder expressions. Show that having a solution is undecidable.

Exercise 142. What is the effect on the decidability of the problem in Exercise 141 if we assume that the set of variables is $\mathbb{A}$, i.e. the natural numbers? What if the variables are atoms and every equation has at most two variables?

Exercise 143. Consider the following atoms[6]. The universe is the set of bit strings $\{0, 1\}^\omega$ which have finitely many 1's. The structure on the atoms is given by the following relation of arity four:

$$a + b = c + d,$$

where addition is coordinate-wise. This structure is oligomorphic. Show two sets that are equivariant and orbit-finite, such that there is a finitely supported bijection between them, but there is no equivariant bijection.

[6]Suggested by Szymon Toruńczyk.

Chapter 8

Vector Spaces

In this section, we discuss an orbit-finite version of finite dimensional vector spaces. The idea is to consider a space that has two kinds of structure: the structure of a vector space, and the structure of a set with atoms. The canonical example is the space $\mathrm{Lin}\mathbb{A}$, which consists of finite linear combinations of atoms, such as

$$\mathrm{John} + 2\mathrm{Tom} - 3\mathrm{Eve}$$

in the case of the equality atoms. In fact, almost all results in this chapter will be restricted to the equality atoms, although some of them could be extended to other atoms, such as the ordered atoms or the graph atoms[1]. In the vector space $\mathrm{Lin}\mathbb{A}$, there are two kinds of structure: (a) there is the structure of a vector space, where we can add vectors and multiply them by scalars; and (b) there is the structure of atoms, where we can apply atom automorphisms to vectors, giving other vectors. The interplay between these two kinds of structure will be the main focus of this chapter. Typically, we will be interested in subsets that are compatible with both kinds of structure, i.e. closed under both linear combinations and atom automorphisms.

There might also be some confusion with a previous use of vector spaces in this book, namely the bit vector atoms from Section 6.3.2. The difference is that in Section 6.3.2, the atoms themselves had a vector space structure (in particular, in order to have oligomorphism, we needed to assume that the underlying field was finite). In this chapter, we start with some atoms $\mathbb{A}$, and then we discuss vector spaces in the context of these atoms, typically over an infinite field such as the rational numbers. Of course, the two constructions could be combined, i.e. we could take vector spaces over the bit vector atoms, but we will avoid this combination to avoid confusion.

8.1 Vector Spaces with Atoms

We begin with a formal definition of a vector space with atoms. The definition is given for an arbitrary choice of atoms and field. However, almost all results in this chapter

[1] The results of this chapter are based on Bojańczyk et al. (2024), where one can also find results for other kinds of atoms, such as the order atoms.

will assume that the atoms are the equality atoms, and that the field has characteristic zero. Characteristic zero means that one cannot obtain zero by adding an element of the field to itself finitely many times. The field of rational numbers has characteristic zero, but finite fields do not. In particular, we will exclude finite fields for most results in this chapter. (Although most of the results would work for arbitrary fields, but with different proofs.)

Definition 8.1 (Vector Space with Atoms)**.** Consider some atom structure $\mathbb{A}$ and some field $\mathbb{F}$. A *vector space with atoms* is a set V equipped with:

1. the structure of a vector space over the field $\mathbb{F}$, i.e. a binary function $v + w$ for addition and a family of unary functions λv for multiplying by a scalar $\lambda \in \mathbb{F}$, subject to the usual axioms; and
2. an action of automorphisms of the atoms $\mathbb{A}$;

subject to the following conditions:

(a) every vector has finite support in the sense of Definition 5.11

(b) for every scalar $\lambda \in \mathbb{F}$, scalar multiplication $v \mapsto \lambda v$ is equivariant;

(c) the addition operation $(v, w) \mapsto v + w$ is equivariant.

In condition (a), we use the notion of finite support from Definition 5.11, which means that for every $v \in V$ there is some finite tuple of atoms $\bar{a}$ such that $\pi(v) = v$ holds for every atom automorphism that fixes $\bar{a}$. This terminology might be slightly confusing when talking about vector spaces, where a different notion of "finitely supported" is sometimes used, namely a vector that has non-zero values in finitely many places. In this book, we never use the second notion, and we always mean the atom sense of Definition 5.11 when talking about finite supports. Also note that in condition (b), the corresponding axiom can be stated as

$$\pi(\lambda v) = \lambda \pi(v).$$

One could imagine that the field also uses atoms, and the axiom is

$$\pi(\lambda v) = \pi(\lambda)\pi(v).$$

We do not use the alternative axiom.

Example 63. Fix an atom structure $\mathbb{A}$ and some field $\mathbb{F}$. Consider the following three vector spaces, of which only the second and third will satisfy the axioms of Definition 8.1.

1. Consider first the space $\mathbb{A} \to \mathbb{F}$ of all functions from atoms to the field. This is a vector space, since functions can be added coordinate-wise, and likewise they can be multiplied by scalars. This space also satisfies axioms (b) and (c) from Definition 8.1. However, it does not satisfy axiom (a). For example, we can take any subset of atoms $X \subseteq \mathbb{A}$ that is not finitely supported, and use its characteristic function, which maps atoms from X to 1 and the remaining atoms to 0. This function will not be finitely supported.

2. A simple way to fix the problem for the previous space is to restrict it to those functions that are finitely supported. This will restore axiom (a), and the remaining axioms will also be preserved. Therefore, this is indeed an example of a vector space with atoms, which we denote by

$$\mathbb{A} \xrightarrow[\text{fs}]{} \mathbb{F}.$$

   A basis for this vector space can be obtained as follows. For each atom $a \in \mathbb{A}$, we take its characteristic function, which maps the atom to 1 and the remaining atoms to zero. On top of that, we add one more function, namely the constant function that maps all atoms to 1.

3. Finally, we can restrict the space from the previous item to functions that have finitely many possible nonzero values (this is the other understanding of the words "finitely supported", which is not used in this book). This is the space

$$\mathrm{Lin}\mathbb{A}$$

   that was mentioned at the beginning of this chapter. For this space, the set of atoms $\mathbb{A}$ is a basis.

□

As mentioned earlier in this chapter, a vector space with atoms has two kinds of structure, namely linear combinations and actions of atom automorphisms. When talking about subspaces, we will be interested in subspaces that preserve both kinds of structure.

Definition 8.2 (Equivariant subspace)**.** Let V be a vector space with atoms. An *equivariant subspace* of V is a subset $W \subseteq V$ that is closed under linear combinations and applying atom automorphisms.

Example 64. [Zero-sum space] Assume the equality atoms, and any field. An example of a subspace in $\mathrm{Lin}\mathbb{A}$ is what we call the *zero-sum space*, which consists of vectors

$$\lambda_1 a_1 + \cdots + \lambda_n a_n \qquad \text{such that } \lambda_1 + \cdots + \lambda_n = 0.$$

This set is clearly an equivariant subspace, since it is closed both under linear combinations and atom automorphisms. We call this the *zero-sum space*. We will prove that this space is spanned (i.e. generated using linear combinations only, and not atom automorphisms), by the set

$$\{\, a - b \mid \ a, b \in \mathbb{A} \ \}. \tag{8.1}$$

To prove that the above set spans the zero-sum subspace, we use induction on the number of atoms that appear in a vector. Indeed, suppose that

$$v = \lambda_1 a_1 + \cdots + \lambda_n a_n$$

is in the zero-sum subspace. If $n \leq 1$, then the vector must be equal to zero. Otherwise, if $n \geq 2$, then we can subtract the vector $\lambda_n a_n - \lambda_n a_{n-1}$, which is spanned by (8.1), to get new vector that is also in the zero-sum space. The new vector uses fewer atoms, and hence we can apply the induction assumption.

We will now show that the zero-sum subspace is the only non-trivial equivariant subspace. This means that apart from the zero-sum space, the only other equivariant subspaces are the zero space and the full space. Indeed, consider an equivariant subspace V that contains some non-zero vector

$$v = \lambda_1 a_1 + \cdots + \lambda_n a_n.$$

Choose some atom permutation π that maps a_n to some fresh atom b, and which fixes the remaining atoms $a_1, \ldots, a_{n-1}$. This can be done in the equality atoms. If we subtract $\pi(v)$ from v, then we get the vector

$$\lambda_n a_n - \lambda_n b,$$

which is in the zero-sum subspace. From this vector, we can use atom automorphisms and scalar multiplication to get all other vectors in the spanning set (8.1), and thus V must contain the zero-sum subspace.

So far, we have shown that every equivariant subspace, except the zero space, must contain the zero-sum subspace. Finally, we will show that there is no equivariant subspace that is strictly between the zero-sum subspace and the full space, which will prove that the zero-sum subspace is the only non-trivial equivariant subspace. Consider any vector

$$v = \lambda_1 a_1 + \cdots + \lambda_n a_n$$

that is not in the zero-sum subspace. Let v' be the vector that is obtained from v by changing the coefficient of a_n so that v' belongs to the zero-sum subspace. Since v is not in the zero-sum subspace, the coefficient must be changed, and therefore $v - v'$ is a vector that uses only the atom a_n, and which is nonzero. From such a vector, we can obtain the full space by using atom automorphisms and linear combinations. □

As the above example shows, requiring that a subset is closed under both linear combinations and atom automorphisms is a strong restriction. The zero-sum space from the above example also witnesses another phenomenon of vector spaces with atoms, which is that they do not necessarily have a basis that is equivariant. This is somewhat to be expected, since finding a basis involves choice, which is problematic in the presence of atoms.

Example 65. [No equivariant basis] We will show that the zero-sum space from Example 64 does not have any equivariant basis. For example, the spanning set (8.1) is not a basis, because it has linear dependencies, such as

$$(\text{John} - \text{Tom}) + (\text{Tom} - \text{John}) = 0.$$

Let us now prove that not only this subset, but also any other equivariant subset cannot be a basis. Toward a contradiction, consider an equivariant basis, and take

any vector v in this basis. Let $a_1, \ldots, a_n$ be all atoms that appear in this vector, and consider the sum

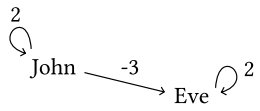
$$w = \sum_{\pi} \pi(v),$$

where π ranges over the finitely many permutations of $\{a_1, \ldots, a_n\}$. The resulting vector also uses the same atoms, however all coefficients are now the same. Since the vector is in the zero-sum space, it follows that all coefficients are zero. This means that we could not have been dealing with a basis, since some finite sum of basis vectors had a zero sum. □

Example 66. In Example 64, we showed that space Lin$\mathbb{A}$ had only finitely many equivariant subspaces, with the only non-trivial one being the zero-sum space. This is not true in general. Consider for example the space Lin($\mathbb{A}^2$). This space is finitely generated, namely by two vectors (John, John) and (John, Eve). An example vector in this space is

$$2(\text{John}, \text{John}) - 3(\text{John}, \text{Eve}) + 2(\text{Eve}, \text{Eve}).$$

Such a vector can be visualised as a finite directed graph with edges weighted by field elements, where the vertices are atoms, as in the following picture:



An equivariant subspace can be seen as a property of such graphs which is closed under isomorphism and linear combinations. Here are some examples of such properties:

1. sum of weight of self-loops is zero;
2. (sum of weight of self-loops) = 3·(sum of weights of non-self-loops);
3. for every vertex, the sum of the weights of incoming edges is zero.

In the second item, we have one equivariant subspace for each field element, and therefore we can have infinitely many such spaces if the field is infinite. □

8.2 The finite length property

One of the essential properties of orbit-finite sets (without vector spaces) was that if we take an increasing chain of equivariant subsets, then this chain must be finite. This property ensured termination for various algorithms, such as graph reachability, where such chains were computed. We will show that, under suitable assumptions, a similar property holds for chains of equivariant subspaces. The assumption is that

the space can be generated by finitely many vectors, by using both kinds of structure available, namely linear combinations and atom automorphisms. This can be seen as a vector space generalization of orbit-finiteness, where sets were finitely generated using only atom automorphisms.

Definition 8.3 (Finitely generated)**.** Let V be a vector space with atoms. For $X \subseteq V$, define the *equivariant subspace generated by* X, denoted by $\langle X \rangle$, to be the least equivariant subspace of V that contains X. A vector space with atoms is called *finitely generated* if it is equal to $\langle X \rangle$ for some finite set of vectors X.

It is not hard to see that the subspace $\langle X \rangle$ consists of vectors of the form

$$\lambda_1 \pi_1(x_1) + \cdots + \lambda_n \pi_n(x_n),$$

where $\lambda_1, \ldots, \lambda_n$ are coefficients from the field, $\pi_1, \ldots, \pi_n$ are atom automorphisms, and $x_1, \ldots, x_n \in X$. In other words, $\langle X \rangle$ can be obtained by first closing X under the action of atom automorphisms, and then closing it under linear combinations. Therefore, a space is finitely generated if and only if it is spanned (using linear combinations only) by some orbit-finite set. Observe that if we first close X under linear combinations, and then under automorphisms, then we get a smaller set than $\langle X \rangle$, which is not necessarily closed under linear combinations.

The main result of this chapter, Theorem 8.5 below, shows that for a finitely generated vector space with atoms, there is a finite upper bound on the length of strictly increasing chains of equivariant subspaces, as described in the following definition.

Definition 8.4 (Length)**.** Let V be a vector space with atoms. The *length* of V is defined to be the maximal n such that V has a strictly increasing chain of equivariant subspaces of the form

$$V_0 \subset V_1 \subset \ldots \subset V_n.$$

The length counts the number of times that the subspace can grow. Of course in a maximal chain, the chain will begin with the zero subspace and end with the full space. In principle, the length can be infinite, but we will show that it is finite for finitely generated vector spaces, under the assumption that the atoms are the equality atoms, and the field has characteristic zero.

Theorem 8.5. *Assume that the atoms are the equality atoms, and that the field has characteristic zero. A vector space with atoms is finitely generated if and only if it has finite length.*

We cannot remove both assumptions in the theorem, since under the bit vector atoms and the two-element field, the length can become infinite, see (Bojańczyk et al., 2024, Section 4.4). For all we know, in the case of characteristic zero, then all oligomorphic atoms have finite chains; we know that this is the case for atoms such as the ordered atoms, or the graph atoms.

Before proving the theorem, let us present an important corollary, which is that finitely generated vector spaces with atoms are closed under taking equivariant subspaces. This corollary is not obvious without the finite length property, since it is

not immediately clear how to find a finite generating set in an equivariant subspace, even in the presence of a finite generating set for the entire space. However, the finite length property is clearly closed under taking equivariant subspaces, and the theorem says that finite length is equivalent to finite generation.

Proof of Theorem 8.5. We begin with the proof with the easier implication, which is

$$\text{finite length} \quad \Longrightarrow \quad \text{finitely generated.}$$

Suppose that V has finite length. We use a greedy process to construct a finite generating set. Take some vector $v_1 \in V$. If this vector generates the entire space, then we are done. Otherwise, let v_2 be a vector that is not generated by v_1. If the space is generated by v_1 and v_2, then we are done. Otherwise, we continue. At each step we have a finitely generated equivariant subspace $\langle v_1, \ldots, v_n \rangle$. This chain of subspaces cannot grow indefinitely, and so we must reach the full space in finitely many steps. Observe that in this part of the proof, we only used the fact that there cannot be infinite chains, however we will prove a stronger result, namely that chains have uniformly bounded length.

We now proceed to the core of the theorem, which is the implication

$$\text{finitely generated} \quad \Longrightarrow \quad \text{finite length.}$$

We begin by reducing to spaces that have a basis, i.e. spaces of the form $\mathrm{Lin}X$, where X is an orbit-finite set. Such a space consists of finite linear combinations of elements from X. To see the reduction, consider some vector space with atoms V that is finitely generated. This means that the space is spanned (using only linear combinations and not atom automorphisms) by some orbit-finite set X. Let

$$\varphi : \mathrm{Lin}X \to V$$

be the equivariant linear map which maps a linear combination of vectors from X to their corresponding value in V. This map is surjective, by the assumption that X is a spanning set. It is not injective, i.e. it is not an isomorphism, because X might not be a basis, and therefore one vector from V might arise as an image of several linear combinations from $\mathrm{Lin}X$. By equivariance, for every chain of equivariant subspaces in V, the inverse images under φ form an equivariant chain of subspaces in $\mathrm{Lin}X$. By surjectivity, if a chain in V is strictly increasing, then the same is true for the chain of its inverse images. Therefore, the length of V is at most the length of $\mathrm{Lin}X$.

In the previous paragraph, we showed that the difficult implication in the theorem reduces to the special case when the vector spaces is of the form $\mathrm{Lin}X$, where X is an orbit-finite set. In the rest of the proof of this theorem, we will prove that

$$\text{length of } \mathrm{Lin}X \quad \leq \quad \text{number of orbits in } X^2. \tag{8.2}$$

The upper bound above is finite, since orbit-finite sets are closed under taking squares, and thus it gives the harder implication in the theorem. It remains to prove this bound.

The main idea will be to reduce the problem to a finite case, where only a finite (although large) set of atoms is used. The finite version will allow us to use a theorem from representation theory, which is called Maschke's Theorem[2].

Maschke's Theorem. Let us begin by introducing some terminology that will be used in Maschke's Theorem. In the definition of a vector space with atoms, we had a vector space with an action of the automorphisms of the atoms. Consider a more general setting, where there is an action of some abstract group G, see Definition 4.3, such that for every group element π, the corresponding function $v \mapsto \pi(v)$ is a linear map in the vector space. This linear map must be invertible, by the definition of a group action. We will call such a vector space a *vector space with an action of* G. Vector spaces with atoms are the special case when G is the permutations of the atoms. In the proof below, we will use the case when the group G is finite, and the vector space has finite dimension.

Equivariance and related notions are naturally extended for vector spaces with an action of a group G; for completeness we describe this now in some detail. For a vector space V with an action of a group G, a subspace is called *equivariant* if it is invariant under the action of G, i.e. if we take a vector in the subspace and apply a group element, then the resulting vector is still in the subspace. Similarly, we define equivariance for a linear map $\varphi : V \to W$ between two vector spaces with an action of the same group G: for every group element g, the following diagram commutes

$$\begin{array}{ccc} V & \xrightarrow{\text{action of } g} & V \\ {\scriptstyle\varphi}\downarrow & & \downarrow{\scriptstyle\varphi} \\ W & \xrightarrow[\text{action of } g]{} & W \end{array}$$

Finally, we can extend the notion of length to the case of vector spaces with an action of a group G: this is the maximal length of a strictly increasing chain of equivariant subspaces.

The case that we will care about is when we have a finite set of atoms $S \subseteq \mathbb{A}$, and the group is the set of all permutations of this set. For an orbit-finite set X, define X_S to be the elements of X that are supported by S, i.e.

$$X_S = \{\, x \in X \mid \ x \text{ is supported by a tuple from } S \ \}.$$

This is a finite set, since for every orbit-finite set, there are finitely many elements with a given support. (This is most easily seen using the representation result from Theorem 4.15.) The set X_S is equipped with an action of permutations of S, because applying such a permutation preserves the property of being supported by X. We will be interested in linear combinations of this set, i.e. the set $\mathrm{Lin}X_S$, which is a finite-dimensional vector space with an action of permutations of S. We can view this space as a finite dimensional approximation of the space $\mathrm{Lin}X$. The following lemma shows that the length of the latter set can be bounded by the lengths of its finite approximations.

[2]The proof here is different from the one in Bojańczyk et al. (2024), and the use of Maschke's Theorem is inspired by work of Jingjie Yang.

Lemma 8.6. *For every orbit-finite set X,*

$$\textit{length of } \mathrm{Lin}X \quad \leq \quad \sup_S \ \textit{length of } \mathrm{Lin}X_S,$$

where the supremum ranges over finite sets of atoms.

Proof. Consider some chain of equivariant subspaces in the infinite case:

$$V_0 \subset V_1 \subset \ldots \subset V_n = \mathrm{Lin}X,$$

where the group is all permutations of the atoms. For each $i \in \{1, \ldots, n\}$, choose some vector that is in V_i but not in V_{i-1}. Let S be the finite set of atoms that appear in these chosen vectors, and define

$$W_i = V_i \cap \mathrm{Lin}X_S.$$

This is a vector space with an action of permutations of S. The restricted spaces W_i are closed under applying permutations of S, since V_i was invariant under applying such permutations (and other permutations as well). Also, after intersecting with $\mathrm{Lin}X_S$, the chain continues to be strictly growing, since it contains vectors that witness the growth of the original chain. Hence, the new chain witnesses that the length of $\mathrm{Lin}X_S$ is at least n. □

Thanks to the above lemma, it remains to show that the length of $\mathrm{Lin}X_S$ is bounded by the number of orbits in X^2. What we have gained is that the vector spaces have finite (albeit unbounded) dimension, and the group actions use finite (albeit unbounded) groups. This will allow use to leverage a result from representation theory, called Maschke's Theorem, which decomposes spaces into irreducible parts. (In the theorem below, $\oplus$ stands for the direct sum of vector spaces, i.e. $V \oplus W$ is the set of pairs (v, w) where $v \in V$ and $w \in W$, with the operations defined coordinate-wise. In terms of bases, the basis of a direct sum is the disjoint union of the bases of the two vector spaces.)

Theorem 8.7 (Maschke's Theorem)**.** *Suppose that V is a finite dimensional vector space with an action of a finite group G, and the field has characteristic zero. Then V can be decomposed as*

$$V = V_1 \oplus \cdots \oplus V_n,$$

where each V_i is an equivariant space that is irreducible, i.e. the only equivariant subspaces of V_i are the zero space and the full space V_i.

Proof. Induction on the dimension of V, which we can do because the dimension is assumed to be finite. The main observation in the proof is the following claim, which will allow us to reduce the dimension in the induction step.

Claim 8.8. *Suppose that U is an equivariant subspace of V. There is a linear map from V to U which: (a) is the identity on vectors from U; and (b) has an equivariant kernel.*

Proof. We begin with a linear map φ_0 that satisfies condition (a) but not necessarily condition (b). Such a map can be found as follows. Take a basis of U, and extend it to a basis of the entire space V. Define φ_0 so that it sends the basis vectors from U to themselves, and the remaining basis vectors to zero. This linear map need not satisfy (b), since we have not taken any care when extending the basis to the entire space.

We will now improve the map φ_0 from the previous paragraph so that it additionally satisfies condition (b), i.e. it has an equivariant kernel. We do this using a form of averaging over group elements, which can be done since the group is assumed to be finite. Define φ as follows:

$$\varphi(v) = \frac{1}{|G|} \sum_{\pi \in G} (\pi^{-1} \circ \varphi_0 \circ \pi)(v).$$

The above map is well-defined thanks to the assumption that the field has characteristic zero, so that it is meaningful to divide by $|G|$. (For example, if the field would have characteristic two, then dividing by an even number would not be meaningful, since it would correspond to division by zero.)

We now show that the new φ satisfies the two conditions in the claim:

(a) We first show that φ is the identity on U. Take some $u \in U$. The following computation shows that each summand in the definition of $\varphi(u)$ is equal to u:

$$(\pi^{-1} \circ \varphi_0 \circ \pi)(u) \underbrace{=}_{\substack{\text{because } \pi(u) \in U \text{ by equivariance and} \\ \varphi_0 \text{ is the identity on } U}} (\pi^{-1} \circ \pi)(u) = u.$$

Therefore, $\varphi(u)$ is an average of several copies of u, which is just u itself.

(b) Let us now show that the kernel of φ is equivariant. We have:

$$\varphi(\sigma(v)) \quad = \quad \frac{1}{|G|} \sigma\Big(\underbrace{\sum_{\pi \in G} (\sigma^{-1} \circ \pi^{-1} \circ \varphi_0 \circ \pi \circ \sigma)(v)}_{\substack{\text{equal to } \varphi(v), \text{ because ranging over } \pi \text{ is} \\ \text{the same as ranging over } \pi \circ \sigma}} \Big).$$

Therefore, if $\varphi(v)$ is zero, then the above is also zero, which establishes that the kernel of φ is equivariant.

This completes the proof of the claim. □

Let us now complete the proof of Maschke's Theorem. The proof is by induction on the dimension of V. If the dimension is one, then V is irreducible, and we are done. Otherwise, take some nontrivial equivariant subspace U of V, and apply the above claim, yielding a linear map $\varphi : V \to U$. Using a standard linear algebra argument, we see that the space V is the direct sum of the image and kernel of φ. The image is irreducible, and the kernel has strictly smaller dimension, so we can apply the induction assumption to it. This completes the proof of Maschke's Theorem. □

We will use Maschke's Theorem to bound the length of a vector space V. For two vector spaces V and W equipped with an action of the same group G, let us write

$$V \underset{\text{lineq}}{\longrightarrow} W$$

for the set of all equivariant linear maps from V to W, with respect to the action of the group G. (The group is implicit in this notation.) Elements of this set are closed under taking linear combinations, and therefore this set can be seen as a vector space (but we do not equip this set with an action of the group, and therefore its only structure is that of a vector space). In particular, it is meaningful to talk about the dimension of this vector space.

Lemma 8.9. *Let V be a finite dimensional vector space, over a field of characteristic zero, which is equipped with an action of some finite group G. Then*

$$\textit{length of } V \quad \leq \quad \textit{dimension of the vector space } V \underset{\textit{lineq}}{\longrightarrow} V.$$

Proof. Apply Maschke's Theorem, yielding a decomposition

$$V = V_1 \oplus \cdots \oplus V_n. \tag{8.3}$$

For every $i \in \{1, \ldots, n\}$ we can define an equivariant linear map from V to itself which is the identity on V_i and maps vectors from the other components to zero. This gives us at least n equivariant linear maps from V to itself. None of these maps is spanned by the other, and hence the dimension is at least n. It remains to show that n is an upper bound on the length of V. This follows from the following claim, since irreducible spaces have length exactly one.

Claim 8.10. *The length of $V = U \oplus W$ is at most the sum of lengths of U and W.*

Proof. We first remark that the bound is in fact an equality, since the converse bound can be shown by taking a maximal chain in U, and following it with a maximal chain in W (with U added in each step). Let us now prove the bound in the claim, which is the one that we actually use. Consider some chain of equivariant subspaces

$$V_0 \subset V_1 \subset \ldots \subset V_m = V$$

in V. Define two equivariant chains in U and W, respectively, as follows:

$$\begin{aligned} U_i &= \{\ u \in U \mid\ (u, 0) \in V_i\ \} \\ W_i &= \{\ w \in W \mid\ (u, w) \in V_i \text{ for some } u \in U\ \}. \end{aligned}$$

The chains need not be strictly increasing, since some information is lost. However, we will show that in each step at least on of the chains must grow, which will give the upper bound in the statement of the claim. Toward a contradiction, suppose that there is some $i \in \{1, \ldots, n\}$ such that

$$U_{i-1} = U_i \quad \text{and} \quad W_{i-1} = W_i.$$

We will show that $V_{i-1} = V_i$, which will contradict the assumption that the chain is strictly increasing. Take some vector in $(u, w) \in V_i$. Then:

(u', w)	$\in V_{i-1}$	for some u' by the assumption that $W_{i-1} = W_i$
$(u - u', 0)$	$\in V_i$	by taking a difference of two vectors in V_i
$(u - u', 0)$	$\in V_{i-1}$	by the assumption that $U_{i-1} = U_i$
(u, w)	$\in V_{i-1}$	by taking a sum of two vectors in V_{i-1}

This shows that $V_{i-1} = V_i$, yielding the desired contradiction. □

Thanks to the above claim, the length of V is at most the number n of irreducible components in the Maschke decomposition (8.3). As we have argued before the claim, we can find at least n linearly independent equivariant linear maps from V to itself, which completes the proof of the lemma. □

Thanks to the above lemma, the length of the vector space $\mathrm{Lin}X_S$ is bounded by the dimension of the space of equivariant linear maps from this space to itself. The next lemma bounds this dimension by the orbit count of X^2, as required by (8.2), and thus completes the problem proof of the theorem.

Lemma 8.11. *For every finite set $S \subseteq \mathbb{A}$ we have*

$$\textit{dimension of the vector space } \mathrm{Lin}X_S \underset{\textit{lineq}}{\longrightarrow} \mathrm{Lin}X_S \quad \leq \quad \textit{number of orbits in } X^2.$$

Proof. To define a linear equivariant function with domain $\mathrm{Lin}X_S$, it is enough to define the function on the basis X_S. Therefore, the vector space in the statement of the lemma is the same as the vector space

$$X_S \underset{\text{eq}}{\longrightarrow} \mathrm{Lin}X_S$$

of equivariant functions (not linear maps) from X_S to $\mathrm{Lin}X_S$. By Currying, the above space is isomorphic to the space

$$X_S \times X_S \underset{\text{eq}}{\longrightarrow} \mathbb{F}.$$

To define an equivariant function, it is enough to choose one field element for each orbit. This is because two inputs in the same orbit will be mapped to the same field element, by equivariance of scalar multiplication. Therefore, the dimension of the space above is equal to the number of orbits of

$$(X \times X)_S = X_S \times X_S$$

under the action of the group of all permutations of S. To finish the proof of the lemma, we will show that this number of orbits is bounded from above by the number of orbits in $X \times X$, under the action of the group of all permutations of the atoms. This will follow from the following claim, applied to $Y = X \times X$. The claim is the only place in the proof where we use the assumption that the atoms are the equality atoms.

Claim 8.12. *Assume the equality atoms. Consider an orbit-finite set Y. For every finite set of atoms S, and every two elements of Y_S, the following conditions are equivalent:*

(i) being in the same orbit of Y_S under the action of permutations of S;

(ii) being in the same orbit of Y under the action of permutations of $\mathbb{A}$.

Proof. The implication (i) ⇒ (ii) is clear, and would work for any homogeneous atom structure, not just the equality atoms: every permutation of S can be extended to a permutation of all atoms. Let us now explain the implication (ii) ⇒ (i). Suppose that $y_1, y_2 \in Y_S$ are in the same orbit under the action of permutations of all atoms. We can improve the permutation that maps y_1 to y_2 so that it swaps the supports, and does not move any atoms outside the support (this would be impossible in some atoms, say the ordered atoms). This way, the improved permutation is a permutation of S. □

Using the above claim, in fact only implication (ii) ⇒ (i), we complete the proof of the lemma. The counter-positive of implication (ii) ⇒ (i) is that if two elements of Y_S are in different orbits, then they also represent different orbits in Y. Therefore, the number of orbits in Y_S is at most the number of orbits in Y. When applied to $Y = X \times X$ we get the result from the claim. □

This completes the proof of the bound (8.2), and hence the proof of Theorem 8.5. □

8.3 Function spaces

As we will show in this section, one of the remarkable things about vector spaces with atoms is that they are closed under taking function spaces. This is in contrast with the situation for orbit-finite sets, which are not closed under taking function spaces. Let us illustrate this on the example of the set of atoms itself.

Example 67. Orbit-finite sets (without vector spaces) are not closed under taking finitely supported powersets. For example, consider the finitely supported powerset of $\mathbb{A}$. We can think of this powerset as the set

$$\mathbb{A} \underset{\text{fs}}{\longrightarrow} (1 + 1)$$

of all finitely supported functions from $\mathbb{A}$ to the Booleans. As we have shown in Example 29, this set is not orbit-finite, since sets of different finite sizes are in different orbits. In other words, the above set is not finitely generated, assuming that the only kind of structure is applying atom permutations.

Consider now a larger set, namely the set

$$\mathbb{A} \underset{\text{fs}}{\longrightarrow} \mathbb{F}$$

of all finitely supported functions from $\mathbb{A}$ to a field $\mathbb{F}$, say the field of rational numbers. This set is larger, since every orbit-finite set can be viewed as a function, namely

its characteristic function, which maps elements of the set to 1 and the remaining elements to 0. What have we gained by considering a larger set? The answer is that the larger set is equipped with the structure of a vector space, i.e. we can take linear combinations of functions. Thanks to this extra structure, the space becomes finitely generated: as we have shown item 2 of Example 63, every function in the space is a linear combination of characteristic functions of singletons and the full set. □

As we have explained in the above example, adding linear combinations can make it possible to finitely generate sets that were previously not finitely generated. This is the idea that will be pursued in this section. In the example, when talking about function spaces, we used finitely supported and not equivariant functions. Let us begin by explaining why this is the right choice.

Why finitely supported functions? Let us explain why finitely supported functions, as opposed to equivariant ones, are the natural choice for function spaces. Let us first discuss this in the case of orbit-finite sets, i.e. when the only available structure is that of atom permutations. Later, we will extend the discussion to vector spaces with atoms. What is a function space? Suppose that X and Y are orbit-finite sets. There are two ideas for the function space, namely

$$\underbrace{X \underset{\text{eq}}{\longrightarrow} Y}_{\substack{\text{equivariant} \\ \text{functions from } X \text{ to } Y}} \quad \subseteq \quad \underbrace{X \underset{\text{fs}}{\longrightarrow} Y}_{\substack{\text{finitely supported} \\ \text{functions from } X \text{ to } Y}}.$$

The smaller set is finite, while the bigger one is not only infinite, but even orbit-infinite. We claim that the "right" notion of function space is the bigger one. Why? This is because we would like function spaces to allow Currying, in the following sense:

> Equivariant functions from $X \times Y$ to Z are in one-to-one correspondence with equivariant functions from X to the function space $Y \to Z$.

It is not hard to see that the smaller equivariant space does not allow Currying, while the bigger finitely supported one does. This is because when we transform an equivariant function from $X \times Y$ to Z into a function from X to some space, then this corresponds to fixing one argument in the function, which will turn an equivariant function into a finitely supported one. Therefore, even if ultimately care only about equivariant functions, our function spaces will need to store finitely supported functions, as long as we want to have Currying. The usefulness of Currying can be seen in the following example that uses automata.

Example 68. Consider a deterministic orbit-finite automaton, with an equivariant transition function

$$\delta : Q \times \Sigma \to Q.$$

Suppose that we want to turn this automaton into a monoid. This is done by considering for each input letter $w \in \Sigma^*$, the corresponding transition function $\delta_w : Q \to Q$.

(This construction does not preserve orbit-finiteness in general, but this is not important for the example.) Even though the original transition function δ was equivariant, the functions δ_w are no longer equivariant, since they have a fixed input word w, and the atoms used by that word will possibly influence the state transformation. Therefore, the monoid constructed this way will be a subset of

$$Q \underset{\text{fs}}{\longrightarrow} Q,$$

i.e. it will use finitely supported but not necessarily equivariant functions. We see here an example of Currying, where one argument of a function is fixed. □

A similar discussion applies to vector spaces with atoms. If we have two vector spaces with atoms V and W, then we can consider two vector spaces:

$$\underbrace{V \underset{\text{lineq}}{\longrightarrow} W}_{\substack{\text{equivariant} \\ \text{linear maps from } V \text{ to } W}} \quad \subseteq \quad \underbrace{V \underset{\text{linfs}}{\longrightarrow} W}_{\substack{\text{finitely supported} \\ \text{linear maps from } V \text{ to } W}}.$$

For the same reason as previously, only the bigger space will allow Currying. This is why we will focus on the bigger space. The main result of this section will be that even the bigger space is finitely generated.

Finitely supported dual. Before considering general function spaces, we begin with the case where the outputs are in the field, i.e. spaces of the form

$$V \underset{\text{linfs}}{\longrightarrow} \mathbb{F}.$$

This space is called the *finitely supported dual* of V. For vector spaces of finite dimension, i.e. spaces of the form $\mathbb{F}^d$ for some $d \in \{0, 1, \ldots\}$, applying the dual gives the same space (essentially it corresponds to replacing row vectors with column vectors). However, in the presence of atoms, this is no longer the case.

Example 69. Consider the finitely supported dual of $\mathrm{Lin}\mathbb{A}$, which is the space

$$\mathrm{Lin}\mathbb{A} \underset{\text{linfs}}{\longrightarrow} \mathbb{F}.$$

Defining a linear map $\mathrm{Lin}\mathbb{A}$ is the same as defining a function on the basis $\mathbb{A}$, and therefore this space is the same as

$$\mathbb{A} \underset{\text{fs}}{\longrightarrow} \mathbb{F}.$$

As we have seen in Example 63, this space is isomorphic to $\mathrm{Lin}(\mathbb{A} + 1)$, because we need one more generator for a constant function. □

Nevertheless, finitely generated vector spaces with atoms are closed under taking finitely supported duals, as we will show in the following theorem.

Theorem 8.13. *Assume the equality atoms and that the field has characteristic zero. If a vector space with atoms V is finitely generated, then the same is true for its finitely supported dual $V \underset{\textit{linfs}}{\longrightarrow} \mathbb{F}$.*

Proof. In the first step of the proof, we reduce the spaces of the form $\mathrm{Lin}\mathbb{A}^{(d)}$.

Lemma 8.14. *It is enough to prove the special case of the theorem where* $V = \mathrm{Lin}\mathbb{A}^{(d)}$.

Proof. First we reduce the theorem to spaces with a basis, i.e. spaces of the form $\mathrm{Lin}X$, where X is an orbit-finite set. By definition, every finitely generated vector space V admits an equivariant surjective linear map

$$\varphi : \mathrm{Lin}X \to V.$$

The finitely supported dual of $\mathrm{Lin}X$ is larger than the finitely supported dual of V, since each linear map in the latter space can be pulled back to a linear map in the former space, by precomposing with φ. Therefore, if the finitely supported dual of $\mathrm{Lin}X$ is finitely generated, then the same is true for V.

So far we have reduced to spaces of the form $\mathrm{Lin}X$, where X is an orbit-finite set. Let us now further reduce to the case where $X = \mathbb{A}^{(d)}$ for some d. Call an orbit-finite set X *good* if the finitely supported dual of $\mathrm{Lin}X$ is finitely generated. This is the same as saying that the space

$$X \underset{\text{fs}}{\longrightarrow} \mathbb{F}$$

is finitely generated. The main observation is that good sets are closed under disjoint unions and images under equivariant functions. For the disjoint unions, we observe that a function with domain $X_1 + X_2$ can be defined independently on each summand. For the images, we observe that if $f : X \to Y$ is a surjective equivariant function, then every function with domain Y can be pulled back to a function with domain X. If the pulled back functions can be finitely generated, then the same is true for the original functions. Using these two closure properties of good sets, we complete the proof of the lemma. Indeed, by Claim 9.5, every orbit-finite can be obtained from sets of the form $\mathbb{A}^{(d)}$ by using disjoint unions and images under surjective equivariant functions. Therefore, if the sets $\mathbb{A}^{(d)}$ are good, then so are all orbit-finite sets. □

It remains to prove the theorem for the special case $V = \mathrm{Lin}\mathbb{A}^{(d)}$. As in the proof of the above lemma, linear maps from $\mathrm{Lin}\mathbb{A}^{(d)}$ are the same as functions from the basis $\mathbb{A}^{(d)}$, and therefore we need to establish that the space

$$\mathbb{A}^{(d)} \underset{\text{fs}}{\longrightarrow} \mathbb{F}$$

is finitely generated. We will be mainly interested in the functions from the above space which are characteristic functions of certain subsets of $\mathbb{A}^{(d)}$. Among all subsets, we will be most interesting in the rectangular ones. Define a *rectangular subset* to be a subset of the form

$$\mathbb{A}^{(d)} \cap (X_1 \times \cdots \times X_d) \qquad \text{where each } X_i \text{ is a finitely supported subset of } \mathbb{A}.$$

The sets $X_1, \ldots, X_d$ are called the *sides* of the rectangular subset. We know that finitely supported subsets of the atoms are finite or co-finite, and thus the sides are finite or co-finite. An *atomic rectangular set* is the special case where each side is either a singleton, or the full set of all atoms.

Lemma 8.15. *Every finitely supported function from $\mathbb{A}^{(d)}$ to the field is a finite linear combination of characteristic functions of atomic rectangular sets.*

Proof. In the proof of the lemma, we abuse notation and identify sets with their characteristic functions. This means that we will talk about a linear combination of sets, when thinking of a linear combination of their characteristic functions. (We have already done this before.) Consider a finitely supported function f from $\mathbb{A}^{(d)}$ to the field, say supported by a tuple $\bar{a}$. Define a $\bar{a}$-orbit in $\mathbb{A}^{(d)}$ to be any set of the form

$$\{\, \pi(x) \mid \pi \text{ is an atom automorphism that fixes } \bar{a} \,\} \qquad \text{for some } x \in \mathbb{A}^{(d)}.$$

This kind of orbits will be discussed again later in the book, see Definition 10.8. An $\bar{a}$-orbit is easily seen to be a rectangular set, where each side is either an atom from the support $\bar{a}$, or the co-finite set of all atoms that are not in the support $\bar{a}$. In particular, there are finitely many possible $\bar{a}$-orbits. A function supported by $\bar{a}$ will give the same output for all inputs in the same $\bar{a}$-orbit, and therefore such a function is a finite linear combination of rectangular sets. Therefore, to prove the lemma, it is enough to show that every rectangular set is a finite linear combination of atomic rectangular sets.

For a rectangular set, define an *illegal side* to be a side that is neither a singleton nor the full set of atoms. An atomic rectangular set is one without illegal sides. We will use inductive procedure to eliminate illegal sides by using linear combinations. Consider a rectangular set with an illegal side, which is either a finite set $\{a_1, \ldots, a_n\}$ or a co-finite set $\mathbb{A} \setminus \{a_1, \ldots, a_n\}$. In the finite case, we can decompose the rectangular set into a disjoint union of n rectangular sets in which the previously illegal side becomes a singleton. Since the union is disjoint, it can be described by a linear combination. In the co-finite case, we can decompose the set as a set difference

$$X \setminus (X_1 \cup \cdots \cup X_n),$$

where X uses $\mathbb{A}$ on the previously illegal side, and X_j uses $\{a_j\}$. This decomposition can be described by a linear combination, since the sets $X_1, \ldots, X_n$ are disjoint and contained in X. All sets in the decomposition have fewer illegal sides, and therefore we can apply the induction hypothesis to each of them. □

The family of atomic rectangular sets is orbit-finite, since an atomic rectangular set is determined by the subset of coordinates that uses singletons (which can be chosen in finitely many ways), and the values of those singletons (which can also be chosen in orbit-finitely many ways). Therefore, thanks to the above lemma, the atomic rectangular sets form an orbit-finite spanning set, which completes the proof of the theorem. □

In the theorem above, we have shown that function spaces are finitely generated if the output space is the field. We now generalise this result to general output spaces that are any finitely generated.

Theorem 8.16. *Assume the equality atoms, and a field of characteristic zero. If V and W are finitely generated vector spaces with atoms, then the same is true for $V \underset{\text{linfs}}{\longrightarrow} W$.*

Proof. We begin with the special case when there is a basis for both the input and output spaces.

Lemma 8.17. *If X and Y are orbit-finite sets, then following space is finitely generated:*

$$\mathrm{Lin}X \underset{\textit{linfs}}{\longrightarrow} \mathrm{Lin}Y.$$

Proof. The function space in the statement of the lemma is the same as

$$X \underset{\text{fs}}{\longrightarrow} \mathrm{Lin}Y, \tag{8.4}$$

since defining a linear map on $\mathrm{Lin}X$ is the same as defining a function on the basis X. This space in turn is contained in the space

$$X \times Y \underset{\text{fs}}{\longrightarrow} \mathbb{F}, \tag{8.5}$$

because a function from X to linear combinations of Y can be seen as a function that inputs a pair (x, y) and returns the field coefficient of y in the output for x. The space (8.5) is finitely generated by Theorem 8.13, which completes the proof of the lemma. □

Before continuing, let us remark on the relationship between the two spaces (8.4) and (8.5) that were used in the proof of the above lemma. The second space can be seen as representing matrices. Each linear map has a corresponding matrix, which explains why the former space is contained in the latter. However, unlike for finite-dimensional spaces, there are more matrices than linear maps. This is because the matrices that arise from linear maps in (8.4) must satisfy the extra restriction that they have finitely many nonzero entries in each column. Therefore, the space in (8.5) is strictly bigger than the space in (8.4).

We now resume the proof of the theorem. Consider finitely generated vector spaces V and W, as in the statement of the theorem. By definition of finitely generated vector spaces, one can find orbit-finite sets X and Y and equivariant surjective linear maps

$$\varphi_V : \mathrm{Lin}X \to V \qquad \text{and} \qquad \varphi_W : \mathrm{Lin}Y \to W.$$

We assume that X is a pof set, which will be useful later in the proof (we could also do this for Y, but this will not be necessary in the proof). We can make this assumption because every orbit-finite set admits a surjective equivariant function from a pof set. As in the proof of Lemma 8.14, we can pull back linear maps from V to W along φ_V, which shows that

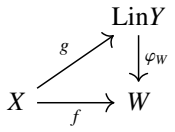
$$V \underset{\text{linfs}}{\longrightarrow} W \quad \text{is an equivariant subspace of} \quad \mathrm{Lin}X \underset{\text{linfs}}{\longrightarrow} W.$$

Therefore, it remains to show that the larger space on the right is finitely generated. To do this, we will show that

$$\underbrace{\mathrm{Lin}X \underset{\text{linfs}}{\longrightarrow} W}_{\substack{\text{same space as} \\ X \underset{\text{fs}}{\longrightarrow} W}} \quad \text{is the image of an equivariant linear map from} \quad \underbrace{\mathrm{Lin}X \underset{\text{linfs}}{\longrightarrow} \mathrm{Lin}Y}_{\substack{\text{same space as} \\ X \underset{\text{fs}}{\longrightarrow} \mathrm{Lin}Y}}.$$

Since the space on the right is finitely generated by the previous lemma, and finitely generated spaces are closed under taking surjective images, this will complete the proof of the theorem.

The linear map witnessing the image will simply post-compose with φ_W. This is clearly a linear equivariant map, but it is not immediately clear that it is surjective. Surjectivity means that every finitely supported function from X to W can be lifted to a finitely supported function from X to $\mathrm{Lin}Y$. In other words, we need to show that for every finitely supported function $f : X \to W$, there is a finitely supported function g which makes the following diagram commute:



(8.6)

The reader might be alarmed at this point, since such a lifting seems to involve choice, which is problematic in the presence of atoms. Indeed, the lifted function g must choose for each input $x \in X$ one of – possibly many – values in $\mathrm{Lin}Y$ that map to $f(x)$ via φ_W. However, in this particular situation we are saved by two things: (a) the set X is a pof set; and (b) the function g is not required to be equivariant, but only finitely supported. As we will see, under these conditions choice is possible. This is proved in the following lemma. The lemma is called the *uniformization lemma*, since the process of finding a function inside a relation is usually called uniformization.

Lemma 8.18 (Uniformization Lemma)**.** *Let $R \subseteq X \times Z$ be a finitely supported binary relation, where X is a pof set and Z is any set with an action of atom permutations. Assume that for every $x \in X$, there is at least one output, i.e. at least one $z \in Z$ such that $R(x, z)$. Then there is a finitely supported function $h : X \to Z$ that is contained in R.*

Once we have proved uniformization, we can apply it in the situation where Z is $\mathrm{Lin}Y$, and the relation R is the composition of f with the inverse of φ_W. This will establish the lifting in (8.6), and thus complete the proof of the theorem. It remains to prove the uniformization lemma.

Proof. We only prove the lemma in the case when the relation R is equivariant. The case when R is finitely supported is proved in the same way, except that we need to take care of the atoms $\bar{a}$ that are in the support of R when talking about orbits; this generalisation is left to the reader. Even under the assumption that the relation R is equivariant, the uniformized function h will still be finitely supported, and it might be impossible to make it equivariant. To see this, consider the example where the input set X is $\mathbb{A}$, the output set Z is $\mathbb{A}^{(2)}$, and the relation R is the full relation $R = X \times Z$. To uniformize this relation, we need to choose for every input atom an output pair that consists of two atoms. This cannot be done in an equivariant way, but it can be done in a finitely supported way, by always choosing the same output pair (John, Eve) for every input atom.

We will begin by reducing the lemma to the case when both X and Z are one-orbit sets. We first do this for the input set X, which is immediate, since the relation can be

uniformized separately for each orbit of the input. Next, we can restrict the relation R to a single orbit, which will only make the problem more difficult, but it will still preserve the assumption that for every input there is at least one output. In particular, the output set will also be restricted to a single orbit. After the restriction to a single orbit, the input set becomes

$$X = \mathbb{A}^{(d)}.$$

It is important that there is no quotient by a group of permutations of coordinates, which would be the case for a generic one-orbit set, see Theorem 4.15. This is because we assumed that the original input set was a pof set, and the orbits in pof sets do not use such quotients. For the output set Z, which does not have this assumption, we might have such a quotient, i.e. the output set will be of the form

$$Z = \mathbb{A}^{(e)}/G \qquad \text{for some subgroup of all permutations of } \{1, \ldots, e\}.$$

However, quotients are safe on the output side, in fact they can be eliminated. This is because we can first uniformize without the quotient on the output side, and then take the quotients. (This does not work on the input side.) Therefore, we assume that the output set is $\mathbb{A}^{(e)}$. Summing up, we have reduced the problem to uniformizing a one-orbit equivariant relation

$$R \subseteq \mathbb{A}^{(d)} \times \mathbb{A}^{(e)}.$$

Such a relation is described by a partial bijection between $\{1, \ldots, d\}$ and $\{1, \ldots, e\}$, that says which output coordinates must be copied from the input coordinates. The remaining output coordinates can be chosen freely, as long as they do not appear in the input. The goal of the uniformization is to choose some values for these remaining output coordinates. This can be done with a finite set of constants, which are fixed before knowing the input, and are used to fill in the missing output coordinates. We fix some finite set of atoms, which has size at least $e + d$, and atoms from this set can be used to fill in the output coordinates that are not copied from the input. The size of this fixed set is large enough so that for every input, we have enough constants that are not in the input to fill in all output coordinates. □

□

Chapter 9

Turing machines

Without atoms, the notion of "computability" is formalised using Turing machines, and there is a thesis – called the Church-Turing Thesis – which says that there is any other hypothetical model of computation would need to be captured by Turing machines.

What happens to this thesis in the presence of atoms? We begin a discussion in this chapter, by introducing orbit-finite Turing machines. This is an interesting and natural model of computation. However, as we will discover already in this chapter and then later on in this book, using Turing machines as the definition of computability is problematic. The main issue is that the basic data structure in a Turing machine is a list, and arranging objects in lists requires a form of choice, which typically impossible in the presence of atoms. We will overcome these limitations in later chapters, by using models of computation that are based on sets instead of lists.

Nevertheless, there are interesting things to say about Turing machines with atoms, and we say some of them in this chapter. The highlights are that, for Turing machines with atoms, one can prove nontrivial separations for complexity classes:

- In Section 9.2, we prove that $\textsc{p} \neq \textsc{np}$ holds in the bit vector atoms. It is worth underlying that this result has no bearing on the usual version of the question, without atoms. This is because we prove a much stronger result, which is clearly false without atoms, namely that deterministic orbit-finite Turing machines running in polynomial time are incapable of recognizing even some languages that are recognised by nondeterministic orbit-finite automata. The separating language is

  $$\{a_1 \cdots a_n \in \mathbb{A} : a_1, \ldots, a_n \text{ are not linearly independent}\}.$$

  This language is recognised in by a nondeterministic automaton (guess the linear combination that gives zero) or deterministic exponential time Turing machine (try all combinations), but it is not recognised in deterministic polynomial time.

- In Section 9.3, we prove a similar separation for the equality atoms. We show that there is a language which is recognised by a nondeterministic Turing ma-

chine (even in polynomial time), but not recognised by any deterministic Turing machine (even without restrictions on running time). This separation is harder than in Section 9.2, and uses the Cai-Fürer-Immerman construction from finite model theory. As in the previous item, this separation is unlikely to be useful in separating complexity classes without atoms. The accompanying proof is based on the limited access that Turing machines have to their input and on the symmetries that result from applying atom automorphisms.

9.1 Orbit-finite Turing machines

Before presenting the separation results about Turing machines, we begin by defining the model and discussing some examples.

Definition 9.1 (Orbit-finite Turing machine)**.** An *orbit-finite Turing machine* is defined in the same way as in Section 3.5, except that instead of polynomial orbit-finite sets, we use (not necessarily polynomial) orbit-finite sets, and the atoms need not be the equality atoms, but they can be any oligomorphic structure.

At first glance, the difference seems to be minor. By Theorems 4.6 and 5.12, an orbit-finite set is the same as a polynomial orbit-finite set quotiented by some partial equivalence relation, and therefore the only difference between pof sets and orbit-finite sets is the presence of subquotients. As we will see, the subquotients have profound impact, and break results such as the characterization from Theorem 3.9, even under the equality atoms.

To illustrate the roles of subquotients, consider the following example. The example is a positive one – i.e. we can still solve the problem using a deterministic orbit-finite Turing machine – but later on we will see negative examples.

Example 70. [An input alphabet with quotients] Consider the equality atoms. Let the input alphabet Σ be sets of atoms of size at most ten. This is the quotient of $\mathbb{A}^{10}$ under the partial equivalence relation that identifies two tuples if they use the same set of atoms. (Formally speaking, this quotient does not cover the empty set. To work around this minor issue, we can either use $\mathbb{A}^{10} + \mathbb{A}^0$, or we can stay with $\mathbb{A}^{10}$ and modify the equivalence relation, so that the empty set is represented by some redundant orbit. This is because sets of non-maximal size, say size 1, are represented by many orbits, and one of these can be spared to represent the empty set.)

Here is an example of an input word over this alphabet Σ that has six letters:

$$\{\text{John}, \text{Eve}, \text{Mary}\}\{\text{John}, \text{Eve}, \text{Tom}\}\{\text{John}, \text{Eve}\}\{\text{John}, \text{Eve}, \text{Mary}\}\{\text{John}, \text{Eve}\}\{\text{Tom}\}.$$

The letters are sets, which means that there is no order on the atoms that appear in a given letter. This will prevent a machine from doing operations like "load the first atom from the letter under the head", witnessing the difficulties of input alphabets with subquotients. However, as we show in this example, the difficulties can be overcome for this particular input alphabet.

Consider the language: "some atom appears in an odd number of letters". This language can easily be recognised by a nondeterministic Turing machine, in fact even

a nondeterministic orbit-finite automaton. The challenge is doing this with a deterministic machine, because there is no apparent mechanism for pointing to the atom that appears in an odd number of letters. To see this, consider the example input word from the previous paragraph. Both atoms John and Eve appear in an odd number of letters, but an equivariant deterministic Turing machine cannot see any difference between these two atoms, because every set contains either both or none of the atoms John and Eve.

Here is a solution to the problem, i.e. a deterministic orbit-finite Turing machine that recognises the language. Suppose that the input word is $A_1 \cdots A_n$, where each A_i is a letter from the alphabet, i.e. a set of at most ten atoms. The Turing machine executes the following program:

1. Generate a copy of the input word. After this step the tape has the form
$$\underbrace{A_1 \cdots A_n}_{\text{first copy}} | \underbrace{A_1 \cdots A_n}_{\text{second copy}}.$$
2. As long as possible, iterate these steps on the second copy of the tape:
   (a) if some set appears multiple times, remove all the duplicates;
   (b) if A, B are two distinct intersecting sets that appear in the second copy, then remove them and add the three sets $A \setminus B, B \setminus A, A \cap B$.

   This stage ends when all sets in the second copy are pairwise disjoint.
3. At this point, the second contains equivalence classes of the relation "appears in the same sets from $A_1, \ldots, A_n$". Check if some equivalence class (i.e. some set from the second copy) is contained in an even number of sets from the first copy.

All the above steps can be performed by a deterministic orbit-finite Turing machine. To process sets, the machine can use state space which stores a set of at most 10 atoms. □

In the above example, the symmetries in the input alphabet required some nontrivial programming tricks. As we will see later in this chapter, when the symmetries get more complicated, the tricks will run out.

Before showing these negative results, we begin with a positive result, which involves nondeterministic machines. In Theorem 3.9, we showed that for the equality atoms and polynomial orbit-finite sets, Turing machines with atoms are computationally complete, in the sense that they recognise the same languages as the usual atom-less Turing machines, assuming that atoms are represented in an atom-less way. The proof of that theorem does not extend to (not necessarily polynomial) orbit-finite sets. The issue is with the deatomisation construction in Lemma 3.10, which assumes that one can order the atoms from the input string in their order of appearance, and speak of the "leftmost atom", or the "second leftmost atom". Such an order exists when the alphabet is a pof set, because letters are ordered tuples of atoms, but it does

not necessarily exist for more general alphabets, e.g. when letters are sets as in Example 70. Therefore, we need to revisit the questions for the more general setting of orbit-finite sets.

Later in this chapter, we will show that not only the proof of Theorem 3.9 fails to work for orbit-finite sets that are not polynomial, but the result itself is false, because deterministic and nondeterministic orbit-finite machines have different computational power. However, we can at least show that the nondeterministic machines are computationally complete, under a mild assumption on the atoms, which is true for all atom structures discussed in this book.

The additional assumption is that the vocabulary is finite, and every first-order formula is equivalent to an existential formula, which is a formula of the shape

$$\varphi(x_1, \ldots, x_n) \quad = \quad \underbrace{\exists y_1, \ldots, y_m}_{\substack{\text{prefix of existential} \\ \text{quantifiers}}} \underbrace{\varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)}_{\text{a quantifier-free formula}}.$$

Most atom structures that we have used so far – such as the equality atoms, the ordered atoms, or the graph atoms – have an even stronger property, namely every formula is equivalent to a quantifier-free formula. This is because they are homogeneous structures over a finite vocabulary, and for homogeneous structures, every first-order formula is equivalent to a quantifier-free formula, see Theorem 6.3. The only exception is the bit-vector atoms, which are homogeneous, but over an infinite vocabulary. However, also these atoms can be made compliant, if we change the vocabulary, as explained in the following example.

Example 71. Consider the bit-vector atoms. As we have defined them in Section 6.3.2, this is a homogeneous structure over an infinite vocabulary, which has a dependence relation for every dimension d. We can, however, use a different vocabulary, namely one ternary relation

$$x + y = z$$

for addition of vectors. This relation is equivariant, since automorphisms of the atoms are linear maps, and therefore it can be defined using a quantifier-free formula by Theorem 6.3, see Exercise 145 for an explicit formula. Conversely, each of the dependence relations can be expressed using only the addition relation. Therefore, we can think of the bit-vector atoms as having only on relation, namely addition. (For this new vocabulary, the structure is no longer homogeneous, see Exercise 146.) In the new vocabulary, every first-order formula is equivalent to an existential one, since: (a) every first-order formula is equivalent to a quantifier-free formula using dependence only, and (b) both dependence and independence can be defined using existential formulas that use addition only. □

In Theorem 9.3 below, we will show that nondeterministic orbit-finite Turing machines are computationally complete, which means that they are equivalent to Turing machines that use atom-less strings as representations of atoms. This notion of representation was formalised in Definition 7.2, as a function $r : 2^* \to \mathbb{A}$, subject to certain assumptions. Once we know how to represent individual atoms as atom-less strings,

we can easily extend the representation to represent elements of orbit-finite sets, as explained in the following definition.

Definition 9.2. Let r be an atom representation. We extend this representation from atoms to elements of pof sets, as well as elements of subquotiented pof sets, as follows:

1. to represent an element of a pof set, we indicate the component and a list of representations of the atoms in the corresponding tuple;
2. to represent an element of subquotiented pof set, we indicate a representation of some element in the corresponding equivalence class;
3. to represent an element of an orbit-finite set, we fix an equivariant bijection with a subquotiented pof set, which must exist by Theorem 5.12, and then use the representations for the latter set.

We assume that all of these representations are words over the binary alphabet, by using some binary encoding of data structures such as lists.

The extensions are used in the following theorem, which shows that nondeterministic orbit-finite Turing machines are the "right" model for languages over orbit-finite alphabets.

Theorem 9.3. *Assume that the atoms are effectively oligomorphic, have a finite vocabulary, and every first-order formula is equivalent to an existential one. Then the following conditions are equivalent for every $L \subseteq \Sigma^*$ where Σ is an orbit-finite alphabet:*

1. *L is recognised by a nondeterministic orbit-finite Turing machine;*
2. *L is equivariant and for every atom representation, see Definition 7.2, the language*

$$\{ w \in 2^* \mid \ w \text{ represents some word in } L \ \}$$

   is recognised by a nondeterministic Turing machine (without atoms);

3. *as in the previous item, but the machine is deterministic;*
4. *as in the previous item, but the representation is quantified existentially.*

Proof. The implications $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4$ are proved in the same way as in Theorem 3.9. Let us very briefly recall those arguments. For the implication $1 \Rightarrow 2$, we extend the representation from atoms to configurations of Turing machines, and show that the one-step successor relation is decidable, which can be used to recognise the language in item 2. The implication $2 \Rightarrow 3$ follows from the fact that deterministic and nondeterministic Turing machines without atoms recognise the same languages. Finally, the implication $3 \Rightarrow 4$ is trivial ("every" implies "some", as long as an atom representation exists, and it does exist by Theorem 7.3).

We are left with the implication $4 \Rightarrow 1$. We begin by reducing to the case where the input alphabet is $\mathbb{A}$.

Lemma 9.4. *If the implication $4 \Rightarrow 1$ holds in the special case when the input alphabet Σ is $\mathbb{A}$, then it holds for every orbit-finite alphabet.*

Proof. The reduction is based on the following observation.

Claim 9.5. *For every orbit-finite set Σ, there exists $d \in \{0, 1, \ldots\}$ and a surjective equivariant partial function $f : \mathbb{A}^d \to \Sigma$.*

Proof. By Theorem 5.12, we know that Σ admits an equivariant bijection with a subquotiented pof set. By removing the subquotients, it follows that there is a surjective equivariant partial function from some pof set (without subquotients) to Σ. (The function is partial, because the subquotienting might have removed some tuples, and once we no longer take the subquotient, the function will not be defined on these tuples. In general, we cannot find a function that is total, see Exercise 144.) Therefore, it enough to prove the claim for the case when Σ is a pof set (without a subquotient). To get such a surjective function

$$f : \mathbb{A}^d \to \underbrace{\mathbb{A}^{d_1} + \cdots + \mathbb{A}^{d_k}}_{\text{a pof set}},$$

we choose d to be the maximal dimension among $d_1, \ldots, d_k$ plus some constant e, which is chosen to be so that $\mathbb{A}^e$ has at least k orbits. We use the orbit of the last e atoms to choose the component in $i \in \{1, \ldots, k\}$, and the remaining atoms to get the content of tuple in $\mathbb{A}^{d_i}$. □

We use the above claim to prove the lemma. Let $L \subseteq \Sigma^*$ be a language over some orbit-finite alphabet, which satisfies condition 4, i.e. for some atom representation r, the language

$$\{ w \in 2^* \mid \ w \text{ represents some word in } L \ \}$$

is recognized by a deterministic Turing machine without atoms. Apply Claim 9.5 to get a surjective function f, and extend it to lists, giving a surjective equivariant function

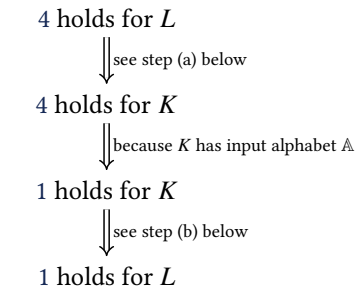
$$f^* : (\mathbb{A}^d)^* \to \Sigma^*.$$

We can pull back the language L along f^*, yielding a new language

$$K = \{ v \in (\mathbb{A}^d)^* \mid \ f^*(v) \in L \ \}.$$

We can think of the new language as being a language over alphabet $\mathbb{A}$, which contains only words with length divisible by k. Therefore, we can apply the implication $4 \Rightarrow 1$ to the pulled back language K, thanks to the assumption of the lemma. To complete the proof of the lemma, we will prove the implication $4 \Rightarrow 1$ for L, by first transferring the assumption from L to the pulled back language, then using the implication for the pulled back language, and then transferring the conclusion back to the original

language, as explained in the following diagram:



To complete the proof of the lemma, it remains to show the steps (a) and (b) from the above diagram.

(a) *Transfer of Condition 4.* Let us first explain how condition 4 transfers from the original language L to the pulled back language K. The main observation is that representations of letters for K can be transformed into representations of the corresponding letters for L, as shown in the following claim.

Claim 9.6. *There is a function g, which is computable in the usual sense without atoms, and makes the following diagram commute:*

$$\begin{array}{ccc} 2^* & \xrightarrow{g} & 2^* \\ {\scriptstyle \text{representation } r \text{ for } \Sigma}\downarrow & & \downarrow{\scriptstyle \text{representation } r \text{ for } \Sigma} \\ \mathbb{A}^d & \xrightarrow[f]{} & \Sigma \end{array}$$

Proof. As in Definition 9.2, when representing elements of Σ we think of it as being equal to a subquotiented pof set. By inspecting the definition of the function f from the proof of Claim 9.5, we see that it can be lifted to representations, as required in the proof of the current claim. The function f begins by checking the orbit of a suffix of the input tuple, and then based on this orbit it gives the output component and the corresponding output atoms, which are selected from the input atoms. □

Using the above claim, we show that the pulled back language K satisfies condition 4, i.e. representations of words in this language are recognised by a Turing machine without atoms. Given a representation of a word for K, a Turing machine can compute a representation of its image under f^*, thanks to the above claim, and then use the Turing machine from assumption the original language. This shows the transfer of condition 4.

(b) *Transfer of condition 1.* Let us now explain how condition 1 transfers from the new language K to the original language L. Condition 1 says that the language is recognised by a nondeterministic orbit-finite Turing machine. Given an input word for the original language L, the orbit-finite machine uses nondeterminism

to guess some input for the new language which maps to it along f^*, and then calls on the Turing machine for the new language K.

The proof of this lemma crucially uses nondeterminism, in the last step where an inverse image under f^* is guessed. As we will see later in this chapter, this is unavoidable, since nondeterministic orbit-finite Turing machines are strictly more expressive than deterministic ones. □

Having proved the reduction in Lemma 9.4, it remains to prove implication 4 ⇒ 1 for languages over the alphabet $\mathbb{A}$. Let then $L \subseteq \mathbb{A}^*$ be a language that satisfies condition 4, i.e. it is recognised by a Turing machine without atoms, under some representation. When we speak of representations of atoms or lists of atoms below, we refer to this representation.

We begin by showing that if we bound the computation time, then the accepted words can be defined using first-order formulas.

Lemma 9.7. *Consider a Turing machine M that inputs representations of words in $\mathbb{A}^*$. Given an input length d and bounds t and s on the time and space of the computation, one can compute a first-order formula which defines the language*

$$\{\, w \in \mathbb{A}^d \mid \begin{array}{l}\textit{some word in the orbit of } w \textit{ has a representation which}\\ \textit{is accepted by } M \textit{ in time at most } t \textit{ and space at most } s\end{array} \,\}.$$

Proof. Let F be the finite set of words (without atoms) that are accepted by M in at most n computation steps, and which represent some word in $\mathbb{A}^d$. This set can be computed given the parameters d, s, t. To prove the lemma, we show that we can compute formulas that describe the orbits of the words represented by F. The main observation is in the following claim, which shows that the partition of $\mathbb{A}^d$ into orbits, with each orbit described by a first-order formula, can be computed.

Claim 9.8. *Given d, one can compute first-order formulas that describe all orbits in $\mathbb{A}^d$.*

Proof. By the assumption that the structure is effectively oligomorphic, we can compute the number of orbits in $\mathbb{A}^d$, say it is n. Next, we can start enumerating n-tuples of first-order formulas in d variables, until we find an n-tuple where all formulas describe nonempty subsets which are pairwise disjoint. These subsets must be the orbits of $\mathbb{A}^d$. The stopping criterion is decidable, since the structure has a decidable first-order theory, and the procedure must stop, since every orbit is first-order definable by Theorem 5.7. □

Using the above claim, we can check which of the orbit formulas are satisfied by the tuples (represented by strings) in F. This check is decidable, by definition of atom representations. The disjunction of the resulting formulas defines the language in the statement of the lemma. □

Using the above lemma, we can complete the proof of implication 4 ⇒ 1, by showing that the language L is recognised by a nondeterministic orbit-finite Turing machine. Thanks to the above lemma, we can compute a first-order formula which tells us if the input word belongs to the language as witnessed by a run of given length.

The Turing machine recognising L will evaluate the formula, for ever larger resource bounds, and it will accept once it finds some resource bounds for which the formula is true. If the word is not in the language, then the procedure will not terminate, since no resource bounds be found. The important thing, however, is that the procedure will terminate with acceptance when the word is in the language. To evaluate the formulas, we use the following lemma.

Lemma 9.9. *There is a nondeterministic orbit-finite Turing machine that inputs*

$$\underbrace{a_1 \cdots a_d \in \mathbb{A}^*}_{\text{a list of atoms}} \qquad \text{and} \qquad \underbrace{\varphi(x_1, \ldots, x_d)}_{\text{a first-order formula}}$$

and answers if the formula is true for the given atoms.

Proof. The input alphabet is the disjoint union of $\mathbb{A}$, which is used for the list of atoms, and some finite alphabet which is used to represent the formula. The proof of this lemma is the only place where we use the assumption that every formula is equivalent to an existential one.

We begin with the special case of the lemma when the formula φ is quantifier-free. A quantifier-free formula is a Boolean combination of atomic formulas

$$R(x_{i_1}, \ldots, x_{i_k})$$

where R is a relation from the vocabulary, and the indices $i_1, \ldots, i_k$ indicate the variables in question. For each such atomic formula, the Turing machine moves it head to the positions $i_1, \ldots, i_k$ on the input list of atoms, and collects their values into a single tuple from $\mathbb{A}^k$. Since the vocabulary is finite, the dimension k has a fixed upper bound that does not depend on the input, and therefore this tuple can be stored in the state. The relations from the vocabulary can be hard-coded into the transition function of the Turing machine, since they are equivariant, and there are finitely many possibilities by the assumption that the vocabulary is finite. Therefore, the Turing machine can use a single transition to check if the relation R is satisfied by a tuple that is stored in its state.

We now consider the general case, where the input formula is not necessarily quantifier-free. By the assumption on the atom structure, we know that the input formula is equivalent to an existential formula

$$\exists y_1 \cdots \exists y_n \; \psi(x_1, \ldots, x_d, y_1, \ldots, y_n).$$

Not only does this existential formula exist, but we can also compute it. This is done by enumerating all candidates for the existential formula and using decidability of the first-order theory to check if the candidate is equivalent to the input formula. We know that an equivalent candidate will eventually be found. Once we have found the appropriate candidate, it can be evaluated for the input list of atoms: use nondeterminism to guess the values of the existential variables, write them on the tape, and then check if the quantifier-free part ψ is true using the procedure from the previous paragraph. □

By putting together the above lemma with Lemma 9.7, we complete the proof of implication 4 ⇒ 1, and also the proof of the theorem. □

In the above theorem, we use nondeterministic Turing machines and the assumption that first-order logic collapses to its existential fragment. This assumption covers all atom structures used in this book, and we are not aware of any example that is not covered. However, it is natural to ask about what happens without this assumption. We will show that without this assumption, a variant of the above theorem can still be recovered, at the cost of using a slightly more general model, namely alternating machines.

Let us begin by describing the model. The syntax of an alternating Turing machine is the same as for a nondeterministic one, except that the states are additionally partitioned into two parts: *existential* and *universal*. (The idea is that a nondeterministic machine is the special case when all states are existential.) The semantics of is defined in terms of a game, which is played by two players, called the *existential player* and the *universal player*. A position in the game is a configuration of the machine. In a given configuration, the player who owns the control state chooses a transition, which results in a new configuration, or in acceptance/rejection. The language recognised by the machine is defined to be the words for which the existential player has a strategy in the game that ensures acceptance, assuming that one begins with the initial configuration for the input string. This means that for every strategy of the universal player, a finite number of rounds is played and then the machine accepts.

Theorem 9.10. *Assume that the atoms are effectively oligomorphic and have a finite vocabulary. Then for every language $L \subseteq \Sigma^*$ over orbit-finite alphabet, conditions 2–4 from Theorem 9.3 are equivalent each other, and also to:*

1 L is recognised by an alternating orbit-finite Turing machine.*

Proof. The implications 1* ⇒ 2 ⇒ 3 ⇒ 4 are proved in the same way as in Theorem 3.9. The only difference is that for the first implication, we need the observation that without atoms, alternating and nondeterministic Turing machines are equivalent. This is a classical result, which is proved by showing that a nondeterministic Turing machine can enumerate through all possible strategies for the existential player.

For the implication 4 ⇒ 1*, we can use the same proof structure as in the proof of Theorem 9.3. As we remarked in that proof, Lemma 9.9 is the only part of the proof which used the assumption that every formula is equivalent to an existential one. Therefore, it remains to prove that lemma, without this assumption, but using alternating machines instead of nondeterministic ones in the conclusion. In this version, the lemma becomes essentially trivial, since the alternation can be used to simulate both kinds of quantifiers, universal and existential. □

Exercises

Exercise 144. Show that Claim 9.5 cannot be strengthened to have a total (not partial) function.

Exercise 145. Consider the bit-vector atoms. Write a quantifier-free formula for $x + y = z$, which uses the dependence relations.

Exercise 146. Consider the bit-vector atoms with the ternary addition relation from Example 71 and no other relations. Show that this structure is not homogeneous.

Exercise 147. Assume oligomorphic atoms. Let M be a nondeterministic orbit-finite Turing machine with input alphabet $\mathbb{A}$. Show that there is a finite family $\mathcal{R}$ of equivariant relations on the atoms such that for every $n, t, s \in \{0, 1, \ldots\}$, the property

$$\{\, w \in \mathbb{A}^n \mid \ M \text{ accepts } w \text{ in time at most } t \text{ and space at most } s \ \}$$

can be defined by an existential formula that uses only relations from $\mathcal{R}$.

Exercise 148. Assume that the atoms are effectively oligomorphic. Prove a converse of Theorem 9.3: if the conditions in the theorem are equivalent, then every first-order formula is equivalent to an existential one.

Exercise 149. Assume that the atoms are oligomorphic. Let Σ be an orbit-finite input alphabet. Show that a language $L \subseteq \Sigma^*$ is recognised by a deterministic orbit-finite Turing machine if and only if:

(*) There is an orbit-finite set $A \supseteq \Sigma$, a finite set $\mathcal{F}$ of functions (each one being an equivariant function $A^k \to A$ for some k) and an equivariant subset $F \subseteq A$ such that given $n \in \mathbb{N}$, one can compute a term using the functions $\mathcal{F}$ and has n variables such that

$$a_1 \cdots a_n \in L \quad \text{iff} \quad t(a_1, \ldots, a_n) \in F \qquad \text{for every } a_1, \ldots, a_n \in \Sigma.$$

Exercise 150. Assume that the atoms are oligomorphic and admit least supports. Show that a language $L \subseteq \mathbb{A}^*$ is recognised by a deterministic orbit-finite Turing machine if and only if:

(**) There exists a finite family $\mathcal{F}$ of functions (each one being an equivariant function $\mathbb{A}^k \to \mathbb{A}$ for some k) and relations (each one being a subset of $\mathbb{A}^k$ for some k) such that given $n \in \mathbb{N}$, one can compute a quantifier-free formula with functions $\mathcal{F}$ that has n free variables and defines $L \cap \mathbb{A}^n$.

Exercise 151. Assume that the atoms admit least supports, and are homogeneous over a relational vocabulary. Show that nondeterministic and deterministic orbit-finite Turing machines recognise the same languages over input alphabet $\mathbb{A}$.

9.2 For bit vector atoms, P ≠ NP

Recall the bit vector atoms that were introduced in Section 6.3.2. This is the vector space over the two-element field of countably infinite dimension, i.e. these are vectors in $\{0, 1\}^\omega$ that have finitely many nonzero entries. Equivalently, this is the Fraïssé limit of finite-dimensional vector spaces over this field. As explained in Example 71, we can view this as a structure with one ternary relation $x + y = z$, or alternatively with infinitely many relations for linear dependence.

In this section, we show that P ≠ NP holds for these atoms. Actually, we prove that deterministic polynomial time orbit-finite Turing machines are not even capable of simulating nondeterministic orbit-finite automata. The separating language consists of lists of vectors that have some nontrivial linear dependency.

Theorem 9.11. *Assume the bit vector atoms. The language*

$$\{\, a_1 \cdots a_n \in \mathbb{A}^* \mid \text{ for some nonempty subset } I \subseteq \{1, \ldots, n\} \text{ we have } 0 = \textstyle\sum_{i \in I} a_i \,\}$$

is recognised by a nondeterministic orbit-finite automaton (and therefore also by a nondeterministic polynomial time orbit-finite Turing machine), but it is not recognised by any deterministic polynomial time orbit-finite Turing machine.

Proof. The upper bound in the theorem – about recognisability by an orbit-finite automaton – was shown in Example 61. One can also have an alternative upper bound: the language can be recognised by a deterministic orbit-finite Turing machine in exponential time, by enumerating through all possible coefficients in the linear combination.

The rest of this section is devoted to proving the lower bound for deterministic Turing machines that run in polynomial time. Fix some deterministic orbit-finite Turing machine. We will show that if the machine runs in polynomial time and rejects some linearly independent tuple, then it will also reject some linearly dependent tuple.

We begin by introducing some notation. Let Γ be the work alphabet and let Q be the state space of the fixed Turing machine. A computation of the machine that uses t time steps and s units of space can be seen as a rectangular grid

$$\rho : \underbrace{\{1, \ldots, t\} \times \{1, \ldots, s\}}_{\substack{\text{pairs in this set} \\ \text{will be called tiles}}} \to \underbrace{\Gamma + \Gamma \times Q}_{\text{labels of tiles}},$$

where labels from $\Gamma \times Q$ are used for tiles containing the head, and labels from Γ are used for the other tiles. Not every function ρ of the above type is a computation, because ρ must also respect the transition function of the machine. The following straightforward lemma says that respecting the transition function is a property that depends on at most three tiles at a time.

Lemma 9.12. *Suppose that*

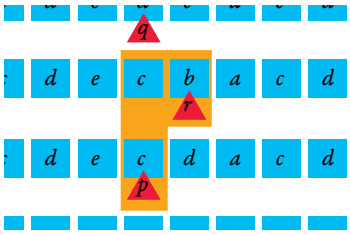
$$\rho, \sigma : \{1, \ldots, t\} \times \{1, \ldots, s\} \to \Gamma + \Gamma \times Q$$

are similar in the sense that for every three tiles x, y, z, the triples

$$(\rho(x), \rho(y), \rho(z)) \qquad (\sigma(x), \sigma(y), \sigma(z))$$

are in the same orbit. Then ρ is a computation if and only if σ is a computation, and ρ is rejecting if and only if σ is rejecting.

Proof. The semantics of a Turing machine involves comparing at most three tiles at the same time, as in the following picture:



The assumption of the lemma could even be weakened to triples of tiles that are adjacent in the grid as in the above picture, but we will not need this stronger variant of the lemma. □

We use the above lemma to show that a rejecting computation of the Turing machine that has polynomial size can be converted into another rejecting computation, whose input word is linearly dependent, and therefore should be accepted. The resulting contradiction will show that the language cannot be recognised by a deterministic polynomial time Turing machine.

By Claim 9.5, there is a surjective equivariant function

$$f : \mathbb{A}^d \to \Gamma + \Gamma \times Q.$$

Consider an input string $w \in \mathbb{A}^*$ that is linearly independent, and let

$$\rho : \{1, \ldots, t\} \times \{1, \ldots, s\} \to \Gamma + \Gamma \times Q$$

be the corresponding computation of the Turing machine. This is a rejecting computation, because the input string is linearly independent. Define a *support list* of this computation to be any

$$\bar{\rho} : \{1, \ldots, t\} \times \{1, \ldots, s\} \to \mathbb{A}^d$$

which is yields ρ after extending with f. The support list can be viewed as a list of atoms of length *std*. This length of this list is polynomial in the input length, since the time and space of the Turing machine is polynomial, and the dimension d is fixed. We claim that if the input length n is large enough, then there is some atom $a \in \mathbb{A}$ which

1. is spanned by the n independent atoms in the input string w;
2. is not spanned by any subset of $3d$ atoms in the support list $\bar{\rho}$.

This is because the number of atoms spanned by $\bar{a}$ is exponential in n, while the number of subsets from the second item is polynomial.

Choose a linear map $\varphi : \mathbb{A} \to \mathbb{A}$ whose kernel is $\{0, a\}$. (This is done by choosing a basis of $\mathbb{A}$ which contains a, and then sending the basis vector a to zero, and the remaining basis vectors to themselves). This linear map is not an atom automorphism, since it is not invertible. Nevertheless, we can still apply it to atoms and lists of atoms (however, it will not be guaranteed to preserve orbits). Apply it to the support list $\bar{\rho}$, yielding a new support list

$$\bar{\sigma} : \{1, \ldots, t\} \times \{1, \ldots, s\} \to \mathbb{A}^d.$$

Finally, let

$$\sigma : \{1, \ldots, t\} \times \{1, \ldots, s\} \to \Gamma + \Gamma \times Q$$

be the result of extending $\bar{\sigma}$ with f. We will now show that σ is in fact a computation of the Turing machine, and that it is also rejecting. This will yield a contradiction, since the input string becomes linearly dependent after applying φ, and therefore we get a rejecting computation on a string that should be accepted.

Lemma 9.13. *The function σ is a rejecting computation of the Turing machine.*

Proof. The essential idea is that while the linear map φ is not an atom automorphism, it still preserves the orbit of short lists of atoms, and this will be enough to preserve computations of the Turing machine.

Thanks to Lemma 9.12, it suffices to show that for every three tiles x, y, z, the triples orbits of these three tiles are the same in ρ and σ. To prove this, we use the following straightforward characterisation of the same orbit relation.

Claim 9.14. *Two tuples $\bar{a}, \bar{b} \in \mathbb{A}^k$ are in the same orbit if and only if*

$$\sum_{i \in I} a_i = 0 \quad \text{iff} \quad \sum_{i \in I} b_i = 0 \qquad \text{for every } I \subseteq \{1, \ldots, k\}.$$

Proof. The left-to-right implication is immediate. For the right-to-left implication, we observe that if the same subsets of coordinates have zero sum, then the two tuples satisfy the same quantifier-free formulas under the vocabulary that uses the dependence relations. Under this vocabulary, the structure $\mathbb{A}$ is homogeneous, see the discussion in Example 71. Therefore, if the two tuples satisfy the same quantifier-free formulas, then they are in the same orbit, by Theorem 6.3. □

Using the claim, we complete the proof of the lemma. By definition, the support list $\bar{\sigma}$ was obtained from $\bar{\rho}$ by applying the linear map φ, which preserved non-zeroness of linear combinations of size at most $3d$. Since a tile uses at most d atoms, it follows from the above claim that for every three tiles, the corresponding triples of labels in $\bar{\sigma}$ and $\bar{\rho}$ are in the same orbit. This relation continues to hold after applying the equivariant map f, and therefore every triple of tiles in ρ has a triple of labels that is in the same orbit as the corresponding triple of labels in σ. This shows that σ is also a rejecting computation of the Turing machine. □

The above lemma shows that the Turing machine must also reject some linearly dependent tuple, which is a contradiction. Therefore, the language cannot be recognised by a deterministic polynomial time Turing machine. □

Exercises

Exercise 152. Assume the bit vector atoms. Show that if the input alphabet is $\mathbb{A}$, then nondeterministic orbit-finite Turing machines have the same expressive power as deterministic orbit-finite Turing machines (although with possibly exponential slowdown).

9.3 For equality atoms, determinisation fails

This section describes another thing that deterministic Turing machines with atoms cannot do. This time, the atoms are the equality atoms.

Theorem 9.15. *Assume the equality atoms. There is a language which:*

1. *is recognised by a nondeterministic orbit-finite Turing machine;*
2. *is not recognised by any deterministic orbit-finite Turing machine.*

In other words, deterministic orbit-finite Turing machines are not computationally complete[1], which witnesses the tightness of Theorem 9.3.

The rest of Section 9.3 is devoted to proving Theorem 9.15.

Recall from Theorem 3.9 that, when the input alphabet is a pof set, then deterministic orbit-finite Turing machines are computationally complete. Therefore, the language in the theorem needs to use an input alphabet that is not a pof set.

The language in Theorem 9.15 will be recognised by a polynomial time nondeterministic machine. Therefore, the theorem gives another example of NP≠P. Again, as mentioned at the beginning of this chapter, the theorem is unlikely to shed new light on the NP≠P question without atoms, since the proof is based on the limited way that a Turing machine can access the atoms in its tape.

The separating language. Define a *tile* to be a tuple of 8 distinct atoms, i.e. an element of $\mathbb{A}^{(8)}$. We draw tiles like this:

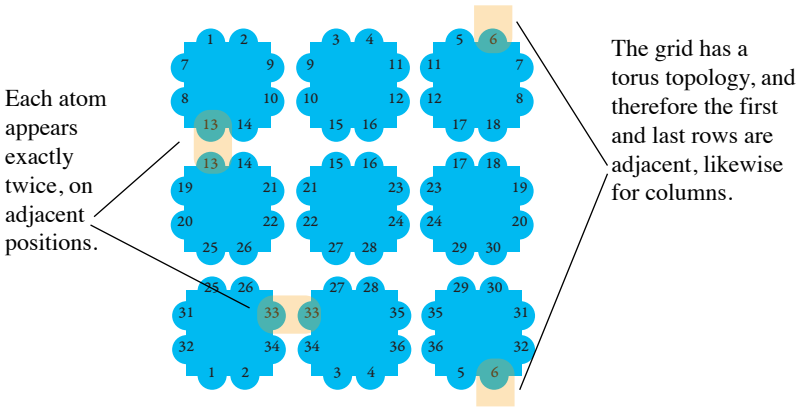
$(1,2,3,4,5,6,7,8)$ is drawn as 

We will arrange tiles on a square grid with torus topology. For $n \in \{1,2,\ldots\}$, define an $n \times n$ tiling to be a function

$$\mathcal{T} : n \times n \to \mathbb{A}^{(8)} \qquad \text{where } n \times n \stackrel{\text{def}}{=} \{0,1,\ldots,n-1\} \times \{0,1,\ldots,n-1\}.$$

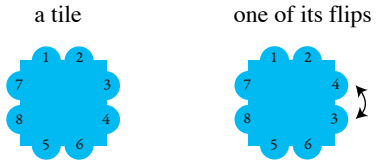
A tiling is called *consistent* if it satisfies the following constraints:

[1] The results in this section are based on Bojańczyk et al. (2013a).

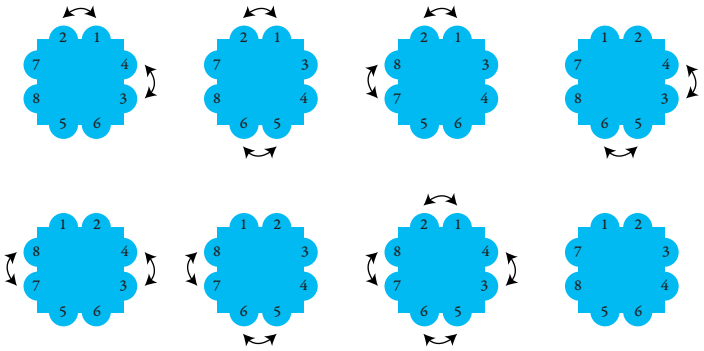


We begin with an informal description of the language that is difficult for deterministic Turing machines. One is given partial information about a tiling, namely each tile is known up to an even number of flips (see below). The question is: can the partial information be instantiated to a tiling that is consistent? This question will turn out to be doable using a nondeterministic machine – by guessing the instantiation – but will be impossible for a deterministic machine.

We now describe the partial information in more detail. A *flip* on a tile is defined to be a transposition of atoms that appear on one side, as shown in the following picture:



Define $\approx$ to be the equivalence relation on tiles, which identifies two tiles if one can be obtained from the other by doing an even number of flips. Each equivalence class of $\approx$ has eight tiles, as shown in the following picture:



Define $\mathbb{A}^{(8)}_{/\approx}$ to be the set of equivalence classes of tiles. This is an orbit-finite set. We are now ready to define the separating language.

Definition 9.16 (CFI property)**.** Define an $n \times n$ $\approx$-tiling to be a function

$$\mathcal{T} : n \times n \to \mathbb{A}^{(8)}_{/\approx}.$$

We say that $\mathcal{T}$ satisfies the CFI property[2] if there exists a consistent tiling

$$\mathcal{S} : n \times n \to \mathbb{A}^{(8)}$$

which projects to $\mathcal{T}$ when tiles are replaced by their equivalence classes.

Formally speaking, the separating language required for Theorem 9.15 should be a set of words, and not $\approx$-tilings, because Turing machines input words. Therefore, we assume some convention on linearly ordering the tiles in a $\approx$-tiling, e.g. the tiles are ordered first by columns then by rows. Under such a convention, an $n \times n$ $\approx$-tiling can be encoded uniquely as a word of length n^2 over the alphabet $\mathbb{A}^{(8)}_{/\approx}$.

To prove Theorem 9.15, we will show that a nondeterministic orbit-finite Turing machine can check if a $\approx$-tiling satisfies the CFI property, but a deterministic one cannot.

The positive part about nondeterministic machines is immediate. The work alphabet of the machine is $\mathbb{A}^{(8)}_{/\approx} \cup \mathbb{A}^{(8)}$ plus additional symbols that are used as markers. Given an input word representing some $\approx$-tiling $\mathcal{T}$, the machine uses nondeterminism to guess the consistent tiling $\mathcal{S}$ which witnesses the CFI property. Then, it deterministically checks if the adjacency constraints of a consistent tiling are satisfied by $\mathcal{S}$. This computation can be done in a polynomial number of steps.

The interesting part is that deterministic machines cannot check the CFI property.

The CFI property is not recognised by any deterministic Turing machine. We begin by discussing a doubt the reader might have at this point. Given an input representing a $\approx$-tiling $\mathcal{T}$, there are only finitely many (if exponentially many) possibilities for choosing the witness $\mathcal{S}$ as in Definition 9.16. Why not use a deterministic algorithm that exhaustively enumerates all the possibilities? The problem is that such an algorithm cannot be implemented as a deterministic Turing machine. The intuitive reason is that even if a $\approx$-equivalence class has only 8 tiles, one cannot choose deterministically any single one among them (i.e. there is no notion of the "first" or "second" element of the equivalence class) to write it down on the tape.

We now proceed to give a formal proof of why the CFI property is not recognised by any deterministic Turing machine. This will be a consequence of Lemma 9.18 below, which says that a deterministic Turing machine, unlike the CFI property, is insensitive to certain well-chosen flips in a $\approx$-tiling.

We lift the notion of flips from tiles to their $\approx$-equivalence classes as follows. If τ is a tile, then the *flip* of its $\approx$-equivalence class is defined to the $\approx$-equivalence class which contains some (equivalently, any) flip of τ. It is easy to see that this notion does not depend on the choice of τ in its $\approx$-equivalence class, nor does it depend on the

[2] The name stands for Cai, Fürer and Immerman, who first studied this property in Cai et al. (1992).

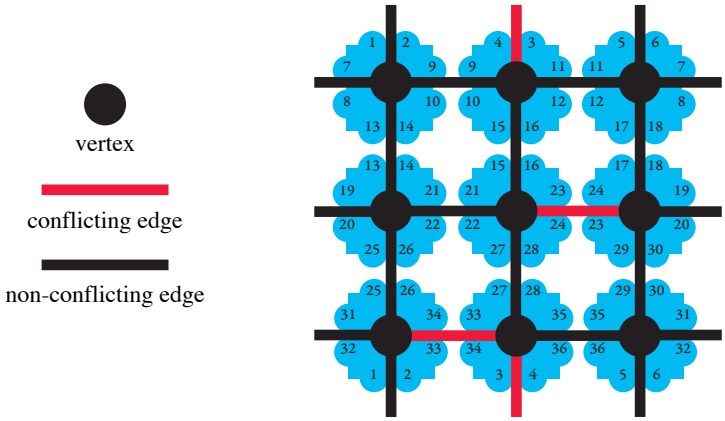
choice of which side was flipped. Flipping is an involution on $\approx$-equivalence classes, i.e. doing a flip twice leads back to the same $\approx$-equivalence class.

The following lemma shows that flips violate the CFI property.

Lemma 9.17. *Let $\mathcal{T}$ be an $n \times n$ $\approx$-tiling which satisfies the* CFI *property. Then for every $x \in n \times n$, the following $\approx$-tiling violates the* CFI *property:*

$$\mathcal{T}_x(y) \quad \stackrel{\text{def}}{=} \quad \begin{cases} \textit{flip of } \mathcal{T}(y) & \textit{if } y = x; \\ \mathcal{T}(y) & \textit{otherwise.} \end{cases}$$

Proof. A parity argument. We view an $n \times n$ grid as a graph, where vertices are grid positions, and grid positions are connected by an edge if they are adjacent in the (torus) grid topology. For $\mathcal{S} : n \times n \to \mathbb{A}^{(8)}$ define the *conflict set* to be the set of edges e in the graph corresponding to $n \times n$ such that the colours of the two sides adjoining on e are different. Here is a picture:



Using this terminology, a $\approx$-tiling $\mathcal{T}$ satisfies the CFI property only if there exists some $\mathcal{S}$ which has an empty conflict set and such that $\mathcal{T}$ is the $\approx$-equivalence class of $\mathcal{S}$. The key observation is that $\mathcal{S} \approx \mathcal{S}'$ implies that the conflict sets have the same parity (i.e. size modulo two); and furthermore making one flip makes this parity change. □

We are now ready to prove the main lemma which witnesses that the CFI property is not recognised by any deterministic orbit-finite Turing machine. Fix a deterministic orbit-finite Turing machine. We use the formalisation of computations from Section 9.2, i.e. a computation is a function $\rho : \mathbb{N}^2 \to \Delta$, where the Δ is the work alphabet plus pairs (letter of the work alphabet, state of the machine). If $\mathcal{T}$ is a $\approx$-tiling, we write $\rho_{\mathcal{T}}$ for the unique computation of the fixed Turing machine on the word representing $\mathcal{T}$.

Lemma 9.18. *There exists $k \in \{0, 1, \ldots\}$ with the following property. Let $n \in \{0, 1, \ldots\}$ be sufficiently large, and let $\mathcal{T}$ be an $n \times n$ $\approx$-tiling which satisfies the* CFI *property. Assuming the notation $\mathcal{T}_x$ defined in Lemma 9.17, the following holds for every $i, j \in \mathbb{N}$:*

$$\rho_{\mathcal{T}}(i, j) = \rho_{\mathcal{T}_x}(i, j) \qquad \textit{for all } x \in n \times n \textit{ with at most } k^2 \textit{ exceptions.} \qquad (*)$$

Before proving the lemma, we use it to finish the proof of Theorem 9.15. Take k as in the lemma, and let n be sufficiently large. Let $\mathcal{T}$ be some $n \times n$ $\approx$-tiling which satisfies the CFI property. Consider the computation $\rho_{\mathcal{T}}$, and let (i, j) be the place in the computation which contains the head at the moment when it accepts. If $n > k^2$, then (*) in the lemma implies that there is some $x \in n \times n$ such that $\rho_{\mathcal{T}_x}$ has the same contents. In particular, the machine also accepts $\mathcal{T}_x$. This contradicts Lemma 9.17.

Proof of Lemma 9.18. Choose k so that

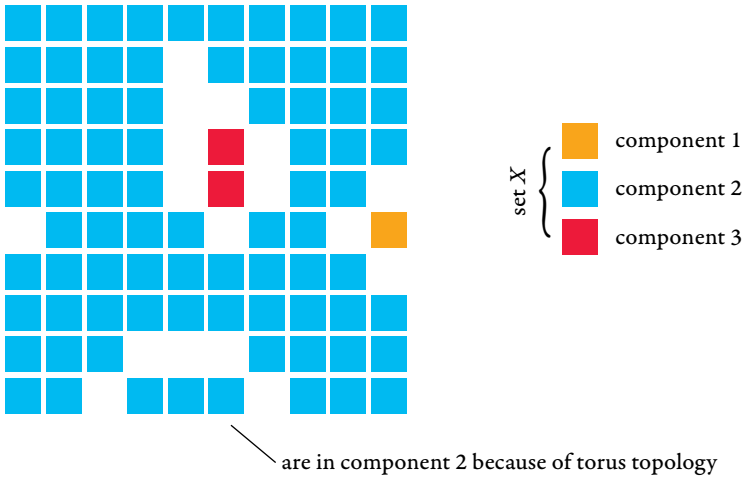
$$k/2 > \underbrace{\text{support size for cell contents}}_{\substack{\text{smallest } l \text{ such that } \rho_{\mathcal{T}}(i,j) \text{ has a} \\ \text{support of size } l \text{ for every } i, j \in \mathbb{N}}}.$$

We prove (*) by induction on i, i.e. the number of computation steps of the Turing machine. For the induction base of $i = 0$, we observe that the contents of a cell in time $i = 0$ depend only on the value of the input in at most one grid position, and hence (*) holds with at most one exception.

For the induction step, suppose that (*) is true for $i - 1$ and consider the case of i. In the computation of a Turing machine, the contents of a cell in time i are uniquely determined by the contents of at most two cells in time $i - 1$: the cell in the same column (offset from the beginning of the tape), plus possibly the contents of the unique cell in time $i - 1$ which contains the head of the machine. Hence, using the induction assumption we can conclude the following weaker version of (*), which uses $2k^2$ exceptions instead of k^2:

$$\rho_{\mathcal{T}}(i, j) = \rho_{\mathcal{T}_x}(i, j) \qquad \text{for all } x \in n \times n \text{ with at most } 2k^2 \text{ exceptions.} \qquad (**)$$

In the rest of this proof, we bring back the number of exceptions down to k^2. To do this, we talk about connected components in $\mathcal{T}$ after removing some grid positions from the input $\mathcal{T}$. For a subset $X \subseteq n \times n$ of grid positions, define its *connected components* to be the connected components in the subgraph of the graph of $n \times n$ (as defined in the proof of Lemma 9.17) that is induced by X. Here is a picture of a set X together with its partition into connected components:



We now resume the proof of the implication from (**) to (*). Choose a tuple of atoms $\bar{a}$ which supports the cell contents $\rho_{\mathcal{T}}(i, j)$. By choice of k, we can assume that $\bar{a}$ has at most $k/2$ atoms. Define

$$Z = \{\ x \in n \times n \mid \text{ some atom from } \bar{a} \text{ appears in } \mathcal{T}(x) \quad \}$$

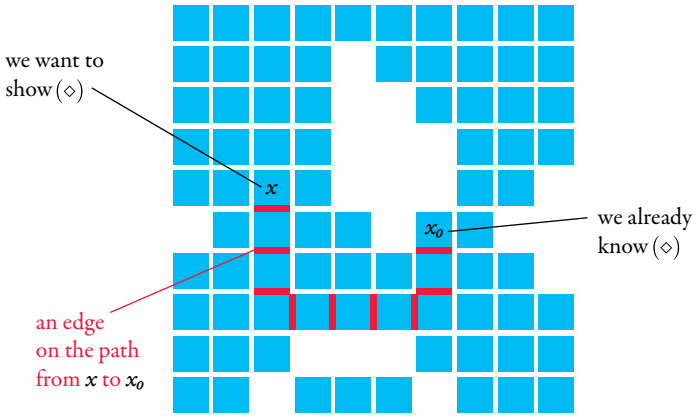
to be the grid positions where the input instance uses at least one atom from $\bar{a}$. The set Z has at most k grid positions, since every atom appears in two grid positions and k is at least twice as large as $\bar{a}$. By a straightforward analysis of connectivity in an $n \times n$ grid, one can conclude that if n is big enough, then the graph corresponding to $n \times n - Z$ has a connected component, call it X, which contains all grid positions from $n \times n$ with at most k^2 exceptions. If n is big enough, then

$$\underbrace{2k^2}_{\text{number of exceptions in (**)}} < \underbrace{n^2 - k^2}_{\text{size of } X},$$

and therefore there is some $x_0 \in X$ which is not an exception, i.e. it satisfies

$$\rho_{\mathcal{T}}(i, j) = \rho_{\mathcal{T}_x}(i, j). \tag{$\diamond$}$$

Using this x_0, we will show that every grid position $x \in X$ also satisfies ($\diamond$), thus proving (*). Let $x \in X$. Since X is connected and disjoint from Z, in the graph corresponding to $n \times n$ there is a path which goes from x to x_0 and avoids grid positions from Z. Here is a picture:



Every edge e of the grid $n \times n$ corresponds to two distinct atoms. The edges on the path from x to x_0 avoid atoms from support $\bar{a}$, because they avoid grid positions from Z. Define π to be the atom automorphism which swaps, for every e on the path from x to x_0, the two atoms that correspond to the edge e. This atom automorphism fixes all atoms from $\bar{a}$. For each tile except those corresponding to x and x_0, the automorphism flips an even number of sides, and hence we have:

$$\mathcal{T}_x = \pi(\mathcal{T}_{x_0}). \tag{9.1}$$

We are now ready to prove that x satisfies (⋄):

$$\begin{aligned}
\rho_{\mathcal{T}_x}(i,j) &= \text{(by (9.1))} \\
\rho_{\pi(\mathcal{T}_{x_0})}(i,j) &= \text{(because the Turing machine is equivariant)} \\
\pi((\rho_{\mathcal{T}_{x_0}})(i,j)) &= \text{(because } x_0 \text{ satisfies (⋄))} \\
\pi((\rho_{\mathcal{T}})(i,j)) &= \text{(because } \pi \text{ fixes the support of } \rho_{\mathcal{T}}(i,j)) \\
\rho_{\mathcal{T}}(i,j).
\end{aligned}$$

This completes the proof of the lemma, and therefore also of Theorem 9.15. □

Exercises

Exercise 153. Assume the equality atoms. Show that if $k \leq 3$ and the input alphabet Σ is k-tuples of atoms modulo some equivariant equivalence relation, then every nondeterministic Turing machine over input alphabet Σ can be determinised.

Exercise 154. In the proof of Theorem 9.15, we used an input alphabet which consisted of 8-tuples of atoms modulo some equivalence relation. Improve the proof to use 6-tuples modulo some equivalence relation[3].

Exercise 155. Assume the equality atoms and consider the alphabet

$$\{\{\{a,b,c\},\{d,e,f\}\} : a,b,c,d,e,f \text{ are distinct atoms}\}.$$

Show that Turing machines over this input alphabet cannot be determinised.

[3]This exercise is based on Klin et al. (2014); in particular Section 5.1 of that paper shows that 5 is the smallest dimension where Theorem 9.15 holds.

Chapter 10

Sets of sets of sets

In this chapter, we discuss an approach to atoms that is based in set theory. In the usual set theory, the element of a set are simpler sets, and the elements of those simpler sets are even simpler sets, and so on until one reaches the empty set. The underlying principle is extensionality, which states that two sets are equal if they have the same elements. An alternative approach is to postulate the existence of atoms, which do not have any elements, and to allow sets to contain such atoms. This approach is called sets with atoms, or sets with urelements, and it was present in Zermelo's set theory from 1907. Atoms became less popular in set theory, but an important exception is the permutation models of Fraenkel and Mostowski in the 1920s and 1930s, which were used to show various independence results. The permutation models are important for this book, because they introduced the finite support condition, which is a key concept here.

There is another reason to discuss sets, apart from its connections to set theory and mathematical logic. The other reason is computational – sets are a good data structure. To understand this, recall the orbit-finite Turing machines that were discussed in the previous chapter. A limitation of this model is its list-based data structure: all information has to be stored on a linearly ordered tape, where each cell stores a constant number of atoms. This limitation is behind the failure of determinisation that was proved in Theorem 9.15; a deterministic Turing machine was unable to store the set of possible ways to order the atoms in CFI instance. Apart from their usefulness in computation, sets can also be a more natural representation. For example, if we want a graph to be the input of an algorithm, it is very natural to assume that the graph is given as two sets (vertices and edges), with the algorithm having access to these sets (such as looping over vertices or edges, or testing if an edge is incident with a vertex). This is arguably superior to the alternative that we have used so far, which is to choose some linear representation of these sets. Such representations can be not only cumbersome, but more importantly, they can be problematic, as it is not always clear how the choice of representation affects the computational power.

For the reasons described above, in the next few chapters we will discuss a set based approach to atoms.

10.1 Sets with atoms

In this section, we define what a set with atoms is. The definition is given below, and it is parameterised by an ordinal number n, which is called the rank of a set. Ordinal numbers are like natural numbers, but they can be infinite. All natural numbers are ordinals, but there are also infinite ordinals, such as ω, which is the first infinite ordinal, or $\omega + 1$, which is the next ordinal after ω. For readers unfamiliar with ordinal numbers, it is useful to simply think of them as natural numbers. This intuition will be sufficient for most cases, in particular the hereditarily orbit-finite sets that will be introduced later in this chapter will only use natural numbers as their ranks.

Definition 10.1 (Set with atoms)**.** For an oligomorphic structure $\mathbb{A}$, sets with atoms over $\mathbb{A}$ are defined as follows by induction on a parameter, which is an ordinal number $n \geq 1$ that is called the *rank*. A set with atoms of rank n is defined to be a set X, such that:

1. elements of X are either atoms $a \in \mathbb{A}$, or sets with atoms of rank $< n$; and
2. there is a finite support, which means that there is some $\bar{a} \in \mathbb{A}^*$ such that

$$\bar{a} = \pi(\bar{a}) \quad \Rightarrow \quad X = \underbrace{\{\, \pi(x) \mid \ x \in X \ \}}_{\text{we call this set } \pi(X)} \qquad \text{for every atom automorphism } \pi.$$

Example 72. We begin by illustrating sets of minimal rank, namely one. For such a set, all of its elements must be atoms, since there are no sets of smaller rank. Therefore, a set of rank one must be a subset of the atoms. This subset must be finitely supported. For example, the full set $\mathbb{A}$ is finitely supported, namely it has empty support, and this is true for any oligomorphic atom structure. Also, every finite subset of $\mathbb{A}$ is finitely supported, since as the support one can take any list that contains all atoms used in the set. Here is an example of an infinite but finitely supported subset, which is specific to the order atoms:

$$\{\, a \mid \ a \in \mathbb{A} \text{ is such that } 1 < a < 3 \ \}.$$

This set is supported by its endpoints 1 and 3, since any atom automorphism that preserves the endpoints will also map the set to itself. If the atoms are the random graph, and we take some $a \in \mathbb{A}$, then its neighbours

$$\{\, b \mid \ b \in \mathbb{A} \text{ has an edge to } a \ \}$$

will be a set supported by a, and therefore also a legitimate set with atoms of rank one. □

Example 73. Let us now consider sets with atoms of rank two. These are sets whose elements are atoms, or sets of rank one. One example is the finite powerset

$$\mathsf{P}_{\text{fin}}\mathbb{A} = \{\, X \mid \ X \subseteq \mathbb{A} \text{ is finite } \ \},$$

Another example that also works for any atoms is the finitely supported powerset of the atoms:

$$\{\, X \mid \ X \subseteq \mathbb{A} \text{ is finitely supported } \}.$$

In the equality atoms, the finitely supported powerset will contain finite and cofinite sets. For the ordered atoms, the finitely supported powerset will contain finite Boolean combinations of intervals. Another example, which also works for any atoms, is the family of sets of size three:

$$\binom{\mathbb{A}}{3} \quad = \quad \{\, \{a,b,c\} \mid \ a,b,c \in \mathbb{A} \text{ are pairwise different } \}.$$

In all the above examples, the set itself is equivariant (has empty support), but its elements are sets that are not necessarily equivariant. Here is an example, which uses the equality atoms, that is not equivariant itself:

$$\binom{\mathbb{A} \setminus \{\text{Mary}\}}{3} \quad = \quad \{\, \{a,b,c\} \mid \ a,b,c \in \mathbb{A} \text{ are pariwise different and not Mary } \}.$$

Similarly, for the ordered atoms, we could consider the family of intervals that avoid 0. □

Example 74. In all the above examples, the sets would contain atoms, or their elements would contain atoms, and so on. This is of course not necessary, and we can consider sets that are built without using atoms, such as:

$$\underbrace{\emptyset}_{\text{rank one}} \qquad \underbrace{\{\emptyset\}}_{\text{rank two}} \qquad \underbrace{\{\{\emptyset\}\}}_{\text{rank three}} \qquad \cdots$$

Since these sets do not use any atoms, atom permutations act on them trivially, i.e. do not move them. In particular, they have empty supports. Also, we can consider the set which contains all the sets described above; this set will have rank ω, since it has elements of arbitrarily large finite ranks. This new set can be thought of as representing the natural numbers. □

One of the advantages of sets with atoms as defined in Definition 10.1 is that they come automatically equipped with an action of atom automorphisms. If we have a set with atoms X and an atom automorphism π, then $\pi(X)$ is defined by applying π to all atoms that are elements of X, the sets that are elements of X, and so on. This is in contrast with the previous approach in Definition 5.11, where each set needed to come equipped with a group action of atom automorphism. We usually used some implicit group action; for example when the set was $\mathbb{A}^d$, we implicitly assumed that atom automorphisms would act on this set coordinate-wise. Now, if we use sets with atoms as in Definition 10.1, we no longer need to specify the action each time – explicitly or implicitly – since it is already given by the definition of sets with atoms.

Sets with atoms are defined using only one constructor, namely sets. However, this constructor is powerful enough to define other constructors, such as pairing.

Example 75. [Pairing] Let x and y be two atoms, or sets with atoms. Their Kuratowski pair is defined to be the following set with atoms

$$(x, y) \quad \stackrel{\text{def}}{=} \quad \{\{x\}, \{x, y\}\}.$$

The point of the construction above is that we can extract the first and second coordinates of a pair. Indeed, if the Kuratowski pair contains one element only, then we know that this element is a singleton $\{x\}$, and the original pair had to be (x, x). Otherwise, if the Kuratowski pair contains two sets, then one of them must be a singleton $\{x\}$, which gives us the first coordinate, and the other one must be a set with two elements, which gives us the second coordinate by removing x from the set. □

Example 76. [Lists] We can use repeated pairing to define lists:

$$[x_1, \ldots, x_n] \quad \stackrel{\text{def}}{=} \quad \begin{cases} \emptyset & \text{if } n = 0, \\ (x_1, [x_2, \ldots, x_n]) & \text{if } n > 0. \end{cases}$$

In particular, we can think of $\mathbb{A}^d$ or $\mathbb{A}^*$ as sets with atoms. □

An informal idea behind an equivariant set is that it can be "defined" without referring to any individual atoms. Using sets with atoms, this informal idea can be given a precise meaning.

Definition 10.2 (Superstructure)**.** For an oligomorphic atom structure $\mathbb{A}$, define its *superstructure* to be the structure that is obtained as follows. The universe is

$$\mathbb{A} \cup \text{sets with atoms},$$

i.e. every element in the universe is either an atom, or a set with atoms. It is equipped with the following relations: (a) a binary relation for membership; (b) all equivariant relations $R \subseteq \mathbb{A}^d$ on the atoms. The relations in item (b) select only tuples of atoms.

Consider a formula $\varphi(x_1, \ldots, x_n)$ over the vocabulary of the superstructure. This formula defines an n-ary relation over the superstructure. Such a relation is said to be *definable in the superstructure*. The superstructure is very rich, since quantification in it ranges over sets, or sets of sets, etc. Therefore, any concept definable in the language of set theory will be definable here.

Example 77. One can define the relation "y is the singleton of x" using the following formula in the superstructure:

$$x \in y \wedge \forall z (z \in y \Rightarrow z = x).$$

Similarly, one can define the relation "y is the Kuratowski pair of x_1 and x_2". Let us now define the set of all lists, i.e. relation "y is equal to x^*". By inspecting the definition of lists from Example 76, we see that x^* is the unique set which is contained in every set z that contains the empty set, and is closed under prepending elements from x:

$$\emptyset \in z \quad \wedge \quad \forall u \forall v \; u \in x \wedge v \in z \Rightarrow (u, v) \in z.$$

Our definition of the set of lists x^* describes a general principle, namely that we can use first-order logic in the downset to formalise inductive definitions. This way, we can define list concatenation, i.e. the relation "y is the concatenation of lists x_1 and x_2". □

As we have seen in the above example, essentially any mathematical construction – and certainly all notions that have been used in this book – can be defined in the superstructure. The following theorem shows that such constructions will be necessarily equivariant.

Theorem 10.3 (Equivariance principle)**.** *Let $\mathbb{A}$ be an oligomorphic atom structure, and let $\varphi(x_1, \ldots, x_n)$ be a first-order formula over the vocabulary of the superstructure. Then*

$$\textit{superstructure} \models \varphi(x_1, \ldots, x_n) \quad \Leftrightarrow \quad \textit{superstructure} \models \varphi(\pi(x_1), \ldots, \pi(x_n))$$

for every automorphism π of the atoms.

Proof. An immediate corollary of the definitions. An automorphism π of the atoms does not affect the relations of the superstructure, i.e. membership and equivariant relations over the atoms. □

The above principle can be used to prove equivariance of essentially all constructions that do not refer to individual atoms. For example, the construction of a minimal automaton can be defined in the superstructure, and therefore it will be equivariant.

10.2 Sets builder expressions

In this section, we present a syntax for describing some simple orbit-finite sets, which is called set builder expressions[1]. We have already seen such expressions in Examples 72 and 73. In case the reader has forgotten those examples, here are two more. The following expression describes the family of all subsets of the equality atoms that miss at most one atom:

$$\{\{x \mid \text{ for } x \in \mathbb{A}\} \cup \{y \mid \text{ for } y \in \mathbb{A} \text{ such that } y \neq x\} \mid \text{ for } x \in \mathbb{A}\}.$$

Another example, this time in the atoms $(\mathbb{Q}, <)$, is the set of all nonempty bounded open intervals that contain only negative numbers:

$$\{\{z \mid \text{ for } z \in \mathbb{A} \text{ such that } x < z < y\} \mid \text{ for } x, y \in \mathbb{A} \text{ such that } x < y \wedge y < 0\}.$$

Definition 10.4 (Set builder expressions)**.** Fix an atom structure $\mathbb{A}$ and an infinite set of variables, which range over atoms. We write x, y, z for these variables. There are two constructors for set builder expressions, namely set comprehension and union:

[1] The idea to use set builder notation as a way of representing hereditarily orbit-finite sets originates from a programming language introduced in Bojańczyk and Toruńczyk (2012), which later developed into the language LOIS (*Looping over Infinite Sets*) of Kopczyński and Toruńczyk (2016); Kopczyński and Toruńczyk (2017).

1. **Set comprehension.** Let α be a variable or an already defined set builder expression, and let φ be a first-order formula over the vocabulary of $\mathbb{A}$, which is called the *guard*. Then

$$\{\alpha \mid \text{ for } x_1, \ldots, x_d \in \mathbb{A} \text{ such that } \varphi\}$$

   is a set builder expression. The free variables of this expression are the variables that are free in α or the guard φ, minus $x_1, \ldots, x_d$.

2. **Union.** If $\alpha_1, \ldots, \alpha_n$ are already defined set builder expressions, then $\alpha_1 \cup \cdots \cup \alpha_n$ is a set builder expression. A variable is free in this expression if it is free in some α_i.

The semantics of set builder expressions are defined in the expected way: if a set builder expression α has d free variables, then its semantics is a function

$$[\![\alpha]\!] : \mathbb{A}^d \to \text{set with atoms} \tag{10.1}$$

which inputs the values of the variables, and outputs the corresponding set with atoms. This function is defined by induction on the structure of the expression in the obvious way. The function is easily seen to be equivariant. By abuse of notation, we sometimes skip the brackets and write $\alpha(\bar{a})$ instead of the formally correct $[\![\alpha]\!](\bar{a})$.

Definition 10.5 (Representable)**.** A set with atoms is called *representable* if it is equal to $\alpha(\bar{a})$ for some set builder expression α and some tuple of atoms $\bar{a}$ that instantiates its free variables.

To write down a representable set, we need to write down the set builder expression, and the values of the atoms. For the latter, we assume an atom representation, as in Definition 7.2. Therefore, whenever we talk about algorithms that input representable sets, we implicitly assume some atom representation, at least as long as we want to talk about sets that are represented by set builder expressions that have free variables.

Not all sets with atoms are representable. For example, every representable set will have finite rank, while some sets with atoms will have ranks that are infinite ordinal numbers, such as ω. Another, related, reason is that there are countably many representable sets, but uncountably many sets with atoms. Consider the sets

$$\underbrace{\emptyset}_{\text{rank one}} \qquad \underbrace{\{\emptyset\}}_{\text{rank two}} \qquad \underbrace{\{\{\emptyset\}\}}_{\text{rank three}} \qquad \cdots$$

that were discussed in Example 74. Individually, each of these sets is representable, but this will no longer be the case when we combine infinitely many of them into a single set. Let X be any set which contains only some (but not necessarily all) such sets. Such a set X can be chosen in uncountably many ways. As explained in Example 74, this X will necessarily be a set with atoms – although a degenerate one that does not use any atoms. If X has infinitely many elements, then it will have infinite rank (namely ω) and as such it will not be represented by any set builder expression.

We will now give a more semantic characterisation of the representable sets with atoms. The general idea is these sets are orbit-finite in a hereditary way: they are orbit-finite, their elements are orbit-finite, the elements of the elements are orbit-finite, and so on. This is, however, not technically accurate, since we are dealing with sets that are not necessarily equivariant, and the definition of orbit-finiteness in Definition 5.11 requires the set to be equivariant (because it is closed under applying all atom automorphisms). To overcome this difficulty, we use a generalisation of orbit-finiteness that is appropriate to non-equivariant sets.

Definition 10.6 (Intersecting finitely many orbits)**.** A set with atoms X *intersects finitely many orbits* if there are finitely many elements $x_1, \ldots, x_n \in X$ such that every element of X is of the form $\pi(x_i)$ for some i and some atom automorphism π.

Example 78. Consider the equality atoms. Any finite set, such as {Mary, John} intersects finitely many orbits. More generally, any finitely supported subset of $\mathbb{A}$ will intersect finitely orbits, namely one orbit, since $\mathbb{A}$ is a set with one orbit. Similarly, any finitely supported subset of $\mathbb{A}^d$ will intersect finitely many orbits. □

Theorem 10.7. *Consider an oligomorphic atom structure $\mathbb{A}$. A set with atoms X is representable if and only if*

(*) Y intersects finitely many orbits for every set Y that is equal to X, an element of X, an element of an element of X, and so on.

Proof. The implication $\Rightarrow$ is proved by a straightforward induction on the size of the set builder expression. We spell this induction out in more detail below, to accustom the reader to the semantics of set builder expressions. The union constructor is immediate, since sets satisfying condition (*) are clearly closed under taking finite unions. Consider now the set comprehension constructor

$$\beta = \{\alpha \mid \text{ for } x_1, \ldots, x_d \in \mathbb{A} \text{ such that } \varphi\}.$$

We want to show that for every choice of values for the free variables, the corresponding set intersects finitely many orbits. Let k be the number of free variables of the inner expression α. Any set of represented by the outer expression β will be obtained by choosing some finitely supported $X \subseteq \mathbb{A}^d$, and returning the set of all values of α for tuples $\bar{a} \in X$. As we have explained in Example 78, the set X intersects finitely many orbits. Also, if we apply an equivariant function (such as the semantics of the inner expression α) to a set that intersects finitely many orbits, then the resulting image will also intersect finitely many orbits. Therefore, any set represented by the outer expression β will intersect finitely many orbits.

We now prove the converse implication $\Leftarrow$, which says that every set with atoms that satisfies (*) is representable. The proof is by induction on the rank of the set. The induction basis of rank zero applies to only one set, the empty set, which is clearly represented by a set builder expression. (For example, we can use an unsatisfiable guard in a set expression, or an empty union in a union expression.) In the induction step, we will use Lemma 10.9 below, which says that the notion of intersecting finitely many orbits is robust under restricting the orbits to respect a certain support, as explained in the following definition.

Definition 10.8. Let x be an atom, a set with atoms, or more generally, an element of a set that is equipped with an action of atom automorphisms. For a tuple of atoms $\bar{a}$, define the *$\bar{a}$-orbit* of x to be the set

$$\{\, \pi(x) \mid \underbrace{\pi \text{ is an atom automorphism that fixes } \bar{a}}_{\text{such an automorphism is called a } \bar{a}\text{-automorphism}} \,\}.$$

We already saw $\bar{a}$-orbits in Lemma 8.15 in the chapter about vector spaces. The notion of intersecting finitely many $\bar{a}$-orbits is defined in the same way as in Definition 10.6, but with $\bar{a}$-orbits instead of the usual orbits. The lemma below shows that the notion does not depend on the choice of $\bar{a}$. (A weaker form of this lemma was implicit in the proof of Lemma 8.15.)

Lemma 10.9. *Let X be a set with atoms, and let $\bar{a}$ and $\bar{b}$ be tuples of atoms. Then X intersects finitely many $\bar{a}$-orbits if and only if it intersects finitely may $\bar{b}$-orbits.*

Proof. It is enough to prove the lemma in the special case $\bar{b}$ is the empty tuple. As the support $\bar{a}$ grows, the $\bar{a}$-orbit gets smaller, because the constraint on π gets tighter. Therefore, we need to show that if X intersects finitely many orbits (in the usual sense, for the empty tuple of atoms) then it intersects finitely many $\bar{a}$-orbits, for every $\bar{a}$. In other words, we need to show that every orbit splits into finitely many $\bar{a}$-orbits. Consider then the orbit (with empty support) of some element x. This orbit is an orbit-finite set in the sense of Definition 5.11, and so we can apply Claim 9.5 to it, yielding an equivariant surjective function

$$f : \mathbb{A}^d \to \{\, \pi(x) \mid \ \pi \text{ is an atom automorphism } \,\}.$$

We need to show that the range of this function splits into finitely many $\bar{a}$-orbits. By equivariance, we know that if two inputs of f are in the same $\bar{a}$-orbit, then the corresponding outputs are also in the same $\bar{a}$-orbit. Therefore, it is enough to show that the domain of the function splits into finitely many $\bar{a}$-orbits. Two tuples from $\mathbb{A}^- d$ are in the same $\bar{a}$-orbit of $\mathbb{A}^d$ if and only if appending $\bar{a}$ to both tuples gives two tuples that are in the same equivariant orbit of $\mathbb{A}^{d+k}$, where k is the length of $\bar{a}$. Since the $\mathbb{A}^{d+k}$ has finitely many equivariant orbits by oligomorphism, it follows $\mathbb{A}^d$ has finitely many $\bar{a}$-orbits. □

Using the above lemma, we prove the induction step in the theorem. Consider a set with atoms X that satisfies (*). Like any set of atoms, this set $\bar{X}$ is supported by some tuple of atoms $\bar{a}$. This means X that is invariant under applying automorphisms that fix $\bar{a}$, which in turn means that X is equal to a union of $\bar{a}$-orbits. By the above lemma, this union is finite. Since sets represented by set builder expressions are closed under taking finite unions, it remains to show that every component of the union, which is of the form

$$\{\, \pi(x) \mid \ \pi \text{ is an atom automorphism that fixes } \bar{a} \ \,\} \tag{10.2}$$

for some $x \in X$, is represented by some set builder expression. The element x can either be an atom, or a set that satisfies (*). We only consider the case when x is a set;

the case when x is an atom is proved in the same way. By induction hypothesis, we know that $x = \beta(\bar{b})$ for some set builder expression β and some valuation $\bar{b}$ of its free variables. By equivariance of the semantics of set builder expressions, we know that the set in (10.2) is equal to

$$\{\, \beta(\pi(\bar{b})) \mid \ \pi \text{ is an atom automorphism that fixes } \bar{a} \ \} = \{\, \beta(\bar{c}) \mid \ \bar{c} \text{ is in the } \bar{a}\text{-orbit of } \bar{b} \ \}.$$

The conditions on $\bar{c}$ in the above set is the same as saying that $\bar{a}\bar{b}$ and $\bar{a}\bar{c}$ are in the equivariant orbit. By Theorem 5.7, the equivariant orbit of $\bar{a}\bar{b}$ can be defined by a first-order formula φ. Therefore, the above set consists of possible values of $\beta(\bar{c})$, with $\bar{c}$ ranging over tuples such that $\bar{a}\bar{c}$ satisfies the formula φ. This is an instance of set comprehension.

Observe that in the proof of the implication $\Rightarrow$, we constructed an instance of a set comprehension expression where the guard referred only to the support $\bar{a}$ and the bound variables $\bar{c}$. Therefore, we have shown a slightly stronger result: if a set with atoms satisfies (*) and is supported by $\bar{a}$, then it is of the form $\alpha(\bar{a})$ for some set builder expression α. In particular, if the set is equivariant, then the expression has on free variables. □

Among all representable sets, we have those which are represented by a set builder expression without free variables. By the remarks at the end of the above proof, these are exactly the representable sets that are equivariant. The following observation shows that these sets cover all orbit-finite sets, as defined in Definition 5.11, up to equivariant bijections.

Corollary 10.10. *Consider an oligomorphic atom structure $\mathbb{A}$. A set is orbit-finite if and only if it admits an equivariant bijection with a set represented by a set builder expression that has no free variables.*

Proof. We begin with the implication $\Leftarrow$. Suppose that X is represented by a set builder expression that has no free variables. By Theorem 10.7, X intersects finitely many orbits. Since the expression has no free variables, the set that it represents is equivariant (this is because the semantics of set builder expressions are equivariant). Therefore, X not only intersects finitely many orbits, but it is equal to finitely many orbits.

Let us now prove the implication $\Rightarrow$. Theorem 5.12 says that every orbit-finite set admits an equivariant bijection with a subquotiented pof set. It is not hard to see that every subquotiented pof set satisfies condition (*) from Theorem 10.7, and therefore it is represented by some set builder expression. By the remarks at the end of the proof of Theorem 10.7, this set builder expression has no free variables. □

Thanks to the above theorem, we get another representation of orbit-finite sets, apart from the representation in terms of subquotiented pof sets from Theorem 5.12. As we will see in the next chapter, this representation can be used as a basis for a programming language that operates on sets.

Algorithms on representable sets. We finish this chapter with some basic algorithms that operate on representable sets. As an input to an algorithm, such sets are given by a set builder expression, together with atoms that instantiate its free variables. As mentioned before, this representation implicitly depends on the representation of the atoms, at least as long as we want to talk about sets that not equivariant, and therefore need free variables in their set builder expressions. We begin with the following theorem, which shows that one can decide membership, equality and inclusion, and also compute Boolean operations.

Theorem 10.11. *Consider an oligomorphic structure $\mathbb{A}$, together with some atom representation. Given representations of sets with atoms X and Y, one can compute:*

1. *the answers to the following questions: $X = Y$, $X \in Y$ and $X \subseteq Y$;*
2. *representations of the sets $X \cup Y$, $X \cap Y$ and $X \setminus Y$.*

Proof. The main observation in the proof, which will be stated in the Symbol Pushing Lemma below, is that the problems in the statement of the theorem can be reduced to the first-order theory of the atoms, using straightforward syntactic transformations. These transformations are even polynomial, if we define the *size* of a first-order formula to be the number of distinct subformulas (even if a subformula is used multiple times, it is counted only once in the size[2]). Similarly, we define the size of a set builder expression to be the number of distinct subexpressions and subformulas of formulas used in the guards.

Lemma 10.12 (Symbol Pushing Lemma)**.** *There is polynomial time algorithm which does the following.*

- **Input.** *Set builder expressions α, β, with free variables $\bar{x}$ and $\bar{y}$, respectively.*
- **Output.** *A first-order formula $\varphi(\bar{x}\bar{y})$ over the vocabulary of the atoms such that:*

$$\mathbb{A} \models \varphi(\bar{a}\bar{b}) \quad \text{iff} \quad \alpha(\bar{a}) \subseteq \beta(\bar{b}) \qquad \text{for every atom tuples } \bar{a}, \bar{b}.$$

Likewise for $\in$ or $=$ instead of $\subseteq$.

Proof. See Figure 10.1. Each of the formulas in Figure 10.1 consists of a fixed part and inductively defined subformulas corresponding to subexpressions of α and β. It follows that the size of the first-order formula produced in Figure 10.1 is approximately the number of subexpressions in α times the number of subexpressions in β. Observe that the proof of this lemma does not need any assumptions on the atom structure, such as oligomorphism or decidability of the first-order theory, since it simply rewrites one formula into another formula. □

Using the Symbol Pushing Lemma, we can answer the questions in the first item of the theorem. For example, if we want to check if $X = Y$, then we compute the

[2] This corresponds to viewing formulas as directed acyclic graphs (circuits), rather than trees. For first-order formulas, this distinction is not very important, because repeated uses of the same subformula can be eliminated by using quantifiers, see Exercise 158.

$$
\begin{array}{rcl}
\underline{\alpha \in \beta} & \stackrel{\text{def}}{=} & \mathit{false} \quad \text{if } \beta \text{ represents an atom} \\
\underline{\alpha \in (\beta_1 \cup \cdots \cup \beta_n)} & \stackrel{\text{def}}{=} & \bigvee_i \underline{\alpha \in \beta_i} \\
\underline{\alpha \in \{\beta \mid \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \varphi\}} & \stackrel{\text{def}}{=} & \exists x_1 \ldots \exists x_n \; \varphi \wedge \underline{\alpha = \beta} \\
\\
\underline{\alpha \subseteq \beta} & \stackrel{\text{def}}{=} & \mathit{false} \quad \text{if } \alpha \text{ represents an atom} \\
\underline{(\alpha_1 \cup \cdots \cup \alpha_n) \subseteq \beta} & \stackrel{\text{def}}{=} & \bigwedge_i \underline{\alpha_i \subseteq \beta} \\
\underline{\{\alpha \mid \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \varphi\} \subseteq \beta} & \stackrel{\text{def}}{=} & \forall x_1 \ldots \forall x_n \; \varphi \Rightarrow \underline{\alpha \in \beta} \\
\\
\underline{\alpha = \beta} & \stackrel{\text{def}}{=} & \alpha = \beta \quad \text{if } \alpha, \beta \text{ represent atoms} \\
\underline{\alpha = \beta} & \stackrel{\text{def}}{=} & \underline{\alpha \subseteq \beta} \wedge \underline{\beta \subseteq \alpha} \quad \text{if } \alpha, \beta \text{ represent sets} \\
\underline{\alpha = \beta} & \stackrel{\text{def}}{=} & \mathit{false} \quad \text{otherwise}
\end{array}
$$

Figure 10.1: In the figure, α and β are either set builder expressions (in which case they represent sets), or variables and atoms constants (in which case they represent atoms). For each relationship, e.g. membership $\alpha \in \beta$, the figure shows a corresponding first-order formula, which is denoted using underlines, e.g. $\underline{\alpha \in \beta}$.

corresponding first-order formula from the Symbol Pushing Lemma, and then we use the assumption on decidability of the first-order theory to check if this formula is true. The same applies to checking membership and inclusion.

A similar approach applies to the Boolean operations from the second item of the theorem. For union $X \cup Y$, there is nothing to do, since it is built into the syntax of set builder expressions. Consider the case of intersection $X \cap Y$. By distributing intersection across union, we can assume that the outermost constructors in the expressions for X and Y are set comprehension:

$$\underbrace{\{\alpha \mid \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \varphi\}}_{X}, \quad \underbrace{\{\beta \mid \text{ for } y_1, \ldots, y_m \in \mathbb{A} \text{ such that } \psi\}}_{Y}.$$

Apply the Symbol Pushing Lemma, yielding a polynomial size formula $\underline{\alpha = \beta}$ which characterises those valuations of the free variables of the set builder expressions α and β which make them equal. The expression for intersection is then

$$\{\alpha \mid \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \varphi \wedge \exists y_1 \ldots \exists y_m \, \psi \wedge \underline{\alpha = \beta}\}.$$

The same kind of argument applies to the set difference $X \setminus Y$. This completes the proof of the theorem. □

In the above theorem, we covered operations such as union or intersection. What if we want to compose binary relations, or project a set of pairs onto the first coordinate? Instead of treating such operations on a case by case basis, we give a generic result, which deals with every operation that can be defined in the language of set theory. This approach is similar to the Equivariance Principle from Theorem 10.3. There we showed that any relation definable in the language of set theory is equivariant, here we use a similar approach to show that such relations can be computed. The difference is that here, unlike in the Equivariance Principle, we start with some set X, and we are only allowed to talk about elements that are below this set, which limits the expressiveness, but makes it possible to compute things

Definition 10.13 (Downset)**.** Let $\in^*$ be the reflexive transitive closure of the membership relation, i.e. $x \in^* X$ if and only if x is equal to X, or an element of X, or an element of an element of X, and so on. Define the *downset* of a set of X to be the logical structure which has universe

$$X{\downarrow} \stackrel{\text{def}}{=} \{ \, x \mid \ x \in^* X \ \}$$

and which is equipped with one binary relation $\in$.

Many constructions can be described using first-order logic on the downset. The following example discusses projection from pairs. Other examples, such as composition of binary relations, can be found in the exercises.

Example 79. [Projection in the downset] The projection function

$$X \times Y \to X$$

can be defined by a first-order formula interpreted in the downset of $X \times Y$. Recall that pairs are defined using Kuratowski pairing

$$(x, y) = \{\{x\}, \{x, y\}\}.$$

Note that if $x \in X$ and $y \in Y$, then

$$x \quad y \quad \{x\} \quad \{x, y\}$$

are all in the downset of $X \times Y$. The point of Kuratowski pairing is that pairing and projections can be done in the language of set theory. The following formula expresses that p is the (set representing the) ordered pair (x, y):

$$\forall z\ z \in p \ \Rightarrow\ \Big(\overbrace{z = \{x\}}^{\substack{x \text{ is the unique} \\ \text{element of } z}} \ \vee \ \overbrace{z = \{x\} \cup \{y\}}^{\substack{z \text{ is the smallest set that} \\ \text{contains } \{x\} \text{ and } \{y\}}} \Big). \tag{10.3}$$

The projection of $X \times Y$ to the first coordinate is the set of elements x in the downset of $X \times Y$ that satisfy the following formula $\varphi(x)$:

$$\exists y \quad \overbrace{y \in X \times Y}^{\substack{X \times Y \text{ is the only} \\ \text{element of the} \\ \text{downset that} \\ \text{does not belong to} \\ \text{any other element}}} \ \wedge \ \overbrace{p = (x, y)}^{\text{expressed in (10.3)}} \ . \tag{10.4}$$

□

The following lemma shows that one can compute in polynomial time all operations which can be formalised using first-order logic over the downset, such as the projection operation given in the above example.

Lemma 10.14 (Downset Lemma)**.** *Consider an oligomorphic structure $\mathbb{A}$, together with some atom representation. There is an algorithm which does the following.*

- **Input.** *A representation of a set with atoms X, together with a first-order formula $\varphi(x_1, \ldots, x_k)$ using a binary relation $\in$.*
- **Output.** *A representation of the set*

$$\{\ (x_1, \ldots, x_k) \mid\ X\downarrow \models \varphi(x_1, \ldots, x_k)\ \}.$$

If the number of variables used by φ and its subformulas of arguments is fixed, the running time of the algorithm is polynomial.

Proof. The approach is very similar to the one used in the proof of Theorem 10.11, i.e. we show that the problem can be reduced to the first-order theory of the atoms. The appropriate version of the Symbol Pushing Lemma is the following claim.

Claim 10.15. *Let $\varphi(x_1, \ldots, x_k)$ be a first-order formula which uses membership $\in$ only, and let $\alpha_0, \ldots \alpha_k$ be set builder expressions. One can compute in polynomial time a first-order formula ψ, over the vocabulary of the atoms, such that*

$$\mathbb{A} \models \psi(\bar{a}_0 \cdots \bar{a}_k) \quad \textit{iff} \quad \underbrace{\alpha_0(\bar{a})}_{X}\downarrow \models \varphi(\underbrace{\alpha_1(\bar{a}_1)}_{x_1}, \ldots, \underbrace{\alpha_k(\bar{a}_k)}_{x_k}).$$

Proof. Induction on the structure of φ. In the induction basis φ is an atomic predicate, i.e. a membership predicate of the form $x_i \in x_j$ or $x_i = x_j$. For this case, we use the Symbol Pushing Lemma.

In the induction step, the case of Boolean combinations $\wedge, \vee$ and $\neg$ is immediate. The only interesting case is quantification. By DeMorgan's laws, we only need to treat the existential quantifier. Suppose that φ is a formula with $n+1$ free variables for which we have already proved the claim, and we now want to prove the claim for the formula obtained from φ by existentially quantifying the last variable. This means that, given set build expressions $\alpha_0, \ldots, \alpha_k$, we want to compute a formula ψ such that

$$\mathbb{A} \models \psi(\bar{a}_0 \cdots \bar{a}_k) \quad \text{iff} \quad \alpha_0(\bar{a})\!\downarrow \models \exists x_{n+1}\ \varphi(\alpha(\bar{a}_1), \ldots, \alpha(\bar{a}_k), x_{n+1}).$$

The only difficulty is that we do not know the set builder expression α_{n+1} used in the representation of x_{n+1}. The solution to this difficulty is that this expression must be a sub-expression of α, since every element of the downset of α_0 is obtained by taking some sub-expression of α_0 and some values of its free variables. (Here a sub-expression is defined in the natural way, it is an expression from which α_0 is constructed.) Therefore, we can use a finite disjunction over all possible sub-expressions of α_0 and an existential quantification over the free variables of this sub-expression to quantify over the set x_{n+1}. □

Using the claim, we complete the proof of the lemma. Assuming that the set X is represented by $\alpha_0(\bar{a}_0)$, the expression defining the output set in the statement of the lemma is

$$\bigcup_{\alpha_1,\ldots,\alpha_k} \{\, (\alpha_1, \ldots, \alpha_k) \mid \text{for } \bar{x}_1 \cdots \bar{x}_k \text{ such that } \underbrace{\psi_{\bar{a}}(\bar{x}_0 \cdots \bar{x}_k)}_{\substack{\text{formula from the above claim} \\ \text{applied to } \alpha_0, \ldots, \alpha_k}} \,\},$$

where the union ranges over all choices of n sub-expressions of α. Once we have fixed the number k of free variables, this construction is polynomial. □

Exercises

Exercise 156. Assume that the atoms are oligomorphism. Let X be a set with atoms and let $\bar{a}$ be a tuple of atoms. For an element $x \in X$, define its $\bar{a}$-orbit to be the set

$$\{\, \pi(x) \mid \ \pi \text{ is an atom automorphism that fixes } \bar{a} \ \,\}.$$

In the case of the empty tuple of atoms, we get the usual notion of orbit. Show that if a set X is orbit-finite, i.e. it has finitely many orbits for the empty tuple $\bar{a}$, then it has finitely many $\bar{a}$-orbits for every tuple of atoms $\bar{a}$.

Exercise 157. Show that if the atoms have at least two elements, then the following problem is PSPACE hard: given two set builder expressions where all guards are quantifier-free, decide if they represent the same set.

Exercise 158. Let $\mathbb{A}$ be a structure with at least two elements. Consider two measures of size for first-order formulas:

1. circuit size (number of distinct subformulas);

2. tree size (number of nodes in the syntax tree).

Circuit size is the notion of size we use in this book. Show that for every formula of first-order logic φ one can compute an equivalent formula whose tree size is polynomial in the circuit size of φ.

Exercise 159. Use the Third Symbol Pushing Lemma to show that composition of binary relations (given by set builder expressions) can be computed in polynomial time.

Exercise 160. Assume the atoms are Presburger arithmetic $(\mathbb{N}, +)$. For which $k \in \{0, 1, \ldots\}$ is the following problem decidable:

- **Input.** A set builder expression representing $R \subseteq \mathbb{A}^{2k}$ and $x, y \in \mathbb{A}^k$.
- **Output.** Is (x, y) in the transitive closure of R, where R is viewed as a binary relation on k-tuples?

Chapter 11

While programs with atoms

So far, when we presented algorithms for problems on orbit-finite sets, such as graph reachability or automaton nonemptiness, we needed to be sensitive to the representation. Representations that we have discussed so far include: pof sets, subquotiented pof sets, and set builder expressions. In this chapter, we present a programming language, called *while programs with atoms*, which works directly with representable sets, and not with their representations. Using the programming language, we can write programs without going into the details of the representation.

The basic idea behind the language is that its variables store atoms or representable sets, and it can loop over elements of sets. To illustrate the programming language, we use our favourite example, which is graph reachability.

Example 80. The algorithm for graph reachability that was discussed in Theorems 1.9 and 5.10 can be implemented using the following code.

```
def reach(V, E, S):
  # the input is a graph with a designated set of source vertices
  # the output will be the reachable vertices
  Reach = S
  New = ∅
  while New ≠ Reach:
      Reach = New
      for v in Reach:
          for w in V:
              if (v, w) ∈ E:
                  New = New ∪ {w}
  return Reach
```

□

This program in the above example is simply a naive implementation of breadth-first search. For inputs that are finite, it is easy to see how the program works; the only difficulty is finding an appropriate order of execution for the for loops. We will show that a proper resolution to this difficulty, namely running the for loops in parallel, will allow us to execute the program also for inputs that are representable, but not necessarily finite.

The language simplifies a lot of the bookkeeping work involved with sets represented by set builder expressions, but it does not magically solve all problems. The programmer still has work to do. For example, in the graph reachability program described above, the programmer needs to justify that the `while` loop will do a finite number of iterations for every input, which will be a consequence of oligomorphism.

In Section 11.1, we define the programming language and explain its semantics. In Section 11.2, we show that the outputs of programs can be computed in finite time, despite the ostensibly infinite `for` loops. We also show that the programming language is complete, in the sense that it implements all operations on representable sets that can be implemented using Turing machines which use representations.

11.1 While programs with atoms

In this section, we introduce the programming language[1], explain its semantics, and give example programs. We assume that the atoms are effectively oligomorphic, although some definitions and results would make sense without assuming effectivity, or even oligomorphism.

Syntax. Fix some countably infinite set of variable names. The programming language is untyped: every variable stores a set with atoms (as in Definition 10.1) or an atom[2]. When actually running the programs, we will only care about sets with atoms that can be represented using set builder expressions, see Theorem 10.7. As we will show, if a program starts execution in a state where all variables store such representable sets, then it will be unable to create sets that are not representable.

The programming language does not have types for Booleans, or integers, or lists. As discussed in Chapter 10, such data structures can be encoded using sets. The following example shows how to encode Booleans.

Example 81. We can encode the Boolean values as

$$\text{false} = \emptyset \qquad \text{true} = \{\emptyset\}.$$

These values can be loaded into variables, and the Boolean operations can be implemented by using case distinction. Therefore, we can assume that the programming language has Boolean variables. □

Example 82. We represent natural numbers using Von Neumann encoding, where 0 is the empty set and $n + 1$ is the set $\{0, \ldots, n\}$. Using this encoding, operations on natural numbers such as addition or multiplication can be programmed in the language. For example, addition `x+y` is implemented as follows:

[1] The programming language is based on the language from Bojańczyk and Toruńczyk (2012), which is an imperative version of the functional programming language Bojańczyk et al. (2012). The functional language was further developed in Moerman et al. (2017), and an implementation can be found at `https://www.mimuw.edu.pl/~szynwelski/nlambda/`. The imperative language further developed in Kopczyński and Toruńczyk (2016) and Kopczyński and Toruńczyk (2017), including a generalisation to atom structures which are not necessarily oligomorphic (this generalisation is used in this section), and an implementation `https://www.mimuw.edu.pl/~erykk/lois`.

[2] In this sense, we build on set based programming languages such as SETL.

```
def add(x,y):
  z = x
  i = ∅
  while i ≠ y:
    z = {z} ∪ z # increment z
    i = {i} ∪ i # increment i
  return z
```

□

A usable variant of the programming language would have more features, such as Booleans, integers, and recursive functions. We present the language using a minimal syntax, concentrating on the aspects that deal with atoms.

Definition 11.1 (Syntax of while programs with atoms)**.** Fix a structure $\mathbb{A}$. A *while program with atoms over* $\mathbb{A}$ is constructed using the following syntax:

- *Assignment.* The following assignments are programs:

  $\underbrace{\mathtt{x} = \emptyset}_{\text{load the empty set}}$ $\underbrace{\mathtt{x} = \mathbb{A}}_{\text{load the set of atoms into x}}$ $\underbrace{\mathtt{x} = R}_{\text{load a relation } R \text{ from the atom structure into x}}$ $\underbrace{\mathtt{x} = \mathtt{y} \cup \mathtt{z}}_{\text{set union}}$ $\underbrace{\mathtt{x} = \{\mathtt{y}\}}_{\text{singleton}}.$

  The relation R is taken from the vocabulary of the atoms[3]. In set union, the instruction has no effect if one of the variables y or z stores an atom.

- *Sequential composition.* If I and J are already defined programs, then the sequential composition I;J is also a program, which first executes I and then J. In the code snippets we use Python syntax, so we omit semicolons.

- *Control flow.* Suppose that x and y are variables, I is an already defined program, and δ is one of $\in, \subseteq, \subsetneq$ or ==. (We use the double equality sign, since single equality describes an assignment.) Then

  ```
  if x δ y: I     while x δ y: I     for x in y: I
  ```

  are programs. The semantics for if and while are as expected. In case of for, the idea is that the instruction I is executed, in parallel, with one thread for every element x of the set y, and the results of the threads are aggregated using set union, see below for a more detailed description.

Semantics. We now define an operational semantics for the language. A *program state* over a finite set of variables X is defined to be a function which maps each variable to either an atom or a representable set. The semantics of a while program I is a partial function, denoted by $\gamma \mapsto \gamma\mathtt{I}$, which maps one program state to another, with the variables of the program states being those that appear in I. The function

[3]The above presentation is suited for atoms with a finite vocabulary. If the vocabulary is countably infinite, one should add an operation "set x to the n-th relation/function in the vocabulary, where n is the Von Neumann numeral stored in variable y". The expressive power of the language depends on the enumeration of the vocabulary.

is partial because its output is defined if and only if the program terminates. In the discussion below, we also talk about the running time of a program, which intuitively stands for the maximal number of sequentially executed instructions, with running times for parallel threads being aggregated using maximum.

We only explain the semantics of programs of the form

```
for x in y: I,
```

the other constructions being handled in the standard way. Suppose that we execute the loop above, starting in some program state γ. If the variable `y` stores an atom or the empty set, then the loop does nothing and the starting and ending program states are the same. Assume otherwise, that `y` stores a nonempty representable set. We first explain when the loop terminates, and then we explain how it affects the program state.

- *When the loop terminates.* For an element x of the set stored in variable `y`, define $\gamma[\mathtt{x} := x]$ to be the program state obtained from γ by setting variable `x` to x. Depending on the choice of x, the program `I` might not terminate from the program state $\gamma[\mathtt{x} := x]$, or it might terminate in a finite number of steps $n_x \in \{1, 2, \ldots\}$ which depends on x. If `I` does not terminate for some x, or the numbers n_x are unbounded, then the `for` loop does not terminate. Otherwise, if `I` terminates for all x in bounded time, then the `for` loop itself also terminates, and its running time is one plus the maximal number n_x.

- *What is the program state after the loop.* The idea is to aggregate the resulting program states into a single one, using set union. More formally, for a set Γ of program states over the same set of variables, define its *aggregation* to be the program state where, for every variable `x`, the stored value is:

  1. x if all program states in Γ have the same value x in `x`; or otherwise
  2. the union of all sets stored in variable `x` by program states from Γ.

  Note that in the second case, where union is used, some program state from Γ might store an atom in `x`. If that happens, the atom will be lost in the aggregation because an atom has no elements, and therefore it is ignored when taking a union (this is also why we have the special case in item 1, since otherwise variables storing atoms would be lost). Using this aggregation, define the result of evaluating the for loop to be the aggregation of the set

  $$\{(\gamma[\mathtt{x} := x])\mathtt{I} : x \text{ is an element of the set stored in variable } \mathtt{y}\}.$$

This completes the definition of the semantics of the for loop.

The aggregation function that we use might seem arbitrary. Why not use intersection instead of union, or some other commutative operation on sets? As we will explain in Section 11.2, the semantics described above lead to a language that is computationally complete, and therefore other (computable) choices of aggregation will necessarily give the same (or less) computational power.

Example programs

The rest of Section 11.1 is devoted to examples of while programs with atoms. In these examples, we will frequently describe programs with inputs and outputs, which define functions. Such functions are not formally part of our language, but they can be modelled by a program that has some designated input variables and a designated output variable for the return value. Therefore, calling the function can be simulated by inlining the appropriate code. Recursive functions (whose implementation would require a bit more than simple inlining of code) could also be implemented, but we do not use them in our examples.

Example 83. [Pairing and unpairing] As we have mentioned at the beginning of this section, the programming language does not have specialised types for Booleans or integers, but these can be (inefficiently) implemented using sets. Similarly, pairing can be implemented using sets, via the Kuratowski paring construction from Example 75:

```
1 def pair(x,y):
2   return  {x,{x,y}}
```

Formally speaking, the in line 2 of the above program needs to be constructed in several steps, using the basic operations of our programming language:

```
q = {x}
p = {y}
p = p ∪ q
p = {p}
p = p ∪ q
```

In the programs below, omit such details, and assume that the reader knows how to construct sets using the basic operations of the programming language. Also, we write `(x,y)` instead of explicitly calling the pairing function defined above.

A pairing function is only useful if it is accompanied by an unpairing function. Here is a program which projects a pair `p` into the singleton of its first coordinate, and returns $\emptyset$ if its argument is not a Kuratowski pair of sets.

```
1 def first(p):
2   ret = ∅
3   for a in p:
4     for x in a:
5       for y in p:
6         if p == (x,y):
7             ret = {x}
8   return ret
```

In Example 86 we show how to get rid of the singleton. The test in line 6 is actually syntactic sugar for loading `(x,y)` into an auxiliary variable and then checking if `p` is equal to that auxiliary variable. We use such syntactic sugar freely in the programs below. □

The following example shows how to implement the semantics of set builder expressions using while programs with atoms.

Example 84. The atoms are $(\mathbb{Q}, <)$. Consider the set of bounded open intervals:

$$X = \{\{z \mid \text{ for } z \in \mathbb{A} \text{ such that } x < z < y\} \mid \text{ for } x, y \in \mathbb{A} \text{ such that } x < y\}$$

Here is a program which loads this set into variable X.

```
X = ∅
for x in 𝔸:
  for y in 𝔸:
    if x < y:
      Z := ∅
      for z in 𝔸:
        if x < z < y:
          Z := Z ∪ {z}
      X = X ∪ {Z}
```

In the program above, the test `x < y` is syntactic sugar for first loading the order relation < into some auxiliary variable, and then checking if that relation contains the pair `(x,y)`. Using the same idea, any set builder expression can be loaded into a variable. □

Example 85. The following program inputs a nonzero natural number in variable `d` and outputs the set $\mathbb{A}^d$ in variable `Y`. The number `d` and operations on it are implemented using Von Neumann encoding from Example 82.

```
def tuples(d):
  Y := 𝔸
  while d > 1:
    d = d - 1
    X = Y
    Y = ∅
    for a in 𝔸:
      for x in X:
        Y = Y ∪ {(a,x)}
  return Y
```

Using similar ideas, one can write a program which inputs a representation of a first-order formula $\varphi(x_1, \ldots, x_n)$ and outputs the set of atom tuples $(a_1, \ldots, a_n)$ which satisfies the formula. □

Example 86. [Desingleton] The following program implements the mapping $\{x\} \mapsto x$. More precisely, if the variable `x` stores a singleton $\{x\}$, then the function will return x, otherwise it will return the empty set.

```
def desingleton(x):
  result = ∅
  for y in x:
    result = y
  return result
```

Using this function, we can improve the projection function from Example 83 so that it returns the actual value of the first coordinate. □

Example 87. [Reachability] Consider the program for graph reachability from Example 80 at the beginning of this chapter. As argued in Section 7.3, if the atoms are oligomorphic, and the input is orbit-finite, then the while loop does finitely many iterations for every input, and therefore the program always terminates. □

Example 88. [Automaton emptiness] Using pairs and projections, we can extend the language with a pattern-matching construction

```
for (x,y) in X: I
```

which ranges over all pairs in `X`. We use a similar convention for tuples of length greater than two. Pattern-matching is convenient if we want to compute the one-step reachability relation in a nondeterministic automaton:

```
E = ∅
for (p,a,q) in delta:
  E = E ∪ {(p,q)}
```

After computing this relation, we can use the program for graph reachability from Example 45 to check if an automaton is nonempty. □

Example 89. [Automaton minimisation] The following program minimises a deterministic orbit-finite automaton. The program is a standard implementation of Moore's minimisation algorithm. We first compute the reachable states, and then we compute the pairs of states which can be distinguished by some input string. Next, the states of the minimal automaton are defined to be equivalence classes of indistinguishability. (It is useful here that the programming language uses sets, and thus equivalence classes are legitimate values of variables.) The main point of writing the program down is so that the reader can follow the code and see that it also works with hereditarily orbit-finite sets. The reason is that – similarly to the program for graph reachability – each saturation process (reachable states, or distinguishable pairs) terminates in finite time, because it adds whole orbits in each step.

```
def minimize(Σ,Q,q0,delta,F):

  # keep only reachable states
  old = ∅
  Q = {q0}
  while old ≠ Q:
   old = Q
   for (p,a,q) in delta:
     if p ∈ Q:
       Q = Q ∪ {q}

  # compute distinguishable pairs of states
  old = ∅
  dist = (F × (Q-F)) ∪ ((Q-F) × F)
  while old ≠ dist:
    old = dist
    for (p1,a1,q1) in delta:
      for (p2,a2,q2) in delta:
        if a1==a2 and (q1,q2) ∈ dist:
```

```
          dist = dist ∪ {(p1,p2)}

  # new states are equivalence classes of non-distinguishability
  newQ = ∅
  for q in Q:
    # the new state will be the equivalence class of q
    newq = ∅
    for p in Q:
      if (p,q) ∉ dist:
        newq = newq ∪ {p}
    # add the new state to the set of new states
    newQ = newQ ∪ {newq}
    # note that the same state could be added several times
    # but sets do ignore duplicates, so this is ok

  # compute new initial state
  newq0 = ∅
  for q in Q:
    if (q0,q) ∉ dist:
      newq0 = newq0 ∪ {q}

  # compute new transition relation (which is a function)
  newdelta = ∅
  for newp in newQ:
    for newq in newQ:
      for (p,a,q) in delta
        if p ∈ newp and q ∈ newq:
          newdelta = newdelta ∪ {(newp,a,newq)}

  # compute new final states
  newF = ∅
  for newq in newQ:
    for q in newq:
      if q ∈ F:
        newF = newF ∪ {newq}

  return (A,newQ,newq0,newdelta,newF)
```

□

We finish the section with an example of a program where a termination proof requires a bit of effort, beyond the mere observation that each iteration of a while loop adds a whole orbit.

Example 90. Call a semigroup S aperiodic if

$$\exists n \in \{1, 2, \ldots\}\ \forall s \in S\ \overbrace{s \cdots s}^{n \text{ times}} = \overbrace{s \cdots s}^{n+1 \text{ times}} .$$

Assume that the atoms are oligomorphic. The following program inputs a semigroup, given as a set S together with a multiplication operation, and checks if it is aperiodic.

```
def aperiodic(S, mult):
  counterexamples := ∅
  for s in S:
    X = ∅
```

```
5     power = s
6     while power ∉ X:
7       X := X ∪ {power}
8       power = mult(power,s)
9     if mult(power,s) ≠ power:
10      counterexamples = counterexamples ∪ {s}
```

Formally speaking, the multiplication operation is given as a set of triples of the form (first input, second input, output). Therefore, when we apply the multiplication operation in lines 8 and 9, we are really executing a subroutine which searches all triples in the multiplication operation, and finds the one that has the appropriate output.

In the program, the set `X` is used to collect consecutive powers $\mathtt{s}, \mathtt{s}^2, \mathtt{s}^3, \ldots$. To prove termination, one needs to show that this set is always finite, even if the semigroup in question is not aperiodic. Furthermore, there is a fixed upper bound on the size of such sets. Let us argue that this is indeed the case. Every power of `s` is supported by whatever supports both the element `s` and the product operation in the semigroup. Like all sets in the programming language, the underlying set of the semigroup is orbit-finite. In an orbit-finite set, there are finitely many elements with a given support, as shown in Exercise 105. It follows that for every `s`, the set of its powers is finite. Furthermore, there is a common upper bound on the size of these sets, because if two elements are in the same orbit, then the number of their powers is the same. □

11.2 Computational completeness of while programs

What is a computable function f that inputs and outputs representable sets? Similarly to the situation in Theorem 9.3, there are two possible definitions. We can use a while program with atoms, which has a designated interface variable, and which has the property that if the designated interface variable is set to a representable set X, then the program terminates and stores $f(X)$ in the interface variable. (This is the formalisation of functions that we have already used in the previous examples.) Alternatively, we can compute the function on representations, using the standard notion of atom-less computation (one formalisation is an always terminating Turing machine that has a write-only output tape). The following result shows that the two definitions are equivalent. In particular, the second definition does not depend on the choice of representation, since the representation is not used in the first definition.

Theorem 11.2 (Computational completeness of while programs[4])**.** *Fix an oligomorphic atom structure, together with an atom representation. For every equivariant function*

$$f : \textit{representable sets} \to \textit{representable sets}$$

the following conditions are equivalent:

1. *given a representation of X, one can compute a representation of $f(X)$;*

[4]The theorem is based on (Bojańczyk and Toruńczyk, 2018, Theorem 3.9), which in turn is based on (Bojańczyk et al., 2013a, Theorems IV.1 and IV.2).

2. *there is a while program with atoms that inputs X and outputs $f(X)$.*

The above theorem is formulated for total functions, but it would also work for partial functions, where the programs are do not terminate when the output is undefined. This extension is left as an exercise for the reader.

From representations to while programs. We begin by proving that if a function is computable under representations, then it can be computed by a while program with atoms. The main observation is that a while program with atoms can reverse the representation, as stated in the following lemma. In the lemma, the representations are seen as strings over a two letter alphabet. Such strings can be inputs or outputs of while programs, since a string can be seen as a representable set, using the Booleans from Example 81 as the alphabet, and the pairing construction from Example 83 to implement lists.

Lemma 11.3. *There is a while program with atoms which inputs a representable set X, and outputs a string in 2^* that represents some set in the same orbit as X.*

Proof. We begin by observing that a while program with atoms can simulate any Turing machine without atoms. This is because representable sets can be used to store lists over a finite alphabet, and therefore also configurations of the Turing machine. The while loop can be used to simulate the computation of the Turing machine.

Next, we observe that the semantics of set builder expressions can be implemented by a while program with atoms. This means that there is a while program which inputs a set-builder expression α and a valuation $\bar{a}$ of its free variables, and returns the representable set $\alpha(\bar{a})$. The set-builder expression is given by an atom-less string in 2^*, while the atoms are given directly (not as representations). This program uses the same principle as was illustrated in Example 84, except that instead of constructing the set represented by a fixed expression, the expression is given on input.

Let us use the above observations to prove the lemma. Suppose that the input is a representable set X. The program enumerates through all set builder expressions, and for each such expression α, it does the following:

- Let d be the number of free variables in α. The program computes the set $\mathbb{A}^d$, using the construction from Example 85. Next, using Claim 9.8, the program computes a list of first-order formulas $\varphi_1, \ldots, \varphi_n$ that define the partition of $\mathbb{A}^d$ into equivariant orbits. For each of these orbits, the program uses a for loop over $\mathbb{A}^d$ to check if there is some tuple $\bar{a}$ in this orbit such that $\alpha(\bar{a}) = x$. If so, the program computes a representation of some tuple in this orbit, and terminates, with the return value being the set builder expression α together with the representation of the atom tuple. Otherwise, the set builder expression α is discarded, and the program continues with the next set builder expression in the enumeration.

By definition of representable sets, this program must eventually find a representation, and so this procedure is bound to terminate. □

Suppose that we are given on input a representable set X. Thanks to the above lemma, we can compute a representation $\alpha(\bar{a})$ of some in the same orbit as X. Since f is computable under representations, we can compute a representation $\beta(\bar{b})$ of $f(\alpha(\bar{a}))$. The while program now computes the set

$$Y = \{\ \bar{c}\bar{d} \mid\ \ X = \alpha(\bar{c}) \text{ and } \bar{a}\bar{b} \text{ is in the same orbit as } \bar{c}\bar{d}\ \ \}.$$

Claim 11.4. *Every $\bar{c}\bar{d} \in Y$ satisfies $f(X) = \beta(\bar{d})$.*

Proof. By definition of the set Y, there is some atom automorphism π that maps $\bar{a}\bar{b}$ to $\bar{c}\bar{d}$. This automorphism maps $\alpha(\bar{a})$ to X. Therefore, by equivariance, it maps

$$f(\alpha(\bar{a})) = \beta(\bar{b})$$

to $f(X)$. Since we know that π maps $\bar{b}$ to $\bar{d}$, the claim follows. □

Thanks to the above claim, if we apply the function

$$\bar{c}\bar{d} \quad \mapsto \quad \beta(\bar{d})$$

to the above set, then we get the singleton $\{f(X)\}$. This application can be computed using a for loop. Finally, we can apply the desingleton function from Example 86 to get the desired result.

From while programs to representations. We will now show that given the source code of a while program with atoms and a representation of the initial program state, we can compute a representation of the final program state (assuming that the program terminates). This will give the remaining implication in Theorem 11.2.

In the semantics of while programs, infinitely many threads are executed in parallel, and therefore it will be easier to work with sets of program states instead of individual program states. A program state can be viewed as a finite tuple of atoms or representable sets. Since representable sets are closed under taking tuples, follows that a program state itself can be seen as a representable set. This is the view taken in the following lemma. By applying the lemma to a singleton set of program states, we get the remaining implication in Theorem 11.2.

Lemma 11.5. *There is an atom-less Turing machine which inputs:*

- *the source code of a while program with atoms* $\mathtt{I}$*;*
- *a representation of set* Γ *of program states over the variables of* $\mathtt{I}$*;*

and does the following:

- *if there is some* $n \in \mathbb{N}$ *such that* $\mathtt{I}$ *terminates in at most* n *steps for all program states in* Γ*, then the Turing machine halts and outputs a set builder expression representing all possible input/output pairs* $\{(\gamma, \gamma\mathtt{I}) : \gamma \in \Gamma\}$*;*
- *otherwise, the Turing machine does not terminate.*

Proof. Structural induction on the source code of the program $\mathtt{I}$. The assignments are immediate, so we only do the proof for the other constructions.

- *Sequential composition.* Given a representation of a set Γ of program states, we want to compute the set of input/output pairs for a sequential composition of two programs $\mathtt{I}_1$ and $\mathtt{I}_2$:

$$\{\ (\gamma, \gamma(\mathtt{I}_1; \mathtt{I}_2)) \mid\ \ \gamma \in \Gamma\ \ \}. \tag{11.1}$$

  By induction, compute a representation of the set

$$R_1 = \{\ (\gamma, \gamma(\mathtt{I}_1)) \mid\ \ \gamma \in \Gamma\ \ \}.$$

  As we have shown in Example 79, we can define the projection onto the second coordinate in the set structure. Therefore, thanks to the Set Structure Lemma, we can compute a representation of the set

$$\Delta = \{\ \gamma\mathtt{I}_1 \mid\ \ \gamma \in \Gamma\ \ \}$$

  Using the induction assumption, we can compute a representation of

$$R_2 = \{\ (\delta, \delta(\mathtt{I}_2)) \mid\ \ \delta \in \Delta\ \ \}.$$

  The relational composition $R_1 \circ R_2$ can be defined in the set structure of (R_1, R_2), and by another application of the Set Structure Lemma we get a representation of the desired set from (11.1).

- *If.* We want to compute a representation of the set

$$\{\ (\gamma, \gamma(\mathtt{if\ x}\ \delta\ \mathtt{y\ then\ I}) \mid\ \ \gamma \in \Gamma\ \ \} \tag{11.2}$$

  where δ is one of $=, \subseteq, \subsetneq, \in$. Consider the set of input program states which satisfy the conditional

$$\Gamma_{\text{true}} \quad \stackrel{\text{def}}{=} \quad \{\ \gamma \in \Gamma \mid\ \ \gamma(\mathtt{x})\ \delta\ \gamma(\mathtt{y})\ \ \}.$$

  This set can be defined using the set structure of Γ, and therefore a representation of it can be computed thanks to the Set Structure Lemma. The set in (11.2) is equal to

$$\{\ (\gamma, \gamma\mathtt{I}) \mid\ \ \gamma \in \Gamma_{\text{true}}\ \ \} \qquad \cup \qquad \{\ (\gamma, \gamma) \mid\ \ \gamma \in \Gamma \setminus \Gamma_{\text{true}}\ \ \}.$$

  By the Symbol Pushing Lemma, the constraints

$$\gamma \in \Gamma_{\text{true}} \qquad \text{and} \qquad \gamma \in \Gamma \setminus \Gamma_{\text{true}}$$

  can be defined by first-order formulas which talk about the free variables of the set builder expressions γ, Γ and Γ_{true}. Using these formulas as guards, we get a set builder expression for the desired set (11.2).

- *While loop.* Consider a definable program of the form

  ```
  while x = y do J.
  ```

  Let Γ be a set of program states on which we want to execute the above program. For $n \in \{0, 1, \ldots\}$, let $\Gamma_n \subseteq \Gamma$ be those program states which take at most n iterations to finish the while loop. Using the same approach as in the previous item, for each n we compute a representation of Γ_n and the semantics of the while loop with domain restricted to Γ_n. We try all n until $\Gamma_n = \Gamma$. This equality can be checked using Theorem 10.11. (This is also the only place where we use the decidability of the atom structure). If no such n exists, then the interpreter does not terminate. Otherwise, if such n exists, then we can eliminate the external while loop and replace it by sequential composition of n copies of the program J, which can be treated using the previous items.

- *For loop.* Our goal is to compute a set builder expression representing

  $$\{\ (\gamma, \gamma(\underbrace{\texttt{for x in X do J}}_{\texttt{I}}))\ |\ \text{ for } \gamma \in \Gamma\ \}. \tag{11.3}$$

  It is enough to show that for every program variable y, we can compute a set builder expression representing

  $$\{\ (\gamma, \text{value of variable } y \text{ in } \gamma\texttt{I})\ |\ \gamma \in \Gamma\ \}, \tag{11.4}$$

  since the resulting sets of pairs can be rearranged using symbol pushing to get relation on program states from (11.3).

  Fix a variable y. Using the Set Structure Lemma, compute a representation of

  $$\Delta = \{\ \gamma[\mathtt{x} := x]\ |\ \gamma \in \Gamma \text{ and } x \in \gamma(\mathtt{y})\ \}.$$

  Using the induction assumption, compute a representation of

  $$R = \{\ (\delta,\ \text{value of variable y in } \delta\texttt{J})\ |\ \delta \in \Delta\ \}.$$

  Applying the Set Structure Lemma to (Δ, R), compute a representation of

  $$S\ = \{\ (\gamma, \text{value of variable y in } (\gamma[\mathtt{x} := x])\texttt{J})\ |\ \gamma \in \Gamma \text{ and } x \in \gamma(\mathtt{x})\ \}.$$

  We now need to aggregate, for each program state $\gamma \in \Gamma$, all the values y with $(\gamma, y) \in S$. Recall that aggregation uses two cases: identity for program states γ which have a unique y, and set union for the remaining program states. Define $\Gamma_\cup \subseteq \Gamma$ to the program states where set union is used for aggregation:

  $$\Gamma_\cup = \{\ \gamma \in \Gamma\ |\ \text{ there are at least two } y \text{ with } (\gamma, y) \in S\ \}.$$

  A representation this set can be computed using the Set Structure Lemma. By definition, the set (11.4) is

  $$\{\ (\gamma, y)\ |\ \gamma \in \Gamma \setminus \Gamma_\cup \text{ and } (\gamma, y) \in S\ \}\quad \cup\quad \{\ (\gamma, \{\ y\ |\ (\gamma, y) \in S\ \})\ |\ \gamma \in \Gamma_\cup\ \}$$

As in the case of the if statements, the guards in the above sets can be seen as first-order formulas, by using the Symbol Pushing Lemma. This means that an expression for the first summand can be computed from an expression for S. For the second summand, we need to show that set builder expressions admit a form of Currying, where a representation of a binary relation $S \subseteq A \times B$ is converted into a representation of the corresponding function $A \to \mathsf{P}B$. This construction is presented in Exercise 161.

□

Exercises

Exercise 161. Show that given a set builder expression without parameters representing a set $R \subseteq X \times Y$, one can compute a set builder expression without parameters representing the set

$$S = \{(x, \{y : (x, y) \in R\}) : x \in X\}.$$

Exercise 162. Consider while programs over the order atoms $(\mathbb{Q}, <)$. Suppose that we remove the order relation $x < y$, but we add a test for "x and y are sets are in the same orbit". Is the new programming language equivalent to the previous one?

Exercise 163. Suppose that we extend while programs with the ability to refer to some atom constants, i.e. these can appear in the source code of the program. Show that if such a program computes a function

$$f : \text{representable sets} \to \text{representable sets},$$

and this function is equivariant, then the same function can be computed by a while program that does not refer to atom constants.

Chapter 12

Fixed dimension polynomial time

In the previous chapter, we discussed computation on representable sets. In this chapter, we propose a notion of computation that is "polynomial time". We are mainly interested in the equality atoms.

The proposed notion will be designed so that many algorithms which are polynomial time on finite sets, such as graph reachability, will continue to be "polynomial time" on representable sets. A minimal requirement is that the notion is general enough to describe equality tests, such as the following program.

```
if x == y:
    print ``equal''
```

Another requirement for the definition is that, for inputs that do not use atoms, such as bit strings, exactly the usual notion of polynomial time is recovered.

The first idea is to consider the programs that work with representations, as in the first item in Theorem 11.2, and which run in polynomial time. This is a bad idea, since it is too restrictive. The issue is that the equality tests, as discussed in the previous paragraph, will not be covered. The problem arises already for the simplest kind of equality tests, namely equality with the empty set. For example, if we want to check if

$$\{\emptyset \mid \text{ for } \bar{x} \in \mathbb{A}^d \text{ such that } \varphi(\bar{x})\}$$

is equal to the empty set, then we need to check if the formula φ is satisfiable. Already in the equality atoms, this problem is PSpace-complete.

The underlying problem is that the first-order theory of the atoms is computationally hard (PSpace hard if there are at least two atoms, see Exercise 164). The solution proposed in this chapter is to identify a source of this hardness, namely the "dimension" of representable sets, which is – roughly speaking – the number of bound variables. The proposed notion of tractability avoids computational hardness, by using algorithms whose running time is polynomial in the size of the input, but the

degree of the polynomial is allowed to depend on the dimension. Hence the name: fixed dimension polynomial time.

12.1 Dimension and size of representable sets

As mentioned above, the idea behind our complexity class is that the running time of the algorithm is polynomial in the "size" of the input, but the polynomial (and its degree) can depend on its "dimension". Roughly speaking, the size of a set is the number of orbits, and the dimension is the maximal size of supports. These are very rough approximations, and the formal definitions will need to take more parameters into account. Before stating these definitions, we discuss – on a simple but central example – how fixing the dimension can improve certain orbit counts from exponential to polynomial.

The central example is the set $\mathbb{A}^d$, which is the paradigmatic set of dimension d. As we have remarked at the beginning of this book, the number of orbits in $\mathbb{A}^d$ is at least exponential in d for any structure that has at least two atoms. After we fix the dimension d, it is no longer meaningful to talk about the asymptotics (such as polynomial or exponential) of the number of orbits, since there is only one set $\mathbb{A}^d$ under consideration. We will, however, be interested in a different aspect of the orbit count, namely counting orbits with respect to some support $\bar{a}$, i.e. the number of $\bar{a}$-orbits from Definition 10.8. Under this framing, we can talk about the asymptotics of the orbit count in terms of the number of atoms in the support $\bar{a}$.

Example 91. Assume the equality atoms. Let us consider the $\bar{a}$-orbits in $\mathbb{A}^d$, where the dimension d is fixed to be 1, but the number of atoms in the support $\bar{a}$ can grow. Suppose first that the support $\bar{a}$ consists of two atoms, John and Mary. In this case, there are three $\bar{a}$-orbits, namely

$$\{\text{John}\} \qquad \{\text{Mary}\} \qquad \underbrace{\mathbb{A} \setminus \{\text{John}, \text{Mary}\}}_{\text{fresh orbit}}.$$

More generally, if the support $\bar{a}$ has n atoms, then the number of orbits is $n + 1$. This is because there is one orbit for each atom in the support, and one orbit for the fresh atoms, which are all the atoms that are not in the support.

Let us now see what happens in dimension $d = 2$. Roughly speaking, an $\bar{a}$-orbit in $\mathbb{A}^2$ is obtained by taking a product of two $\bar{a}$-orbits in $\mathbb{A}^1$, such as

$$\{\text{Mary}\} \times (\mathbb{A} \setminus \{\text{John}, \text{Mary}\}).$$

There is one exception to this rule, namely if we take the product of the fresh orbit with itself, then we need to distinguish between equal and non-equal pairs (this distinction is not necessary for the other pairs of orbits, since the answer to equality is known). Therefore, the number of orbits is going to be $(n + 1)^2 + 1$. □

In the above example, the number of orbits was linear in $\bar{a}$ for dimension $d = 1$, and quadratic for dimension $d = 2$. This suggests that this orbit count could be polynomial for every fixed dimension, but with the polynomial (and its degree) depending on the

dimension. This is indeed the case, if we use the equality atoms, as shown in the following lemma.

Lemma 12.1. *Assume the equality atoms. For every fixed d, the number of $\bar{a}$-orbits in $\mathbb{A}^d$ is polynomial in the number of atoms in $\bar{a}$. The polynomial, and in particular its degree, depends on the dimension.*

Proof. Consider some support $\bar{a}$, whose size n is possibly much larger than the fixed dimension d. For a tuple in $\mathbb{A}^d$, define its $\bar{a}$-profile to be the function

$$\{1, \ldots, d\} \to \{\underbrace{a_1, \ldots, a_n}_{\text{atoms from } \bar{a}}, \text{fresh}\}$$

that tells us which coordinates store atoms from $\bar{a}$ and which ones store fresh atoms. Once the dimension is fixed, the number of profiles is polynomial in n, with the degree of the polynomial being d. A $\bar{a}$-orbit is then uniquely determined by its profile, and the information about which fresh coordinates are equal to each other. The second piece of information is exponential in d, but it is fixed when d is fixed. □

The following example shows that the assumption on equality atoms is important, since the orbit count can be exponential for certain atom structures, even if we fix the dimension to $d = 1$.

Example 92. Assume the graph atoms. Already in dimension $d = 1$, the number of $\bar{a}$-orbits in $\mathbb{A}^d$ is exponential in the number of atoms in the support $\bar{a}$. This is because such an orbit is determined by the presence/absence of edges with the atoms in the support, which can be chosen in exponentially many ways. A similar situation arises for the bit vector atoms, where each linear combination of atoms from $\bar{a}$ gives a different orbit. □

Motivated by the above example, in the rest of this chapter we will only talk about the equality atoms. The results would also apply to the linearly ordered atoms, but they will not apply to the atoms discussed in the previous example.

As we will see in this chapter, many algorithms will be polynomial once we have fixed a bound on the dimension. Before presenting the results in more detail, let us first discuss our running example in this book, which is the graph reachability problem.

Example 93. [Running time for graph reachability] Assume the equality atoms, and consider the graph reachability problem. We give a sketch of why this problem can be solved in polynomial time, once the dimension is fixed. A more formal proof of this result will be given in Corollary 12.12 later in this chapter.

Assume first that the instance of the problem is equivariant, i.e. the set of vertices is equivariant, and the same holds for the edges and the source/target sets. Also, for the sake of this sketch, assume that the set of vertices is a subset of a pof set, i.e. it is of the form

$$V \subseteq \mathbb{A}^{d_1} + \cdots + \mathbb{A}^{d_n} \qquad \text{for some } d_1, \ldots, d_n \in \{0, 1, \ldots\}. \tag{12.1}$$

In this case, the dimension d of V is the maximal exponent $d_1, \ldots, d_n$. If the dimension is assumed to be fixed, then the number of orbits is polynomial – in fact linear – in the

number of summands n, since each summand $\mathbb{A}^{d_i}$ contributes a constant number of orbits. The graph reachability algorithm, see the source code in Example 80, executes a while loop, where the i-th iteration computes the set V_i of vertices that are reachable in at most i steps from the source vertices. Because the graph is equivariant, we know that each set V_i is equivariant, i.e. it is a sum of orbits. Since V_i grows with i, it follows that the number of iterations is bounded by the number of orbits, and therefore it is polynomial – in fact linear – in n.

The linear bound discussed above will become polynomial, once we take into account graphs that are not equivariant. To see why, suppose that V is still as in (12.1), but it is no longer assumed to be equivariant, but only finitely supported. Let $\bar{a}$ be a tuple of atoms that supports the set of vertices, and also the edges and source/target vertices. The same analysis applies as in the equivariant case, except that now each V_i is a $\bar{a}$-supported subset of the vertices. Therefore, the number of iterations in the algorithm is bounded by the number of $\bar{a}$-orbits in the vertices, which is at most

$$(\text{number of } \bar{a}\text{-orbits in } \mathbb{A}^{d_1}) + \cdots + (\text{number of } \bar{a}\text{-orbits in } \mathbb{A}^{d_n}).$$

By Lemma 12.1, once we have fixed an upper bound on the dimensions d_j, this quantity is linear in n and polynomial in the number of atoms in the support $\bar{a}$. □

12.1.1 Dimension

So far, we have only considered the dimension and its algorithmic consequences for powers of the atoms, i.e. sets of the form $\mathbb{A}^d$. In this section, we give a general definition of dimension, which applies to all representable sets. Before presenting the definition, however, we discuss some other examples of representable sets, and their expected dimensions.

Example 94. An example of a representable set is

$$\binom{\mathbb{A}}{2} = \{\{x, y\} \mid \text{ for } x, y \in \mathbb{A} \text{ such that } x \neq y\}.$$

An element of this set stores two atoms (i.e. it has a support of size 2), and for this reason, its dimension will turn out to be 2. However, this example might still be misleading, since it suggests a definition of dimension which is the maximal support size of its elements. This will be incorrect for representable sets that atoms shared by the entire set, since such atoms will not contribute to the dimension. For example, the set

$$\{ (\text{John}, x, y) \mid \ x, y \in \mathbb{A} \ \}$$

will also have dimension 2, despite its elements having supports of size 3. The general idea is that the dimension refers to the atoms which are a source of infinity, i.e. the atoms which can be chosen freely. For this reason, any finite set such as

$$\{\text{John}, \text{Mary}, \text{Eve}\}$$

will have dimension 0, since it does not contain any free atoms. □

The ideas discussed in the above example are formalised in the following definition. The definition will need to take care of one more phenomenon, which is the nested nature of representable sets.

Example 95. Consider the set $\{\mathbb{A}^4\}$. This is a set that has one element only, namely $\mathbb{A}^4$, and this element has empty support. Nevertheless, the dimension of this set will be 4 and not 0, because the definition of dimension will take into account not only elements of a set, but elements of elements, and so on. This is because algorithms will potentially look at such elements of elements, and this needs to be taken into account when defining the dimension. □

We are now ready to present the formal definition of dimension.

Definition 12.2 (Dimension of a representable set)**.** Assume the equality atoms. Define the *dimension* of a representable set X, denoted by $\dim X$, to be

$$\max_x \quad \text{number of atoms in least support of } x \text{ that are not in the least support of } X,$$

where x ranges over the downset $X{\downarrow}$, see Definition 10.13.

Example 96. [Dimension of finitely supported subsets of $\mathbb{A}^d$] Consider the set $\mathbb{A}^d$. We will show that the dimension of this set, if we apply Definition 12.2, is the expected value of d. Recall that tuples are encoded as nested pairs, with pairs using Kuratowski encoding, see Examples 75 and 76. Here is an example in dimension $d = 3$:

$$[\text{John}, \text{Mary}, \text{Eve}] = \{\{\text{John}\}, \{\text{John}, \{\{\text{Mary}\}, \{\text{Mary}, \{\{\text{Eve}\}, \{\text{Eve}, \emptyset\}\}\}\}\}\}.$$

All sets used in a tuple $\bar{a}$ are constructed using the atoms from the tuple. Therefore, all supports in the corresponding downset will contain only atoms from the tuple, i.e. at most d atoms. This means that the dimension of $\mathbb{A}^d$ is d.

Consider now a subset of $\mathbb{A}^d$ which is finitely supported, but not necessarily equivariant. This subset will have dimension at most d, but the dimension might be strictly smaller. For example, if we consider a set of the form

$$\{\bar{a}\} \times \mathbb{A}^{d_2} \qquad \text{for some } d_1 + d_2 = d \text{ and } \bar{a} \in \mathbb{A}^{d_1}, \tag{12.2}$$

then this will be a subset of $\mathbb{A}^d$ that has atom dimension d_2, despite the fact that every tuple in this set has support d. □

The notion of size from Definition 12.2 was defined in a semantic way, referring to the representable set and not its representation via a set builder expression. However, this semantic definition can also be reflected in a syntactic one, which is defined in terms of the set builder expression. In the syntactic, we will be counting free variables in the set builder expression that represents the set. The semantic notion of dimension was hereditary, in the sense that we talked not only about the set X itself, but also about sets in X_*, i.e. the elements, their elements, etc. The same will be true for the syntactic dimension, where we will be interested also in sub-expressions. Apart from

the sub-expressions, we will also be interested in subformulas, which are first-order formulas that are used in guards. Here is an example that illustrates these notions:

$$\{\overbrace{\{\underbrace{x, y, z}_{\text{sub-expression}} \mid \text{for } z \in \mathbb{A} \text{ such that } \underbrace{z \neq x}_{\text{subformula}}\}}^{\text{sub-expression}} \mid \text{for } x, y \in \mathbb{A} \text{ such that } \exists z\ (z \neq x \wedge \underbrace{\exists u\ u \neq y}_{\text{subformula}})\}$$

Each sub-expression or subformula has some free variables, which range over atoms[1]. In the dimension, we will count the number of free variables that are not free in the entire expression[2].

Definition 12.3 (Dimension of a set builder expression)**.** Let α be a set builder expression, where all guards are quantifier-free. The *dimension* of α is defined to be

$$\max_{\beta} \quad \text{number of variables that are free in } \beta \text{ but not in } \alpha,$$

where β ranges over sub-expressions and subformulas of α.

The following lemma shows that the two notions of dimensions coincide, i.e. the semantic dimension of a representable set is the minimal syntactic dimension of its representations.

Lemma 12.4. *A representable set has dimension at most d (in the sense of Definition 12.2) if and only if it is represented by some set builder expression that has dimension at most d (in the sense of Definition 12.3).*

Proof. Same proof as Theorem 10.7. □

12.1.2 Size

So far, we have discussed the dimension. The main topic of this chapter is algorithms that run in polynomial time, once the dimension is fixed. This naturally raises the question: polynomial in what? This requires having some notion of size for a representable set. In this section, we define this notion.

Clearly, we cannot use the usual notion of size, i.e. the number of elements. This is because representable sets will typically be infinite. Our notion of size can be defined in two ways: (a) syntactically, as the smallest size of a set builder expression; or (b) semantically, by counting orbits. In this section, we prove Lemma 12.5, which says that these two notions give the same result, up to polynomial factors, once the dimension is fixed. This will mean that there is a robust notion of size, as long as we

[1] By Theorem 6.3, we know that the guards can be made quantifier-free. If the guards are indeed quantifier-free, then counting the free variables in guards is spurious and already covered by counting free variables in subexpressions. However, in our algorithms we will encounter guards that are not quantifier-free, and therefore we want the notion of dimension to account for such guards as well. That is why we will also discuss subformulas of guards when discussing the notion of dimension.

[2] For this to be meaningful, we assume that the free variables of the entire expression are never reused as bound variables inside the expression. This assumption can always be ensured by renaming bound variables.

fix the dimension, and we do not care about polynomial factors. (This is a weaker correspondence than the one from Lemma 12.4, where the two notions of dimension were equal, not just polynomially related.)

Lemma 12.5. *Fix a dimension $d \in \{0, 1, \ldots\}$. For every representable set X of dimension d, the following quantities are bounded by polynomials of each other (the polynomials, and in particular their degree, depend on d):*

1. *The semantic size of X, which is defined to be the number of $\bar{a}$-orbits in the downset $X{\downarrow}$, where $\bar{a}$ is the least support of X, and*

2. *The syntactic size of X, which is defined to be the least size of a set builder expression that represents it, and which uses the optimal[3] dimension d.*

Proof. We begin by showing that the semantic size is bounded by a polynomial function of the syntactic size. Suppose that X is represented by a set builder expression α together with a valuation $\bar{a}$ of its free variables. We will show that the semantic size of X is at most polynomial in the size of α (the bound will be most meaningful when the representation has minimal size). The semantic size of X is defined by counting orbits in the downset with respect to the least support, which is smaller or equal to the number of $\bar{a}$-orbits in the downset, because $\bar{a}$ is not necessarily the least support. Each element of the downset is obtained by taking a subexpression β of α, and a suitable valuation of its free variables. By the assumption that α has optimal dimension, for each subexpression β, its free variables consist of the free variables of the original expression α, plus at most d extra variables. Therefore, each element of the downset is of the form $\beta(\bar{a}\bar{b})$, where β is a subexpression and $\bar{b}$ is a tuple of at most d atoms. The semantic size of X is at most

$$\sum_{\beta} \text{number of } \bar{a}\text{-orbits in the set } \underbrace{\{\, \bar{b} \mid \beta(\bar{a}\bar{b}) \text{ is in the downset of } X \,\}}_{\text{call this set } X_\beta},$$

where β ranges over subexpressions and $\bar{b}$ ranges over the free variables of β that are not free in α. Since the length of the tuple $\bar{b}$ is bounded by d, the set X_β has dimension at most d, and therefore by Lemma 12.1 its number of $\bar{a}$-orbits is polynomial in the size of $\bar{a}$.

Let us now show the converse bound. Consider a representable set X of dimension d. We will find a set builder expression whose size is polynomial in its semantic size. (Recall that the size of a set builder expression is the number of distinct subexpressions and subformulas in the guards.) Let $\bar{a}$ be the least support of X. Consider an element x of the downset $X{\downarrow}$. By definition of dimension, we know that x is either an atom (in which case it has a support of size one) or it is a set which admits a support of the form $\bar{a}\bar{b}$, where $\bar{b}$ contains at most d atoms. By Theorem 10.7, in the latter case x has a representation of the form $\alpha(\bar{a}\bar{b})$ for some set builder expression α. It is not hard to see that expression depends only on the $\bar{a}$-orbit of x, because if π is a $\bar{a}$-automorphism,

[3] We need the assumption that the set builder expression has optimal dimension. As discussed in Exercise 174, there one can get an exponential decrease in size by using suboptimal dimension.

then we have

$$\pi(x) = \alpha(\pi(\bar{a}\bar{b})) = \alpha(\bar{a}\pi(\bar{b})).$$

Let Γ be the set of all possible expressions α that arises this way. The number of such expressions is at most the number of $\bar{a}$-orbits in $X\downarrow$, which is the semantic size. Let us now show that we can ensure that Γ is closed under taking subexpressions, and that the guards in all expressions from Γ are of polynomial size. This will establish that each expression in Γ has polynomial size, since we define the size of an expression to be the number of different subexpressions plus the size of the guards. Consider some expression $\alpha(\bar{x}\bar{y}) \in \Gamma$, where $\bar{x}$ are the variables for $\bar{a}$ and $\bar{y}$ are the remaining variables. (We write d_α for the number of remaining variables; this number depends on α but is guaranteed to be at most d.) Every element of $\alpha(\bar{x}\bar{y})$ is either an atom, or a simpler expression, and therefore we know that

$$\begin{aligned}\alpha(\bar{x}\bar{y}) \quad = \quad & \{z \mid \text{ for } z \in \mathbb{A} \text{ such that } z \in \alpha(\bar{x}\bar{y})\} \quad \cup \\ & \bigcup_{\beta} \{\beta(\bar{x}\bar{z}) \mid \text{ for } \bar{z} \in \mathbb{A}^{d_\beta} \text{ such that } \beta(\bar{x}\bar{z}) \in \alpha(\bar{x}\bar{y})\},\end{aligned}$$

where β in the union ranges over expressions from Γ that have strictly smaller rank that α. (The rank is the nesting depth of set brackets.) The expression on the right-hand size of the above equality is not strictly speaking a set builder expression, since it uses guards of the form

$$z \in \alpha(\bar{x}\bar{y}) \qquad \text{and} \qquad \gamma(\bar{x}\bar{z}) \in \alpha(\bar{x}\bar{y}).$$

However, by the Symbol Pushing Lemma, these guards can be rewritten as first-order formulas, in fact as quantifier-free formulas (because the equality atoms have quantifier elimination). It remains to show that these quantifier-free formulas have polynomial size, assuming fixed dimension. We will apply the guards only to valuations where the tuple $\bar{x}$ is instantiated to $\bar{a}$. Therefore, the truth value of the guards will depend only on the $\bar{a}$-orbit of the remaining variables in the guards. The number of the remaining variables is constant for fixed dimension, and therefore, the number of such orbits is polynomial in the length of $\bar{a}$, by Lemma 12.1. Therefore, the guards can be expressed as quantifier-free formulas of polynomial size. □

12.2 Fixed dimension polynomial time

Having defined the dimension and size of representable sets, we are now ready to define our complexity class. The idea is that the algorithm inputs a representation, and its running time is polynomial (in the size of the input representation) once the dimension is fixed, although we allow the degree of the polynomial to depend on the dimension. By the results from the previous section, the notions of dimension and size are robust, and can be defined both syntactically and semantically. Also, there is a second – and less important – requirement that the output of the function has bounded dimension; this will ensure that the functions in the class can be composed. For functions with Boolean outputs, the second requirement will be superfluous, since there will be only two possible outputs, and hence bounded dimension.

Definition 12.6 (Fixed dimension polynomial time)**.** Consider an oligomorphic atom structure and an atom representation. A function

$$f : \text{representable sets} \to \text{representable sets}$$

is said to be in *fixed dimension polynomial time*, FDP for short, if there is an algorithm which inputs a representation $\alpha(\bar{a})$ of a set X and outputs a representation of $f(X)$, subject to the following constraints:

- The running time is at most

$$O(|\alpha|^{g_1(\dim(\alpha))}) \qquad \text{for some computable } g_1 : \mathbb{N} \to \mathbb{N},$$

- The dimension of the output expression is at most

$$g_1(\dim(\alpha)) \qquad \text{for some computable } g_2 : \mathbb{N} \to \mathbb{N}.$$

In the above definition, we use the syntactic notions of size and dimension, as defined in Definition 12.2 and Condition 2 from Lemma 12.5, respectively. However, thanks to Lemmas 12.4 and 12.5, these values coincide with their semantic counterparts. We have defined the class for general atoms, but it will only be useful for certain atoms. For the purpose of simplicity, we consider only the equality atoms. In this case, an atom representation is any function $2^* \to \mathbb{A}$ which allows a decidable equality check, i.e. we can check if two strings represent the same atom. For the equality atoms, the representation is not particularly important, and we can simply assume that it is a bijection, i.e. the atoms are simply the same thing as strings in 2^*, without any nontrivial equalities.

In the rest of this section, we will show that the class defined above contains many algorithms that we have discussed so far in this book. We begin with the simplest problems, such as testing equality $X = Y$ or membership $X \in Y$, which were treated in Theorem 10.11. The following result strengthens Theorem 10.11, by adding that the algorithms are in syntactic FDP, assuming that the equality atoms are used.

Theorem 12.7. *Consider the equality atoms, together with a bijective atom representation. Given representations of sets with atoms X and Y, in syntactic FDP one can compute:*

1. *the answers to the following questions: $X = Y$, $X \in Y$ and $X \subseteq Y$;*
2. *representations of the sets $X \cup Y$, $X \cap Y$ and $X \setminus Y$.*

Proof. We use the same proof as for Theorem 10.11, except that we observe that it is compatible with syntactic FDP. In the previous proof, we used the Symbol Pushing Lemma to reduce the problems to the first-order theory of the atoms, with parameters. In this new proof, we simply observe that the reduction, and the theory of the atoms, can both be treated in syntactic FDP.

Before continuing, we need to extend the notion of dimension from set builder expressions to formulas, since the reduction and deciding the theory are both problems which involve formulas. This is done in the natural way:

$$\dim \varphi \quad = \quad \max_{\beta} \quad \text{number of variables that are free in } \beta \text{ but not in } \varphi,$$

where β ranges over subformulas of φ.

Consider a question as in the statement of the theorem, e.g. a membership test $X \in Y$. Using the Symbol Pushing Lemma, compute in polynomial time a first-order sentence $\varphi(\bar{x})$ and a valuation $\bar{a} \in \mathbb{A}^{\bar{x}}$ of its free variables, such that $X \in Y$ holds if and only if $\varphi(\bar{a})$ is true in the atom structure. The first observation, which follows from a straightforward inspection of its proof, is that the Symbol Pushing Lemma itself is FDP, i.e. the dimension of the output formula is bounded by the dimension of the input set builder expressions. (The time needed to compute was already shown to be polynomial, even independently of the input dimension.) Therefore, it remains to show that the question $\varphi(\bar{a})$ can be decided in FDP, which is done in the following lemma.

Lemma 12.8. *For every fixed d, there is a polynomial time algorithm which does this:*

- **Input.** *A first-order formula $\varphi(\bar{x})$ of dimension at most d, and atoms $\bar{a} \in \mathbb{A}^{\bar{x}}$.*
- **Output.** *Is $\varphi(\bar{a})$ true in the atom structure?*

Proof. It is easier to think of $\varphi(\bar{a})$ as a formula that does not have any free variables, but which uses constants from the atoms. Consider a subformula of $\varphi(\bar{a})$, which is of the form $\psi(\bar{a}\bar{y})$, for some variables $\bar{y}$. The number of variables in $\bar{y}$ is at most d, by the assumption on dimension. This subformula defines a subset of $\mathbb{A}^{\bar{y}}$ that is supported by $\bar{a}$. Like any subset supported by $\bar{a}$, this subset is a union of $\bar{a}$-orbits. The number of $\bar{a}$-orbits is polynomial in the size of the support $\bar{a}$, as shown in Lemma 12.1. Each individual $\bar{a}$-orbit is described by a formula of polynomial size, which compares the free variables $\bar{y}$ to each other and the constants in $\bar{a}$. Therefore, the union of orbits can be defined by a formula polynomial in $\bar{a}$. Summing up, we have shown that for each subformula of $\varphi(\bar{a})$, there is an equivalent formula that has size polynomial in $\bar{a}$. This formula can be easily computed by an induction on the size of subformulas, in the same way as the quantifier elimination procedure from Theorem 6.3. □

Applying the algorithm from the above lemma to the output of the Symbol Pushing Lemma, we get an answer to the questions in the first item of the theorem. The same argument applies to the construction of the sets in the second item. □

Another result on representable sets was the Downset Lemma. We do not even need to revisit this lemma, since it was already polynomial, at least once we have fixed the formula φ, or more precisely, the number of variables used in this formula.

The algorithms given in Theorem 12.7 and the Downset Lemma are useful, but they only correspond to basic operations that would be invoked in a single step by an algorithm. Of course an algorithm, such as the one for graph reachability, would need to do more than just a single step. To deal with such algorithms, we will present a meta-theorem, namely that any "saturation algorithm" on the downset can be implemented in syntactic FDP.

Inflationary fixpoints. We will formalise "saturation procedures" by using fixpoints. Let us begin with a motivating example based on graph reachability.

Example 97. Consider the relation "there is a nonempty path from x to y" in a graph. This relation is the binary relation R on vertices which contains the edge relation E and which is closed under composition. This means that we can think of this set as being the least fixpoint of the following operator, which inputs and outputs binary relations on vertices:

$$R \quad \mapsto \quad E \cup R \circ R \cup R.$$

The red part, where R is added to the output, is not strictly necessary, but it will better fit our narrative about inflationary fixpoints. The fixpoint can be computed by starting with R as the empty set, and applying the operator as many times as it takes to stabilize. Since the output of the operator contains the input, because of the red part, the operator will reach a fixpoint in a finite number of steps, assuming that the graph is finite. □

In the above example, we had an operator on sets, and we kept on iterating it until it stabilised. However, as mentioned in the example, one needs to ensure that the iteration process stabilises. We take a very straightforward approach, which is to ensure that the relation grows or stays the same in each step, by simply adding its previous value[4].

Definition 12.9 (Inflationary fixpoint)**.** Consider an operator f on subsets of A^k for some set A and some power $k \in \{1, 2, \ldots\}$. The *inflationary fixpoint* of this operator is the limit of the sequence

$$\begin{aligned} R_0 &= \emptyset \\ R_{i+1} &= f(R_i) \cup R_i \end{aligned}$$

If the set A in the above definition is finite, then the inflationary fixpoint is guaranteed to exist, and the number of steps required to reach it is at most $|A|^k$. For infinite sets, the inflationary fixpoint might not exist. This could be repaired by using transfinite induction, i.e. extending the sequence to possibly infinite ordinal numbers. However, such a repair will not be necessary in our context, because we will use the fixpoint for representable sets. As we will see, such sets behave more like finite sets, and the fixpoint will stabilise in a finite number of steps. Furthermore, it can be computed in FDP.

Theorem 12.10. *Fix a formula $\varphi(x_1, \ldots, x_k)$ which uses a binary relation $\in$ and a relation name R that has k arguments. The following can be computed in syntactic FDP:*

- **Input.** *A representable set X.*

[4] An alternative approach, more commonly used for fixpoint logics, is to require the operator to be monotone, i.e. if the input relation is made bigger as a set, then the output relation is also made bigger or stays the same.

- **Output.** *The k-ary relation on the downset of X, which is obtained by taking the inflationary fixpoint of the operator defined by φ. In particular, the inflationary fixpoint is defined.*

Proof. Fix a dimension bound d for the representation of the input set X. When we talk about something being polynomial in the following proof, we mean polynomial once d and the formula φ have been fixed. (Actually, we do not even need to fix the formula φ, but have to fix the number of variables used in this formula and its subformulas.) The input to the algorithm is a representation of X, say $\alpha(\bar{a})$. Consider the k-ary relations on X

$$R_0 \subseteq R_1 \subseteq R_2 \subseteq$$

that are used in the definition of the inflationary fixpoint. We can use the Equivariance Principle to conclude that each of these relations is supported by $\bar{a}$. Therefore, the number of steps that is needed to reach the fixpoint bounded by the number of $\bar{a}$-orbits in the downset $X{\downarrow}$. The following claim shows that this bound is polynomial. (In particular, the bound is finite, and therefore the inflationary fixpoint is defined.)

Claim 12.11. *The number of $\bar{a}$-orbits in the downset $X{\downarrow}$ is polynomial in the size of α.*

Proof. Each element of the downset is obtained by taking some subexpression of α and some valuation of its free variables. Consider a subexpression, which has the form $\beta(\bar{x}\bar{z})$, where $\bar{x}$ are the free variables of the original expression α, and $\bar{z}$ are the remaining free variables. The number of variables in $\bar{z}$ is bounded by the fixed dimension d. An element of the downset that corresponds to β will arise by using the fixed support $\bar{a}$ for $\bar{x}$, and then choosing some values of the variables for $\bar{z}$. Since the length of $\bar{z}$ is fixed, the number of $\bar{a}$-orbits for the latter choice is polynomial in $\bar{a}$, thanks to Lemma 12.1. □

Let n be the polynomial bound from the above claim. By inlining the formula φ in itself n times, we can compute in polynomial time a new formula φ_n that computes the n-th iteration of the fixpoint computation. More formally, define φ_0 the formula "false", and define

$$\varphi_{i+1} \quad \stackrel{\text{def}}{=} \quad \varphi_i \vee \varphi[R := \varphi_i]$$

The size of the formula φ_i is polynomial in i, since we measure the size of formulas by counting subformulas. Furthermore, the number of variables in φ_i is bounded by the number of variables in φ, at least if we reuse variables. Therefore, we can apply the Downset Lemma to evaluate the formula φ_n in the set X in polynomial time, yielding the desired result. □

Many of the algorithms that we have discussed so far in this book can be implemented using fixpoints. Therefore, these algorithms will be in fact in FDP for representable inputs, as stated in the following corollary.

Corollary 12.12. *The following problems for representable inputs are in FDP:*

1. *graph reachability;*
2. *emptiness for nondeterministic automata;*
3. *minimisation for deterministic automata;*
4. *emptiness for context-free grammars.*

Proof. We begin with graph reachability. An input to this problem is a tuple

$$\underbrace{V}_{\text{vertices}} \quad \underbrace{E \subseteq V^2}_{\text{edges}} \quad \underbrace{S \subseteq V}_{\text{sources}} \quad \underbrace{T \subseteq V}_{\text{targets}},$$

where all sets are representable. The input can be seen as a single set $X = (V, E, S, T)$. The set of reachable vertices is the inflationary fixpoint of the operator on unary relations R that is defined by the following formula:

$$\varphi(x) \qquad = \qquad x \in S \vee \exists y\, E(y, x) \wedge R(x).$$

We can now apply Theorem 12.10 to show that this fixpoint can be computed in FDP. Formally speaking, to apply the theorem, we should ensure that the formula uses membership $\in$ and the relation R. This can be done, since the relations S and E can be defined purely in terms of membership, by extracting them from the downset of X.

Automaton nonemptiness reduces to graph reachability, and the reduction is FDP. Therefore, automaton nonemptiness is also in FDP.

Let us now deal with minimisation of deterministic automata. Using reachability, we can trim the state space to the reachable states. Next, we use a fixpoint algorithm to compute the distinguishability relation on states. This is the inflationary fixpoint of the following operator on binary relations R that is defined by the following formula:

$$\varphi(x_1, x_2) \;=\; \vee \begin{cases} \underbrace{Q(x_1) \wedge Q(x_2) \wedge (F(x_1) \Leftrightarrow \neg F(x_2))}_{\text{one of the states accepts the empty word, the other does not}} \\ \underbrace{\exists a\, \exists p_1\, \exists p_2\, \delta(x_1, a, p_1) \wedge \delta(x_2, a, p_2) \wedge R(p_1, p_2)}_{\text{some letter takes } (x_1, x_2) \text{ to a pair of states that accept different words}} \end{cases}.$$

Once we have computed this fixpoint, we can use Currying to compute the set of equivalence classes in polynomial time (see Exercise 161), and from this set we can compute the minimal automaton.

The case of emptiness for context-free grammars is left to the reader. □

Exercises

Exercise 164. Assume any atom structure with at least two elements. Show that the following problem is PSPACE-complete: given a sentence of first-order logic, decide if it is true in the atoms.

Exercise 165. Assume the equality atoms. Show that the following problem is PSPACE-complete: given two set builder expressions without atom parameters, decide if they represent the same set.

Exercise 166. Show that the problem from Exercise 165 remains PSPACE-complete even if we require the guards in set builder expressions to be quantifier-free.

Exercise 167. Show that the dimension of $\mathbb{A}^k$, according to Definition 12.2, is k.

Exercise 168. Consider atoms which are not necessarily the equality atoms. We say that the atoms have *fixed dimension polynomial orbit count* if for every $\bar{a} \in \mathbb{A}^n$, the number of $\bar{a}$-orbits in $\mathbb{A}^k$ is

$$O(n^{f(k)}) \qquad \text{for some computable } f : \mathbb{N} \to \mathbb{N}.$$

Which of the following atoms have fixed dimension polynomial orbit count?

1. The bit vector atoms.
2. The tree atoms.
3. The equivalence relation atoms.
4. A finite atom structure.

Exercise 169. Let $\mathbb{A}$ be a homogeneous structure. Show that $\mathbb{A}$ has fixed dimension polynomial orbit count if and only if there is a fixed dimension polynomial time program which does this:

- **Input.** A set builder expression α and an atom tuple $\bar{a} \in \mathbb{A}^*$;
- **Output.** A Von Neumann numeral which represents the number of $\bar{a}$-orbits in the set represented by $\mathbb{A}^k$.

Exercise 170. Assume that the atoms:

1. are homogeneous over a finite vocabulary;
2. have a computable Ryll-Nardzewski function;
3. have fixed dimension polynomial orbit count, as defined in Exercise 168;
4. admit a polynomial time algorithm which answers questions of the form

$$\mathbb{A} \models \exists x_1, \ldots, x_n \, \varphi$$

where φ is quantifier-free and in DNF.

Show that there is a fixed dimension polynomial program which inputs a first-order formula over the vocabulary of the atoms (possibly using constants from the atoms), and which outputs an equivalent formula that is quantifier-free.

Exercise 171. Let $\mathscr{A}$ be a class of finite structures over a finite relational vocabulary, such that membership in $\mathscr{A}$ can be tested in polynomial time. Show that the Fraïssé limit of $\mathscr{A}$ satisfies the assumptions in Exercise 170.

Exercise 172. Assume the equality atoms. Consider an algorithm which inputs and outputs set builder expressions. We say that the algorithm is *fixed dimension tractable* if its running time on a set builder expression is at most

$$f(\dim \alpha) \cdot |\alpha|^c$$

for some computable function f and constant c that does not depend on the dimension. Show that there is no fixed dimension tractable algorithm for graph reachability, under the following assumption from the field of fixed parameter tractable algorithms:

(*) There is no algorithm which inputs a finite graph G and a first-order sentence φ using a binary edge relation, outputs whether $G \models \varphi$, and runs in time

$$f(\varphi) \cdot |G|^c$$

for some computable function f and constant c.

Exercise 173. The issues in Exercise 172 go away if we disallow parameters. Show that there is a fixed dimension tractable algorithm for graph reachability, which works for equivariant inputs (i.e. the input is a set builder expression without atom constants).

Exercise 174. In Lemma 12.5, when measuring the syntactic size, we use expressions of optimal dimension. Show that this assumption is important because even for sets of dimension 0, one can make a set builder expression exponentially smaller at the cost of using dimension > 0. Consider the set $\{\underline{1}, \underline{2}\}^n$. This is a finite set, and its dimension is 0. Any 0-dimensional set builder expression that represents this set needs to have size at least 2^n, because it needs to essentially enumerate the set. On the other hand, one can represent this set with an n-dimensional expression of size polynomial in n, namely

$$\{(x_1, \ldots, x_k) \mid \text{ for } x_1, \ldots, x_k \in \mathbb{A} \text{ such that } \varphi(x_1, \ldots, x_k)\}$$

where φ is the formula which says that all of its arguments are either $\underline{1}$ or $\underline{2}$.

Exercise 175. Show that the $|X|$ is not upper bounded by any fixed dimension polynomial function of the parameter "number of equivariant orbits that intersect X_*".

Bibliography

Blass, Andreas. 2013. *Power-Dedekind Finiteness*. http://www.math.lsa.umich.edu/~ablass/pd-finite.pdf. [Online; accessed September 6, 2019].

Bojańczyk, Mikołaj, and Toruńczyk, Szymon. 2012. Imperative Programming in Sets with Atoms. Pages 4–15 of: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*.

Bojańczyk, Mikołaj, and Toruńczyk, Szymon. 2018. On computability and tractability for infinite sets. Pages 145–154 of: Dawar, Anuj, and Grädel, Erich (eds), *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 9–12, 2018*. ACM.

Bojańczyk, Mikołaj, Braud, Laurent, Klin, Bartek, and Lasota, Sławomir. 2012. Towards nominal computation. Pages 401–412 of: *POPL*.

Bojańczyk, Mikołaj, Klin, Bartek, Lasota, Sławomir, and Toruńczyk, Szymon. 2013a. Turing Machines with Atoms. Pages 183–192 of: *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*.

Bojańczyk, Mikołaj, Segoufin, Luc, and Toruńczyk, Szymon. 2013b. Verification of database-driven systems via amalgamation. Pages 63–74 of: *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*.

Bojańczyk, Mikołaj, Klin, Bartek, and Lasota, Sławomir. 2014. Automata theory in nominal sets. *Logical Methods in Computer Science*, **10**(3).

Bojańczyk, Mikołaj, Fijalkow, Joanna, Klin, Bartek, and Moerman, Joshua. 2024. Orbit-Finite-Dimensional Vector Spaces and Weighted Register Automata. *TheoretiCS*, **Volume 3**(May).

Cai, Jinyi, Fürer, Martin, and Immerman, Neil. 1992. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, **12**(4), 389–410.

Cheng, Edward Y. C., and Kaminski, Michael. 1998. Context-Free Languages over Infinite Alphabets. *Acta Informatica*, **35**(3), 245–267.

Clemente, Lorenzo, and Lasota, Sławomir. 2015a. Reachability Analysis of First-order Definable Pushdown Systems. Pages 244–259 of: *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*.

Clemente, Lorenzo, and Lasota, Sławomir. 2015b. Timed Pushdown Automata Revisited. Pages 738–749 of: *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6–10, 2015*.

Courcelle, Bruno, and Engelfriet, Joost. 2012. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Encyclopedia of Mathematics and Its Applications, vol. 138. Cambridge University Press.

Engeler, ERWIN. 1959. A characterization of theories with isomorphic denumerable models. *Notices of the American Mathematical Society*, **6**, 161.

Ferrari, Gian Luigi, Montanari, Ugo, and Pistore, Marco. 2002. Minimizing Transition Systems for Name Passing Calculi: A Co-algebraic Formulation. Pages 129–158 of: *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*.

Gabbay, Murdoch, and Pitts, Andrew M. 2002. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, **13**(3-5), 341–363.

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

Klin, Bartek, and Lelyk, Mateusz. 2017. Modal mu-calculus with atoms. Pages 30–1 of: *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Klin, Bartek, Lasota, Sławomir, Ochremiak, Joanna, and Toruńczyk, Szymon. 2014. Turing machines with atoms, constraint satisfaction problems, and descriptive complexity. Pages 58:1–58:10 of: *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14–18, 2014*.

Klin, Bartek, Kopczyński, Eryk, Ochremiak, Joanna, and Toruńczyk, Szymon. 2015. Locally finite constraint satisfaction problems. Pages 475–486 of: *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society.

Kopczyński, Eryk, and Toruńczyk, Szymon. 2016. LOIS: an Application of SMT Solvers. Pages 51–60 of: *Proceedings of the 14th International Workshop on Satisfiability Modulo Theories affiliated with the International Joint Conference on Automated Reasoning, SMT@IJCAR 2016, Coimbra, Portugal, July 1–2, 2016*.

Kopczyński, Eryk, and Toruńczyk, Szymon. 2017. LOIS: syntax and semantics. Pages 586–598 of: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*.

Moerman, Joshua, Sammartino, Matteo, Silva, Alexandra, Klin, Bartek, and Szynwelski, Michal. 2017. Learning nominal automata. Pages 613–625 of: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*.

Murawski, Andrzej, Ramsay, Steven, and Tzevelekos, Nikos. 2014. Reachability in Pushdown Register Automata. Pages 464–473 of: *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*.

Murawski, Andrzej, Ramsay, Steven, and Tzevelekos, Nikos. 2015. Biosimilarity in Fresh-Register Automata. Pages 156–167 of: *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6–10, 2015*.

Pitts, Andrew M. 2013. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science, vol. 57. Cambridge University Press.

Reingold, Omer. 2008. Undirected connectivity in log-space. *J. ACM*, **55**(4).

Ryll-Nardzewski, Czesław. 1959. On the categoricity in power $\leq \aleph_0$. *Bull. Acad. Polon. Sci. Sér. Sci. Math. Astr. Phys*, **7**, 545–548.

Svenonius, Lars. 1959. No-categoricity in first-order predicate calculus 1. *Theoria*, **25**(2), 82–94.

Thomas, Wolfgang. 1990. Automata on infinite objects. Pages 135–191 of: van Leeuwen, J. (ed), *Handbook of Theoretical Computer Science*, vol. vol. B, Formal models and semantics. Elsevier.

Author index

Subject index