Slightly Infinite Sets

Mikołaj Bojańczyk

May 22, 2019
The latest version can be downloaded from:
https://www.mimuw.edu.pl/~bojan/paper/atom-book

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 this 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.

PART ONE

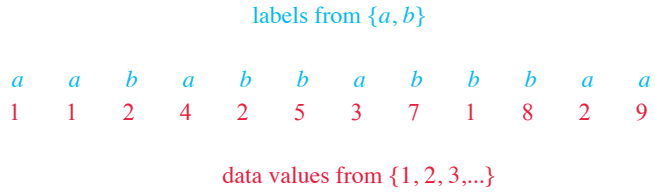
AUTOMATA FOR DATA WORDS

We begin with an investigation of concrete automata models for words over infinite alphabets. The goal of this part is to build intuitions for the more abstract models that will be presented in the later parts.

1

Register automata

A data word is a word where each letter carries two pieces of information: a *label* from a finite set, and a *data value* from an infinite set. Here is a picture:



For the rest of Part I, fix a countably infinite set $\mathbb{A}$. Elements of this secot, called the *atoms*, will be used for the data values. Formally, a *data word* over a finite set of labels Σ is defined to be a word in

$$w \quad \in \quad (\underbrace{\Sigma}_{\text{label}} \times \underbrace{\mathbb{A}}_{\text{data value}})^*.$$

When describing properties of data words, we will be able to test the labels explicitly by asking questions like

does the second letter have $a \in \Sigma$ as its label?

but we will only test the data values for equality e.g. ask

do the third and fifth letters have the same data value?

Later in the book, we will formalise what it means to only test data values for equality, but for now the intuitive understanding should be enough.

Example 1.1. By abuse of notation, we assume that a word in $\mathbb{A}^*$ is also a data word, which uses no labels. Here are examples of languages of data words, in all of these examples we use no labels:

(1) the first data value is the same as the last data value;
(2) some data value appears twice;
(3) no data value appears twice;
(4) the first data value appears again;
(5) every three consecutive data values are pairwise distinct.

We will introduce automata models for data words that capture the properties above. These models use registers to talk about data values.

1.1 Nondeterministic register automata

We begin our discussion with one of the simplest automaton models for data words, namely nondeterministic and deterministic register automata[1].

Definition 1.2 (Nondeterministic register automaton)**.** The syntax of a *nondeterministic register automaton* consists of:

- a finite set Σ of *labels*;
- a finite set Loc of *locations*[2];
- a finite set R of *register names*;
- an *initial location* $\ell_0 \in \mathsf{Loc}$ and a set of *accepting locations* $F \subseteq \mathsf{Loc}$;
- a *transition relation*

$$\delta \subseteq \underbrace{\mathsf{Loc} \times \overbrace{(\mathbb{A} \cup \{\bot\})^R}^{\substack{\text{register valuations, i.e.}\\ \text{partial functions}\\ \text{from registers}\\ \text{to atoms}}}}_{\text{states}} \times \underbrace{\Sigma \times \mathbb{A}}_{\text{input}} \times \underbrace{\mathsf{Loc} \times (\mathbb{A} \cup \{\bot\})^R}_{\text{states}} \tag{1.1}$$

subject to an equivariance condition described below.

The automaton is used to accept or reject data words with labels Σ, i.e. words where each position is labelled by $\Sigma \times \mathbb{A}$. After processing part of the input, the automaton keeps track of a *state*, which is defined to be a location plus a register valuation. Initially, the state consists of the initial location and a completely undefined register valuation. For each input letter, the state is updated according to the transition relation δ, and the automaton accepts if at the end

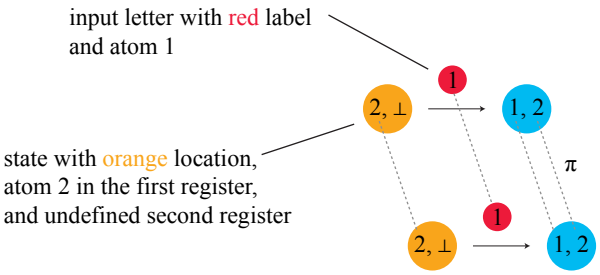
[1] Register automata where introduced in Kaminski and Francez (1994), under the name of *finite memory automata*, together with a decidability proof for the emptiness problems in the deterministic and nondeterministic one-way cases (Theorem 1.7 in this text). The presentation using syntactic and semantics equivariance, in particular Lemma 1.3, is essentially due to Bojańczyk (2013); Bojańczyk et al. (2014).

[2] We use the name location instead of state, because the state of the automaton will store additional information, namely the contents of the registers.

of the input word the state is accepting, in the sense that the location belongs to the accepting set.

How is the transition relation described? Since the state space is infinite, some restrictions on the transition relation are needed to represent it in a finite way. We choose the following restriction, called *equivariance*: the transition relation can only compare atoms with respect to equality, and is not allowed to depend on any specific atoms. Equivariance can be formalised in two different ways below.

Semantic equivariance. A permutation $\pi : \mathbb{A} \to \mathbb{A}$ of the atoms (i.e. a bijection from the atoms to themselves) can be applied to states in the natural way, and therefore also to triples in the transition relation δ (the locations and undefined values are not affected, only the atoms). Here is a picture:



We say that δ is *semantically equivariant* if the set of transitions is invariant under actions of atom permutations, i.e.

$$\pi(t) \in \delta \qquad \text{for every } t \in \delta \text{ and every permutation } \pi : \mathbb{A} \to \mathbb{A}.$$

The advantage of semantic equivariance is that the definition is short, and easy to generalise to other models, like alternating automata or pushdown automata. The disadvantage is that it is not clear how to represent a semantically equivariant transition relation, e.g. for the input of a nonemptiness algorithm. The converse situation holds for syntactic equivariance, as presented below.

Syntactic equivariance. We say that δ is *syntactically equivariant* if it can be defined by a finite boolean combination of constraints of the following types:

(1) the location in the source (respectively, target) state is $\ell \in \mathsf{Loc}$;
(2) the label in the input letter is $a \in \Sigma$;
(3) register $r \in R$ is undefined in the source (respectively, target) state;
(4) the atom in the input letter is the same as in register $r \in R$ of the source (respectively, target) state;

(5) register $r \in R$ of the source (respectively, target) state stores the same atom as register $s \in R$ of the (respectively, target) source state.

In item (5) above, there are four possibilities regarding the choice of source vs target, since the choice is taken independently for r and s.

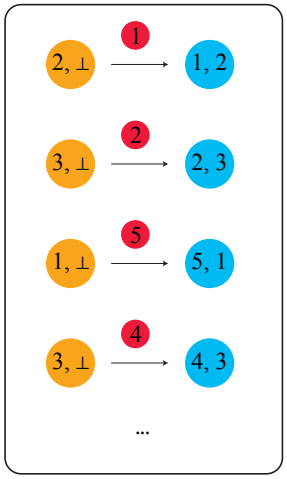
Lemma 1.3. *Semantic and syntactic equivariance are the same.*

Proof It is not difficult to see that semantically equivariant subsets of the set (1.1) are closed under boolean combinations. Since the bijections of data values do not affect satisfaction of the constraints 1-5 used in the definition of syntactic equivariance, it follows that syntactic equivariance implies semantic equivariance.

We now show that semantic implies syntactic. Define an *orbit of transitions* to be a set of the form

$$\{\pi(t) : \pi \text{ is a permutation of the atoms}\} \qquad \text{for some transition } t \in \delta.$$

Here is a picture of an orbit of transitions:



nonempty subset of the set (1.1) which is semantically equivariant and which is minimal for that property with respect to inclusion.

Claim 1.4. *Every orbit of transitions is syntactically equivariant.*

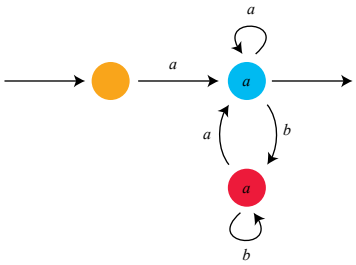
Proof of Claim An orbit of transitions is uniquely defined by its locations, which registers are undefined, what is the label of the input letter, and what is the equality type of the tuple of atoms in the defined registers and the input

letter. All of this information can be expressed using the constraints 1-5 in the definition of syntactic equivariance. □

Once the number of registers and locations is fixed, there are finitely many possible constraints as in the definition of syntactic equivariance. Boolean combinations make the number of possibilities grow, but it remains finite. Therefore, thanks to the above claim, there are finitely many possible orbits of transitions. Finally, every semantically equivariant relation is easily seen to be the union of the orbits contained in it. This union is finite, and each part of the union is syntactically equivariant, and thus the result follows. □

This completes the definition of nondeterministic register automata: the transition relation is required to be equivariant in either of the two equivalent senses defined above. The transition relation is called *deterministic* if the source state and the input letter determine uniquely the target state.

Example 1.5. Here is a deterministic register automaton which recognises language 1 from Example 1.1, i.e. the words in $\mathbb{A}^*$ where the first and last data values are equal. The automaton stores the first data value in its register, and then toggles between accepting or rejecting states depending on whether the input agrees with the register. Here is a picture:



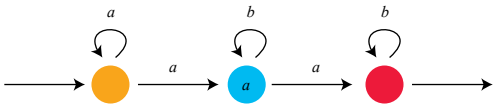
The above picture should be interpreted as follows. There are three locations, standing for the three coloured circles, with initial and final locations depicted by the dangling arrow. Since there is one register, a state consists of a location and a possibly undefined atom. Such states can be found in the picture above. For every pair of distinct atoms $a \neq b$, we add a transition from the above picture to the automaton. Note how every arrow in the picture corresponds to an orbit of transitions.

The method of drawing above has its limitations. For example, if we wanted to add a transition that would involve the orange location with an undefined register, we would need to draw a separate instance of the orange state.

Example 1.6. Languages recognised by nondeterministic register automata are not closed under complement. Consider the language

$$L = \{w \in \mathbb{A}^* : \text{some data value appears twice in } w\}.$$

This languages is recognised by a nondeterministic register automaton with one register and three locations, which uses nondeterminism to guess the repeating data value. Here is a picture:



We now show that the complement of this language – namely the words where no data value repeats – is not recognised by any nondeterministic register automaton. Toward a contradiction, suppose that there is such a nondeterministic automaton, say with $< k$ registers, and consider an accepting run over a word with $2k$ distinct data values:

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{2k}} q_{2k} \tag{1.2}$$

Since the automaton has $< k$ registers then there must be some atoms

$$a \in \{a_1, \ldots, a_k\} \qquad b \in \{a_{k+1}, \ldots, a_{2k}\}$$

such that neither a nor b appear in the registers of state q_k. Let π be the atom permutation which swaps a and b. If we apply π to the second half of the run in (1.2), then we also get an accepting run (because transitions and accepting states are closed under applying permutations, and the permutation π does not affect the state q_k in the middle of the run). This new run sees the atom a twice.

Exercises

Exercise 1. Show that deterministic register automata can recognise languages 4 and 5 from Example 1.1.

Exercise 2. Show that the expressive power of nondeterministic register automata is not affected if we allow ε-transitions.

Exercise 3. For languages of data words one can also define the Myhill-Nerode relation, as used in minimisation of deterministic automata. Show a

language of data words where every deterministic register automaton distinguishes (by its state) some two words which are Myhill-Nerode equivalent.

Exercise 4. Show there is a language of data words, for which there are at least two nonisomorphic deterministic register automata with a minimal number of registers and locations (lexicographically, with the number of registers being more important than the number of locations).

Exercise 5. Show that a nondeterministic register automaton can recognise language 2 from Example 1.1, but a deterministic one cannot.

Exercise 6. Call a nondeterministic register automaton *guessing* if there exists a transition $t \in \delta$ such that some data value in the target state appears neither in the source state nor in the input. Give an example of a language that needs guessing to be recognised.

A corollary of the above two exercises is that:

$$\text{deterministic} \subsetneq \text{nondeterministic without guessing} \subsetneq \text{nondeterministic}.$$

Exercise 7. Call a nondeterministic register automaton *weakly guessing* if every accepting run has the following property: if the transition reading the i-th letter loads an atom a into some register r, then a appears in some position $j \geq i$ such that the transitions reading letters $\{i, \ldots, j\}$ do not remove a from register r. Show that for every nondeterministic register automaton there is a weakly guessing one which accepts the same words.

1.2 Emptiness and universality for register automata

In this section we discuss two decision problems for register automata:

- nonemptiness (does the automaton accept at least one input word); and
- universality (does the automaton accept all input words).

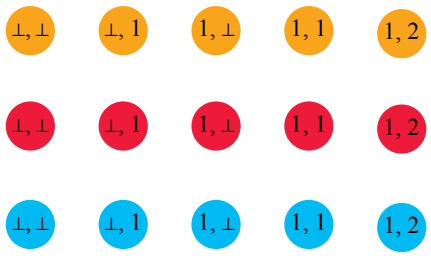
When talking about decidability, we assume that the transition function in a register automaton is represented according to the syntactic equivariance condition.

Theorem 1.7. *Emptiness is decidable for nondeterministic register automata.*

Proof This proof just sketches the decidability argument, the complexity is discussed in Exercise 8. Similarly to the orbits of transitions used in the proof of Lemma 1.3, we define an *orbit of states* to be a set of the form

$$\{\pi(q) : \pi \text{ is a permutation of the atoms}\} \qquad \text{for some state } q.$$

For example, if the automaton has three locations (orange, red and blue) and two registers, then there are 15 orbits of states, as shown in the following picture, with orbits represented by examples of states that use atoms 1 and 2:

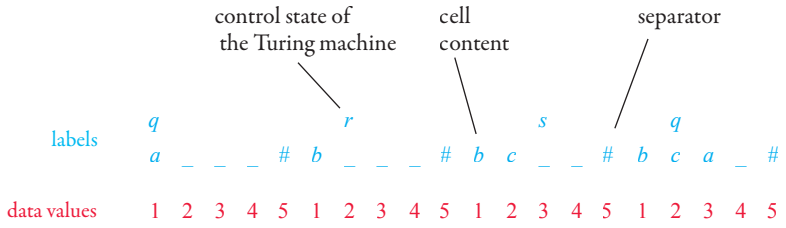


As in Lemma 1.3, an orbit of states can be defined by saying what is the location, which are the defined registers, and what is the equality type of the atoms stored in the defined registers. Such a description takes finite space to store, and there are finitely many possible descriptions (although the number of orbits is exponential in the number of registers). The key observation is that being in the same orbit of states respects reachability, i.e. if two states are in the same orbit, then both are reachable or both are unreachable. The algorithm for nonemptiness computes the orbits of reachable states. Initially, we have the orbit of the unique initial state, which has all registers undefined. If we have the equality type of some state, we can easily compute the equality types of all states reachable from it in one step; thus finishing the description of the algorithm. □

Theorem 1.8. *Universality is undecidable*[3] *for nondeterministic register automata.*

Proof We reduce from the halting problem for Turing machines, i.e. the problem of deciding if a given Turing machine has at least one accepting computation. Suppose that we have a Turing machine which is an instance of the halting problem. We encode a computation of a Turing machine as a data word according to the following picture:

[3] This proof follows the same lines as the undecidability for a stronger model, namely timed automata, see (Alur and Dill, 1994, Theorem 5.2).



Each letter encodes a single cell in a single configuration of the Turing machine. The data word represents a sequence of configurations, padded with blanks so that they all have the same length, and separated by a letter #. The labels are used to store the contents of the cell plus the control state of the head if the head happens to be over that cell. Finally, each cell gets a unique identifier, which is its data value (the same cell in consecutive configurations gets the same identifier).

Claim 1.9. *There is a nondeterministic register automaton which accepts a data word if and only if it is* not *an encoding of an accepting computation of the Turing machine.*

It follows that the Turing machine has no accepting computation if and only if the nondeterministic register automaton accepts all inputs. Therefore, universality of nondeterministic register automata is undecidable.

Proof of the claim To prove the claim, we list the mistakes that can happen in a data word that does not encode an accepting computation of a Turing machine:

(1) The data values identifying the cells are chosen wrong. This means that:
   (i) the separator # is used with more than one data value; or
   (ii) there exist positions i, j with the same data value such that the successor positions $i + 1$ and $j + 1$ are defined and have distinct data values.

   The first condition can be tested using one register, the second condition using two registers.

(2) There is a mistake between two consecutive configurations. Assuming that the identifiers are chosen correctly, this can be tested using only one register, to tell which cells correspond to which ones in the following configuration.

(3) The first configuration is not initial, or the last configuration is not accepting. For this, no registers are needed.

□

□

Exercises

Exercise 8. The complexity of the emptiness problem for nondeterministic register automata depends on how the size $|\mathcal{A}|$ of the input automaton is measured. Show that that emptiness is:

- PSPACE-complete if $|\mathcal{A}|$ is the number of locations and registers;
- NP-complete if $|\mathcal{A}|$ is the number of reachable orbits of states;
- polynomial time if $|\mathcal{A}|$ is the number of orbits of transitions.

Exercise 9. The undecidability proof in Theorem 1.8 used automata with two registers but no guessing (as defined in Exercise 6). Show that, in the presence of guessing, universality remains undecidable even with one register.

Exercise 10. To express properties of data words, we can use first-order logic, where the quantifiers range over positions, and there are predicates for the order on positions, equality of data values, and the labels. For example, the following formula says that every position with label a is followed by a position with label b and the same data value:

$$\underbrace{\forall x}_{\text{for every position } x} \Big(\underbrace{a(x)}_{x \text{ has label } a} \Rightarrow \underbrace{\exists y}_{\text{exists a position } y} \big(\underbrace{y > x}_{y \text{ is after } x} \wedge \underbrace{y \sim x}_{\substack{x \text{ and } y \\ \text{have the same} \\ \text{data value}}} \wedge \underbrace{b(y)}_{y \text{ has label } b} \big) \Big)$$

Show that satisfiability is undecidable for this logic, i.e. one cannot decide if a given formula is true in some data word.

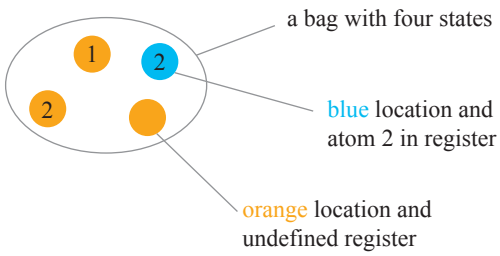
1.3 Alternating register automata

In a nondeterministic automaton, each transition is chosen nondeterministically in favour of acceptance, i.e. for acceptance it suffices that there is at least one choice of transitions that gives an accepting run. An alternating automaton is a generalisation of a nondeterministic automaton, where the syntax specifies which locations choose transitions in favour of acceptance, and which locations choose transitions against acceptance. The main result of this section is

that emptiness is decidable for a restricted version of alternating register automata[4].

Alternating register automata. The syntax of an *alternating register automaton* is defined the same way as for a nondeterministic register automaton, except that there is an additional partition of the locations into two parts, called *existential* and *universal*.

We define the semantics of the automaton using *bags*[5], where a bag is defined to be a set of states. Here is a picture of a bag:



We write P, Q for bags. Bags can be infinite, but for the automata that we will mainly be interested in – non-guessing ones – only finite bags will play a role. If a is an input letter (consisting of a label and a data value) and P, Q are bags then we write

$$P \xrightarrow{a} Q$$

if the following conditions hold:

- for every state $p \in P$ with an existential location, the bag Q contains some state q such that (p, a, q) is a transition; and
- for every state $p \in P$ with an universal location, the bag Q contains all states q such that (p, a, q) is a transition.

A data word $a_1 \cdots a_n$ is accepted by an alternating automaton if there exists a run, which is defined to be a sequence of bags

$$\overbrace{\{\text{initial state}\}}^{\text{initial bag}} = Q_0 \xrightarrow{a_1} Q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} Q_n$$

[4] The result from this section, namely decidability of emptiness for alternating one-way register automata with one register, was first shown in Demri and Lazić (2009). A tree extension of the result can be found in Jurdziński and Lazić (2011).

[5] An alternative but equivalent semantics would use a game played by two players, called "universal" and "existential".

where the last bag is accepting, in the sense that all of its states use accepting locations. We define $\rightarrow$ to be the union of all relations $\xrightarrow{a}$, ranging over all letters a. In terms of this notation, an alternating automaton is nonempty if and only if some accepting bag is reachable from the initial bag via a finite number of steps using the relation $\rightarrow$.

Languages recognised by alternating register automata are closed under complementation, essentially by design, see Exercise 11.

An emptiness algorithm using well quasi-orders. Nondeterministic register automata are the special case of alternating register automata where all states are existential. By Exercise 11, the emptiness and universality problems for alternating register automata are essentially the same problem, which is undecidable by Theorem 1.8.

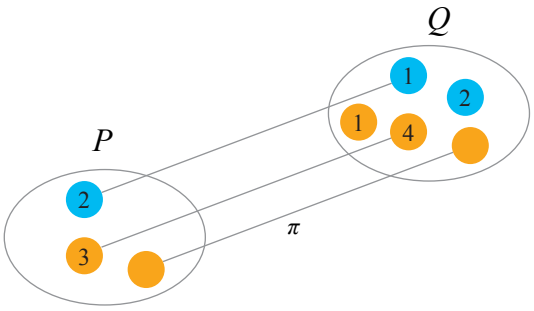
We now identify a restriction on alternating automata which makes emptiness (or universality) decidable. We consider alternating automata that are *non-guessing*, see Exercise 6, which means that for every transition (p, a, q), each atom that appears in q must appear in either p or a. By the remarks in Exercise 9, universality is undecidable for non-guessing nondeterministic automata with two registers, or even guessing nondeterministic automata with one register. Therefore, emptiness is undecidable for alternating automata that are either non-guessing with two registers, or guessing with one register. Anything below that is decidable:

Theorem 1.10. *Emptiness is decidable for alternating non-guessing automata with one register.*

The rest of this section is devoted to proving the above theorem. Nonemptiness is semi-decidable, i.e. there is an algorithm (guess a word and run the automaton on it) which terminates if and only if the input automaton is nonempty. Therefore, in order to prove decidability it suffices to show that emptiness is also semi-decidable. The rest of this section is devoted to designing an algorithm which inputs an automaton (alternating, non-guessing, and with one register) and terminates if and only if the input automaton is empty. In other words, we are searching for a finite and computable witness of emptiness.

Fix an alternating non-guessing automata with one register. Because the automaton is non-guessing, only finite bags can be reached from the initial state. Therefore, from now on, all bags are assumed to be finite.

As in the definition of semantic equivariance from Section 1, permutations of the atoms can be applied to states and to bags of states. The following order on bags is the key to our proof: we write $P \leq Q$ if there is some permutation of the atoms π such that $P \subseteq \pi(Q)$. Here is a picture:



The relation $\leq$ is easily seen to be a quasi-ordering, i.e. it is transitive and reflexive, but not necessarily anti-symmetric. Call a set of bags *upward closed* if whenever it contains a bag P, and $P \leq Q$, then it also contains Q. The *upward closure* of a set of bags is the least upward closed set of bags that contains it. To show that emptiness is semi-decidable, we will use upward closed invariants as described in the following lemma.

Lemma 1.11. *An alternating non-guessing automata with one register is empty if and only if there is an upward closed invariant, i.e. a family of bags $\mathcal{Q}$ which:*

(1) *is upward closed; and*

(2) *contains no accepting bags; and*

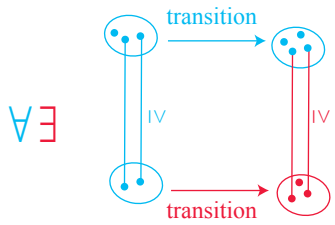
(3) *contains the initial bag; and*

(4) *is closed under transitions, i.e. $P \to Q$ and $P \in \mathcal{Q}$ imply $Q \in \mathcal{Q}$.*

Proof Clearly if there is an upward closed invariant, then the automaton is empty. For the converse implication, suppose that the automaton is nonempty, and define $\mathcal{Q}$ to be the bags that cannot reach an accepting bag. By definition $\mathcal{Q}$ contains no accepting bags, and by assumption on emptiness $\mathcal{Q}$ contains the initial bag. To prove the lemma, it remains to show that $\mathcal{Q}$ is upward closed. This will follow from the following property of the transition relation.

Claim 1.12. *The order $\leq$ on bags is compatible with the transition relation $\to$ in following sense: for every transition $P \to Q$ and every $P' \leq P$ there exists some $Q' \leq Q$ with $P' \to Q'$.*

Proof Here is the picture of compatibility:



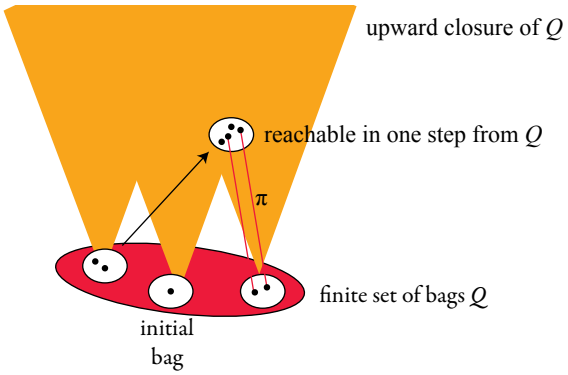
Because $\rightarrow$ is closed under atom permutations, and also closed under making the first argument a smaller bag. □

Since the family of accepting bags is downward closed, a corollary of compatibility as in the above claim is that if a bag can reach an accepting bag, then the same is true for any smaller bag. The contrapositive is that if a bag belongs to $\mathcal{Q}$, then the same is true for any bigger bag. □

The general idea behind the semi-algorithm for emptiness is to search for upward closed invariants as in the above lemma. To represent these invariants in a finite way, we will use the following result.

Lemma 1.13. *Every upward closed invariant is the upward closure of some finite family of bags.*

Before proving the above lemma, we use it to complete the proof of Theorem 1.10. This semi-algorithm for emptiness works as follows: searches through all finite families of bags, and terminate with success if there is a finite family whose upward closure is an upward closed invariant in the sense of Lemma 1.11. By Lemmas 1.11 and 1.13, this semi-algorithm terminates with success if and only if the automaton is empty. It remains to show how one can check, given a finite family of bags, if its upward closure is an upward closed invariant. The first condition, on upward closure is vacuously satisfied. The second condition, on having no accepting bags, corresponds to checking that the finite family has no accepting bags, because upward closure cannot add accepting bags. The third condition, on containing the initial bag, corresponds to checking if the finite family contains either the initial bag or the empty bag (which is the unique bag that is strictly smaller than the initial bag). Finally, we are left with checking if the upward closure is closed under transitions. By compatibility, see Claim 1.12, the upward closure of a finite family $\mathcal{Q}$ is closed under taking transitions if and only if for every bag $Q \in \mathcal{Q}$ and every transition $Q \rightarrow P$, the target bag is in the upward closure of $\mathcal{Q}$, which can easily be checked by enumerating all finitely many candidates for the transition $Q \rightarrow P$. Closure under transitions is illustrated in the following picture:



The idea is that the orange area, i.e. the upward closure of Q_0, is a trap in the sense that no transition can leave the orange area.

It remains to prove Lemma 1.13, which we do in the rest of this section. This is done using well quasi-orders. We say that a quasi-order is a *well quasi-order* if it is well-founded (no infinite strictly decreasing chains) and has no infinite antichains. The technique of well quasi-orders[6], as used in the following proof, is a common method of proving decidable properties for systems with infinitely many configurations.

Lemma 1.14. *In a well quasi-order, every upward closed set is the upward closure of a finite set.*

Proof By well-foundedness, every upward closed set is the upward closure of its minimal elements. The minimal elements form an antichain, and hence there can only be finitely many of them (up to the equivalence where x and y are equivalent if $x \leq y$ and $y \leq x$). □

To prove Lemma 1.13, and therefore also Theorem 1.10, it is enough to establish that the relation $\leq$ on bags is a well quasi-order.

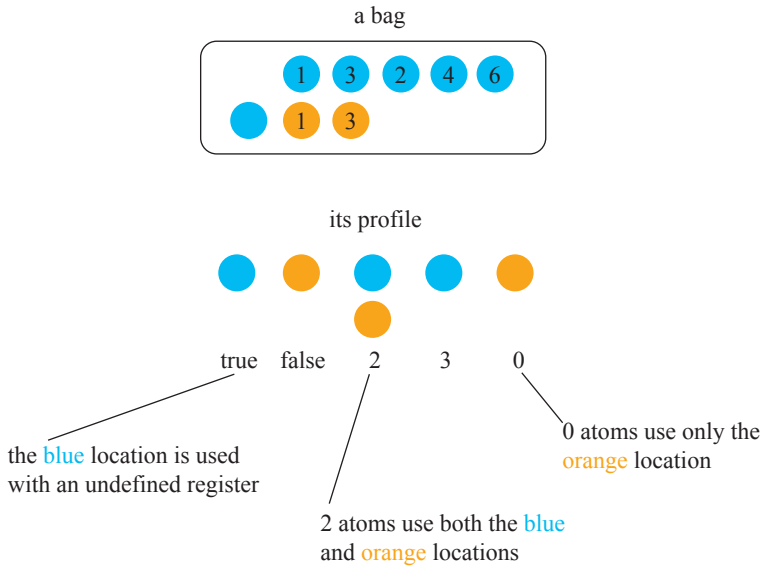
Lemma 1.15. *The relation $\leq$ on bags is a well quasi-order.*

Proof It is clear that the relation is well-founded, since a strict decrease on bags implies a strict decrease in the cardinality (recall that we only consider finite bags). It remains to show that there is no infinite antichain. Define

$$\text{bags} \xrightarrow{\text{profile}} \{\text{true, false}\}^{\text{locations}} \times \mathbb{N}^{\mathsf{P}(\text{locations})-\{\emptyset\}}$$

[6] The well quasi-order technique was independently introduced in Abdulla et al. (2000) and Finkel and Schnoebelen (2001), and is currently known as the technique of *well-structured transition systems*.

to be the function which maps a bag to the information explained in the following picture:



The profile mapping reflects the order in the following sense

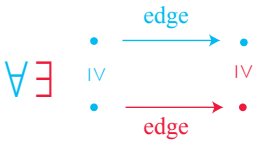
$$\text{profile}(P) \leq \text{profile}(Q) \quad \text{implies} \quad P \leq Q, \tag{1.3}$$

where the order on profiles is coordinate-wise, with false $\leq$ true (see Exercise 15 for why the converse implication fails). Because the order is reflected, the profile mapping maps antichains of bags to antichains of profiles. Since there are no infinite anitichains of profiles by Exercise 14, it follows that there are no infinite antichains of bags. □

The general technique. Using the same proof, we obtain the following generalisation of Theorem 1.10.

Theorem 1.16. *The following problem is decidable.*

- **Input.**
  - *A directed graph where every node has finite outdegree;*
  - *A well quasi-order $\leq$ on vertices such that*



– *A source vertex plus a set of target vertices that is downward closed.*

The input is represented by algorithms for: enumerating the vertices, testing membership in the target set, testing the well quasi-order, and computing the neighbour list of a given vertex.

- **Output.** *Is there a path from the source to one of the targets?*

A temporal logic for data words. One register alternating automata can be dressed up in the syntax of a temporal logic. The idea is to add one register to linear temporal logic LTL. We do not give the detailed syntax and semantics, only some examples. We are extending LTL, so we can write a formula

$$a \text{ until } b,$$

which is true in a (data) word if there is some position with label b such that all earlier positions have label a. Instead of a, b we could have used previously defined formulas, and Boolean combinations are also allowed. There is also an operator next to access the next position. For example, the formula

$$\underbrace{(a \vee \neg a)}_{\top} \text{ until } (a \wedge \text{ next } a)$$

says that there exist two consecutive positions with label a. We use finally φ as syntactic sugar for $\top$ until φ. If we only use the operators until and next, then we have exactly the logic LTL, which is insensitive to the data values. To access the data values, we can add an operator store which stores the current data value, and a formula same which is true whenever the current value is equal to the stored one. For example, the formula

$$\text{store}(\text{next } \neg(\text{finally same}))$$

says that the first data value does not repeat, i.e. after storing it one cannot find the same one again. In principle we could have several different registers for storing data values, but if we want to translate the logic to one register alternating automata, then only one register is allowed (and hence there is no need to give it a name). The register can be reused, e.g. the following formula says that whenever the first data value of the word is used, then the next two positions have distinct data values:

$$\text{store}(\text{next } \neg(\text{finally}(\text{same} \wedge \text{next } (\text{store}(\text{next same})))))$$

Every formula of this temporal logic can be converted into an alternating one register automaton, and therefore one can decide if a formula is true in at least one data word.

Exercises

Exercise 11. Show that languages recognised by alternating register automata are closed under complementation.

Exercise 12. Show that languages recognised by one way non-guessing alternating automata are not closed under reversals.

Let L be the language "for every position x with label *inc*, there is a later position y with label *dec* and the same data value, such that label *zero* does not appear between positions x and y.". This language is recognised by a one way non-guessing alternating automaton. If its reversal were also recognised, then we could use the the same proof as in Exercise 9 to get undecidability.

Exercise 13. Show that a quasi-order is a well-quasi-order if and only if every infinite sequence contains a monotone subsequence, i.e. one where $i \leq j$ implies $x_i \leq x_j$.

Exercise 14. Show that for every dimension $d \in \{1, 2, \ldots\}$, the set $\mathbb{N}^d$ is a well quasi-order with respect to the coordinatewise ordering.

Exercise 15. Show that the converse implication in (1.3) is not true. Find a well quasi-order on profiles which turns the implication into an equivalence.

Exercise 16. For a possibly infinite alphabet Σ, define the Higman ordering on Σ^* to be the relation of not necessarily connected substrings. Show that this is a well quasi-ordering.

Exercise 17. Suppose that the atoms are equipped with a total order. Show that emptiness remains decidable for one register alternating automata without guessing, even when the machine can use the order to compare the register with the current data value[7] .

Exercise 18. For a Turing machine with one tape, define a *gain* to be the process of taking a configuration and inserting nondeterministically one new cell in some position not below the head, with any label from the work alphabet. Define a *gainy computation step* of a Turing machine to be a finite (possibly zero) number of gains followed by a normal step of computation. Show that

[7] The paper Lasota and Piórkowski (2018) investigates what structure on the atoms can be used so that the order $\leq$ on bags is a well quasi-order.

the halting problem is decidable for Turing machines with semantics defined using gainy computation steps.

Exercise 19. Show that there is an infinite antichain for the following order on $\mathbb{A}^*$:

$w \leq v$ if w is Higman smaller or equal to $\pi(v)$ for some permutation of $\mathbb{A}$.

Exercise 20. Show that there is a language $L \subseteq \mathbb{A}^*$ that is upward closed under the Higman order, but is not recognised by a nondeterministic register automaton.

1.4 Most models of register automata are inequivalent

The goal of this section is to collect exercises which paint a depressing picture: with one exception, the only inclusions between models of register automata are the ones that trivially follow from the definitions. To have a richer landscape, we also consider the two-way variant of register automata, where the head of the automaton can move both ways, with the input being extended by markers on both sides[8]. For the purpose of this section, we assume that all models allow ε-transitions.

Exercise 21. Find a deterministic two-way register automaton which recognises the language

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

Exercise 22. 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):

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

[8] An in-depth study of various kinds of register automata can be found in Neven et al. (2004), including undecidability of universality of nondeterministic one-way automata (Theorem 1.8). The non-equivalence results summarised in Figure 1.1 are originally found in Kaminski and Francez (1994); Neven et al. (2004) and an unpublished Master's thesis in Polish Wysocki (2013). For a survey on automata and logic for infinite alphabets, see also Segoufin (2006).

Exercise 23. Find a language of data words that is recognised by an alternating register automaton with guessing, but not by any alternating register automaton without guessing.

Exercise 24. Show that every two-way nondeterministic register automaton can be simulated by an alternating register automaton (with guessing and ε-transitions).

We now present a series of exercises with a more systematic study of the following models of automata: one-way deterministic and nondeterministic, two-way deterministic and nondeterministic, as well as one-way alternating with or without guessing. We assume ε-transitions are allowed in all models. The picture with these six models is in Figure 1.1. The picture shows the obvious containments which follow from the syntax as well as the less obvious containment from Exercise 23. In the solutions to the following exercises, one is allowed to give answers conditional on open problems in complexity theory such as $\textsc{p} = \textsc{np}$.

Exercise 25. Show a language that witnesses point 3 in Figure 1.1.

Exercise 26. Show a language that witnesses point 4 in Figure 1.1, possibly conjectures about complexity classes being distinct.

Exercise 27. Show a language that witnesses point 5 in Figure 1.1, possibly using conjectures about complexity classes being distinct.

Exercise 28. Show that all coloured areas in Figure 1.1 contain languages.

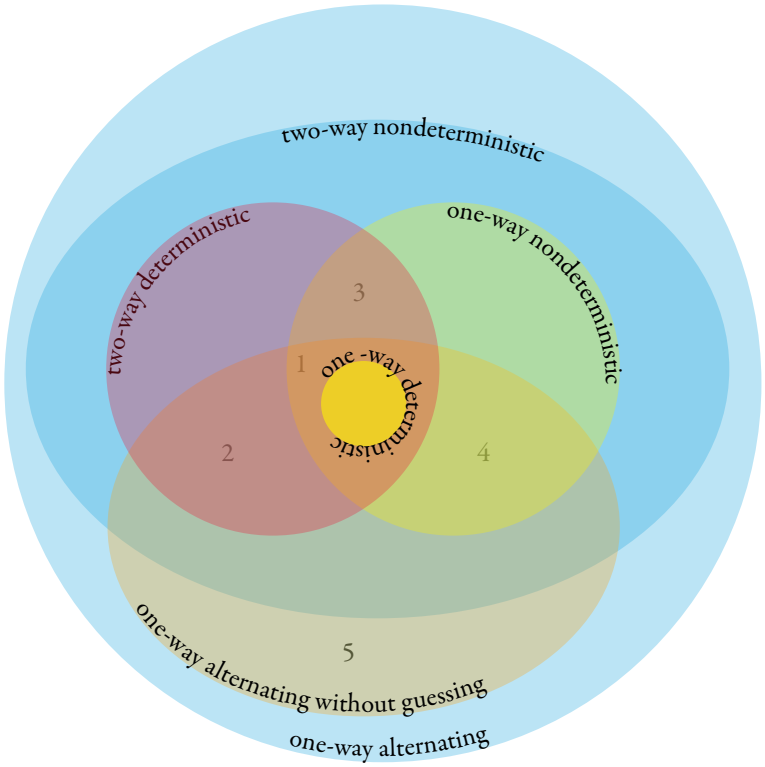


Figure 1.1 Six classes of register automata and their combinations. Point 1 is the language: “last letter appears only once”, while point 2 is the language “all letters are distinct”. The remaining points 3, 4, 5 are Exercises 25-27, while Exercise 28 sums up the results by saying that all combinations are possible.

2

Two variable logic and data automata

This chapter presents an automaton model – *data automata* – which recognises properties of data words without using registers. There are three reasons to discuss data automata: the study of emptiness of data automata is a pretext to discuss an important decidability result about vector addition systems; there is a nontrivial result that data automata generalise nondeterministic register automata; and data automata have a natural correspondence to two variable logic over data words[1].

2.1 Data automata

In the definition of a data automaton, we use a nondeterministic transducer over words without data, so we begin by describing this transducer.

Letter-to-letter transductions. Suppose that Γ and Σ are finite alphabets (no atoms involved). Consider a nondeterministic finite automaton with input alphabet Σ where every transition is labelled by a letter of Γ. We view this automaton as a device which inputs a word over Σ, and outputs all possible words that label accepting runs. In other words, the semantics of such an automaton is a binary relation

$$R \subseteq \Sigma^* \times \Gamma^*.$$

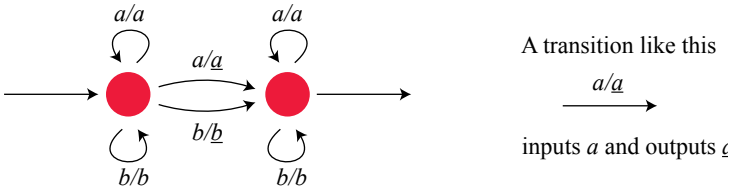
A relation is called a *nondeterministic letter-to-letter transduction* if it can be described this way. Such a relation will only contain pairs of words of same length.

[1] The model of data automata and its application to logic is from Bojańczyk et al. (2011). Tree generalisations of two variable logic on data words and data automata were discussed in Bojańczyk et al. (2009) and Jacquemard et al. (2016); these problems are connected to *branching vector addition systems*, see Göller et al. (2016) and the references therein.

Example 1. Consider the set of pairs

$$(w, v) \in \{a, b\}^* \times \{a, b, \underline{a}, \underline{b}\}^*$$

such that v is obtained from w by underlining exactly one position. This relation is realised by the following automaton:



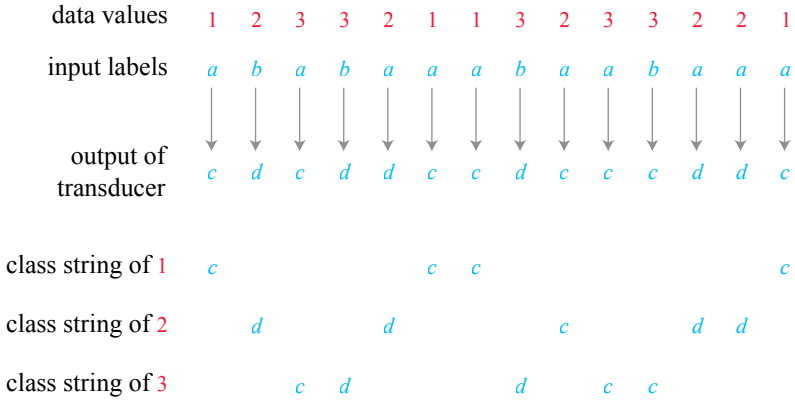
□

Data automata. We are now ready to define data automata.

Definition 2.1. The syntax of a data automaton is given by:

- finite input and work alphabets Σ and Γ;
- a nondeterministic letter-to-letter transduction $R \subseteq \Sigma^* \times \Gamma^*$;
- a regular language $L \subseteq \Gamma^*$ called the *class condition*.

A data automaton is used to accept or reject data words in $(\Sigma \times \mathbb{A})^*$. For a data word, define a *class* to be a maximal set of positions with the same data value, and define a *class string* to be a sequence in Σ^* obtained by taking some class and reading all of its labels from left to right. A data automaton accepts a data word if the sequence of labels can be transformed by the transducer so that in the resulting data word in $(\Gamma \times \mathbb{A})^*$, all class strings are in L. Here is a picture:



The language recognised by a data automaton is the set of accepted data words.

Example 2. A data automaton can check that every data value appears exactly twice. The transducer is the identity, while the class condition contains all words of length exactly two. □

Example 3. A data automaton can check that some data value appears an even number of times. The transducer underlines exactly one position, as in Example 1. The class condition says that if a word contains an underlined position, then it has even length. □

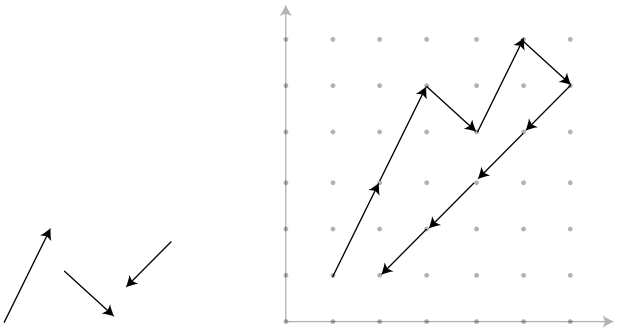
Example 4. Suppose that the labels are $\{a, b\}$. Consider a data automaton where the transducer is the identity and the class condition is $\{ab, ba\}$. The language recognised by this data automaton, after erasing data values, is the set of words in $\{a, b\}^*$ where a occurs the same number of times as b. The same idea would work for labels $\{a, b, c\}$. □

Vector addition systems. We will prove that emptiness for data automata is decidable, because it reduces to reachability problem for vector addition systems, which is known to be decidable, although highly challenging.

Definition 2.2 (Vector Addition System)**.** A *vector addition system* is any finite set $\delta \subseteq \mathbb{Z}^d$ of integer vectors with a common dimension d, called the *transitions*. A run of a vector addition system is a sequence

$$v_0, v_1, \ldots, v_n \in \mathbb{N}^d \qquad \text{such that } v_i - v_{i-1} \in \delta \text{ for every } i \in \{1, \ldots, n\}.$$

Note that all vectors in the run must be non-negative on all coordinates, even though δ can use negative numbers. Here is a picture in dimension two:



transitions a run

The *reachability problem* for vector addition systems is to decide, given two vectors of natural numbers, if there exists a run that begins in the first vector and ends in the second one. The following famous result uses one of the more difficult decidability proofs in computer science[2]; this proof is not included in these lecture notes.

Theorem 2.3. *The reachability problem for vector addition systems is decidable.*

A vector addition system can be used as a language recogniser in the following way. Define a *multicounter automaton* to be a vector addition system $\delta \subseteq \mathbb{Z}^d$ together with designated initial and final vectors in $\mathbb{N}^d$, plus an output function $\delta \to \Sigma^*$ which associates to each transition a word that is produced by that transition. A word in Σ^* is accepted if there exists a run from the initial to the final vector, which produces the word after applying the output function to each transition. Emptiness for multicounter automata is the same problem as reachability for vector addition systems, and is therefore decidable.

Example 5. Here is a multicounter automaton which recognises the set of words over $\{a, b\}$ where the number of a's is equal to the number of b's. We use two counter names a, b. When reading an a letter, we can either increment the a counter, or decrement the b counter. When reading a b, we can either increment the b counter, or decrement the a counter. The initial vector is $(0, 0)$ and the final vector is also $(0, 0)$. □

Lemma 2.4. *Every regular language is recognised by a multicounter automaton.*

Proof Consider a regular language recognised by a nondeterministic automaton with states which are numbers $\{1, \ldots, n\}$. By using ε-transitions, we can assume that the automaton has one initial state, one accepting state, and these are not the same state. To simulate the automaton, we use a multicounter automaton with n counters, where state q is encoded by a vector

$$(0, \ldots, 0, \underbrace{1}_{q\text{-th counter}}, 0, \ldots, 0).$$

A transition which goes from q to p is represented by the integer vector which decrements q and increments p. □

[2] The decidability of reachability for vector addition systems, i.e. Theorem 2.3, was originally shown by Mayr in Mayr (1984), other proofs include Kosaraju (1982) and Leroux (2010). The current best lower bound for the problem is a tower of exponentials Czerwiński et al. (2018) and the current best upper bound is primitive recursive for fixed dimension Leroux and Schmitz (2019).

For a language $L \subseteq \Sigma^*$, define $\mathsf{shuffle}L$ to be all words in Σ^* which can be labelled with data values so that all class strings are in L.

Lemma 2.5. *If L is regular, then $\mathsf{shuffle}L$ is recognised by a multicounter automaton.*

Proof Suppose that L is recognised by a deterministic automaton with states Q. Without loss of generality assume that this automaton has no self-loops, i.e. transitions which have the same source and target state. We define a multicounter automaton with one counter per state from Q. The initial and final vectors are the same, namely the zero vector. For every transition $q \xrightarrow{a} p$ of the automaton recognising L, we create a transition in the multicounter automaton which reads a, decrements counter q and increments counter p. If q is the initial state, then we also create a transition which reads a and only increments counter p. If p is a final state, then we also create a transition which reads a and only decrements counter q. □

Emptiness for data automata. In the proof of the following theorem, we show that emptiness for data automata is the same thing as emptiness for multicounter automata, and therefore the same thing as reachability for vector addition systems.

Theorem 2.6. *Emptiness is decidable for data automata.*

Proof A data automaton is nonempty if and only if there exists a word over the work alphabet which is a possible output of the transducer, and such that the word can be labelled by data values so that every class string is in the class condition of the data automaton. The set of possible outputs of the transducer is easily seen to be a regular language (a nondeterministic automaton can guess the input and the run of the transducer on it). Using the shuffle terminology, we have just shown that emptiness of data automata reduces to the following problem: given regular languages $L, K \subseteq \Gamma^*$ decide if

$$K \cap \mathsf{shuffle}L = \emptyset.$$

The language K is recognised by a multicounter automaton thanks to Lemma 2.4, while $\mathsf{shuffle}L$ is recognised by a multicounter automaton thanks to Lemma 2.5. Languages recognised by multicounter automata are easily seen to be closed under intersection, and therefore the problem above boils down to testing nonemptiness for an effectively obtained multicounter automaton, which is decidable thanks to Theorem 2.3. □

Exercises

Exercise 29. Theorem 2.6 reduces emptiness for data automata to reachability for vector addition systems. Show a converse reduction.

Exercise 30. Show that languages recognised by data automata are not closed under Kleene star.

Exercise 31. Show that emptiness is decidable for vector addition systems if the definition of a run is modified so that the intermediate vectors are allowed to use negative coordinates, i.e. the intermediate coordinates are vectors in $\mathbb{Z}^d$.

2.2 Two-variable first-order logic on data words

At the end of Section 1.3, we described a logic for data words that had decidable satisfiability by virtue of a translation into alternating automata. We now do the same thing for data automata: we describe a logic for data words which has decidable satisfiability by virtue of a translation into data automata. The logic is a fragment of first-order logic, i.e. it uses variables and quantifiers, as opposed to the temporal logic in Section 1.3, which had a variable-free syntax. We already saw this logic in Exercise 10, but we now describe it in more detail.

To describe properties of data words using first-order logic, we view a data word with labels from a finite set Σ as a relational structure, where the universe is the positions in the data word, and which is equipped with the following relations:

$$\underbrace{x < y}_{\substack{\text{position } x \\ \text{is before} \\ \text{position } y}} \qquad \underbrace{x = y + 1}_{\substack{\text{position } x \text{ is} \\ \text{the successor} \\ \text{of position } y}} \qquad \underbrace{x \sim y}_{\substack{\text{positions } x, y \\ \text{have the same} \\ \text{data value}}} \qquad \underbrace{a(x).}_{\substack{\text{position } x \\ \text{has label } a \in \Sigma}}$$

Using the above representation of data words as relational structures, we can use first-order logic to express properties of data words. For example,

$$\forall x \, \forall y \; x = y + 1 \Rightarrow x \nsim y$$

says that every two consecutive positions have different data values, while

$$\forall x \, \forall y \; a(x) \wedge a(y) \Rightarrow x \sim y$$

says that all positions with label a have the same data value. The successor relation can be defined using first-order logic in terms of order, because

$$x = y + 1 \quad \text{iff} \quad y < x \wedge \neg \exists z \; (y < z \wedge z < x).$$

However, we will be mainly interested in the fragment of first-order logic which can only use two variables, and in this fragment the definition of successor in terms of order no longer works.

The idea of representing words with relational structures, and then using logic to express their properties, dates back to the famous result of of Büchi, Elgot and Trakhtenbrot[3], which says that – without data values – every formula of first-order logic (even monadic second-order logic) can be translated into an equivalent finite automaton. Does this work for data words? As shown in Exercise 10, the satisfiability for first-order logic over data words is undecidable, and the proof only needs three variables. The rest of this section is devoted to showing that the two variable fragment is decidable.

Theorem 2.7. *The following problem is decidable.*

- **Input.** *A sentence of first-order logic which uses two variables and relations*

$$x < y \quad x = y + 1 \quad x \sim y \quad a(x).$$

- **Output.** *Is the sentence true in some finite word?*

Before proving the theorem, let us give some more examples which illustrate the power of the two variable fragment. The sentence

$$\forall x \, \forall y \; x \neq y \Rightarrow x \nsim y$$

says that all positions have different data values. This property is not recognised by any nondeterministic register automaton. Another example is

$$\forall x \; a(x) \Rightarrow \exists y \; (y < x \wedge y \sim x),$$

which says that no class string begins with a. This property is not recognised by any alternating one register automaton without guessing, which can be proved using the same reasoning as in Exercise 12.

An important feature of two variable first-order logic is reusing variables. This allows us to say that there are at least four positions, without assigning four variable names to those positions:

$$\exists x \; \overbrace{(\exists y \; y > x \wedge \underbrace{(\exists x \; x > y \wedge (\exists y \; y > x)}_{\text{there are at least two positions after } y})}^{\text{there are at least three positions after } x}.$$

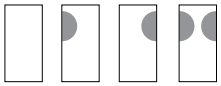
To prove Theorem 2.7, we will show that for every formula of two variable first-order logic there is a data automaton which accepts the same data words. Before we do this, we show how data automata can deal with the successor relation.

[3] See Thomas (1997) for an introduction to the topic.

Recognising equality with successors

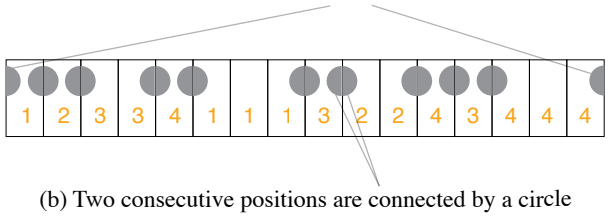
In two variable first-order logic, we have a successor predicate, which can be used to compare data values in consecutive positions, e.g. by saying "every two consecutive positions have different data values". On the way to our final result, which is a translation from two variable first-order logic into data automata, we first show how data automata can compare data values in consecutive positions. This is done in the following lemma.

Lemma 2.8. *There is a data automaton which accepts a data word over labels*



if and only if it satisfies conditions (a) and (b) described below:

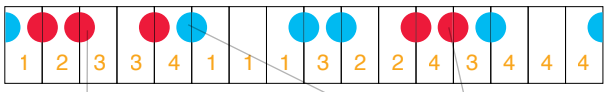
(a) All circles are full, except for an opening semicircle at the beginning and a closing semicircle at the end.



(b) Two consecutive positions are connected by a circle if and only if they have different data values.

Proof For a data word over the alphabet from the statement of the lemma, define a *consistent colouring* to be a colouring of the semicircles with two colours as in the following picture:

(1) On an edge connecting two consecutive positions, there is either a monochromatic circle, or no circle at all.

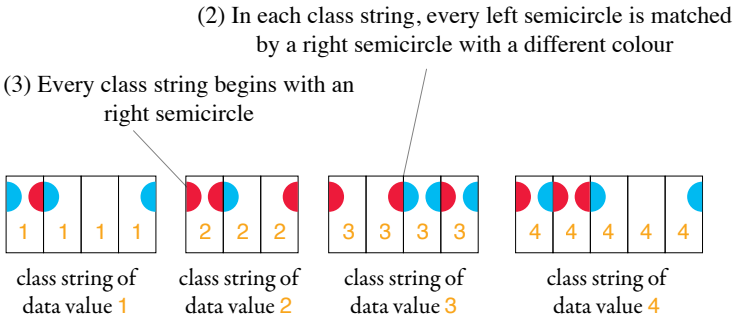


(2) Circles are not monochromatic for consecutive appearances of the same atom.

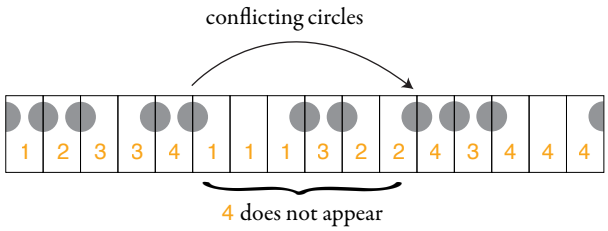
(3) The first appearance of each atom has a semicircle on its left sideO

To prove the lemma, we show that there is a consistent colouring if and only if the word belongs to the language from the statement of the lemma, and furthermore a data automaton can check if there is a consistent colouring.

- We first show that data automaton can check if there is a consistent colouring. The transducer chooses the colours, and checks if condition (1) in the definition of a consistent colouring is satisfied. Conditions (2) and (3) are checked by looking at the class strings, as explained in the following picture:



- We now show that every data word in the language admits a consistent colouring. Consider a data word in the language. We need to colour each grey circle (i.e. pairs of consecutive positions with different data values) with a single colour so that condition (2) is satisfied. We say that two circles are in conflict if the left half of the left circle has the same data value as the right half of the right circle, and this data value does not appear in between, as in the following picture:



Condition (2) in the definition of consistency says that conflicting circles cannot have the same colour. If we view the conflict relation as a directed graph on circles, with arrows pointing from left to right, then this graph is a disjoint union of paths, i.e. every circle has in-degree and out-degree at

most one. Such a graph can be coloured with two colours so that edges have endpoints with different colours.

- Finally, we show that if a data word can be coloured consistently, then it is in the language. Suppose that a data word can be coloured consistently. We need to show that there is a circle connecting two consecutive positions if and only if these positions have different data values.

  For the left-to-right implication, consider a circle connecting x and its successor. By condition (2) the next position in the class of x has its left side coloured with a different colour, and hence the next position in the class of x cannot be the successor of x, because all circles are monochromatic by condition (1).

  We prove the right-to-left implication by doing an inductive left-to-right pass. Consider consecutive positions x and $x+1$ with different data values. To prove that they are connected by a circle, by condition (1) it suffices to prove that the left side of $x+1$ has a semicircle. If $x+1$ is the first position in its class, then it has a semicircle on its left by condition (3), otherwise $x+1$ has a previous position in its class and then we use the induction assumption.

□

The ideas in the above proof can be extended to show that data automata are more expressive than nondeterministic register automata, see Exercise 36.

Two-variable logic

We now use the results about successor to complete the proof of Theorem 2.7, about satisfiability for two variable first-order logic over data words. We show that every formula can be converted into a data automaton that accepts the same data words. We begin by converting the formula into a normal form.

Normal form. We say that a formula of two-variable logic is in *normal form*[4] if every subformula with one free variable is a Boolean combination of formulas of the form

$$\varphi(x) = \exists y\, (\alpha \wedge \beta \wedge \underbrace{\psi(y)}_{\text{normal form}})$$

where α and β are quantifier-free constraints of the following kinds:

$$\begin{array}{lll} \alpha & \in & \{y < x-1, \quad y = x-1, \quad y = x, \quad y = x+1, \quad y > x+1\} \\ \beta & \in & \{x \sim y, \quad x \nsim y\}. \end{array}$$

[4] The normalisation process in Lemma 2.9 is related to what is known as Scott Normal Form for two variable first-order logic logic, see Scott (1962).

Lemma 2.9. *Every formula is equivalent to one in normal form.*

Proof Induction on formula size. Since formulas in normal form are closed under Boolean combinations, it is enough to show how to normalise a formula that uses a quantifier:

$$\varphi(x) = \exists y\, \psi(x, y).$$

For every positions x and y in a data word, there is some choice of α and β as in the definition of normal form which make $\alpha \wedge \beta$ true. Therefore, $\varphi(x)$ is equivalent to

$$\bigvee_{\alpha,\beta} \exists y\, \alpha \wedge \beta \wedge \psi(x, y),$$

where α and β range over formulas as in the definition of normal form. To finish the proof of the lemma, it is enough to show that for every fixed α, β the formula

$$\exists y\, \alpha \wedge \beta \wedge \psi(x, y)$$

is equivalent to one in normal form. Like any formula with two variables, the formula $\psi(x, y)$ is a Boolean combination of formulas which are either binary relations from the vocabulary (e.g. $x \sim y$ or $x < y$), or have one free variable. Each binary relation is either implied by or contradictory with a given choice of $\alpha \wedge \beta$. Therefore, once we have fixed α and β, each of the binary relations in the Boolean combination constituting $\psi(x, y)$ can be replaced by either true or false, leading to a Boolean combination of formulas with one free variable. To the formulas with one free variable, we can apply the induction assumption. □

From a formula to a data automaton. To complete the proof of Theorem 2.7, we convert below every normal form sentence φ into an equivalent data automaton. The idea is that the data automaton guesses, for each subformula with one free variable, what positions satisfy it, and then checks in parallel, for each subformula, if its guessed positions are consistent with those of its immediate subformulas. Let Σ be the labels used in φ and let Γ be the set of subformulas of φ that have exactly one free variable. For a data word w with labels Σ – i.e. for a potential model of φ – define its *annotation* w^φ to be the data word with labels $\Sigma \times \mathsf{P}\Gamma$ that is obtained from w by extending the label of each position with the set of formulas from Γ that are satisfied when the unique free variable is set to that position.

Example 2.10. Consider the formula

$$\varphi = \forall x \, (\underbrace{\overbrace{\exists y \; y = x + 1 \wedge x \sim y \wedge a(y)}^{\phi_2(x)} \vee \overbrace{\exists y \; y = x - 1 \wedge x \nsim y \wedge b(y).}^{\phi_3(x)}}_{\phi_1(x)})$$

The set Γ is

$$\{\phi_1(x), \phi_2(x), \phi_3(x), a(y), b(y)\}.$$

Although each formula in Γ has one free variable, this free variable is sometimes x and sometimes y. Here is an example of annotation for φ:

labels of w^φ	$\phi_1(x)$ $\phi_2(x)$ $a(y)$	$a(y)$	$b(y)$	$\phi_1(x)$ $\phi_3(x)$ $a(y)$	$\phi_1(x)$ $\phi_2(x)$ $b(y)$	$a(y)$
input labels	a	a	b	a	b	a
data values	1	1	2	3	2	2

Lemma 2.11. *The following language is recognised by a data automaton*

$$\{w^\varphi : w \text{ is a data word with labels } \Sigma\}.$$

Before proving the above lemma, let us use it to complete the proof of Theorem 2.7. Every sentence φ of two variable logic is a positive Boolean combination of quantified formulas

$$\exists x \, \phi(y) \qquad \text{or} \qquad \forall x \, \phi(x)$$

Such a quantified formula is true in w if and only if the annotation has $\phi(y)$ in the label of some (respectively, every) position. This can be checked by a data automaton running on the annotation w^φ, without even looking at the data values. Since languages recognised by data automata are closed under positive Boolean combinations, it follows from Lemma 2.11 that there is a data automaton which recognises annotations of data words that satisfy φ. If the data automaton is only given w and not its annotation w^φ, it can use nondeterminism of the transducer to guess the annotation, and therefore the language of φ is also recognised by a data automaton. Together with the decidability of emptiness for data automata, we get Theorem 2.7. It remains to prove the lemma.

Proof of Lemma 2.11 Define an *annotation candidate* to be a data word with labels in $\Sigma \times \mathsf{P}\Gamma$. We say that an annotation candidate is correct for a subformula $\psi \in \Gamma$ if ψ belongs to the label of exactly those positions which satisfy ψ. By definition, w^φ is the unique annotation candidate which is correct for all subformulas and projects to w if the annotation is ignored. To prove the lemma, we will show that for every ψ there is a data automaton $\mathcal{A}_\psi$ which recognises the following property of annotation candidates:

(*) If the input annotation candidate is correct for all proper subformulas of ψ, then $\mathcal{A}_\psi$ accepts if and only if the input is also correct for ψ.

In (*), there are no requirements for the behaviour of $\mathcal{A}_\psi$ on annotation candidates that are not correct for proper subformulas of ψ. An annotation candidate is of the form w^φ if and only if it is accepted by all automata $\mathcal{A}_\psi$, with ψ ranging over subformulas with one free variable. Since languages recognised by data automata are closed under intersection, the lemma will follow once we construct the automata $\mathcal{A}_\psi$.

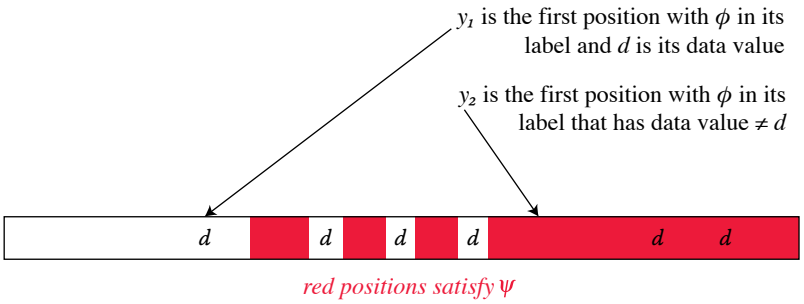
If ψ is a conjunction $\psi_1 \wedge \psi_2$, then $\mathcal{A}_\psi$ simply checks if ψ is used in the labels of exactly those positions that have both ψ_1 and ψ_2 in their labels. The same idea words for $\vee$ and $\neg$, while formulas $a(x)$ are checked by comparing the annotation with the label in the underlying data word. The interesting case is when ψ is a quantified formula

$$\psi(x) = \exists y\ \alpha \wedge \beta \wedge \phi(y).$$

where α and β are as in the definition of normal form. Without loss of generality we assume that the quantifier is existential, since the formulas can use negation.

If β is $x \sim y$, then $\psi(x)$ corresponds to a regular property of the class strings, which is checked by the class condition. To know which positions in the class string are consecutive in the input word (i.e. are not separated by positions from outside the class), we use the labelling from Lemma 2.8. If α says that y is the successor or predecessor of x, then we can also use Lemma 2.8 to check if the input word is correct for ψ.

The remaining case is when β is $x \nsim y$ and α is one of $y < x-1$ or $y > x+1$. By symmetry, we assume that α is $y < x-1$. In words, $\psi(x)$ says that there exits a position $y < x-1$ which has a different data value than x and $\phi(y)$ in its label. The general idea is that there are two best candidates for this position y, namely y_1 and y_2 as described in the following picture:



The positions y_1 and y_2 might be undefined, e.g. y_1 is undefined if no position has ϕ in its label, and y_2 is undefined if at most one data value is used for positions with ϕ in their label. A short analysis reveals that the label ψ should be found in the label of positions x which satisfy:

$$x \nsim y_1 \wedge y_1 < x - 1 \qquad \vee \qquad x \sim y_1 \wedge y_2 < x - 1.$$

The positions x satisfying the above criteria can be identified using a finite automaton, assuming that the positions y_1 and y_2, their successors, and the are marked. These positions can be guessed and checked by a data automaton, with special colours used to mark the classes of y_1 and y_2. □

Exercises

Exercise 32. Let $k \in \{0, 1, \ldots\}$. Show that there is a data automaton which recognises the set of data words where each position x is labelled by the set of those $i \in \{0, 1, \ldots, k\}$ such that x and $x + i$ have the same data value.

Exercise 33. Recall the notion of class string in the definition of a data automaton, where the positions from outside the class are erased, as in this picture:

data values	1	2	3	3	2	1	1	3	2	3	3	2	2	1	1
labels	*a*	*b*	*a*	*b*	*a*	*a*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*	*a*
class string of 1	*a*					*a*	*a*							*a*	*a*

Consider an alternative definition of class string, where the positions from outside are replaced by question marks, like this:

data values	1	2	3	3	2	1	1	3	2	3	3	2	2	1	1
labels	*a*	*b*	*a*	*b*	*a*	*a*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*	*a*
class string of 1	*a*	*?*	*?*	*?*	*?*	*a*	*a*	*?*	*?*	*?*	*?*	*?*	*?*	*a*	*a*

Show that data automata defined with this alternative notion of class string have the same expressive power as the original model of data automata[5].

Exercise 34. In the spirit of the previous exercise, consider yet another definition of class string, where the positions from outside the class are coloured red, like this:

data values	1	2	3	3	2	1	1	3	2	3	3	2	2	1	1
labels	*a*	*b*	*a*	*b*	*a*	*a*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*	*a*
class string of 1	*a*	*b*	*a*	*b*	*a*	*a*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*	*a*

Show that data automata defined with this alternative definition are strictly more expressive than the original model of data automata[6].

Exercise 35. Consider a sequence of data values where every position is labelled by a subset of {cut, chosen}. We say a position is chosen if its label contains "chosen" and we say that two positions $x < y$ are in *the same interval* if "cut" does not appear in the labels of positions $x + 1, \ldots, y$. Show that there is a data automaton recognising the data words which satisfy the following two conditions:

- all chosen positions in the same interval have the same data value; and
- there is no non-chosen position which has the same data value as some chosen position in the same interval.

Exercise 36. Show that every language recognised by a nondeterministic register automaton is also recognised by a data automaton[7]. (Hint: use the previous exercise.)

[5] This exercise is based on (Alur et al., 2009, Theorem 2).
[6] This model is called *class automata*, and is studied in Bojańczyk and Lasota (2012a); Bárány et al. (2012).
[7] This exercise is based on (Björklund and Schwentick, 2010, Theorem 4.1).

Exercise 37. Extend two variable logic, as considered in Theorem 2.7, with a binary relation $s(x, y)$ called *class successor* which holds if y is the least position $y > x$ that has the same value as x. Show that the resulting logic is still decidable.

Exercise 38. Show that satisfiability for the logic in Exercise 37 is at least as hard as emptiness for data automata.

Exercise 39. Show that satisfiability for two variable logic becomes undecidable if the data values are ordered, and there is a predicate $x \preceq y$ for testing the order on these data values.

Exercise 40. Consider the following alternating automaton based on the modal logic[8]. The automaton has a set of states Q, an initial state q_0, a partition of states into universal and existential, and a finite transition relation

$$\delta \subseteq Q \times (\text{formulas of two variable logic with free variables } x, y) \times Q$$

The automaton accepts a data word $w \in (\Sigma \times \mathbb{A})^*$ if player $\exists$ has a winning strategy in the following game played by players $\exists$ and $\forall$. The game begins in the first position and the initial state. If the game is in position x and state q, then the player who owns q chooses a transition $(q, \varphi(x, y), p) \in \delta$ and a position y such that $\varphi(x, y)$ holds. If there is no such transition or position, the player who owns q loses immediately. Otherwise, the game proceeds to state p and position y. If the game lasts forever, player $\forall$ wins. Show that such an automaton can recognise the language

$$\{a_1 \cdots a_n a_1 \cdots a_n : a_1, \ldots, a_n \text{ are distinct data values}\}.$$

Exercise 41. Show that the automaton model in Exercise 40 has undecidable emptiness[9]. Show that if infinite plays are won by $\exists$ then emptiness becomes decidable[10].

[8] Similar automata models are considered in Manuel et al. (2016) and Colcombet and Manuel (2014)
[9] Undecidability is based on (Manuel et al., 2016, Proposition 9).
[10] Decidability is based on (Colcombet and Manuel, 2014, Theorem 3.8).

PART TWO

ORBIT-FINITE SETS

In the previous part, we discussed data words and their automata. In this part, we move to a more abstract and general setting, where data words turn out to be words over a "finite" alphabet, with an appropriate notion of finiteness, and register automata turn out to be finite automata.

3

Sets with atoms and orbit-finiteness

This chapter introduces two fundamental concepts studied in this book:

- **Section 3.1: sets with atoms**[1]**.** Roughly speaking, a set with atoms is any object that can be built using atoms. Examples include: the set of all atoms, the state space of a register automaton, or the set of all data words.
- **Section 3.2: orbit-finiteness.** One can introduce a notion of finiteness that is more relaxed than the usual one: a set with atoms is *orbit-finite* if it has finitely many elements up to renaming atoms. Examples of orbit-finite sets include the set of all atoms and the state space of a register automaton, but not the set of all data words. In the world of sets with atoms, orbit-finite sets play the role of finite sets.

Atoms as a logical structure. In Part I of this book, about automata for data words, we assumed that the atoms are equipped with equality only. Much of the theory that we develop starting with this chapter works for atoms with more structure, e.g. an ordering. To model this additional structure, we use the notion of "structure" in the sense of model theory: a universe together with some relations and functions. Examples of structures representing atoms that

[1] Sets with atoms (also known as *ur-elements*) have their origin in the work of Frankel in 1922, further developed by Mostowski in the 1930s. The original application was to have a model of set theory which violates the axiom of choice, and other axioms. In computer science, atoms (with equality only) were rediscovered by Gabbay and Pitts in Gabbay and Pitts (2002), under the name *nominal sets*, as a formalism for modelling name binding. Since then, nominal sets have become a lively topic in semantics, see e.g. the book of Pitts Pitts (2013). Nominal sets were also independently rediscovered by the concurrency community, as a basis for syntax-free models of name-passing process calculi, see Montanari and Pistore (1999, 2005).

will appear in this text are:

- $(\mathbb{N}, =)$ natural numbers (or any countably infinite set) with equality;
- $(\mathbb{Q}, \leq)$ the rational numbers with their order.

We use the name "equality atoms" for the first structure, which corresponds to the data values in the first part about data words. For the equality atoms, as well as the ordered rational numbers, the notions of orbit-finiteness will be useful. Non-examples, i.e. structures where the notions such as orbit-finiteness will not be useful, are:

- $(\mathbb{Z}, <)$ the integers with their order;
- $(\mathbb{Z}, +)$ the integers with addition.

3.1 Sets with atoms

Roughly speaking, a set with atoms is a set which contains atoms and simpler sets with atoms; although not every object built this way is going to be considered a set with atoms. The intuitive idea is that a set with atoms must be built using only the structure given by the atoms (i.e. relations and functions from the vocabulary of the atoms), and finitely many constants referring to specific atoms. For example, in the equality atoms[2] the set of even numbered atoms $\{\underline{0}, \underline{2}, \underline{4}, \ldots\}$ would *not* be a set with atoms, because a definition of this set would need to either explicitly mention infinitely many atoms, or refer to the notion of "even-numbered" which does not exist in the structure. The intuitive idea of finitely many constants is formalised below using actions of atom automorphisms.

Definition 3.1 (Sets with atoms)**.** Let $\mathbb{A}$ be a logical structure, whose elements are called atoms.

- **The cumulative hierarchy.** The *cumulative hierarchy over* $\mathbb{A}$ is a hierarchy of sets indexed by ranks that are ordinal numbers. The empty set is the unique set of rank 0. For an ordinal number $\alpha > 0$, a set of rank α is any set whose every element is an atom or a set of rank $< \alpha$.

[2] Under the equality atoms, or the rational number atoms, a natural number like 2 can be interpreted in two ways: as an atom, or as the set $\{\emptyset, \{\emptyset\}\}$ which represents 2 according to the Von Neumann numeral encoding. To avoid this confusion, we use an underlined number $\underline{2}$ for the first meaning, and 2 for the second one.

- **Action of atom automorphisms.** Let π be an automorphism of the atoms, i.e. a permutation of the universe of $\mathbb{A}$ which preserves all predicates and functions in the structure. We inductively extend π from atoms to sets in the cumulative hierarchy over $\mathbb{A}$ by defining $\pi(x)$ to be $\{\pi(y) : y \in x\}$.
- **Supports.** If an atom automorphism fixes a tuple of atoms $\bar{a} = (a_1, \ldots, a_n)$ (i.e. maps the tuple to itself), then it is called an $\bar{a}$-automorphism. If x is in the cumulative hierarchy over $\mathbb{A}$ and $\bar{a}$ is a tuple of atoms, then we say that $\bar{a}$ is a *support* of x if

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

  i.e. x is fixed by every $\bar{a}$-automorphism[3] . We say that x is *finitely supported* if it is supported by some finite tuple of atoms.
- **Set with atoms.** A *set with atoms over* $\mathbb{A}$ is any x in the cumulative hierarchy which is finitely supported, has only finitely supported elements, and so on until atoms are reached. We write $\mathsf{set}\mathbb{A}$ for all sets with atoms over $\mathbb{A}$.

The rest of Section 3.1 is devoted to exercises and examples which illustrate the above definitions. An intuitive description of the support of a set is that the support consists of the atoms that are "hard-coded" into the definition of the set. The support of a set with atoms is not unique, because supports are closed under adding atoms. (For some atoms such as $(\mathbb{N}, =)$ or $(\mathbb{Q}, <)$, a canonical least support can be found, see Chapter 6.) A set with empty support is called *equivariant*. Intuitively speaking, an equivariant set is one which can be defined without referring to any specific atoms.

Example 3.2. Let $\mathbb{A}$ be the equality atoms $(\mathbb{N}, =)$ and consider the set

$$\{a : \text{ for } a \in \mathbb{A} \text{ such that } a \neq \underline{2}\}.$$

This set is supported by the atom $\underline{2}$, because any $\underline{2}$-automorphism will map the set to itself, although it might rearrange its elements. The set is not equivariant, so $\underline{2}$ is a minimal support, actually it is the least finite support.

Example 3.3. Let the atoms $\mathbb{A}$ be the ordered rational numbers $(\mathbb{Q}, <)$ and consider the set of open intervals that contain the atom $\underline{2}$:

$$\{\{c : \text{ for } c \in \mathbb{A} \text{ such that } a < c < b\} : \text{ for } a, b \in \mathbb{A} \text{ such that } a < \underline{2} < b\}.$$

[3] The order or repetition of atoms in the tuple $\bar{a}$ is not relevant for the support, i.e. only the set of atoms that appear in the tuple matters. For this reason, some authors use a set of atoms as a support, instead of a tuple of atoms. We use tuples so that we can distinguish between an $\{a_1, a_2\}$-automorphism and an (a_1, a_2)-automorphism. The former can swap a_1 and a_2, while the latter needs to fix both a_1 and a_2.

This set is supported by $\underline{2}$. An element of this set is the open interval

$$\{c : \text{ for } c \in \mathbb{A} \text{ such that } \underline{0} < c < \underline{3}\},$$

which is a set that is supported by the atom tuple $(\underline{0}, \underline{3})$.

Sets are – no surprises here – a natural choice for foundations of mathematics. In particular, using sets we can simulate data structures such as pairs, tuples, etc. To define pairs, we use Kuratowski pairing:

$$(x, y) \quad \stackrel{\text{def}}{=} \quad \{\{x\}, \{x, y\}\}.$$

It is easy to see that if x is supported by a tuple of atoms $\bar{a}$ and y is supported by a tuple of atoms $\bar{b}$, then the pair (x, y), in the Kuratowski sense defined above, is supported by the tuple $\bar{a}\bar{b}$. In particular, a pair of finitely supported objects is also finitely supported, and therefore sets with atoms are closed under pairing.

Example 3.4. Regardless of the choice of atoms, the set $\mathbb{A}^*$ (defined using pairing in the natural way) is a set with atoms. It is equivariant, but its elements are typically not equivariant. For example, in the equality atoms, $\underline{12345} \in \mathbb{A}^*$ is finitely supported, but any support must include $\underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}$. In particular, $\mathbb{A}^*$ contains elements with unboundedly large supports.

Using pairs, we can define sets with atoms which are binary relations, and using binary relations, we can define sets with atoms which are functions.

Example 3.5. Consider the equality atoms. Define a *choice function for unordered pairs* to be a function

$$f : \{\{a, b\} : a, b \in \mathbb{A}\} \to \mathbb{A} \qquad \text{such that } f(\{a, b\}) \in \{a, b\} \text{ for every } \{a, b\},$$

i.e. a function which chooses an element for each unordered pair of atoms. We claim that there is no finitely supported choice function. (For this example it is crucial that the atoms have equality only. If there would be a linear order in the atoms, then max would be a choice function.) Toward a contradiction, suppose that f is a choice function with finite support $\bar{a}$. Choose two atoms b, c that do not appear in the support $\bar{a}$, and let π be the transposition which swaps b with c. By definition of supports, since π fixes $\bar{a}$, it must also fix f seen as a set of pairs, i.e. it must fix the graph of f. Therefore, the graph of f must contain both pairs

$$(\{b, c\}, b) \quad \text{and} \quad (\{b, c\}, c),$$

which contradicts the assumption that f is a function[4].

[4] This example touches on the origins of sets with atoms. In 1922, Abraham Fraenkel showed that, when the atoms have equality only, then sets with atoms:

The following example shows that finite supports are meaningless in atoms such as $(\mathbb{Z}, <)$.

Example 3.6. Consider the integer atoms $(\mathbb{Z}, <)$. For these atoms, the automorphisms are translations, i.e. functions of the form $a \mapsto a + k$ for some $k \in \mathbb{Z}$. The atom $\underline{2}$ is supported by itself, but it is also supported by $\underline{1}$, because there is only one $\underline{1}$-automorphism, namely the identity. One explanation is that $\underline{2}$ can be defined as "the smallest element after $\underline{1}$". In fact, every set of atoms is finitely supported, e.g. by $\underline{1}$, and therefore for the atoms $(\mathbb{Z}, <)$ there is no difference between a set in the cumulative hierarchy and a set with atoms. If we extend the structure $(\mathbb{Z}, <)$ by adding a constant 0, the the only automorphism is the identity, and therefore every set in the cumulative hierarchy has empty support.

An arbitrary subset of a set with atoms might not be finitely supported, and therefore sets with atoms are not closed under taking arbitrary subsets, but only under taking finitely supported subsets.

Example 3.7 (Finitely supported subsets of the equality atoms)**.** Consider the equality atoms. We show below that the finitely supported subsets of the atoms are exactly the finite and co-finite sets. It is not difficult to see that the finite and co-finite sets are finitely supported. For the converse implication, consider a set $X \subseteq \mathbb{A}$ that is neither finite nor co-finite. We will show that X cannot have finite support. Suppose that a finite tuple of atoms $\bar{a}$ is a candidate for a finite support. Since both X and its complement are infinite, there must be atoms $a \in X$ and $b \notin X$ such that both a and b do not appear in the tuple $\bar{a}$. The permutation of atoms which swaps a and b, and is the identity on other atoms, is an $\bar{a}$-automorphism. Therefore, it should fix X, but it does not.

Example 3.8 (Finitely supported subsets of ordered rational numbers)**.** Consider the atoms $(\mathbb{Q}, <)$. In this case, the automorphisms are order preserving bijections. We claim that the finitely supported subsets of atoms are exactly finite unions of intervals. Consider a set X of atoms 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

- fail the axiom of choice, as shown in this exercise, but
- satisfy axioms similar to the Zermelo-Fraenkel axioms of set theory.

  The axioms satisfied by sets with atoms are not the real Zermelo-Fraenkel axioms, e.g. extensionality fails because every atom has the same elements as the empty set. The independence of the axiom of choice from the real Zermelo-Fraenkel axioms had to wait for Cohen and forcing. See Bell (2019) for a discussion of this topic.

of the order. There is an $\bar{a}$-automorphism 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 Examples 3.7 and 3.8, the finitely supported subsets of the atoms coincide with subsets of the atoms that can be defined by quantifier-free formulas which can use constants from the atoms. The reason is that both examples of atoms are *homogeneous* structures, see Chapter 7. When the atoms are homogeneous, finitely supported relations on the atoms are exactly those that can be defined using quantifier-free formulas.

Exercises

Exercise 42. 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 43. For the equality atoms, find all equivariant binary relations on $\mathbb{A}$.

Exercise 44. For the atoms $(\mathbb{Q}, <)$, find all equivariant binary relations on $\mathbb{A}$.

Exercise 45. Show that a function $f : X \to Y$ is supported by a tuple of atoms $\bar{a}$ if and only if the following diagram commutes for every $\bar{a}$-automorphism π:

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle\pi}\downarrow & & \downarrow{\scriptstyle\pi} \\ X & \xrightarrow[f]{} & Y \end{array}$$

Exercise 46. Consider the equality atoms. Show a finitely supported graph, which admits a two-colouring that is not finitely supported, but does not admit any finitely supported two-colouring.

Exercise 47. Consider the equality atoms. Show that for every finitely supported partial order $<$ on $\mathbb{A}$, all atoms outside the support are incomparable.

Exercise 48. Consider the atoms $(\mathbb{Q}, <)$. Show that there is no finitely supported well-founded total order on $\mathbb{A}$.

Exercise 49. 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.

3.2 Orbit-finiteness

Roughly speaking, orbit-finite sets are sets like

$$\{(a, b, c) \in \mathbb{A}^3 : a \neq \underline{2} \text{ or } b = \underline{1}\},$$

or the state space of a nondeterministic register automaton, which have finitely many elements up to atom automorphisms.

The definition of orbit-finiteness[5] makes sense only for certain atom structures, namely the oligomorphic ones, so we begin by defining oligomorphism.

Definition 3.9. A structure $\mathbb{A}$ is called *oligomorphic*[6] if for every $n \in \{0, 1, \ldots\}$, the structure $\mathbb{A}^n$ has finitely many elements up to automorphisms of $\mathbb{A}$. More precisely, for every n, the equivalence relation on n-tuples of atoms 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 3.10. The equality atoms $(\mathbb{N}, =)$ are oligomorphic. Two n-tuples of atoms are equivalent up to automorphisms if and only if they have the same equality type. The number of equality types is the same as the number of partitions of $\{1, \ldots, n\}$, i.e. the n-th Bell number. For similar reasons, the ordered rational numbers $(\mathbb{Q}, <)$ are oligomorphic: n-tuples of atoms are equivalent up to automorphisms if and only if they have the same order type.

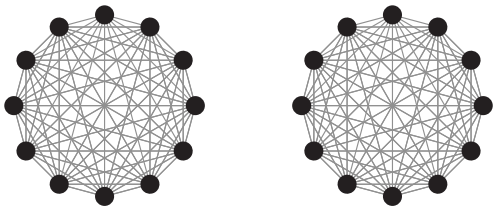
[5] To the author's best knowledge, the notion of orbit finiteness was first introduced in Bojańczyk (2011), which studied orbit-finite monoids as recognisers of languages of data words.

[6] 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 in Chapter 4, which discusses how orbit-finite sets can be represented using formulas of first-order logic.

Example 3.11. The integers with order $(\mathbb{Z}, <)$ are not oligomorphic. An automorphism is a translation, as discussed in Example 3.6. For $n = 1$, there is only one equivalence class of integers with respect to translations, but for $n = 2$ there are infinitely many equivalence classes: the equivalence class of a pair $(a, b) \in \mathbb{Z}^2$ is determined by the difference $a - b$.

Example 3.12. Every structure with a finite universe is oligomorphic.

Example 3.13. 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

$$(a_1, \ldots, a_n) \quad \text{and} \quad (b_1, \ldots, b_n)$$

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 clique". Since there are finitely many possibilities for every choice of n, it follows that these atoms are oligomorphic.

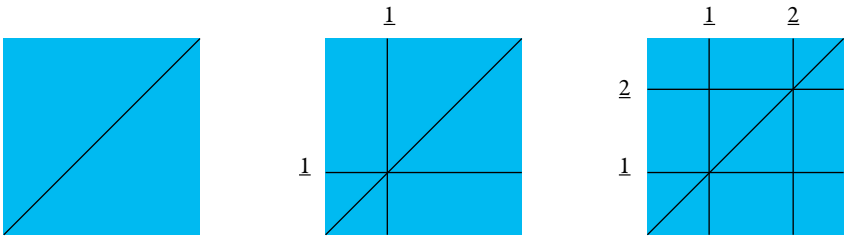
The precise definition of orbit-finiteness requires a little care to cover sets that are not equivariant, so we begin with some terminology.

Definition 3.14 (Orbits)**.** Let $\bar{a}$ be a tuple of atoms. We say X, Y (which are either atoms or sets with atoms) are $\bar{a}$-equivalent if there is some $\bar{a}$-automorphism which maps X to Y. This is an equivalence relation on sets with atoms, and its equivalence classes are called *$\bar{a}$-orbits*. When $\bar{a}$ is the empty tuple of atoms, we talk about *equivariant orbits*.

By definition of supports, a set with atoms X is supported by $\bar{a}$ if and only if

membership in x is invariant under $\bar{a}$-equivalence; in other words, X is a union, possibly infinite, of $\bar{a}$-orbits. Adding atoms to a tuple $\bar{a}$ makes it support more sets, but it makes the orbits smaller. This trade-off is illustrated in the following example.

Example 3.15. Consider the atoms $(\mathbb{Q}, <)$. Here is the partition of $\mathbb{A}^2$ into orbits, with supports of size 0, 1 and 2:



13 $\underline{1}$-orbits:
6 of dimension 2
6 of dimension 1
1 of dimension 0

31 $\underline{12}$-orbits:
12 of dimension 2
15 of dimension 1
4 of dimension 0

The 3 equivariant orbits can be described by quantifier-free formulas:

$$x > y \qquad x = y \qquad x < y.$$

These formulas are quantifier-free types, i.e. they specify all relations over the vocabulary (which, in this case, contains only the order relation). The $\underline{1}$-orbits can also be described by quantifier-free formulas which are allowed to use the constant $\underline{1}$:

$$x > y > \underline{1} \qquad x > \underline{1} = y \qquad x > \underline{1} > y \qquad \cdots$$

More generally, when the atoms are $(\mathbb{Q}, <)$, then every $\bar{a}$-orbit in $\mathbb{A}^n$ can be described by a quantifier-free formula with n free variables that uses constants from $\bar{a}$. Quantifier-free formulas are enough because the atoms $(\mathbb{Q}, <)$ are *homogeneous*, a property of atom structures that is discussed in Chapter 7. Some atom structures are oligomorphic but not homogeneous, and for those structures quantifier-free formulas will not be enough, but first-order formulas with quantifiers will.

The key point is that even though the orbits get smaller, the number of orbits never goes from finite to infinite. Therefore, it will make sense to talk about sets with finitely many orbits, with specifying the support of the orbits.

Theorem 3.16. *Assume that the atoms are oligomorphic. For every set with atoms X, the following conditions are equivalent.*

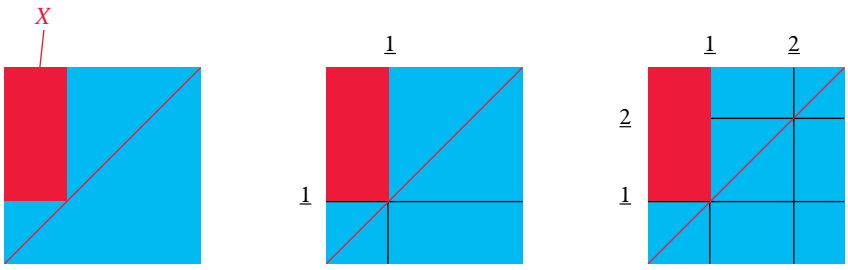
(1) *For some tuple of atoms $\bar{a}$, X is contained in a finite union of $\bar{a}$-orbits.*
(2) *For every tuple of atoms $\bar{a}$, X is contained in a finite union of $\bar{a}$-orbits.*

Before proving the theorem, we give some examples and discuss its consequences.

Example 3.17. Consider the set X which is the union

$$\underbrace{\{(a,b) : \text{ for } a,b \in \mathbb{A} \text{ such that } x \leq \underline{1} < y\}}_{X_1} \cup \underbrace{\{(a,b) : \text{ for } a,b \in \mathbb{A} \text{ such that } x = y\}}_{X_2}$$

which is depicted in the following picture:



The set is contained in two equivariant orbits, it is contained in (in fact, equal to) five $\underline{1}$-orbits, and it is contained (again, equal to) eleven $\underline{12}$-orbits. It is not exactly clear how to answer the question

how many orbits does X have?

without specifying the support. One idea is to use the least support, which exists for the atoms $(\mathbb{Q}, <)$, as we will see in Chapter 6. It is not immediately clear if this is a good idea, for example sizes defined this way do not sum up when taking unions of sets with different supports:

$$\overbrace{|X_1 \cup X_2|}^{\text{least support } \underline{1}} = 5 \qquad \overbrace{|X_1|}^{\text{least support } \underline{1}} = 2 \qquad \overbrace{|X_2|}^{\text{equivariant}} = 1$$

We will revisit counting orbits in Chapter 9.

Although it is not clear what is the exact number of orbits in a set with atoms, as discussed in the above example, the question

does X have finitely many orbits?

has an unambiguous answer by Theorem 3.16. This motivates the following definition, which is the central notion of this book.

Definition 3.18 (Orbit-finite sets)**.** Assume that the atoms are oligomorphic. A set with atoms which satisfies any of the two equivalent conditions from Theorem 3.16 is called *orbit-finite*.

We do not talk about orbit-finiteness when the atoms are not oligomorphic. Exercise 50 shows how the conditions in the above theorem are not equivalent in the non-oligomorphic structure $(\mathbb{Z}, <)$.

By definition, every set with atoms is finitely supported, which means that it is equal to a union of $\bar{a}$-orbits for some finite tuple of atoms $\bar{a}$. For orbit-finite sets, this union is finite, which means that a set is orbit-finite if and only if it is equal to a finite union of $\bar{a}$-orbits for some tuple of atoms $\bar{a}$. As the tuple $\bar{a}$ grows, the $\bar{a}$-orbits become smaller, and therefore the largest orbits are the equivariant orbits. This means that a set with atoms is orbit-finite if and only if it is contained in a finite union of equivariant orbits.

We now prove Theorem 3.16. The first observation is that, in the definition of an oligomorphic structure, we could have talked about $\bar{a}$-orbits instead of equivariant orbits, and nothing would change.

Lemma 3.19. *If the atoms are oligomorphic, then for every atom tuple $\bar{a}$ and every dimension $n \in \{0, 1, \ldots\}$ there are finitely many $\bar{a}$-orbits in $\mathbb{A}^n$.*

Proof Let k be the dimension of $\bar{a}$. Two n-tuples of atoms $\bar{b}$ and $\bar{c}$ are in the same $\bar{a}$-orbit if and only if the $(k+n)$-tuples $\bar{a}\bar{b}$ and $\bar{a}\bar{c}$ are in the same $\emptyset$-orbit. By oligomorphism there are finitely many possibilities for the latter. □

A corollary of the above lemma is that Theorem 3.16 is true for subsets of $\mathbb{A}^n$, because every finitely supported subset of $\mathbb{A}^n$ satisfies both conditions in the theorem. To go from subsets of $\mathbb{A}^n$ to arbitrary sets with atoms, we use the following lemma. The lemma does not need the assumption on oligomorphism.

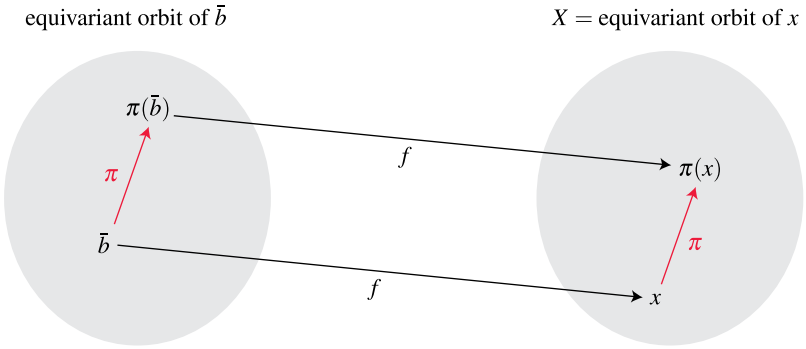
Lemma 3.20. *Let X be a set with atoms that is a single equivariant orbit. There is some $n \in \{1, 2, \ldots\}$ and a surjective equivariant function*

$$f : Y \to X \qquad \text{for some equivariant } Y \subseteq \mathbb{A}^n.$$

Proof Choose some $x \in X$. Because the finite support condition is hereditary for sets with atoms, x has a finite support $\bar{b}$.. Consider the equivariant orbit of the pair $(\bar{b}, x)$, i.e. the set

$$f = \{\pi(\bar{b}, x) : \pi \text{ is an atom automorphism}\}.$$

Here is a picture of f:



We claim that f is in fact a function, i.e. for every input there is exactly one output. Indeed, by Exercise 42, every atom automorphisms π, σ satisfy

$$\pi(\bar{b}) = \sigma(\bar{b}) \qquad \text{implies} \qquad \pi(x_0) = \sigma(x_0),$$

which means that f is a function. Its domain is the equivariant orbit of $\bar{b}$, which is an equivariant subset of $\mathbb{A}^n$, where n is the length of the support $\bar{b}$. Because f is equivariant, the image of the equivariant orbit of $\bar{b}$ is equal to the equivariant orbit of x, see Exercise 45, thus proving the lemma. □

Using the two above results, we finish the proof of Theorem 3.16.

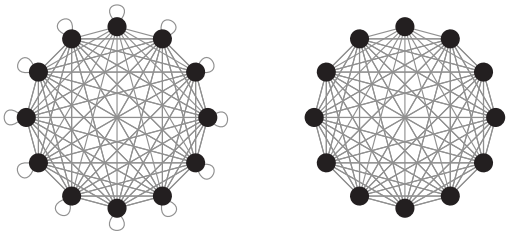
Proof of Theorem 3.16 The only non-trivial implication is (1) ⇒ (2), i.e. if X is contained in a finite union of $\bar{a}$-orbits for some $\bar{a}$, then the same is true for every $\bar{a}$. Since the biggest orbits are the equivariant orbits, the theorem boils down to showing that if a set is contained in a finite union of equivariant orbits, then for every atom tuple $\bar{a}$ it is contained in a finite union of $\bar{a}$-orbits. This, in turn, boils down to showing that every equivariant orbit splits into finitely many $\bar{a}$-orbits, for every atom tuple $\bar{a}$. Let then X be an equivariant orbit. Apply Lemma 3.20 to X yielding an equivariant surjective function

$$f : Y \to X.$$

By Lemma 3.19, the set Y splits into finitely many $\bar{a}$-orbits. Under an equivariant function, the image of an $\bar{a}$-orbit is also an $\bar{a}$-orbit, and therefore also X is a finite union of $\bar{a}$-orbits. □

Example 3.21. When the atoms are oligomorphic, then by definition of oligomorphism, the set $\mathbb{A}$ of all atoms is orbit-finite; the same is also true for $\mathbb{A}^n$. In the special case of the equality atoms or the rational numbers with order $(\mathbb{Q}, <)$, then the set $\mathbb{A}$ is even a single equivariant orbit. There are oligomorphic atom

structures with more than one equivariant orbit of atoms. An example is a variation of the two clique example from Example 3.13, where one of the two cliques has self-loops, but the other one does not:



The following example shows that the number of orbits in a product $X \times Y$ is not polynomially bounded (in fact, not bounded by any function) by the number of orbits in X, Y.

Example 3.22. Consider the equality atoms. For $n \in \mathbb{N}$, we write $\mathbb{A}^{(n)}$ for the set of non-repeating n-tuples of atoms, i.e. tuples where all coordinates are pairwise distinct. This set is one equivariant orbit, because every non-repeating tuple can be mapped to every other non-repeating tuple by an automorphism of atoms, since the only structure is equality. The square of this set,

$$\mathbb{A}^{(n)} \times \mathbb{A}^{(n)}$$

has a number of equivariant orbits that is exponential in n, corresponding to the ways in which the first n coordinates can be equal to the second n coordinates. In particular, the number of orbits of $X \times X$ cannot be bounded by a function of the number of orbits in X.

A representation theorem. Recall Lemma 3.20, which said that every equivariant one-orbit set is the image, under an equivariant function, of some orbit of tuples of atoms. Below, we build on that lemma to get a representation of every orbit-finite set up to finitely supported bijections.

Define a *partial equivalence relation* to be a relation that is transitive, symmetric, but not necessarily reflexive. Like (total) equivalence relations, partial equivalence relations also have disjoint equivalence classes, but these no longer need to cover the entire set. We write $X_{/\sim}$ for the family of equivalence classes in a set X modulo a partial equivalence relation $\sim$.

The following straightforward representation result says that – up to finitely supported bijections – every orbit-finite set can be obtained by quotienting tuples of atoms modulo some partial equivalence relation.

Theorem 3.23. *Assume the atoms are oligomorphic, and there are at least two atoms. Every $\bar{a}$-supported orbit-finite set admits an $\bar{a}$-supported bijection with a set of the form $\mathbb{A}^n_{/\sim}$ where $n \in \{0, 1, \ldots\}$ and and $\sim$ is an $\bar{a}$-supported partial equivalence relation on $\mathbb{A}^n$.*

Proof We first prove the lemma for sets which are a single $\bar{a}$-orbit, because the representation in the theorem is closed under finite unions. Let then X be a set with atoms which is a single $\bar{a}$-orbit. Apply Lemma 3.20 to the equivariant orbit that contains X, yielding an partial equivariant function

$$f : \mathbb{A}^n \to \text{sets with atoms}$$

whose image contains X. Define $\sim$ to be the partial equivalence relation on $\mathbb{A}^n$ which identifies two atom tuples if they have the same image under f, and that image belongs to X. This equivalence relation is supported by $\bar{a}$. It is the same as the kernel of the function f restricted to tuples with image in X. As usual with kernels, the set X is in bijective correspondence with the quotient $\mathbb{A}^n_{/\sim}$.

Consider now the general case, where X is not necessarily a single $\bar{a}$-orbit. Thanks to the assumption that there are at least two atoms, sets of the form $\mathbb{A}^n_{/\sim}$ are closed under finite union. To represent a union of k such sets, each one using dimension $\leq n$, we use tuples of dimension $n + \log k$, with the equality type of the last $\log k$ coordinates being used to encode which of the k sets is being used. □

If there is only one atom (which is not a very interesting case, because for finite atom structures orbit-finite sets are the same as finite sets), then the theorem above also holds, except one needs to use a finite disjoint union of sets of the form $\mathbb{A}^n_{/\sim}$.

Closure properties. Some closure properties of finite sets are shared by orbit-finite sets, some are not. Here is a summary:

operation on orbit-finite sets	is the result orbit-finite?
$X \cup Y$	yes (Lemma 3.24)
$X \cap Y$	yes (Lemma 3.24)
$X \times Y$	yes (Lemma 3.24)
take some finitely supported subset of X	yes (Lemma 3.24)
image $f(X)$ where f has an orbit-finite graph	yes (Lemma 3.24)
set of all finitely supported subsets of X	no (Example 3.25)
set of all finitely supported functions $X \to Y$	no (Exercises 52 and 53)

The positive results from the above table are given in the following lemma.

Lemma 3.24. *Assume that the atoms are oligomorphic. Orbit-finite sets are closed under binary union, binary product, finitely supported subsets, projections (from products), and images under finitely supported functions.*

Proof Binary union is immediate, and finitely supported subsets are immediate in view of Condition (2) of Theorem 3.16. For projections, we observe that the projection of one orbit in a product is one orbit: if (x, y) and (x', y') are in the same equivariant orbit, then the same is true for y and y'. For images under orbit-finite functions, we simply observe that the co-domain of an orbit-finite functions is orbit-finite, by projections.

We are left with binary products. By condition (1) of Theorem 3.16, it suffices to show that the product $X_1 \times X_2$ of two equivariant orbits is contained in finitely many equivariant orbits. Apply Lemma 3.20, yielding equivariant functions

$$f_i : \mathbb{A}^{n_i} \to \text{sets with atoms} \qquad \text{for } i = 1, 2$$

such that X_i is contained in the image of f_i. The product $X_1 \times X_2$ is contained in the image of $\mathbb{A}^{n_1} \times \mathbb{A}^{n_2}$ under the equivariant function obtained by pairing f_1 and f_2 in the natural way. By assumption on oligomorphism, $\mathbb{A}^{n_1} \times \mathbb{A}^{n_2}$ has finitely many equivariant orbits, and therefore the same is true for $X_1 \times X_2$ thanks to the previous item. □

Example 3.25. An important operation that does not preserve orbit-finiteness is (finitely supported) powerset. If the atoms are an infinite oligomorphic structure, then the finitely supported powerset of $\mathbb{A}$ is necessarily not orbit-finite, because subsets of different size are in different orbits.

Exercises

Exercise 50. Show that in the atoms $(\mathbb{Z}, <)$, which are not oligomorphic, the two conditions in Theorem 3.16, and also the condition "X is equal to a finite union of $\bar{a}$-orbits", are pairwise non-equivalent.

Exercise 51. Assume that the atoms are oligomorphic. Let $R \subseteq X \times X$ be a binary relation which is an orbit-finite set with atoms. Show that the transitive closure of R is also orbit-finite.

Exercise 52. 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 $X \to Y$.

Exercise 53. Assume oligomorphic atoms. Let X, Y be orbit-finite sets and let F be a finitely supported set of finitely supported functions of type $X \to Y$. 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 54. Show the following converse of Theorem 3.16: if the two conditions in the theorem are equivalent for every set with atoms X, then the atoms are oligomorphic.

Exercise 55. 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 56. Show a counterexample, in the equality atoms, to the converse implication from Exercise 55. In other words, show a set which is not orbit-finite, but where every tuple of atoms supports finitely many elements.

Exercise 57. 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 58. Show that Exercise 57 fails in $(\mathbb{Q}, <)$.

Exercise 59. Show that Exercise 57 fails in some atoms, even for a relation R such that for every first argument, there are finitely many second argument related by the relation.

Exercise 60. 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).

Exercise 61. Assume that the atoms are oligomorphic. Show that if X is an orbit-finite set and $\bar{a}$ is an atom tuple, then

$$\{\pi(x) : x \in X \text{ and } \pi \text{ is an } \bar{a}\text{-automorphism}\}$$

is also orbit-finite.

Exercise 62. Assume that the atoms are oligomorphic. Show that orbit-finite sets are closed under orbit-finite union in the following sense. If X is an orbit-finite set and f is a finitely supported function that maps each element of X to an orbit-finite set, then

$$\bigcup_{x \in X} f(x)$$

is an orbit-finite set.

Exercise 63. Show that in the equality atoms (actually, under any oligomorphic atoms), every orbit-finite is Dedekind finite[7], i.e. does not admit a finitely supported bijection with a proper subset of itself.

Exercise 64. Show that in the equality atoms, there is a set that is not orbit-finite, but Dedekind finite in the sense from Exercise 63.

Exercise 65. 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 66. Call a family $\mathcal{X}$ of sets *uniformly supported*[8] 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 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 67. Show that the following statement is true in the equality atoms but not in $(\mathbb{Q}, <)$. A set X is orbit-finite if and only if: (***) for every set with atoms $\mathcal{X} \subseteq \mathsf{P}X$ which is totally ordered by inclusion, there is a maximal element.

Exercise 68. Assume that the atoms are oligomorphic. 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.

[7] This exercise is inspired by Blass (2013).
[8] This exercise is inspired by (Pitts, 2013, Section 5.5.).

4

Representing orbit-finite sets

How does one represent an orbit-finite set X so that it can be processed by algorithms? One idea is to choose a support $\bar{a}$, and elements

$$x_1, \ldots, x_n \in X$$

which represent all $\bar{a}$-orbits, and then write X as

$$X = \bigcup_{i \in \{1,\ldots,n\}} \bar{a}\text{-orbit of } x_i,$$

with $x_1, \ldots, x_n$ being represented by induction assumption. This representation works – assuming that the atoms can be represented in a finite way – for *hereditarily orbit-finite sets*, which are sets that are orbit-finite, their elements are orbit-finite, and so on recursively until atoms are reached. Two drawbacks of this representation are: (a) it is not immediately clear how to work with it, e.g. how to test two representations for equality; and (b) a lot of space is required to represent simple sets, e.g. representing $\mathbb{A}^n$ requires enumerating all of the exponentially many orbits. In Section 9.2 we revisit this representation, but in this chapter we work with a different idea.

We propose a representation, using set builder expressions, which avoids the drawbacks (a) and (b), and has the further advantage that it works for models that are not necessarily oligomorphic, such as Presburger arithmetic $(\mathbb{N}, +)$. We also show how set builder expressions can be transformations by algorithms, with some transformations (like union) being polynomial time, and some transformations (like testing equality) being as hard as the first-order theory of the underlying atom structure. Then, we we show that if the atoms are oligomorphic, then the set builder expressions coincide with hereditarily orbit-finite sets.

4.1 Set builder expressions

Set builder expressions[1] are a way of defining sets, which contain sets, which contain sets, etc. Here is the family of all subsets of $\mathbb{A}$ that miss at most one atom:

$$\{\{x : \text{ for } x \in \mathbb{A}\}\} \cup \{\{y : \text{ for } x \in \mathbb{A} \text{ such that } y \neq x\} : \text{ for } x \in \mathbb{A}\}.$$

Another example, this time in the atoms $(\mathbb{Q}, <)$, is the set of all nonempty open intervals to the left of $\underline{1}$:

$$\{\{z : \text{ for } z \in \mathbb{A} \text{ such that } x < z < y\} : \text{ for } x, y \in \mathbb{A} \text{ such that } x < y \wedge y < \underline{1}\}.$$

Definition 4.1 (Set builder expressions)**.** Fix a logical structure $\mathbb{A}$ for the atoms, not necessarily oligomorphic. Fix an infinite set of variables, which range over atoms. We write x, y, z for these variables. There are two kinds of *set builder expression*, defined by structural induction:

(1) **Set expression.** Let α be a variable or an already defined set builder expression, and let φ be a first-order formula over the vocabulary of $\mathbb{A}$ with parameters from the atoms[2]. Then

$$\{\alpha : \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \varphi\}$$

is a set builder expression, called a *set expression*. The formula φ is called the *guard* of the expression. The free variables of the expression are the variables that are free in α or φ, minus $x_1, \ldots, x_n$.

(2) **Union expression.** If $\alpha_1, \ldots, \alpha_n$ are already defined set expressions, then $\alpha_1 \cup \cdots \cup \alpha_n$ is a set builder expression, called a *union expression*. A variable is free in the union if it is free in some α_i.

By design (and by name), set builder expressions represent sets, and not individual atoms. For a valuation

$$\bar{a} : \text{free variables of } \alpha \to \mathbb{A},$$

we define the *set represented by α under valuation $\bar{a}$*, denoted by $\alpha(\bar{a})$, in the natural way. If α has no free variables, then we simply speak of the set represented by α. We represent the valuations as tuples of atoms, assuming some implicit order on the free variables.

[1] The idea to use set builder notation as a way of representing hereditarily orbit finite sets (Theorem 4.10) originates from the programming language LOIS (*Looping over Infinite Sets*) Kopczyński and Toruńczyk (2016); Kopczyński and Toruńczyk (2017). Earlier papers on sets with atoms, such as Bojańczyk et al. (2014, 2012), used different representations, based on least supports.

[2] A formula with parameters is one that can use atoms as constants. For example $x = \underline{1} \vee x = \underline{2}$ is a formula which uses parameters $\underline{1}, \underline{2}$ and has free variable x.

Define the *parameters* of a set builder expression to be the parameters that appear in its guards. For example, the set builder expression

$$\{\{y : \text{ for } y \in \mathbb{A} \text{ such that } y \neq x \vee y = z\} : \text{ for } x \in \mathbb{A} \text{ such that } x \neq \underline{2}\}$$

has parameter $\underline{2}$ and free variable z. By induction on the size of α one shows that

$$\bar{a} \qquad \mapsto \qquad \text{set represented by } \alpha(\bar{a})$$

seen as a function from atom tuples (indexed by free variables of α) to sets with atoms, which is supported by the parameters of α. In particular, if α has no parameters, then the the function is equivariant. The distinction between parameters and variables is rather thin, but it will play a role in the definition of fixed dimension polynomial time that will be given in Chapter 9.

Before continuing, we discuss some syntactic sugar for set builder expressions. If $\bar{x}$ is a tuple of variables $x_1, \ldots, x_n$, then we sometimes write

$$\{\alpha : \text{ for } \bar{x} \in \mathbb{A}^n \text{ such that } \varphi\} \quad \text{instead of} \quad \{\alpha : \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \varphi\}.$$

If the number n of bound variables might be zero and the guard φ is "true", then we write $\{\alpha\}$. We write $\{\alpha, \beta\}$ as syntactic sugar for the union of two singletons. Likewise, we write (α, β) as syntactic sugar for Kuratowski pairs. Using these conventions, examples of set builder expressions include finite sets like

$$\{\underline{1}, \underline{3}, \underline{4}\},$$

and also hereditarily finite sets like

$$\{\{\{\underline{1}, \underline{5}\}, \underline{1}, \underline{3}\}, \{\underline{4}\}\}.$$

Symbol pushing lemmas

Many operations on set builder expressions, such as union or set difference, can be implemented by straightforward syntactic transformations on formulas of first-order logic. In many cases, the 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[3]). In a similar way, we define the size of a set builder expression to be the number of distinct subexpressions and subformulas of formulas used in the guards.

[3] 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 70.

The following straightforward lemma, which makes no assumptions on the atom structure, even decidability of the first-order theory, says that membership and inclusion questions for (sets represented by) set builder expressions reduce to the first-order theory of the atom structure.

Lemma 4.2 (First Symbol Pushing Lemma)**.** *Assume that the atoms can be represented in a finite way. 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:*

$$\mathbb{A} \models \varphi(\bar{a}\bar{b}) \quad \textit{iff} \quad \alpha(\bar{a}) \subseteq \beta(\bar{b}) \qquad \textit{for every atom tuples } \bar{a}, \bar{b}.$$

Likewise for $\in$ or $=$ instead of $\subseteq$.

Proof See Figure 4.1. The formulas produced in Figure 4.1 produce a fixed formula and then refer to inductively defined subformulas which correspond to subexpressions of α and β. It follows that the size of the first-order formula produced in Figure 4.1 is approximately the number of subexpressions in α times the number of subexpressions in β. □

In the above lemma, the construction of the formula φ based on α and β is a simple syntactic transformation, which requires no computation on the atom parameters that appear in the input set builder expression. Therefore, the assumption that the atoms can be represented in a finite way is stronger than necessary, and the lemma would be true without any assumptions on the atoms, but with a computation model where atoms can be moved around (without actually checking any of their properties) at unit cost. The same will be true for the remaining Symbol Pushing Lemmas below.

A corollary of the First Symbol Pushing Lemma is that if the atoms have decidable first-order theory, then membership, inclusion and equality are decidable for sets represented by set builder expressions.

Corollary 4.3. *Assume that the atoms can be represented in a finite way so that the following problem is decidable:*

- **Input.** *A sentence of first-order logic φ over the vocabulary*[4] *of the atoms, which is allowed to contain parameters from the atoms.*
- **Output.** *Is φ true in the atom structure?*

[4] This assumes that the vocabulary is either finite, or infinite but can be represented in some way.

$$
\begin{aligned}
\underline{\alpha \in \beta} &\stackrel{\text{def}}{=} \mathit{false} \quad \text{if } \beta \text{ represents an atom} \\
\underline{\alpha \in (\beta_1 \cup \cdots \cup \beta_n)} &\stackrel{\text{def}}{=} \bigvee_i \underline{\alpha \in \beta_i} \\
\underline{\alpha \in \{\beta : \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \varphi\}} &\stackrel{\text{def}}{=} \exists x_1 \ldots \exists x_n \underline{\alpha = \beta} \\
\\
\underline{\alpha \subseteq \beta} &\stackrel{\text{def}}{=} \mathit{false} \quad \text{if } \alpha \text{ represents an atom} \\
\underline{(\alpha_1 \cup \cdots \cup \alpha_n) \subseteq \beta} &\stackrel{\text{def}}{=} \bigwedge_i \underline{\alpha_i \subseteq \beta} \\
\underline{\{\alpha : \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \varphi\} \subseteq \beta} &\stackrel{\text{def}}{=} \forall x_1 \ldots \exists x_n \underline{\alpha \in \beta} \\
\\
\underline{\alpha = \beta} &\stackrel{\text{def}}{=} \alpha = \beta \quad \text{if } \alpha, \beta \text{ represent atoms} \\
\underline{\alpha = \beta} &\stackrel{\text{def}}{=} \underline{\alpha \subseteq \beta} \wedge \underline{\beta \subseteq \alpha} \quad \text{if } \alpha, \beta \text{ represent sets} \\
\underline{\alpha = \beta} &\stackrel{\text{def}}{=} \mathit{false} \quad \text{otherwise}
\end{aligned}
$$

Figure 4.1 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 between, e.g. membership $\alpha \in \beta$, the figure shows a corresponding first-order formula, which is denoted using underlines, e.g. $\underline{\alpha \in \beta}$.

Then membership, equality and inclusion are decidable for sets represented by set builder expressions.

While the formulas in the conclusion of the First Symbol Pushing Lemma are produced in polynomial time, checking if these formulas are true in the atom structure is a different story. In fact, first-order model checking is PSpace-hard for any structure with at least two elements, because it first-order logic generalises quantified Boolean formulas. This hardness is intrinsic to the original problem (checking if set builder expressions represent the same set, likewise for membership and inclusion), and not just the solution presented in the First Symbol Pushing Lemma, because first-order formulas are built into the syntax of set builder expressions. For example, checking

$$\emptyset = \{\bar{x} : \text{ for } \bar{x} \in \mathbb{A}^n \text{ such that } \varphi(\bar{x})\}$$

is the same as checking if the first-order guard $\varphi(\bar{x})$ is true for at least one atom tuple. Furthermore, the quantifiers guards can be simulated using the nesting of set brackets, see Exercise 69.

Example 4.4. Examples of structures which satisfy the assumptions of Corol-

lary 4.3 include:

$$(\mathbb{N}, =), \qquad (\mathbb{Q}, <), \qquad \underbrace{(\mathbb{N}, +)}_{\text{Presburger arithmetic}}, \qquad \underbrace{(\mathbb{N}, \times)}_{\text{Skolem arithmetic}}.$$

For the first two structures, decidability follows from quantifier elimination, see Chapter 7. For Presburger and Skolem arithmetic, decidability is easily proved using monadic second-order logic on words and trees[5]. Presburger and Skolem arithmetic are not oligomorphic. A non-example is the real field $(\mathbb{R}, +, \times)$, which has decidable first-order theory, but fails the assumption on representing elements[6].

All oligomorphic structures in this book satisfy the assumptions of Corollary 4.3.

Lemma 4.5 (Second Symbol Pushing Lemma)**.** *Assume that the atoms can be represented in a finite way. There is polynomial time algorithm which does the following.*

- **Input.** *Sets X and Y, given by set builder expressions.*
- **Output.** *Set builder expressions representing the sets*

$$X \cup Y, \quad X \cap Y, \quad X - Y, \quad \{X\}, \quad \bigcup X, \quad X \times Y.$$

Proof All cases are straightforward, with intersection $X \cap Y$ and difference $X - Y$ using the Symbol Pushing Lemma. We only do one of the cases, namely intersection $X \cap Y$. By distributing intersection across union, we assume that X and Y are represented by set expressions

$$\underbrace{\{\alpha : \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \varphi\}}_{X}, \quad \underbrace{\{\beta : \text{ for } y_1, \ldots, y_m \in \mathbb{A} \text{ such that } \psi\}}_{Y}.$$

Apply the Symbol Pushing Lemma, yielding a polynomial size formula $\underline{\alpha = \beta}$ which characterises those valuations of the free variables which make the expressions α and β equal. The expression for intersection is then

$$\{\alpha : \text{ for } x_1, \ldots, x_n \in \mathbb{A} \text{ such that } \exists y_1, \ldots, y_m \, \psi \wedge \underline{\alpha = \beta}\}.$$

□

[5] Presburger's original proof Presburger (1929), used quantifier elimination. For an approach that uses automata, see (Thomas, 1997, page 399) for Presburger arithmetic and Blumensath and Gradel (2000) for Skolem arithmetic.

[6] This problem would be solved by considering the field of real algebraic numbers, which has the same first-order theory as the field of reals, and allows finite representation of its elements.

The Second Symbol Pushing Lemma does not cover some operations, such as composition of binary relations, or the projection operations on products, etc. Instead of treating each such operation separately, we give a generic result, which deals with every operation that can be defined in the language of set theory.

Definition 4.6 (Set Structure)**.** Let X be a set. Define X_* to be the set which contains X, the elements of X, their elements, and so on. In other words,

$$X_* \quad \stackrel{\text{def}}{=} \quad \{X\} \cup \bigcup_{x \in X} x_*.$$

Define the *set structure of* X to be the logical structure which has universe X_*, one binary relation $\in$ and one constant $\emptyset$.

Many natural constructions can be described using first-order logic on the set structure. The following example discusses projection from a Cartesian product. Other examples, such as composition of binary relations, can be found in the exercises.

Example 4.7. We claim that if that the projection function

$$X \times Y \to X$$

can be defined by a first-order formula interpreted in the set structure 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 universe of the set structure 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):

$$\forall z \; z \in p \;\Rightarrow\; (\overbrace{z = \{x\}}^{\substack{x \text{ is the unique} \\ \text{element of } z}} \;\vee\; \overbrace{z = \{x\} \cup \{y\}}^{\substack{z \text{ is the smallest set that} \\ \text{contains } \{x\} \text{ and } \{y\}}}). \tag{4.1}$$

The projection of $X \times Y$ to the first coordinate is the set of elements x in the set

structure of $X \times Y$ that satisfy the following formula $\varphi(x)$:

$$\exists y \quad \overbrace{y \in X \times Y}^{\substack{X \times Y \text{ is the only} \\ \text{only element of the} \\ \text{set structure that} \\ \text{does not belong to} \\ \text{any other element}}} \quad \wedge \quad \overbrace{p = (x, y)}^{\text{expressed in (4.1)}} \ . \tag{4.2}$$

The Third Symbol Pushing Lemma says that one can compute in polynomial time all operations which can be formalised using first-order logic over the set structure, such as the projection operation given in the above example.

Lemma 4.8 (Third Symbol Pushing Lemma)**.** *Assume that the atoms can be represented in a finite way, and let $\varphi(x_1, \ldots, x_n)$ be a formula of first-order logic using a binary relation $\in$ and a constant $\emptyset$. There is polynomial time algorithm*[7] *which does the following.*

- **Input.** *A set X, represented by a set builder expression.*
- **Output.** *A set builder expression which representing the set*

$$\{(x_1, \ldots, x_n) \in (X_*)^n : X_* \models \varphi(x_1, \ldots, x_n)\}.$$

Proof The set X_* is obtained from X using

$$(X, Y) \mapsto X \cup Y \qquad X \mapsto \bigcup X,$$

which were treated in the Second Symbol Pushing Lemma, and therefore one can compute in polynomial time a set builder expression α_* which describes X_*. Like any set builder expression, α_* is a finite union of set expression:

$$\bigcup_{i \in I} \{\alpha_i(\bar{y}_i) : \text{ for } \bar{y}_i \in \mathbb{A}^{k_i} \text{ such that } \varphi_i(\bar{y}_i)\}.$$

In the above, $\bar{y}_i$ is a tuple of k_i variables. (These variables range over atoms, while the variables $x_1, \ldots, x_n$ of the formula in the statement the lemma range over elements of X_*.) Note that some of the expressions α_i might be atom constants or variables representing individual atoms, in which case they are not formally speaking set builder expressions (because the latter necessarily contain set brackets). The statement of the lemma follows immediately from the following claim.

Claim 4.9. *Let $\varphi(x_1, \ldots, x_n)$ be a first-order formula (whose free variables are contained in, but not necessarily equal to, $\{x_1, \ldots, x_n\}$) that uses $\in$ and $\emptyset$ only, and let $i_1, \ldots, i_n \in I$. There is a first-order formula, denoted by*

$$\underline{\varphi(x_1, \ldots, x_n) \ : \ i_1, \ldots, i_n}, \tag{4.3}$$

[7] The degree of the polynomial depends on φ.

$$
\begin{aligned}
\underline{y_j \in y_k \ : \ \alpha_{i_1}, \ldots, \alpha_{i_n}} &\stackrel{\text{def}}{=} \overbrace{\underline{\alpha_{i_j} \in \alpha_{i_k}}}^{\substack{\text{First Symbol}\\ \text{Pushing Lemma}}} \\
\underline{y_j = y_k \ : \ \alpha_{i_1}, \ldots, \alpha_{i_n}} &\stackrel{\text{def}}{=} \overbrace{\underline{\alpha_{i_j} = \alpha_{i_k}}}^{\substack{\text{First Symbol}\\ \text{Pushing Lemma}}} \\
\underline{(\varphi_1 \wedge \varphi_2) \ : \ \alpha_{i_1}, \ldots, \alpha_{i_n}} &\stackrel{\text{def}}{=} \underline{\varphi_1 \ : \ \alpha_{i_1}, \ldots, \alpha_{i_n}} \wedge \underline{\varphi_2 \ : \ \alpha_{i_1}, \ldots, \alpha_{i_n}} \\
\underline{\neg\varphi \ : \ \alpha_{i_1}, \ldots, \alpha_{i_n}} &\stackrel{\text{def}}{=} \neg\underline{\varphi_1 \ : \ \alpha_{i_1}, \ldots, \alpha_{i_n}} \\
\underline{\exists x_n \, \varphi \ : \ \alpha_{i_1}, \ldots, \alpha_{i_{n-1}}} &\stackrel{\text{def}}{=} \bigvee_{i_n \in I} \exists \bar{y}_{i_n} \ \underbrace{\alpha_{i_n}(\bar{y}_{i_n}) \in \alpha_*}_{\substack{\text{First Symbol}\\ \text{Pushing Lemma}}} \wedge \underline{\varphi(x_1, \ldots, x_n) \ : \ \alpha_{i_1}, \ldots, \alpha_{i_n}}.
\end{aligned}
$$

Figure 4.2 Proof of Claim 4.9. The connectives $\vee$ and $\exists$ are treated the same way as $\wedge$ and $\forall$, respectively. For the quantifier $\forall$, we assume for notational simplicity that the quantified variable is the last one.

which uses the vocabulary of the atoms and atom parameters, and has

$$k_{i_1} + \cdots + k_{i_n}$$

free variables, such that a tuple of atoms

$$\bar{a}_1 \cdots \bar{a}_n \in \mathbb{A}^{k_{i_1}} \times \cdots \times \mathbb{A}^{k_{i_n}}$$

satisfies (4.3) *if and only if*

$$(X_*, \in, \emptyset) \models \varphi(\alpha_{i_1}(\bar{a}_1), \ldots, \alpha_{i_n}(\bar{a}_n)).$$

Proof Induction on formula size, see Figure 4.2. □

Using the claim, we complete the proof of the lemma. The set builder expression defining the interpretation of $\varphi(x_1, \ldots, x_n)$ is the union, ranging over all $i_1, \ldots, i_n \in I$, of set builder expressions of the form

$$\{(\alpha_{i_1}, \ldots, \alpha_{i_n}) : \text{ for } \bar{y}_{i_1}, \ldots, \bar{y}_{i_n} \in \mathbb{A} \text{ such that } \underline{\varphi(x_1, \ldots, x_n) \ : \ \alpha_{i_1}, \ldots, \alpha_{i_n}}\}.$$

All the constructions in the above proof are clearly computable. Furthermore, if we fix the formula φ, then the final set builder expression has size polynomial in α. The exponent of the polynomial depends on the maximal number of free variables used by subformulas of φ, because a formula with n free variables will require ranging over I^n. □

Exercises

Exercise 69. 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 70. 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 71. 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 72. 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?

4.2 Hereditarily orbit-finite sets

In this section we show that if the atoms are oligomorphic, then the hereditarily orbit-finite sets (orbit-finite, the elements are orbit-finite, their elements are orbit-finite, and so on) described at the beginning of this chapter are the same as the those defined by set builder expressions.

Theorem 4.10. *Assume that the atoms are countable and oligomorphic. A set is defined by a set builder expression if and only if it is hereditarily orbit-finite.*

The key part of the above theorem is the following lemma, which is based on the proof of Ryll-Nardzewski, Engeler and Svenonius about countable oligomorphic being the same thing as ω-categorical.

Lemma 4.11. *In a countable oligomorphic model, a subset $X \subseteq \mathbb{A}^n$ is equivariant if and only if it is first-order definable.*

Proof Consider the following game (known as the Ehrenfeucht-Fraïssé game), which is parametrised by two tuples $\bar{a}, \bar{b} \in \mathbb{A}^n$ 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 a quantifier-free formulas, otherwise Duplicator wins.

The lemma follows immediately from the equivalence of items 1 and 4 in the following claim.

Claim 4.12. *In a countable oligomorphic structure $\mathbb{A}$, the following conditions are equivalent for every tuples $\bar{a}, \bar{b} \in \mathbb{A}^n$:*

(1) *satisfy the same formulas of first-order logic*[8]*;*
(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) *are in the same equivariant orbit.*

Proof

- *1 implies 2.* This is (half of) the classical Ehrenfeucht-Fraïssé theorem[9], which says that if two tuples satisfy the same formulas of quantifier rank at most k, then player Duplicator has a winning strategy in the k-round game.
- *2 implies 3.* In this step, we use oligomorphism. We need to show that if Duplicator has a winning strategy for every $k < \omega$, then Duplicator also has a winning strategy in the ω-round game. Consider the situation in the ω-round when Spoiler is about to extend a tuple $\bar{a}$ by a new atom, and the other tuple is $\bar{b}$. If atoms a and a' are in the same $\bar{a}\bar{b}$-orbit, then extending the tuple $\bar{a}$ by a or extending it by a' will give the same results for Spoiler as far as winning the game is concerned. Since there are finitely many $\bar{a}\bar{b}$-orbits, Spoiler has essentially finitely many different choices. The same holds for Duplicator. Therefore, one can use the same argument as in the König lemma to show that if Spoiler wins the ω-round game, then already he wins the k-round game for some $k < \omega$.

[8] These are formulas that use the vocabulary of $\mathbb{A}$. They cannot use constants (parameters) other than those which are explicitly present in the vocabulary of $\mathbb{A}$.
[9] See (Hodges, 1993, Section 3.2)

- *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

$$(a_1, \ldots, a_n) \quad (b_1, \ldots, b_n),$$

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

$$a_1, a_2, \ldots \qquad b_1, b_2, \ldots$$

of atoms such that for every $i \in \mathbb{N}$, the tuples $(a_1, \ldots, a_i)$ and $(b_1, \ldots, b_i)$ satisfy the same quantifier-free formulas. By the choice of Spoiler's strategy, every atom appears in the sequence $a_1, a_2, \ldots$ and every atom appears in the sequence $b_1, b_2, \ldots$. Therefore the function $a_i \mapsto b_i$ is an automorphism of the atoms.
- *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 claim, and therefore also of the lemma. □

□

Corollary 4.13. *Suppose that the atoms are a countable oligomorphic structure. If $X \subseteq \mathbb{A}^k$ is supported by a tuple of atoms $\bar{a} \in \mathbb{A}^n$, then it is definable by a first-order formula with k free variables and parameters from $\bar{a}$.*

Proof Define

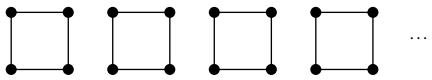
$$Y = \{\pi(\bar{a}\bar{b}) : \pi \text{ is an atom automorphism and } \bar{b} \in X\}.$$

This is an equivariant set, and therefore by Lemma 4.11 it is defined by a formula of first-order logic φ with $n + k$ free variables. A tuple $\bar{b}$ belongs to X if and only if it satisfies $\varphi(\bar{a}\bar{b})$. □

As discussed in Examples 3.7, in the equality atoms the finitely supported

subsets of $\mathbb{A}^k$ coincide with those that can be defined using quantifier-free formulas, which is a stronger statement than first-order definability as per Corollary 4.13). The same is true for $(\mathbb{Q}, <)$, see Example 3.8. The reason is that these atom structures are *homogeneous*, see Chapter 7. Not all oligomorphic structures are homogeneous, and therefore sometimes quantifiers are needed to define finitely supported relations on the atoms, as shown in the following example.

Example 6. Let $\mathbb{A}$ be the undirected graph which consists of a countably infinite disjoint union of cycles of length 4:



This is an oligomorphic structure. Consider the equivariant set

$$\{(a, b) : a, b \in \mathbb{A} \text{ are antipodal, i.e. } a \neq b \wedge \exists c\ E(a, c) \wedge E(c, b)\}.$$

The set is definable in first-order logic, but not without quantifiers. □

Before proving Theorem 4.10, let us note a further corollary of Corollary 4.13. Recall Theorem 3.23 which represented orbit-finite sets using tuples of atoms modulo partial equivalence relations. If we view a partial equivalence relation on n-tuples of atoms as a set of $2n$-tuples of atoms, and apply Corollary 4.13, we see that the partial equivalence can be defined by a formula of first-order logic with $2n$ free variables, possibly using parameters from the atoms. Putting this together with Theorem 3.23, we see that every $\bar{a}$-supported orbit-finite set admits an $\bar{a}$-supported bijection with a finite union of sets, each one of which is quotients of atom tuples modulo a partial equivalence relation that can be defined in first-order logic with parameters from $\bar{a}$.

Proof of Theorem 4.10 We begin with the left-to-right implication. By induction on the size of a set builder expression α, we show that

$$\bar{a} \quad \mapsto \quad \text{set represented by } \alpha(\bar{a})$$

is a function that inputs tuples of atoms and outputs hereditarily orbit-finite sets. Also, this function is supported by the parameters that appear in α. Consider the interesting case in the induction step, which is when α is a set expression of the form

$$\alpha(\bar{x}) = \{\beta(\bar{x}\bar{y}) : \text{ for } \bar{y} \in \mathbb{A}^n \text{ such that } \varphi(\bar{x}\bar{y})\}.$$

By definition, the set represented by $\alpha(\bar{a})$ is the image of the set

$$\{\bar{b} \in \mathbb{A}^n : \varphi(\bar{a}\bar{b})\}, \tag{4.4}$$

under the function

$$\bar{b} \quad \mapsto \quad \text{set represented by } \beta(\bar{a}\bar{b}). \tag{4.5}$$

The set (4.4) is orbit-finite as a finitely supported subset of the orbit-finite set $\mathbb{A}^n$. Finitely supported functions map orbit-finite sets to orbit-finite sets, and therefore $\alpha(\bar{a})$ is orbit-finite. Its elements are hereditarily orbit-finite by induction assumption.

We now turn to the right-to-left implication in Theorem 4.10. The proof is by induction on the rank in the cumulative hierarchy, i.e. the nesting depth of set brackets. Let then X be a hereditarily orbit-finite set supported by $\bar{a}$. Since set builder expressions have union in the syntax, it suffices to consider the case when X consists of a single $\bar{a}$-orbit. Choose some $x \in X$, with support $\bar{b}$. In particular, x is also supported by $\bar{a}\bar{b}$. By induction assumption, x is represented by some set builder expression $\alpha(\bar{a}\bar{b})$. The set X is therefore equal to

$$\{\pi(\alpha(\bar{a}\bar{b})) : \pi \text{ is } \bar{a}\text{-automorphism}\}.$$

Since $\bar{a}\bar{b} \mapsto \alpha(\bar{a}\bar{b})$ is an equivariant function, it commutes with atom automorphisms, and thus

$$X = \{\alpha(\bar{a}\bar{c}) : \bar{c} \in Y\} \tag{4.6}$$

where Y is the $\bar{a}$-orbit of the tuples $\bar{b}$. By Corollary 4.13, the set Y is is defined by a formula of first-order logic which uses parameters from $\bar{a}$. Substituting this formula for Y in 4.6 gives a set builder expression defining X. □

Exercises

Exercise 73. Assume that there are at least two atoms, but the atoms are not necessarily oligomorphic. Show the following variant of Theorem 3.23 for sets represented by set builder expressions: for every set X represented by a set builder expression there is some $n \in \{0, 1, \ldots\}$, a first-order definable partial equivalence $\sim$ on $\mathbb{A}^n$ (which can use atom parameters) and a bijection

$$f : \mathbb{A}^n_{/\sim} \to X$$

that is represented by a set builder expression.

Exercise 74. Assume that the atoms are oligomorphic. Show that for every orbit-finite set X there is a finitely supported function

$$f : \mathbb{A}^* \to (\text{finitely supported subsets of } X)^*$$

such that for every $\bar{a} \in \mathbb{A}^*$, $f(\bar{a})$ is a list of all $\bar{a}$-orbits that contained in X.

5
Case studies

In this section, we show how natural algorithms on finite objects can be generalised to orbit-finite sets. This is illustrated with case studies for: graph reachability, automaton emptiness and minimisation, equivalence of pushdown automata with context-free grammars, and graph homomorphisms. The point of these case studies is to explain why:

- orbit-finiteness has some of the advantages of finiteness, in particular how such sets can be transformed and searched by algorithms;
- combining the semantic approach (orbit-finite sets) and the syntactic approach (set builder expressions) can be useful;
- orbit-finite automata generalise the automata models from Part I, and why this generalisation is useful;
- it is useful to make an additional decidability assumption about the atoms: computability of the Ryll-Nardzewski function.

5.1 Graph reachability

We begin our case studies with directed graph reachability.

Theorem 5.1. *Assume that the atoms are oligomorphic and have decidable first-order theory as in the assumptions of Corollary 4.3. The following problem is decidable:*

- **Input.** *A directed graph (V, E), and source and target subsets $S, T \subseteq V$, all given by set builder expressions.*
- **Output.** *Is there a directed path from some source to some target?*

Proof For $n \in \{1, 2, \ldots\}$, let V_n be the vertices that are reachable in at most n

steps from some source. These sets are defined by

$$V_n = \begin{cases} S & n = 0 \\ V_{n-1} \cup V_{n-1}E & n > 0. \end{cases}$$

By the Third Symbol Pushing Lemma, a set builder expression describing the set V_n can be computed, given a set builder expression for V_{n-1}. For each $n = 0, 1, 2, \ldots$, compute the expression for V_n, until a fixpoint is reached, i.e. some n such that $V_n = V_{n+1}$. Whether or not a fixpoint is reached can be checked using Corollary 4.3. The algorithm returns true or false, depending on whether the fixpoint intersects the target vertices. The following claim shows that the fixpoint is always reached in a finite number of steps, and therefore the algorithm terminates.

Claim 5.2. *There is some n such that $V_n = V_{n+1}$.*

Proof Let $\bar{a}$ be a tuple of atoms that supports S, V and E. (For example, one can take $\bar{a}$ to be all of the atoms that appear in the set builder expressions that define these sets.) One can show by induction that this tuple also supports V_n for every n. Later in this section, we give a more general explanation for such results (a tuple that supports one thing must also support other related things), using a principle called the equivariance principle. Therefore all the sets

$$V_0 \subseteq V_1 \subseteq \cdots \subseteq V$$

are unions of $\bar{a}$-orbits. Because V is defined by a set builder expression, it is orbit-finite by Theorem 4.10. By Theorem 3.16, V is a finite union of $\bar{a}$-orbits It follows that for some n, there are no more $\bar{a}$-orbits to add when going from V_n to V_{n+1}. □

□

The above proof illustrates how it is useful to have both syntactic descriptions (set builder expressions) and semantic ones (hereditarily orbit-finite sets). The semantic description is used to show that the fixpoint is reached in a finite number of steps, while the syntactic description is used to compute this fixpoint.

Equivariance principle. In the proof above, we said that any tuple of atoms which supports the graph and its source vertices will also support the vertices that can be reached in at most n steps. Other examples of such statements are: "a tuple that supports an automaton will also support the language recognised by the automaton" or "a tuple that supports system of equations with a unique solution will also support that solution". We describe below a result, called the

equivariance principle, which implies all such statements. The principle says that, if a function (e.g. the function that maps an automaton to its recognised language, or the partial function that maps a system of equations to its unique solution if it exists) can be defined in the language of set theory, then that function is equivariant. In particular, any tuple supporting the input to that function will also support the output of that function, because for equivariant functions, any support of the input is also a support of the output.

Lemma 5.3 (Equivariance principle)**.** *Let $\mathbb{A}$ be a structure, not necessarily oligomorphic, and consider the structure*

$$\mathsf{set}\mathbb{A} \quad \stackrel{\text{def}}{=} \quad (\textit{atoms and sets with atoms over } \mathbb{A}, \in, \emptyset).$$

Suppose that $\varphi(x, y)$ is a formula of first-order logic which uses only $\in$ and $\emptyset$, and whose interpretation in the above structure is a function

$$f : \textit{atoms and sets with atoms over } \mathbb{A} \to \textit{atoms and sets with atoms over } \mathbb{A}.$$

Then f is an equivariant function. In particular, if an atom tuple supports a set with atoms X, then the same atom tuple also supports $f(X)$.

Proof Take an automorphism π of $\mathbb{A}$. It is not hard to see that π, when lifted to sets with atoms over $\mathbb{A}$, is an automorphism of the structure $\mathsf{set}\mathbb{A}$. First-order logic is invariant under automorphism, i.e. if a pair (X, Y) of elements in the universe of the structure $\mathsf{set}\mathbb{A}$ satisfies the formula defining f, then the same will be true for $(\pi(X), \pi(Y))$. □

The language of first-order logic is rich enough to cover most constructions used in this book. An example of such a construction is pairing and unpairing as described in Example 4.7. For such constructions we can use the equivariance principle to prove equivariance. Later in the book, we will no longer do detailed analysis as in Example 4.7, and we will simply refer to the equivariance principle when making statements such as "if an automaton is supported by an atom tuple $\bar{a}$, then its recognised language is also supported by $\bar{a}$".

Example 5.4. When applying the Equivariance Principle, one needs to remember that the structure $\mathsf{set}\mathbb{A}$ only talks about finitely supported sets (because sets with atoms are, by definition, finitely supported). To illustrate the potential for mistakes, consider the statement:

(*) If a graph is nonempty and has at least one outgoing edge for each vertex, then it has an infinite path.

The statement can easily be formalised using set theory, but such a formalisation turns out to be false in $\mathsf{set}\mathbb{A}$ for some choices of atoms. To see this,

consider the atoms $(\mathbb{Q}, <)$, and the graph where the vertices are the atoms and the edge relation is $\{(a, b) : a < b\}$. Every vertex has at least one outgoing edge, and indeed the graph contains an infinite path, but it does not contain any infinite finitely supported path, because such a path would need to use infinitely many atoms. Hence, (*) is not true in $\mathsf{set}\mathbb{A}$. In the equality atoms, (*) is true for orbit-finite graphs (see Exercise 76) but it is false in general (see Exercise 75).

Exercises

Exercise 75. Assume the equality atoms. Show a graph which has an infinite path, but does not have any infinite finitely supported path.

Exercise 76. Consider the following two conditions for a directed graph with a distinguished source $s \in V$ and set of target vertices $T \subseteq V$.

(1) there is an infinite directed path which starts in s and visits T infinitely often;
(2) for some $t \in T$, there is a path from s to t and a path from t to t..

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 77. Show that under the assumptions of Theorem 5.1, there is an algorithm that checks if condition (1) of Exercise 76 is satisfied, assuming that the graph, source and targets are all hereditarily orbit-finite. Likewise for condition (2).

Exercise 78. An instance of *alternating reachability* is defined the same way as an instance of graph reachability, except that there is an additional function $V \to \{0, 1\}$ which assigns an *owner* to each vertex. A yes-instance of alternating reachability is one where player 0 wins the following game, played by players 0 and 1. The game begins in the initial vertex s. In each round, the player who owns the current vertex picks an outgoing edge; if there is no outgoing edge, then the picking player loses immediately. If the play reaches a vertex in T, player 0 wins[1]; otherwise the play goes on forever and player 1 wins. In the game, we do not assume that the strategies of the players are finitely supported. Show that under the assumptions of Theorem 5.1, alternating reachability is decidable for instances that are hereditarily orbit-finite.

[1] This type of game is called a *reachability game*. More general games, namely parity games, are studied in (Klin and Łełyk, 2017, Section 5.2)

Exercise 79. Consider the atoms $(\mathbb{N}, +1)$, which are not oligomorphic. Show that graph reachability is undecidable.

Exercise 80. Show a structure that is not oligomorphic, but where graph reachability is decidable.

Exercise 81. Do the same as in Exercise 80, but using a finite vocabulary.

5.2 Orbit-finite automata

In this section, we discuss the atom versions of nondeterministic and deterministic finite automata. The general idea is that these are a mild generalisation of the register automata from Chapter 1.

Orbit-finite automata. The definition of the orbit-finite automata is the same as in the finite case, except that the word "finite set" is replaced by "orbit-finite set with atoms".

Definition 5.7. A *nondeterministic orbit-finite automaton* (over an oligomorphic atom structure $\mathbb{A}$) is a tuple

$$\mathcal{A} = (\underbrace{Q}_{\text{states}} \quad \underbrace{\Sigma}_{\text{input alphabet}} \quad \underbrace{I \subseteq Q}_{\text{initial states}} \quad \underbrace{F \subseteq Q}_{\text{states}} \quad \underbrace{\delta \subseteq Q \times \Sigma \times Q}_{\text{transitions}})$$

where all components are orbit-finite sets with atoms over $\mathbb{A}$.

An automaton is called *deterministic* if it has one initial state, and δ is a function from $Q \times \Sigma$ to Q. Acceptance is defined in the usual way. The language recognised by an automaton is the set of words it accepts. By the equivariance principle, the language recognised by an automaton is supported by whatever supports the automaton, in particular it is finitely supported.

Example 5.8. Register automata, as defined in Chapter 1, are a special case of orbit-finite automata, under the equality atoms. The input alphabet is finitely many copies of the atoms – and therefore orbit-finite – while the state space is of the form

$$\mathsf{Loc} \times (\mathbb{A} \cup \{\bot\})^k,$$

and therefore also orbit-finite. Register automata are equivariant, because atom parameters cannot be used in their definitions, although allowing atom parameters in register automata would still lead to a special case of orbit-finite automata.

For orbit-finite automata, the input alphabet does not need to consist of pairs (colour from a finite set, atom), as illustrated in the following example.

Example 5.9. Consider the equality atoms. Let the input alphabet be

$$\Sigma = \{\{a, b\} : a \neq b \in \mathbb{A}\},$$

i.e. each letter is an unordered set of two distinct atoms. Consider the language "the word is empty, or some atom appears in all letters", i.e.

$$L = \epsilon \cup \{a_1 \cdots a_n \in \Sigma^* : a_1 \cap \cdots \cap a_n \neq \emptyset\}.$$

A nondeterministic orbit-finite automaton which recognizes this language has states

$$Q = \mathbb{A}.$$

All states are both initial and accepting. (This does not mean that the automaton accepts all words, because sometimes no transition will be enabled.) The idea is that the automaton guesses which atom will appear in all letters, and then scans the word to see if its guess was correct. Therefore, the transition relation is

$$\delta = \{(a, \{a, b\}, a) : a \neq b \in \mathbb{A}\}.$$

Example 5.10. The automaton from Example 5.9 example can be determinised. The deterministic automaton stores in its state the intersection of all letters it has read so far; with a special initial state indicating that it has read no letters. The initial state can be modelled as the set of all atoms, there is a rejecting sink state, and the other states as nonempty sets of atoms of size at most two.

$$Q = \overbrace{\{\mathbb{A}\}}^{\text{initial}} \cup \{\{a, b\} : \text{ for } a, b \in \mathbb{A}\} \cup \overbrace{\{\emptyset\}}^{\text{rejecting sink}}.$$

The transition function is defined by

$$\delta(X, \{a, b\}) = X \cap \{a, b\}.$$

The accepting states are all states except $\emptyset$.

Hereditarily orbit-finite automata. In Definition 5.7, we use orbit-finite sets. We now explain why not much would change by restricting to hereditarily orbit-finite sets. Define an *isomorphism* between two automata to be two bijections – one for the states and one for the input letters – which are consistent with the transition relations and accepting/final states in the natural way. By Theorem 3.23, every orbit-finite set admits a finitely supported bijection to a

hereditarily orbit-finite set (even of a very simple form, namely tuples of atoms modulo some equivalence relation). Therefore, every orbit-finite automaton is isomorphic to one that is hereditarily orbit-finite via a finitely supported isomorphism. Since isomorphism does not affect the notions for automata that we study, like determinism, minimality, emptiness or universality, we will freely confuse orbit-finite and hereditarily orbit-finite automata.

Relationship with register automata. Consider the equality atoms. As discussed in Example 5.8, register automata from Chapter 1 are a special case of orbit-finite automata. The following theorem shows that, as far as expressive power is concerned, nondeterministic orbit-finite automata do not add any new expressive power to register automata, subject to two restrictions: equivariance, and the input alphabet consists of pairs (colour from a finite set, atom). A similar result, Theorem 6.6, is true for deterministic automata, but it will only be proved in Section 6.2, when the necessary tools are available.

Theorem 5.11. *Consider the equality atoms. For every finite set* Σ *and every language* $L \subseteq (\Sigma \times \mathbb{A})^*$*, the following conditions are equivalent:*

(1) *L is recognised by a nondeterministic register automaton;*
(2) *L is recognised by an equivariant nondeterministic orbit-finite automaton.*

Proof The implication (1) $\Rightarrow$ (2) was discussed in Example 5.8. We are left with the implication (1) $\Leftarrow$ (2). Suppose that L is recognised by an equivariant orbit-finite automaton with states Q. By Theorem 3.23, there is a surjective partial equivariant function

$$f : \mathbb{A}^n \to Q$$

for some $n \in \{0, 1, \ldots\}$. The domain of f, which is an equivariant subset of $\mathbb{A}^n$, can be viewed as an equivariant subset of the state space of an n-register automaton with one location. Note how this subset does not use undefined registers $\perp$. The transitions (likewise the initial and final states) are defined by taking inverse images under f of the transitions in $\mathcal{A}$. □

Exercise 84 shows that the conditions in the above theorem are also equivalent to: (3) the language L is equivariant and is recognised by a (not necessarily equivariant) nondeterministic orbit-finite automaton.

Here are some benefits of the more general setting of orbit-finite automata.

- The definition is more similar to the usual definition of orbit-finite automata, thus justifying the choice of the register automata model as a generalisation of nondeterministic automata to infinite alphabets;

- Using the language of orbit-finite sets, it is clear how to generalise automata to atoms other than the equality atoms. In Section 7, we will see some interesting examples of atoms, which describe data structures such as trees or graphs.
- Deterministic orbit-finite automata can be minimised, unlike register automata. We discuss minimisation later in this section.
- Orbit-finite automata can consider unusual input alphabets, e.g. unordered pairs of atoms as considered in Example 5.9. The importance of such alphabets will be described in Chapter 10.

Also there is little or no price to pay for the added generality of orbit-finite sets. For example, the emptiness problem is decidable for orbit-finite automata:

Theorem 5.12. *Assume that the atoms are oligomorphic and have decidable first-order theory as in the assumptions of Corollary 4.3. Then emptiness is decidable for hereditarily orbit-finite automata, represented by set builder expressions.*

Proof The emptiness problem for nondeterministic automata reduces to graph reachability from Theorem 5.1. In the reduction, we need to compute the one-step reachability relation on states

$$E = \{(p, q) : \text{there is a transition } (p, a, q) \text{ for some input letter } a\},$$

which is done using the Third Symbol Pushing Lemma. □

Other examples of positive results that generalise easily to orbit-finite automata are: elimination of ϵ-transitions (Exercise 82) or deciding if a nondeterministic automaton is deterministic/unambiguous (Exercise 83)

The negative results about register automata transfer to orbit-finite automata. A corollary of Theorem 5.11 is that languages recognised by nondeterministic orbit-finite automata are not closed under complementation, because the same is true for register automata, see Example 1.6. Also, universality is undecidable for nondeterministic orbit-finite automata, because it is undecidable for nondeterministic register automata, see Theorem 1.8.

Minimization of deterministic automata Orbit-finite automata can have state spaces such as "unordered sets of atoms" which do not arise in register automata. An advantage of these new state spaces is that they allow minimisation via a Myhill-Nerode construction[2], unlike deterministic register automata. To

[2] Orbit-finite sets were originally introduced to model language recognisers such as automata or monoids in a machine independent way, i.e. via a theorem in the style of Myhill-Nerode (Theorem 5.14). To do this, one needed a representation of the state space which would: a) be

see the problems with minimization for register automata, consider the following example, which is based on Exercise 3.

Example 5.13 (Automata with registers do not minimise)**.** Consider the equality atoms. Let the input alphabet be the atoms, and consider the language of words where at most two atoms appear, possibly with repetitions. To recognize this language, we can use a deterministic automaton with two registers. If we view this deterministic automaton as an orbit-finite automaton, then its state space is

$$\underbrace{(\mathbb{A} \cup \{\bot\})}_{\substack{\text{possible values} \\ \text{of register 1}}} \times \underbrace{(\mathbb{A} \cup \{\bot\})}_{\substack{\text{possible values} \\ \text{of register 2}}} \quad \cup \quad \{\text{reject}\}.$$

The automaton begins in the state $(\bot, \bot)$. When it sees an atom which is not in the registers, it loads it into the first undefined register, if both registers are full it rejects. In particular, states of the form $(\bot, a)$ are unreachable.

We have just described a deterministic automaton, which recognizes the language, and which uses registers. The problem with this automaton is that it does not store the minimal amount of information. Because the registers are ordered, the states (a, b) and (b, a) are different, but they are equivalent in the sense that they accept the same words. To get the minimal automaton, we should have states which are unordered sets, i.e. the state space should be

$$\{\{a, b\} : \text{ for } a, b \in \mathbb{A}\} \quad \cup \quad \{\emptyset, \bot\}.$$

The above state space is orbit-finite, but it cannot be represented as the state space of any register automaton. This example shows that in order to store the minimal amount of information, registers – as opposed to orbit-finite sets – are not always the right choice.

We now show that the above example is not an accident, and in fact all deterministic orbit-finite automata can be minimised. Suppose that $L \subseteq \Sigma^*$ is a language. Its *Myhill-Nerode equivalence* is the equivalence relation on Σ^* defined by

$$w \sim w' \qquad \text{if} \qquad \forall v \in \Sigma^* \; wv \in L \Leftrightarrow w'v \in L.$$

simple enough to allow finite representations; b) generalise the configuration space of a register automaton; and c) allow quotienting as required in the Myhill-Nerode theorem. Orbit-finite sets are a solution to these requirements. The Myhill-Nerode theorem for orbit-finite sets was originally proved in (Bojańczyk, 2013, Lemma 3.3) for monoids and then in (Bojańczyk et al., 2014, Theorem 3.8) for automata; these respective papers are the ones which introduced orbit-finite monoids (and orbit-finiteness in general) and orbit-finite automata. A variant of the Myhill-Nerode theorem for timed automata is presented in Bojańczyk and Lasota (2012b); the difficulty there is to deal with atoms which are not oligomorphic.

By the principle of equivariance, the Myhill-Nerode equivalence relation is finitely supported, assuming that the language itself was finitely supported. A classical result of automata theory says that (even if the alphabet is infinite and the language is not necessarily regular) the Myhill-Nerode equivalence relation is a congruence with respect to appending letters:

$$w \sim w' \quad \text{implies} \quad wa \sim wa' \qquad \text{for every } a \in \Sigma,$$

and therefore it makes sense to consider an automaton where the states are the equivalence classes, and where the transition function is defined by

$$(\text{equivalence class of } w, a) \qquad \mapsto \qquad \text{equivalence class of } wa.$$

This automaton is called the *syntactic automaton of L*.

Theorem 5.14. *A language is recognised by a deterministic orbit-finite automaton if and only if its syntactic automaton has an orbit-finite state space.*

Proof The right-to-left implication is immediate. For the left-to-right implication, we observe that states of the syntactic automaton can be obtained from the states of an arbitrary deterministic automaton recognising the language, by quotienting under the finitely supported equivalence relation defined by

$$q \sim q' \qquad \text{if} \qquad q \text{ and } q' \text{ accept the same words.}$$

Since quotienting under finitely supported equivalence relations preserves orbit-finiteness, it follows that the syntactic automaton must have an orbit-finite state space, if the original automaton did. □

The proof of the above theorem also yields a minimisation algorithm[3], which is the atom generalisation of the Moore algorithm. We revisit this algorithm in Example 8.9. Minimisation will even fall under the scope of "tractable computation", as described in Chapter 9.

Exercises

Exercise 82. Show that adding ϵ-transitions does not change the expressive power of nondeterministic orbit-finite automata.

Exercise 83. Assume that the atoms are oligomorphic and have decidable first-order theory with constants. Show that one can check if a nondeterministic orbit-finite automaton is deterministic. Likewise for unambiguous (each input admits at most one accepting run).

[3] The computational complexity of automata minimisation is studied in Murawski et al. (2015).

Exercise 84. Assume that the atoms are oligomorphic. Consider a language that is recognised by an orbit-finite nondeterministic automaton. Show that if the language is supported by a tuple of atoms $\bar{a}$, then it is also recognised by a orbit-finite nondeterministic automaton which is supported by $\bar{a}$.

Exercise 85. Same as Exercise 84, but for deterministic automata.

Exercise 86. Consider the following weakening of Minsky machines. The automaton has a finite set of states, as well as a finite set of counters, which store natural numbers. The automaton can test a counter for zero. Instead of the increment and decrement operations in Minsky machines, the automaton can execute operations of the form "make counter c strictly bigger" and "make counter c strictly smaller". The model is nondeterministic, since the automaton does not control the aumount by which the counter is increased or decreased. The automaton accepts by reaching an accepting state. Show that emptiness is decidable.

Exercise 87. Assume that the atoms are oligomorphic. Show that the class of languages recognised by nondeterministic orbit-finite automata is closed under orbit-finite union, in the sense of Exercise 62.

Exercise 88. Assume the equality atoms. Show that languages recognised by nondeterministic orbit-finite automata (same for deterministic) are not closed under orbit-finite intersection.

Exercise 89. Assume the equality atoms. Show that languages recognised by nondeterministic orbit-finite automata are not closed under orbit-finite intersection, in the sense defined in Exercise 62.

Exercise 90. Consider the following extension of register automata to arbitrary orbit-finite alphabets: the state space is of the form

$$\mathsf{Loc} \times (\{\bot\} \cup \mathbb{A})^R$$

for some finite (not just orbit-finite) sets Loc and R. Show that such an automaton cannot recognise the language from Example 5.9.

Exercise 91. 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 92. 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 to the sub-monoid of functions $Q \to Q$ which is generated by the state transition functions $\{q \mapsto qa\}_{a \in \Sigma}$ of the syntactic automaton.

Exercise 93. 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 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 94. 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 95. 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?

5.3 Pushdown automata and context-free grammars

In this section, we discuss orbit-finite variants of pushdown automata[4] and context-free grammars. We show that basic results, such as equivalence of pushdown automata and context-free grammars, or decidability of emptiness, transfer easily to the orbit-finite setting. We also motivate the models by giving examples of automata and grammars that use atoms.

[4] Context-free languages for infinite alphabets were originally introduced by Cheng and Kaminski (1998), whose 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).

Definition 5.15. An *orbit-finite pushdown automaton* consists of

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

and a transition relation

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

where all components are orbit-finite.

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 an orbit-finite pushdown grammar.

Definition 5.16. An *orbit-finite context-free grammar* consists of

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

where all components are orbit-finite.

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

By the equivariance principle, the languages corresponding to pushdown automata and grammars inherit the supports of their respective devices.

Example 5.17. [Pushdown automaton for palindromes.] For an orbit-finite alphabet Σ, consider the language of palindromes, i.e. words which are equal to their reverse. This language is recognised by a orbit-finite pushdown automaton which works exactly the same way as the usual automaton for palindromes, with the only difference that the stack alphabet Γ is now an orbit-finite 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 5.18. [Pushdown automaton for modified palindromes.] The automaton in Example 5.17 had two control states. In some cases, it might be useful to have a set Q of control states that is orbit-finite. 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 5.19 (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 orbit-finite sets. Consider the atoms $(\mathbb{Q}, <)$ and 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
    return b
  else if b > a then  // the program can use the order on atoms
    return f(b) // do a recursive call
  else
    fail() // terminate the computation
end
```

The behaviour of this program can be modelled by an orbit-finite 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 is orbit-finite, and therefore the stack alphabet is orbit-finite.

Orbit-finite pushdown automata could also be used to model more sophisticated examples: many mutually recursive functions, boolean variables, other homogeneous data types for the atoms.

In the above examples, we considered the oligomorphic atoms, where (hereditarily) orbit-finite sets are exactly those defined by set builder expressions. As for automata, when the atoms are oligomorphic, we make little distinction between orbit-finite pushdown automata and hereditarily orbit-finite pushdown automata (and therefore also pushdown automata defined by set builder expressions), because these are the same up to finitely supported isomorphisms. When the atoms are not oligomorphic, set builder expressions can still be used, unlike orbit-finite sets. The following theorem shows that, even without oligomorphism, context-free grammars and pushdown automata are equivalent, assuming that the devices are described using set builder expressions.

Theorem 5.20. *Consider atoms that are not necessarily oligomorphic, and do not necessarily have decidable first-order theory. The following models recognise the same languages:*

- *Pushdown automata represented by set builder expressions;*
- *Context-free grammars represented by set builder expressions.*

The constructions are effective[5] *and in polynomial time.*

Proof We just redo the classical constructions, which are so natural that they easily go through with sets represented by set builder expressions.

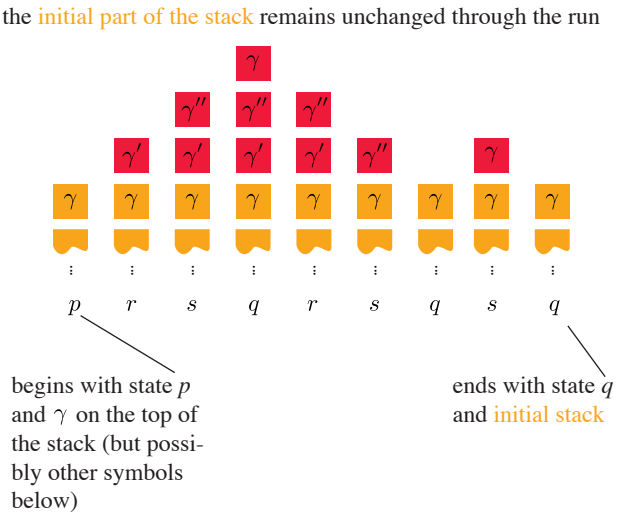
- *From a pushdown automaton to a context-free grammar.* Without loss of generality, we assume that each transition either: pops nothing and pushes one symbol; or pops one symbol and pushes nothing. We also assume that in every accepting run, the stack is nonempty until the last configuration. Every 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, using the Symbol Pushing Lemma.

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

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

  A set builder expression for this set can easily be computed, based on the set builder expression for the pushdown automaton. The language generated by a nonterminal (p, γ, q) is going to be the set of words which label runs of the following form:

[5] Here we use the same computation model as in the Symbol Pushing Lemmas, namely that an algorithm can copy, at unit cost, atom parameters that appear in set builder expressions.



To describe these runs, we use the following grammar rules. All of the sets below can be described by set builder expressions using the Symbol Pushing Lemmas:

(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 transitions

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

there is a rule in the grammar of the form:

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

(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 in the grammar of the form:

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

- *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).

□

When the atoms are furthermore oligomorphic and have decidable first-order theory as in the assumptions of Corollary 4.3, then emptiness is decidable for context-free grammars. The idea is to use the same kind of fixpoint algorithm as in Theorem 5.1 about graph reachability. In Chapter 9, we show that the fixpoint algorithm is even "polynomial time", under a suitable definition.

Exercises

Exercise 96. Consider the equality atoms. We say that two sets with atoms X, Y are *fresh* with respect to each other if they can be supported by disjoint tuples of atoms. Assume that the input alphabet is the atoms. Consider the extension[6] of orbit-finite pushdown automata, as per Definition 5.15, where a new kind of transition is allowed:

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

When executing this transition, the automaton reads letter a and changes state from q to p, but only under the condition that a is fresh with respect to every letter on the stack and the current state q. Show that emptiness is decidable.

Exercise 97. Consider the equality atoms and the following higher-order variant of orbit-finite pushdown automata[7]. 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 98. Show that a language that is orbit-finite context-free, but is not generated by any orbit-finite context-free grammar with a finite (not just orbit-finite) set of nonterminals.

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

5.4 Graph homomorphisms

In this case study, we consider graph homomorphisms[8]. One of the purposes of this case study is to use the following effectivity assumption on the atoms.

Definition 5.22. An oligomorphic structure $\mathbb{A}$ is said to have a *a computable Ryll-Nardzewski function* if given $n \in \{1, 2, \ldots\}$ one can compute the number equivariant orbits in $\mathbb{A}^n$.

Example 5.23. In the equality atoms, the number of equivariant orbits is the number of equality types, which is the Bell number, and can be computed. Likewise for $(\mathbb{Q}, <)$, except using order types. Therefore each of these structures has a computable Ryll-Nardzewski function.

All oligomorphic structures discussed in this book have a computable Ryll-Nardzewski function. It is hard to come up with a natural structure that is oligomorphic, has a decidable first-order theory, but has a non-computable Ryll-Nardzewski function. Nevertheless, using forcing, this is possible, see Schmerl (1978).

The assumption on a computable Ryll-Nardzewski function turns out to be useful when we want to compute the partition of an orbit-finite set into orbits, e.g. for the purpose of enumerating all subsets with a given support. This will be the case below, when studying a decision problem about homomorphisms.

We begin by defining the problem.

Definition 5.24 (Graph homomorphism)**.** A *homomorphism*

$$h : G_1 \to G_2$$

between directed graphs is a function from vertices in G_1 to vertices in G_2 which takes edges to edges, in the sense that the following implication holds:

$$G_1 \text{ has an edge from } v \text{ to } w \quad \text{implies} \quad G_2 \text{ has an edge from } h(v) \text{ to } h(w).$$

We consider the decision problem of checking if a homomorphism exists, given two hereditarily orbit-finite graphs. There are three variants of this decision problem – see (1), (2) and (3) below – depending on the requirements for the support of the homomorphism.

- **Input.** Two hereditarily orbit-finite directed graphs G_1 and G_2.
- **Output.** Is there a homomorphism from G_1 to G_2 which is:

(1) supported by a given tuple $\bar{a}$?
(2) finitely supported?

[8] This case study is based on Klin et al. (2016)

(3) not necessarily finitely supported?

The three variants are indeed different, as illustrated in the following examples. We show in this section that variant (1) is decidable. Variants (2) and (3) are undecidable[9], but we do not show this. If we modify variant (3) to ask for a not necessarily finitely supported *isomorphism*, we get an open problem.

Example 5.25 (Variants (1) and (2) have different answers)**.** Consider the equality atoms. By the equivariance principle, any tuple of atoms that supports a homomorphism must also support its domain. In particular, if a tuple $\bar{a}$ does not support the vertices of the input graph, then there cannot be a $\bar{a}$-supported homomorphism from G_1 to G_2. Therefore, if we take $G_1 = G_2$ to be some graph that is finitely supported but not supported by $\bar{a}$, then there exists a homomorphism, but not any homomorphism that is supported by $\bar{a}$. We now give a slightly more interesting example, where $\bar{a}$ supports both graphs G_1 and G_2, there is a finitely supported homomorphism, but there is no homomorphism that is supported by $\bar{a}$.

Let $\bar{a}$ be the empty tuple. Both graphs G_1 and G_2 have no edges. The graph G_1 has exactly one vertex v which is equivariant, while the vertices of G_2 are the atoms $\mathbb{A}$. The homomorphisms between these two graphs are exactly the functions

$$h : \{v\} \to \mathbb{A}.$$

Clearly there is such a function if we allow finite support, e.g. $v \mapsto \underline{1}$, but there is no such function with empty support.

Example 5.26 (Variants (2) and (3) have different answers)**.** We now show that there might be a homomorphism, but none that is finitely supported. Consider the equality atoms. Both graphs G_1 and G_2 are cliques without self-loops, i.e. vertices are connected by an edge if and only if they are different. The vertices of G_1 are $\mathbb{A}^2$ and the vertices of G_2 are $\mathbb{A}$. The homomorphisms between these two graphs are exactly the injective functions

$$h : \mathbb{A}^2 \to \mathbb{A},$$

because mapping two vertices in G_1 to the same vertex in G_2 would require a self-loop. Since both sets are countable, there is clearly a homomorphism h, if we do not require finite supports. We claim that there is no finitely supported homomorphisms. To see this, suppose h is finitely supported and injective. Take distinct atoms atoms a, b that are not in the support of h. Assume that $h(a, b) \neq a$, the case of $h(a, b) \neq b$ is treated the same way. Take π to be the

[9] (Klin et al., 2016, Theorems 14 and 12, respectively)

atom automorphism which swaps a with some atom a' that is also not in the support of h, which yields

$$h(a, b) = \pi(h(a, b)) = h(a', b)$$

contradicting injectivity.

The rest of this section is devoted to showing that variant (1) of the homomorphism decision problem – where we are given explicitly a support of the homomorphism – is decidable, assuming that the atoms are oligomorphic, have decidable first-order theory with constants, and a computable Ryll-Nardzewski function.

We are given hereditarily orbit-finite directed graphs G_1, G_2 and a tuple of atoms $\bar{a}$. We want to decide if there exists a homomorphism from G_1 to G_2 that is supported by $\bar{a}$. The algorithm is very straightforward: enumerate through all $\bar{a}$-supported functions, and check if one of them is a homomorphism. It remains to explain how, under the assumptions on the atom structure, this algorithm can be implemented.

When seen as a set of pairs, a homomorphism from G_1 to G_2 is a subset of

$$h \subseteq V_1 \times V_2 \qquad \text{where } V_i \text{ are the vertices of } G_i.$$

If h is supported by $\bar{a}$, then it is a union of $\bar{a}$-orbits. By oligomorphism, the union is finite. The subtle point is that we also need to compute these orbits. This is done using the following lemma.

Lemma 5.27. *Assume that the atoms are oligomorphic, have decidable first-order theory with constants, and a computable Ryll-Nardzewski function. Given a hereditarily orbit-finite set X and a tuple of atoms $\bar{a}$, one can compute the list of all $\bar{a}$-orbits which intersect X.*

Before proving the lemma, we finish the algorithm for Question 1. Apply the lemma to $V_1 \times V_2$, yielding a list of $\bar{a}$-orbits. For every h which is a union of these orbits, use the Symbol Pushing Lemmas to check if h is a homomorphism. It remains to prove the lemma.

Proof of Lemma 5.27 The idea is to first consider sets of tuples, and then lift that result to hereditarily orbit-finite sets. The case of sets of tuples is treated in the following claim, which uses the assumption on the Ryll-Nardzewski function.

Claim 5.28. *Given a tuple of atoms $\bar{a}$ and $n \in \mathbb{N}$, one can compute the partition*

$$\mathbb{A}^n = Y_1 \cup \cdots \cup Y_k$$

into $\bar{a}$-orbits, with each $\bar{a}$-orbit Y_i represented by a formula of first-order logic that uses constants from $\bar{a}$.

Proof

- Consider first the case when $\bar{a}$ is empty. The number k of orbits is given by the Ryll-Nardzewski function, and therefore can be computed. It remains to find the first-order formulas. By Lemma 4.11, such formulas exist. Therefore we can use exhaustive enumeration until we find k formulas without constants, each one with n free variables, such that the corresponding subsets of $\mathbb{A}^n$ are nonempty and pairwise disjoint.
- Consider the general case. Let m be the length of the tuple $\bar{a}$ in the assumption of the claim. Apply the previous case to $\mathbb{A}^{m+n}$, yielding the partition

  $$\mathbb{A}^{m+n} = Y_1 \cup \cdots \cup Y_k.$$

  For each part Y_i, define Z_i to be the tuples $\bar{z} \in \mathbb{A}^n$ such that $\bar{a}\bar{z} \in Y_i$. Two tuples in $\mathbb{A}^n$ are in the same $\bar{a}$-orbit if and only if they belong to the same Z_i. Therefore, the formulas defining the nonempty sets Z_i are the ones required by the statement of the lemma.

This completes the proof of the claim. □

We now use the claim to compute the partition into orbits for sets that are not necessarily subsets of $\mathbb{A}^n$. Let X be a hereditarily orbit-finite set. If X is given by a union set-builder expression, then we can compute the lists of orbits for each component of the union, and then put them together. (If we want the list of orbits to avoid repetitions, we can use the Symbol Pushing Lemma to check which sets in the list are equal.) We can therefore assume that X is defined bvy a set builder expression of the form

$$\{\beta(\bar{y}) : \text{ for } \bar{y} \in \mathbb{A}^n \text{ such that } \varphi(\bar{y})\}.$$

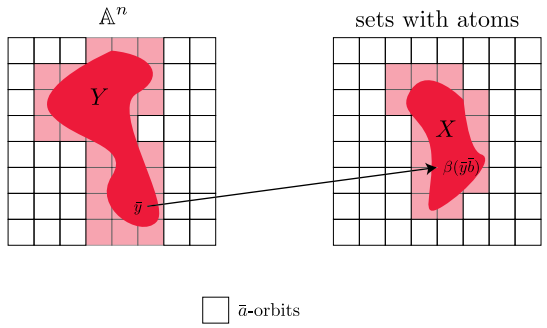
Define $Y \subseteq \mathbb{A}^n$ to be the set of tuples $\bar{y}$ which satisfy $\varphi(\bar{y}\bar{b})$. The set X is the image of Y under the function

$$\bar{y} \quad \mapsto \quad \beta(\bar{y}). \tag{5.3}$$

Apply the special case of the lemma for atom tuples to Y yielding a list

$$Y_1, \ldots, Y_n \subseteq \mathbb{A}^n$$

of all the $\bar{a}$-orbits that intersect Y, each one described by a formula of first-order logic possibly using constants from $\bar{a}$. Here is a picture:



The $\bar{a}$-orbits that intersect X are exactly the images of these orbits under the function (5.3). These are clearly represented by set builder expressions. □

Exercises

Exercise 99. Let $\mathbb{A}$ be a countable oligomorphic structure with a computable Ryll-Nardzewski function. Show that if $\mathbb{A}$ has a decidable first-order theory without constants, then the same is true with constants.

Exercise 100. Assume that $\mathbb{A}$ is oligomorphic and has decidable first-order theory with constants. Show that the following conditions are equivalent:

(1) given $n \in \mathbb{N}$, one can compute a first-order formula with $2n$ free variables that defines the "same equivariant orbit" on $\mathbb{A}^n$;

(2) the Ryll-Nardzewski function is computable.

Exercise 101. Assume the equality atoms. A *Büchi game* has the same syntax as alternating reachability from Exercise 78. 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).

5.5 Systems of equations

Consider a system of equations[10] 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 would need to get the same value, violating the equations. The system had finitely many equations. In this section, we consider systems where the set of equations is orbit-finite, but each individual equation is finite.

Example 5.30. 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 choice function, which chooses for every atoms $a \neq b \in \mathbb{A}$ exactly one of the pairs (a,b) or (b,a). It follows by Example 3.5 that the above system has a solution, but no finitely supported solution.

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 5.31. *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 3.23, there is some $k \in \{0, 1, 2 \ldots\}$ and an equivariant surjective function

$$f : \mathbb{A}^k \to X.$$

[10] This section is based on Klin et al. (2015)

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 if $\mathcal{E}$ 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 5.32. *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 5.31, it is enough to check if the system has an equivariant solution. By Lemma 5.27, 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 102. 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 103. How is decidability of the problem in Excercise 102 affected 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 104. Consider the following atoms[11]. 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 coordinatewise. 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.

[11] Suggested by Szymon Toruńczyk.

6
Least supports

In this chapter we show that in the equality atoms, one can always find a least support, i.e. a support that is contained in all other supports. This result is also true for some other types of atoms, but we only prove it for the equality atoms. We use least supports to get a representation theorem for orbit-finite sets in the equality atoms, which is stronger than the representation theorem from Theorem 3.23 about atom tuples modulo partial equivalence. Using the stronger representation theorem, we prove that deterministic register automata have the same expressive power as deterministic orbit-finite automata, assuming input letters are pairs of the form (label from a finite set, atom).

For the rest of this chapter, we assume the equality atoms.

6.1 Least supports

In this book, we generally use tuples of atoms as supports. An alternative is to use finite sets of atoms, because whether or not a tuple $(a_1, \ldots, a_n)$ supports a set with atoms does not depend on the ordering and repetitions in the tuple. Therefore, it makes sense talking about one support being contained in some other support. The following theorem shows that there is a least finite support[1].

Theorem 6.1 (Least Support Theorem)**.** *Assume the equality atoms. For every x which is an atom or a set with atoms, there is a finite support that is contained in all finite supports of x.*

Another way of stating the above theorem is that finite supports are closed under intersection. It is important that we consider finite supports. For example,

[1] 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, kind be found in (Bojańczyk et al., 2014, Section 10).

the atom $\underline{1}$ is supported by the infinite set $\mathbb{A} - \{\underline{1}\}$, since fixing this set is the same as fixing $\underline{1}$. The intersection of the two supports $\{\underline{1}\}$ and $\mathbb{A} - \{\underline{1}\}$ is empty, but $\underline{1}$ does not have empty support.

Let us write $\mathbb{A}^{(n)}$ for the set of non-repeating n-tuples of atoms. This is an equivariant single-orbit set. The key observation is the following lemma, which says that one can represent every equivariant single-orbit set as non-repeating tuples modulo an equivalence relation, such that equivalent tuples must necessarily agree as sets.

Lemma 6.2. *For every equivariant single-orbit set X there is an equivariant surjective function*

$$f : \mathbb{A}^{(n)} \to X \qquad \text{for some } n \in \{0, 1, 2, \ldots\}$$

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

$$f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n) \qquad \text{implies} \qquad \{a_1, \ldots, a_n\} = \{b_1, \ldots, b_n\}.$$

Proof By Lemma 3.20 there is an equivariant surjective function

$$f : Y \to X \qquad \text{for some equivariant } Y \subseteq \mathbb{A}^n.$$

Take some equivariant orbit of f, with f viewed as a subset of $Y \times X$. This orbit is still an equivariant function that is onto X. In other words, we can assume without loss of generality that Y is a single equivariant orbit in $\mathbb{A}^n$. Such an orbit is an equality type. By projecting away the duplicated coordinates in the equality type, we can assume that Y contains only nonrepeating tuples. Summing up, we know that there is a surjective equivariant function

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

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

$$f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n) \tag{6.1}$$

holds for some tuples $\bar{a}, \bar{b}$ which are not equal as sets. Without loss of generality we assume that a_n does not appear in the tuple $\bar{b}$. Choose some atom automorphism π which fixes $a_1, \ldots, a_{n-1}, b_1, \ldots, b_n$ but does not fix a_n. We have

$$f(\bar{a}) \overset{(6.1)}{=} f(\bar{b}) \overset{\pi \text{ fixes } \bar{b}}{=} f(\pi(\bar{b})) \overset{\text{equivariance}}{=} \pi(f(\bar{b})) \overset{(6.1)}{=} \pi(f(\bar{a})) \overset{\text{equivariance}}{=} f(\pi(\bar{a})).$$

Therefore, we have shown that

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

The set of tuples $a, a_1, \ldots, a_n$ which satisfies the condition above is equivariant subset of $\mathbb{A}^{(n+1)}$, by equivariance of f. Therefore, if some tuple satisfies the condition, then all tuples in $\mathbb{A}^{(n+1)}$ 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 $n-1$ coordinates. Therefore,

$$\{((a_1, \ldots, a_{n-1}), f(a_1, \ldots, a_n)) : a_1, \ldots, a_n \in \mathbb{A}^{(n)}\}$$

is an equivariant surjective function from $\mathbb{A}^{(n-1)}$ to X, and we can use the induction assumption. □

Proof of the Least Support Theorem. Let x be an atom or a set with atoms. Apply Lemma 6.2 to the equivariant orbit of x:

$$X = \{\pi(x) : \pi \text{ is an atom automorphism}\}$$

yielding some equivariant function

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

where tuples with equal images must be equal as sets. 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 $a_1, \ldots, a_n$, and therefore one can choose some $\bar{b}$-automorphism which does not preserve the set $\{a_1, \ldots, a_n\}$. We have

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

Since the 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 for equality atoms

Apart from the Least Support Theorem, another application of Lemma 6.2 is the following representation theorem for equivariant orbit-finite sets in the equality atoms. Let X be an equivariant single-orbit set. Apply Lemma 6.2, yielding an equivariant function

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

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

$$f(a_1, \ldots, a_n) = f(a_{g(1)}, \ldots, a_{g(n)}) \text{ for some } (a_1, \ldots, a_n) \in \mathbb{A}^{(n)} \tag{6.2}$$

$$f(a_1, \ldots, a_n) = f(a_{g(1)}, \ldots, a_{g(n)}) \text{ for every } (a_1, \ldots, a_n) \in \mathbb{A}^{(n)}. \tag{6.3}$$

Permutations g which satisfy condition (6.3) form a group, call it G. We claim:

$$f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)$$
$$\text{iff}$$
$$\exists g \in G \; (a_1, \ldots, a_n) = (b_{g(1)}, \ldots, b_{g(n)}).$$

The bottom-up implication is by definition. For the top-down implication, recall that Lemma 6.2 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

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

to be $\mathbb{A}^{(n)}$ 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[2]:

Theorem 6.3 (Representation for orbit-finite sets in the equality atoms)**.** *Assume the equality atoms. Every equivariant single-orbit set admits an equivariant bijection to a set of the form*

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

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

Example 6.4. Let $n \in \{1, 2, \ldots\}$ and let G be the group of all permutations of $\{1, \ldots, n\}$. In this case $\mathbb{A}^{(n)}/_G$ is the same as unordered sets of atoms with exactly n elements. If G is the group of cyclic shifts, then $\mathbb{A}^{(n)}/_G$ is n-tuples of distinct atoms modulo cyclic shifts.

Exercises

Exercise 105. Show an oligomorphic atom structure which fails the Least Support Theorem, as stated in Theorem 7.27.

[2] 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).

Exercise 106. Assume the equality atoms. Let S, T be finite sets of atoms. Show that every atom automorphism π which fixes $S \cap T$ can be presented as a composition

$$\pi = \pi_1 \circ \cdots \circ \pi_n$$

such that each π_i is an atom automorphism that fixes either S or T.

Exercise 107. For a set with atoms X, let us write $\sup(X)$ for the set of atoms in its least support. Let X be an orbit-finite set, and let

$$X = X_1 \cup \cdots \cup X_n$$

be its partition into orbits with respect to the least support (i.e. with respect to atom automorphisms that are the identity on the least support). Show that

$$\sup(X) = \sup(X_1) \cup \cdots \cup \sup(X_n).$$

Exercise 108. Show that the atoms $(\mathbb{Q}, <)$ also have least supports.

Exercise 109. Show an example of oligomorphic atoms without least supports.

Exercise 110. Assume the equality atoms. Show that if a group is orbit-finite, then it is finite.

Exercise 111. Does Exercise 110 generalise to all oligomorphic choices of the atoms?

Exercise 112. Assume that the atoms are oligomorphic and admit least supports. Let X be an orbit-finite set and let $f : X^n \to X$ be a finitely supported function. Show that there exists $k \in \mathbb{N}$ and finitely supported functions

$$g : \mathbb{A}^k \to X \qquad f' : \mathbb{A}^{n \cdot k} \to \mathbb{A}^k$$

which make the following diagram commute

$$\begin{array}{ccc} \mathbb{A}^{n \cdot k} & \xrightarrow{(g,\ldots,g)} & X^n \\ {\scriptstyle f'}\downarrow & & \downarrow{\scriptstyle f} \\ \mathbb{A}^k & \xrightarrow[g]{} & X \end{array}$$

Exercise 113. Assume the equality atoms. Show that if $f : X \to X$ is a finitely supported surjective function, and X is orbit-finite, then f is a bijection.

6.2 Extended example: deterministic automata

In the case study on orbit-finite automata from Section 5.2, we showed in Theorem 5.11 that nondeterministic register automata have the same expressive power as nondeterministic orbit-finite automata, for alphabets where the two notions can be compared. Using the representation result from Theorem 6.3, we prove a similar result for deterministic automata.

Theorem 6.6. *Assume the equality atoms. For every finite set Σ and every language $L \subseteq (\Sigma \times \mathbb{A})^*$, the following conditions are equivalent:*

(1) *L is recognised by a deterministic register automaton;*
(2) *L is recognised by an equivariant deterministic orbit-finite automaton.*

By Exercise 85, the conditions in the above theorem are also equivalent to: (3) the language L is equivariant, and it is recognised by a (not necessarily equivariant) deterministic orbit-finite automaton.

The rest of Section 6.2 is devoted to proving the above theorem. In the proof, we use an intermediate automaton model, based on the following definition (which can be used for any atoms, not just the equality atoms that are considered in this chapter).

Definition 6.7 (Straight set)**.** A *straight set* is a set which admits a finitely supported bijection with a a set of the form

$$\mathbb{A}^{n_1} + \cdots + \mathbb{A}^{n_k} \qquad \text{for some } k, n_1, \ldots, n_k \in \{0, 1, \ldots\}.$$

If the bijection is equivariant, then we talk about an equivariant straight set[3] .

Examples of straight sets are: input alphabets of register automata; state spaces of register automata, and sets of the form $\mathbb{A}^{(n)}$ used in Theorem 6.3. A non-example is the set of unordered pairs of atoms $\{\{a, b\} : a \neq b \in \mathbb{A}\}$, which created problems for choice in Example 3.5.

We prove Theorem 6.6 in two steps:

deterministic orbit-finite equivariant automata $\overset{\text{Lemma 6.9}}{=}$ deterministic orbit-finite equivariant automata with straight state spaces $\overset{\text{Lemma 6.8}}{=}$ deterministic register automata

[3] These sets are also known as *strong nominal sets*.

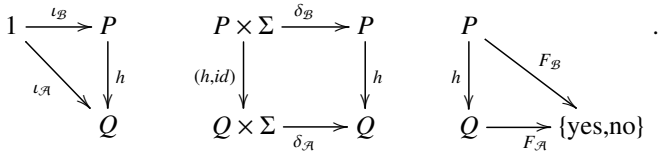
In the proofs, we use categorical notation for automata, i.e. an automaton $\mathcal{A}$ over an input alphabet Σ consists of a state space Q and three functions

$$\underbrace{\iota_{\mathcal{A}} : 1 \to Q}_{\text{initial state}} \qquad \underbrace{\delta_{\mathcal{A}} : Q \times \Sigma \to Q}_{\text{transition function}} \qquad \underbrace{F_{\mathcal{A}} : Q \to \{\text{yes,no}\}}_{\text{accepting states}},$$

where 1 stands for a set which has a unique equivariant element, e.g. $1 = \{\emptyset\}$. We care about automata which are equivariant, i.e. all of the functions described above and the sets that they use are equivariant. A *homomorphism* of automata with the same input alphabet

$$\mathcal{B} \xrightarrow{\ h\ } \mathcal{A}$$

is a function from the states of $\mathcal{B}$ (call them P) to the states of $\mathcal{A}$ (call them Q) which makes the following diagrams commute:



It is not hard to see that if there is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$, then the two automata recognise the same language. We use homomorphisms to prove that deterministic automata recognise the same languages in Lemmas 6.8 and 6.9 below, which will complete the proof of Theorem 6.6.

Lemma 6.8. *Deterministic register automata recognise the same languages as equivariant deterministic orbit-finite automata with a straight state spaces.*

Proof The state space of a deterministic register automaton is clearly a straight orbit-finite set, which gives the left-to-right inclusion in the lemma. For the converse inclusion, consider a deterministic orbit-finite automaton with a straight state space Q. Let k be the number of orbits in Q and let n be the maximal dimension of tuples used in Q. It is easy to see that there is an equivariant injective function

$$h : Q \to \underbrace{\{1, \ldots, k\} \times (\mathbb{A} \cup \{\bot\})^n}_{P}.$$

Using h and its (one-sided) inverse, one can impose an automaton structure on P which turns h into an automaton homomorphism. The target of this homomorphism is a register automaton with k locations and n registers. □

The more difficult step is turning the state space of a deterministic orbit-finite automaton into a straight set. This is done using the representation theorem from the previous section.

Lemma 6.9. *For every equivariant deterministic orbit-finite automaton, there is an equivalent one with a straight state space.*

Proof Consider an equivariant deterministic orbit-finite automaton $\mathcal{A}$ with state space Q that is not necessarily straight. Apply Theorem 6.3, yielding a representation of Q as

$$\mathbb{A}^{(n_1)}_{/G_1} + \cdots + \mathbb{A}^{(n_k)}_{/G_k}$$

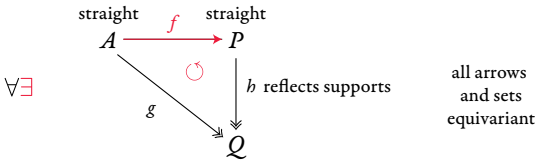
for some dimensions n_i and groups G_i. Define P to be the straight set

$$\mathbb{A}^{(n_1)} + \cdots + \mathbb{A}^{(n_k)}$$

and define $h : P \to Q$ to be the surjective function which quotients a tuple with respect to the appropriate group action. The function h is surjective and equivariant. To prove the lemma, we will show that one can define an automaton structure on P which turns h into a homomorphism of automata.

In Lemma 6.8, defining the homomorphism was easy, because h had a (one-sided) inverse. This is no longer true in our case. Nevertheless, a weaker property holds, namely *h reflects supports* in the sense that if a tuple of atoms supports $h(p) \in Q$, then it also supports p (the opposite implication is also true, thanks to equivariance). The following claim shows that support-reflecting functions with straight domains admit a certain form of choice.

Claim 6.10.

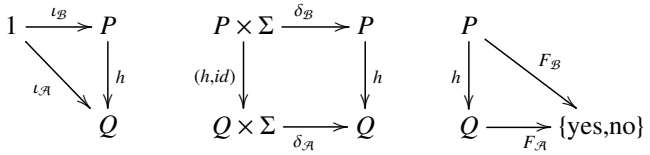


Proof Take some $\bar{a} \in A$, which is a tuple of atoms because A is straight. By surjectivity of h, there is some $\bar{b} \in P$ such that

$$g(\bar{a}) = h(\bar{b}).$$

Because g preserves supports and h reflects supports, it follows that $\bar{a}$ supports $\bar{b}$. It follows that $\bar{a} \mapsto \bar{b}$ can be extended to an equivariant function from the equivariant orbit of $\bar{a}$ to the equivariant orbit of $\bar{b}$. By doing this for all orbits in A, we get the result. □

We now define an equivariant automaton structure $\mathcal{B}$ on P which turns h into a homomorphism of automata, i.e. it makes the following diagrams commute:



The initial state $\iota_{\mathcal{B}}$ and the transition function $\delta_{\mathcal{B}}$ is defined using Claim 6.10, while acceptance is defined as the composition $F_{\mathcal{A}} \circ h$. □

Exercises

Exercise 114. Assume the equality atoms. Let X be a straight set with atoms. Show that for every set with atoms Y and every finitely supported

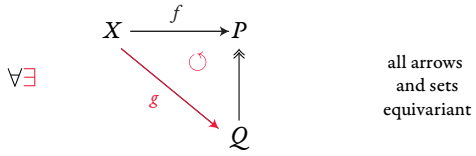
$$F : X \to \text{nonempty finitely supported subsets of } Y$$

there exists a finitely supported function $f : X \to Y$ such that

$$f(x) \in F(x) \quad \text{for every } x \in X.$$

Exercise 115. Show that Exercise 114 fails in the atoms $(\mathbb{Q}, <)$.

Exercise 116. Assume the equality atoms. Show that an equivariant orbit-finite set X is straight if and only if it is projective in the sense of category theory:



Exercise 117. Show an example of a function which preserves and reflects supports, but which is not equivariant

7

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 is embeds all of 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.

7.1 Homogeneous structures

Consider two structures $\mathbb{A}, \mathbb{B}$ over the same vocabulary, which may include function symbols. An *embedding* $f : \mathbb{A} \to \mathbb{B}$ is any injective function from the inverse of $\mathbb{B}$ to the universe of $\mathbb{A}$ which is preserves and reflects the relations in the following sense

$$\underbrace{R(a_1, \ldots, a_n)}_{\text{in } \mathbb{A}} \quad \text{iff} \quad \underbrace{R(f(b_1), \ldots, f(b_n))}_{\text{in } \mathbb{B}},$$

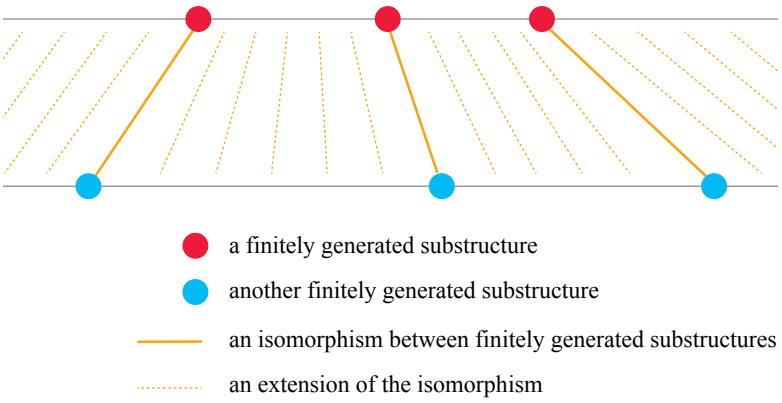
and which is also consistent with the functions (which is the same as preserving and reflecting the graphs of the functions). An *embedded substructure* of a structure is defined to be any structure which embeds into it. A *substructure* is the special case where the embedding is simply inclusion. For a structure, the substructure *generated* by a subset of the universe is defined by restricting the universe to the smallest subset which contains the generators and is closed

[1] This is a basic notion in model theory. For further information, see e.g. (Hodges, 1993, Section 7).

under applying functions from the vocabulary. If there are no functions, then the generated subset is simply the generators. A *finitely generated* substructure is one generated by a finite subset of the universe.

Definition 7.1 (Homogeneous structure)**.** A structure is called *homogeneous* if every isomorphism between finitely generated substructures extends to a full automorphism.

Example 7.2. The equality atoms and $(\mathbb{Q}, <)$ are homogeneous structures over finite vocabularies. The proof for $(\mathbb{Q}, <)$ is in this picture



In Theorem 7.6 below, we will show that if a structure is homogeneous, and satisfies a further condition that is true whenever the vocabulary has no functions, then it is oligomorphic. This explains why the structures mentioned above are oligomorphic.

Example 7.3. Consider the family of finite sets of natural numbers

$$(\mathsf{P}_{\text{fin}}(\mathbb{N}), \cup),$$

equipped with a binary union function. This structure is not homogeneous, because $\emptyset \mapsto \{1\}$ is a finite partial automorphism which does not extend to a full automorphism.

Example 7.4 (Integers are or are not homogeneous, depending on the vocabulary)**.** Unlike for oligomorphism, whether or not a structure is homogeneous depends on the choice of vocabulary, and not just the automorphism group. Consider the structures

$$\underbrace{(\mathbb{Z}, <)}_{\text{integers with an order relation}}, \qquad \underbrace{(\mathbb{Z}, +1)}_{\text{integers with a unary successor function}}.$$

The two structures have the same automorphism group, namely the translations. The first structure is not homogeneous, because the partial function

$$\underline{0} \mapsto \underline{0} \qquad \underline{1} \mapsto \underline{2}.$$

is an isomorphism between finitely generated substructures, which does not extend to an automorphism. The second structure is homogeneous, because finitely generated substructures are half-intervals of the form $\{i, i+1, \ldots\}$. This example shows that a homogeneous structure need not be oligomorphic, since $(\mathbb{Z}, +1)$ is not oligomorphic for the same reasons as $(\mathbb{Z}, <)$, see Example 3.6.

Quantifier elimination. In an oligomorphic structure, equivariant sets of tuples of atoms are necessarily definable in first-order logic, see Lemma 4.11. For homogeneous structures, quantifier-free formulas are enough.

Lemma 7.5. *Let $\mathbb{A}$ be homogeneous, but not necessarily oligomorphic. Two tuples in $\mathbb{A}^n$ satisfy the same quantifier-free formulas with constants from $\bar{a}$ if and only if they are in the same $\bar{a}$-orbit.*

Proof For the right-to-left implication, the truth-value of quantifier-free formulas with constants from $\bar{a}$ is preserved under $\bar{a}$-automorphisms. Conversely, if two tuples of atoms satisfy the same quantifier-free formulas with constants from $\bar{a}$, then one can build an isomorphism between the substructures generated by them, which extends the identity on $\bar{a}$. By definition of homogeneous structures, this extends to a full automorphism, and therefore the tuples are in the same $\bar{a}$-orbit. □

If the vocabulary is infinite, or if there are functions in the vocabulary, then there might be infinitely many quantifier-free formulas which are not equivalent. For example, in the homogeneous structure $(\mathbb{Z}, +1)$, the formulas

$$a = b \qquad a = b + 1 \qquad a = b + 1 + 1 \qquad \cdots$$

are all quantifier-free but pairwise non-equivalent. This gives further illustration of the phenomenon that homogeneous structures are noe necessarily oligomorphic. The following theorem explains which homogeneous structures are oligomorphic, at least under the assumption that the vocabulary is finite.

Theorem 7.6. *Let $\mathbb{A}$ be a homogeneous structure over a finite vocabulary, possibly including functions. Then $\mathbb{A}$ is oligomorphic if and only if*

(*) *$\forall n \in \mathbb{N}\ \exists k \in \mathbb{N}$ all substructures with n generators have size at most k.*

Furthermore, assuming (*), a subset of $\mathbb{A}^n$ is $\bar{a}$-supported if and only if it is definable by a quantifier-free formula with constants from $\bar{a}$.

Proof We first show that (*) implies oligomorphism. By a pumping argument, if an atom is generated from n other atoms using function symbols, then it is generated by a term of height at most k, where k is obtained by applying the assumption (*) to n. Since the vocabulary is finite, there are finitely many terms of height at most k. It follows that, up to logical equivalence, there are only finitely many quantifier-free formulas with n variables, because the terms appearing in them can be assumed to be small. This, together with Lemma 7.5, implies that there are finitely many equivariant orbits of n-tuples, which proves oligomorphism.

We now show that oligomorphism implies (*). Suppose that $\bar{a}$ is a tuple of atoms. Every atom generated by $\bar{a}$, i.e. every atom that can be obtained from $\bar{a}$ by applying functions from the vocabulary of the structure, is a singleton $\bar{a}$-orbit. By oligomorphism there are finitely many $\bar{a}$ orbits, and therefore the substructure generated by $\bar{a}$ is finite. It remains to give a uniform upper bound on the size of this finite substructure. Consider the function

$$f : \mathbb{A}^* \to \mathbb{N}$$

which maps a tuple of atoms to the size of the substructure generated by the tuple (this size is necessarily finite by the above observations). This function is clearly equivariant. It follows from oligomorphism that for every n there are finitely many values of f for arguments from $\mathbb{A}^n$, thus proving (*).

Consider now the "Furthermore" part. The right-to-left implication is immediate. For the left-to-right implication, we use oligomorphism to show that an $\bar{a}$-supported set of n-tuples of atoms must necessarily contain finitely many $\bar{a}$-orbits, and each such orbit can be defined using by a quantifier free formula thanks to Lemma 7.5. □

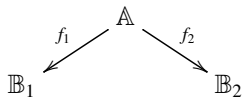
Example 7.7. The structure $(\mathbb{Z}, +1)$ violates condition (*) from Theorem 7.6, because every integer generates an infinite set. In the powerset of the natural numbers from Example 7.3, condition (*) is satisfied, because n elements generate at most 2^n sets (however the structure is not homogeneous, and therefore Theorem 7.6 cannot be applied).

A corollary of Theorem 7.6 is that in structures satisfying its assumptions, every formula of first-order logic is equivalent to a quantifier-free formula. The same is true for richer logics, such as higher-order logics, and for formulas which contain atoms as constants.

7.2 The Fraïssé limit

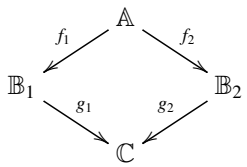
Before defining the Fraïssé limit, consider the following weaker notion of limit: the limit of a class $\mathscr{A}$ of finite structures over a common vocabulary is any structure whose finitely generated embedded substructures are exactly $\mathscr{A}$. In this sense, the equality atoms are the limit of the class of finite structures with equality only, and the ordered rational numbers are the limit of the class of finite total orders. The limit, in the sense defined above, is not necessarily unique, for example any infinite totally ordered set will also be a limit of the class of finite orders. However, it turns out that if we require the limit to be countable and homogeneous, then it is unique up to isomorphism (if it exists). Furthermore, a countable homogeneous limit exists if and only if the class $\mathscr{A}$ is closed under embedded substructures and amalgamation. This result will be stated in Theorem 7.10, which is the main result of this section. Before stating the theorem, we explain amalgamation.

Definition 7.8 (Amalgamation)**.** An *instance of amalgamation* is two embeddings with a common source:



(7.1)

A *solution* of the instance is a structure $\mathbb{C}$ and two embeddings g_1, g_2 such that the following diagram commutes:



(7.2)

We say that a class of structures is *closed under amalgamation* if for every instance of amalgamation which uses structures from the class, there is a solution which also uses a structure from the class.

Definition 7.9 (Fraïssé class)**.** A *Fraïssé class* is a class of finitely generated structures (over a common vocabulary) that is closed under amalgamation and embedded substructures. A Fraïssé class is called *countable* if it has finitely many elements, 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.

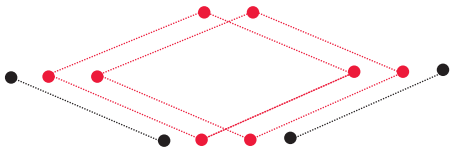
Theorem 7.10 (Fraïssé Theorem)**.** *The map*

countable structure $\mapsto$ *its finitely generated embedded substructures*

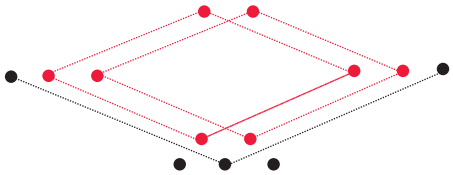
is a bijection between countable homogeneous structures (modulo isomorphism) and countable Fraïssé classes.

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 7.10, we give some examples and non-examples of Fraïssé classes. In all examples below, closure under embedded substructures is immediate, and only amalgamation need be discussed.

Example 7.11. Consider the class of finite structures over an empty vocabulary (in which case first-order 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 and solution of amalgamation:

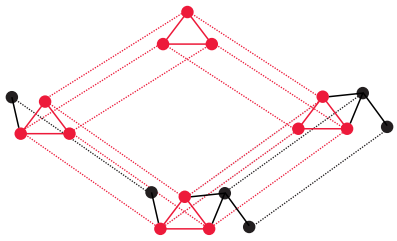


In general, the same instance might have several solutions. Here is an example of a different solution to the instance above:



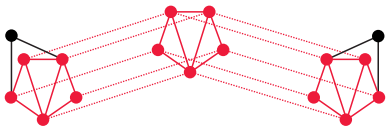
In fact, the above instance has infinitely many solutions (because the solution can be arbitrarily large). Note how the second solution uses the same black element as the target of both black nodes in the second row.

Example 7.12. 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. 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:



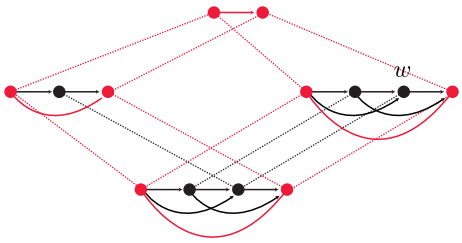
As in Example 7.11, there are also other solutions to the above instance. More generally, for every relational vocabulary, the class of all finite structures over this vocabulary is closed under amalgamation. In particular, by Theorem 7.10, each of these classes has a Fraïssé limit. The limit for undirected graphs will be discussed in more detail in Section 7.3.1.

Example 7.13. Consider the class of finite planar graphs. The class is not closed under amalgamation, here is an instance without a solution:



Example 7.14. 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:

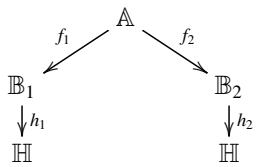
Example 7.15. Consider the class of finite total orders. This class is closed under amalgamation. Here is an example of an instance and solution of amalgamation:



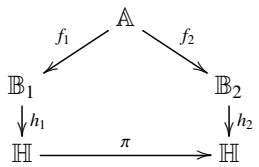
We now begin the proof of the Fraïssé Theorem. We first show that the map from the theorem takes each (not necessarily countable) homogeneous structure to a Fraïssé class. The next steps will be to show that the map is injective and surjective, assuming that the inputs are countable and homogeneous.

Lemma 7.16. *For every homogeneous structure, not necessarily countable, its embedded finitely generated substructures form 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, 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 finitely generated substructures of $\mathbb{H}$. Therefore, by homogeneity there is an automorphism π which extends this partial automorphism. In other words, the following diagram commutes:

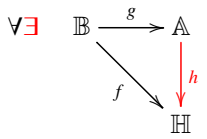


If we restrict the right copy of $\mathbb{H}$ to the image of the maps h_2 and $\pi \circ h_1$, then we get a solution of amalgamation. □

If a homogeneous structure is countable, then it has countably many embedded finitely generated substructures (because the generators can be chosen in countably many ways). Therefore, by the above lemma, the map in the Fraïssé Theorem takes countable homogeneous structures to countable Fraïssé classes. We now establish that the map is injective on countable homogeneous structures, which is the "furthermore" part of the following lemma.

Lemma 7.17. *A countable structure $\mathbb{H}$ is homogeneous if and only if:*

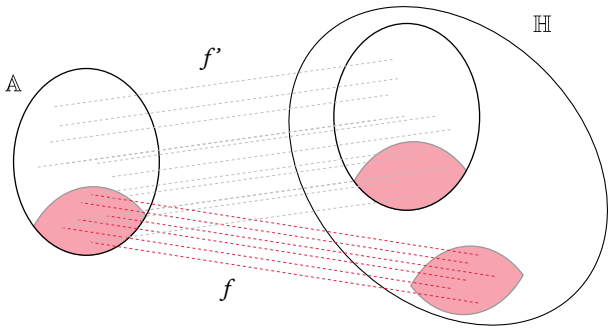
(*) If $\mathbb{A}, \mathbb{B}$ are finitely generated substructures of $\mathbb{H}$ then



Furthermore, countable homogeneous structures with the same finitely generated substructures are isomorphic.

Proof

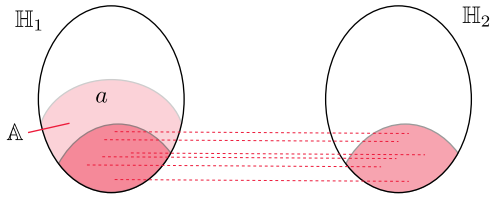
- *Homogeneous structures satisfy (*).* Let g, f be as in (*). We assume without loss of generality that g is an inclusion. Let f' be an embedding of $\mathbb{A}$ into $\mathbb{H}$ which exists by assumption that $\mathbb{A}$ is a substructure. Here is a picture:



  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 7.18. *Let $\mathbb{H}_1, \mathbb{H}_2$ be countable structures with the same finitely generated substructures. If both satisfy (*), then every partial isomorphism between finitely generated substructures of $\mathbb{H}_1$ and $\mathbb{H}_2$ extends to a full isomorphism.*

  Proof Let f be an isomorphism between finitely generated substructures of $\mathbb{H}_1$ and $\mathbb{H}_2$, and let a be an element of $\mathbb{H}_1$. Let $\mathbb{A}$ be the substructure of $\mathbb{H}_1$ generated by a plus the domain of f. Here is a picture:



The structure $\mathbb{A}$ is a finitely generated substructure 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 finitely generated substructures of $\mathbb{H}_1$ and $\mathbb{H}_2$, 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 finite 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 7.18 applied to the empty partial isomorphism between $\mathbb{H}_1, \mathbb{H}_2$, we see that countable homogeneous structures are uniquely determined – up to isomorphism – by their finitely generated substructures.

□

To finish the proof of Fraïssé Theorem, we need to show that the map in the theorem is surjective.

Lemma 7.19. *Every countable Fraïssé class arises as the finitely generated embedded substructures of some countable homogeneous structure.*

Proof Consider some enumeration

$$\mathbb{A}_1, \mathbb{A}_2, \dots$$

of the structures in $\mathscr{A}$ and consider some enumeration

$$f_1, f_2, \dots$$

of all embeddings between structures in $\{\mathbb{A}_1, \mathbb{A}_2, \dots\}$. Such enumerations can

be found by the assumption that $\mathscr{A}$ is countable (for the enumeration of embeddings, we observe that an embedding between $\mathbb{A}_i$ and $\mathbb{A}_j$ is uniquely determined by the images of the finitely many generators in $\mathbb{A}_i$, which can be chosen in countably many ways). Define a sequence

$$\mathbb{H}_1 \subseteq \mathbb{H}_2 \subseteq \cdots$$

of structures in $\{\mathbb{A}_1, \mathbb{A}_2, \ldots\}$ as follows. Choose $\mathbb{H}_1$ arbitrarily, say it is $\mathbb{A}_1$. Suppose that $\mathbb{H}_n$ has already been defined. Consider all instances of of amalgamation

$$\begin{array}{ccc} & \mathbb{A} & \\ \swarrow & & \searrow \\ \mathbb{H}_n & & \mathbb{B} \end{array} \tag{7.3}$$

where both of the embeddings are in $\{f_1, \ldots, f_n\}$. There are finitely many such instances. By doing successive amalgamation steps for each such instance, we can find a structure in $\mathscr{A}$ which can be used as a solution for all of these instances. This structure is isomorphic to some structure in $\{\mathbb{A}_1, \mathbb{A}_2, \ldots\}$, and the latter structure is defined to be $\mathbb{H}_{n+1}$.

Define $\mathbb{H}$ to be the limit (i.e. union) of the sequence $\mathbb{H}_1, \mathbb{H}_2, \ldots$. By construction, $\mathbb{H}$ embeds all structures from $\mathscr{A}$, and no others. By construction, $\mathbb{H}$ satisfies condition (*) from Lemma 7.17, and is therefore homogeneous. □

This completes the proof of Fraïssé Theorem.

Computability. We finish this section by giving a sufficient condition which ensures that the limit of a Fraïssé satisfies the assumptions used by the algorithms on orbit-finite sets that were described in Chapters 4 and 5.

Theorem 7.20. *Let $\mathscr{A}$ be a Fraïssé class over a finite vocabulary such that for every $k \in \{1, 2, \ldots\}$ there are (up to isomorphism) finitely many structures in $\mathscr{A}$ with at most k generators, and a list of these structures can be computed given k (in particular, the Fraïssé class is countable). Then the Fraïssé limit:*

(1) *is oligomorphic; and*
(2) *has a computable Ryll-Nardzewski function; and*
(3) *has a decidable first-order theory with parameters.*

Proof Oligomorphism of the Fraïssé limit follows from Theorem 7.6.

Consider the Ryll-Nardzewski function. By homogeneity, two tuples in the Fraïssé limit are in the same orbit if and only if their generated substructures are the same. It follows that the number of orbits of k-tuples is the same as the

number of isomorphism types of structures in $\mathbb{A}$ with k generators. The latter number can be computed by the assumption on $\mathscr{A}$.

We are left with deciding the first-order theory with parameters.

We begin by explaining how elements of the Fraïssé limit, call it $\mathbb{H}$, can be represented in a finite way. Recall that the Fraïssé limit is constructed, in Lemma 7.19, as limit of a sequence of structures

$$\mathbb{H}_1 \subseteq \mathbb{H}_2 \subseteq \cdots .$$

We now argue that this construction is effective.

Claim 7.21. *Given n, one can compute $\mathbb{H}_n$.*

Proof An inspection of the proof in Lemma 7.19 shows that, in order to compute the structure $\mathbb{H}_n$, one needs the following assumptions on $\mathscr{A}$: (a) the class $\mathscr{A}$ is recursively enumerable; and (b) given an instance of amalgamation, one can compute a solution. To prove the claim, we show that both (a) and (b) follow from our assumption on $\mathscr{A}$. For (a), we first enumerate all structures with 1 generator, then all structures with 2 generators, and so on. For (b), we observe that if an instance of amalgamation uses structures of size at most k, then the solution is generated by at most $2k$ elements, and therefore the solution can be found by exhaustive search. □

Thanks to the above claim, we can use the following finite representation for elements in the Fraïssé limit: each element is represented as a pair (n, a) where $n \in \{1, 2, \ldots\}$ is such that a appears in $\mathbb{H}_n$ for the first time. For elements represented this way, we can evaluate quantifier-free formulas, since a quantifier-free formula can be evaluated in $\mathbb{H}_n$ with n chosen large enough so that it covers all arguments.

It remains to show that the first-order theory is decidable. To do this, we show that for every first-order formula not only an equivalent quantifier-free formula exists (which follows from Theorem 7.6), but also this formula can be computed. It is enough to eliminate one quantifier

$$\exists x \underbrace{\varphi(x_1, \ldots, x_n, x)}_{\text{quantifier free}}.$$

Consider the structures in $\mathscr{A}$, along with distinguished elements corresponding to the free variables, which satisfy the formula φ:

$$\mathscr{A}_\varphi = \{(\mathbb{A}, \overbrace{a_1, \ldots, a_n}^{\bar{a}}, a) : \mathbb{A} \in \mathscr{A} \text{ is generated by } \bar{a}a \text{ and } \mathbb{A} \models \varphi(\bar{a}a)\}.$$

Up to isomorphism, the above set is finite and can be computed thanks to the assumption on $\mathscr{A}$. 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 component a from the valuation and keep only those elements of $\mathbb{A}$ that are generated by $\bar{a}$. 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. □

All Fraïssé classes discussed in this chapter satisfy the assumptions of the above theorem, in particular:

(1) finite sets with equality only (Example 7.11);
(2) finite directed and undirected graphs (Example 7.12);
(3) finite total orders (Example 7.15);
(4) finite partial orders (Exercise 118);
(5) finite trees (Section 7.3.3);
(6) finite vector spaces over the two element field (Section 7.3.2).

In particular, for each of the classes above, the Fraïssé limit can be used as atoms, leading to algorithms for problems such as graph reachability, automaton emptiness, or automaton minimisation.

Exercises

Exercise 118. 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 119. Are series parallel graphs closed under amalgamation?

Exercise 120. 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 (7.2), is exactly the image of $\mathbb{A}$.

Exercise 121. Assume a finite relational vocabulary. Suppose that $\mathscr{A}$ is a class of structures that satisfies the assumptions of Theorem 7.10, and let $\mathbb{A}$ be its Fraïssé limit. Show that if membership in $\mathscr{A}$ is decidable, $\mathbb{A}$ is an effective structure.

Exercise 122. 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 123. 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 124. [3] 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 there are unary predicates $a(x)$ for the labels. 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.

7.3 Examples of homogeneous atoms

We end this chapter with three extended examples of homogeneous structures. We use these examples to illustrate the theory developed in the Chapters 3–6.

7.3.1 The random graph

In this section, we consider the Fraïssé limit of all finite undirected graphs. As shown in Example 7.12, this is a Fraïssé class, and therefore it has a Fraïssé limit by the Fraïssé Theorem. Call this limit the *random graph*. The name is justified by the following observation.

[2] For an introduction to MSO and Rabin's Theorem, see (Thomas, 1990, Theorem 6.8).
[3] This exercise is essentially (Bojańczyk et al., 2013b, Proposition 2).

Theorem 7.22. *Consider a countably infinite undirected graph, where each the presence/absence of an edge is chosen independently with equal probability one half*[4]*. Almost surely (i.e. with probability one) this graph is isomorphic to the random graph.*

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". It is not hard to see that 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 7.17, and therefore it is isomorphic to the random graph. □

Since the class of finite undirected graphs is clearly enumerable, its Fraïssé limit is oligomorphic and has all of the computability properties in the conclusion of Theorem 7.20. It follows that problems such as graph reachability, automaton emptiness, checking graph homomorphism with a given support, etc. are decidable, assuming that the inputs are represented by set builder expressions over the random graph.

Example 7.23. Assume that the atoms are the random graph. 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 stores the last seen vertex. Automata for other properties of vertex sequences are discussed in the exercises.

Path decompositions. 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.

Definition 7.24 (Path decomposition)**.** For $k \in \{1, 2, \ldots\}$, define a *width k path decomposition* to be a sequence $V_1, \ldots, V_n$ of sets of at most k atoms (in the random graph), called *bags*, such that

(1) if atoms appear in the decomposition (possibly in different bags) and are connected by an edge, then they appear together in some bag; and

[4] The conclusion of the theorem would not changed if we used a different distribution, e.g. there would be an edge with probability 0.99.

(2) for every atom, the indexes of bags where it appears is a connected interval in $\{1, \ldots, n\}$;

The *underlying graph* of a path decomposition is defined to be the subgraph of $\mathbb{A}$ that is induced by the union of all bags.

The graphs of pathwidth at most k are defined the underlying graphs of width k path decompositions, or graphs isomorphic to them. The family of sets of at most k atoms is orbit-finite: each orbit is an isomorphism types of graphs of size at most k. Therefore, a width k path decomposition can be seen as a word over an orbit-finite alphabet – namely the bags – and therefore it 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.

Proposition 7.25. *There is a deterministic orbit-finite automaton $\mathcal{A}$ such that*

$$\mathcal{A} \text{ accepts } V_1 \cdots V_n \quad \text{iff} \quad \text{the underlying graph } V_1 \cup \cdots \cup V_n \text{ is connected}$$

holds for every width k path decomposition[5].

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 which identifies vertices from the last bag if they are in the same connected component of the underlying graph $V_1 \cup \cdots \cup V_n$. We make no claims about the state of the automaton if the input is not a path decomposition.

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 transition function is defined as follows. Suppose that the current state is $(V_n, \sim_n)$ and the input letter is V_{n+1}. Let $\sim$ be the smallest equivalence relation on $V_n \cup V_{n+1}$ which contains both $\sim_n$ and the edges on V_{n+1} (as defined by the graph structure of the atoms). If there is an equivalence class of $\sim$ which is disjoint with V_{n+1}, then this equivalence class will remain forever disconnected, and therefore the automaton rejects immediately. Otherwise, the new state is set to V_{n+1} with the equivalence relation $\sim_{n+1}$ being $\sim$ restricted to V_{n+1}. □

Similar constructions as in the above proposition can be done for any property of graphs of bounded pathwidth that is recognisable in the sense of Courcelle, this covers all graph properties which can be defined in monadic second-

[5] The automaton does not check if the input is a path decomposition, in fact this cannot be done, see Exercise 128.

order logic[6]. Using tree automata instead of word automata, one can also cover tree decompositions.

Exercises

Exercise 125. 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 126. 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 127. Assume that the atoms are the random graph. Show that there is no finitely supported total order on the random graph.

Exercise 128. Show that there is no orbit-finite automaton, even nondeterministic, which recognises the language of width k path decompositions.

Exercise 129. 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 130. Assume that the atoms are the random graph. Show that solving equations, as discussed in Section 5.5, is undecidable.

7.3.2 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 10, where we will

[6] For more on recognisability, pathwidth, and monadic second-order logic, see (Courcelle and Engelfriet, 2012, Chapter 5.3)

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. We model a vector space as a structure where the universe is the vector space, and there is a binary function + for addition of vectors. Every finite vector space is isomorphic to

$$(\{0,1\}^n, +) \qquad \text{for some } n \in \{1,2,\ldots\}$$

where addition is modulo two. There is no need to have a constant for the zero vector, because adding any vector to itself gives the zero vector.

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 n_1 and n_2, one needs a vector space of dimension $\max(n_1, n_2)$. Therefore, there is a Fraïssé limit of the finite vector spaces; and thanks to Theorem 7.20 this limit is oligomorphic, has a computable Ryll-Nardzewski function, and a decidable first-order theory with parameters.

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 is 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 coordinatewise addition modulo two:

$$0101 + 11001 = 10011.$$

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. (A similar result is also true for the random graph discussed in Section 7.3.1, but the proof for bit vectors is more interesting, and will also be used in Section 10.2.) 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 7.26. *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 7.27 (Least Support Theorem)**.** *Assume the bit vector atoms. Let x be an atom or a set with atoms. 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 $\bar{b}$.*

The proof idea is similar to the proof of the Least Support Theorem for equality atoms in Chapter 6. The key is a representation result, Lemma 7.28 below, which says that orbit-finite sets can be represented by tuples of atoms, along a function which preserves and reflects supports. To state this representation result, we need to recall two definitions from Chapter 6. Recall the notion of straight sets from Definition 6.7, these are disjoint unions of finitely supported sets of atom tuples. A function f between two sets with atoms is said to *preserve and reflect supports* if for every input x satisfies

$$\bar{a} \text{ supports } x \quad \text{iff} \quad \bar{a} \text{ supports } f(x) \qquad \text{for every atom tuple } \bar{a}.$$

Preserving supports, i.e. the left-to-right implication above, is true for all equivariant functions.

Lemma 7.28. *Assume the bit vector atoms. For every equivariant orbit-finite set X there is an equivariant straight set Y and a surjective equivariant function*

$$f : Y \to X$$

which preserves (this follows from equivariance) and reflects supports.

Before proving the lemma, we use it to prove the Least Support Theorem. Let x be an atom or a set with atoms, and let $X \ni x$ be its equivariant orbit. Apply Lemma 7.28 to X, yielding an equivariant surjective function

$$f : Y \to X$$

which preserves and reflects supports. Because f preserves and reflects supports, it is enough to show that elements of Y have least supports, in the sense

of Theorem 7.27. This follows from Lemma 7.28. The above argument uses no assumptions about the bit vector atoms, except that they satisfy Lemma 7.28, and therefore any atoms which satisfy the lemma will also have the Least Support Theorem. It remains to prove the lemma.

Proof of Lemma 7.28. Without loss of generality we assume that X has one orbit. The proof follows the same lines as the proof of Lemma 6.2, except that vector independence plays the role of equality. We first show that there is a surjective equivariant function $f : Y \to X$ from a straight set which satisfies a weaker property, see (7.4) below, and then we show that this condition implies that f preserves and reflects supports.

(1) Let us write $\mathbb{A}^{\langle n \rangle}$ for the set of n-tuples of nonzero atoms which are linearly independent. This is a one orbit set. By eliminating vectors that are linearly dependent, every equivariant one orbit set admits an equivariant bijection with $\mathbb{A}^{\langle n \rangle}$ for some n. By Theorem 3.23 there is a surjective equivariant function

$$f : \mathbb{A}^{\langle n \rangle} \to X \qquad \text{for some } n.$$

We will show that, either f satisfies the following condition[7]

$$f(\bar{a}) = f(\bar{b}) \qquad \text{implies} \qquad \bar{a} \text{ and } \bar{b} \text{ span the same spaces,} \tag{7.4}$$

or the dimension n can be made smaller. By iterating this argument at most n times, we arrive at a function f that satisfies (7.4).

Suppose that f violates condition (7.4), as witnessed by tuples

$$\overbrace{(a_1, \ldots, a_n)}^{\bar{a}} \qquad \overbrace{(b_1, \ldots, b_n)}^{\bar{b}}$$

which have the same image under f but span different spaces. Without loss of generality, we assume that the first i coordinates $a_1, \ldots, a_i$ of the tuple $\bar{a}$ are not spanned by $\bar{b}$, and the remaining coordinates are spanned. In other words, the tuple

$$(a_1, \ldots, a_i, b_1, \ldots, b_n)$$

is linearly independent. Since the vector space $\mathbb{A}$ has infinite dimension, one can choose $a'_1, \ldots, a'_i \in \mathbb{A}$ such that adding them to the above tuple retains independence. It follows that

$$(a_1, \ldots, a_i, b_1, \ldots, b_n) \qquad (a'_1, \ldots, a'_i, b_1, \ldots, b_n)$$

[7] Condition (7.4), as well as the related condition from Lemma 6.2, 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).

are in the same equivariant orbit, namely the equivariant orbit of linearly independent tuples. This means that

$$(a_1, \ldots, a_i) \quad (a'_1, \ldots, a'_i)$$

are in the same $\bar{b}$-orbit. Since $a_{i+1} \ldots, a_n$ are supported by $\bar{b}$, it follows that

$$(a_1, \ldots, a_n) \quad (a'_1, \ldots, a'_i, a_{i+1}, \ldots, a_n)$$

are in the same $\bar{b}$-orbit. Having the same value under f as $\bar{b}$ is a $\bar{b}$-supported property, and since the first tuple above has the same value under f as $\bar{b}$, the same must be true for the second tuple, i.e.

$$f(a_1, \ldots, a_n) = f(a'_1, \ldots, a'_i, a_{i+1}, \ldots, a_n). \tag{7.5}$$

In other words, there exists a tuple

$$(a_1, \ldots, a_n, a'_1, \ldots, a'_i) \in \mathbb{A}^{\langle n+i \rangle},$$

which satisfies equality (7.5). By equivariance, all tuples in $\mathbb{A}^{\langle n+i \rangle}$ satisfy the equality. In other words, the first i coordinates in a tuple from $\mathbb{A}^{\langle n \rangle}$ can be replaced by a 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 n.

(2) By the previous item, there is some equivariant $f : \mathbb{A}^{\langle n \rangle} \to X$ which satisfies condition (7.4). By virtue of equivariance, f preserves supports. We show that condition (7.4) implies that f reflects supports, i.e. any tuple that supports the output also supports the input. To prove this, suppose that $\bar{b}$ supports $f(\bar{a})$, for some $\bar{a} \in \mathbb{A}^{\langle n \rangle}$. This means that every $\bar{b}$-automorphism π satisfies

$$f(\bar{a}) = \pi(f(\bar{a})) = f(\pi(\bar{a})).$$

Because f was obtained from Lemma 7.28, the second equality above implies that $\bar{a}$ and $\pi(\bar{a})$ span the same spaces. Therefore, applying any $\bar{b}$-automorphism to $\bar{a}$ results in a tuple $\pi(\bar{a})$ which spans the same space. This implies that $\bar{b}$ must span $\bar{a}$ (and therefore support it).

□

Register automata. Recall Theorem 6.6, which said that, under the equality atoms, deterministic register automata were expressively complete for abstract deterministic orbit-finite automata, assuming alphabets where the register automata could be used. We conclude Section 10.2 with an extension of this result to the bit vector atoms.

We begin by describing the bit vector versions of straight automata and register automata, which were used in Theorem 6.6. Call a deterministic orbit-finite automaton *straight* if its state space is straight. This implies that all remaining components (the accepting states and the transition function) of the automaton are straight, since the remaining components are obtained from the state space using products and finitely supported subsets, and such operations preserve straightness. Define a *register automaton* to be an automaton where the state space is of the form

$$\mathsf{Loc} \times (\mathbb{A} + \{\bot\})^n \qquad \text{for some } n \in \{0, 1, \ldots\} \text{ and finite equivariant set } \mathsf{Loc}.$$

Actually, the undefined value $\bot$ is not needed, since in the bit vector atoms, the role of $\bot$ can be played by the zero vector. In the spirit of Lemma 1.3, one can give a more syntactic description of equivariant register automata. In the syntactic description, the transition function tests the register contents and the input letter for linear dependencies, and conditional on the linear dependencies that are satisfied, it updates the registers using linear combinations of the previous register contents and the input letter.

Theorem 7.29. *Assume the bit vector atoms. For languages over straight input alphabets, the following models have the same expressive power:*

(1) *deterministic orbit-finite automata;*
(2) *straight deterministic orbit-finite automata;*
(3) *register automata.*

The inclusion (3) $\subseteq$ (1) is immediate. The inclusions

$$(2) \subseteq (3) \qquad (1) \subseteq (2)$$

are proved the same way as in Section 6.2, where the only property of the equality atoms that was used was the existence of functions from straight sets that preserved and reflected supports, as in Lemma 7.28.

Exercises

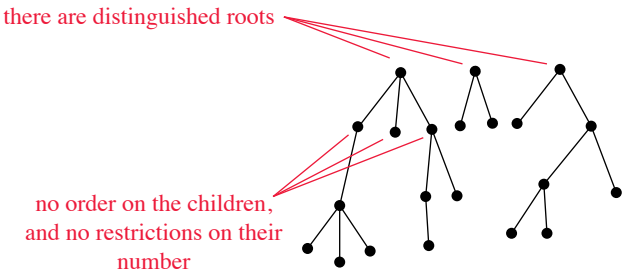
Exercise 131. Let $\mathbb{B}$ be the structure where the universe is the same as in the bit vector atoms, but where the binary addition function is replaced by the ternary predicate "the atoms a, b, c are linearly independent". Show that $\mathbb{B}$ has the same automorphisms as the bit vector atoms.

Exercise 132. Show that the structure $\mathbb{B}$ from Exercise 131 is not homogeneous. Show that the structure becomes homogeneous if we add linear independence predicates for every arity, i.e. for every n there is an n-ary predicate which says that atoms $a_1, \ldots, a_n$ are linearly independent.

Exercise 133. Consider vector spaces over the field of rational numbers (as opposed to the two-element field as considered in this section). We model a vector space as a structure with one binary function for addition. Is the class of finite-dimensional vector spaces a Fraïssé class? If yes, is its Fraïssé limit oligomorphic?

7.3.3 Trees and forests

In this section we study the Fraïssé limit of trees and forests [8]. The trees and forests we study are rooted, unlabelled, and unordered, as explained in the following picture:

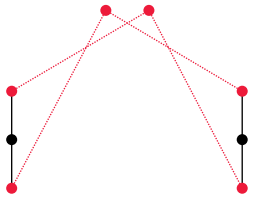


A tree is the special case of a forest when there is exactly one root.

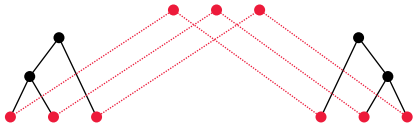
The purpose of this section is to show that care is needed when choosing predicates and functions to model a combinatorial object, like a tree or forest, if we want to have a Fraïssé limit. The following list shows three ways of modelling trees as logical structures; only the third way will admit a Fraïssé 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:

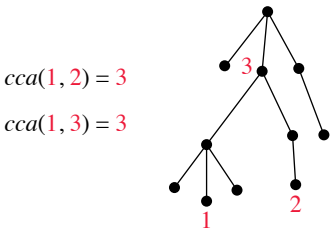
[8] This section is based on Bojańczyk et al. (2013b).



(2) There is 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) In this item, we assume that we want to model trees, i.e. there is exactly one root. To model the tree structure, we have a function $cca(x, y)$ which inputs two nodes and returns their closest common ancestor, as explained in the following picture:



The class of trees modelled this way is closed under amalgamation, as illustrated in Figure 7.1. Therefore, it has a Fraïssé limit, which we call the *universal tree*.

Exercises

Exercise 134. Assume that universal tree atoms. Find a finitely supported equivalence relation on the atoms which has infinitely many infinite equivalence classes.

Exercise 135. Assume that universal tree 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 136. Show that the universal tree has decidable MSO theory.

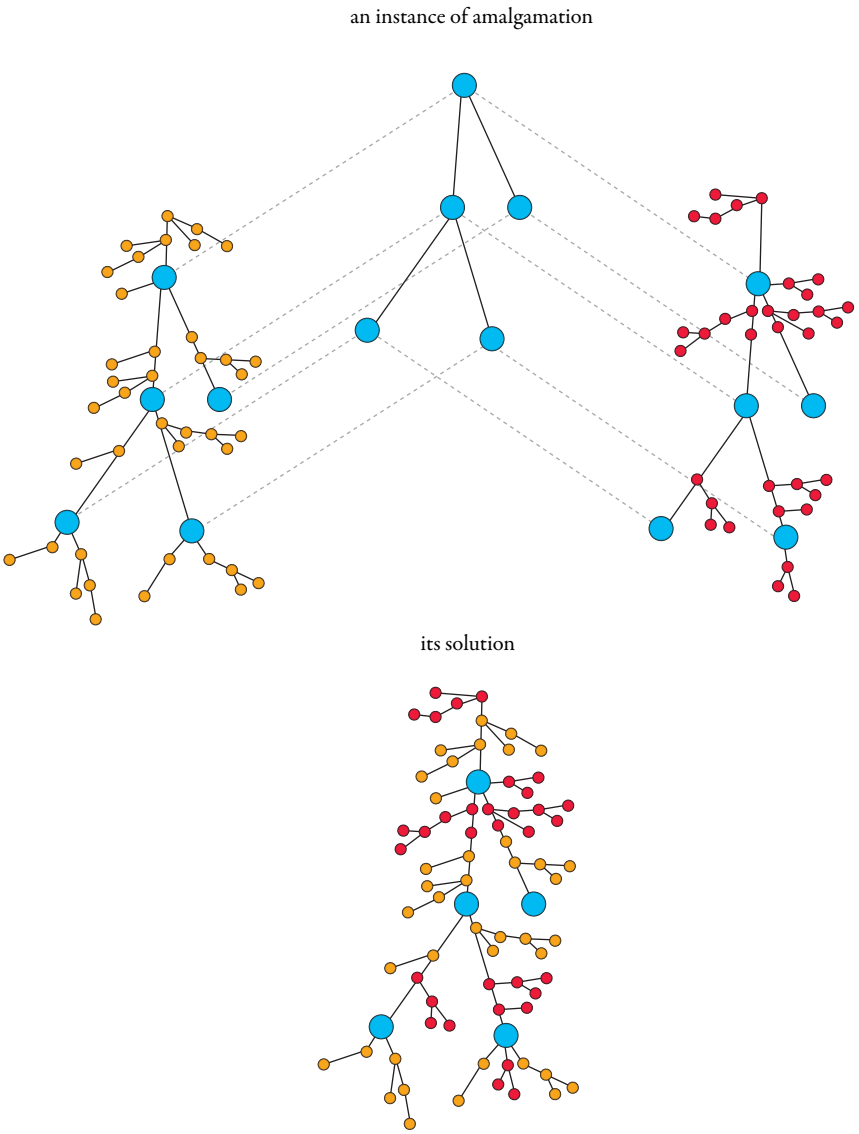


Figure 7.1 Amalgamation for trees with a closest common ancestor function.

PART THREE

COMPUTATION WITH ATOMS

In Part II, we defined orbit-finite sets, and showed how orbit-finite sets behave similarly to finite ones. We also discussed algorithms that operated on orbit-finite sets. This part discusses computation in more depth. We show that there are robust notions of computability for orbit-finite sets, and even a notion similar to "polynomial time".

8

Computable functions on sets with atoms

What is a computable function on (hereditarily) orbit-finite sets, or more generally sets represented by set builder expressions? One idea is to use algorithms that manipulate set builder expressions representing the sets. A drawback of this idea is that it is closely based on a syntactic representation, and one could imagine that different choices of representation would lead to different notions of computability.

In this chapter, we present a programming language, called *while programs with atoms*, which works directly with sets, and not with set builder expressions that represent them. The programs store sets in variables, and can loop all elements of such a set. The programming language has two goals:

(1) to identify a robust notion of computability; and
(2) to write algorithms without going into representation details.

To illustrate the programming language, consider the following program, based on the case study in Section 5.1, which computes the vertices of a graph (V, E) that can be reached from a designated set $S \subseteq V$.

```
Reach := S
New := ∅
while New ⊊ Reach do
  Reach := New
  for v in Reach do
    for w in V do
       if (v,w) ∈ E then
         New := New ∪ {w}
```

The program is simply a naive implementation of breadth-first search. The catch is that the variables store sets that are not necessarily finite. The key programming construct is the `for` loop, which ranges over elements of a possibly

infinite set. The general idea is that the `for` loop is executed in parallel. It will turn out that, if the initial program state uses only sets represented by set builder expressions, then this will remain the case during program execution.

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 is true for oligomorphic atoms but false in general.

In Section 8.1, we define the programming language and explain its semantics. In Section 8.2 we show that, under reasonable assumptions on the atoms, 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 sets represented by set builder expressions that can be implemented using, say, Turing machines. This shows that sets represented by set builder expressions have a robust notion of computability.

8.1 While programs with atoms

In this section, we introduce the programming language[1], explain its semantics, and give example programs. We make no assumptions on the atoms. The atoms need not be oligomorphic, and their first-order theory might be undecidable.

Syntax. Fix some countably infinite set of variable names. The programming language is untyped: every variable stores a set (represented by a set builder expression) or an atom[2]. As discussed in Chapter 3, one can encode other data structures using sets. For example, natural numbers can be encoded using Von Neumann ordinals, where 0 is the empty set and $n + 1$ is the set $\{0, \ldots, n\}$. Using the Von Neumann representation, 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.

```
z := x
i := ∅
while i ⊊ y do
  z := {z} ∪ z
  i := {i} ∪ i
```

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 8.1 (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:

  $\underbrace{\texttt{x := } \mathbb{A}}_{\substack{\text{load the set}\\ \text{of atoms into x}}}$ $\underbrace{\texttt{x := } R}_{\substack{\text{load a relation } R\\ \text{from the atom}\\ \text{structure into x}}}$ $\underbrace{\texttt{x := y} \cup \texttt{z}}_{\text{set union}}$ $\underbrace{\texttt{x := \{y\}.}}_{\text{singleton}}$

  The relation R is taken from the vocabulary of the atoms, with functions being treated as a special case of relations[3]. In set union, the program aborts with failure 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`.
- *Control flow.* Suppose that `x` and `y` are variables, `I` is an already defined program, and δ is one of $\in, \subseteq, \subsetneq$ or $=$. Then

  `if x` δ `y then I` `while x` δ `y do I` `for x in y do I`

  are all 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`, see below for a more detailed description.

Semantics. We show below 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 set. (We will be interested in the case when the sets are represented by set builder expressions, but the semantics make sense for other sets as well.) The semantics of a while program `I` is

[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.

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 is partial, because its output is defined if and only if when 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 do I,
```

the other constructions being handled in the standard way. Suppose that we execute the loop above in a program state γ. If the variable `y` stores an atom, then the loop does nothing and the input and output program states are the same. Assume otherwise, that `y` stores a 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 this set, 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 on $\gamma[\mathtt{x} := x]$, or it might terminate in a finite number of steps $n_x \in \{1, 2, \ldots\}$. If `I` does not terminate on some $\gamma[\mathtt{x} := x]$, 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, 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 x in `x`;
(2) the union $\bigcup_{\gamma \in \Gamma} \gamma(\mathtt{x})$ otherwise.

Note that in the second case, where union is used, some $\gamma \in \Gamma$ 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 ignored). 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 8.2, 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 8.1 is devoted to examples of while programs with atoms. Like in Python, we use indentation to distinguish blocks in programs.

Example 8.2 (Pairing and unpairing)**.** The code

```
p := {x,{x,y}}
```

loads the ordered pair `(x,y)`, according to Kuratowski pairing, into the variable `p`. Formally speaking, the above is syntactic sugar for the following sequence of operations:

```
q := {x}
p := {y}
p := p ∪ q
p := {p}
p := p ∪ q
```

In the programs below, we write `p := (x,y)` instead of the above code.

We can also implement unpairing. 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.

```
ret := ∅
for a in p do
    for x in a do
      for y in p do
        if p = (x,y) then
          ret:={x};
```

In Example 8.5 we show how to get rid of the singleton. The test `p = (x,y)` 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 the semantics of set builder expressions can be implemented by while programs with atoms.

Example 8.3. The atoms are $(\mathbb{Q}, <)$. Consider the set of bounded open intervals:

$$X = \{\{z : \text{ for } z \in \mathbb{A} \text{ such that } x < z < y\} : \text{ for } x, y \in \mathbb{A} \text{ such that } x < y\}$$

Here is a program which loads this set into variable `X`.

```
X := ∅
for x in 𝔸 do
  for y in 𝔸 do
    if x < y then
      Z := ∅
      for z in 𝔸 do
        if x < z < y then
          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 without atom parameters can be loaded into a variable.

Example 8.4. The following program inputs a nonzero natural number in variable `n` and outputs the set $\mathbb{A}^n$ in variable `Y`. The number `n` and operations on it are implemented using Von Neumann encoding.

```
Y := 𝔸
while n > 1 do
  n := n - 1
  X := Y
  for a in 𝔸 do
    for x in X do
      Y := Y ∪ {(a,x)}
```

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 8.5 (Desingleton)**.** The following program implements the mapping $\{x\} \mapsto x$. More precisely, if the variable `x` stores a singleton $\{x\}$, then after

running the program the variable `result` will store x, otherwise `result` will store the empty set.

```
result := ∅
for y in x do
  y:=y
if x = {y} then
 result := y
```

Example 8.6. Assume that the atoms are Presburger arithmetic, i.e. $(\mathbb{Z}, +)$. One can write a multiplication function, by using a while loop to implement multiplication in terms of iterated addition. One can also write a primality test. The following program looks like it computes the set of all primes in finite time:

```
P := ∅
for x in ℕ do
  if prime(x) then
    P := P ∪ {x}
```

Actually, the program does not terminate, because the body of the for loop has unbounded running time (because primality tests take more time for bigger numbers), and the semantics says that such loops do not terminate.

Example 8.7 (Reachability)**.** Consider the program for graph reachability from the beginning of this chapter. As argued in Section 5.1, if the atoms are oligomorphic, then the while loop does finitely many iterations for every input, and therefore the program always terminates.

The program can also be run for atoms that are not oligomorphic. For some inputs, the program will even terminate (and thus give the correct result). For example, if the atoms are Presburger arithmetic $(\mathbb{Z}, +)$, and the input is:

- the vertices are natural numbers,
- the edges are all pairs of natural numbers which disagree modulo 3,
- there is one source, namely 0,

then the program will terminate in two iterations of the while loop and return the set of all natural numbers. However, for other inputs, e.g., when the edges are the successor relation, then the program does not terminate.

Example 8.8 (Automaton emptiness)**.** Using pairs and projections, we can extend the language with a pattern-matching construction

```
for (x,y) in X do 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 do
  E := E ∪ {(p,q)}
```

After computing this relation, we can use the program for graph reachability from Example 8.7 to check if an automaton is nonempty.

Example 8.9 (Automaton minimisation)**.** Suppose that we are given a deterministic automaton:

```
(A,Q,q0,F,delta).
```

Non-reachable states can be discarded using graph reachability, and therefore we can assume that all states are reachable. We describe below a standard implementation of Moore's minimisation algorithm. The only point of writing it down is so that the reader can follow the code and see how it works with atoms.

In a first step, we compute in a variable equiv the equivalence relation, viewed as a set of pairs, which identifies states that recognise the same languages. To do this, we compute the complement of the equivalence relation[4]. We initialise the non-equivalence relation to states that are distinguished by the empty word:

```
nonequiv := (F × (Q-F)) ∪ ((Q-F) × F)
```

(The code above uses ×, which is implemented using for.) Then iterate the following code using a while loop, until the set R does not grow any more:

```
for (p,a,q) ∈ delta do
 for (p',a',q') ∈ delta do
  if a=a' and (q,q') ∈ R then
   nonequiv := nonequiv ∪ {(p,p')}
```

Once the set nonequiv has stabilised, it contains exactly the pairs of states which recognise different languages. Therefore, we get the "same language" relation by doing complementation:

```
equiv := (Q × Q) - nonequiv
```

[4] The semantics of for loops uses unions, which are better suited for inflationary fixpoints, such as the one used in the definition of non-equivalence. This is why we do not compute equivalence directly, but go through non-equivalence.

For the states of the minimal automaton, we use the equivalence classes of the relation `equiv`, which are produced by the following code.

```
for q in Q do
  for (p,r) in equiv do
    if q=p then
      class := class ∪ {r}
  classes := classes ∪ {class}
```

The remaining part of the minimisation program goes as expected: the states are the equivalence classes, and the remaining components of the automaton are defined as usual.

We finish the section with an a example of a program where a termination proof requires a bit of effort.

Example 8.10. 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 binary function `Product` for the products, and checks if it is aperiodic. The program simply executes the definition of aperiodicity; the semigroup is aperiodic if and only the variable `counterexamples` is empty at the end of the execution.

```
counterexamples := ∅
  for s in S  do
    X := ∅
    power := s
    while power ∉ X do
       X := X ∪ {power}
       power := Product(power,s)
    if power ≠ Product(power,s) then
      counterexamples := counterexamples ∪ {s}
```

In the program above, `Product(power,s)` is actually syntactic sugar for a subroutine which examines the graph of the product operation, and finds the unique element `x` which satisfies `(power,s,x) ∈ Product`.

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. Every power of `s` is supported by whatever supports `s` and the product operation in the semigroup. Since the carrier of the semigroup is represented by a set builder expression, it is also orbit-finite, by Theorem 4.10. In an orbit-finite set, there are finitely many elements with a given support, as shown in Exercise 55. 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.

8.2 Computational completeness of while programs

The point of while programs is that they work directly on sets, and not on their representations as set builder expressions. In this section, we show that Turing machines working on set builder expressions would have the same computational power This shows that while programs are, in a sense, computationally complete, and any other computation model which can be evaluated using set builder expressions would necessarily be subsumed by while programs.

Turing machines with first-order access to the atoms. We begin by formalising computation on set builder expressions. As the underlying model we use Turing machines. The question is: how does a Turing machine accesses the atoms which appear as parameters in a set builder expression? Our design choice is to equip a Turing machine with an oracle that answers first-order queries about atoms that appear as parameters in a set builder expression, see below for more details. An alternative model, which uses bit strings that represent atoms, is discussed at the end of this chapter.

We now describe in more detail the variant of Turing machines that we use. The machine has four tapes: a read-only input tape, a work tape, an oracle tape, and a write-only output tape. The set builder expression is given on the input tape, but the atom parameters are represented as question marks. For example, the input expression is

$$\{\{\{x, \underline{5}\} : \text{ for } x \in \mathbb{A} \text{ such that } x \neq \underline{2} \wedge x \neq y\} : \text{ for } y \in \mathbb{A} \text{ such that } y \neq \underline{5}\}$$

then the input tape will contain

$$\{\{\{x, ?\} : \text{ for } x \in \mathbb{A} \text{ such that } x \neq ? \wedge x \neq y\} : \text{ for } y \in \mathbb{A} \text{ such that } y \neq ?\}.$$

Thanks to the question marks, the input tape needs only a finite alphabet (assuming that the variables are, say, bit strings). Assume that the parameters in the input set builder expression are $a_1, \ldots, a_n$, listed in order of appearance.

In the example set builder expression above, the parameters are $\underline{5}, \underline{2}, \underline{5}$. At any point during its computation, the Turing machine can write on the oracle tape a first-order formula over the vocabulary of the atoms, which has n free variables. An oracle then answers, in unit time, whether or not this formula is true in the atom structure. The oracle can be consulted more than one time, and their oracle's answers can influence further computation. The output of the machine is represented on the output tape, with parameters being defined using first-order logic in the following way. Each parameter a in the output set builder expression is represented by a formula $\varphi(x, x_1, \ldots, x_n)$ which has the semantic property that a is the unique atom which satisfies $\varphi(a, a_1, \ldots, a_n)$. If the semantic property is not satisfied, then the computation fails. A (deterministic) Turing machine in the above model defines a partial function from set builder expressions to set builder expressions. The function is undefined if the Turing machine does not terminate, or it poses an ill-formatted query to the oracle, or if the output parameters are not represented by formulas which define unique atoms. Any partial function defined this way is called *recognised by a Turing machine with first-order access to the atoms*.

Computational completeness. The following theorem shows that while programs (working directly on sets) define exactly the same functions as Turing machines with first-order access to the atoms (working on representations via set builder expressions) . The theorem uses no assumptions on the atoms (such as oligomorphism or decidable first-order theory), except that they have a finite vocabulary. The assumption on finite vocabulary could also be lifted, assuming some enumeration of the vocabulary.

Theorem 8.11 (Computational completeness of while programs[5])**.** *Assume that the atoms have a finite vocabulary. The following conditions are equivalent for every function*

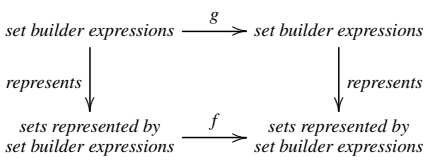
$$\begin{array}{c}\textit{sets represented by}\\ \textit{set builder expressions}\end{array} \xrightarrow{\ f\ } \begin{array}{c}\textit{sets represented by}\\ \textit{set builder expressions}\end{array}$$

(1) *There is while program with atoms, with a designated variable, such that if the program is executed in a program state where the designated variable is set x, then the program terminates with $f(x)$ in the designated variable.*

(2) *There is some g computed by a Turing machine with first-order access to the*

[5] 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).

atoms which makes the following diagram commute:



The above theorem is formulated for total functions, corresponding to programs that always terminate. The extension to partial functions, corresponding to programs that are not required to always terminate, is also true and can be proved in the same way.

From Turing machines to while programs. We begin by proving the implication (2)⇒(1) in Theorem 8.11. Consider a function g which inputs and outputs set builder expressions, and which is computed by a Turing machine with first-order access to the atoms. Our goal is to give a while program with atoms, which inputs a set represented by a set builder expression α and outputs the set represented by $g(\alpha)$. By assumption (2), the output does not depend on the choice of α, i.e. if set builder expressions α and β represent the same set, then the same will be true for $g(\alpha)$ and $g(\beta)$.

The difficulty is that the while program has access to the set represented by α, but not to the expression itself. The general idea is that the while program will compute the expression, or some expression equivalent to it.

The argument below uses two views on atom parameters used by set builder expression. The first view is to have no free variables, and inline the atom parameters directly into the expressions, as in the following example:

$$\alpha = \{\{\underline{2}, x\} : \text{ for } x, y \in \mathbb{A} \text{ such that } x = \underline{1} \vee y \neq \underline{2}\}.$$

The second view, which will be useful below, is to use free variables for the parameters. For example, to recover the set represented by α above, we could use the expression

$$\beta(a, b) = \{\{a, y\} : \text{ for } x, y \in \mathbb{A} \text{ such that } x = b \vee y \neq a\}$$

and then instantiate the free variables (a, b) to the atoms $(\underline{2}, \underline{1})$. The two views are clearly equivalent. The following lemma uses the second view.

Lemma 8.12. *Let M be a Turing machine with first-order access to the atoms. There is while program which does the following:*

- **Input.** *A number $i \in \{0, 1, \ldots\}$ and set builder expressions α, β without parameters, but with free variables.*

- **Output.** *A first-order formula φ with the same free variables as α, β such that a tuple $\bar{a}$ satisfies $\varphi(\bar{a})$ if and only if M produces output $\beta(\bar{a})$ on input $\alpha(\bar{a})$ using at most i computation steps.*

Proof By examining the decision tree of a Turing machine with first-order access to the atoms, and putting all of the oracle calls into the formula φ. □

We are now ready to complete the proof of the implication (2)⇒(1). Given a set x, the simulating while program does the following. It loops through all candidates for $i \in \{0, 1, \ldots\}$ and α, β as in Lemma 8.12. Note that i and α, β have no atoms, and therefore they can be represented using data structures without atoms, such as binary strings, and the search space of binary strings can be enumerated. For each choice of i, α, β the while program loops through all valuations of the free variables $\bar{a}$, and checks if:

(1) the input x is represented by $\alpha(\bar{a})$; and
(2) the Turing machine computing g takes $\alpha(\bar{a})$ to $\beta(\bar{a})$ in at most i steps.

The second step is checked using Lemma 8.12. For the first item, the while program simply implements the semantics of set builder expressions, as illustrated in Example 8.3. The first-order formulas in the guards of a set builder expression are evaluated as illustrated in Example 8.4.

From while programs to Turing machines. We now prove the top-down implication (1)⇒(2) in Theorem 8.11. We show that given:

- a while program `I`;
- a set builder expression representing a program state γ;

a Turing machine with first-order access to the atoms can compute a set builder expression representing the program state γ`I` after running `I` on γ.

Since the semantics of while programs involve executing infinitely many threads in parallel, it will be easier to work with sets of program states instead of individual program states. If a program `I` has n variables, then a program state can be viewed as an n-tuple of sets or atoms, and therefore it makes sense to talk about program states, or sets of program states, that are represented by set builder expressions. We adopt this view in the proof below; it allows us to talk about set builder expressions that represent program states or sets of program states.

By applying the following lemma to a singleton set of program states, we get the implication (1)⇒(2) in Theorem 8.11.

Lemma 8.13. *There is a Turing machine with first-order access to the atoms which inputs:*

- *a while program* $\mathtt{I}$ *with atoms;*
- *a set builder expression representing a 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*

$$\{(\gamma, \gamma\mathtt{I}) : \gamma \in \Gamma\};$$

- *otherwise, the Turing machine does not terminate.*

Proof Structural induction on the text of the program $\mathtt{I}$. The assignments are immediate, so we only do the proof for the other constructions.

In the proof, we use the Third Symbol Pushing Lemma, which we now recall. Consider a hereditarily definable set X. We write X_* for the set which contains X, its elements, their elements, etc. For example, if X is a set of program states, then X_* will contain, among other elements, all values of all variables used by those program states. Suppose that an n-ary relation

$$R \subseteq X_* \times \cdots \times X_*.$$

can be defined by a first-order formula φ interpreted in the structure structure $(X_*, \in)$. For example, if X is a set of program states, then R could be the set of pairs (x, y) such that some program state from X has x in one variable and has y in some other variable. The Third Symbol Pushing Lemma says that R is also represented by a set builder expression, which can be computed (in polynomial time) based on φ and a set builder expression representing X. In the proof below, any construction using this lemma will be called "symbol pushing". Symbol pushing will be enough to formalise all operations used in the semantics of the programming language, with one exception, which we call Currying.

Having recalled the Third Symbol Pushing Lemma, we proceed to prove the lemma.

- *Sequential composition.* We want to compute the set

$$\{(\gamma, \gamma(\mathtt{I}_1; \mathtt{I}_2)) : \gamma \in \Gamma\}. \tag{8.1}$$

By induction, compute an expression representing

$$R_1 = \{(\gamma, \gamma(\mathtt{I}_1)) : \gamma \in \Gamma\}.$$

The projection of a relation R_1 onto its second coordinate can be defined in

the language of set theory, and therefore symbol pushing yields an expression representing the image of R_1:

$$\Delta = \{\delta : (\gamma, \delta) \in R_1 \text{ for some } \gamma\}.$$

By induction, compute an expression representing

$$R_2 = \{(\delta, \delta(\mathtt{I}_2)) : \delta \in \Delta\}.$$

Since (relational) composition can be defined using the language of set theory, symbol pushing can be used to compute the composition $R_1 \circ R_2$, which is the desired set from (8.1).

- *If.* We want to compute a set builder expression representing

$$\{(\gamma, \gamma(\mathtt{if\ x}\ \delta\ \mathtt{y\ then\ I})) : \gamma \in \Gamma\} \tag{8.2}$$

where δ is one of $=, \subseteq, \subsetneq, \in$. The set of program states

$$\Gamma_{\text{true}} \quad \stackrel{\text{def}}{=} \quad \{\gamma \in \Gamma : \gamma(\mathtt{x})\ \delta\ \gamma(\mathtt{y})\}$$

which satisfy the conditional can be defined using the language of set theory, and therefore a set builder expression for it can be computed by symbol pushing. The set in (8.2) is equal to

$$\{(\gamma, \gamma\mathtt{I}) : \gamma \in \Gamma_{\text{true}}\} \qquad \cup \qquad \{(\gamma, \gamma) : \gamma \in \Gamma - \Gamma_{\text{true}}\},$$

and symbol pushing can be used to compute the appropriate set builder expression.

- *While loop*. Consider a definable program of the form

$$\mathtt{while\ x = y\ do\ J}.$$

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 can compute set builder expressions which represent Γ_n and the semantics of the while loop with domain restricted to Γ_n. We try all n until $\Gamma_n = \Gamma$. Checking the equality $\Gamma_n = \Gamma$ reduces to checking if a first-order sentence is true in the atoms, by the First Symbol Pushing Lemma, and the truth of this sentence is determined using the first-order oracle available in our computation model (this is also the only place where we use the oracle). If no such n exists, then the interpreter does not terminate.

- *For loop*. Our goal is to compute a set builder expression representing

$$\{(\gamma, \gamma(\underbrace{\texttt{for x in X do J}}_{\texttt{I}})) : \gamma \in \Gamma\}. \tag{8.3}$$

It is enough to show that for every program variable $\texttt{y}$, we can compute a set builder expression representing

$$\{(\gamma, y) : \gamma \in \Gamma \text{ and } y \text{ is the value of } \texttt{y} \text{ in } \gamma\texttt{I}\}, \tag{8.4}$$

since the resulting sets of pairs can be rearranged using symbol pushing to get relation on program states from (8.3).

Fix a variable $\texttt{y}$. Compute a representation of

$$\Delta = \{\gamma[\texttt{x} := x] : \gamma \in \Gamma, x \in \gamma(\texttt{y})\}.$$

and use the induction assumption to compute a representation of

$$\{(\delta, \delta\texttt{J}) : \delta \in \Delta\}.$$

Using the above, compute a representation of

$$R = \{(\gamma, y) : \gamma \in \Gamma \text{ and } y \text{ is the value of } \texttt{y} \text{ in } (\gamma[\texttt{x} := x])\texttt{J} \text{ for some } x \in \gamma(\texttt{X})\}.$$

We now need to aggregate, for each program state $\gamma \in \Gamma$, all of the values y with $(\gamma, y) \in R$. 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:

$$\Gamma_\cup = \{\gamma \in \Gamma : \text{there are at least two } y \text{ with } (\gamma, y) \in R\}.$$

A representation of the above set can be used to compute symbol pushing. By definition, the set (8.4) is

$$\{(\gamma, y) \in R : \gamma \in \Gamma - \Gamma_\cup\} \quad \cup \quad \{(\gamma, \{y : (\gamma, y) \in R)\}) : \gamma \in \Gamma_\cup\}, \tag{8.5}$$

A representation of the first summand above is computed using symbol pushing. For the second summand, we need to show that set builder expressions admit a form of Currying, where a binary relation $R \subseteq A \times B$ is converted into the corresponding function $A \to \mathsf{P}B$. This construction is presented in Exercise 137.

□

Turing machines that use representations of the atoms

In item (2) of Theorem 8.11, to process set builder expressions we used a computation model where first-order queries about the atoms could be answered in unit time. This is a questionable assumption. We present below a model without such an assumption, which we call *Turing machines with representation access to the atoms.* In this model, we assume that atoms can be represented as bit strings, and therefore one can use standard Turing machines to process set builder expressions, assuming that the atom parameters are represented using bit strings. This model depends on the way atoms are represented as bit strings, but for most structures (e.g. natural numbers) such a representation is clear, and even in cases where there are several representations (e.g. for the random graph), they lead to the same notion of computability.

Turing machines with representation access to the atoms were used implicitly in Theorem 5.1. Below we show that the two models of Turing machine are equivalent (the result below discusses only functions from tuples of atoms to tuples of atoms, but the extension to set-builder expressions as used in Theorem 8.11 is straightforward).

Theorem 8.14. *Assume:*

(1) *the atom structure is oligomorphic;*
(2) *the atom structure has a computable Ryll-Nardzewski function;*
(3) *each atom can be represented in a finite way so that the first-order theory with atom parameters is decidable, as in the assumptions of Corollary 4.3.*

Then for every equivariant partial function $f : \mathbb{A}^* \to \mathbb{A}^*$ *the following conditions are equivalent:*

(A) is computed by a Turing machine with first-order access to the atoms, as described in Section 8.2;
(B) is computed by a Turing machine where atoms are represented as in assumption (3).

Proof The implication (A)⇒(B) is clear: run the Turing machine from assumption (A) and use assumption (3) to answer the oracle calls. This implication does not need the assumptions on oligomorphism or computability of the Ryll-Nardzewski function.

The more interesting part is (B)⇒(A); here we use all assumptions[6]. Suppose that the input to the function f is an atom tuple $\bar{a} \in \mathbb{A}^*$. Our goal is to compute $f(\bar{a})$ using only first-order access to the atoms $\bar{a}$.

[6] I do not have an example which shows that all of the assumptions are needed.

Claim 8.15. *One can compute the representation of an n-tuple of atoms which is in the same equivariant orbit as $\bar{a}$.*

Proof By Claim 5.28, we can compute a finite set of first-order formulas, each one with n free variables, which define the partition of $\mathbb{A}^n$ into orbits. Using the oracle, we can find the unique formula φ in this set which is true for $\bar{a}$. We can then enumerate through all representations of n-tuples of atoms, until we find one which satisfies φ, using the assumption on decidable first-order theory with parameters. □

Let $\bar{b}$ be the tuple from the above claim. Using assumption (B), compute a representation of the tuple $f(\bar{b})$. Using the same argument as in the above claim, compute a first-order formula ψ which defines the equivariant orbit of the tuple $\bar{b}f(\bar{b})$. By equivariance, the output $f(\bar{a})$ is the unique tuple $\bar{c}$ such that $\bar{a}\bar{c}$ satisfies ψ. This shows how the output of f can be computed using first-order access to the atoms. □

Exercise 137. 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

$$S = \{(x, \{y : (x, y) \in R\}) : x \in X\}.$$

9

Fixed dimension polynomial time

In the previous chapter, we discussed computable operations on sets represented by set builder expressions. In this chapter, we discuss the special case of computable operations which are "tractable" or "polynomial time". We are mainly interested in the equality atoms.

It is not clear what tractability should mean, and the main goal of this chapter is to propose a definition. Such a definition should cover the usual polynomial time operations, such as computing the reachable vertices in a graph. At the very least, it should contain equality tests. For example, the program

```
if x = y then print "equal"
```

should be tractable. Another requirement on the definition is that, for inputs that do not use atoms, such as bit strings, the usual notion of polynomial time is recovered.

The first idea for a definition of tractable computation is to consider operations on set builder expressions which can be computed by a Turing machine in polynomial time. This idea can be formalised in at least three ways – all of them wrong – depending on how the atoms in a set builder expression are handled by Turing machines:

(1) Turing machines have an oracle which answers first-order queries to the atoms, as described in Section 8.2. This model is too strong to be called "tractable", because the oracle can solve PSPACE-hard problems in unit time, see Exercise 138.

(2) As in the previous item, except that only quantifier-free queries can be sent to the oracle. This model is too weak, because it does not even capture set equality, assuming that NP≠P. Indeed, the equality test

$$\emptyset = \{\emptyset : \text{ for } \bar{x} \in \mathbb{A}^k \text{ such that } \varphi(\bar{x})\},$$

corresponds to checking if some atom tuple satisfies φ, which is an NP-hard problem even if we assume that guards in set builder expressions – such as $\varphi(\bar{x})$ – are quantifier-free formulas with only (see Exercises 139 and 140 for a stronger version of this argument, which gives PSPACE lower bound instead of NP, even under the assumption of quantifier-free guards).

(3) The atoms are represented using bit strings, as in the assumptions of Corollary 4.3. This model suffers from the same weakness as the previous one, i.e. it cannot decide set equality in polynomial time.

In all of the above approaches, the underlying problem is that the first-order theory of the atoms is computationally hard (it is PSPACE hard if there are at least two atoms, see Exercise 138). The solution proposed in this chapter is to identify a source of this hardness: the "dimension" of formulas and set builder expressions, which is roughly speaking the number of variables. The proposed notion of tractability avoids computational hardness, by using algorithms whose running time is polynomial, but the degree of the polynomial is allowed to depend on the dimension.

9.1 Fixed dimension polynomial time on set builder expressions

In this section, we present a syntactic description of fixed dimension polynomial time, which talks about programs that input and output set builder expressions. Later, in Section 9.2, we describe the complexity class in a way which is less dependent on the representation using set builder expressions.

To motivate the complexity class, consider the graph reachability problem. Graph reachability will turn out to be in the class, along with other problems that can be solved using fixpoint algorithms.

Example 9.1. Consider graph reachability in the equality atoms: we are given a directed graph and a set of source vertices, and we want to compute the vertices that are reachable from the sources. Recall the algorithm described in Section 5.1, which computes sets

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq \cdots \subseteq V \tag{9.1}$$

where V_i stores vertices of the graph that are reachable in at most i steps from the sources. The sequence stabilises in a finite number of steps, because at each step it adds some orbits of V, with respect to the support of the graph, and there are finitely many of these orbits. Let us calculate in more detail the

number of orbits, which gives an upper bound on the number of steps needed for stabilisation.

To simplify the calculation, assume that vertex set in the graph is a subset of

$$\mathbb{A}^{k_1} + \cdots + \mathbb{A}^{k_n} \qquad \text{for some } k_1, \ldots, k_n \in \{0, 1, \ldots\},$$

i.e. it is a straight set in the sense of Definition 6.7. Let $\bar{a}$ be a tuple of atoms that supports the graph (its vertices, edges and sources). For every $i \in \{0, 1, \ldots\}$ the vertices in V_i are a union of $\bar{a}$-orbits in the vertices. The number of $\bar{a}$-orbits in the vertices is at most

$$\sum_{j \in \{1,\ldots,n\}} \text{number of } \bar{a}\text{-orbits in } \mathbb{A}^{k_j}.$$

One can describe a $\bar{a}$-orbit in $\mathbb{A}^{k_j}$ by a function of type

$$\{1, \ldots, k_j\} \to \{1, \ldots, k_j\} + \text{atoms in } \bar{a} \tag{9.2}$$

which says, for each coordinate, if the coordinate uses an atom from $\bar{a}$ or a fresh atom. The fresh atoms are described using identifiers from $\{1, \ldots, k_j\}$, identifiers can be used several times when coordinates are equal to each other. Different functions can describe the same orbit, by permuting identifiers, and therefore the number of functions as in (9.2) is only an upper bound on the number of orbits. Summing up (literally), the number of $\bar{a}$-orbits of vertices is at most

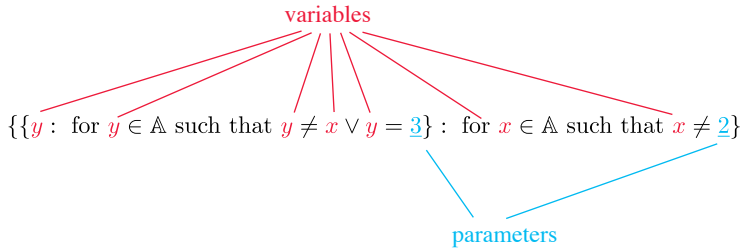
$$\sum_{j \in \{1,\ldots,n\}} (k_j + \text{number of atoms in } \bar{a})^{k_j}.$$

The number above is polynomial once we fix an upper bound on the dimensions $k_1, \ldots, k_n$, but keep the number n of orbits variable. If we think of the vertices as being states of a register automaton, then the dimension is the number of registers, while the number of orbits is related to the number of locations.

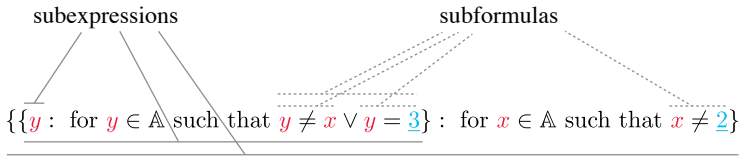
The above example illustrates the idea behind fixed dimension polynomial time: the algorithm should run in polynomial time once the dimension is fixed. In the example, we made a simplifying assumption that the vertices are tuples of atoms, in which case there was a clear notion of dimension. For the general definition, we need to define dimension for arbitrary sets represented by set builder expressions. We do this below.

Size and dimension for set builder expressions. We begin by fixing some terminology. When defining the dimension of a set builder expression, we pay attention to the distinction between parameters (atoms that are written in the

guard formulas) and variables (atoms that are bound inside the set builder expression). This distinction is illustrated in the following picture:

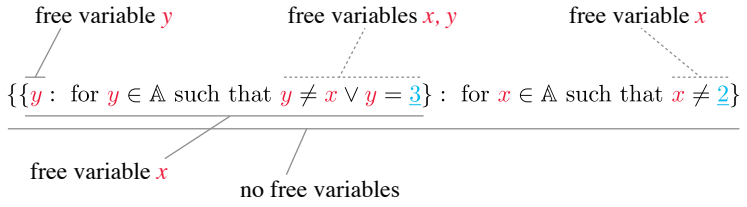


As in Section 4.1, if α is a set builder expression, then a *subexpression* is defined to be any set builder expression that appears nested inside α, and a *subformula* is defined to be any first-order logic formula that appears in a guard. Here is a picture:



Definition 9.2 (Size and dimension of set builder expressions)**.** Let α be a set builder expression. The *size* of α, denoted by $|\alpha|$ is defined to be the number of distinct subexpressions and subformulas[1]. The *dimension* of α, denoted by $\dim(\alpha)$, is defined to be the least $k \in \{0, 1, \ldots\}$ such that every subexpression has at most k free variables, and every subformula has at most $2k$ free variables.

Example 9.3. The set builder expression in the picture before Definition 9.2 has dimension 1, because each of its subexpressions has at most 1 free variable, and each of its subformulas has at most 2 free variables, as explained in the following picture:



[1] This notion of size was already used in Section 4.1.

If the atoms are homogeneous, then every guard can be made quantifier-free. When the guards are quantifier-free, and each subexpression has at most k free variables, then each guard has at most $2k$ free variables, with the upper bound of $2k$ achieved for expressions of the form

$$\alpha(\bar{x}) = \{\beta(\bar{y}) : \text{ for } \bar{y} \in \mathbb{A}^k \text{ such that } \underbrace{\varphi(\bar{x}, \bar{y})}_{\substack{2k \text{ free} \\ \text{variables}}}\}$$

where the free variables of the outer and inner expressions are disjoint. Therefore, if the guards are quantifier-free, the restriction on the variables in the subformulas becomes superfluous. For similar reasons, if a set builder expression has dimension k, then its bound variables can be renamed so that at most $2k$ distinct variables appear in the expression. Therefore, an alternative but essentially equivalent notion of dimension would arise if we simply counted the number of distinct variables that appear in the expression. For reasons that will be explained in Section 9.2, we want the expression in Example 9.3 to have dimension 1 and not 2, which is why we do not use the alternative definition.

Example 9.4. Definition 10.15 gives the expected dimension k for sets of the form $\mathbb{A}^k$, see Exercise 141. A hereditarily finite set, such as this one

$$\{\underline{1}, \{\underline{1}, \underline{5}, \{\underline{6}\}\}, \{\underline{2}, \underline{4}\}\}$$

has no variables, and therefore it has dimension 0.

Definition of fixed dimension polynomial time. Having defined size and dimension of set builder expressions, we can define the complexity class that is the topic of this chapter.

Definition 9.5 (Fixed Dimension Polynomial Time)**.** A program which inputs and outputs set builder expressions[2] is said to run in *fixed dimension polynomial time* if the running time on a set builder expression α is

$$O(|\alpha|^{f(\dim(\alpha))}) \qquad \text{for some computable } f : \mathbb{N} \to \mathbb{N},$$

and the dimension of the output set builder expression is at most

$$g(\dim(\alpha)) \qquad \text{for some computable } g : \mathbb{N} \to \mathbb{N}.$$

[2] The program is modelled as a Turing machine with quantifier-free access to the atoms. This is the same model as in Section 8.2, except that only quantifier-free formulas can be sent to the oracle. For example, if the atoms are the equality atoms, then the Turing machine has an oracle which tests the atoms for equality. An equivalent model would be to use an oracle which inputs, instead of quantifier free formulas, only formulas of the form $R(a_1, \ldots, a_n)$ where $a_1, \ldots, a_n$ are atoms and R is either equality or a relation in the vocabulary of the atoms.

The class in Definition 9.5 is similar to the class XP from parameterized complexity, where the degrees of the polynomials are allowed to depend on the parameter. An alternative would be to consider "fixed dimension tractable" algorithms, see Exercises 146 and 147, where the degree of the polynomial is fixed, but the coefficient in the O notation is allowed to depend on the dimension.

For the rest of this section, we only consider the equality atoms. We will show that natural problems, such as graph reachability, can be solved in fixed dimension polynomial time. The key property of the equality atoms that makes the theory work is that for every $\bar{a} \in \mathbb{A}^n$, the number of $\bar{a}$-orbits in $\mathbb{A}^k$ is at most

$$(n+k)^k,$$

which is polynomial once k is fixed. We assume the equality atoms to make notation lighter, but as shown in the exercises, similar results can be obtained for other choices of atoms, e.g. $(\mathbb{Q}, <)$, where the key property is also true. There are however atoms, e.g. the graph atoms from Section 7.3.1, where the key property and the accompanying results fail.

Quantifier elimination. From now on, we assume the equality atoms. We know that first-order formulas admit quantifier elimination (Theorem 7.6), and that quantifier elimination is effective (the proof of Theorem 7.20). We now strengthen the effectivity result, by showing that quantifier elimination can be done in fixed dimension polynomial time.

Lemma 9.6. *Assume the equality atoms. There is an algorithm which inputs a first-order formula over the vocabulary of the atoms, possibly using atom parameters, and outputs an equivalent quantifier-free formula, and whose running time is at most*

$$\underbrace{(k + (\textit{number of atom parameters}))^k}_{c} \cdot O(\textit{number of subformulas})$$

where k is the maximum number of free variables, ranging over subformulas in φ. The size of the output quantifier-free formula is at most c.

Proof By induction on the formula size, we convert each subformula of φ into a quantifier-free formula. If the subformula has $i \leq k$ free variables and atom parameters $\bar{a} \in \mathbb{A}^n$, then the equivalent quantifier-free formula can be written as a union of $\bar{a}$-orbits in $\mathbb{A}^i$, which has size at most

$$\underbrace{(\text{number of } \bar{a}\text{-orbits in } \mathbb{A}^i)}_{\leq (n+k)^k} \cdot \underbrace{(\text{size of a formula defining a single } \bar{a}\text{-orbit})}_{O(i)}.$$

□

The same argument as in the above lemma works also for the atoms $(\mathbb{Q}, <)$, but it does not work for the random graph, see Exercise 142.

Symbol pushing. Recall the three Symbol Pushing Lemmas from Chapter 4, which showed how operations on set builder expressions (like testing equality or computing the intersection) boil down to operations on first-order formulas (like conjunction or quantification). It turns out that the transformations in the Symbol Pushing Lemmas do increase the dimensions too much, and therefore we can use fixed dimension polynomial time algorithms to process the resulting first-order formulas and set builder expressions. Here is one example of this reasoning, which is a fixed dimension polynomial time version of Corollary 4.3.

Lemma 9.7. *Assume the equality atoms. There is a fixed dimension polynomial time algorithm which does this:*

- **Input.** *Sets X and Y, given by set builder expressions.*
- **Output.** *Which of the relationships $X \in Y$, $X = Y$ and $X \subseteq Y$ are true.*

Proof Consider the first question $X \in Y$. Using the First Symbol Pushing Lemma, compute in polynomial time a first-order sentence φ, such that $X \in Y$ holds if and only if $\mathbb{A} \models \varphi$. A straightforward analysis of the proof of the First Symbol Pushing Lemma shows that the number of free variables in subformulas of φ is bounded by the sum of dimensions in the set builder expressions representing X and Y. Therefore, Lemma 9.6 can be applied to convert φ into a quantifier-free sentence in fixed dimension polynomial time. A quantifier-free sentence is simply a Boolean combination of equalities and inequalities on atom parameters, like this example

$$\underline{1} \neq \underline{2} \wedge (\underline{1} = \underline{1} \vee \underline{2} = \underline{3}),$$

and its truth value can be checked using the oracle in the Turing machine. □

Fixpoint logic. So far, we have shown that fixed dimension polynomial time allows basic operations on set builder expressions, such as testing equality, computing intersection, or quantifier elimination. We now generalise these results, by showing that every fixpoint algorithm runs in fixed dimension polynomial time. This will prove that many natural problems – such as graph reachability or automaton minimisation – can be solved in fixed dimension polynomial time.

To formalise fixpoint algorithms, we use *fixpoint logic*, which is the following extension of first-order logic. Apart from the usual constructors of first

order logic (Boolean operations and quantifiers), we allow a fixpoint operator defined as follows. Suppose that φ is an already defined formula of fixpoint logic, which has free variables $x_1, \ldots, x_n$. Then for every n-ary relation symbol R in the vocabulary of φ, fixpoint logic also has a formula denoted by $\mu R\, \varphi$. This formula has the same free variables as φ, and its vocabulary is the same as for φ but without R. The semantics of this formula are defined as follows. For a structure $\mathbb{B}$ over the vocabulary of $\mu R\, \varphi$, consider the following operation on n-ary relations in $\mathbb{B}$:

$$R \quad \mapsto \quad R \cup \text{tuples that satisfy } \varphi \text{ in } \mathbb{B} \text{ extended with } R. \tag{9.3}$$

Define R_0 to be the empty n-ary relation on $\mathbb{B}$, and for an ordinal number $i > 0$ define R_i to be the result of applying the operation (9.3) to the union $\bigcup_{j<i} R_j$. Since the operation (9.3) is inflationary, i.e. the output contains the input, there must be some fixpoint, i.e. some ordinal number i such that $R_j = R_i$ for all $j > i$. The fixpoint R_i is defined to be the semantics of the formula $\mu R\, \varphi$ in the structure $\mathbb{B}$. This completes the definition of the syntax and semantics of fixpoint logic.

Example 9.8 (Graph reachability)**.** We view a directed graph as a structure where the universe is the vertices and there is one binary relation E for the edges. The fixpoint formula

$$\psi(x_1, x_2) \quad = \quad \mu R\, (x_1 = x_2 \vee \exists z\, E(x_1, z) \wedge R(z, x_2))$$

describes the reachability relation in the graph, and therefore a graph is strongly connected if and only if it satisfies

$$\forall x_1 \forall x_2\, \psi(x_1, x_2).$$

Example 9.9 (Automaton minimisation)**.** We view a nondeterministic finite automaton as a relational structure, where the universe is the disjoint union of states and input letters, there are unary predicates for the states (Q), initial states (I), final states (F) and input letters (I), and there is a ternary predicate (δ) for the transition relation. Suppose that the automaton is deterministic. The following fixpoint formula

$$\psi(q_1, q_2) \;= \mu R \vee \begin{cases} \underbrace{Q(q_1) \wedge Q(q_2) \wedge (F(q_1) \Leftrightarrow \neg F(q_2))}_{\text{one of the states accept the empty word, the other does not}} \\ \underbrace{\exists a\, \exists p_1\, \exists p_2\, \delta(q_1, a, p_1) \wedge \delta(q_2, a, p_2) \wedge R(p_1, p_2)}_{\text{some letter takes } (q_1, q_2) \text{ to a pair of states that accept different words}} \end{cases}$$

selects pairs of states which accept different words. Therefore, the quotient of the reachable states (which can also be described in fixpoint logic, as shown in

Example 9.8) under the equivalence relation defined by the complement of ψ gives the minimal automaton.

Other examples of algorithms that can be formalised in fixpoint logic include nonemptiness for nondeterministic automata, or nonemptiness for context-free grammars. All of these algorithms will run in fixed dimension polynomial time thanks to the following theorem.

Theorem 9.10. *Assume the equality atoms. For every formula $\varphi(x_1, \ldots, x_n)$ of fixpoint logic there is a fixed dimension polynomial time algorithm which does this:*

- **Input.** *A relational structure over the vocabulary of φ.*
- **Output.** *The n-ary relation defined by φ in the input structure.*

Both the input and output are represented by set builder expressions.

Proof Induction on the size of the formula. For the usual operators of first-order logic, namely

$$\varphi \vee \psi \quad \varphi \wedge \psi \quad \exists x\, \varphi \quad \forall x\, \varphi \quad \neg\varphi,$$

we use the Symbol Pushing Lemmas to construct set builder expressions for the semantics of the bigger formula using the semantics of the smaller formulas. The interesting case is the fixpoint formula

$$\mu R\, \varphi(x_1, \ldots, x_n). \tag{9.4}$$

Suppose we want to evaluate the above formula in a relational structure $\mathbb{B}$. Let $\bar{a}$ be the atom parameters that appear in the set builder expression representing $\mathbb{B}$. The tuple $\bar{a}$ supports $\mathbb{B}$ and any relation on $\mathbb{B}$ that can be defined in first-order logic. In particular, it supports the finite approximations

$$\emptyset = R_0 \subseteq R_1 \subseteq \cdots \subseteq \mathbb{B}^n$$

in the definition of the semantics of the fixpoint operator. For the same reasons as in the algorithm for graph reachability, each of these subsets is supported by $\bar{a}$, and therefore the fixpoint will stabilise in a finite number of steps (without needing infinite ordinal numbers such as ω). The rest of this proof is devoted to arguing that the finite number of steps is, in fact, fixed dimension polynomial.

By the Second Symbol Pushing Lemma, the Cartesian product of two sets can be computed in polynomial (and not just fixed dimension polynomial time), assuming representation by set builder expressions. Therefore, since the number of variables n in (9.4) is fixed, it follows that taking the n-th power of a set can be computed in polynomial time. We now show that the partition into orbits can be computed in fixed dimension polynomial time.

Claim 9.11. *The following can be done in fixed dimension polynomial time:*

- **Input.** *A set builder expression α and a tuple of atoms $\bar{a}$ which contains all atom parameters used in α;*
- **Output.** *Set builder expressions that represent the partition of (the set represented by) α into $\bar{a}$-orbits.*

Proof If α is a union expression $\alpha_1 \cup \cdots \cup \alpha_n$, then apply the algorithm to the expressions $\alpha_1, \ldots, \alpha_n$, collect the resulting partitions, and eliminate possible duplicates using Lemma 9.7. It remains to treat the case when α is of the form

$$\alpha = \{\beta(\bar{x}) : \text{ for } \bar{x} \in \mathbb{A}^k \text{ such that } \varphi(\bar{x})\}.$$

Because the guard $\varphi(\bar{x})$ uses only atom parameters from $\bar{a}$, it describes an $\bar{a}$-supported subset of $\mathbb{A}^k$, and it therefore can be rewritten as a disjunction

$$\varphi(\bar{x}) = \varphi_1(\bar{x}) \vee \cdots \vee \varphi_m(\bar{x})$$

where each $\varphi_i(x)$ describes a $\bar{a}$-orbit. By Lemma 9.6, the number of these formulas and the time required to compute them is fixed dimension polynomial. Define $\alpha_1, \ldots, \alpha_m$ to be set builder expressions that are obtained from α by replacing the guard φ with the formulas $\varphi_1, \ldots, \varphi_m$, respectively. Since $\bar{a}$ contains all the atom parameters in β, the function $\bar{b} \mapsto \beta(\bar{b})$ maps $\bar{a}$-orbits contained in $\mathbb{A}^k$ to $\bar{a}$-orbits contained in the set represented by α. Therefore $\alpha_1, \ldots, \alpha_n$ is the list of $\bar{a}$-orbits contained in α. This list might contain duplicates, since $\bar{b} \mapsto \beta(\bar{b})$ need not be injective, but duplicates can be eliminated using Lemma 9.7. □

Apply the above claim to the n-th power of the universe of $\mathbb{B}$, yielding a list Φ of set builder expressions which represent the partition of $\mathbb{B}^n$ into $\bar{a}$-orbits. Consider the sequence of n-ary relations $\{R_i\}_i$ used when defining the semantics of fixpoint logic. Since the semantics of fixpoint logic are equivariant, it follows that each $R_i \subseteq \mathbb{B}^n$ is supported by $\bar{a}$, and can therefore be represented by set a builder expression, call it α_i, which is a union of set builder expressions taken from the list Φ. In particular, the number of steps needed to reach the fixpoint is at most $|\Phi|$, and is therefore also fixed dimension polynomial. We then compute the fixpoint using the induction assumption, by starting with the empty set, and applying in each step the formula φ to the relation computed in the previous step. There is a small caveat: at each step in the fixpoint computation, we need to normalise the set builder expression representing R_i so that it is a union of expressions from Φ, since otherwise the resulting expressions might grow too large. □

Corollary 9.12. *The following problems for hereditarily orbit-finite inputs are in fixed dimension polynomial time:*

(1) *graph reachability;*
(2) *emptiness for nondeterministic automata;*
(3) *minimisation for deterministic automata;*
(4) *emptiness for context-free grammars.*

Exercises

Exercise 138. 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 139. 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 140. Show that the problem from Exercise 139 remains PSpace-complete even if we require the guards in set builder expressions to be quantifier-free.

Exercise 141. Show that the dimension of $\mathbb{A}^k$, according to Definition 9.2, is k.

Exercise 142. 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 polynomial orbit count?

(1) The bit vector atoms.
(2) The tree atoms.
(3) The equivalence relation atoms.
(4) A finite atom structure.

Exercise 143. 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 144. 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 142;
(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 145. 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 144.

Exercise 146. 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 or not $G \models \varphi$, and runs in time

$$f(\varphi) \cdot |G|^c$$

for some computable function f and constant c.

Exercise 147. The issues in Exercise 146 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).

9.2 A semantic version

In the previous section, we defined size and dimension for set builder expressions. These notions were based on the syntax of set builder expressions. One could imagine that different choices of syntax could lead to different notions of size and dimension, and therefore also to different notions of fixed dimension polynomial time computation. In this section we argue that this is not the case, because size and dimension can be defined without referring to set builder expressions, as explained in the following definition.

Definition 9.13 (Size and dimension of a set)**.** Assume the equality atoms. Let X be an atom or a hereditarily orbit-finite set. Define the *dimension of* X, denoted by $\dim X$, to be

$$\max_{x \in X_*} \underbrace{|\overbrace{\sup(x)}^{\substack{\text{least}\\ \text{support}\\ \text{of } x}} - \overbrace{\sup(X)}^{\substack{\text{least}\\ \text{support}\\ \text{of } X}}|}_{\text{difference of two sets}}$$

where X_* denotes the set that contains X, its elements, their elements, and so on (see Definition 4.6). Define the *size of* X, denoted by $|X|$ to be the number of orbits in X_* with respect to atom automorphisms that fix all atoms in the set $\sup(X)$.

The above definition defines dimension and size in purely semantic terms, without referring to syntactic representations, such as set builder expressions. We will argue that the above definition agrees with the syntactic notions from the previous section. The dimension will turn out to be the exact same number, i.e. the dimension from the above definition will be equal to the least dimension of set builder expressions defining the give set. For size, the exact numbers will be different, but they will agree up to fixed dimension polynomial corrections.

Example 9.14. Consider the set $X = \mathbb{A}^k$. This set can be represented by a set

builder expression of the form

$$\{ \underbrace{(x_1, \ldots, x_k)}_{\substack{\text{syntactic sugar for} \\ \text{iterated application} \\ \text{of Kuratowski pairs}}} : \text{for } (x_1, \ldots, x_k) \in \mathbb{A}^k \}$$

which has size linear in k. What is the dimension of X, as per Definition 9.13? The set X is equivariant (i.e. it has empty support), and the maximal size of a support elements of X_* is k. Therefore, the dimension is k. What is the size? The number of orbits in X_* with respect to all automorphisms (because X has empty support) is at least[3] the number of equivariant orbits in $\mathbb{A}^k$, and is therefore exponential in k.

The above example shows that the size of a set, as per Definition 9.13, can be exponentially larger than the size of a set builder expression that represents it. Nevertheless, as we explain below, the exponential uses the dimension, and becomes a polynomial once the dimension is fixed.

Canonical expressions. To relate the two kinds of dimension and size, i.e. the syntactic one from the previous section and the semantic one from Definition 9.13, we associate to each hereditarily orbit-finite set X a set builder expression, called its *canonical expression*, and we show that its size and dimension (as defined for set builder expressions) is related to the size and dimension of the represented set X (as defined for sets in Definition 9.13). A set might get several canonical expressions, but these will be isomorphic in a sense that will be defined below.

Definition 9.15 (Canonical expression)**.** Canonical expressions are defined by induction on rank, i.e. the nesting depth of set brackets.

- For an atom, its canonical expression is the atom itself.
- For the empty set, its canonical expression is the set itself.
- Consider a nonempty hereditarily orbit-finite set X. Let

  $$X = X_1 \cup \cdots \cup X_n$$

  be the partition of X into orbits with respect to its least support. In other words, each of the sets $X_1, \ldots, X_n$ is an orbit with respect to atom automorphisms that are the identity when restricted to the least support of X. If $n \geq 2$, then define the canonical expression of X to be the union of the canonical expressions of the orbits $X_1, \ldots, X_n$. This expression depends on the ordering

[3] We write "at least" because X_* contains elements that are not k-tuples of atoms, due to the definition of tuples as sets in Kuratowski encoding.

of the orbits $X_1, \dots, X_n$ and the choices of canonical expressions for these orbits.

We are left with the case when X has exactly one orbit with respect to its least support. Choose some $x \in X$, and choose some canonical expression α of x, obtained by induction assumption. (The resulting expression will depend on these choices, but all choices will lead to isomorphic expressions.) Consider the atom parameters that appear in α but do not appear in the least support of X. Order these atom parameters according to their order of appearance α, yielding a list of atoms $b_1, \dots, b_k$. In the expression α, replace the atoms $b_1, \dots, b_k$ with fresh variables $x_1, \dots, x_k$, yielding a set builder expression $\beta(x_1, \dots, x_k)$. The canonical expression for X is defined to be

$$\{\beta(x_1, \dots, x_k) : \text{ for } x_1, \dots, x_k \in \mathbb{A} \text{ such that } \varphi(x_1, \dots, x_k)\}$$

where φ is any quantifier-free formula defining the orbit of the tuple $(b_1, \dots, b_k)$ with respect to the least support of X.

Example 9.16. A canonical expression of

$$(\underline{1}, \underline{2}) = \{\underline{1}, \{\underline{1}, \underline{2}\}\}$$

is the one give on the right side above. In general, for every set that is hereditarily finite (not just orbit-finite), the canonical expression has dimension 0, i.e. it uses no bound variables.

The canonical expression of $\mathbb{A}^2$ is

$$\{(x_1, x_2) : \text{ for } (x_1, x_2) \in \mathbb{A}^2 \text{ such that } x_1 \neq x_2\} \quad \cup \quad \{x : \text{ for } x \in \mathbb{A}\}.$$

Call two set builder expressions are *isomorphic* if one can be obtained from the other by changing the order of unions, renaming bound variables, and replacing guards by equivalent quantifier-free formulas which use the same atom parameters. For example

$$\{x : \text{ for } x \in \mathbb{A} \text{ such that } x \neq \underline{2}\} \cup \{\underline{1}\}$$

is isomorphic to

$$\{\underline{1}\} \cup \{y : \text{ for } y \in \mathbb{A} \text{ such that } \underline{2} \neq y \wedge y = y\}.$$

The following lemma shows that the canonical expression is unique up to isomorphism (and therefore it makes sense to talk about *the* canonical expression in contexts that do not distinguish between isomorphic expressions), uses only atoms from the least support, and has optimal dimension and small size.

Lemma 9.17. *Let X be a hereditarily orbit-finite set, and let α be a canonical expression. Then*

(1) *all other canonical expressions of X are isomorphic to α:*
(2) *the atom parameters in α are exactly the least support of X;*
(3) *the dimension of α is exactly* $\dim X$*;*
(4) *there is a constant c that depends only on* $\dim X$ *such that*

$$\underbrace{|\alpha|}_{\text{Definition 9.2}} \leq \underbrace{c \cdot |X|}_{\text{Definition 9.13}}.$$

Proof The first two items are proved by a straightforward induction on the rank of X, so we focus only on items (3) and (4). By definition, every subexpression β of the canonical expression α can be obtained as follows: take a canonical expression of some $x \in X_*$ and replace all atom constants that are not in in the least support of X with free variables. The number of free variables introduced this way is at most $\dim X$, thus proving item (3). The same argument shows that the number of subexpressions – *modulo isomorphism* – in the canonical expression of X is exactly $|X|$. Since the size of a set builder expression is the number of subexpressions not modulo isomorphism, to prove (4) we additionally observe that one can rename the variables in the canonical expression so that at most c distinct subexpressions are isomorphic to each other, where c is a constant that depends only on the dimension k and does not depend on X. □

The previous lemma showed that canonical expression is not too big. The following lemma shows that it is not too small: its dimension is minimal, and its size is minimal among expressions with the same dimension, up to a polynomial correction.

Lemma 9.18. *Let X be a hereditarily orbit-finite set, and let α be any set builder expression that represents it. Then the dimension of α is at least* $\dim X$*. Furthermore, if the dimension of α has the optimal value*[4] $\dim X$*, then*

$$|X| \leq |\alpha|^{2 \cdot (\dim X)}.$$

Proof We begin by showing that α has dimension at least $\dim X$. To show this, we prove that some subexpression of α has at least $\dim X$ free variables. Let $x \in X_*$ be a witness for $\dim X$, which means that the least support of x contains atoms $b_1, \ldots, b_{\dim X}$ that are not in the least support of X. Choose an atom automorphism π which is the identity on the least support of X and which maps $b_1, \ldots, b_{\dim X}$ to some fresh atoms that not appear as constants in the expression α. Since π is the identity on the least support of X (which is also the least support of X_*) it follows that $\pi(x) \in X_*$. The set $\pi(x)$ must be obtained

[4] Exercise 148 explains the assumption on optimal value.

from some subexpression of α by instantiating variables to atom parameters, and since α does not use the atoms $\pi(b_1), \ldots, \pi(b_{\dim X})$ in its syntax, it follows that the subexpression must have at least $\dim X$ free variables.

It remains to show the upper bound on the size of $|X|$. Let α be a set builder expression of optimal dimension $\dim X$ which represents X, and let $\bar{a}$ be the atoms that appear in α. The atoms $\bar{a}$ support X. Every $x \in X_*$ is obtained by taking some subexpression β of α and assigning atoms parameters to its free variables. Because only atoms from $\bar{a}$ appear in β, it follows that β maps $\bar{a}$-orbits to $\bar{a}$-orbits, and therefore every $\bar{a}$-orbit contained in X is of the form $\beta(Y)$ where β is a subexpression of α and Y is an $\bar{a}$-orbit of valuations of the free variables in β. It follows that the number of $\bar{a}$-orbits in X_* is at most

$$(\text{number of subexpressions of } \alpha) \cdot (\text{number of } \bar{a}\text{-orbits in } \mathbb{A}^{\dim X}). \tag{9.5}$$

In the above we use the assumption that α has optimal dimension, and therefore every subexpression β has at most $\dim X$ free variables. The size $|X|$ is defined to be the number of orbits in X_* with respect to the least support of X, and this least support is contained in $\bar{a}$. Since bigger supports lead to more orbits, it follows (9.5) is an upper bound on $|X|$. If $\bar{a}$ has n atoms, then the number of $\bar{a}$-orbits in $\mathbb{A}^{\dim X}$ is at most $(n + \dim X)^{\dim X}$, and since $n + \dim X$ is at most the size of α, we get the desired inequality. □

Putting the above two lemmas together, we see that the optimal dimension for expressions defining a set X is exactly $\dim X$ and

$$\sqrt[d]{|X|} \overset{\text{Lemma 9.18}}{\leq} \underbrace{|\alpha|}_{\substack{\text{miminal size of expression} \\ \text{that represents } X \text{ and} \\ \text{has dimension } \dim X}} \overset{\text{Lemma 9.17}}{\leq} c|X|$$

for constants c, d that depend only on $\dim X$.

By formalising the definition of canonical expressions, one can show that there is a fixed dimension polynomial time algorithm which inputs a set builder expression and outputs the canonical expression of the underlying set. A corollary is that a function

$$f : \text{hereditarily orbit-finite sets} \to \text{hereditarily orbit-finite sets}$$

can be computed in fixed dimension polynomial time if and only if there is an algorithm which inputs a canonical expression of a set X, and outputs the canonical expression of $f(X)$, and which runs in polynomial time for sets of fixed dimension $\dim X$. This characterisation is less syntactic than the original definition. It is still somehow syntactic, since it uses the canonical expression,

although one could argue that the canonical expression can be defined without using set builder notation.

Fixed dimension polynomial while programs. To end this chapter, we show a sufficient condition for fixed dimension polynomial time, which does not mention any representation of hereditarily orbit-finite sets such as set builder expressions. The sufficient condition is defined as a restriction on the while programs from Chapter 8. The general idea is to bound the time and space used by a while program so that it is fixed dimension polynomial. To do this, we need to define the time and space used by a while program. To define time, we add a new program variable called `time`, which stores the running time encoded as a Von Neumann numeral. The variable is initialised to 0. To update the `time` variable, we modify the original program by inserting an instruction

$$\texttt{time} := \texttt{time} \cup \{\texttt{time}\}$$

after every instruction. The above instruction simply increments the numeral representing the time. For Von Neumann numerals, maximum is the same as set union, and therefore the running time of a for loop is the same as the maximal running time of each of its threads. The running time of a program is then defined to be the value of the `time` variable at the end of the program.

Example 9.19. Recall the program which projects a Kuratowski pair `p` to its first coordinate. Here is its annotation with the `time` variable:

```
ret := ∅
time := time ∪ {time}
for a in p do
  for x in a do
    for y in p do
      if p = {x,{x,y}} then
        ret:={x};
        time := time ∪ {time}
      time := time ∪ {time}
    time := time ∪ {time}
  time := time ∪ {time}
time := time ∪ {time}
```

The running time of this program is 5 or 6, depending on whether the `if` is executed. More generally, every program that does not use `while` has constant running time.

To define the space consumption, we use a similar idea. We add a variable `space`, initially the empty set, and modify the original program by adding

$$\texttt{space := space} \cup \{\texttt{x}, \texttt{y}, \texttt{z}, ..\}$$

after each instruction, where `x`, `y`, ... are all of the program variables (a finite list for every program). The space consumption is then defined to be the size of the variable `space` at the end of program execution, with size measured according to Definition 9.13.

Definition 9.20. A *fixed dimension polynomial while program* is a while program[5] where the running time and space consumption (both of which are natural numbers) are fixed dimension polynomial in terms of the input (the size and dimension of the input are measured according to Definition 9.13). Furthermore, the dimension of the output must be bounded by a function of the dimension of the input.

A more detailed analysis of the semantics of while programs shows that

$$\underbrace{\text{fixed dimension polynomial while programs}}_{\text{Definition 9.20}} \quad \subseteq \quad \underbrace{\text{fixed dimension polynomial time.}}_{\text{Definition 9.5}}$$

In general, the above inclusion is strict, which follows from similar results about Choiceless Polynomial Time, see (Rossman, 2010, Section 6). For decision problems, where the output is just true or false, the above inclusion could be an equality, for all we know. However, proving equality for decision problems would imply that Choiceless Polynomial Time captures polynomial time for decision problems, which is a well-known open problem. All of these results can be found in Bojańczyk and Toruńczyk (2018).

Exercises

Exercise 148. In Lemma 9.17, the bound $|X| \leq |\alpha|^{2 \dim X}$ assumes that α has minimal dimension. Show that the assumption on minimal dimension 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.

Exercise 149. In Lemma 9.17, the bound $|X| \leq (\text{size of } \alpha)^{\dim X}$ has $\dim X$ in

[5] We assume that while programs have a built-in atomic operation `X:=|Y|` which computes the size of the set stored in `Y`, and loads the corresponding Von Neumann numeral into `X`. This operation can be computed in exponential time – and is therefore not needed if we do not care about running time – but it is needed as a primitive if we want any hope to capture polynomial time.

the exponent. Show that one cannot replace the exponent by a constant independent of $\dim X$.

Exercise 150. Show that the $|X|$ is not upper bounded by any fixed dimension polynomial function of the parameter "number of equivariant orbits that intersect X_*".

10
Turing machines

In Chapters 8 and 9, we used while programs for computation with atoms.

There is a reason why we did not use Turing machines.

The input of a Turing machine is a string over some input alphabet. Therefore, using Turing machines for objects that are not strings requires representing those objects as strings. Even without atoms, string representation can be problematic. For example, representing a graph as a string imposes an order on the vertices, an order that could potentially be exploited by an algorithm[1]. In the presence of atoms, the issues related with string representation get only harder. This is why Turing machines are not the most natural model to compute on orbit-finite objects, and in Chapters 8 and 9 we used while programs that worked directly on sets.

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 10.2, we prove that $\textsc{p} \subsetneq \textsc{np}$ holds in the bit vector atoms. This is witnessed by the language

  $$\{a_1 \cdots a_n \in \mathbb{A} : a_1, \ldots, a_n \text{ are not linearly independent}\},$$

  which is not recognised by any deterministic Turing machine in polynomial time, but it is recognised in nondeterministic polynomial time (guess the linear combination that gives zero) or deterministic exponential time (try all combinations).
- In Section 10.3, we prove a similar separation for the equality atoms. Namely, there is a language that is recognised by a nondeterministic Turing machine

[1] To read more about these issues, see Grohe (2008).

(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 10.2, and uses the Cai-Fürer-Immerman construction from finite model theory.

The separations described above are unlikely to be useful in separating complexity classes without atoms. The accompanying proofs are based on the limited access that Turing machines have to their input and on the symmetries that result from applying atom automorphisms.

10.1 Orbit-finite Turing machines

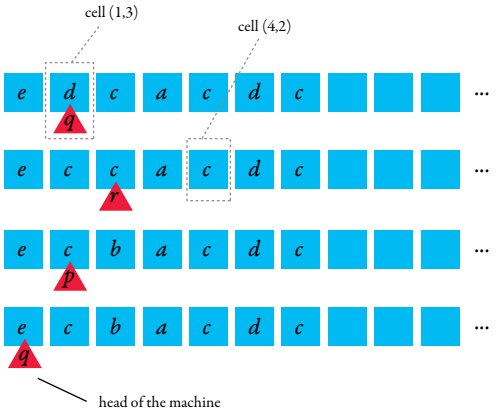
Before presenting the separation results about Turing machines, we begin by introducing Turing machines with atoms, discussing some examples, and explaining the importance of alternating Turing machines. We assume for this section that the atoms are oligomorphic.

An orbit-finite Turing machine is defined the same way as a Turing machine, except that all components (states, input and work alphabets, transitions) are required to be orbit-finite sets with atoms. For the sake of concreteness, we use a model which has single tape, infinite to the right, where the transition relation is a subset of:

$$\underbrace{(\overbrace{\Gamma}^{\text{work alphabet}} \cup \{\text{blank}\}) \times \overbrace{Q}^{\text{states}}}_{\text{what the machine sees}} \times \underbrace{(\{\text{accept, reject}\} \cup (\overbrace{\Gamma \times Q \times \{\text{left, stay, right}\}}^{\text{write a symbol and move the head}}))}_{\text{what the machine does}}.$$

Recall that every orbit-finite set admits a finitely supported bijection with a set that is hereditarily-orbit finite. Since the questions about Turing machines that we study are invariant under renaming state spaces and alphabets, we can assume without loss of generality that the Turing machines are hereditarily orbit-finite (and therefore can be represented using set builder expressions by Theorem 4.10).

Here is a picture of a computation:



Many results for Turing machines remain true for orbit-finite Turing machines, such as equivalence of single-tape and multi-tape machines, using the standard proofs. What does not work, however, is determinisation; this will be discussed in Sections 10.2 and 10.3. For all we know, nondeterministic machines are weaker than alternating machines, this will be discussed later in this section.

Example 10.1 (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 Turing machine which accepts words where all letters are distinct, and the atom $\underline{5}$ does not appear. 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 $a \neq \underline{5}$, replace it by a blank and load a into the state, scan the word to check that a does not appear again, and reading the entire word go back to the beginning of the tape. If the first non-blank letter on the tape is $\underline{5}$, then reject immediately. The set of states is

$$\underbrace{\{q_0\}}_{\text{initial state}} \cup \underbrace{\mathbb{A} - \{\underline{5}\}}_{\text{scan to the right if this atom reappears}} \cup \underbrace{\{q_1\}}_{\text{return to the beginning}} .$$

An accepting run of this machine is illustrated in Figure 10.1.

Example 10.2 (Deatomisation)**.** Consider the equality atoms. This example shows that when the input alphabet is the atoms, then a Turing machine can begin by removing the atoms from the input, and then carry on its computation without using atoms.

Define the *deatomisation* of a sequence of atoms $a_1 \cdots a_n$ to be the word

$$i_1 \# i_2 \# \cdots \# i_n \in \{0, 1, \#\}^*$$

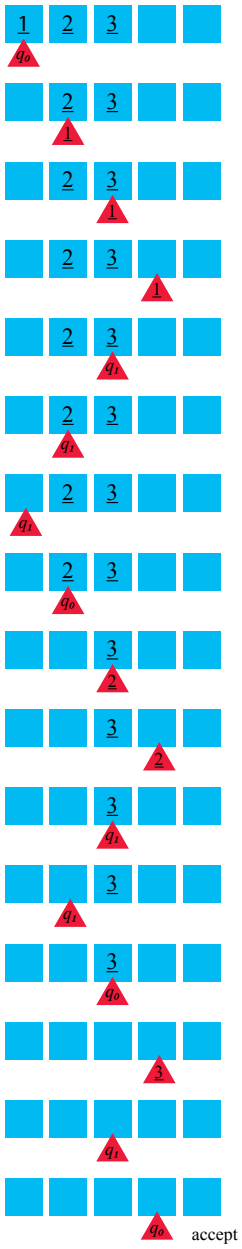


Figure 10.1 An accepting run of the Turing machine from Example 10.1 which checks checks if all input letters are distinct and different from 5.

such that i_k is the binary encoding of the number which represents the first position where atom a_k appears. Here is a picture:

a sequence of atoms	2		1		1		9		1
position numbers in binary	1		10		11		100		101
deatomisation	1	#	10	#	10	#	100	#	10

The point of deatomisation is that it stores all information about the input word up to atom automorphisms. It is not difficult to write a deterministic Turing machine which transforms a sequence of atoms into its deatomisation. The machine only needs to test letters for equality. Therefore, any decidable and equivariant (or finitely supported) language $L \subseteq \mathbb{A}^*$ can be computed by a deterministic Turing machine, which first computes the deatomisation, and then runs a Turing machine without atoms. Deatomisation works when the input alphabet is $\mathbb{A}$. It would also work for some other input alphabets, see the next example. As we will see in Section 10.3, a deterministic deatomisation procedure is not going to be possible for some orbit-finite alphabets.

Example 10.3 (A more fancy input alphabet)**.** Consider the equality atoms. Let the input alphabet be sets of atoms of size at most ten. We describe below a deterministic Turing machine which recognises the language: "there exists an atom that appears in an odd number of letters".

The difficulty is with indicating the atom that appears in an odd number of letters. As an example, suppose that the input word is:

$$\{\underline{1}, \underline{2}, \underline{3}\}\{\underline{1}, \underline{2}, \underline{4}\}\{\underline{1}, \underline{2}\}\{\underline{1}, \underline{2}, \underline{3}\}\{\underline{1}, \underline{2}\}\{\underline{4}\}.$$

Both atoms $\underline{1}$ and $\underline{2}$ 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 $\underline{1}$ and $\underline{2}$. (Which means that swapping $\underline{1}$ and $\underline{2}$ is an atom automorphism that does not change the input word.) Here is a solution to the problem. Suppose that the input word is

$$A_1 \cdots A_n \qquad \text{where } A_1, \ldots, A_n \subseteq \mathbb{A} \text{ have size at most 10.}$$

The Turing machine executes the following program:

(1) In a fresh part of the tape, generate a copy of the input word. After this step

the tape has the form

$$\underbrace{A_1 \cdots A_n}_{\text{first part}} \mid \underbrace{A_1 \cdots A_n}_{\text{second part}}$$

(2) As long as possible, iterate these steps on the second part of the tape:
   (i) if some set appears multiple times, remove all of the duplicates;
   (ii) if A, B are distinct intersecting sets, then remove them and add the nonempty sets among $A - B, B - A, A \cap B$.

(3) At this point, the second part 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 part) is contained in an even number of sets from the first part.

All of the above steps can be performed by an orbit-finite Turing machine. To process sets, the machine can use state space, which can store a set of at most 10 atoms.

Computational completeness of Turing machines

In Chapter 8, we proved that while programs were computationally complete, in the sense that they could compute exactly those functions that could be computed using set builder representation. In this section, we revisit computational completeness for Turing machines.

Assume that the atoms have a decidable first-order theory with parameters, which means that hereditarily orbit-finite sets can be represented in a finite way, using set builder expressions. Suppose that the input alphabet Σ in a Turing machine is hereditarily orbit-finite. This means that letters of the input alphabet and words over the input alphabet can be represented in a finite way, and we can ask if orbit-finite Turing machines are equivalent to the usual Turing machines working on representations:

Question. Are the following conditions equivalent for a finitely supported language $L \subseteq \Sigma^*$ over a hereditarily orbit-finite alphabet Σ?

(1) Some Turing machine (in the usual, not orbit-finite, sense) decides membership $w \in L$, assuming that w is represented by a set builder expression.
(2) Some orbit-finite Turing machine decides membership $w \in L$, assuming that w is given directly, not by its representation.

In the above question, we care about decidable languages, i.e. we require the Turing machines to give a yes or no answer in finite time, but similar results will hold for semi-decidable languages. Recall that in Theorem 8.11, we

proved that condition (1) is equivalent to L being recognised by a while program with atoms. Therefore, another way of posing the above question is: for languages over hereditarily orbit-finite alphabets, are Turing machines equivalent to while programs?

The question above is less fundamental than the one discussed in Section 8.2 about while programs, because it only talks about computation over inputs which are words over a fixed hereditarily orbit-finite alphabet. As discussed at the beginning of this chapter, such inputs are too restricted to naturally model objects like orbit-finite graphs or automata. Nevertheless, the question is interesting because it has a slightly unexpected answer: (a) for some atoms (e.g. equality), deterministic orbit-finite Turing machines are not computationally complete; (b) for some atoms (e.g. equality), nondeterministic orbit-finite Turing machines are computationally complete; and (c) for all atoms, alternating orbit-finite Turing machines are computationally complete. The positive results (b) and (c) are given in Theorem 10.4 below, while the negative result (a) about deterministic machines is presented in Section 10.3.

Alternating machines. Before proving computational completeness of alternating Turing machines, we describe the underlying model. The syntax of an alternating Turing machine is the same as for normal Turing machines, except that the control states are partitioned into four groups: *existential*, *universal*, *accepting* and *rejecting*. Define a run of an alternating Turing machine to be a well-founded tree whose nodes are labelled by configurations, such that nodes that use existential control states have one child with a successor configuration, nodes that use universal control states have all possible successor configurations as children, and nodes with accepting or rejecting control states are leaves. A run is accepting if the root has an initial configuration (i.e. an input word with the head over the first position in the initial state) and where all leaves are accepting. The language recognised by a definable alternating Turing machine is those input words which admit at least one accepting run.

The main result in this section is the following theorem about computational completeness of Turing machines.

Theorem 10.4. *Assume that the atoms*

- *have finite vocabulary;*
- *are oligomorphic;*
- *have a computable Ryll-Nardzewski function;*
- *have decidable first-order theory with parameters.*

Then the following conditions are equivalent for every $L \subseteq \Sigma^*$ where Σ is hereditarily orbit-finite:

(1) *L is finitely supported and there is a Turing machine which decides $w \in L$, with w represented by a set builder expression;*
(2) *L is recognised by an orbit-finite alternating Turing machine.*

If additionally the atoms have quantifier elimination, then the above conditions are also equivalent to

(3) L is recognised by an orbit-finite nondeterministic Turing machine.

The theorem implies that for atoms such as $(\mathbb{N}, =)$ or $(\mathbb{Q}, <)$, nondeterministic machines are computationally complete. Actually, the assumption on quantifier elimination can be relaxed, see Exercise 155, and – to the author's best knowledge – might not be needed at all.

The rest of Section 10.1 is devoted to proving Theorem 10.4.

We begin with the implications (2) $\Rightarrow$ (1) and (3) $\Rightarrow$ (1), which say that orbit-finite Turing machines can be simulated when inputs are given by representations. By Theorem 8.11, it is enough to show that alternating (and therefore also nondeterministic) orbit-finite Turing machines can be simulated by while programs. By Exercise 68, if an alternating orbit-finite Turing machine has a well-founded run, then it also has one with some finite bound $n \in \mathbb{N}$ on the length of all paths; this bound is called the *depth* of the run. Such runs can be searched by a while program with atoms.

It remains to show the converse implications.

We begin by describing the reason why alternating machines are used: they can evaluate formulas of first-order logic. A formula φ of first-order logic can be encoded as a bit string in a natural way, let us write $\underline{\varphi}$ for this encoding. Alternating Turing machines are a perfect model for evaluating first-order formulas, as shown in the following lemma.

Lemma 10.5. *If the atoms have a finite vocabulary, then the following language over alphabet $\mathbb{A} + \{0, 1\}$ is recognised by an alternating Turing machine:*

$$\{a_1 \cdots a_n \underline{\varphi} : a_1, \ldots, a_n \in \mathbb{A},\ \varphi \text{ has n free variables, and } \mathbb{A} \models \varphi(a_1, \ldots, a_n)\ \}.$$

For quantifier-free formulas, a deterministic machine is enough.

Proof The machine simply implements the semantics of first-order logic, using universal states for the universal quantifiers and existential states for the existential quantifiers. The assumption that the vocabulary of the atoms is finite is used in the induction base, for atomic formulas, in which case the relations of the atoms are simply hard-coded into the Turing machine. □

The above lemma gives an alternative solution to Example 10.2, since having only distinct letters can be expressed by a quantifier-free formula. The

following lemma shows computational completeness in the special case when the input alphabet is $\mathbb{A}$. A special case of the lemma for the equality atoms is the deatomisation procedure that was discussed in Example 10.2.

Lemma 10.6. *If a language $L \subseteq \mathbb{A}^*$ satisfies* (1) *from Theorem 10.4, then it is recognised by an alternating orbit-finite Turing machine. If the atoms have quantifier elimination, a deterministic machine is enough.*

Proof Suppose that L satisfies (1), i.e. it is finitely supported, say by some atom tuple $\bar{a}$, and there is a Turing machine without atoms which recognises the representations of words in L. Suppose that the input word is

$$b_1 \cdots b_n \in \mathbb{A}^*.$$

By the same reasoning as in the proof of Theorem 8.14, a Turing machine can compute a finite set of formulas

$$\varphi_1(x_1, \ldots, x_n), \quad \ldots, \quad \varphi_k(x_1, \ldots, x_n)$$

of first-order logic which define the $\bar{a}$-orbits of $\mathbb{A}^n$. By Lemma 10.5, an alternating orbit-finite Turing machine can check which formula φ_i is true for the input word. Since the language is supported by $\bar{b}$, acceptance of the input word depends only on φ_i. Using the φ_i and the assumption that L can be computed based on representations, membership in L can then be determined.

In the above argument, we used alternation only to evaluate the formulas $\varphi_1, \ldots, \varphi_k$ in the input word. All the other steps could be done by a deterministic Turing machine. If the atoms have quantifier elimination, then the formulas can be assumed to be quantifier-free, and therefore only a deterministic orbit-finite Turing machine is needed. (It is worth pointing out that, in the presence of a decidable first-order theory with parameters, quantifier elimination must necessarily be effective: one can enumerate through all quantifier-free formulas, and halt when an equivalent one is found, with equivalence being computed using the first-order theory.) □

We are now ready to complete the proof of Theorem 10.4. Suppose that $L \subseteq \Sigma^*$ satisfies (1). Like for any orbit-finite set Σ, there exists some k and a finitely supported surjective function

$$f : \mathbb{A}^k \to \Sigma.$$

Extend this function to a partial function

$$f^* : \mathbb{A}^* \to \Sigma^*$$

which is defined only on words of length divisible by k and simply applies f to

every block of k letters. It is not difficult to see that the inverse image of L under f^* also satisfies condition (1). Therefore, by Lemma 10.6, the inverse image of L under f^* is recognised by an alternating Turing machine (and a deterministic one in case $\mathbb{A}$ admits quantifier elimination). Using nondeterminism, one can guess a word in $\mathbb{A}^*$ that maps to the input word via f^*, and then run the machine from Lemma 10.6 on this guessed word. This completes the proof of Theorem 10.4.

Exercises

Exercise 151. 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 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 a finitely supported function $A^k \to A$ for some k) and a finitely supported set $F \subseteq A$ such that for every $n \in \mathbb{N}$ there one can compute a term over the functions $\mathcal{F}$ which has n free variables and satisfies

$$a_1 \cdots a_n \in L \quad \text{iff} \quad t(a_1, \ldots, a_n) \in F \qquad \text{for every } a_1, \ldots, a_n \in \Sigma.$$

Exercise 152. Assume that the atoms are oligomorphic and admit least supports. Show that a language $L \subseteq \mathbb{A}^*$ is recognised by a deterministic Turing machine if and only if:

(**) There exists a finite family $\mathcal{R}$ of functions (each one being a finitely supported 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 for every $n \in \mathbb{N}$ one can compute a quantifier-free formula with n free variables over vocabulary $\mathcal{R}$ which defines $L \cap \mathbb{A}^n$.

Exercise 153. Assume that the atoms are oligomorphic. Show that a language $L \subseteq \mathbb{A}^*$ is recognised by a nondeterministic Turing machine if and only if:

(***) There exists a finitely supported relation $S \subseteq \mathbb{A}^k$ such that for every $n \in \mathbb{N}$, one can compute an existential formula that uses only the relation S and equality, and which defines the $L \cap \mathbb{A}^n$. Here an existential formula is one of the form $\exists \bar{y} \in \mathbb{A}^m \varphi(\bar{x}\bar{y})$ where φ is quantifier-free.

Exercise 154. Assume that the atoms admit least supports, and are homogeneous over a relational vocabulary. Show that nondeterministic and deterministic Turing machines recognise the same languages over input alphabet $\mathbb{A}$.

Exercise 155. Let $\mathbb{A}$ be an oligomorphic structure, with decidable first-order theory with parameters and a computable Ryll-Nardzewski function. Show that following conditions are equivalent:

(1) nondeterministic Turing machines recognise the same languages as alternating ones;
(2) $\mathbb{A}$ is has the same automorphism group as a structure where the vocabulary is finite, and for every first-order formula (with free variables) one can compute an equivalent existential one (i.e. one which uses only existential quantifiers in prenex normal form).

10.2 For bit vector atoms, P ≠ NP

In this section we show that P ≠ NP holds for the bit vector atoms, which were introduced in Section 7.3.2. Actually, we prove that even orbit-finite nondeterministic automata can go beyond deterministic polynomial time orbit-finite Turing machines.

We begin by recalling the bit vector atoms. This is the vector space over the two-element field of countably infinite dimension. More formally, the universe of the structure is the vectors in $\{0,1\}^\omega$ that have finitely many ones, equipped with one binary operation for coordinatewise addition modulo 2.

We say that a tuple $\bar{a} \in \mathbb{A}^n$ is *linearly dependent* if

$$0 = \sum_{i \in I} a_i \qquad \text{for some nonempty } I \subseteq \{1, \ldots, n\}.$$

The definition takes into account the tuple and not just the underlying set. If the tuple contains a repetition, then it is linearly dependent. If a tuple is not linearly dependent, then it is called independent.

This section is devoted to proving the following separation result.

Theorem 10.9. *Assume the bit vector atoms. The language*

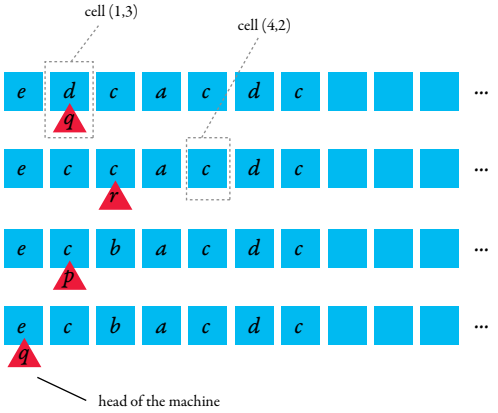
$$\{\bar{a} \in \mathbb{A}^* : \bar{a} \text{ is linearly dependent}\}$$

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.

To recognise the language using a nondeterministic orbit-finite automaton, one simply guesses the nonempty subset of positions which leads to a zero sum[2]. The partial sum is an atom, and therefore it can be stored in the state of an orbit-finite automaton.

The rest of this section is devoted to proving the lower bound about deterministic Turing machines. Fix some deterministic orbit-finite Turing machine. To make notation lighter, we assume that the machine is equivariant, i.e. its alphabets, states and transition function are equivariant. The proof can then be easily extended to work for finitely supported machines. 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 can be visualised as a grid:



Such a grid is formalised as a function

$$\rho : \underbrace{\mathbb{N}^2}_{\text{cells}} \to \underbrace{\Gamma + \Gamma \times Q}_{\text{cell contents}},$$

where labels from $\Gamma \times Q$ are used for cells containing the head, and labels from Γ are used for the other cells. 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 cells at at time.

[2] Alternatively, we could use a deterministic exponential time Turing machine, which tries out all possible choices of coefficients. This contrasts the situation in Section 10.3, where even a deterministic Turing machine with arbitrary time will be unable to compute the language of interest.

Lemma 10.10. *Suppose that*

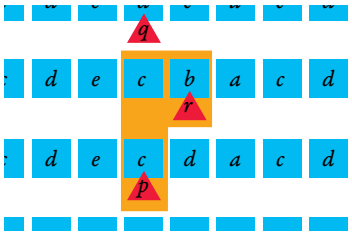
$$\rho, \sigma : \mathbb{N}^2 \to \Gamma + \Gamma \times Q$$

are similar in the sense that for every cells $x, y, z \in \mathbb{N}^2$, the triples

$$(\rho(x), \rho(y), \rho(z)) \qquad (\sigma(x), \sigma(y), \sigma(z))$$

are in the same equivaraint 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 cells at the same time, as in the following picture:



□

We use the above lemma to show that every rejecting computation of the Turing machine can be converted into another rejecting computation, but whose input is linearly dependent, and therefore should be accepted.

Apply Theorem 3.23 yielding a surjective equivariant function

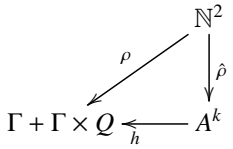
$$h : \mathbb{A}^k \to \Gamma + \Gamma \times Q \qquad \text{for some } k \in \{0, 1, \ldots\}.$$

Let $a_1 \cdots a_n \in \mathbb{A}^n$ be linearly independent atoms, with n sufficiently large, and let ρ be the corresponding computation of our fixed Turing machine. For each cell of the computation choose a k-tuple of atoms which maps to the cell contents via the function h, leading to a set A of atoms such that $h(A^k)$ contains all cells of the computation ρ. Because the Turing machine runs in polynomial time, and k is fixed, we can assume that the size of A is polynomial in the length n of the input. We can assume without loss of generality that A contains the atoms $a_1, \ldots, a_n$ that appear in the input word.

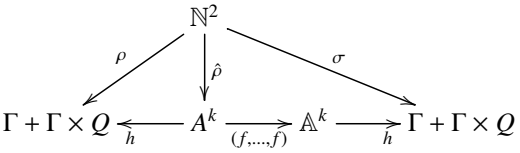
Lemma 10.11. *If n is large enough, there is a linear map $f : \mathbb{A} \to \mathbb{A}$ such that:*

(1) *For every $\bar{b} \in A^{3k}$, the tuples $\bar{b}$ and $f(\bar{b})$ are in the same equivariant orbit.*
(2) *The vectors $f(a_1), \ldots, f(a_n)$ are linearly dependent.*

Before proving the lemma, we use it to complete the proof of the theorem. By choice of A, there is a function $\hat{\rho}$ which makes the following diagram commute



Apply Lemma 10.11, yielding a function f, and define σ to be the composition of the down and right arrows in the following diagram:



By condition (1) in Lemma 10.11, the functions ρ and σ are similar in the sense described by the assumptions of Lemma 10.10, and therefore σ is a rejecting computation of the Turing machine. From condition (2) in Lemma 10.11 it follows that the input of the run σ is linearly dependent, and therefore it should be accepted. This shows that the Turing machine in question does not recognise the language of linearly dependent tuples, and completes the proof of Theorem 10.9.

It remains to prove Lemma 10.11.

Proof of Lemma 10.11 We use the following characterisation of tuples being in the same equivariant orbit for bit vector atoms.

Claim 10.12. *Tuples $\bar{b}, \bar{c} \in \mathbb{A}^m$ are in the same equivariant orbit if and only if*

$$0 = \sum_{i \in I} b_i \quad \textit{iff} \quad 0 = \sum_{i \in I} c_i \qquad \textit{for every } I \subseteq \{1, \ldots, m\}.$$

Proof The left-to-right implication is immediate. For the converse implication, suppose that $\bar{b}$ and $\bar{c}$ satisfy the same equalities as in the statement of the claim. Because every equality can be converted into an equality of the form as in the statement of the claim, and quantifier-free formulas over vocabulary $+$ are Boolean combinations of equalities, it follows that $\bar{b}$ and $\bar{c}$ satisfy the same quantifier-free formulas. The bit vector atoms are homogeneous, as discussed in Section 7.3.2. By Lemma 7.5, in a homogeneous structure the orbit is determined by the quantifier-free theory, and therefore $\bar{b}$ and $\bar{c}$ are in the same orbit. □

The second ingredient in the proof is the following claim, which shows that every tuple of independent vectors can be converted into a tuple of dependent vectors that satisfies almost all of the same equations.

Claim 10.13. *For every $I \subseteq \{1, \ldots, n\}$ there are $b_1, \ldots, b_n \in \mathbb{A}$ such that*

$$0 = \sum_{i \in J} b_i \quad \textit{iff} \quad J = I \vee J = \emptyset \qquad \textit{for every } J \subseteq \{1, \ldots, n\}.$$

Proof Start with all vectors being independent. If I is empty, there is nothing to do, otherwise choose some $i \in I$ and replace b_i with the sum of all vectors with indexes in $I - \{i\}$. □

We now use Claims 10.12 and 10.13 to prove the lemma. By Claim 10.12, in order to prove the lemma, it suffices to find a linear map f so that

$$0 = \sum_{b \in J} b \quad \text{iff} \quad 0 = \sum_{b \in J} f(b) \qquad \text{for every } J \subseteq A \text{ of size at most } 3k \quad (10.1)$$

and the tuples $f(a_1), \ldots, f(a_n)$ are linearly dependent. The left-to-right implication in (10.12) follows from linearity.

Choose a basis $B \subseteq A$, i.e. a set of vectors such that every vector spanned by A is a unique linear combination of basis vectors. Since the vectors $a_1, \ldots, a_n \in A$ are linearly independent, we can choose the basis so that it contains $a_1, \ldots, a_n$. The size of the set A is polynomial in n. Since k is fixed, the number of subsets of size at most $3k$ in A is also polynomial in n . Therefore, if n is large enough, some nonempty subset

$$I \subseteq \{a_1, \ldots, a_n\}$$

has a sum that cannot be obtained by taking a sum of at most $3k$ vectors from A. Apply Claim 10.13 to I viewed as a subset of B, yielding a function $f : B \to \mathbb{A}$ such that

$$0 = \sum_{b \in J} f(b) \quad \text{iff} \quad J = I \qquad \text{for every } J \subseteq B. \qquad (10.2)$$

Extend f to a linear map $f : \mathbb{A} \to \mathbb{A}$. Clearly the vectors $f(a_1), \ldots, f(a_n)$ are linearly dependent, as required in the statement of the lemma, as witnessed by taking the sum ranging over I. The right-to-left implication from (10.1) follows from the conclusion of Claim 10.2 and choice of I. □

Exercises

Exercise 156. Assume the bit vector atoms. Show that if the input alphabet is $\mathbb{A}$, then nondeterministic orbit-finite Turing machines have the same expressiver power as deterministic orbit-finite Turing machines (although with possibly exponential slowdown)

10.3 For equality atoms, Turing machines do not determinise

This section describes another thing that deterministic Turing machines with atoms cannot do. This time, the atoms are the equality atoms.

Theorem 10.14. *Assume that the atoms are* $(\mathbb{N}, =)$*. 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.*

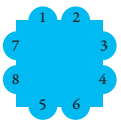
In other words, deterministic orbit-finite Turing machines are not computationally complete[3]. This witnesses the tightness of Theorem 10.4 from Section 10.1, which said that nondeterministic Turing machines are computationally complete when the atoms admit quantifier elimination (as is the case for the equality atoms).

The rest of Section 10.3 is devoted to proving Theorem 10.14.

Recall from Lemma 10.6 that, when the input alphabet is $\mathbb{A}$ – or actually any straight set, as defined in Section 6.2 – then deterministic orbit-finite Turing machines are computationally complete. Therefore, the language in the theorem needs to use an input alphabet that is not straight.

The language in Theorem 10.14 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, because the proof is based on the limited way that a Turing machine can access the atoms in on 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:

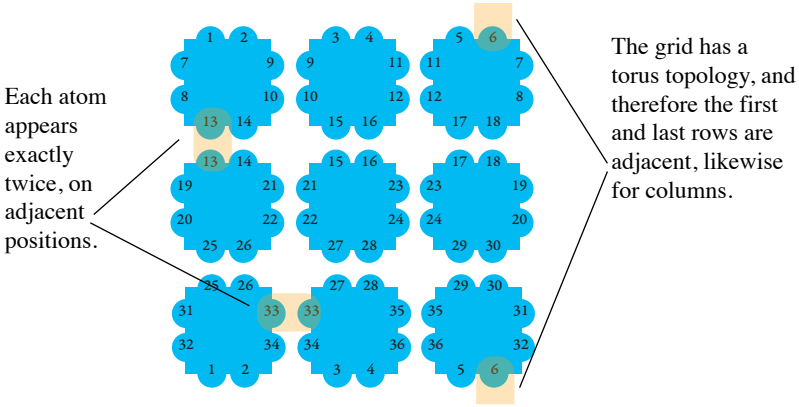
$(1, 2, 3, 4, 5, 6, 7, 8)$ is drawn as 

[3] The results in this section are based on Bojańczyk et al. (2013a)

We will arrange tiles on square grid with torus topology. For $n \in \{1, 2, \ldots\}$, define an $n \times n$ tiling to be a function

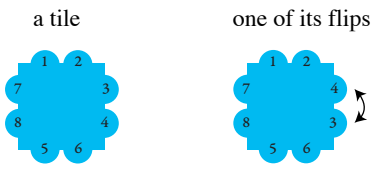
$$\mathcal{T} : n \times n \to \mathbb{A}^{(8)} \qquad \text{where } n \times n \stackrel{\text{def}}{=} \{0, 1, \ldots, n-1\} \times \{0, 1, \ldots, n-1\}.$$

A tiling is called *consistent* if it satisfies the following constraints:

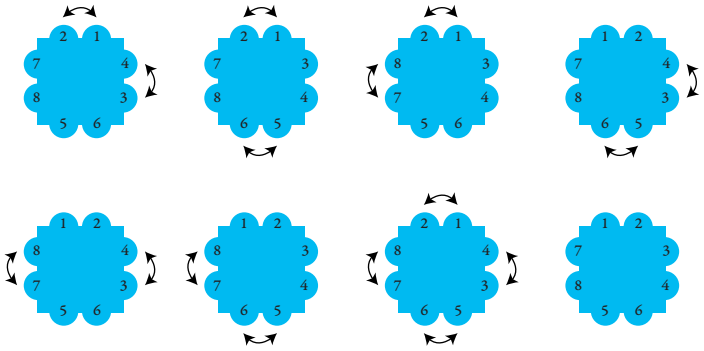


We begin with an informal description of the language that is difficult for deterministic Turing machines. One is given partial information about a tiling, namely each tile is known up to an even number of flips (see below). The question is: can the partial information be instantiated to a tiling that 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 10.15 (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[4] 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 10.14 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 an $\approx$-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 10.14, we will show that a nondeterministic orbit-finite Turing machine can check if an $\approx$-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.

[4] The name stands for Cai, Fürer and Immerman, who first studied this property in Cai et al. (1992).

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 10.15. 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 one of 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 10.17 below, which says that a deterministic Turing machine, unlike the CFI property, is insensitive to certain well chosen flips in an $\approx$-tiling.

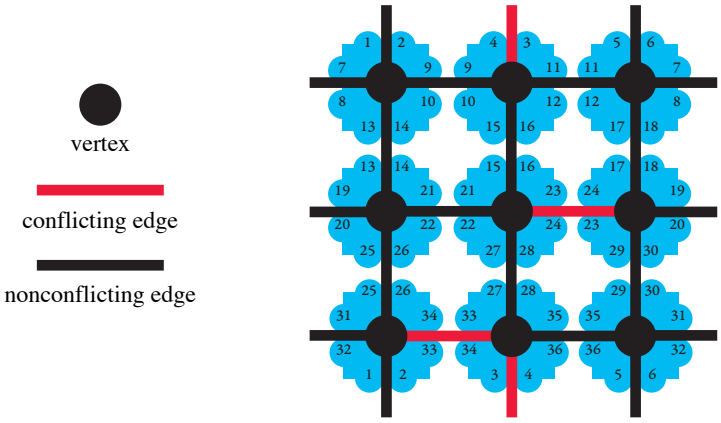
We lift the notion of flips from tiles to their $\approx$-equivalence classes as follows. If τ is a tile, then the *flip* of its $\approx$-equivalence class is defined to the $\approx$-equivalence class which contains some (equivalently, any) flip of τ. It is easy to see that this notion does not depend on the choice of τ in its $\approx$-equivalence class, nor does it depend on the choice of side that 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 10.16. *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:*

$$\mathcal{T}_x(y) \stackrel{\text{def}}{=} \begin{cases} \textit{flip of } \mathcal{T}(y) & \textit{if } y = x; \\ \mathcal{T}(y) & \textit{otherwise.} \end{cases}$$

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, an $\approx$-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 10.2, 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 an $\approx$-tiling, we write $\rho_{\mathcal{T}}$ for the unique computation of the fixed Turing machine on the word representing $\mathcal{T}$.

Lemma 10.17. *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 10.16, the following holds for every $i, j \in \mathbb{N}$:*

$$\rho_{\mathcal{T}}(i, j) = \rho_{\mathcal{T}_x}(i, j) \qquad \text{for all } x \in n \times n \text{ with at most } k^2 \text{ exceptions.} \qquad (*)$$

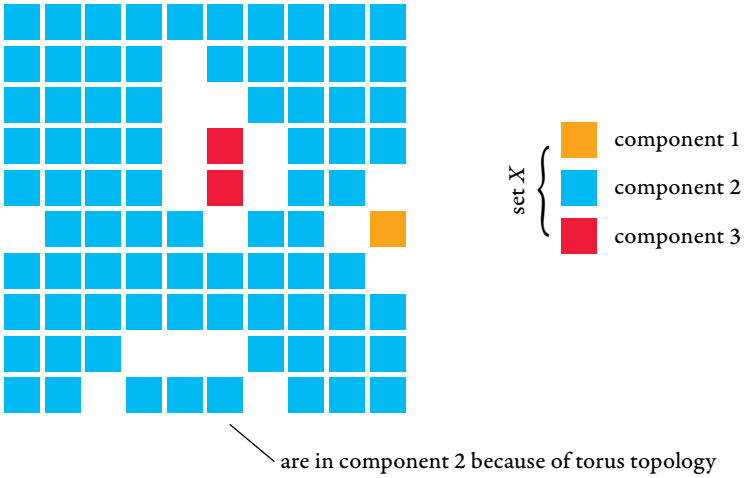
Before proving the lemma, we use it to finish the proof of Theorem 10.14. Take k as in the lemma, and let n be sufficiently large. Let $\mathcal{T}$ be some $n \times n$ $\approx$-tiling which satisfies the CFI property. Consider the computation $\rho_{\mathcal{T}}$, and let (i, j) be the place in the 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 10.16.

Proof of Lemma 10.17. Choose some tuple of atoms which supports the fixed Turing machine, and let k be twice the dimension of this tuple. 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 10.16) 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 (*). Consider the support of the Turing machine that was chosen at the beginning of the proof.

Define

$$Z \subseteq n \times n$$

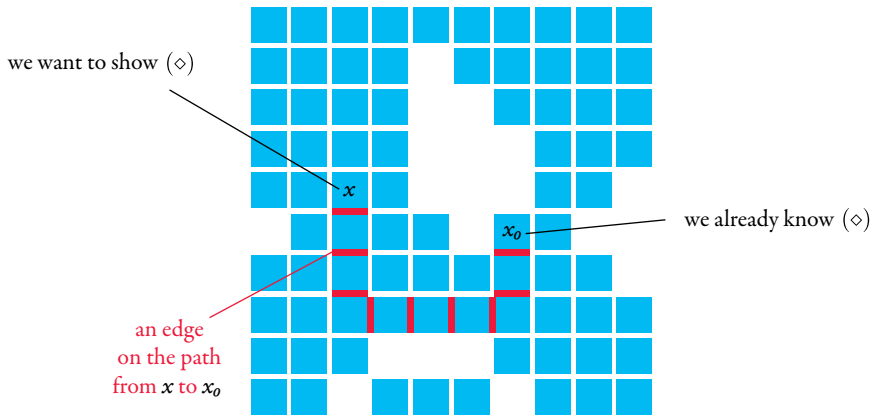
to be the grid positions where $\mathcal{T}$ uses an atom from this support. The set Z has at most k grid positions, since every atom appears in at most two grid positions, and k was chosen to be twice the size of the support of the machine. By a straightforward analysis of connectivity in an $n \times n$ 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 satisfies

$$\rho_{\mathcal{T}}(i, j) = \rho_{\mathcal{T}_x}(i, j). \qquad (\diamond)$$

Using this x_0, we will show that all $x \in X$ also satisfy $(\diamond)$, 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. 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. For each tile except those corresponding to x, x_0, the automorphism flips an even number of sides, and

hence we have:

$$\mathcal{T}_x = \pi(\mathcal{T}_{x_0}). \tag{10.3}$$

The path from x to x_0 was chosen so that it avoids atoms in the support of the Turing machine, and therefore

$$\pi(\rho_{\mathcal{T}}) = \rho_{\pi(\mathcal{T})} \qquad \text{for every input } \mathcal{T} \text{ to the machine.} \tag{10.4}$$

We are now ready to prove that x satisfies ($\diamond$):

$$\begin{aligned}
\rho_{\mathcal{T}_x}(i,j) &= \text{(by (10.3))} \\
\rho_{\pi(\mathcal{T}_{x_0})}(i,j) &= \text{(by (10.4))} \\
\pi(\rho_{\mathcal{T}_{x_0}})(i,j) &= (x_0 \text{ satisfies } (\diamond)) \\
\pi(\rho_{\mathcal{T}})(i,j) &= \text{(by (10.4))} \\
\rho_{\mathcal{T}}(i,j).
\end{aligned}$$

This completes the proof of the lemma, and therefore also of Theorem 10.14. □

Exercise 157. 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 158. In the proof of Theorem 10.14, 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[5].

Exercise 159. Assume the equality atoms and consider the alphabet

$$\{\{\{a,b,c\},\{d,e,f\}\} : a,b,c,d,e,f \text{ are distinct atoms}\}.$$

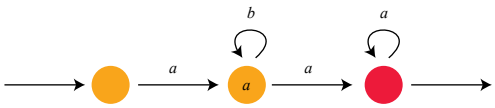
Show that Turing machines over this input alphabet do not determinise.

[5] 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 10.14 can be proved.

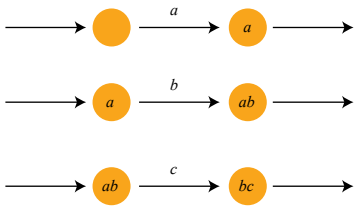
PART FOUR

SOLUTIONS TO THE EXERCISES

Solution to Exercise 1.
Consider language 4, i.e. the first atom appears again. The automaton stores the first atom in its unique register and then waits for a repetition to enter an accepting sink state. Here is the picture:



Consider now language 5, i.e. every three consecutive atoms are pairwise distinct. The automaton uses two registers to store the last two atoms. There is only one control state. Here is the picture:



The unique location is the yellow one shown above, and thus different occurrences of the yellow state should be seen as self-loops. The picture depicts three kinds of self-loops in this unique control state: a self-loop which goes from zero defined registers to one defined register, a self-loop which goes from one defined register to two defined registers, and a self-loop from two defined registers to two defined registers.

Solution to Exercise 2.
The classical construction works. This is most easily seen using semantic equivariance. Let Q be the states of the automaton (recall that a state consists of a location and a register valuation). Consider an automaton which uses ε transitions, i.e. it has two transition relations:

$$\underbrace{\delta_\varepsilon \subseteq Q \times Q}_{\varepsilon\text{-transitions}} \qquad \underbrace{\delta \subseteq Q \times (\Sigma \times \mathbb{A}) \times Q}_{\text{usual transitions}},$$

both of which are semantically equivariant. It is not hard to see that if δ_ε is semantically equivariant, then the same is true for its reflexive transitive closure δ_ε^*. Also, semantically equivariant relations are closed under composition, and therefore $\gamma = \delta \circ \delta_\varepsilon^*$ is semantically equivariant. We replace the original

transition relation by γ. We also replace the final states by those states that can reach a final state via δ_ε^*, the resulting set of final states is also equivariant. The resulting automaton has no ε-transitions, and it recognises the same language as the original one.

Solution to Exercise 3.
The language is $\{abc : a, b, c \in \mathbb{A} \text{ are distinct}\}$. After reading ab, the automaton should be in the same state as after reading ba. This example would go away if automata would have registers that can store unordered pairs of atoms. But then we could consider the following language, where addition is done modulo 3:

$$\{a_0a_1a_2a_ia_{i+1}a_{i+2} : a_0, a_1, a_2 \in \mathbb{A} \text{ are distinct}\}.$$

To have a minimal automaton for the above language, we would need registers that store triples of atoms modulo cyclic permutations. Groups other than $\mathbb{Z}_3$ could also be used. In Section 5.2, we introduce an extension of register automata that does not suffer from the problems described in this exercise.

Solution to Exercise 4.
Consider the language

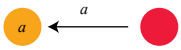
$$\{ab(c^n) : a, b, c \in \mathbb{A} \text{ are distinct and } n \in \mathbb{N}\}$$

One locations and two registers are necessary and sufficient. The automaton begins by loading the first two atoms values into the two registers. Then the automaton loads c into one of the registers, say the first one. However, one needs to make a design decision: should the second register be erased or not? Both choices lead to nonisomorphic automata. This example would go away if we would allow a register automaton to have a different number of registers depending on the location.

Solution to Exercise 5.
Language 2 says that some data value appears twice. After reading sufficiently many letters, a deterministic register automaton will necessarily forget one of the previously read letters, in the sense that it will not be in any register. This letter can be read again. How arguments of this type should be written formalised can be seen in the solution to the next exercise, Exercise 6.

Solution to Exercise 6.
The alphabet is $\mathbb{A}$ and the language, call it L, is "the atom in the last position does not appear on other positions". In other words, this is the reverse of the language 4. (This exercise also shows that nondeterministic automata without

guessing are not closed under reverses.) With guessing, the language L can easily be recognised, by simply reversing all arrows in the automaton for language 4 from Exercise 1. The guessing corresponds to this reversed arrow:



Let us prove that L is not recognised by any nondeterministic automaton without guessing. Toward a contradiction, suppose that L is recognised by an automaton without guessing. Let n be strictly bigger than the number of registers. The word $a_1 \cdots a_{n+1}$ consisting of $n+1$ distinct atoms belongs to the language, and hence must admit an accepting run. Decompose this accepting run as $\sigma \cdot t$ where t is the last transition, which reads the letter a_{n+1}, and σ is the rest of the run, which reads the letters $a_1 \cdots a_n$. Since the automaton is not guessing, none of the state in the run σ contains a_{n+1}. Furthermore, by assumption on n being greater than the number of registers, some $a \in \{a_1, \ldots, a_n\}$ does not appear in the last state of σ. Let π be a permutation of the atoms which swaps a with a_{n+1}. Applying π to σ yields a new run $\pi(\sigma)$ which has the same last state as σ, since the swapped atoms are not present in that state. Therefore $\pi(\sigma) \cdot t$ is also an accepting run, but the word it accepts contains the last letter a_{n+1} twice.

Solution to Exercise 7.
Instead of storing an atom that does not appear in the input before it is erased from the registers, use an undefined register with a special marker stored in the control state.

Solution to Exercise 8.

(1)
- PSPACE membership. A nondeterministic PSPACE algorithm can guess the accepted word. If the automaton has n registers, then data values that are numbers $\{0, \ldots, 2n\}$ are enough.
- PSPACE hardness. The problem is already hard for automata which ignore the input letters in the sense that acceptance for a word is uniquely determined by its length. If the state space is n-tuples of atoms, then an arbitrary vector of $n-1$ bits can be encoded by the pattern in which the coordinates $2, \ldots, n$ are equal to the first coordinate. Therefore, one can think of the state as coding vector of $n-1$ bits, which can be used to store the tape contents of a Turing machine. A quantifier-free formula of size polynomial in n can be used to describe the transitions of the machine.

(2) • NP hardness. We reduce from the following problem: given a formula

$$\varphi(a_1, \ldots, a_n, b_1, \ldots, b_n)$$

which is a Boolean combinations of equalities and inequalities, decide if there is a satisfying assignment where all a_i are pairwise different and all b_i are pairwise different. This is an NP-hard problem, because the pattern of equalities between $\bar{a}$ and $\bar{b}$ can be used to encode an arbitrary vector of n bits (say that bit i is true if and only if the vectors $\bar{a}$ and $\bar{b}$ agree on coordinate i). The above problem is at least as hard as emptiness for register automata, even when there are three orbits of reachable states. Indeed, suppose that the automaton has two locations ℓ_0 and ℓ_1, one initial and final, and three orbits of reachable configurations:

$$\underbrace{\ell_0(\bot, \ldots, \bot)}_{\text{orbit 1}} \qquad \underbrace{\ell_0(\overbrace{a_1, \ldots, a_n}^{\text{distinct data values}})}_{\text{orbit 2}} \qquad \underbrace{\ell_1(\overbrace{b_1, \ldots, b_n}^{\text{distinct data values}})}_{\text{orbit 3}}$$

The formula φ could be used as a guard in a transition that goes from the second orbit to the third orbit.

- NP membership. Consider a graph, where the vertices are orbits of states, and there is an edge from orbit Q_1 to orbit Q_2 if and only there is some transition from some state in Q_1 to some state in Q_2. Because the automaton is equivariant, the following conditions are equivalent

  (i) There exists a state q_1 in orbit Q_1 and a state q_2 in orbit Q_2 such that some transition leads from q_1 to q_2 in one step.

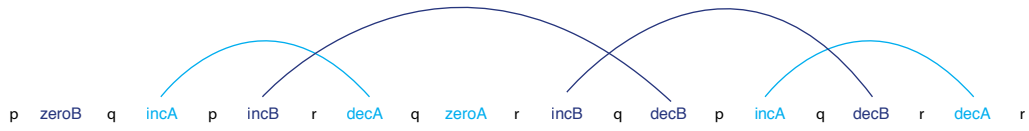
  (ii) For every state q_1 in orbit Q_1 there exists a state q_2 in orbit Q_2 such that some transition leads from q_1 to q_2 in one step.

  It follows that the automaton is nonempty if and only if the graph described above contains a path from some orbit in the initial states to some orbit in the accepting states. Necessarily such a path has length bounded by the number of orbits. By testing quantifier-free formulas for satisfiability, one can test this in NP.

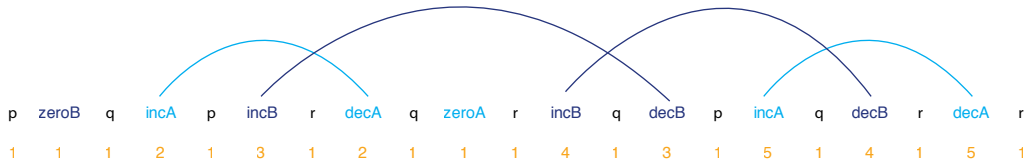
(3) PTIME membership. We do the same argument as in NP membership, only this time the edges of the graph can be computed in polynomial time.

Solution to Exercise 9.
We prove that the (non-)halting problem for Minsky machines reduces to this universality. Recall that a Minsky machine has a finite set of states, two counters storing natural numbers, and a set of transitions which can increment the counters, decrement them and test them for zero. It is an undecidable problem

to decide, given a Minsky machine and two control states p, q, if the machine admits a run that goes from p with both counters empty to q with both counters empty. We can view a run of a Minsky machine as a sequence which alternates between control states and counter operations, in a way consistent with the transition relation, as in the following picture:



The counter operations are valid if for every counter $c \in \{\mathrm{A}, \mathrm{B}\}$, one can pair (the arcs in the picture above) the increments and decrements on counter c such that the increment comes before, and there is no zero test in between. Such a run with a pairing can be encoded as a data word, by adding a unique data value for each arc and using some special data value for positions that are not on arcs (i.e. states or zero tests), as in the following picture:



We claim that a nondeterministic automaton with one register (but with guessing) can recognise the set of data words that are not the encoding of an accepting run with a pairing, and hence undecidability of universality follows in the same way as in Theorem 1.8. The most interesting type of problem is that some arc is wrong: for this the automaton guesses some atom a at the beginning, and checks that this atom is either not used exactly two times, or the first use is not an increment, or the second use is not a decrement of the same counter, or in between there is a test for zero on the appropriate counter.

Solution to Exercise 10.
One can write a formula which is true exactly in the encodings of runs of Turing machines as used in Theorem 1.8. Alternatively, one can write a formula which is true exactly in the encodings of runs of Minsky machines as used in Exercise 9.

Solution to Exercise 11.
Let $\mathcal{A}$ be an alternating register automaton, and define the *dual* of $\mathcal{A}$ to be

the same automaton but where we swap universal locations with existential locations, and we swap accepting locations with nonaccepting locations. We claim that $\mathcal{A}$ accepts a word if and only if its dual rejects.

We prove that for every state q and input data word w, the automaton $\mathcal{A}$ accepts w starting in the bag $\{q\}$ if and only if the dual rejects w starting in $\{q\}$. The proof is by induction on the length of the input. For the induction base of empty inputs, we use the fact that accepting and nonaccepting locations are swapped. Let us do the induction step. Suppose that the input is aw for some letter a and remaining input w. If the state q uses an existential location, then saying that $\mathcal{A}$ accepts aw from q means that there is some transition (q, a, p) such that $\mathcal{A}$ accepts w from p. By the induction assumption, the dual rejects w from p. Since q is universal in the dual, it follows that the dual rejects aw from q, since there is some transition which leads to rejection. The case when q uses a universal state in $\mathcal{A}$ is done the same way.

Solution to Exercise 12.
a

Solution to Exercise 13.
The right-to-left implication is immediate, because infinite antichains and infinite strictly decreasing sequences are both examples of infinite sequences without infinite monotone subsequences. Let us prove the remaining implication, i.e. in a well quasi-order every infinite sequence has an infinite monotone subsequence.

Let $x_1, x_2, \ldots$ be some sequence in a well quasi-order. Consider the set of minimal elements that appear in the sequence. This set must be finite up to equivalence in the quasi-order, since otherwise we would have an infinite antichain. Furthermore, for every element in the sequence there must be some smaller or equal element that is minimal, since otherwise we would have an infinite strictly decreasing sequence. Cut off a finite prefix of the sequence where all minimal elements are found up to equivalence, and reapply the argument, and continue doing this forever. In the limit we get a partition of the sequence into finite factors

$$\underbrace{x_1, \ldots, x_{i_1}}_{\text{factor 1}}, \underbrace{x_{i_1+1}, \ldots, x_{i_2}}_{\text{factor 2}}, \ldots$$

such that every element from outside the first factor is greater or equal to some element that appears in the previous factor. We can view this factorisation as a directed acyclic graph on the indices $\{1, 2, \ldots\}$ which has an edge from i to j if $x_i \leq x_j$ and i, j are in consecutive factors. This directed acyclic graph has finite

degree, because factors are finite, and it has arbitrarily long paths. Therefore it must have an infinite path by König's lemma.

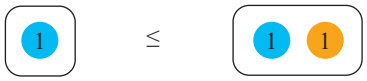
Another solution uses Ramsey's Theorem. Take some infinite sequence $x_1, x_2, \ldots$ and colour each pair $i < j$ with "smaller", "bigger or equal" or "incomparable", depending on the relationship of x_i and x_j. By Ramsey's theorem, there is an infinite subsequence where all pairs get the same colour. This colour has to be "bigger or equal", since the other possibilities would imply an infinite antichain or descending sequence.

Solution to Exercise 14.
Using Exercise 13 it suffices to show that every infinite sequence in $\mathbb{N}^d$ has an infinite monotone subsequence. This is shown by induction on d. The induction base of $d = 1$ is easy to see. For the induction step, consider a sequence

$$x_1, x_2, \ldots \in \mathbb{N}^{d+1}.$$

By the induction assumption, there is an infinite subsequence such that the projection onto the first coordinate is monotone. By induction assumption again, that subsequence has an infinite subsequence where the projection onto the remaining coordinates is monotone, and the result follows.

Solution to Exercise 15.
The counterexample is the bags



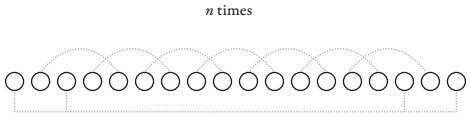
which have profiles that are coordinate-wise incomparable (an equivalence in (1.3) would be recovered if we considered a slightly different order on profiles, but we do not do this, because only the implication (1.3) is needed for our reasoning).

Solution to Exercise 16.
See (Schmitz and Schnoebelen, 2012, Exercise 1.10)

Solution to Exercise 17.
The same proof as without order. The only difference is that order-preserving bijections are used in the definition of the quasi-order, and the Higman order is used to show that this is a well quasi-order. More specifically, when proving the variant of Lemma 1.15, instead of using vectors of natural numbers indexed by subsets of locations, we use sequences of subsets of locations, ordered by the Higman ordering.

Solution to Exercise 18.
Using Theorem 1.16 and the Higman ordering on configurations.

Solution to Exercise 19.
This solution is by Klin and Lasota. Let W be the set of words like the one depicted on the following picture, with circles denoting consecutive data values, and dotted lines denoting equality:



Note that the atom in the first letter is special, because it appears four times, and all other atoms appear two times. We claim that if w and v are words in W of different lengths, then w is not in the same orbit (i.e. equivalent up to atom permutations) as any subsequence of v. In other words, there is no mapping f from positions of w to positions of v which preserves the order and equality on data values. Indeed, such a mapping would have to map the first position to the first position (because the first letter contains the special atom that appears four times), and therefore also the third position to the third position. It follows that the second position must be mapped to the second position, and therefore also the fifth position to the fifth position. Arguing inductively, we see that the i-th position needs to be mapped to the i-th position. In other words, w needs to be mapped to a prefix of v. This cannot be, because, the last position of w is mapped to the last position of v.

Solution to Exercise 20.
Consider the set W of words in the solution to Exercise 19. Let $W_P \subseteq W$ be the subset of words that have a prime number of different atoms. Finally, let L be the upward closure of W_P under the Higman order. We claim that this language is not recognised by a nondeterministic register automaton. Otherwise, such an automaton would need to tell the difference between words from W that have prime and non-prime length. By choosing some non-computable set of numbers instead of the prime numbers, we can get a language that is not computable.

Solution to Exercise 21.
If all positions have distinct atoms, then storing the atom from position i in the register can be seen as storing a pointer to position i. The automaton can

increment such pointers, test them for equality, and it can move its head to a pointer. Using this one can implement simple arithmetic on pointers.

Solution to Exercise 22.
It will be easier to work with a slightly more general model, called *two-way automata with regular lookaround*, where the transitions can ask about regular queries about the sequence of labels to the left (or to the right) of the head. For example, the automaton could empty its register conditionally on the property "the number of b labels to the left of the head is even". From now on, when talking about two-way register automata we assume it has one register, it is nondeterministic, but it is allowed to use regular lookaround.

A configuration of a two-way register automaton is called *local* if the register is either empty or its content is equal to the data value under the head. Call a two-way register automaton local if every change of registers is done only in local configurations (i.e. the automaton can either load the current data value into the register assuming the register was previously empty, or it can empty the register assuming that the register previously stored the data value under the head). One first shows that every two-way register automaton can be made local without affecting the expressive power on data words with pairwise distinct data values. For this, the automaton nondeterministically guesses the last local configuration before emptying the register and does the emptying at that moment.

It remains to prove the exercise for two-way register automata that are local. For a data word

$$\frac{b_1}{a_1}\frac{b_2}{a_2}\cdots\frac{b_n}{a_n} \qquad b_1,\ldots,b_n \in \Sigma \quad a_1,\ldots,a_n \in \mathbb{A}$$

and locations ℓ, ℓ', we say that the automaton admits a (ℓ, ℓ')-loop in position $i \in \{1,\ldots,n\}$ if it can start in position i in the local configuration (ℓ, a_i) and then do a finite number of transitions that do not change the register and lead back to position i in the local configuration (ℓ', a_i). It is not difficult to see that the existence of a (ℓ, ℓ')-loop depends only on the label under the head and regular properties of the labels to the left and right of the head. Therefore, the instead of doing a (ℓ, ℓ')-loop, the automaton could simply do an ϵ-transition conditional on some regular lookaround. After eliminating (ℓ, ℓ')-loops this way, we are left with a two-way automaton which has the property that whenever it loads something into a register, it empties the register in the next step. For such automata, the register is superfluous, and we are left with a two-way automaton without registers, which recognise only regular properties of the labels.

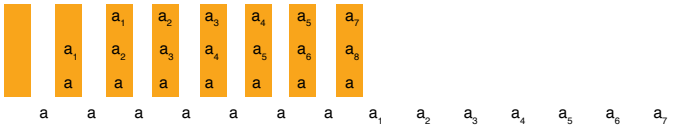
Solution to Exercise 23.
This solution comes from the Master's thesis of Tomasz Wysocki Wysocki (2013). Consider the following language over $\mathbb{A}$:

$$\{a^n a_1 \cdots a_n : n \in \mathbb{N} \text{ and } a, a_1, \ldots, a_n \in \mathbb{A} \text{ are all distinct}\}$$

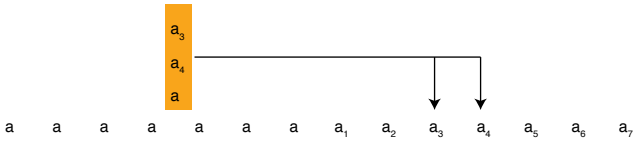
Let us first argue that an alternating register automaton without guessing cannot recognise the language. After reading a prefix of the form a^n, the bag can only have a in its registers. Since there are finitely many possibilities for such bags, there must be some $n < m$ such that the set of reachable bags after reading a^n is the same as the set of reachable bags after reading a^m. Therefore, if the automaton accepts $a^n a_1 \cdots a_n$, then it also accepts $a^m a_1 \cdots a_n$.

Let us recognise this language with guessing. An alternating automaton can easily check that a word is of the form $a^n a_1 \cdots a_m$ for distinct data values $a, a_1, \ldots, a_m$. The challenge is to check that $n = m$. Since languages recognised by alternating automata are closed under intersection, we assume that the input is of the form $a^n a_1 \cdots a_m$.

We only present the main idea using pictures. The automaton has three registers. A main thread of the automaton will read the first n letters, and after reading the i-th letter it will be in a configuration with the initial state and register values a, a_{i-1}, a_i as in the following picture (the orange boxes represent these configurations, with the first two boxes being corner cases):



The contents of the registers are above are guessed, but they are verified using alternation: the initial state is universal, and in each step it spawns off a parallel thread that checks if the current configuration corresponds to two consecutive data values in the future, as in this picture:

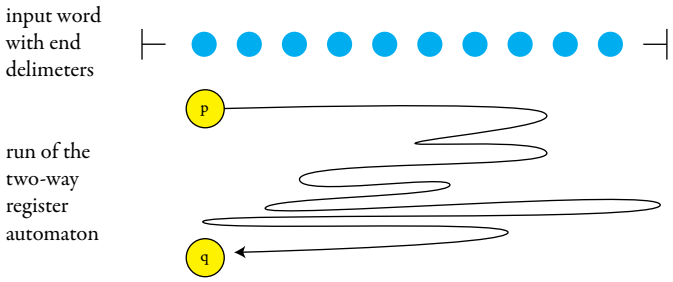


Solution to Exercise 24.
This solution comes from the Master's thesis of Tomasz Wysocki Wysocki

(2013). Consider a two-way nondeterministic automaton $\mathcal{A}$, where the locations are Loc and the registers are R. For two states of this automaton

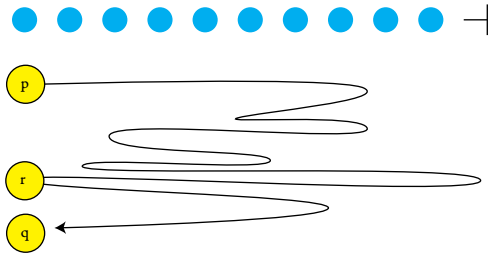
$$p, q \in \mathsf{Loc} \times \text{register valuations}$$

we say that a word admits a (p, q)-loop if there is a run of the automaton which begins in state p, ends in state q, and never tries to move to the left beyond the first position of the word. Here is the picture, note how the run is allowed to revisit the first position or the end delimiter $\dashv$ but it is not allowed to see the start delimiter $\vdash$.



The crucial point is to recognise loops: we will sketch that there is an alternating register automaton, such that if is initialised in a state that stores both p and q, then it accepts if and only if there is a (p, q)-loop. Once loops are recognised, it is not difficult to simulate the two-way automaton (one needs to deal with the initial state and visiting the start marker $\vdash$.) To recognise loops, we observe that a data word admits a (p, q)-loop if and only if one of the following conditions holds:

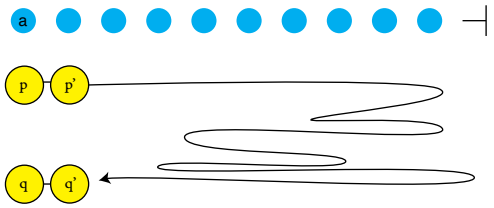
- There is some intermediate state r such that the word admits a (p, r)-loop and an (r, p)-loop, as in this picture.



To check this, the simulating alternating register automaton does an ϵ-transition

where it guesses r and temporarily stores it in the registers. Next it universally branches by into threads for (p, r) and (r, q). Guessing is crucial, because r might contain data values from the future of the word, and ϵ-transitions are used because the two-way automaton might revisit the first position an unbounded number of times.

- The loop does not revisit the first position, as in the following picture:



The simulating alternating register automaton guesses the two configurations p', q', subject to the transition requirement, and advances to the next position.

Solution to Exercise 25.
We want a language that is two-way deterministic, also one-way nondeterministic, but not one-way alternating without guessing. This language is:

$$\{w \in \mathbb{A}^* : \text{some letter appears exactly once}\}.$$

The language is clearly recognised by a one-way nondeterministic automaton, by guessing the letter which appears exactly once. Let us now find a deterministic two-way automaton which does this language. The automaton implements the following procedure:

(1) Put the head on the first letter.
(2) Check if the letter under the head appears exactly once. If yes, accept immediately, otherwise return the head to its previous position (this can be done by a subroutine which first searches to the left for a duplicate, then searches to the right for a duplicate, and returns after finding the first duplicate).
(3) If the head is on the last position, reject, otherwise move the head one step to the right and goto 2.

It remains to show that the language cannot be done by an alternating one-way automaton without guessing. This, honestly speaking, is just a conjecture.

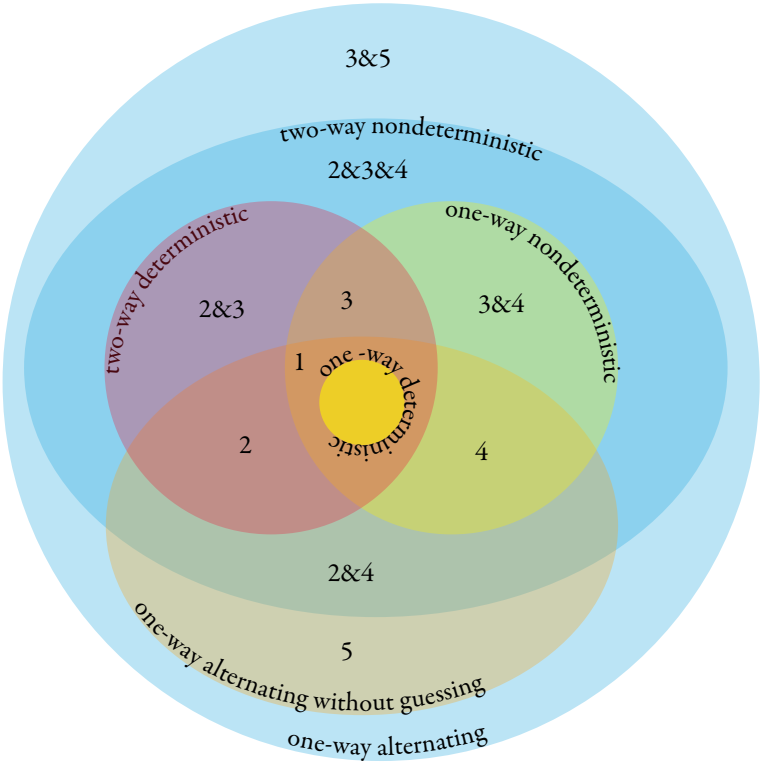
Solution to Exercise 26.
We want a language that is one-way nondeterministic and one-way alternating without guessing, but not two-way deterministic. For this, consider the set of even length sequences of atoms

$$a_1 b_1 \cdots a_n b_n \in \mathbb{A}^*$$

such that there is a path from 1 to n in the graph whose vertices are $\{1, \ldots, n\}$ and where the edge relation contains all pairs $i \to j$ such that $i < j$ and $b_i = a_j$. This language is clearly seen to be recognised by a one-way nondeterministic register automaton without guessing (and therefore also by an alternating one). However, if the language were recognised by a two-way deterministic register automaton, then the language would be in deterministic LogSpace. However, every instance of directed graph reachability can be encoded as a membership question in this language, and therefore we would get that directed graph reachability is in deterministic LogSpace, thus implying that LogSpace can be determinised.

Solution to Exercise 27.
We want a language that is one-way alternating without guessing but which is not two-way nondeterministic. We use the same type of graph problem as in Exercise 26, except that instead of graph reachability we use alternating graph reachability. Since alternating graph reachability is complete for polynomial time, the language cannot be done by a nondeterministic two-way automaton, since otherwise nondeterministic LogSpace would be equal to polynomial time.

Solution to Exercise 28.
Suppose that L is a language that can be done by model A but not B, and K is a language that can be done by model A but not C. Define $L\&K$ to be the concatenation of L and K separated by a fresh symbol. As long as A, B, C are one of the six models in the figure, then the language $L\&K$ can be done by model A but neither by B or C. Using this idea, we can find examples for all coloured areas as in the following picture:



Solution to Exercise 29.

Solution to Exercise 30.
If there was closure under Kleene star, then we would have undecidable emptiness, by finding a data automaton recognising the encodings of computations of Minsky machines used in Exercise 9. Since data automata are closed under intersections, it suffices to find data automaton recognising just one counter with zero tests. If there was closure under Kleene star, then we could check one counter with zero tests: the zero tests can be performed only when the star proceeds to the next iteration.

Solution to Exercise 31.
Suppose that the transitions are

$$\delta_1, \ldots, \delta_n \in \mathbb{Z}^d.$$

There is a run (which can use negative coordinates) that goes from $v \in \mathbb{Z}^d$ to $w \in \mathbb{Z}^d$ if and only

$$v = w + a_1\delta_1 + \cdots + a_n\delta_n \qquad \text{for some } a_1, \ldots, a_n \in \mathbb{N}$$

This is an instance of integer linear programming, and it is known that such instances can be solved in NP. Another answer is that integer linear programming is a special case of Presburger arithmetic, which is decidable.

Solution to Exercise 32.
Languages recognised by data automata are closed under inverse images of the following operations on data words: "remove the first position" and "keep only positions divisible by k". Therefore, we can apply Lemma 2.8 to get the desired result.

Solution to Exercise 33.
Let us use the name *enriched data automaton* for the model from this exercise, and the name *standard data automaton* for the original model. To prove the exercise, we introduce an intermediate model, called a *semi-enriched data automaton*. In semi-enriched model, there is some k such that only the following information is stored about each block of question marks: the exact length if the block has length $\leq k$, and the remainder modulo k if it the block has length $\geq k$. It is not difficult to see that the enriched and semi-enriched models have the same expressive power. To show that the semi-enriched model has the same expressive power as the standard one, we use Exercise 32 and a labelling of every position by its offset from the beginning modulo k.

Solution to Exercise 34.
With the more powerful model from this exercise, one can recognise computations of counter machines as used in Exercise 9. This would contradict Theorem 2.6 that emptiness is decidable for data automata.

Solution to Exercise 35.
A position is called *opening* if it is the first chosen position in its interval, and a *closing* position if it is the last chosen position in its interval. The following lemma characterises the language in the statement of the exercise in terms of a condition that can clearly be recognised by a data automaton. Therefore, to solve the exercise it remains to prove the lemma.

Lemma 2.12. *A data word belongs to the language in the exercise if and only if:*

(1) *every class string satisfies the following expression:*

$$((\underbrace{open}_{\text{opening but not closing}} \quad \overbrace{middle^*}^{\text{remaining cases}} \quad \underbrace{close)}_{\text{closing but not opening}} + \underbrace{clopen}_{\text{opening and closing}} + \underbrace{\bot}_{\text{not chosen}})^*$$

(2) *one can colour the intervals with four colours so that:*

(i) *for every opening position, the previous position with the same data value does not exist or is in an interval with a different colour;*

(ii) *for every closing position, the next position with the same data value does not exist or is in an interval with a different colour.*

Proof We begin with the bottom-up implication. Suppose that conditions 1, 2 hold. We show membership in the language:

- *All chosen positions in the same interval have the same data value.* By induction on the left-to-right order on positions, we prove that every chosen position x has the same data value as all earlier chosen positions in its interval. If x is an opening position, then this statement vacuously true, since there are no earlier chosen positions in the same interval. Assume then that x is chosen but not opening. By condition 1, x cannot be the first position in its class. Let y be the previous position in the class of x. We need to show that y is in the same interval as x. By condition 1, y is a chosen but not closing position. Therefore, there must be a closing position in the interval of y, call it z, which is strictly after y. We cannot have $y < z < x$ since then we could apply the induction assumption to z and show that it is in the same class as x, contradicting the choice of y as the previous position in the class. Therefore $z \geq x$, and thus x is in the same interval as y.
- *There is no non-chosen position which has the same data value as some chosen position in the same interval.* Consider an interval. If the interval has no chosen position, the condition is vacuously true. Otherwise, let d be the data value in the chosen positions, which is unique by the previous item. There cannot be any non-chosen position in the interval with data value d that is before the opening position, since otherwise we would get a contradiction with 2a). A symmetric argument holds for non-chosen positions after the closing position. Between the opening and closing position there cannot be non-chosen positions by condition 1.

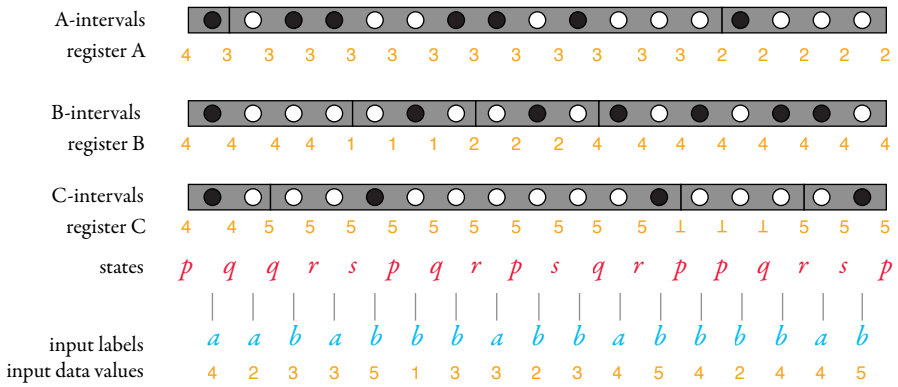
We now show that top-down implication. Condition 1 is easy to see, so we focus on condition 2. We say that two intervals I and J are in conflict if I contains the class predecessor (i.e. previous position in the same class) of the

opening position in J. Condition 2a) says that conflicting intervals have different colours. The key observation is that the conflict relation is a forest, because every interval has at most one opening position, and every opening position has at most one class predecessor. Every forest can be coloured with two colours so that no edge is monochromatic, which shows that two colours are enough to satisfy 2a). A symmetric argument shows that two colours are enough to satisfy 2b), and therefore the product colouring with four colours will satisfy both 2a) and 2b). □

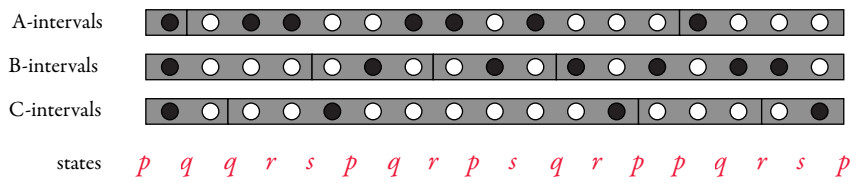
Solution to Exercise 36.
For the sake of this exercise, we consider a model of register automaton where undefined registers are not allowed. The initial configuration has the initial state and the first data value in all the registers. It is not difficult to see that this model is equivalent to the original model of register automata.

Take some nondeterministic register automaton where undefined registers are not allowed, in the sense described above. Without loss of generality, we assume that it is weakly guessing in the sense of Exercise 7. Consider a run of this automaton.

For a register r, an *r-interval* is a maximal connected set of positions in the input word such that every transition in the interval has the same contents of r in its source configuration. Define the *r-chosen* data value of an r-interval, which may be undefined, to be the contents of register r that is used throughout the interval (in the source configurations). Call a position *r-chosen* if the input data value is equal to the r-chosen data value of the containing r-interval. Here is a picture of a run for an automaton with registers {A, B, C} together with the corresponding intervals and their chosen positions.



In the picture above, the chosen positions are marked by black circles and the non-chosen positions are marked by white circles. To describe the run, the data automaton uses nondeterminism to guess this part of the above picture:



Using the solution to Exercise 35, the data automaton checks for each register r that for every r-interval, all r-chosen positions have the same data value, and all non-r-chosen positions have a different data value than the r-chosen ones. Since the automaton is weakly guessing, every r-interval contains some r-chosen position. The above picture is sufficient to reconstruct the entire run including the register contents, in a way which can be checked by the finite states of the transducer in the data automaton.

Solution to Exercise 37.
Immediate.

Solution to Exercise 38.
If the run of the transducer is given explicitly in the data word, and every position is labelled by the state in the run of an automaton recognising the class language, then the correctness of such a labelling can be checked by a formula of the logic.

Solution to Exercise 39.

Solution to Exercise 40.
The automaton checks that the following conditions are all satisfied:

(1) Every data value appears exactly twice.
(2) Let us use the name *middle* for the second appearance of the data value in the first position. Every data value before the middle appears also after or at the middle. Every data value after the middle appears also before the middle.
(3) Regardless of the choices by player $\forall$, the following procedure is bound to terminate by reaching the last position in step (c).
   (i) Player $\forall$ chooses a position x before the middle.
   (ii) Let x' be the position after or at the middle with the same data value as x.

(iii) If x is the last position, then terminate. Otherwise, player $\forall$ chooses some position $y > x'$.
(iv) Let x be the position before the middle with the same data value as y.
(v) Goto (b)

It is not difficult to see that the conditions above are satisfied by every word from the language. For the converse, we prove that if a word does not belong to the language, then items 1 and 2 imply that 3 does not hold. Suppose that 1 and 2 hold, which means that the word is of the form

$$a_1 \cdots a_n a_{\pi(1)} \cdots a_{\pi(n)}.$$

for some distinct data values $a_1, \ldots, a_n$ and some permutation π of $\{1, \ldots, n\}$. In particular, there must be some $i < j$ such that $\pi(i) > \pi(j)$. In step (a), player $\forall$ chooses a_j before the middle, and in step (b) player $\forall$ chooses a_i after the middle.

Solution to Exercise 41.
For undecidability, we could extend the idea from Exercise 40 to recognise use the encoding of Turing machine computations from Theorem 1.8. For the decidability, we use a data automaton. The data automaton guesses for each position what are the states from which this position would be accepted, i.e. from which states would player $\exists$ win if the game started in that position. Then only a local consistency check is needed, in the spirit of Lemma 2.11.

Solution to Exercise 42.
The right-to-left implication is immediate. For the left-to-right implication, assume that $\bar{a}$ is a tuple of atoms that supports x and that π, σ are atom automorphisms that satisfy $\pi(\bar{a}) = \sigma(\bar{b})$. In particular, $\pi^{-1} \circ \sigma$ is an atom automorphism that fixes $\bar{a}$, and therefore

$$(\pi^{-1} \circ \sigma)(x) = x$$

Applying π to both sides of the above equality we get the described

$$\sigma(x) = \pi(x).$$

Solution to Exercise 43.
There are only four equivariant (having empty support) binary relations on atoms, namely the empty and full relations, the equality relation, and the disequality relation:

$$\emptyset \qquad \mathbb{A} \times \mathbb{A} \qquad \{(a, a) : a \in \mathbb{A}\} \qquad \{(a, b) : a \neq b \in \mathbb{A}\}.$$

It suffices to show that if an equivariant relation contains some equality pair (a, a) then it contains all other equality pairs as well, and if it contains some disequality pair (a, b) with $a \neq b$, then it contains all other disequality pairs as well. The reason is that every equality pair can be mapped to every other equality pair by an automorphism of the equality atoms, likewise for disequality pairs. The reader will easily generalise this argument to show that an n-ary relation is equivariant if and only if it can be defined by a quantifier-free formula that uses only equality.

Solution to Exercise 44.
All of the four equivariant relations mentioned in Example 43 are still valid. (In general, when the atoms gain structure, there are more equivariant sets.) However, there are four new binary relations, which refer to the total order, namely:

$$\{(a, b) : a < b\} \qquad \{(a, b) : a \leq b\} \qquad \{(a, b) : a > b\} \qquad \{(a, b) : a \geq b\}.$$

Observe again these are exactly the binary relations that can be defined by quantifier-free formulas.

Solution to Exercise 45.
By unravelling the definition, the commuting diagram says that

$$(\pi(x), \pi(f(x))) \in f \qquad \text{for every } x \in X$$

which is equivalent to

$$(x, f(x)) \in \pi^{-1}(f) \qquad \text{for every } x \in X.$$

Since applying an automorphism, such as π^{-1}, to the function f results in a function, the above is equivalent to saying that the functions f and $\pi^{-1}(f)$ are identical, for every $\bar{a}$-automorphism π. This is the same thing as saying that f is supported by $\bar{a}$.

Solution to Exercise 46.
The vertices are ordered pairs of atoms. From a vertex (a, b) there is exactly one edge, which connects it to (b, a). This graph is bipartite, so it admits a two-colouring. A finitely supported two-colouring, say by colours blue and yellow, would give a choice function, namely map a set $\{a, b\}$ to the unique pair in $\{(a, b), (b, a)\}$ which is coloured by blue.

Solution to Exercise 47.
Suppose that $<$ is a partial order, and a, b are atoms outside the support. Choose π to be the transposition that swaps a and b; in particular π is the identity on the

support of $<$. It follows that $<$ is preserved when π is applied to its arguments, and therefore $a < b$ is equivalent to $a > b$. By antisymmetry, neither property can hold.

Solution to Exercise 48.
Suppose that R is a binary relation on the atoms with finite support. Let c be the smallest atom in the finite support. If $a_1 < b_1$ and $a_2 < b_2$ are atoms which are smaller than c, then R selects the pair (a_1, b_1) if and only if it selects the pair (a_2, b_2), because these pairs can be mapped to each other by an automorphism of the rational numbers that fixes all rational numbers greater or equal to c. It follows that for atoms smaller than c, the order imposed by R is either that of the rational numbers or its opposite, neither of which is well-founded.

This example goes back to Andrzej Mostowski, who was one of the main figures in sets with atoms, which is why they are sometimes called Fraenkel-Mostowski sets. The example shows that in sets with atoms there exist sets which can be totally ordered, but not in a well-founded way.

Solution to Exercise 49.
This exercise might be connected to (Mac Lane and Moerdijk, 1992, Section III.9), but I'm not sure.

We first observe that the choice of enumeration is not important. This is because the topology on bijections does not depend on the enumerations. In other words, the notion of convergent sequence (of bijections) does not depend on the enumeration: a sequence of bijections is convergent if and only if it is pointwise ultimately constant, i.e. for every argument, all but finitely many bijections give the same result.

The equivalence in the exercise says that the following conditions are equivalent:

(1) if a sequence of bijections $\pi_1, \pi_2, \ldots$ of atom automorphisms is pointwise ultimately constant, then the sequence of bijections $f_1, f_2, \ldots$ on X defined by $f_n(x) = \pi_n(x)$ is also pointwise ultimately constant.
(2) every element of X is finitely supported.

For the bottom up implication, suppose that $\pi_1, \pi_2, \ldots$ is pointwise ultimately constant. To show that $f_1, f_2, \ldots$ is pointwise ultimately constant, take some element $x \in X$. By assumption 2, there is some finite atom tuple $\bar{a}$ that supports x. By assumption on $\pi_1, \pi_2, \ldots$ being pointwise ultimately constant, it follows that all but finitely many of the automorphisms $\pi_1, \pi_2, \ldots$ give the same result on the tuple $\bar{a}$. This implies that all but finitely many of the functions $f_1, f_2, \ldots$ give the same result on x.

For the top-down implication, suppose that some $x \in X$ does not have finite support. Let $a_1, a_2, \ldots$ be an enumeration of $\mathbb{A}$. Since x does not have finite support, it follows that for every $n \in \{1, 2, \ldots\}$ there is some atom automorphism π_n which is the identity on $a_1, a_2, \ldots, a_n$ but is not the identity on x. Consider the sequence

$$\pi_1, \mathrm{id}, \pi_2, \mathrm{id}, \pi_3, \mathrm{id}, \ldots.$$

This sequence is pointwise ultimately constant (its limit is the identity). However, if we apply the atom automorphisms from the sequence to x, then on even numbered positions we will get x, and on even numbered positions we will not get x.

Solution to Exercise 50.
Condition 1 (x is a finite union of $\bar{a}$-orbits for some atom tuple $\bar{a}$) is satisfied by $X \subseteq \mathbb{Z}$ if and only if X is finite or $X = \mathbb{Z}$. Condition 2 (x is contained in a finite union of equivariant orbits) is satisfied by all subsets of $\mathbb{Z}$. Condition 3 (for every atom tuple $\bar{a}$ that supports x, x is a finite union of $\bar{a}$-orbits) is satisfied by $X \subseteq \mathbb{Z}$ if and only if X is finite.

Solution to Exercise 51.
Define $S \supseteq R$ to be those pairs which can be obtained by taking some first coordinate of R and pairing it with some second coordinate of R. The set S is obtained from R by taking the product of the projections of R onto the first and second coordinates. Since projection is an equivariant function, it follows from Fact 3.24 that S is orbit-finite. Choose some tuple $\bar{a}$ which supports both R and S. It is easy to see that the transitive closure does not increase the support, and therefore the transitive closure of R is a subset of S that is union of $\bar{a}$-orbits. Since S is orbit-finite, this union must be finite.

Solution to Exercise 52.
Consider the equality atoms. For every finite set of atoms C, the following is a finitely supported function:

$$f_C(a) = \begin{cases} \underline{1} & \text{if } a \in C \\ \underline{2} & \text{otherwise.} \end{cases}$$

When C, D are finite sets of atoms with different sizes, then the functions f_C and f_D are not in the same orbit.

Solution to Exercise 53.

Solution to Exercise 54.
First observe that the assumption that the atoms have finitely many equivariant orbits is necessary to get the converse. As an example, take some infinite structure without any automorphisms, e.g. $\mathbb{A} = (\mathbb{N}, <)$. In this case every $\bar{a}$-orbit is a singleton, regardless of the choice of the atom tuple $\bar{a}$, and therefore conditions 1 and 3 in the statement of the theorem are equivalent.

Let us now prove the statement in the exercise. By induction on $n \in \{1, 2, \ldots\}$ we show that $\mathbb{A}^n$ has finitely many equivariant orbits. The induction base is the assumption from the exercise. Let us now do the induction step, i.e. consider $\mathbb{A}^{n+1}$. Take some n-tuple of atoms $\bar{a}$. By the induction base, there are finitely many $\bar{a}$-orbits in $\mathbb{A}$, which means that there finitely many equivariant orbits in $\mathbb{A}^{n+1}$ that contain tuples which begin with $\bar{a}$. We have just shown that the mapping

$$f : \mathbb{A}^n \to \mathsf{P}(\mathbb{A}^{n+1})$$

which maps a tuple $\mathbb{A}$ to those $\emptyset$-orbits in $\mathbb{A}^{n+1}$ that contain a tuple beginning with $\bar{a}$ always produces finite families of subsets. Since every tuple in $\mathbb{A}^{n+1}$ must begin with some tuple in $\mathbb{A}^n$, it follows that the family of $\emptyset$-orbits in $\mathbb{A}^{n+1}$ is

$$\bigcup_{\bar{a} \in \mathbb{A}^n} f(\bar{a})$$

It is also easy to see that f is equivariant, and therefore its value only depends on the $\emptyset$-orbit of the argument. Therefore, in the union above we could take only one tuple $\bar{a}$ for every $\emptyset$-orbit of $\mathbb{A}^n$, of which there are finitely many by induction assumption. Therefore, the family of $\emptyset$-orbits in $\mathbb{A}^{n+1}$ is finite, as a finite union of finite families. We have shown that $\mathbb{A}^{n+1}$ has finitely many $\emptyset$-orbits. By the assumptions that the two conditions in Theorem 3.16 are equivalent, it follows that $\mathbb{A}^{n+1}$ has finitely many $\bar{a}$-orbits for every tuple of atoms.

Solution to Exercise 55.
Suppose that X is orbit-finite. Choose some support $\bar{b}$ of X. For every tuple of atoms $\bar{a}$, there are finitely many $\bar{a}\bar{b}$-orbits of X. If an element $x \in X$ is supported by $\bar{a}$, then its $\bar{a}\bar{b}$-orbit is a singleton, hence there are finitely many elements of X supported by $\bar{a}$.

Solution to Exercise 56.
Consider the set of all non-repeating tuples of atoms. Since tuples can have arbitrarily large dimensions, and atom automorphisms preserve dimensions of tuples, the set is not orbit-finite. Nevertheless, a given tuple of atoms can only

support finitely many tuples, namely those tuples that are contained in it (and possibly reordered).

Solution to Exercise 57.
We begin with the following observation.

Claim 3.26. *There exists a tuple of atoms $\bar{c}$ which supports R and such that for every $\bar{a} \in \mathbb{A}^k$ there exists a tuple $\bar{b} \in \mathbb{A}^k$ such that $R(\bar{a}\bar{b})$ and every atom in $\bar{b}$ appears in $\bar{a}\bar{c}$.*

Proof Let $\bar{d}$ be some support of R. Choose $\bar{c}$ to be $\bar{d}$ plus $2k$ fresh distinct atoms. The tuple $\bar{c}$ is designed so that for every $\bar{a} \in \mathbb{A}^n$ there are at least k atoms in $\bar{c}$ which are not in $\bar{d}$ and do not appear in $\bar{a}$. By the assumption that $\bar{d}$ supports R and because we are in the equality atoms, membership $\bar{a}\bar{b} \in R$ depends only on the equality type of the tuple $\bar{a}\bar{b}\bar{d}$. Hence one can always choose $\bar{b}$ so that those coordinates which are not from $\bar{a}\bar{d}$ are from $\bar{c}$. □

Let $\bar{c}$ be as in the above claim. Take some $\bar{a} \in \mathbb{A}^n$ and apply the above lemma, yielding a tuple $\bar{b} \in \mathbb{A}^k$. Define

$$f_{\bar{a}} = \{\pi(\bar{a}, \bar{b}) : \pi \text{ is a } \bar{c}\text{-automorphism}\}.$$

The above relation is contained in R and it is a partial function by the assumption that every atom in $\bar{b}$ appears in $\bar{a}\bar{c}$. The domain of $f_{\bar{a}}$ is the $\bar{c}$-orbit of $\bar{a}$. Since $\mathbb{A}^n$ has finitely many $\bar{c}$-orbits, we can take a finite union of partial functions of the form $f_{\bar{a}}$ and get a total function.

Solution to Exercise 58.
Consider the total order atoms, and the relation $a < b$. There is no finitely supported function which maps each atom to a strictly bigger one.

Solution to Exercise 59.
The atoms are the undirected graph which consists of infinitely many triangles. The binary relation is "different but in the same triangle". The function would need to pick, for each atom, one of the two other vertices in the same triangle.

Solution to Exercise 60.
Let E be the family in the exercise. The set E is finite, because $X \times X$ is orbit-finite and therefore has finitely many $\bar{a}$-supported subsets thanks to Exercise 55. Therefore, to prove the exercise it suffices to show that for every two equivalence relations $\sim_1, \sim_2 \in E$ there exists an equivalence relation $\sim \in E$ which is coarser than both $\sim_1$ and $\sim_2$. Consider the following binary relation

on X:

$$R = \sim_1 \circ \sim_2 .$$

This relation is supported by $\bar{a}$. Define $\sim$ to be the transitive closure of R. This is an equivalence relation and it is supported by $\bar{a}$. It suffices to show that $\sim$ has finite equivalence classes. Define $R_n \subseteq X \times X$ to be the set pairs which can be connected by a path of length at most n in the graph (X, R). We know

$$R = R_1 \subseteq R_2 \subseteq R_3 \subseteq \cdots \subseteq X \times X$$

are all subsets of $\bar{a}$ that are supported by $\bar{a}$. By 55 there are finitely many subsets of $X \times X$ that are supported by $\bar{a}$, and therefore there must be some n such that R_n is transitive, i.e. $R_n = \sim$. By the assumption that $\sim_1, \sim_2$ have finite equivalence classes, the graph (X, R) has finite degree, i.e. for each $x \in X$ there are finitely many $y \in X$ such that $R(x, y)$. Therefore, the graph (X, R_n) also has finite degree, which shows that $\sim$ is in E.

Solution to Exercise 61.
Choose some $x \in X$ and some atom tuple $\bar{b}$ which supports x. Consider the set of pairs

$$\{(\pi(\bar{b}), \pi(x)) : \pi \text{ is an } \bar{a}\text{-automorphism.}\}$$

This set of pairs is a surjective function from atom tuples to X, by the assumption that $\bar{b}$ supports x. The domain of the function is the $\bar{a}$-orbit of $\bar{b}$, which is an orbit-finite set. From Fact 3.24 it follows that the range of the function is orbit-finite.

Solution to Exercise 62.
Suppose that X and f are supported by an atom tuple $\bar{a}$. Since orbit-finite sets are clearly closed under finite unions, it suffices to consider the case when X is one $\bar{a}$-orbit. Choose some $x \in X$, and let $\bar{b}$ be an atom tuple which supports $f(x)$. Since $f(x)$ is orbit-finite, it is a union of finitely many $\bar{b}$-orbits, and therefore one can choose $y_1, \ldots, y_n$ so that every element of $f(x)$ is obtained from some y_i by applying some $\bar{b}$-automorphism. It follows that an element belongs to the union in the exercise if and only if it can be obtained by taking some y_i, applying some $\bar{b}$-automorphism, and then applying some $\bar{a}$-automorphism. The result then follows from Exercise 61.

Solution to Exercise 63.
Suppose that X is an orbit-finite set, and $f : X \to X$ is an injective function. It is not difficult to see that if $\bar{a}$ is a support of f, then f maps injectively $\bar{a}$-orbits

to $\bar{a}$-orbits. In particular, since X has finitely many $\bar{a}$-orbits, then the image of f must have the same number of $\bar{a}$-orbits, and is therefore the whole set X.

Solution to Exercise 64.
Before giving the solution, we remark that Dedekind finiteness can be used to characterise orbit-finite sets, but one needs to use the (finitely supported) powerset. The following theorem, which is given here without proof, was shown by Andreas Blass.

Theorem 3.27. *For every choice of atoms, not necessarily oligomorphic ones, a set is all-support orbit-finite if and only if its powerset is Dedekind finite.*

Let us now solve the exercise. Consider the equality atoms and the set

$$\mathbb{A}^{(*)} \quad \stackrel{\text{def}}{=} \quad \bigcup_n \mathbb{A}^{(n)},$$

i.e. the set of non-repeating tuples of arbitrary lengths. This set is not orbit-finite, yet we claim that it is Dedekind finite, i.e. that every finitely supported injection

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

is a bijection. Suppose that f is supported by a finite tuple of atoms $\bar{a}$. For a tuple in $\mathbb{A}^{(*)}$ define its $\bar{a}$-dimension to be the number of atoms in the tuple, not counting the atoms from $\bar{a}$. All tuples in a single $\bar{a}$-orbit have the same $\bar{a}$-dimension, and therefore it makes sense to talk about the $\bar{a}$-dimension of an $\bar{a}$-orbit.

Claim 3.28. *For every $\bar{a}$-orbit Z, the image $f(Z)$ is a $\bar{a}$-orbit with the same $\bar{a}$-dimension.*

Proof The image under f of an $\bar{a}$-orbit in $\mathbb{A}^{(*)}$ is also an $\bar{a}$-orbit. The $\bar{a}$-dimension cannot increase when applying f, since the function is $\bar{a}$-supported, but it cannot decrease as well (since the inverse of f is also $\bar{a}$-supported). □

The key property is that for every $n \in \mathbb{N}$, the set $\mathbb{A}^{(*)}$ has finitely many $\bar{a}$-orbits of $\bar{a}$-dimension n. It follows that for every n, f is a bijection between $\bar{a}$-orbits of $\bar{a}$-dimension n, and therefore f is a bijection.

Solution to Exercise 65.
The left-to-right implication is clear. For the converse implication, if X is not finite, then the family of finite subsets of X is directed but has no maximal elements.

Solution to Exercise 66.
Let us introduce a further condition: (**) there is a maximal element in every set of atoms $\mathcal{X} \subseteq \mathsf{P}X$ which is a chain (i.e. totally ordered by inclusion) and uniformly supported. We will show that orbit-finiteness, (*) and (**) are all equivalent. The implication from (*) to (**) is immediate. For the implication from (**) to orbit-finiteness of X, choose some support $\bar{a}$ of X. If X had infinitely many $\bar{a}$-orbits, then we could construct a uniformly supported infinite chain without a maximal element, by successively adding these orbits. For the implication from orbit-finiteness to (*), suppose that $\mathcal{X}$ is a uniformly supported directed family of subsets of an orbit-finite set X. Let $\bar{a}$ a tuple of atoms that supports every set in $\mathcal{X}$. The union $\bigcup \mathcal{X}$ is a finitely supported subset of X, and therefore must be orbit-finite by oligomorphism. The union partitions into finitely many $\bar{a}$-orbits, call them $X_1, \ldots, X_n$. Every set from $\mathcal{X}$ is simply a union of some of the $\bar{a}$-orbits $X_1, \ldots, X_n$, and therefore $\mathcal{X}$ must contain $X_1 \cup \cdots \cup X_n$, a maximal element.

Solution to Exercise 67.
Let us begin with a counterexample for $(\mathbb{Q}, <)$. The set of all atoms is orbit-finite, but it admits a chain of subsets without a maximal element, namely the family of all downward closed intervals.

We now prove that the statement in the exercise is true in the equality atoms. We will show that (***) is equivalent to (**) from the solution of Exercise 66 and therefore it is equivalent to orbit-finiteness. Actually, we show a stronger property.

Lemma 3.29. *Consider the equality atoms. If a set with atoms $(X, \leq)$ is a total order, then some tuple of atoms supports all elements of X.*

Proof We use the following property of the equality atoms:

(†) Every finite partial automorphism of the atoms can be extended to a complete automorphism that is the identity on almost all atoms.

The above property is not true in $(\mathbb{Q}, <)$ but it true e.g. in the random graph that will be discussed in Section 7.

Let $\bar{a}$ be a support of both X and the total order, which we denote by $\leq$. We show that every element $x \in X$ is supported by $\bar{a}$. Let then π be some $\bar{a}$-automorphism of the atoms. We need to show that $\pi(x) = x$. Let $\bar{b}$ be a finite support of x (eventually we will show that x is supported by $\bar{a}$). Since supports are closed under adding elements, assume that that all atoms in $\bar{a}$ appear also in $\bar{b}$. By property (†), there must be some automorphism of the atoms σ, which agrees with π on $\bar{b}$, but which is the identity on almost all atoms. Since π

and σ agree on the support of x, it follows that $\pi(x) = \sigma(x)$. Also, σ is an $\bar{a}$-automorphism since it agrees with π on $\bar{b}$ which contains all elements of $\bar{a}$.

Since X is supported by $\bar{a}$, it follows that $\sigma(x)$ belongs to X. Since $\leq$ is a total order, x and $\sigma(x)$ must be comparable under $\leq$. Without loss of generality, we assume that

$$x \leq \sigma(x).$$

Since $\leq$ is supported by $\bar{a}$, we can apply the $\bar{a}$-automorphism σ to both sides of the inequality, yielding

$$\sigma(x) \leq \sigma^2(x).$$

By doing this a finite number of times, we get

$$x \leq \sigma(x) \leq \cdots \leq \sigma^n(x)$$

Since σ is the identity on almost all atoms, there must be some n for which σ^n is the identity. Therefore, we see that $x \leq \sigma(x) \leq x$, and therefore $x = \sigma(x)$, which is the same as $\pi(x)$. □

Solution to Exercise 68.
Same proof as for the standard lemma, plus this observation: the depth of a subtree is invariant under applying atom automorphisms.

Solution to Exercise 69.

Solution to Exercise 70.

Solution to Exercise 71.

Solution to Exercise 72.

Solution to Exercise 73.
As explained in the proof of Theorem 3.23, sets which admit a representation as in the statement of the exercise are closed under finite union. Therefore, it is enough to treat the case of a set which is defined by a set expression

$$\{\alpha(\bar{x}) : \text{ for } \bar{x} \in \mathbb{A}^n \text{ such that } \varphi(\bar{x})\}.$$

The function

$$\bar{a} \in \mathbb{A}^n \mapsto \alpha(\bar{a})$$

is the desired bijection, assuming that we restrict its domain to tuples that satisfy φ, and quotient it under the kernel

$$\bar{a} \sim \bar{b} \quad \text{if} \quad \alpha(\bar{a}) = \alpha(\bar{b}),$$

which is defined in first-order logic thanks to the First Symbol Pushing Lemma.

Solution to Exercise 74.
Since every orbit-finite set admits a finitely supported bijection with one that is hereditarily orbit-finite, we can assume without loss of generality that X is hereditarily orbit-finite, and therefore definable. For $\bar{a} = (a_1, \ldots, a_n)$, every $\bar{a}$-orbit that is represented by a set builder expression that uses parameters from $\bar{a}$. There is a well-founded $\bar{a}$-supported total order on such set builder expressions, by viewing each set builder expression as a string over a finite alphabet, with the atom parameters ordered as $a_1 < \cdots < a_n$. The function f uses this order to produce a list of orbits.

Solution to Exercise 75.
The vertices are nonrepeating tuples of atoms, and there is an edge $\bar{a} \to \bar{b}$ whenever $\bar{a}$ is a proper prefix of $\bar{b}$. This graph clearly contains an infinite path, but every such path uses infinitely many atoms, and is therefore not finitely supported.

Solution to Exercise 76.

- *Not equivalent.* The atoms are $(\mathbb{Q}, <)$. Consider the graph where the vertices are atoms, and the edge relation is $\{(v, w) : v < w\}$. Take T to be all vertices, and s to be any vertex. There exists an infinite path from s which sees T infinitely often – actually this is true for every infinite path – but there is no cycle in the graph.
- *Equivalent.* The atoms are the equality atoms $(\mathbb{N}, =)$. Let (V, E) be such that there is an infinite path that begins in s and visits $T \subseteq V$ infinitely often. Let $\bar{a}$ be a tuple of atoms that supports the graph and the target set T. By the pigeon-hole principle, the infinite path must contain two vertices t, t' that are in the same $\bar{a}$-orbit, i.e.

$$\pi(t) = t' \qquad \text{for some } \bar{a}\text{-automorphism } \pi.$$

Suppose that t' is visited first by the infinite path, i.e.

$$t \xrightarrow{p} t' \qquad \text{for some finite path } p \text{ in the graph.}$$

It follows that for every n, there is a path from t to $\pi^n(t)$, namely

$$t \xrightarrow{p} \pi(t) \xrightarrow{\pi(p)} \pi^2(t) \xrightarrow{\pi^3(p)} \cdots \xrightarrow{\pi^{n-1}(p)} \pi^n(t).$$

By the same argument as in the solution to Exercise 67, we may assume that π is the identity on all but finitely many atoms, and therefore $\pi^n(t) = t$ for some n, thus showing that there is a cycle containing t.

Solution to Exercise 77.

(1) *Infinite path that sees T infinitely often.* Suppose that $\bar{a}$ supports the graph. We claim that condition (1), i.e. existence of an infinite path that begins in s and visits T infinitely often, is equivalent to:

(*) there exist paths $s \to t \to t'$ such that t and t' are in the same $\bar{a}$-orbit.

The left-to-right implication follows from the pigeon-hole principle, while for the right-to-left implication we use the path

$$t \to \pi(t) \to \pi^2(t) \to \cdots .$$

It remains to decide if (*) holds. The binary relation "in the same $\bar{a}$-orbit" is a finitely supported relation on V. Therefore, the question in the definition of (*) can be formalised using the set structure. (There is a hole in this argument, namely that it assumes that we can compute the relation "in the same $\bar{a}$-orbit". This is actually an assumption that needs to be made about the atom structure, see the footnote for Exercise 99.)

(2) *One can reach a finite cycle that intersects T.* Condition (2) is checked using the set structure from the Third Symbol Pushing Lemma.

Solution to Exercise 78.
Define V_0 to be T. For $n > 0$ define $V_n \subseteq V$ to be V_{n-1} plus all vertices $v \in V$ such that:

- v is owned by 0 and some $(v, w) \in E$ satisfies $w \in V_n$;
- v is owned by 1 and all $(v, w) \in E$ satisfy $w \in V_n$.

Using the set structure, one shows that each V_n is a hereditarily definable set that can be computed. By the same argument as for graph reachability, there is some fixpoint, i.e. some n such that $V_{n+1} = V_n$. If the source vertex is in this fixpoint, then player 0 wins the game. We claim that the converse implication is also true, thus completing the algorithm.

To prove this claim, suppose that the source vertex belongs to the complement of the fixpoint, call this complement W. By definition, if $v \in W$ then

- v is owned by 0 then all $(v, w) \in E$ satisfy $w \in W$;
- v is owned by 1 then some $(v, w) \in E$ satisfies $w \in W$.

It follows that player 1 has a strategy that ensures staying in the complement W, and thus never reaching the set T.

Solution to Exercise 79.
First observe that binary relations represented by set builder expressions are not closed under taking transitive closure. As an example, consider the successor relation $\{(n, n+1) : n \in \mathbb{N}\}$. This relation is clearly defined by a set builder expression, but its transitive closure is the order relation, which is not.

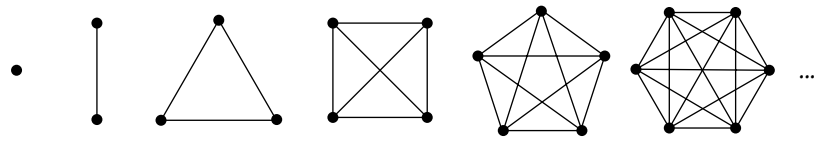
To get the undecidability result, one considers a graph where vertices are configurations of a Minsky machine, and the edge relation is one step of computation.

Solution to Exercise 80.
The atoms are

$$\mathbb{A} = (\mathbb{N}, P_1(x), P_2(x), \ldots) \qquad \text{where } P_i(x) = x < i.$$

Define $\mathbb{A}_n$ to be the structure with the same universe, but where only the unary predicates $P_1, \ldots, P_n$ are allowed. The structure $\mathbb{A}$ is not oligomorphic, but the structures $\mathbb{A}_1, \mathbb{A}_2, \ldots$ are oligomorphic. Every set builder expression over $\mathbb{A}$ is already a set builder expression over $\mathbb{A}_n$ for some finite n. For such set builder expressions, the graph reachability algorithm terminates in finite time.

Solution to Exercise 81.
Let $\mathbb{A}$ be the undirected graph (which is modelled a structure where the universe is the vertices and there is one binary symmetric relation), which is a union of all finite cliques (one clique for each size):



This structure is not oligomorphic, since an automorphism can only map each clique to a permutation of itself. Nevertheless, we show below that graph reachability is decidable for graphs represented by set builder expressions over these atoms.

For $k \in \{1, 2, \ldots, \omega\}$ define $\mathbb{A}_k$ be the structure obtained from $\mathbb{A}$ by adding for each $i < k$ a unary predicate P_i which selects elements of the clique of size i.

Claim 5.5. *Every first-order formula over $\mathbb{A}$ of quantifier-rank r is equivalent to a quantifier-free formula over $\mathbb{A}_r$.*

Proof Induction on r □

Claim 5.6. *If R is a binary relation on $\mathbb{A}^k$ which is first-order definable, then there is some n such that the transitive reflexive closure R^* is equal to $R^{\leq n}$.*

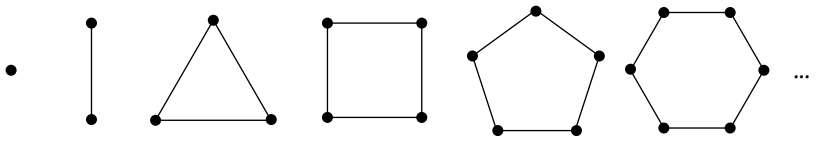
Proof By Claim 5.5, the relation R is defined by a quantifier-free formula over $\mathbb{A}_r$ for some r. Consider a path in the graph where the vertices are $\mathbb{A}^k$ and the edge relation is given by R:

$$\bar{x}_1 \xrightarrow{R} \bar{x}_2 \xrightarrow{R} \cdots \xrightarrow{R} \bar{x}_n.$$

If n is sufficiently large with respect to r, one can find $i < j$ such that the quantifier-free types in $\mathbb{A}_r$ are the same for the $2k$-tuples $\bar{x}_i\bar{x}_{i+1}$ and $\bar{x}_i\bar{x}_{j+1}$. We can then shorten the path, by going directly from $\bar{x}_i$ to $\bar{x}_{j+1}$. □

A corollary of the above claim is that the fixpoint algorithm for graph reachability terminates in finitely many steps, for graphs where the vertices are tuples of atoms. By Exercise 73, any the vertices of any graph represented by a set builder expression can be viewed as tuples of atoms modulo a first-order definable partial equivalence, and therefore the fixpoint algorithm terminates in finitely many steps also for graphs represented by set builder expressions.

An extension of this argument would work for the structure which is the union of all finite cycles:



Solution to Exercise 82.
The usual proof works. It is important that transitive closure preserves orbit-finiteness.

Solution to Exercise 83.
Determinism can be formalised in first-order logic and then checked using the Symbol Pushing Lemmas. To check if an automaton is unambiguous we construct the product of the automaton with itself, and check if there is a pair (p, q) of state $p \neq q$ that is both reachable and co-reachable.

Solution to Exercise 84.
Consider a nondeterministic orbit-finite automaton $\mathcal{A}$ which recognises a language L supported by $\bar{a}$. Define a new automaton, which is a disjoint union of all automata of the form $\pi(\mathcal{A})$ where π is a $\bar{a}$-automorphism. This new automaton is orbit-finite and supported by $\bar{a}$. Finally, the recognised language is the same, because all automata in the disjoint union recognise the same language, namely the original language L.

Solution to Exercise 85.
The construction from Exercise 84 does not work, but we can apply the Myhill-Nerode Theorem. Since the syntactic automaton is obtained only from the language, it has the same support as the language, by the equivariance principle.

Solution to Exercise 86.
Instead of natural numbers, we could use the positive rational numbers, and the answer to emptiness would be the same. This is because a run that uses positive rational numbers can be changed into a run that uses natural numbers, by scaling. After assuming that the counters store positive rational numbers, we end up with a special case of nondeterministic orbit-finite automata, over the total order atoms. (The automaton is not equivariant, since it uses the constant 0.) As we shall prove later on, emptiness for such automata is decidable.

Solution to Exercise 87.

Suppose that we have a union

$$\bigcup_{i \in I} L_i$$

where I is an orbit-finite set and each L_i is recognised by an nondeterministic orbit-finite automaton with state space Q_i. Then the union is recognised by an automaton with state space

$$\biguplus_{i \in I} Q_i$$

which is an orbit-finite set by Exercise 62.

Solution to Exercise 88.
The language of words $w \in \mathbb{A}^*$ where all atoms are distinct, is an orbit-finite intersection

$$\bigcap_{a} \quad \text{atom } a \text{ appears at most once.}$$

The language of representations of accepting runs of Turing machines, as described in the proof of Theorem 1.8, is also seen to be an orbit-finite intersection of languages recognised by orbit-finite deterministic automata.

Solution to Exercise 89.
Intuitively speaking, the problem is that intersection corresponds to product on automata, and we cannot do orbit-finite products. Here is the counterexample. For every $a \in \mathbb{A}$, the language "a appears at most once" is recognised by a (deterministic) orbit-finite automaton. If we could intersect all these languages, then we would get a nondeterministic automaton for the language "all letters are distinct". By Theorem 5.11, this would mean that "all letters are distinct" could be recognised by a register automaton, which is not the case.

Solution to Exercise 90.
The problem is that when the automaton reads the first letter of the input, say the unordered set $\{\underline{1}, \underline{2}\}$, then it cannot load any atoms into its registers, since this would require a form of choice.

Solution to Exercise 91.
Let M be the syntactic monoid of a language. There is a deterministic automaton with states M which recognises the same language; the transition function is simply defined by $(q, a) \mapsto qa$ where qa is the product operation in the syntactic monoid.

The failure of the converse implication is witnessed by the language

$$\{w \in \mathbb{A}^* : \text{the first letter appears also later in the word}\}.$$

The language is recognised by an deterministic automaton which keeps the first letter in its state, and is hence orbit-finite. To see that the syntactic monoid is not orbit-finite, we observe that if two words w, v have different sets of atoms that appear in them, then they are not equivalent with respect to the two-sided Myhill Nerode equivalence relation. Indeed, if a is an atom that appears in w but not in v, the aw is in the language but av is not. Therefore, the syntactic monoid must store the set of distinct atoms in a given word, which cannot be done in an orbit-finite way.

Solution to Exercise 92.

Solution to Exercise 93.

Solution to Exercise 94.
Assume that the set of states Q in the syntactic automaton is orbit-finite. By Exercise 92, the syntactic monoid consists of state transformations of states in the syntactic automaton. Therefore, saying that the syntactic monoid is aperiodic amounts to showing that every input word w satisfies

$$\exists k \in \{0, 1, \ldots\}\ \forall q \in Q \qquad qw^k = qw^{k+1}. \tag{5.1}$$

The exercise asks if the last two quantifiers can be swapped in the above, i.e. if the above condition is equivalent to

$$\forall q \in Q\ \exists k \in \{0, 1, \ldots\} \qquad qw^k = qw^{k+1}. \tag{5.2}$$

Clearly (5.1) implies (5.2), so we only show the converse implication. Choose m so that for every state q, there is a tuple of at most m atoms which supports w, q and the transition function of the syntactic automaton. The value of m depends on w. The function

$$\bar{a} \in \mathbb{A}^m \quad \mapsto \quad \text{number of } \bar{a}\text{-orbits in } Q$$

is a finitely supported function from tuples of atoms to natural numbers, and therefore it has finitely many possible values. It follows that there is some $k \in \{0, 1, \ldots\}$ such that for every tuple $\bar{a}$ of at most k atoms, there are at most k $\bar{a}$-orbits in Q. Let $q \in Q$, and choose some tuple $\bar{a}$ which supports w, q and the syntactic automaton; this tuple has at most m atoms. All elements in the set

$$\{q, qw, qw^2, \ldots\}$$

are supported by $\bar{a}$, and therefore each of the elements of the above set is a singleton $\bar{a}$-orbit in Q. It follows that the above set has size at most k, which proves (5.1).

Solution to Exercise 95.
Yes. Consider the monoid

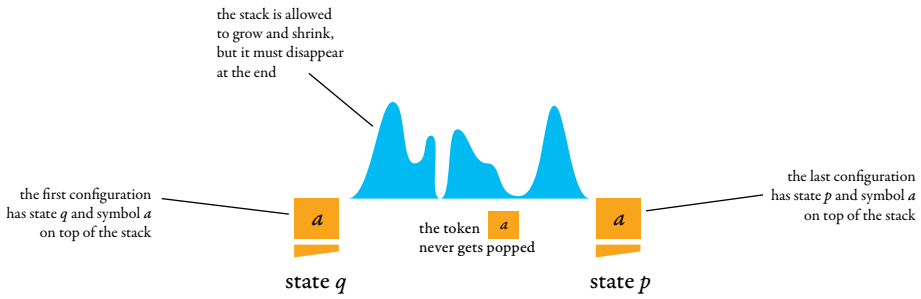
$$M = 1 + \mathbb{A}^2$$

where 1 is the identity and product for non-identity elements is defined by $ab = b$. Removing any finite set of non-identity elements still yields a monoid, and hence one can obtain an infinite chain as in the statement of the exercise.

Solution to Exercise 96.
Using the usual construction, one can convert the automaton into one which operates on the stack only via push and pop, i.e. apart from the fresh transitions of the type in the statement of the exercise, we only allow transitions of the form:

$q \xrightarrow{\text{read}(a)} p$	read input letter $a \in \Sigma$ and do not change the stack
$q \xrightarrow{\text{push}(a)} p$	read nothing and push symbol a on the stack
$q \xrightarrow{\text{push}(a)} p$	read nothing and pop symbol a from the stack

For states q, p and a stack symbol a, we write $q \overset{a}{\Rightarrow} b$ if the automaton has a run of the following form:



The following claim implies decidability, because it shows that the relation $q \overset{a}{\Rightarrow} b$ can be computed, and hence one can check if there is a run which eventually pops the initial stack symbol.

Claim 5.21. *Suppose that $\bar{c}$ is some tuple of atoms which supports the transition relation. The relation $q \overset{a}{\Rightarrow} b$ is generated by the following rules:*

(1) *for every stack symbol a, the binary relation $\overset{a}{\Rightarrow}$ is transitive and reflexive.*
(2) *for every $q, p, q', p', \in Q, b \in \Sigma, a, a' \in \Gamma$ we have*

$$q \overset{read(b)}{\to} p \quad \text{implies} \quad q \overset{a}{\Rightarrow} b$$

$$q \overset{fresh(b)}{\to} p \quad \text{implies} \quad q \overset{a}{\Rightarrow} b \qquad \text{if } b \text{ is fresh with respect to } q, a, \bar{c}$$

$$q \overset{push(a')}{\to} q' \overset{a'}{\Rightarrow} p' \overset{pop(a')}{\to} p \quad \text{implies} \quad q \overset{a}{\Rightarrow} b$$

Proof The proof has two parts: soundness (the relation $q \overset{a}{\Rightarrow} b$ satisfies the rules) and completeness (the relation $q \overset{a}{\Rightarrow} b$ is the least one which satisfies the rules). Completeness of the rules is shown as usual for pushdown automata (by induction on the length of the run). Soundness needs a little care because of the rule for freshness. Here the observation is that we can always map the stack to some stack which is fresh with respect to b, by using a $\bar{c}$-automorphism which fixes the state q and the topmost stack symbol a. Such an operation is admissible, because reachable configurations are closed under applying $\bar{c}$-automorphisms. □

Solution to Exercise 97.
Before giving the solution, we point out that without atoms, emptiness is decidable for higher order pushdown automata, even for orders ≥ 3. For undecidability it suffices to have a stack of at most two stacks. We assume that ϵ-transitions are available, which changes the expressive power of the model, but does not influence decidability of emptiness.

We only show that such an automaton can recognise

$$L = \{(w\#)^n : w \in \mathbb{A}^* \text{ has no repetitions and } n \in \mathbb{N}\}$$

over the alphabet $\mathbb{A} \cup \{\#\}$. The same construction can be modified so that the automaton checks that consecutive blocks between # symbols, instead of being equal as in L, are consecutive configurations of a Turing machine.

In a first phase, the automaton puts w into the (first) stack and checks that it has no repetitions. This is done as follows. For every new letter a, the automaton stores a in its state. Then it duplicates the stack, and searches if a appears on the duplicated stack, destroying the duplicate in the process. If it does not find a on the duplicated stack, it pushes a onto the first stack, and proceeds to the next input letter.

Once it has checked that w has no repetitions, and stored w on the stack, the automaton proceeds to the second phase, which checks that the rest of the input consists of copies of w separated by # symbols. The second phase is done

essentially the same way as the first. For every two consecutive letters a and b in the rest of the input the automaton does the following.

If $a = \#$ then b must be the first letter of w, which is stored in the state. If $b = \#$, then a must be the last letter of w, which is stored in the state. Finally, suppose that neither a nor b are #. The automaton needs to check that a and b are consecutive letters in w. To do this, the automaton duplicates the stack, and searches through this stack to check that a and b are consecutive symbols on the stack.

Maybe the above undecidability argument shows that our definition of higher-order pushdown automata for atoms is the wrong one. If it is wrong, then which one is right?

Solution to Exercise 98.
The language is odd length palindromes where the first letter is equal to the middle letter. If it were generated by an orbit-finite context-free grammar with finitely many terminals (but possibly an orbit-finite set of rules), then the language would have the following property for some tuple of atoms $\bar{a}$ (the support of the hypothetical grammar), which it does not have:

For every sufficiently long w, there is a decomposition $w = w_1 w_2 w_3$, with w_2 and $w_1 w_3$ nonempty such that

$$w_1(\pi \cdot w_2)w_3$$

is a palindrome for every $\bar{a}$-automorphism π.

Solution to Exercise 99.
Let $\mathbb{A}$ be a structure that is effectively oligomorphic and has decidable first-order theory. Our goal is to extend the vocabulary of the structure with constants $c_1, c_2, \ldots$ such that the the structure $(\mathbb{A}, c_1, c_2, \ldots)$ has decidable first-order theory, and every element of the universe is represented by some constant.

For $k \in \{0, 1, \ldots\}$ we say that a tuple of atoms $a_1 \ldots a_n$ is k-saturated if it is non-repeating, and every k-tuple of atoms is in the same equivariant orbit as some tuple of the form

$$a_{n_1} a_{n_2} \cdots a_{n_k} \qquad \text{for some } n_1, \ldots, n_k \in \{1, \ldots, k\}.$$

If $k = 0$, then the condition is trivially satisfied. If $\mathbb{A}$ is oligomorphic, then every k-saturated tuple can be extended to one which is $(k+1)$-saturated. It follows that there is an infinite sequence of atoms $a_1, a_2, \ldots$ which is ω-saturated in the following sense: for every k, some finite prefix $a_1, \ldots, a_n$ is a k-saturated tuple of atoms.

Claim 5.29. *If a sequence of atoms $a_1, a_2, \ldots$ is ω-saturated then the structure $\mathbb{A}$ is isomorphic to its substructure induced by $\{a_1, a_2, \ldots\}$.*

Proof One shows that Duplicator can win the infinite round Ehrenfeucht-Fraïssé game between $\mathbb{A}$ and the induced substructure, which implies isomorphism. □

Using the assumption that the formulas for the "same orbit" equivalence relation can be computed, it follows that there is some infinite ω-saturated sequence of atoms which is computable in the following sense: for every n, one can compute a first-order formula $\varphi_n(x_1, \ldots, x_n)$ which defines the equivariant orbit of the first n elements in the infinite sequence. Fix some ω-saturated and computable infinite sequence of atoms $a_1, a_2, \ldots$. Let $\mathbb{B}$ be the substructure of $\mathbb{A}$ induced by this sequence, extended with constants $c_1, c_2, \ldots$ representing the atoms $a_1, a_2, \ldots$. By the computable assumption, $\mathbb{B}$ has decidable first-order theory, and by the claim it is isomorphic to $\mathbb{A}$.

Solution to Exercise 100.
Clearly 1 implies 2. Let us show that 2 implies 1. By assumption 2, we can compute the number k of $\emptyset$-orbits of $\mathbb{A}^n$. By Lemma 4.11, each such orbit has a different first-order theory. Therefore, it suffices to find k inequivalent formulas with n free variables, these formulas can be found using brute force.

It is not difficult to see that 1 implies 3: if we can axiomatise each orbit by a formula, then being in the same orbit boils down to satisfying the same axiomatising formula, for which there are finitely many possibilities.

Let us show that 3 implies 2. We want to count the number of $\emptyset$-orbits in $\mathbb{A}^n$. Consider the following procedure. Let $A \subseteq \mathbb{A}^n$ be a finite set of tuples of atoms, which are in pairwise different orbits. Initially, A is empty. Using assumption 3 and decidable model checking, we can decide if there exists a tuple $\bar{a} \in \mathbb{A}^n$ which is in a different orbit than all tuples in A. If there exists no such tuple, then we have found the number of orbits. Otherwise, we can find such a tuple, by enumerating through all possible candidates. We add this tuple to A and continue. The algorithm is bound to stop because of oligomorphism.

Solution to Exercise 101.
The difficulty is that the memoryless determinacy theorem uses choice, and produces strategies that are not necessarily finitely supported. In fact, one can give an example of a Büchi (even reachability) game where player 0:

- has a winning strategy that is not finitely supported;
- does not have a finitely supported winning strategy.

A solution to this difficulty is to consider nondeterministic strategies. Define a *memoryless nondeterministic strategy* for player $i \in \{0, 1\}$ to be a set of pairs

$$S_i \subseteq (V_i \times V) \cap E$$

such that if a vertex owned by player i has at least one outgoing edge in E, then it also has at least one outgoing edge in S_i. We say that S_i is *winning* for player i if every path that starts in the source vertex s and uses only edges from S_i will necessarily see T infinitely often. We claim that if player i has a winning memoryless strategy (not necessarily finitely supported) in a Büchi game, then he has a winning memoryless nondeterministic strategy, which is supported by whatever supports the game. There are finitely many such strategies; these can then be enumerated and checked

(todo complete)

Solution to Exercise 102.
A reduction from the tiling problem.

Solution to Exercise 103.
Still undecidable. We can view a configuration of a Minsky machine as a natural number

$$2^a 3^b 5^c$$

where a, b are the values of the counters and c is the number of the control state. One can write a Presburger formula $\varphi(x, y)$ which holds if and only if y represents the successor of the configuration represented by x. The system of equations says that: (a) the variables that represent the source and target states have different values; (b) variables that represent consecutive configurations have the same value. This system has a solution if and only if the source configuration cannot reach the target configuration.

Solution to Exercise 104.
The first set is the atoms. The second set is pairs of atoms, modulo the equivalence relation defined by

$$(a_1, a_2) \sim (b_1, b_2) \qquad \text{if} \qquad \underbrace{a_2 - a_1 = b_2 - b_1}_{b_1 + a_2 = a_1 + b_2}.$$

Let c be some atom. It is not hard to see that the function

$$a \in \mathbb{A} \qquad \mapsto \qquad \text{equivalence class of } (a, c)$$

is a c-supported bijection between the two sets. We now establish that there is

no equivariant bijection. Toward a contradiction, suppose that f is an equivariant bijection. For an atom $a \in \mathbb{A}$, let (b, c) be an element of the equivalence class $f(a)$. It is not hard to see that for every atom d, the function

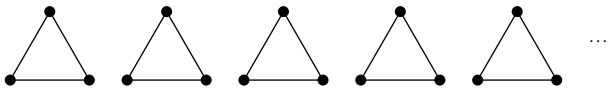
$$a \quad \mapsto \quad a + d$$

is an automorphism of the atoms. Since equivariant functions commute with automorphisms, it would follow that $(b + d, c + d)$ belongs to the equivalence class $f(a + d)$. However,

$$(b, d) \sim (b + d, c + d),$$

contradicting injectivity of f.

Solution to Exercise 105.
The atoms are the undirected graph which is an infinite union of triangles:

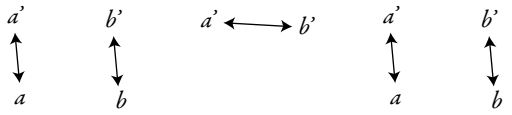


Consider a triangle which involves atoms a, b, c. This triangle is supported by a, and it is also supported by b (or c). Nevertheless, the atom a does not support b, because one can swap b and c while fixing a.

Solution to Exercise 106.
Before proving the exercise, we observe that it gives an alternative proof of the Least Support Theorem. To prove the Least Support Theorem, it suffices to show that finite sets of atoms supporting x are closed under intersection. Suppose then that x is supported by S and also supported by T. By the exercise, if an atom automorphism π fixes $S \cap T$, then it can be decomposed as a finite compositions of atom automorphisms that fix either S or T. Since such automorphisms fix x, it follows that π also fixes x. It remains to prove the exercise.

We do this in several steps.

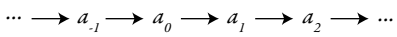
(1) *Transpositions.* Suppose first that π from the assumption of the lemma is a transposition, i.e. it swaps two atoms $a, b \notin S \cap T$. Choose atoms $a', b' \notin S \cup T$. Swapping a, b is the same as performing the following sequence of transpositions:



By the assumption that a, b are not in $S \cap T$ and a', b' are not in $S \cup T$, each of the above transpositions fixes either S or T.

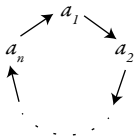
(2) *Finite permutations.* An automorphism (i.e. permutation) of the atoms is called finite if it moves finitely many atoms. Every finite permutation is a finite composition of transpositions, and thus the previous item implies that the conclusion of the lemma is also true when π is a finite permutation.

(3) *Infinite cycles.* Suppose that π is an infinite cycle, as in the following picture:



We do not assume that the cycle contains all atoms. Since $S \cup T$ is finite, up to renumbering we can assume that there is some n such that elements from $S \cup T$ can appear only in $\{a_2, \ldots, a_n\}$. If we compose π with the transposition

then we get the permutation consisting of two cycles (one finite, one infinite) as in the following picture:



The permutations drawn in blue fix $S \cup T$. Therefore, we have shown that the infinite cycle π is a composition of two permutations that fix $S \cup T$, and one finite cycle. To the finite cycle we can apply the previous item.

(4) *General case.* Every permutation can be decomposed into independent cycles, some finite and some infinite. Both types of cycles were dealt with in the previous items. We only need to apply the construction to the finitely many cycles that contain atoms from $S \cup T$.

Solution to Exercise 107.
Clearly anything that supports all the sets $X_1, \ldots, X_n$ will also support X, which proves the inclusion

$$\sup(X) \subseteq \sup(X_1) \cup \cdots \cup \sup(X_n).$$

For the converse inclusion, we observe that the notions of least support, and the partition of a set with respect to its least support can all be defined using the language of set theory, and therefore the functions

$$X \mapsto \sup(X) \qquad X \mapsto \{X_1, \ldots, X_n\} \qquad X \mapsto \bigcup_i \sup(X_i)$$

can all be defied using the language of set theory. In particular, by the equivariance principle, these functions are equivariant. Since the last function is equivariant, anything that supports X, e.g. its least support, will also support $\bigcup_i \sup(X_i)$. Therefore,

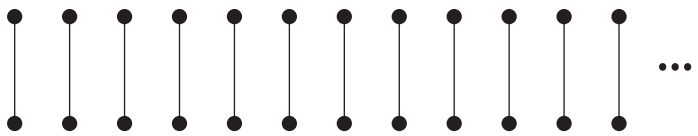
$$\sup(X) \quad \text{supports} \quad \sup(X_1) \cup \cdots \cup \sup(X_n).$$

Since both sides of the above are finite sets of atoms, and for finite sets of atoms "supporting" is the same as "containing", we get the inclusion

$$\sup(X) \supseteq \sup(X_1) \cup \cdots \cup \sup(X_n).$$

Solution to Exercise 108.
See (Bojańczyk et al., 2014, Corollary 9.5).

Solution to Exercise 109.
Suppose that the atoms are a graph with infinitely many edges that do not share any nodes.



This structure is oligomorphic, actually it is homogeneous (see Section 7). Every atom is supported by itself, or the other side of its edge.

Solution to Exercise 110.
This exercise is based on (Colcombet et al., 2015, Lemma 2.14). Consider the least support of the multiplication operation in the group. This least support also supports the universe of the group, and the inverse operation $g \mapsto g^{-1}$. For

an element g of the group, define $[g]$ to be the set of atoms that are in the least support of g but are not in the least support of the multiplication operation of the group. If a set of atoms supports g, h and the multiplication operation, then it also supports the product gh. It follows that

$$[gh] \subseteq [g] \cup [h]. \tag{6.4}$$

For the same reasons, we have

$$[g^{-1}] = [g]. \tag{6.5}$$

Take some g in the group which maximises the size $[g]$. Such a maximum exists, since the size of $[g]$ depends on that $\bar{a}$-orbit of g, of which there are finitely many. Since we are dealing with the equality atoms, we can choose an atom automorphism π so that

$$\pi([g]) \cap [g] = \emptyset. \tag{6.6}$$

We have

$$g = \pi(g)\pi(g)^{-1}g.$$

Combining this with (6.4), we get

$$[g] \subseteq [\pi(g)] \cup [\pi(g)^{-1}g]$$

Combining this with (6.6), we get

$$[g] \subseteq [\pi(g)^{-1}g]$$

By maximality of $[g]$ the above is actually an equality, i.e.

$$[g] \subseteq [\pi(g)^{-1}g] \tag{6.7}$$

The same proof also yields

$$[\pi(g)] \subseteq [g^{-1}\pi(g)] \tag{6.8}$$

Using a similar reasoning applied to

$$\pi(g)^{-1} = g^{-1}\pi(g)\pi(g)^{-1}$$

we conclude that

$$[\pi(g)] \overset{(6.8)}{\subseteq} [g^{-1}\pi(g)] \overset{(6.5)}{=} [\pi(g)^{-1}g] \overset{(6.7)}{=} [g].$$

From (6.6) it follows that $[\pi(g)]$ is empty. Therefore $[g]$ must also be empty, since $[\_]$ commutes with $\bar{a}$-automorphisms. By maximality of $[g]$ it follows that all elements of the group have value $\emptyset$ under $[\_]$ which implies that all elements of the group are supported by $\bar{a}$. In an orbit-finite set there can only

be finitely many elements with a given support (Exercise 55). Therefore the group is finite. The same proof would work for some other atoms, e.g. $(\mathbb{Q}, <)$.

Solution to Exercise 111.
No. The bit vector atoms are oligomorphic, but the atoms themselves are a group.

Solution to Exercise 112.
Using the same ideas as in Theorem 6.3, we get the following result.

Claim 6.5. *Let X be an orbit-finite set. There exists k and a finitely supported surjective function $g : \mathbb{A}^k \to X$ such that for every $x \in X$ there is some tuple in $g^{-1}(x)$ only uses atoms from the least support of x.*

Take g and k as in the above claim. Take $\bar{c}$ to be some tuple of atoms which supports both g and f. We will show that for every $\bar{a} \in \mathbb{A}^{n \cdot k}$ there is some $\bar{c}$-supported partial function f' which satisfies the commuting diagram in the statement of the picture when its domain is restricted to the $\bar{c}$-orbit of $\bar{a}$. By putting these functions together we get the desired result.

Let then $\bar{a} \in \mathbb{A}^{n \cdot k}$. Consider

$$x = f \circ (g, \ldots, g)(\bar{a}).$$

The element x is supported by $\bar{a}\bar{c}$, because the value of a function is supported by any tuple which supports both the function and its arguments. Therefore the least support of x uses only atoms that appear in $\bar{a}\bar{c}$. By Claim 6.5, there is some $\bar{b} \in \mathbb{A}^k$ which uses only atoms from $\bar{a}\bar{c}$ and such that $g(\bar{b}) = x$. Consider the relation

$$f' \stackrel{\text{def}}{=} \{\pi(\bar{a}, \bar{b}) : \pi \text{ is a } \bar{c}\text{-automorphism}\}$$

By choice of $\bar{b}$ this relation is a partial function from $\mathbb{A}^{n \cdot k}$ to $\mathbb{A}^k$ and it satisfies the commuting diagram in the exercise when restricted to its domain.

Solution to Exercise 113.

Solution to Exercise 114.
Let $\bar{a}$ be a support of F. Since the choice function f can be defined separately for each $\bar{a}$-orbit, we can assume without loss of generality that X is a single $\bar{a}$-orbit. By definition of straight sets, this means that (up to finitely supported bijections) X is a single $\bar{a}$-orbit in $\mathbb{A}^{(k)}$ for some k.

Claim 6.11. *There is some tuple of atoms $\bar{c}$ such that for every $x \in X$, there is some $y \in F(x)$ which is supported by the atoms in x plus $\bar{c}$.*

Proof For $x \in X$ define $n \in \{0, 1, 2, \ldots\}$ to be the minimal number n such that some element of $F(x)$ has support of size n. The number does not depend on the choice of x, since X is contained in one equivariant orbit. Let $\bar{c}$ be any tuple of $k + n$ distinct atoms that do not appear in $\bar{a}$. Take some $x \in X$ and choose some $y \in F(x)$ with support of size at most n. Let $b_1, \ldots, b_i$ be the atoms in the least support of y which are not in the least support of x, there are at most n of these. Since $\bar{c}$ has $k + n$ atoms, one can find atoms $c_1, \ldots, c_i$ in the tuple $\bar{c}$ which do not appear in $x \in \mathbb{A}^{(k)}$. The atom automorphism π that swaps $(b_1, \ldots, b_i)$ with $(c_1, \ldots, c_i)$ fixes the supports of both x and F. Therefore, $\pi(y) \in F(x)$. The least support of $\pi(y)$ uses only atoms from $\bar{c}$ and the least support of x. □

By Exercise 74, there is a finitely supported function which maps every $x \in X$ to an ordered list of the $x\bar{c}$-orbits that are contained in $F(x)$. By the above claim, one of these orbits is a singleton, and the function f can simply output the unique element of that singleton (the first singleton in the list).

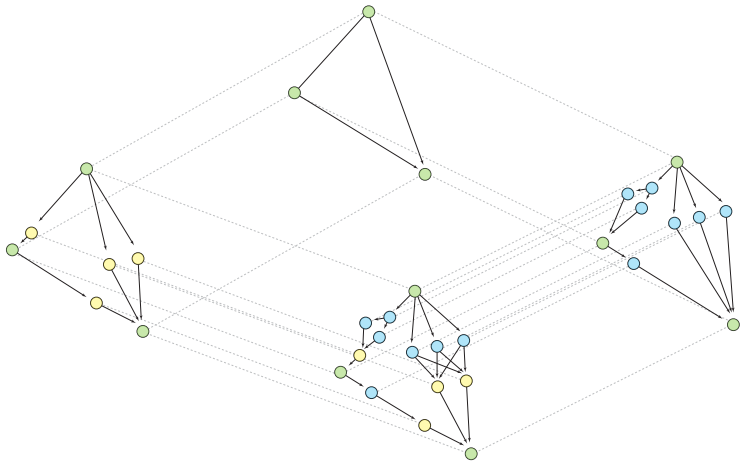
Solution to Exercise 115.
Take F to be the function which maps an atom $a \in \mathbb{A}$ to all strictly bigger atoms.

Solution to Exercise 116.

Solution to Exercise 117.

$$(a, b) \in \mathbb{A}^2 \qquad \mapsto \qquad \begin{cases} (a, b) & \text{if } a \neq \underline{1} \\ (b, a) & \text{otherwise.} \end{cases}$$

Solution to Exercise 118.
Here is an example of an instance and solution of amalgamation:



One way of amalgamating two partial orders, which is illustrated in the picture above, is to put the elements of the left (yellow) order after the elements of the right (blue) order, as long as they have the same relationship with the common (green) elements. Other strategies lead to other amalgamations.

Solution to Exercise 119.
No.

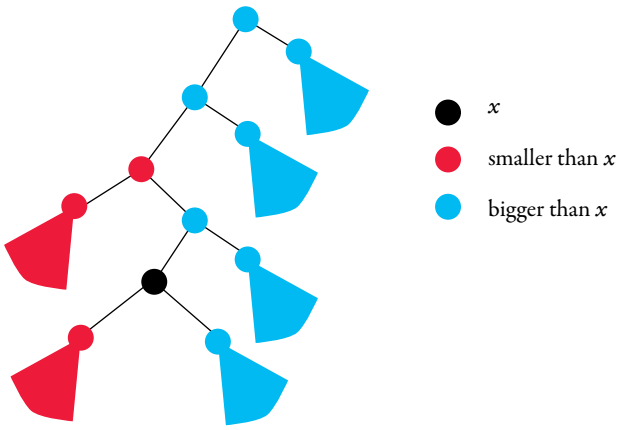
Solution to Exercise 120.
The family of structures with equality only (i.e. an empty vocabulary) that have size at most 2.

Solution to Exercise 121.
In a homogeneous structure, two tuples are in the same orbit if they satisfy the same quantifier-free formulas. By the assumption that the vocabulary is relational (i.e. has no function symbols) and finite, up to logical equivalence there are finitely many quantifier-free formulas over a given set of variables, and they can be computed. By the assumption on $\mathscr{A}$ having decidable membership, one can decide which quantifier-free formulas are satisfiable in the Fraïssé limit $\mathbb{A}$. Furthermore, one can effectively eliminate quantifiers, i.e. for every first-order formula (possibly with free variables) over the vocabulary of $\mathbb{A}$, one can compute an equivalent one which is quantifier-free. Using this observation, it follows that the first-order theory of $\mathbb{A}$ is decidable. Furthermore, $\mathbb{A}$ is effectively oligomorphic, in the sense of Exercise 99, since the "same orbit" formula is the quantifier-free formula which checks that the same predicates from the finite

vocabulary are satisfied. Therefore, $\mathbb{A}$ satisfies the assumptions of Exercise 99 and is thus an effective structure.

Solution to Exercise 122.
By going through the proof of Theorem 7.10.

Solution to Exercise 123.
A rational number can be viewed as node in Rabin's tree $\{0,1\}^*$ as follows



Solution to Exercise 124.
See (Bojańczyk et al., 2013b, Proposition 2).

Solution to Exercise 125.
No. The vertices might come in the wrong order.

Solution to Exercise 126.
Properties X for which L_X is recognizable by an automaton include "contains a clique of size three", "is not a clique", "contains a vertex connected to all other vertices" but do not include the complementary properties "does not contain a clique of size three" or "is a clique". A sufficient condition is definability by a formula of first-order logic with a quantifier prefix $\exists^*\forall$. Is this condition necessary?

Solution to Exercise 127.
Toward a contradiction, suppose that $\leq$ is a total order on the random graph

that is supported by a tuple $\bar{a} = (a_1, \ldots, a_n)$. Choose some atoms b, c which are isolated in the subgraph induced by $\{a_, \ldots, a_n, b, c\}$. It follows that

$$(a_1, \ldots, a_n, b, c) \quad \mapsto \quad (a_1, \ldots, a_n, c, b)$$

is a partial automorphism of the random graph. By homogeneity, it extends to a $\bar{a}$-automorphism, which preserves the total order, but swaps b with c

Solution to Exercise 128.
Both conditions (a) and (b) in the definition of path decompositions are problematic. Let us focus on condition (a), i.e. that for every atom, the positions where it appears is an interval. To prove that a nondeterministic automaton cannot check this condition, one uses the same proof as in Exercise 10.1.

Solution to Exercise 129.
The truth value of an MSO formula $\varphi(x_1, \ldots, x_n)$ depends only on the orbit of the free variables. The random directed graph is homogeneous and without functions, and therefore by Exercise 121 one can decide if a first-order formula with free variables has at least one satisfying assignment. Since the tuples which satisfy φ form an equivariant set, this set is definable in first-order logic by Theorem 4.13. If the translation to first-order logic were computable, then one would be able to decide the MSO theory of the random directed graph. Since the random directed graph contains all finite directed graphs as induced subgraphs, e.g. all directed grids, it has undecidable MSO theory.

Solution to Exercise 130.

Solution to Exercise 131.
Linear independence can be expressed in terms of addition, and therefore all automorphisms of the bit vector atoms are also automorphisms of $\mathbb{B}$. For the converse inclusion, we observe that

$$a + b = c$$

is equivalent to:

(1) $a = c$ and $b = 0$; or
(2) $b = c$ and $a = 0$; or
(3) all of a, b, c are distinct and nonzero, and the tuple abc is linearly dependent.

Solution to Exercise 132.
To see that $\mathbb{B}$ is not homogeneous, consider the four vectors:

$$100 \quad 010 \quad 001 \quad 111.$$

Every three out of four are linearly independent, but all four are linearly independent. Therefore, mapping the above four vectors to some four independent vectors is a partial automorphism which does not extend to a full automorphism.

We now show that the structure becomes homogeneous if we add independent predicates of all arities.

(todo complete)

Solution to Exercise 133.
Yes. Its Fraïssé limit is the countably dimension vector space over the rationals. This limit is not oligomorphic. The pairs

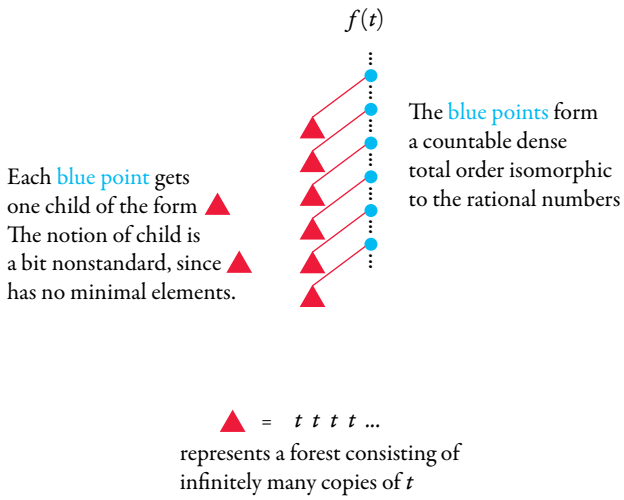
$$(1,1) \quad (1,2) \quad (1,3) \quad (1,4) \quad \cdots$$

are in different orbits.

Solution to Exercise 134.
Fix some atom a. Define the equivalence relation to be $b \sim c$ if the closest common ancestor of b, c is a proper descendant of a.

Solution to Exercise 135.

Solution to Exercise 136.
We will show how this universal tree can be interpreted in the complete binary tree using MSO. To prove this, consider a transformation f which inputs a tree t (i.e. a structure with the closest common ancestor function where for each element, the ancestors are a totally ordered) and outputs the tree depicted in the following picture:



Define t_∞ to be a limit of this procedure, i.e. a tree satisfying

$$t_\infty \quad \text{is isomorphic to} \quad f(t_\infty).$$

We will show that t_∞ is the universal tree. To show this, we need to show that it is a) homogeneous; and b) it contains every tree as an induced substructure. Both properties are not difficult to show. Using the idea from Exercise 123 and the recursive nature of Rabin's tree, one can show that the structure $(\mathbb{Q}^*, \preceq)$ consisting of sequences of rational numbers ordered lexicographically has decidable MSO theory. The tree t_∞ can be described in terms of $(\mathbb{Q}^*, \preceq)$, with nodes being coded as odd length sequences of rational numbers (i.e. every second level of $(\mathbb{Q}^*, \preceq)$ is used), and the descendant relation defined using an MSO formula $\varphi(x, y)$ which uses the definition of the tree t_∞ in terms of the function f. This implies that t_∞ has decidable MSO theory, since it can be interpreted inside a structure with decidable MSO theory.

Solution to Exercise 137.
Essentially rearranging parentheses in a set builder expression. Using the Definable Relation Lemma, compute set builder expressions which represent the

projections of R onto the first and second coordinates, respectively:

$$\bigcup_{i\in I}\{\alpha_i(\bar{x}_i) : \text{ for } \bar{x}_i \in \mathbb{A}^{n_i} \text{ such that } \varphi_i(\bar{x}_i)\}$$
$$\bigcup_{j\in J}\{\beta_j(\bar{y}_j) : \text{ for } \bar{y}_j \in \mathbb{A}^{k_j} \text{ such that } \psi_i(\bar{y}_j)\}.$$

Using the Symbol Pushing Lemma, compute for each $i \in I$ and $j \in J$ a formula of first-order logic θ_{ij} such that

$$\theta_{ij}(\bar{x}_i\bar{y}_j) \qquad \text{iff} \qquad (\alpha_i(\bar{x}_i),\beta_j(\bar{y}_i)) \in R.$$

The expression for S is now

$$\bigcup_{i\in I}\{(\alpha_i(\bar{x}_i),\hat{\beta}_i(\bar{x}_i) : \text{ for } n_i \in \mathbb{A}^{\bar{x}_i} \text{ such that } \varphi_i(\bar{x}_i)\}$$

where

$$\hat{\beta}_i(\bar{x}_i) = \bigcup_{j\in J}\{\beta_j(\bar{y}_j) : \text{ for } \bar{y}_j \in \mathbb{A}^{k_j} \text{ such that } \theta_{ij}(\bar{x}_i\bar{y}_j)\}.$$

Solution to Exercise 138.

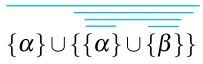
Solution to Exercise 139.

Solution to Exercise 140.

Solution to Exercise 141.
Formally speaking, a pair is defined using Kuratowski pairing:

$$(\alpha,\beta) \stackrel{\text{def}}{=} \{\alpha\} \cup \{\{\alpha\} \cup \{\beta\}\}$$

Apart from the subexpressions of α and β, the Kuratowski pair has five subexpressions as indicated in the following picture:



It follows that the size of a k-tuple satisfies

$$|(\alpha_1,\ldots,\alpha_k)| \leq |\alpha_1| + \ldots + |\alpha_k| + O(k).$$

Note that Kuratowski pairing introduces no new variables, apart from the ones used by its components. For example, the dimension of the set

$$\mathbb{A}^k = \{(a_1, \ldots, a_k) : \text{ for } a_1, \ldots, a_k \in \mathbb{A}\}$$

is k, while its size is linear in k.

Solution to Exercise 142.

(1) The bit vector atoms do not have fixed dimension polynomial orbit count. Let $\bar{a}$ be a tuple of n linearly independent atoms. Then the number of $\bar{a}$-orbits in $\mathbb{A}$ is 2^n, because each atom can be a linear combination of any nonempty subset of the atoms $\bar{a}$ – or be linearly independent from all of them.

Solution to Exercise 143.

Solution to Exercise 144.
Fix some variable names $x_1, \ldots, x_k$. Define P to be the set of atomic formulas, i.e. formulas Define an *atomic formula* to be a relation from the vocabulary of the atoms applied to arguments which are either variables $x_1, \ldots, x_k$ of constants from the tuple $\bar{a}$. The number of atomic formulas is

$$\sum_R (k+n)^{\text{arity of } R},$$

where R ranges over relations in the vocabulary of the atoms.

Solution to Exercise 145.

Solution to Exercise 146.

Solution to Exercise 147.

Solution to Exercise 148.
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

$$\{(x_1, \ldots, x_k) : \text{ for } x_1, \ldots, x_k \in \mathbb{A} \text{ such that } \varphi(x_1, \ldots, x_k)\}$$

where φ is the formula which says that all of its arguments are either $\underline{1}$ or $\underline{2}$.

Solution to Exercise 149.
Consider $X = \mathbb{A}^k$. The natural set builder expression has size $O(k)$, while the size $|X|$ is not polynomial in k.

Solution to Exercise 150.
For distinct atoms $a_1, \ldots, a_n$, consider the powerset of $\{a_1, \ldots, a_n\}$. This set has dimension 0, since it is finite. Its semantic size is 2^n, while the number of equivariant orbits that intersect it is n.

Solution to Exercise 151.
Let us begin by explaining some of the definitions used in the exercise. A term is a finite tree where internal nodes are the functions, and the leaves are variables or constant operations (operations of arity zero). Given a term t with n variables, and elements $a_1, \ldots, a_n$ of the universe of the algebra, we write $t(a_1, \ldots, a_n)$ for the element in the universe of the algebra which is obtained by evaluating the term, using a_i instead of the i-th variable.

Let us now prove the equivalence in the exercise.

Let us begin with the right-to-left implication. Assume (*). Since the number of operations is finite, one can easily write a deterministic Turing machine which on input $a_1 \cdots a_n$ enumerates all terms with n variables, and then evaluates each term on arguments $a_1, \ldots, a_n$. It is important here that we have finitely many operations. If the family of operations would be orbit-finite, then a nondeterministic Turing machine would be needed to produce the terms.

Let us now do the left-to-right implication. Consider a deterministic Turing machine. The set A is going to contain the work alphabet and the state space of the Turing machine, and the functions $\mathcal{F}$. For every $n, k, i \in \mathbb{N}$ there exist terms $s_{n,k}$ and $t_{n,k,i}$, each one with k arguments, such that for every input word $a_1 \cdots a_k$, the value

$$s_{n,k}(a_1, \ldots, a_k)$$

is the state of the machine after n computation steps, and the value

$$t_{n,k,i}(a_1, \ldots, a_k)$$

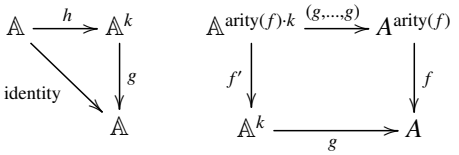
is the symbol of the work alphabet that is stored in the i-th cell of the tape after n computation steps. These terms are produced by unfolding the definition of the computation of a deterministic Turing machine.

Solution to Exercise 152.
The right-to-left implication is shown the same way as in Exercise 151. For the

left-to-right implication, we need a stronger version of Exercise 151, where in condition (*) the set A is equal to $\mathbb{A}^k$ for some k. Suppose then that (*) holds for some $A, \mathcal{F}$ and F. By Exercise 112 (actually, a small strengthening to tuples of functions which can be obtained using the same proof), there exists some $k \in \mathbb{N}$ and finitely supported functions

$$g : \mathbb{A}^k \to A \qquad h : \mathbb{A} \to \mathbb{A}^n \qquad \{f' : \mathbb{A}^{\text{arity}(f)\cdot k} \to \mathbb{A}^k\}_{f\in\mathcal{F}}$$

such that the following diagrams commute for every $f \in \mathcal{F}$:



From (*) it follows that for every n one can compute a term t over the functions $\{f'\}_{f\in\mathcal{F}}$ such that a word $a_1 \cdots a_n$ is accepted by the Turing machine if and only if

$$g(t(h(a_1), \ldots, h(a_n))) \in F.$$

The above is a quantifier-free formula using the functions $\{f'\}_{f\in\mathcal{F}}$ and h, together with the relation $g^{-1}(F)$.

Solution to Exercise 153.
One could use Exercises 151 and 152, but we present here a self-contained proof.

The right-to-left implication is straightforward, so we only do the left-to-right implication. Consider a nondeterministic Turing machine which recognises the language L. Let Q be the states of the machine and let Γ be the work alphabet of the machine. We assume that the work alphabet already contains the blank symbol. Define a *configuration* of the machine to be a word in

$$\Gamma^*(\Gamma \times Q)\Gamma^*$$

with the usual interpretation. Define

$$\Delta = \Gamma \cup (\Gamma \times Q).$$

The following claim is a formalisation of the standard observation that the contents of cell i in a configuration depend only on the contents of cells $i - 1, i, i + 1$ in the previous configuration.

Claim 10.7. *There exists a finite family of finitely supported relations*

$$\mathcal{R} = \{R_i \subseteq \Delta^{n_i}\}_{i\in I}$$

such that for every $n \in \mathbb{N}$, the relations

$$\bar{a}, \bar{b} \in \Delta^n \text{ are configurations and the machine can go in one step from } \bar{a} \text{ to } \bar{b}$$

is defined by a quantifier-free formula in $2n$ variables using only relations from $\mathcal{R}$.

Proof We use relations to check if: a letter contains the state (arity 1), a transition is correctly applied (arity 3). □

The following claim uses the previous claim and existential quantification to guess computations.

Claim 10.8. *There exists a finite family of finitely supported relations*

$$\mathcal{R} = \{R_i \subseteq \Delta^{n_i}\}_{i \in I}$$

such that for every $n \in \mathbb{N}$, the relation

$$\bar{a} \in \mathbb{A}^n \text{ is accepted by the Turing machine } M$$

is defined by a formula of the form $\exists \bar{b} \in \Delta^m \varphi(\bar{a}\bar{b})$ such that φ is quantifier-free formula and uses only relations from $\mathcal{R}$.

Proof Let $\bar{c}$ be a support of the Turing machine M. By Lemma 5.3, the function

$$t : \mathbb{A}^* \to \mathbb{N} \cup \{\infty\}$$

which maps an input word to the smallest length of an accepting computation (which is ∞ for rejected inputs) is also supported by $\bar{c}$.

Let $n \in \mathbb{N}$. By the assumption that the atoms are oligomorphic, there are finitely many $\bar{c}$-orbits of $\mathbb{A}^n$. Let m be the maximal value finite of the function t on $\mathbb{A}^n$, i.e. every input of length n is either rejected or accepted in at most m computation steps. The formula in the statement of the claim quantifies existentially over computations of length at most m, and then uses the quantifier-free formula from Claim 10.7 to check if a computation is correct (we also need additional predicates to check if a configuration is initial/accepting). □

By the assumption on oligomorphism there exists some n and a surjective finitely supported function $g : \mathbb{A}^k \to \Delta$. Let $\mathcal{R}$ be the family from Claim 10.8. For $i \in I$ define $S_i \subseteq \mathbb{A}^{n_i \cdot k}$ to be the relation defined by

$$S_i(\bar{a}_1, \ldots, \bar{a}_{n_i}) \quad \text{iff} \quad R_i(g(\bar{a}_1), \ldots, g(\bar{a}_{n_i})) \qquad \text{for } \bar{a}_1, \ldots, \bar{a}_{n_i} \in \mathbb{A}^k.$$

From Claim 10.8 it follows that for every n, the set

$$\bar{a} \in \mathbb{A}^n \text{ is accepted by the Turing machine } M$$

is defined by a formula of the form $\exists \bar{b} \in \mathbb{A}^m \varphi(\bar{a}\bar{b})$ such that φ is quantifier-free and uses only relations from $\{S_i\}_{i \in I}$.

To finish the exercise, we only need to reduce the finite set of relations $\{S_i\}_{i \in I}$ to a single relation. Suppose that the finite set consists of relations $S_1, \ldots, S_p$. Without loss of generality, we assume that all relations have the same arity n (otherwise we can add unused arguments). We can code them as a single relation S of arity $p + 1 + n$ defined by

$$S(a_0, a_1, \ldots, a_p, b_1, \ldots, b_n) \quad \text{iff} \quad \bigwedge_{i \in \{1,\ldots,p\}} (a_0 = a_i) \Rightarrow S_i(b_1, \ldots, b_n).$$

Each of the relations S_i can be defined in terms of S using an existential formula. For this encoding to work, one need $\mathbb{A}$ to have size at least two, but if $\mathbb{A}$ has only one element, then the exercise is immediate, since there is only one word of each length.

Solution to Exercise 154.
In this case, conditions (**) from Exercise 152 and (***) from Exercise 153 are the same.

Solution to Exercise 155.
By the proof of Theorem 10.4, item 1 from the exercise is equivalent to the following property:

(*) there exists a nondeterministic Turing machine which recognises the following language over input alphabet $\mathbb{A} \cup \{0, 1\}$:

$$\{a_1 \cdots a_n \underline{\varphi} : a_1, \ldots, a_n \in \mathbb{A}, \varphi \text{ has } n \text{ free variables, and } \mathbb{A} \models \varphi(a_1, \ldots, a_n)\}.$$

Therefore, to prove the exercise, we will show that item 2 in the exercise is equivalent to (*). The implication from 2 to (*) is straightforward. For the converse implication, one uses Exercise 153 and (*) to show that there exists a single finitely supported relation $S \subseteq \mathbb{A}^k$ such that every first-order formula φ is equivalent to an existential formula $\hat{\varphi}$ (i.e. a prefix of existential quantifiers followed by a quantifier-free formula) that uses only the predicate S. Since the language in (*) is equivariant, from the proof of Exercise 153 one can conclude that also S is equivariant. Furthermore, the formula $\hat{\varphi}$ can be computed based on φ; this is because the language from (*) is self-dual, and therefore one can compute for every n an upper bound on the length of computations needed to accept all inputs of length at most n. It follows that the structure $\mathbb{A}$ has the same automorphism group as the structure (A, S).

Solution to Exercise 156.
One can compute a representation of the input without atoms, by checking which linear combinations of the input atoms are zero.

Solution to Exercise 157.
By looking at the finitely many possible equivariant relations, and then doing a deatomisation procedure in each case.

Solution to Exercise 158.

Solution to Exercise 159.
See (Klin et al., 2014, Example 2.5 and discussion at the end of Section 5).

Bibliography

Abdulla, Parosh Aziz, Cerans, Karlis, Jonsson, Bengt, and Tsay, Yih-Kuen. 2000. Algorithmic Analysis of Programs with Well Quasi-ordered Domains. *Inf. Comput.*, **160**(1-2), 109–127.

Alur, Rajeev, and Dill, David L. 1994. A theory of timed automata. **126**(2), 183–235.

Alur, Rajeev, Černý, Pavol, and Weinstein, Scott. 2009. *Algorithmic Analysis of Array-Accessing Programs*. Berlin, Heidelberg: Springer Berlin Heidelberg. Pages 86–101.

Bárány, Vince, Bojańczyk, Mikołaj, Figueira, Diego, and Parys, Pawel. 2012. Decidable classes of documents for XPath. Pages 99–111 of: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*.

Bell, John L. 2019. The Axiom of Choice. In: Zalta, Edward N. (ed), *The Stanford Encyclopedia of Philosophy*, spring 2019 edn. Metaphysics Research Lab, Stanford University.

Björklund, Henrik, and Schwentick, Thomas. 2010. On notions of regularity for data languages. *Theor. Comput. Sci.*, **411**(4-5), 702–715.

Blass, Andreas. 2013. *Power-Dedekind Finiteness*. http://www.math.lsa.umich.edu/ ablass/pd-finite.pdf.

Blumensath, Achim, and Gradel, Erich. 2000. Automatic structures. Pages 51–62 of: *Logic in Computer Science, 2000. Proceedings. 15th Annual IEEE Symposium on*. IEEE.

Bojańczyk, Mikołaj. 2011. Data Monoids. Pages 105–116 of: *STACS*.

Bojańczyk, Mikołaj. 2013. Nominal Monoids. *Theory Comput. Syst.*, **53**(2), 194–222.

Bojańczyk, Mikołaj, and Lasota, Sławomir. 2012a. An extension of data automata that captures XPath. *Logical Methods in Computer Science*, **8**(1).

Bojańczyk, Mikołaj, and Lasota, Sławomir. 2012b. A Machine-Independent Characterization of Timed Languages. Pages 92–103 of: *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*.

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 09-12, 2018*. ACM.

Bojańczyk, Mikołaj, Muscholl, Anca, Schwentick, Thomas, and Segoufin, Luc. 2009. Two-variable logic on data trees and XML reasoning. *J. ACM*, **56**(3), 13:1–13:48.

Bojańczyk, Mikołaj, David, Claire, Muscholl, Anca, Schwentick, Thomas, and Segoufin, Luc. 2011. Two-variable logic on data words. **12**(4), 27:1–27:26.

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).

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 Inf.*, **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*.

Colcombet, Thomas, and Manuel, Amaldev. 2014. μ-Calculus on Data Words. *CoRR*, **abs/1404.4827**.

Colcombet, Thomas, Ley, Clemens, and Puppis, Gabriele. 2015. Logics with rigidly guarded data tests. *Logical Methods in Computer Science*, **11**(3).

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.

Czerwiński, Wojciech, Lasota, Sławomir, Lazić, Ranko, Leroux, Jérôme, and Mazowiecki, Filip. 2018. The Reachability Problem for Petri Nets is Not Elementary.

Demri, Stéphane, and Lazić, Ranko. 2009. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, **10**(3), 16:1–16:30.

Engeler, ERWIN. 1959. A characterization of theories with isomorphic denumerable models. *Notices Amer. Math. Soc*, **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*.

Finkel, Alain, and Schnoebelen, Philippe. 2001. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, **256**(1-2), 63–92.

Gabbay, Murdoch, and Pitts, Andrew M. 2002. A New Approach to Abstract Syntax with Variable Binding. *Formal Asp. Comput.*, **13**(3-5), 341–363.

Göller, Stefan, Haase, Christoph, Lazić, Ranko, and Totzke, Patrick. 2016. A Polynomial-Time Algorithm for Reachability in Branching VASS in Dimension One. Pages 105:1–105:13 of: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*.

Grohe, Martin. 2008. The quest for a logic capturing PTIME. Pages 267–271 of: *Logic in Computer Science, 2008. LICS'08. 23rd Annual IEEE Symposium on*. IEEE.

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

Jacquemard, Florent, Segoufin, Luc, and Dimino, Jerémie. 2016. FO2(<, +1, ˜) on data trees, data tree automata and branching vector addition systems. *Logical Methods in Computer Science*, **12**(2).

Jurdziński, Marcin, and Lazić, Ranko. 2011. Alternating automata on data trees and XPath satisfiability. *ACM Trans. Comput. Log.*, **12**(3), 19:1–19:21.

Kaminski, Michael, and Francez, Nissim. 1994. Finite-Memory Automata. *Theor. Comput. Sci.*, **134**(2), 329–363.

Klin, Bartek, and Łełyk, Mateusz. 2017. Modal Mu-Calculus with Atoms. 21 pages.

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.

Klin, Bartek, Lasota, Sławomir, Ochremiak, Joanna, and Toruńczyk, Szymon. 2016. Homomorphism problems for first-order definable structures. In: *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 65. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

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.*

Kosaraju, S. Rao. 1982. Decidability of Reachability in Vector Addition Systems (Preliminary Version). Pages 267–281 of: *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*.

Lasota, Sławomir, and Piórkowski, Radosław. 2018. WQO Dichotomy for 3-Graphs. Pages 548–564 of: *FoSSaCS*. Lecture Notes in Computer Science, vol. 10803. Springer.

Leroux, Jérôme. 2010. The General Vector Addition System Reachability Problem by Presburger Inductive Invariants. *Logical Methods in Computer Science*, **6**(3).

Leroux, Jérôme, and Schmitz, Sylvain. 2019. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension.

Mac Lane, Saunders, and Moerdijk, Ieke. 1992. *Sheaves in geometry and logic: a first introduction to topos theory*. Springer.

Manuel, Amaldev, Muscholl, Anca, and Puppis, Gabriele. 2016. Walking on Data Words. *Theory Comput. Syst.*, **59**(2), 180–208.

Mayr, Ernst W. 1984. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comput.*, **13**(3), 441–460.

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*.

Montanari, Ugo, and Pistore, Marco. 1999. Finite State Verification for the Asynchronous π-Calculus. Pages 255–269 of: *TACAS*.

Montanari, Ugo, and Pistore, Marco. 2005. History-Dependent Automata: An Introduction. Pages 1–28 of: *SFM*.

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. Bisimilarity 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*.

Neven, Frank, Schwentick, Thomas, and Vianu, Victor. 2004. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, **5**(3), 403–435.

Pitts, Andrew M. 2013. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science, vol. 57. Cambridge University Press.

Presburger, Mojżesz. 1929. Über Die Vollständigkeit Eines Gewissen Systems Der Arithmetik Ganzer Zahlen, in Welchem Die Addition Als Einzige Operation Hervortritt. Pages 92–101 of: *Congrès Des Mathématiciens Des Pays Slaves, Warszawa*, vol. 129.

Rossman, Benjamin. 2010. Choiceless Computation and Symmetry. Pages 565–580 of: Blass, Andreas, Dershowitz, Nachum, and Reisig, Wolfgang (eds), *Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg.

Ryll-Nardzewski, Czesław. 1959. On the categoricity in power 0. *Bull. Acad. Polon. Sci. Sér. Sci. Math. Astr. Phys*, **7**, 545–548.

Schmerl, James. 1978. A decidable $\aleph_0$-categorical theory with a non-recursive Ryll-Nardzewski function. *Fundamenta Mathematicae*, **98**, 121–125.

Schmitz, Sylvain, and Schnoebelen, Philippe. 2012 (Aug.). *Algorithmic Aspects of WQO Theory*. Lecture.

Scott, Dana. 1962. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, **27**(377), 74.

Segoufin, Luc. 2006. Automata and Logics for Words and Trees over an Infinite Alphabet. Pages 41–57 of: *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*.

Svenonius, Lars. 1959. No-categoricity in first-order predicate calculus 1. *Theoria*, **25**(2), 82–94.

Thomas, Wolfgang. 1990. Automata on Infinite Objects. Pages 133–192 of: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*.

Thomas, Wolfgang. 1997. Languages, Automata, and Logic. Pages 389–455 of: *Handbook of Formal Languages, Volume 3: Beyond Words.*

Wysocki, Tomasz. 2013. *Automaty alternujace z rejestrami na skończonych słowach (Alternating Register Automata on Finite Words)*. Masters' Thesis, University of Warsaw.

Author index

Subject index