Slightly Infinite Sets

Mikołaj Bojańczyk

December 4, 2018
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 not necessarily finite, but are 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 the usual 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 were initially studied.

PART ONE

AUTOMATA FOR DATA WORDS

We begin with an investigation of concrete automata models for words over infinite alphabets. One goal of this part is to build up 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. For the rest of Part I, fix a countably infinite set $\mathbb{A}$. Elements of this set, called the *atoms*, will be used for the data values. Define a *data word* over a finite set of labels Σ to be a word in

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

The idea is that we can test labels explicitly by asking questions like

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

but we can only test the data value 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 over the alphabet $\mathbb{A}$ 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 alphabet Σ of *labels*;
- a finite set Loc of *locations*;
- 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*

$$\delta \subseteq \underbrace{\mathsf{Loc} \times (\mathbb{A} \cup \{\bot\})^R}_{\text{states}} \times \underbrace{\Sigma \times \mathbb{A}}_{\text{input}} \times \underbrace{\mathsf{Loc} \times (\mathbb{A} \cup \{\bot\})^R}_{\text{states}} \tag{1.1}$$

  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 (i.e. a partial function from register names to atoms, partial because some registers are empty). 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 of the word the state is accepting, in the sense that the location belongs to the accepting set.

How to describe the transition relation? 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. 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). We say that δ is *semantically*

[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 1.6 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).

equivariant if

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

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) the atom is undefined in register $r \in R$ of the source (respectively, target) state;
(4) the atom in the input letter equals the contents of register $r \in R$ in the source (respectively, target) state;
(5) the atom in register $r \in R$ of the source (respectively, target) state equals the atom in 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. *Semanticx 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 subset of the set (1.1) which is semantically equivariant and which is minimal for that property with respect to inclusion.

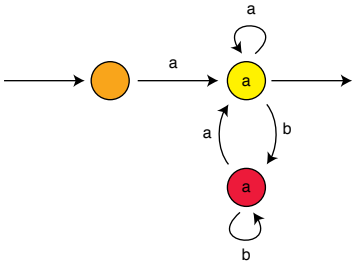
Claim 1.4. *Every orbit of transitions is syntactically equivariant.*

of Claim An orbit of transitions is uniquely defined by its locations, which registers are undefined, and what is the equality type of the tuple of atoms in the defined registers. 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 yellow location with an undefined register, we would need to draw a separate instance of the yellow state.

Exercises

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

Exercise 2. 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 3. 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 4. Show that a nondeterministic register automaton can recognise language 2 from Example 1.1, but a deterministic one cannot.

Exercise 5. Call a nondeterministic register automaton *guessing* if there exists a transition $t \in \delta$ such that some data value in the target register valuation appears neither in the source register valuation 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 6. 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: emptiness (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 1.6. *Emptiness is decidable for nondeterministic register automata.*

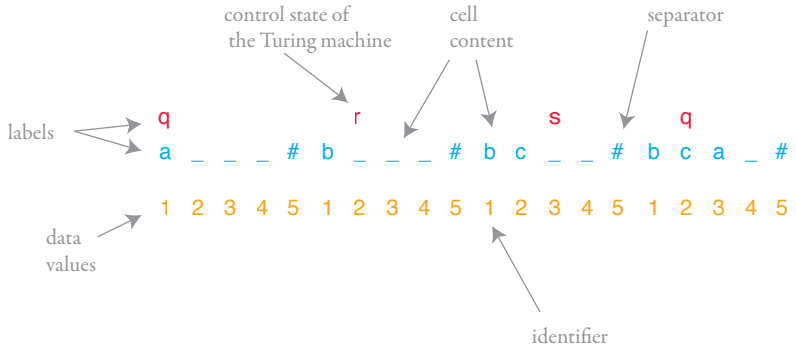
Proof This proof just sketches the decidability argument, the complexity is discussed in Exercise 7. Define an *orbit of states* to be a set of states

$$Q \quad \subseteq \quad \mathsf{Loc} \times \text{register valuations}$$

that is closed under permutations of atoms. As in Lemma 1.3, an orbit of configurations can be defined by saying what is the location, which are the defined registers, and what is the equality type on the atoms stored in the defined registers. Such a description takes finite space to store, and there are finitely many possible descriptions. The key observation is that being in the same orbit of states is a congruence with respect to transitions, 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 equality type of the unique initial states, which can be easily computed. 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 1.7. *Universality is undecidable for nondeterministic register automata.*

Proof We reduce from the halting problem for Turing machines. Suppose that we have a Turing machine which is an instance of the halting problem. We encode a run of a Turing machine as a data word according to the following picture:



Each letter encodes a single cell in a single configuration of the Turing machine. The 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 (blue in the picture), plus the control state

(red) of the head if the head happens to be over that cell. Finally, each cell gets a unique identifier, a data value (orange). The following claim shows that the halting problem reduces to universality of nondeterministic register automata, thus proving the theorem.

Claim 1.8. *There is a nondeterministic register automaton which accepts a data word if and only if it is* not *an encoding of an accepting run of the Turing machine.*

Proof To prove the claim, we list the mistakes that can happen in a word that does not encode an accepting run 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 $j + 1$ 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 7. The complexity of the emptiness problem 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 8. The undecidability proof in Theorem 1.7 used automata with two registers but no guessing (as defined in Exercise 5). Show that, in the presence of guessing, universality remains undecidable even with one register.

Exercise 9. 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:

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

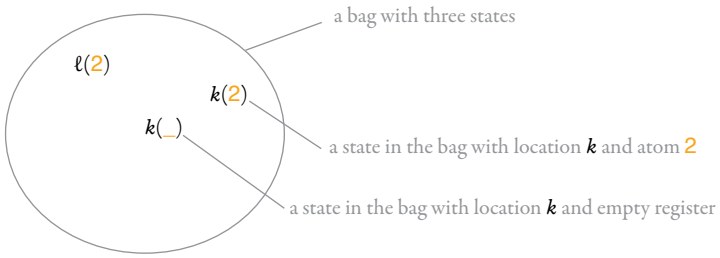
Show that satisfiability is undecidable for this logic, i.e. one cannot decide if a given formula is true in some data word.

1.3 Alternating register automata

In a nondeterministic automaton, the 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 states chose transitions in favour of acceptance, and which states chose transitions against acceptance. The main result of this section is that emptiness is decidable for a restricted version of alternating register automata[2].

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*, where a bag is defined to be a finite set of states. Here is a picture of a bag:



[2] Decidability of emptiness for alternating one-way register automata with one register, Theorem 1.9 in this text, was first shown in Demri and Lazic (2009). A tree extension of the result can be found in Jurdzinski and Lazic (2011)

We use finite bags because we are interested in automata without guessing, where a state can split only into finitely many states when reading an input letter. We write P, Q for bags. For every input letter a (consisting of a label and a data value), we define a binary relation $\stackrel{a}{\to}$ on bags, such that $P \stackrel{a}{\to} Q$ holds if:

- 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;
- 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 if there exists a run

$$\text{initial bag } = Q_0 \stackrel{a_1}{\to} Q_1 \stackrel{a_2}{\to} \cdots \stackrel{a_n}{\to} Q_n \in \text{accepting bags}$$

where the initial bag is defined to be the singleton of the initial state (the initial state and all registers undefined), and an accepting bag is defined to be any bag that contains only accepting states (states with an accepting location). We define $\to$ to be the union of all relations $\stackrel{a}{\to}$, 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 of $\to$.

Languages recognised by alternating register automata are closed under complementation, see Exercise 10.

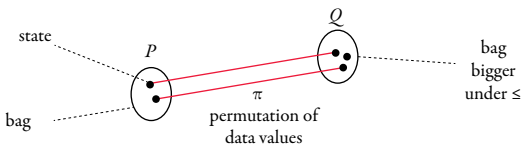
An emptiness algorithm using well quasi-orders Nondeterministic register automata are the special case of alternating register automata where all states are existential. By Exercise 10, the emptiness and universality problems for alternating register automata are essentially the same problem, which is undecidable by Theorem 1.7. Furthermore, by the remarks in Exercise 8, this undecidability is true already for two registers and no guessing, or even one register with guessing. That is the limit of undecidability:

Theorem 1.9. *Emptiness is decidable for one register alternating automata without guessing.*

The rest of this section is devoted to proving the above theorem. Nonemptiness for alternating automata 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 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.

As in the definition of semantic equivariance from Section 1, permutations of the atoms can be applied to states and to bags 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:



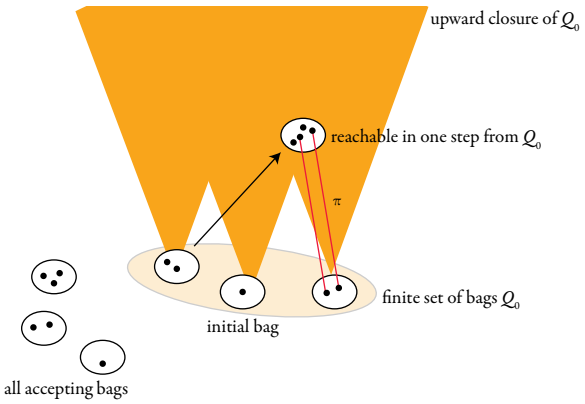
This is easily seen to be a quasi-ordering, i.e. a transitive and reflexive relation. Call a set of bags *upward closed* if it is upward closed with respect to this quasi-order. The *upward closure* of a set of bags is the least upward closed set of bags that contains it.

The following lemma gives the semi-decidability of emptiness.

Lemma 1.10 (Finite Emptiness Witness Lemma)**.** *An alternating one register automaton without guessing accepts no words if and only if there exists some finite set of bags $\mathcal{Q}_0$ which is a witness in the following sense:*

(1) *$\mathcal{Q}_0$ contains no accepting bags, i.e. each bag in $\mathcal{Q}_0$ contains a non-accepting state;*
(2) *the initial bag is in $\mathcal{Q}_0$;*
(3) *if $P \to Q$ and $P \in \mathcal{Q}_0$, then P is in the upward closure of $\mathcal{Q}_0$.*

Here is a picture of the witness from the above lemma:



The idea is that the orange area, i.e. the upward closure of C_0, is a trap in the sense that no transition can leave the orange area. It is straightforward to see that existence of a witness is semi-decidable: guess the set C_0 and check the three conditions. For the third condition, it is useful that there is no guessing, since lack of guessing ensures that the relation $\rightarrow$ has finite outdegree. Therefore, the Finite Emptiness Witness Lemma completes the proof of Theorem 1.9. It remains to prove the lemma, which we do in the rest of this section. Fix an alternating one register automaton.

There are two key properties of the relation $\leq$ which make the Finite Emptiness Witness Lemma true: it is a well quasi-order and it is compatible with transition relation $\rightarrow$ on bags. These are explained and proved below.

Well quasi-order. 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, as used in the following proof, is a common method of proving decidable properties for systems with infinitely many configurations.

Lemma 1.11. *The relation $\leq$ on finite bags is a well quasi-order.*

Proof It is clear that the relation is well-founded, since a strict decrease on finite bags implies a strict decrease in the cardinality. It remains to show that there is no infinite antichain. Define the *profile* of a bag Q to be the following information:

- for which locations ℓ does the bag contain the state $\ell(\_)$, i.e. the state with location ℓ and an undefined register;
- for each nonempty set of locations X, what is the size of the set:

$$\{a \in \mathbb{A} : X \text{ is the set of locations } \ell \text{ such that } \ell(a) \text{ is in the bag}\}$$

A profile can be seen as a binary vector indexed by locations plus a vector of natural numbers indexed by nonempty subsets of locations. The profile mapping takes incomparable pairs (of bags under $\leq$) to incomparable pairs (of profiles seen as vectors ordered coordinatewise). Therefore, the profile mapping takes infinite antichains to infinite antichains. Since there are no infinite anitichains in the latter space by Exercise 12, it follows that there are no infinite antichains in the former space. □

In our proof, we will be using the following corollary of being a well quasi-order.

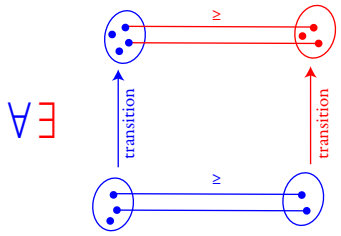
Lemma 1.12. *Every upward closed set of bags is the upward closure of a finite set of bags.*

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 renamings). □

Compatibility. The following lemma shows that the order $\leq$ on bags is compatible with the transition relation $\rightarrow$ on bags in the sense that making the source bag smaller makes doing transitions easier.

Lemma 1.13. *The relation $\leq$ on bags is compatible with $\rightarrow$ in following sense: for every transition $P \rightarrow Q$ and every $P' \leq P$ there exists some $Q' \leq Q$ with $P' \rightarrow Q'$.*

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. □

Using compatibility, we can prove the right-to-left implication in the Finite Emptiness Witness lemma. Suppose then that $\mathcal{Q}_0$ is a finite set of bags which satisfies the three conditions in the lemma. Let $\mathcal{Q}$ be the upward closure of $\mathcal{Q}_0$, i.e. a bag belongs to $\mathcal{Q}$ if and only if it contains – up to atom permutations – some bag in $\mathcal{Q}_0$. Since non-accepting bags are upward closed, condition 1 in the lemma implies that $\mathcal{Q}$ contains only non-accepting bags. Condition 2 implies that $\mathcal{Q}$ contains the initial bag. Finally, compatibility ensures that if $P \in \mathcal{Q}$ and $P \rightarrow Q$, then also $Q \in \mathcal{Q}$. In other words, $\mathcal{Q}$ is an invariant which witnesses that the initial bag cannot reach any accepting bag.

Finding the finite emptiness witness. To complete the proof of the Finite Emptiness Witness lemma, we need to prove the left-to-right implication, i.e. find the finite witness C_0 in any alternating automaton that accepts no words.

Define $\mathcal{R}$ to be the set of bags which can reach some accepting bag in a finite number of steps in the relation $\rightarrow$.

Lemma 1.14. *$\mathcal{R}$ is downward closed.*

Proof Take any finite path

$$Q_n \to Q_{n-1} \to \cdots \to Q_1 \in \text{accepting bags.}$$

To prove the lemma, we prove by induction on n that if $P \leq Q_n$, then $P \in \mathcal{R}$. The induction base is the fact that the set of accepting bags is downward closed. For the induction step, we use the compatibility established in Lemma 1.13. □

Stated differently, the above lemma says that the complement of $\mathcal{R}$ is upward closed. Apply Lemma 1.12 to this complement, yielding a finite set of bags $\mathcal{Q}_0$. We will prove that $\mathcal{Q}_0$ is a witness in the sense of the Finite Emptiness Witness Lemma. By definition of $\mathcal{Q}_0$, every bag Q satisfies

$$Q \text{ cannot reach an accepting bag} \quad \text{iff} \quad Q \text{ is in the upward closure of } \mathcal{Q}_0.$$

To prove that $\mathcal{Q}_0$ is a witness, let us check the three conditions from the Finite Emptiness Witness Lemma. Clearly there can be no accepting bags in $\mathcal{Q}_0$, because an accepting bag can reach an accepting bag in zero steps. By assumption that the automaton is empty, the initial bag cannot reach an accepting bag, and hence the initial bag is in the upward closure of $\mathcal{Q}_0$. Only the empty bag is smaller than the initial bag, and the empty bag is accepting, hence the initial bag must actually be in $\mathcal{Q}_0$, and not only in its upward closure. Finally, let us prove the third condition. The upward closure of $\mathcal{Q}_0$ is closed under taking a step of $\to$, since if Q cannot reach an accepting bag, then the same is true for anything reachable from Q. This implies the third condition. This completes the proof of the Finite Emptiness Witness Lemma and of Theorem 1.9.

The general technique. Using the same proof, we obtain the following generalisation [3] of Theorem 1.9.

Theorem 1.15. *The following problem is decidable.*

- **Input**.
  - *A directed graph where every node has finite outdegree;*
  - *A well quasi-order $\leq$ on vertices which is compatible with the edge relation;*
  - *A source vertex plus a set of target vertices that is downward closed.*

[3] 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*

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.

- **Question.** *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 a prefix there is some position with label b such that all earlier positions have label a. Instead of a, b we could have used simpler formulas, and Boolean combinations are allowed. There is also an operator to access the next position, so e.g. 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:

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

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

Exercise 11. 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 12. 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 13. 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 14. 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.

Exercise 15. 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 the halting problem is decidable for Turing machines with semantics defined using gainy computation steps.

Exercise 16. 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 17. 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 show that, 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[4]. For the purpose of this section, we assume that all models allow ϵ-transitions.

Exercise 18. 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 19. Show that for every nondeterministic two-way register automaton $\mathcal{A}$ with one register, if the labels are Σ, then the following language is regular:

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

Exercise 20. Find a language that is recognised by an alternating register automaton with guessing, but not by any alternating register automaton without guessing.

Exercise 21. 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

[4] 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 1.7). The non-equivalence results summarised in Figure 1.1 are originally found in Kaminski and Francez (1994); Neven et al. (2004) and an unpublished Masters' thesis in Polish Wysocki (2013). For a survey on automata and logic for infinite alphabets, see also Segoufin (2006).

containment from Exercise 20. In the solutions to the following exercises, one is allowed to give answers conditional on open problems in complexity theory such as P = NP.

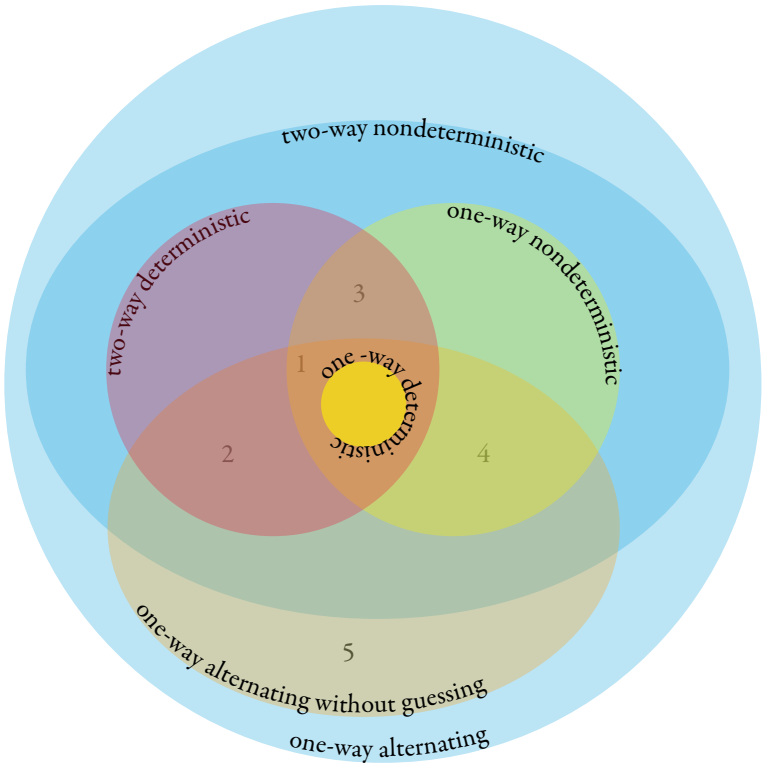


Figure 1.1 Six classes of register automata and their combinations. Point 1 is the language: "last letter appears only once", while point 2 is the language "all letters are distinct". The remaining points 3, 4, 5 are Exercises 22-24, while Exercise 25 sums up the results by saying that all combinations are possible.

Exercise 22. Show a language that witnesses point 3 in Figure 1.1.

Exercise 23. Show a language that witnesses point 4 in Figure 1.1, possibly conjectures about complexity classes being distinct.

Exercise 24. Show a language that witnesses point 5 in Figure 1.1, possibly using conjectures about complexity classes being distinct.

Exercise 25. Show that all coloured areas in Figure 1.1 contain languages.

2

Two variable logic and data automata

In this section we define an automaton model – *data automata* – which recognises properties of data words without using registers. There are three reasons to discuss data automata: emptiness of data automata are 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.

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. Iin other words, the semantics of such an automaton is a binary relation

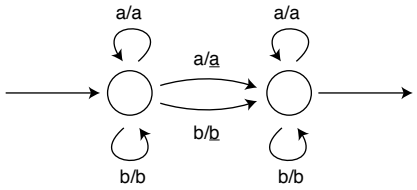
$$R \subseteq \Sigma^* \times \Gamma^*.$$

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.

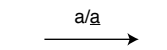
Example 1. Consider the set of pairs

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

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



Explanation
A transition like this



inputs a and outputs a

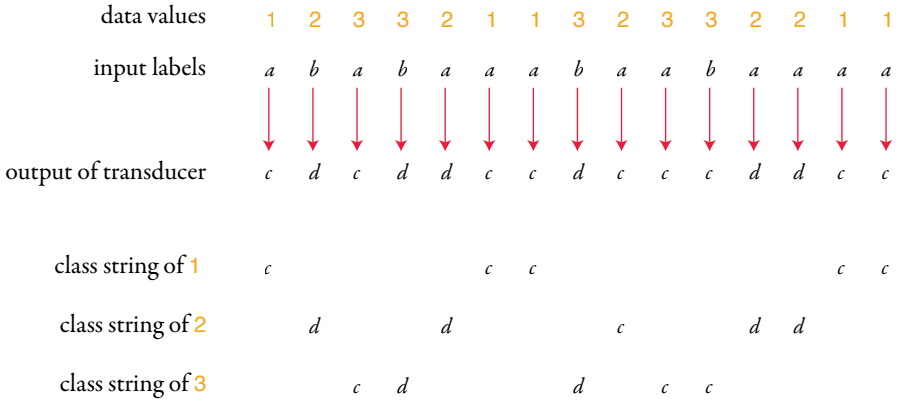
□

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 rejects data words in $(\Sigma \times \mathbb{A})^*$. For a data word, define a *class* to 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. □

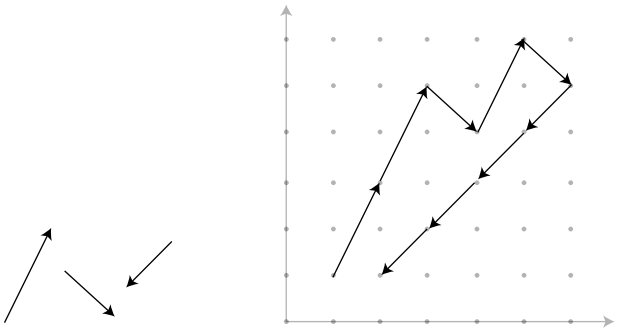
Example 4. Suppose that the labels in the input alphabet are $\{a, b, c\}$. Consider a data automaton where the transducer is the identity and the class condition says that each letter occurs exactly once. The language recognised by this data automaton, after erasing data values, is the set of words in $\{a, b, c\}^*$ where each letter appears the same number of times. □

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 nonnegative on all coordinates, even if 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 last one. The following famous result uses one of the most difficult decidability proofs; 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 vector addition system $\delta \subseteq \mathbb{Z}^d$ together with designated initial and final vectors in $\mathbb{N}^d$, as well as a relation $\gamma \subseteq \delta \times \Sigma$ which associates to each transition the input letters that can be used for it. A word in Σ^* is accepted if there exists a run from the initial to the final vector, which is consistent with the input word according to γ. 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 $\{1, \ldots, n\}$. To simulate this 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. (To be completely precise, the lemma only works for regular languages $L \subseteq \Sigma^+$, i.e. regular languages of nonempty words, since otherwise we would need the initial and final vectors to be the same. If a regular language does not contain the empty word, then one can find a nondeterministic automaton with exactly one initial and exactly one accepting state.) □

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 see 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. □

Exercise 26. Show that languages recognised by data automata are not closed under Kleene star.

Exercise 27. 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 $\mathbb{Z}^d$.

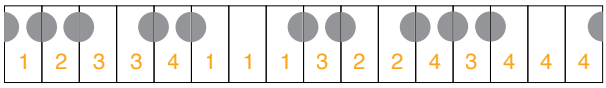
2.2 Logic on data words

Recognising equality with successors

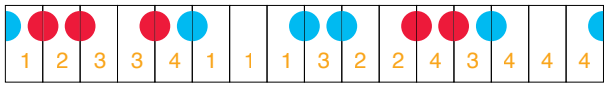
We now show that a data automaton can compute which positions in a data word have the same data value as their neighbours.

Lemma 2.7. *There is a data automaton which recognises the set of data words where the label of each position is the subset of* {*predecessor, successor*} *that indicates which neighbours of the position have the same data value.*

Proof Instead of writing a set we draw a semicircle on the left side if the label does not contain "predecessor" and a semicircle on the right side if the label does not contain "successor", as in the following picture:



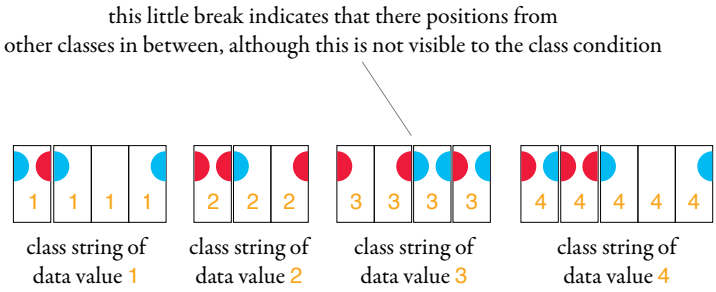
Given an input data word, the transducer in the data automaton guesses a colouring of the semicircles with two colours as in the following picture:



Such a colouring is called consistent (the above picture is consistent) if it satisfies the following conditions:

(1) An edge connecting two consecutive positions is labelled by a monochromatic circle, or no circle at all (technically speaking, the label of such an edge is distributed across the two connected positions).
(2) If x, y are consecutive positions in some class (but they might be separated by positions from other classes), then the right side of x and the left side of y

either form no circle at all, or have different colours. To see that this condition is true in the data word from the picture above, consider the following picture which shows the class strings:

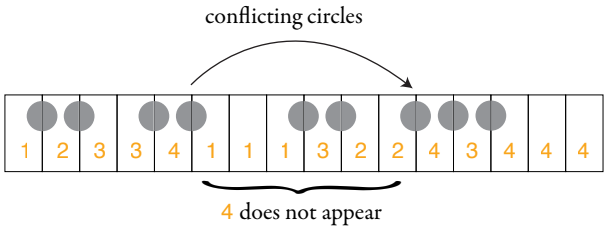


(3) The first position in each class has a semicircle on its left, and the last position in each class has a semicircle on its right.

The notion of consistency is designed so that condition 1 can be checked by the transducer in a data automaton, and conditions 2 and 3 can be checked by the class condition. In the following two claims, we show that a data word belongs to the language if and only if it can be coloured in a consistent way.

Claim 2.8. *Every data word in the language can be coloured in a consistent way.*

Proof 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 second 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 forest, i.e. every circle has indegree at most one. A forest can always be coloured with two colours so that edges have endpoints with different colours. □

Claim 2.9. *If a data word can be coloured consistently, then it is in the language.*

Proof 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 1 this circle is monochromatic, say it has colour c. By condition 2 of consistency, the next position in the class of x has its left side coloured with a different colour than c, and hence the next position in the class of x cannot be the successor of x.

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. □

By Claims 2.8 and 2.9, a data word belongs to the language in the statement of the lemma if and only if its circles can be coloured in a consistent way. Checking if such a colouring exists is done by the data automaton. □

Exercise 28. Let $k \in \{0, 1, \ldots\}$. Show that there is a data automaton which recognises the set of data words where a 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 29. 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 original model of data automata.

Exercise 30. 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.

Exercise 31. 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 32. Show that every language recognised by a nondeterministic register automaton is also recognised by a data automaton. (Hint: use the previous exercise.)

A logic recognised by data automata

In this section we use the decidability of data automata and Lemma 2.7 to show that satisfiability is decidable for a certain modal logic expressing properties of data words.

Define a *modality* to be a formula describing a relationship between two positions x and y in a data word, which is of the form $\alpha \wedge \beta$ where α is either "x has the same data value as y" or "x has a different data value than y" and β is one of the following four formulas:

$$x < y - 1 \qquad x = y - 1 \qquad x = y + 1 \qquad x > y + 1.$$

There are eight modalities in total. Using these modalities, we define a logic on data words, call it *data word modal logic*. Each formula of this logic selects a set of positions in a data word. The formulas of the logic are:

- Every letter $a \in \Sigma$ is a formula, which selects all positions that have label a.
- The set of formulas is closed under Boolean combinations, including negation.
- If m is a modality and φ is a formula, then $\langle m \rangle \varphi$ is a formula, which selects a position x if there exists a position y selected by φ such that $m(x, y)$.

Define the *language* of a formula to be the data words whose first position is selected.

Theorem 2.11. *Every language defined by a formula of the modal logic is recognised by a data automaton.*

The main step in the proof is to show that every modality can be recognised by a data automaton, in the sense made explicit by the following lemma.

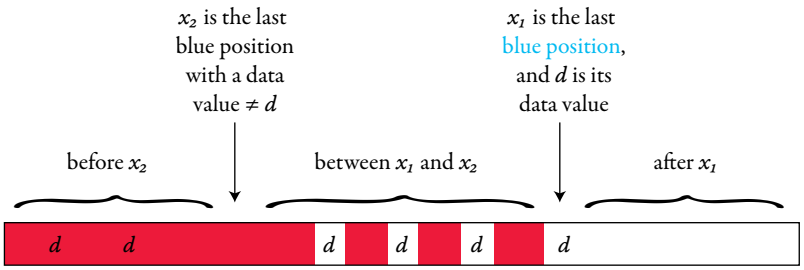
Lemma 2.12. *Let m be one of the modalities. Consider data words each position is coloured by a subset of $\{red, blue\}$. The following property is recognised by data automaton:*

A position x is red if and only if there is a blue position y with $m(x, y)$.

Proof Consider a modality $\alpha \wedge \beta$. If α says that "x and y have the same data value", then the property in the statement of the lemma can be checked using the class automaton, with the help of the labelling from Lemma 2.7 to test which neighbours have the same data value. If β says that y is the successor or predecessor of x, then we can also use Lemma 2.7. The only remaining case is when the modality m is

(*) y has different data value than x and $y > x + 1$

or the symmetric one with $y < x-1$. Because languages recognised by data automata are closed under reversing, we only consider the modality (*). Consider the following parts of a data word:



Some parts might be undefined, e.g. x_1 is undefined if there are no blue positions and x_2 is undefined if at most one data value is used for blue positions. For the modality (*), the property in the statement of the lemma can be reformulated as follows

(1) if a position has data value $\neq d$, then it is red if and only if it is $< x_1 - 1$.
(2) if a position has data value d, then it is red if and only if it is $< x_2 - 1$.

Both of these properties are regular properties of the labelling, assuming that the partition into the parts concerning x_1 and x_2 is known. This partition can be guessed and checked by a data automaton, assuming that special colours are used to mark the classes of x_1 and x_2. □

of Theorem 2.11 Let φ be a formula of data word modal logic. Given an input word, the data automaton nondeterministically guesses for each position x in the data word a set Γ_x of subformulas of φ, intuitively speaking those formulas which select it. Then it checks that:

(1) The set Γ_1 contains φ.
(2) A position x has label a if and only if $a \in \Gamma_x$.
(3) For every position x, the set Γ_x contains a subformula $\varphi_1 \wedge \varphi_2$ of φ if and only if Γ_x contains both φ_1, φ_2. Likewise for $\vee$ and $\neg$.
(4) For every position x, the set Γ_x contains a subformula $\langle m \rangle \psi$ of φ if and only if there is some position y such that $m(x, y)$ and $\psi \in \Gamma_y$.

The first three conditions are regular properties of the labelling and can be checked by the transducer while guessing the sets Γ_x. For every fixed choice of subformula $\langle m \rangle \psi$, the last condition can be verified by a data automaton using Lemma 2.12, by choosing red for those positions that have $\langle m \rangle \psi$ in their label

and choosing blue for positions that have ψ in their label. All of these checks can be done in parallel, hence one data automaton is sufficient. □

Exercise 33. Show that satisfiability for data word modal logic is at least as hard as emptiness for data automata.

Exercise 34. Consider the following alternating automaton based on the modal logic. The automaton has a set of states Q, an initial state q_0, a partition of states into universal and existential, and a transition relation

$$\delta \subseteq Q \times \Sigma \times \text{modalities} \times Q$$

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 $(q, \sigma, m, p) \in \delta$ such that σ is the label of position x, and a position y that is related to x via the modality m. If there is no such transition or position, the player who owns q loses immediately. Otherwise, the game proceeds to state p and some 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 35. Show that the automaton model in Exercise 34 has undecidable emptiness. Show that if infinite plays are won by $\exists$ then emptiness becomes decidable.

Exercise 36. Consider first-order logic on data words, as described in Exercise 9. Show that adding a predicate $x = y + 1$ for testing the successor relation and limiting the logic to two variables, we get a logic that has the same expressive power as data word modal logic.

Bibliographic notes for Section 2. Data automata, the algorithm for their emptiness using vector addition systems, were introduced in Bojańczyk et al. (2011). The original application was for deciding two-variable logic, as in Excercise 36. The model from Bojańczyk et al. (2011) had a built in test for "the next data value is different than the current one"; the fact that this is redundant (Lemma 2.7) was shown in (Björklund and Schwentick, 2010, Proposition 3.6). Exercise 32 is also due to Björklund and Schwentick (2010). Exercise 30 describes a model called *class automata*, which was studied in Bojańczyk and Lasota (2012a); Bárány et al. (2012). Excercise 31 is (Alur et al., 2009, Theorem 2).

Theorem 2.3 on reachability for vector addition systems was originally shown by Mayr in Mayr

(1984), other proofs include Kosaraju (1982) and Leroux (2010). 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 are problems are connected to *branching vector addition systems*, see Göller et al. (2016) and the references therein.

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.** 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. Since sets with atoms are built using atoms, any function on atoms can be lifted naturally to sets with atoms.
- **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 atom permutations. 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 the 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 such structure, we use structures in the sense of model theory: a universe together with some relations and functions. Examples of structures representing atoms that 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. 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

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[1] 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 using actions of atom automorphisms.

Definition 3.1 (Sets with atoms)**.** Let $\mathbb{A}$ be a model, 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 elements are atoms or sets of rank $< \alpha$.
- **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

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

[1] 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.

i.e. x is fixed by every $\bar{a}$-automorphism[2] . 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. One reason for non-uniqueness is that supports are closed under adding atoms, although for some atoms such as $(\mathbb{N}, =)$ or $(\mathbb{Q}, <)$ a canonical least support can be found, see Section 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 6. Consider the equality atoms $\mathbb{A} = (\mathbb{N}, =)$ and the set

$$\{a : a \in \mathbb{A} \text{ with } a \neq \underline{2}\}$$

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 7. Consider the rational number atoms $\mathbb{A} = (\mathbb{Q}, <)$ and the set of open intervals that contain the atom $\underline{2}$:

$$\{\{c : c \in \mathbb{A} \text{ with } a < c < b\} : a, b \in \mathbb{A} \text{ with } a < \underline{2} < b\}$$

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

$$\{c : c \in \mathbb{A} \text{ with } \underline{0} < c < \underline{3}\},$$

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

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

[2] 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.

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 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 8. 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 9. Consider the equality atoms $\mathbb{A} = (\mathbb{N}, =)$. 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, and let π be the transposition which swaps b with c. 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

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

which contradicts the assumption that f is a function[3] □

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

Example 10. 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 $k \in \mathbb{Z}$.

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

- 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.

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 $(\mathbb{Z}, \leq)$ there is no difference between a set in the cumulative hierarchy and a set with atoms. The same issue holds in $(\mathbb{Z}, +)$, where 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 11. [Finitely supported subsets of the equality atoms] Consider the equality atoms. The finitely supported subset of $\mathbb{A}$ 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 then 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 $\bar{a}$. Let π be the permutation of atoms which swaps a and b, and is the identity on other atoms. This permutation is an $\bar{a}$-automorphism, so it should fix X, but it does not. □

Example 12. [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 a, b that are not in $\bar{a}$ and are not separated by an atom in $\bar{a}$ in terms of the order. There is an $\bar{a}$-automorphism which maps a to b. Since the set X is supported by $\bar{a}$, it follows that $a \in X$ if and only if $b \in X$. □

In Examples 11 and 12, the finitely supported sets of atoms coincide with subsets of 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 Section 7. When the atoms are homogeneous, then finitely supported relations on the atoms are exactly those that can be defined using quantifier-free formulas.

Exercises

Exercise 37. 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 38. For the equality atoms, find all equivariant binary relations on $\mathbb{A}$.

Exercise 39. For the atoms $(\mathbb{Q}, <)$, find all equivariant binary relations on $\mathbb{A}$.

Exercise 40. 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 41. 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 42. Consider the equality atoms. Show that for every finitely supported partial order $<$ on $\mathbb{A}$, all atoms outside the support are incomparable.

Exercise 43. Consider the atoms $(\mathbb{Q}, <)$. Show that there is no finitely supported well-founded total order on $\mathbb{A}$.

Exercise 44. 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

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

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

In Section 3.1, we defined sets with atoms. For this book, the point of sets with atoms is to define their orbit-finite fragment. 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 renaming atoms (although one has to be a bit careful about which renamings are allowed, due to constants such as $\underline{1}, \underline{2}$ used above that can appear in the definition of the set). This section is devoted to defining orbit-finiteness. The definition of orbit-finiteness makes sense only for certain atom structures, namely the oligomorphic ones, so we begin by defining oligomorphism.

Definition 3.2. A structure $\mathbb{A}$ is called *oligomorphic* if for every $n \in \{1, 2, \ldots\}$, the structure $\mathbb{A}^n$ has finitely many elements up to automorphisms of $\mathbb{A}$. More precisely, for every n the following equivalence relation on n-tuples of atoms has finitely many equivalence classes:

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

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

Example 13. 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, and there are finitely many equality types for every fixed n. 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. The integers with order $(\mathbb{Z}, <)$ are not oligomorphic. An automorphism is a translation, as discussed in Example 10. For $n = 1$, there is only one equivalence class, but for $n = 2$ there are infinitely many equivalence classes, because the equivalence class of $(a, b) \in \mathbb{Z}^2$ is determined by the difference $a - b$. □

Example 14. Consider the following model, call it the *bipartite atoms*: a countably infinite universe equipped with an equivalence relation that has two

infinite equivalence classes. More formally, the bipartite atoms can be defined as the following model:

$$\mathbb{A} = (\mathbb{N}, E) \qquad \text{where } E(a, b) \text{ holds if and only if } a \equiv b \bmod 2.$$

The automorphisms of this structure are generated by: permutations of the even numbers, permutations of the odd numbers, and swapping the even numbers with the odd numbers. In particular, tuples

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

are equal up to atom automorphisms if and only they have the same equality types, and the same equivalence types (i.e. the same coordinates are equivalent with respect to the equivalence relation with two equivalence classes). Since there are finitely many possibilities for every choice of n, it follows that the bipartite 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 3.3 (Orbits)**.** For a tuple of atoms $\bar{a}$, define $\bar{a}$-equivalence to be the following equivalence relation on sets with atoms:

$$x \overset{\bar{a}}{\sim} y \qquad \text{iff} \qquad x = \pi(y) \text{ for some } \bar{a}\text{-automorphism } \pi.$$

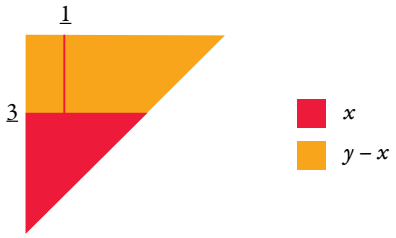
An *$\bar{a}$-orbit* is an equivalence class of this equivalence relation. When $\bar{a}$ is the empty tuple of atoms, we talk about *equivariant orbits*.

By definition of supports, a set x is supported by $\bar{a}$ if and only membership in x is invariant under $\bar{a}$-equivalence; in other words, x 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. The key point is that even though the orbits get smaller, the number of orbits never goes from finite to infinite.

Example 15. Consider the atoms $(\mathbb{Q}, <)$, and let

$$x = \{(a, b) : a, b \in \mathbb{A} \text{ where } a < b \wedge (a = \underline{1} \vee b < \underline{3})\}.$$

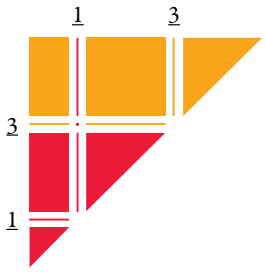
Here is a picture:



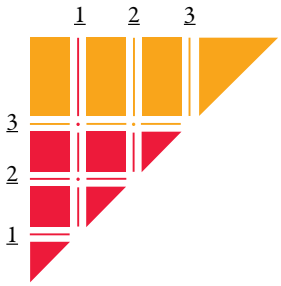
This set is contained in a single equivariant orbit

$$y = \{(a, b) : a, b \in \mathbb{A} \text{ where } a < b\}.$$

The set x is not equivariant, and therefore it does not exhaust the entire orbit. If we increase the support to $\underline{13}$, the partition of y into orbits becomes fine enough so that x becomes a union of orbits, and this union is finite:



We can further increase the support to $\underline{123}$, which further refines the partition into orbits, but still leaves finitely many of them, as in the following picture:



□

The three scenarios described in the above example turn out to be equivalent, as stated in the following theorem.

Theorem 3.4. *Assume that the atoms $\mathbb{A}$ are oligomorphic. For every set with atoms x, the following conditions are equivalent:*

(1) *x is contained in a finite union of equivariant orbits.*
(2) *x is a finite union of $\bar{a}$-orbits for some atom tuple $\bar{a}$;*
(3) *for every atom tuple $\bar{a}$ that supports x, x is a finite union of $\bar{a}$-orbits.*

A set with atoms which satisfies any of the above conditions is called *orbit-finite*. We do not talk about orbit-finiteness when the atoms are not oligomorphic. Exercise 45 shows how the conditions in the above theorem are not equivalent in the non-oligomorphic structure $(\mathbb{Z}, <)$.

The first observation used to prove the theorem is that in the definition of oligomorphism, we could have talked about $\bar{a}$-orbits instead of equivariant orbits, and nothing would change.

Lemma 3.5. *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 3.4 is true for subsets of $\mathbb{A}^n$, because every finitely supported subset of $\mathbb{A}^n$ satisfies all three conditions in the theorem. The following lemma, which does not need the assumption on oligomorphism, allows us to lift results from sets of tuples of atoms to other kinds of sets with atoms.

Lemma 3.6. *Let x be a set with atoms that is a single equivariant orbit. There is some $n \in \{1, 2, \ldots\}$ and an equivariant function*

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

such that x is equal to the image, under f, of some equivariant orbit $y \subseteq \mathbb{A}^n$.

Proof Choose some $x_0 \in x$. Because the finite support condition is hereditary for sets with atoms, there is some tuple of atoms $\bar{b}$ which supports x_0. Consider the equivariant orbit of the pair $(\bar{b}, x_0)$, i.e. the set

$$f = \{\pi(\bar{b}, x_0) : \pi \text{ is an atom automorphism}\}.$$

By Exercise 37, every atom automorphisms π, σ satisfy

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

and therefore f is a function. Because f is equivariant, the image of the equivariant orbit of x_0 is equal to the equivaraint of the image $f(x_0)$, see Exercise 40, thus proving the lemma. □

Using the two above results, we finish the proof of Theorem 3.4.

Proof of Theorem 3.4

- (2) implies (1). Suppose that x is a finite union of $\bar{a}$-orbits for some atom tuple $\bar{a}$. Increasing the support makes an orbit smaller, and therefore every $\bar{a}$-orbit is contained in some equivariant orbit. It follows that x is contained in a finite union of equivariant orbits.
- (1) implies (3). Suppose that x is contained in a finite union $x_1 \cup \cdots \cup x_n$ of equivariant orbits, and let $\bar{a}$ be an atom tuple that supports x. Let $i \in \{1, \ldots, n\}$. Apply Lemma 3.6 to x_i yielding an equivariant function

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

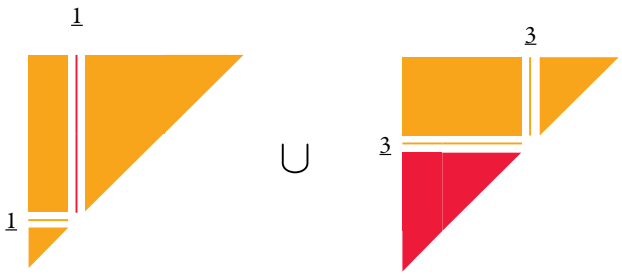
  and some equivariant orbit $y \subseteq \mathbb{A}^n$ such that $x_i = f(y)$. By Lemma 3.5, the set y splits into finitely many $\bar{a}$-orbits. Under an equivariant function, the image of an $\bar{a}$-orbit is also a $\bar{a}$-orbit, and therefore x_i is a finite union of $\bar{a}$-orbits. Since this was true for each of the finitely many x_i that partition x, it follows that x itself is a finite union of $\bar{a}$-orbits.
- (3) implies (2). Suppose that x is a finite union of $\bar{a}$-orbits for every $\bar{a}$ that supports x. To prove (2), we need to show that this is true for some $\bar{a}$, which boils down to saying that there is some $\bar{a}$-tuple that supports x, which follows from the assumption that x is a set with atoms.

□

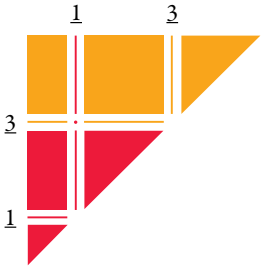
Example 16. If the atoms are oligomorphic, then the set $\mathbb{A}$ of all atoms is orbit-finite, and the same is true for $\mathbb{A}^n$, by definition of oligomorphism. If the atoms are the equality atoms or the rational numbers with order $(\mathbb{Q}, <)$, then the set $\mathbb{A}$ is even a single equivariant orbit. For the equality atoms, the number of equivariant orbits in $\mathbb{A}^n$ is the number of equality types on n variables, in particular this number is exponential in n. There are oligomorphic atom structures with more than one equivariant orbits of atoms. An example is two disjoint copies of the equality atoms, one coloured red (a unary predicate) and the other not red. □

The following example shows that it is not clear how to count orbits in an orbit-finite set that is not equivariant. We discuss counting orbits in more detail in Section 5.4 and also in Chapter 9.

Example 17. Consider the set from Example 15. This set is the union



of two sets, which are a $\underline{1}$-orbit and a $\underline{3}$-orbit, respectively. In this sense, the set has two orbits. If we want to use a common support, then we get a decomposition with seven $\underline{13}$-orbits:



□

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

Example 18. Consider the equality atoms. For $n \in \mathbb{N}$, we write $\mathbb{A}^{(n)}$ for the set of non-repeating n-tuples of atoms. 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, when 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 3.6, 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 3.7. *Every $\bar{a}$-supported orbit-finite set admits a $\bar{a}$-supported bijection with finite union of sets of the form*

$$\mathbb{A}^n_{/\sim} \quad \textit{where } n \in \mathbb{N} \textit{ and } \sim \textit{ is a } \bar{a}\textit{-supported partial equivalence relation on } \mathbb{A}^n.$$

Proof It is enough to prove the lemma for sets which are a single $\bar{a}$-orbit. Let then x be a single $\bar{a}$-orbit. Apply Lemma 3.6 to x_i, yielding an equivariant function

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

such that the image of f 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 $\sim$ is in bijective correspondence with the quotient $\mathbb{A}^n_{/\sim}$. □

If $\mathbb{A}$ has at least two elements, then the finite union in Theorem 3.7 is not needed. Indeed, under 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.

Closure properties. Orbit-finiteness has some of the same closure properties as finiteness.

Lemma 3.8. *Assume that the atoms are oligomorphic. Orbit-finite sets are closed under binary union, binary product, finitely supported subsets and images under finitely supported functions.*

Proof Binary union is immediate, and finitely supported subsets are immediate in view of Condition (1) of Theorem 3.4. Consider below the remaining cases.

- **Images under finitely supported functions.** Let $f : X \to Y$ be a finitely supported function and let $X_0 \subseteq X$ be orbit-finite. Let $\bar{a}$ be a tuple of atoms which supports both f and X_0. It is not difficult to see that whenever $x, y \in X_0$ are in the same $\bar{a}$-orbit, then also their images $f(x), f(y)$ are in the same $\bar{a}$-orbit. It follows that the number of $\bar{a}$-orbits in the image $f(X_0)$ is at most as big as the number of $\bar{a}$-orbits in X_0, and therefore finite.
- **Binary products.** By condition (1) of Theorem 3.4, it suffices to show that the product $X_1 \times X_2$ of two equivariant orbits has finitely many equivariant orbits. Apply Lemma 3.6, 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.

□

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 45. Show that in the atoms $(\mathbb{Z}, <)$, all three conditions in Theorem 3.4 are pairwise non-equivalent.

Exercise 46. 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 47. Show the following converse of Theorem 3.4: if the atoms have finitely many equivariant orbits and the conditions 1 and 3 in the statement of the theorem are equivalent for every set with atoms X, then the atoms are oligomorphic.

Exercise 48. Assume that the atoms are oligomorphic. Show that in an orbit-finite set, for every atom tuple $\bar{a}$ there are finitely many elements supported by $\bar{a}$.

Exercise 49. Show a counterexample, in the equality atoms, to the converse implication from Exercise 48. In other words, show a set which is not orbit-finite, but where every tuple of atoms supports finitely many elements.

Exercise 50. 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 51. Show that Exercise 50 fails in some oligomorphic atoms.

Exercise 52. 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 53. 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 54. 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 55. Show that in the equality atoms (actually, under any oligomophic atoms), every orbit-finite is Dedekind finite, i.e. does not admit a finitely supported bijection with a proper subset of itself.

Exercise 56. Show that in the equality atoms, there is a set that is not orbit-finite, but Dedekind finite in the sense from Exercise 55.

Exercise 57. 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 58. Call a family $\mathcal{X}$ of sets *uniformly supported* 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 59. 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 60. 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.

Bibliographic notes for Section 2. The idea to have a model of set theory with atoms (also known as *ur-elements*) originates from the work of Frankel in 1922, further developed by Mostowski in the 1930s. At that time, these models were used to prove independence of the axiom of choice, and other axioms. In computer science, atoms were rediscovered by Gabbay and Pitts in Gabbay and Pitts (2002), under the name *nominal sets*, as a formalism for modeling 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).

The notion of oligomorphism made its appearance in model theory in 1959, thanks to a theorem proved independently by Engeler, Ryll-Nardzewski and Svenonius: for a countable structure, being oligomorphic is equivalent to having an ω-categorical theory. One implication of this theorem will be used later, in Lemma 4.7.

To the author's best knowledge, the notion of orbit finiteness was first introduced in Bojanczyk (2011), which is the conference version of Bojańczyk (2013). The purpose of the papers Bojanczyk (2011); Bojańczyk (2013) was to find a model of monoids which would capture data values, and also admit minimisation (also known as syntactic monoids).

Exercise 58 is inspired by (Pitts, 2013, Section 5.5.). Exercise 55 is inspired by Blass (2013).

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 using some 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 relatively simple sets, e.g. representing $\mathbb{A}^n$ requires enumerating all of the exponentially many orbits.

In this chapter, with study a different 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 show that if the atoms are oligomorphic, then the set-builder expressions coincide with hereditarily orbit-finite sets.

Decidability assumptions. To describe sets with atoms, we use first-order formulas. To make the representations effective, we use the following assumption.

Definition 4.1. We say that a structure $\mathbb{A}$ has *decidable first-order theory with constants* if elements of the universe and predicates in the vocabulary can be represented in a finite way so that the following problem is decidable:

- **Input** A sentence of first-order logic, using the vocabulary of $\mathbb{A}$, with arbitrary atoms allowed as constants.
- **Question.** Is the sentence true in $\mathbb{A}$?

A weaker assumption is that the atoms have a decidable first-order theory, without allowing constants in the formulas. The weaker assumption holds for the real field $(\mathbb{R}, +, \times)$, which clearly fails the assumption on representing elements. Nevertheless, the assumption on representing constants is not very restrictive. For example, the field of real algebraic numbers has the same first-order theory as the field of reals, but its elements can be represented in a finite way.

Example 19. Examples of structures with a decidable first-order theory with constants 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[1]. Presburger and Skolem arithmetic are not oligomorphic □

All oligomorphic structures in this book have a decidable first-order theory with constants. As shown in the above example, the converse is not true. For many results in this chapter, we only assume decidable first-order theory with constants, without assuming oligomorphism.

4.1 Hereditarily definable sets

Fix a logical structure $\mathbb{A}$ for the atoms, not necessarily oligomorphic. Hereditarily definable sets are those which can be defined using set builder notation. We begin with some examples of this notation. Here is the family of all subsets of $\mathbb{A}$ that miss at most one atom:

$$\{\{a : \text{ for } a \in \mathbb{A}\}\} \cup \{\{b : \text{ for } b \in \mathbb{A} \text{ such that } b \neq a\} : \text{ for } a \in \mathbb{A}\}$$

Another example, this time in the atoms $(\mathbb{Q}, <)$, is the set of all nonempty closed intervals that do not contain $\underline{1}$:

$$\{\{c : \text{ for } c \in \mathbb{A} \text{ such that } a < c < b\} : \text{ for } a, b \in \mathbb{A} \text{ such that } (a < b < \underline{1}) \vee (\underline{1} < a < b)\}$$

[1] Presburger's original proof, Presburger (1929), used quantifier elimination. For an approach that uses automata, see (Thomas, 1990, pages 399) for Presburger arithmetic and Blumensath and Gradel (2000) for Skolem arithmetic.

We now present the formal definition of set builder expressions.

Definition 4.2 (Set builder expressions)**.** Fix an infinite set of variables, which range over atoms. For a structure $\mathbb{A}$, define the *set builder expressions over* $\mathbb{A}$ as follows by structural induction (this notion depends only on the vocabulary of $\mathbb{A}$):

(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 $\mathbb{A}$, possibly with free variables. Then

$$\{\alpha(\bar{x}\bar{y}) : \text{ for } \overbrace{x_1, \ldots, x_n}^{\bar{x}} \in \mathbb{A} \text{ such that } \varphi(\bar{x}\bar{y})\}$$

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 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.

If α is a set builder expression with free variables $x_1, \ldots, x_n$, define

$$\alpha(a_1, \ldots, a_n) \qquad \text{for } a_1, \ldots, a_n \in \mathbb{A}$$

to be the set with atoms obtained in the natural way. By induction on the size of α one shows that

$$(a_1, \ldots, a_n) \qquad \mapsto \qquad \alpha(a_1, \ldots, a_n)$$

is an equivariant function from atom tuples to sets with atoms.

Definition 4.3 (Hereditarily definable sets)**.** Let $\mathbb{A}$ be a structure, not necessarily oligomorphic. A *hereditarily definable set* over $\mathbb{A}$ is any set of the form $\alpha(\bar{a})$ where α is a set builder expression and $\bar{a}$ is a valuation of its free variables.

By abuse of notation we inline atom constants into the guards, writing

$$\{x : \text{ for } x \in \mathbb{A} \text{ such that } x \neq \underline{1}\}$$

instead of the formally correct

$$\{x : \text{ for } x \in \mathbb{A} \text{ such that } x \neq y\}(\underline{1}).$$

Example 20. [Singletons] In a set expression, the number n of bound variables

might be zero and the guard might be "true", in which case the expression describes a singleton

$$\{\alpha(\bar{y})\}.$$

We write $\{\alpha(\bar{x}), \beta(\bar{y})\}$ as syntactic sugar for the union of two singletons. Likewise, we write $(\alpha(\bar{x}), \beta(\bar{x}))$ as syntactic sugar for Kuratowski pairs. □

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 set builder expressions reduce to the first-order theory of the atom structure.

Lemma 4.4 (Symbol Pushing Lemma)**.** *Let α, β be set builder expressions with free variables contained in $\bar{x}$. There is a formula $\varphi(\bar{x})$ of first-order logic such that*

$$\mathbb{A} \models \varphi(\bar{a}) \quad \textit{iff} \quad \alpha(\bar{a}) \subseteq \beta(\bar{a})$$

holds for every atom tuple $\bar{a}$ of same length as $\bar{x}$. Likewise for $\in$ instead of $\subseteq$.

Proof Induction on the size of the set builder expressions.

- *Inclusion* $\subseteq$. The interesting case is when the left side is a set expression, i.e. we want to define those tuples which satisfy the inclusion

  $$\{\alpha(\bar{x}\bar{y}) : \text{ for } \bar{y} \in \mathbb{A} \text{ such that } \psi(\bar{x}\bar{y})\} \quad \subseteq \quad \beta(\bar{x})$$

  The inclusion is true for a tuple $\bar{x}$ if and only if the following formula of first-order logic is satisfied:

  $$\forall \bar{y}\, \psi(\bar{x}\bar{y}) \;\Rightarrow\; \underbrace{\alpha(\bar{x}\bar{y}) \in \beta(\bar{x})}_{\text{induction assumption}} .$$

  This can be formalised in first-order logic, using the induction assumption to get a constraint on the variables $\bar{x}\bar{y}$ which makes the membership true.
- *Membership* $\in$. The interesting case is when the right side is a set expression:

  $$\alpha \quad \in \quad \{\beta(\bar{x}\bar{y}) : \text{ for } \bar{y} \in \mathbb{A} \text{ such that } \psi(\bar{x}\bar{y})\}$$

  This membership is true if and only if

  $$\exists \bar{y}\, \psi(\bar{x}\bar{y}) \;\wedge\; \underbrace{\alpha(\bar{x}\bar{y}) = \beta(\bar{x})}_{\text{induction assumption}} .$$

  The equality underlined above is expressed using the induction assumption as follows. If α and β are variables, then the constraint for $\alpha = \beta$ is that the

corresponding variables are equal. If one of α, β is a variable and the other is not, then the constraint is unsatisfiable. Otherwise, if both α and β are set builder expressions, then equality is the same as inclusion both ways, which can be described using the induction assumption.

□

Corollary 4.5. *If the atoms have decidable first-order theory with constants, then membership, inclusion and equality are decidable for hereditarily definable sets.*

The constants in the assumption of Corollary 4.5 are used to deal with definable sets that use atom constants. Without atom constants – i.e. for set builder expressions without free variables – it is enough to consider structures with a decidable first-order theory but where the elements are not necessarily representable, e.g. the real field $(\mathbb{R}, +, \times)$.

4.2 Hereditarily orbit-finite sets

In this section we show that if the atoms are oligomorphic, then hereditarily orbit-finite sets described at the beginning of Section 4 are the same as the hereditarily definable sets described in Section 4.1.

Theorem 4.6. *Assume that the atoms are countable and oligomorphic. A set is hereditarily definable if and only if it is hereditarily orbit-finite.*

The key part of the above theorem is the following lemma, which uses the ideas of Ryll-Nardzewski, Engeler and Svenonius about countable oligomorphic being ω-categorical.

Lemma 4.7. *In a countable oligomorphic model, a subset $X \subseteq \mathbb{A}^n$ is equivariant if and only if it is first-order definable.*

Proof We use Ehrenfeucht-Fraïssé games. Consider the following game, which is parameterised by two finite tuples $\bar{a}, \bar{b}$ of the same length and a number of rounds $k \in \mathbb{N}$. There game is played by two players, called Spoiler and Duplicator. In each round:

- Spoiler chooses one of the tuples and extends it with an atom.
- Duplicator responds by extending the other tuple with an atom.

Duplicator wins the game if after playing k rounds, the (extended) tuples satisfy the same quantifier-free formulas, otherwise Spoiler wins.

There is also a variant of the game with ω rounds. In the ω variant, player Spoiler wins if at some point – after finitely many rounds – the two extended tuples do not satisfy the same quantifier-free formulas. If this never happens, Duplicator wins.

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

Claim 4.8. *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;*
(2) *Duplicator has a winning strategy in the k-round game for every $k \in \mathbb{N}$;*
(3) *Duplicator has a winning strategy in the ω-round;*
(4) *are in the same equivariant orbit.*

Proof

- *1 implies 2.* This is (half of) the classical Ehrenfeucht-Fraïssé theorem[2], which says that if two tuples satisfy the same formulas of quantifier rank at most k, then 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 $k \in \mathbb{N}$, 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 element, and the other tuple is $\bar{b}$. If elements 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 König lemma to show that Spoiler wins the infinite round game if and only if for some k he wins the k-round game.
- *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. 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:

[2] See (Hodges, 1993, Section 3.2)

- in even-numbered rounds, extend the $\bar{a}$ tuple with the least (according to the enumeration) atom that does not appear in it;
- in odd-numbered rounds, do the same for the $\bar{b}$ tuple.

Suppose that Duplicator responds to the above strategy with a winning strategy. In the resulting play, we get two infinite sequences

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

such that for every $i \in \mathbb{N}$, the tuples $(a_1, \dots, a_i)$ and $(b_1, \dots, b_i)$ satisfy the same quantifier-free formulas. By the choice of Spoiler's strategy, every atom appears in the sequence $a_1, a_2, \dots$ and every atom appears in the sequence $b_1, b_2, \dots$. 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.

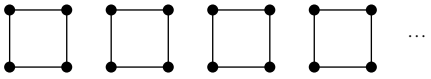
This completes the proof of the claim, and therefore also of the lemma. □

□

Example 21. Consider the atoms $(\mathbb{N}, =)$. It is not difficult to see that every equivariant orbit of n-tuples of atoms is determined by its equality type, and therefore it is first-order definable, even without quantifiers. The same works for $(\mathbb{Q}, <)$, except that order types are used. □

In the above example, even quantifier-free formulas were sufficient to define equivariant sets of atom tuples. The reason is that the atom structures in these examples satisfy a property stronger than oligomorphism, which will be discussed in Section 7, namely they are *homogeneous*. Not all oligomorphic structures are homogeneous, as shown in the following example of an oligomorphic structure without quantifer elimination.

Example 22. Let $\mathbb{A}$ be the undirected graph which consists of a countably infinite disjoint union of cycles of length 4:



It is not difficult to show that 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 clearly definable in first-order logic, but not without quantifiers. □

Corollary 4.9. *Suppose that the atoms are a countable oligomorphic structure. If a set of n-tuples of atoms is supported by a tuple of atoms $\bar{a}$, then it is definable by a first-order formula with n free variables and constants from $\bar{a}$.*

Proof Consider a set $X \subseteq \mathbb{A}^n$ that is supported by a tuple $\bar{a}$ of dimension k. 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 4.7 it is defined by a formula of first-order logic φ. A tuple $\bar{b}$ belongs to X if and only if it satisfies $\varphi(\bar{a}\bar{b})$. □

Before proving Theorem 4.6, let us note a further corollary of the above results. Recall Theorem 3.7, which says that every one-orbit set admits a finitely supported bijection to a quotient of $\mathbb{A}^n$ under some finitely supported partial equivalence relation $\sim$. If we view $\sim$ as a unary relation on $\mathbb{A}^{2n}$ and apply Corollary 4.9, we see that $\sim$ is definable by a formula of first-order logic with $2n$ free variables, possibly using constants from the atoms.

Proof of Theorem 4.6 Let us begin with the left-to-right implication. We show that for every set builder expression α, its semantics is an equivariant function that inputs tuples of atoms and outputs hereditarily orbit-finite sets. Equivariance is immediate, since truth of first-order formulas is invariant under automorphisms. To show that the outputs are hereditarily orbit-finite, we use induction on the size of α. Consider the interesting case in the induction step, which is when α is a set builder expression of the form

$$\beta(\bar{x}) = \{\alpha(\bar{x}\bar{y}) : \text{ for } \bar{y} \text{ such that } \varphi(\bar{x}\bar{y})\}.$$

For a tuple of atoms $\bar{a}$ which evaluates the free variables, we see that $\beta(\bar{a})$ is the image under α of the set X of $\bar{x}\bar{y}$-tuples of atoms that begin with $\bar{a}$ and satisfy the formula φ. The set X is a set of atoms tuples of fixed dimension which is supported by $\bar{a}$, and therefore it is orbit-finite thanks to oligomorphism. By induction assumption α is an equivariant function from atom tuples to hereditarily orbit-finite sets. From Fact 3.8 it follows that hereditarily orbit-finite sets are closed under images of finitely supported functions which output only hereditarily orbit-finite sets, and thus the result follows.

We now turn to the right-to-left implication in Theorem 4.6. The proof is by induction on the rank in the cumulative hierarchy. Let then X be a hereditarily orbit-finite set supported by $\bar{a}$. Since definable sets are closed under finite unions, 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, $x = \alpha(\bar{a}\bar{b})$ for some set builder expression α. Therefore,

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

Since α is equivariant, it commutes with atom automorphisms, and thus

$$X = \{\alpha(\bar{a}\bar{c}) : \bar{c} \in Y\} \qquad \text{where } Y = \{\pi(\bar{a}\bar{b}) : \pi \text{ is an } \bar{a}\text{-automorphism}\}$$

To show that X is definable, it suffices to show that Y can be defined by a first-order formula over the atoms, with free variables $\bar{x}\bar{y}$ and constants from $\bar{a}$. This follows from the observation that Y is supported by $\bar{a}$ and Corollary 4.9. □

Bibliographic notes for Section 2. Definable sets are based on set builder notation from set theory. The idea to use set builder notation as a way of representing hereditarily orbit finite sets (Theorem 4.6) originates from the programming language LOIS (*Looping over Infinite Sets*) Kopczynski and Toruńczyk (2016, 2017). Earlier papers on sets with atoms, such as Bojańczyk et al. (2014); Bojanczyk et al. (2012), used different representations for orbit finite sets based on least supports (see Section 6). The name "definable" is inspired by terminology from model theory, where "definable set" means a set of tuples in a model that can be defined by a first-order formula with free variables, see (Hodges, 1993, historical remarks on p. 82). As mentioned in the bibliographical notes for Section 3, Lemma 4.7 is part of a theorem proved independently proved by Engeler, Ryll-Nardzewski and Svenonius, see (Hodges, 1993, Theorem 7.3.1).

The idea to use orbit-finite sets (equivalently, definable sets) to model register automata, as is done in Excercises **??**, 68 and **??** is from Bojanczyk (2011); Bojanczyk et al. (2011); Bojańczyk et al. (2014), and was one of the main motivations in developing the theory of sets with atoms. We will come back to automata in Section **??**.

5
Case studies

In this section, we show how natural algorithms on finite objects – such as graph reachability, or automaton nonemptiness – 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:

- how orbit-finiteness has some of the advantages of finiteness, in particular how such sets can be transformed and searched by algorithms;
- how the semantic approach (orbit-finite sets) can be useful;
- how the syntactic approach (hereditarily definable sets) can be useful;
- how orbit-finite automata generalise the automata models from Part I, and why this generalisation is useful;
- how a computable Ryll-Nardzewski function is useful.

5.1 Graph reachability

We begin our case studies with directed graph reachability.

- *Input.* A directed graph (V, E), and source and target subsets $S, T \subseteq V$.
- *Question.* Is there a path from some source to some target?

Suppose that the directed graph, the source and targets are hereditarily definable over some atoms $\mathbb{A}$. In other words, suppose that the instance of the problem

$$(V, E, S, T),$$

is a tuple that is a hereditarily definable set [1]. If the atoms can be represented in a finite way, then the instance can be represented in a finite way, and therefore it makes sense to talk about reachability for hereditarily definable graphs as a decision problem.

Theorem 5.1. *Reachability for hereditarily definable graphs is decidable, assuming that the atoms are oligomorphic and have decidable first-order theory with constants.*

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

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

Hereditarily definable sets are closed – in a computable way – under taking singletons, set union and images under binary relations. This is not hard to see, but we present a more general explanation for such closure properties later in this section, using a data structure called the set structure.

Consider the following algorithm. For each $n = 0, 1, 2, \ldots$, compute the set V_n, until a fixpoint is reached, i.e. some n such that $V_n = V_{n+1}$. If the fixpoint is reached in finitely many steps, then it is easily seen to be the set of vertices reachable from at least one source. 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. One can show by induction that this tuple supports also 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 hereditarily definable, it is orbit-finite by Theorem 4.6. By Theorem 3.4, 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}. □

[1] Since hereditarily definable sets are closed under pairing, this is the same as saying that each of V, E, S and T are hereditarily definable sets.

The assumption that the atoms are effective was used to compute the sets V_n, while the assumption that they are oligomorphic was used in the claim about the fixpoint. □

The above proof illustrates how it is useful to have both syntactic descriptions (hereditarily definable sets) 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.

In the proof, we announced to general methods – the equivariance principle for reasoning about supports and the set structure for transforming hereditarily definable sets. We describe these methods below.

Equivariance principle. The equivariance principle is used to prove statements such as: "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". 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.

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

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

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

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

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 $\mathsf{set}\mathbb{A}$. In particular, any set of pairs in $\mathsf{set}\mathbb{A}$ that is defined by a formula of first-order logic is going to be invariant under (lifting to sets with atoms) of automorphisms of $\mathbb{A}$. □

The following example applies the equivariance principle to sets of reachable vertices in a graph.

Example 23. In the proof of Theorem 5.1, we said that for every n, the set of vertices reachable from a source vertex is supported by whatever supports the source vertices and the set of edges in the graph. Let us prove this claim using the equivariance principle. Recall that pairs are defined using Kuratowski pairing

$$(x, y) = \{\{x\}, \{x, y\}\}.$$

The point of Kuratowski pairing is that pairing and projections can be done in the language of set theory. Consider pairing. The following formula expresses that p is the (set representing) the ordered pair (x, y):

$$\forall z\, z \in p \;\Rightarrow\; (\overbrace{z = \{x\}}^{\substack{x \text{ is the unique} \\ \text{element of } z}} \;\vee\; \overbrace{z = \{x\} \cup \{y\}}^{\substack{z \text{ is the smallest set that} \\ \text{contains } \{x\} \text{ and } \{y\}}}). \tag{5.1}$$

Using the above formula, we can write say that x is the first coordinate of the pair p:

$$\exists y \quad \overbrace{p = (x, y)}^{\text{expressed in (5.1)}} \;.$$

Once we have pairing, we can easily say that Y is the image of X under a binary relation E:

$$\forall y\, y \in Y \;\Leftrightarrow\; (\exists x \in X\, (x, y) \in E)$$

Applying the equivariance principle to the above formula, we see that any tuple supporting X and E will also support Y. □

It is not hard to show directly that reachability sets are supported by whatever supports the graph and the set of initial vertices. The point of the above example is to show how supports are inherited across any definition that uses the language of set theory. In the example we carefully work out the details of how vertices can be paired (to get edges) and un-paired in the language of set theory. Later in the book, we will no longer do such an analysis, 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 24. When applying the Equivariance Principle, one needs to remember that the structure $\mathsf{set}\mathbb{A}$ only talks about finitely supported sets. 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 there is 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 62) but it is false in general (see Exercise 61). □

The set structure. In the proof of Theorem 5.1, we claimed that certain natural operations on hereditarily definable sets – such as Boolean operations, images under binary relations or functions – are effective, i.e. there are algorithms which implement these operations. Another example of such an operation, which will be used in testing emptiness of an automaton, is taking a transition relation

$$\delta \subseteq Q \times \Sigma \times Q$$

and projecting away one of the coordinates, yielding for example

$$\{(q, p) : (q, a, p) \in \delta \text{ for some } a \in \Sigma\} \subseteq Q \times Q.$$

To deal with such transformations, we model a hereditarily definable set as a logical structure, and then show that operations defined in terms of this logic structure can be implemented using algorithms.

Definition 5.4 (Set Structure)**.** Let x be hereditarily definable. Define x_* to be the set which contains x, the elements of x, their elements, and so on. In other words,

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

Define the *set structure of* x to be the logical structure

$$(x_*, \in, \emptyset).$$

In other words, the set structure of x is the same as the structure of set theory considered in the Equivariance Principle, restricted to x_*. For example, suppose consider a set with atoms G which is a directed graph

$$G = (V, E, s, T), \qquad E \subseteq V \times V,\ s \in V,\ T \subseteq V$$

with tuples modelled as usual using the Kuratowski encoding. Because pairs

can be manipulated using the language of set theory – see Example 23 – one can write for every $n \in \{0, 1, \ldots\}$ a sentence $\varphi(v)$ of first-order logic that uses $\in$ such that

$$(G_*, \in \emptyset) \models \varphi_n(v) \qquad \text{iff} \qquad \text{there is a path from } s \text{ to } v \in V \text{ of length at most } n.$$

In other words, the formula φ_n defines the set V_n that was considered in the proof of Theorem 5.1. The following lemma tells us that properties definable in terms of the set structure – e.g. the definition of V_n – will be hereditarily definable, in an effective way.

Lemma 5.5 (Definable Relation Lemma)**.** *Let x be hereditarily definable over atoms $\mathbb{A}$, and let $\varphi(y_1, \ldots, y_n)$ be a first-order formula over $(x_*, \in, \emptyset)$, which defines an n-ary relation R on x_*. Then R is also hereditarily definable, and a set-builder expression for R can be computed given a set-builder expression for x.*

Proof An extension of the Symbol Pushing Lemma. Suppose that

$$x = \alpha(\bar{a})$$

for some set-builder expression and some atom tuple $\bar{a}$. Let Γ be the set of set-builder expressions (without valuations) that appear as sub-expressions of α. By abuse of notation, if β is a set-builder expression, we write $\mathbb{A}^\beta$ for all tuples of atoms that evaluate the free variables of β. Every set in x_* is of the form

$$\beta(\bar{b}) \qquad \text{for some } \beta \in \Gamma \text{ and } \bar{b} \in \mathbb{A}^\beta.$$

Note that the converse is not necessarily true, i.e. not every $\beta(\bar{b})$ necessarily describes an element of x_*, because there might be some constraints on the tuple $\bar{b}$ that are needed to make $\beta(\bar{b}) \in x_*$ true. But these constraints can be formulated in first-order logic as stated in the following claim.

Claim 5.6. *For every $\beta \in \Gamma$ there is a first-order formula ψ over the vocabulary of the atoms extended with constants from $\bar{a}$ such that*

$$\mathbb{A} \models \psi(\bar{b}) \quad \textit{iff} \quad \beta(\bar{b}) \in x_* \qquad \textit{for every atom tuple } \bar{b} \in \mathbb{A}^\beta.$$

Proof Induction on β, starting with bigger expressions and proceeding to smaller expressions. In the induction step, the Symbol Pushing Lemma is used. □

The statement of the lemma follows immediately from the following claim.

Claim 5.7. *Let $\varphi(y_1, \ldots, y_n)$ be as in the lemma, and let $\beta_1, \ldots, \beta_n \in \Gamma$. There*

is a formula ψ of first-order logic over the vocabulary of the atoms extended with constants from $\bar{a}$ such that

$$\mathbb{A} \models \psi(\bar{b}_1, \ldots, \bar{b}_n) \quad \textit{iff} \quad (x_*, \in, \emptyset) \models \varphi(\beta_1(\bar{b}_1), \ldots, \beta_n(\bar{b}_n))$$

holds for every atom tuples $\bar{b}_1 \in \mathbb{A}^{\beta_1}, \ldots, \bar{b}_n \in \mathbb{A}^{\beta_n}$.

Proof Induction on the size of φ. The Symbol Pushing Lemma and Claim 5.6 take care of the induction base, which is when φ is $y \in z$ or $y = \emptyset$ for some variables y, z. The induction step is straightforward as well. □

All the constructions in the above proof are clearly computable. □

Exercises

Exercise 61. Assume the equality atoms. Show a graph which has an infinite path, but does not have any infinite finitely supported path.

Exercise 62. 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) there is a path from s to some $t \in T$ which is on a cycle.

Find an atom structure where the following two conditions are equivalent, and also an atom structure where only the implication (1) $\Leftarrow$ (2) is true.

Exercise 63. Show that under the assumptions of Theorem 5.1, there is an algorithm that checks if condition (1) of Exercise 62 is satisfied, assuming that the graph, source and targets are all hereditarily definable. Likewise for condition (2).

Exercise 64. 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; otherwise the play goes on forever and player 1

wins. Show that under the assumptions of Theorem 5.1, alternating reachability is decidable for instances that are hereditarily definable.

Exercise 65. Consider the atoms $(\mathbb{N}, +1)$, which are not oligomorphic. Show that graph reachability is undecidable.

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 the same as the register automata from Section 1.

Orbit-finite automata. The definition of an nondeterministic orbit-finite automaton is the same as the definition of a nondeterministic finite automaton, except the word "finite set" is replaced by "orbit-finite set with atoms".

Definition 5.8. A *nondeterministic orbit-finite automaton* (over an 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.

Hereditarily definable automata. In Definition 5.8, we use orbit-finite sets. Not much would change if we would use herditarily definable nondeterministic automata. Define an *isomorphism* between automata

$$\mathcal{A}_i = (Q_i, \Sigma_i, I_i, F_i, \delta_i) \qquad \text{for } i \in \{1, 2\}$$

to be a pair of bijections $f : Q_1 \to Q_2$ and $g : \Sigma_1 \to \Sigma_2$ which are consistent with the transition relations and accepting/final states in the natural way. By Theorem 3.7, every orbit-finite set admits a finitely supported bijection to a hereditarily definable set (even of a very simple form). Therefore, every orbit-finite automaton is isomorphic to one that is hereditarily definable 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 definable automata.

Relationship with register automata. Consider the equality atoms $(\mathbb{N}, =)$. Recall that in Part I of this book, we used register automata to recognise properties of data words. A data word can be seen as a word over a special kind of orbit-finite alphabet, namely one of the form $\Sigma \times \mathbb{A}$, where Σ is some finite (not orbit-finite) equivariant set of labels. The following theorem shows that, as far as expressive power is concerned, nondeterministic orbit-finite automata are the same as register automata. A similar result, Theorem 6.5, is true for deterministic automata, but it will only be proved in Section 6.3, when the necessary tools are available.

Theorem 5.9. *Consider the equality atoms $\mathbb{A} = (\mathbb{N}, =)$. 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) is immediate[2]: the state space of a register automaton is easily seen to be equivariant and orbit-finite, and likewise for the other components (transitions, initial and finite states). 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 3.7, 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 uses only one location, and 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 $\mathcal{A}$. □

In view of the above theorem, one can ask what is the benefit of the more general setting of orbit-finite automata. There are three main benefits:

[2] The implication (1) $\Rightarrow$ (2) is true essentially by design, since orbit-finite sets were originally introduced to model register devices for data words Bojanczyk (2011); Bojanczyk et al. (2011).

- One can consider 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 mimimisation later in this section.
- Orbit-finite automata can consider unusual input alphabets, e.g. unordered pairs of atoms as considered in Example 25. The importance of such alphabets will be described in Section 10.

A secondary benefit is that orbit-finite automata can be used to

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, assuming that the atoms are atoms are oligomorphic and have decidable first-order theory with constants. This is because 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 Definable Relation Lemma. Other examples of positive results that generalise easily to orbit-finite automata are: elimination of ϵ-transitions (Exercise 66) or deciding if a nondeterministic automataon is deterministic/unambiguous (Exercise 67)

The negative results about register automata transfer to orbit-finite automata. A corollary of Theorem 5.9 is that nondeterministic orbit-finite automata are not closed under complement, because nondeterministic register automata are not closed under complement. Also, universality is undecidable for nondeterministic orbit-finite automata, because it is undecidable for nondeterministic register automata (Theorem 1.7).

Example 25. Consider the equality atoms. Let the input alphabet be

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

i.e. each letter is a 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 26. The automaton from the previous 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, and the other states as sets of atoms of size at most two. The transition function is defined by

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

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

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. To see the problems with miminisation for register automata, consider the following example, which is very close to Exercise 2.

Example 27. [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. More formally, the state space is

$$\underbrace{(\mathbb{A} \cup \{\bot\})}_{\text{register 1}} \times \underbrace{(\mathbb{A} \cup \{\bot\})}_{\text{register 2}} \quad \cup \quad \{\text{reject}\}$$

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. (The automaton does not use states of the form $(\bot, a)$.)

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, while with respect to our

language, the two states should be equivalent. In other words, the automaton should have states which are unordered sets of atoms of size at most two. This example shows that in order to store the minimal amount of information, registers are not always the right choice. □

The problems mentioned above go away when we move from register automata to orbit-finite automata. We begin by recalling the standard definition of Myhill-Nerode equivalence. Suppose that $L \subseteq \Sigma^*$ is a language. Define two words $w, w' \in \Sigma^*$ to be L-equivalent if for every word $v \in \Sigma^*$, the language L contains either both or none of the words wv and $w'v$. As usual, this equivalence relation is a congruence with respect to appending letters, and therefore it makes sense to consider an automaton where the states are the equivalence classes, and where the transition function is defined so that after reading a word w, the state is the equivalence class of w. This automaton is called the *syntactic automaton of L*.

Theorem 5.10. *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 a finitely supported equivalence relation. (The equivalence relation is finitely supported thanks to the Equivariance Principle.) 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. □

Exercises

Exercise 66. Show that adding ϵ-transitions does not change the expressive power of nondeterministic orbit-finite automata.

Exercise 67. 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).

Exercise 68. 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 definable nondeterministic automaton which is supported by $\bar{a}$.

Exercise 69. 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 has no control over the increase or decrease of a counter. The automaton accepts by reaching an accepting state. Show that emptiness is decidable.

Exercise 70. Assume that the atoms are oligomorphic. Show that the expressive power of nondeterministic orbit-finite automata is not changed if ϵ-transitions are allowed.

Exercise 71. 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 54.

Exercise 72. Assume the equality atoms. Show that languages recognised by nondeterministic orbit-finite automata (same for deterministic) are not closed under orbit-finite intersection.

Exercise 73. Consider 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 54.

Exercise 74. Consider the following extension of register automata to arbitrary orbit-finite alphabets: this is the special case of nondeterministic orbit-finite automata where the set of states is of the form

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

for some finite (not just orbit-finite) sets Q and R. Show that such an automaton cannot recognise the language from Example 25.

5.3 Pushdown automata and context-free grammars

In this section, we discuss orbit-finite and hereditarily definable variants of pushdown automata 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.

Definition 5.11. An *orbit-finite pushdown automaton* consists of

$$\underbrace{Q}_{\text{states}} \quad \underbrace{\Sigma}_{\substack{\text{input}\\\text{alphabet}}} \quad \underbrace{\Gamma}_{\substack{\text{stack}\\\text{alphabet}}} \quad \underbrace{q_0 \in Q}_{\text{initial state}} \quad \underbrace{\gamma_0 \in \Gamma}_{\text{initial stack}} \quad \underbrace{\delta \subseteq Q \times \overbrace{\Gamma^*}^{\text{popped}} \times \overbrace{(\Sigma \cup \epsilon)}^{\text{input}} \times Q \times \overbrace{\Gamma^*}^{\text{pushed}}}_{\text{transitions}}.$$

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 5.12. 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.

Hereditarily definable pushdown automata and context-free grammars are defined in the same way as the orbit-finite ones. As in Section 5.2, when the atoms are oligomorphic, we make little distinction between orbit-finite pushdown automata and hereditarily definable ones, because these are the same up to finitely supported isomorphisms. The following theorem says that pushdown automata are the same as context-free grammars. The theorem is phrased in terms of hereditarily definable sets. When the atoms are oligomorphic, the theorem implies that orbit-finite pushdown automata are the same as orbit-finite context-free grammars.

Theorem 5.13. *Consider atoms that are not necessarily oligomorphic. Hereditarily definable pushdown automata and hereditarily definable context-free grammars describe the same languages.*

Proof We just redo the classical constructions, which are so natural way that they easily go through with hereditarily definable sets.

- *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 hereditarily definable pushdown automaton can be converted into one of this form, without changing the recognised language, by using additional states and ϵ-transitions.

  The nonterminals in the grammar are

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

  This set is easily seen to be hereditarily definable. The language generated by a nonterminal (p, γ, q) is going to be the set of words which take the automaton from a configuration with state p and γ on top of the stack, to another configuration with state p and γ on top of the stack, such that during the run the symbol γ is not removed from the stack. The rules of the grammar are as follows, all of the sets below are hereditarily definable thanks to Definable Relation Lemma:

(1) *Transitive closure*. For every $p, q, r \in Q$ and $\gamma \in \Gamma$, we add a rule

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

(2) *Push-pop.* For every transitions

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

  we add a rule to 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

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

  we add a rule to 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.

□

When the atoms are furthermore oligomorphic, 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.

Example 28. [Pushdown automaton for palindromes.] For an orbit-finite alphabet Σ, consider the language of palindromes, i.e. those words in Σ^* 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 29. [Pushdown automaton for modified palindromes.] The automaton in Example 28 had two control states. In some cases, it might be useful to have a set Q of control states that is orbit-finite. Consider for example 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 would keep the first letter in every token on the stack. This automaton would have a stack alphabet of $\Gamma = \Sigma \times \Sigma$, and after reading letters $a_1 \cdots a_n$ its stack would be

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

This automaton would only need 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 exercise gives some motivation for studying orbit-finite pushdown automata.

Example 30. [Modelling recursive programs] Pushdown automata without atoms are sometimes used to model the behaviour of recursive programs with Boolean variables. This modelling can be extended with atoms, which means that we can also model programs that have variables ranging over orbit-finite sets. Consider the order 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.

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

□

Exercises

Exercise 75. Consider the equality atoms. We say that two sets of 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 a model of orbit-finite pushdown automata extended by one new kind of transition:

$$q \xrightarrow{\text{fresh}(a)} p,$$

where q, p are states and a is an input letter. 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 76. Consider the equality atoms and the following higher order variant orbit-finite pushdown automaton. The automaton has a stack 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 77. 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.

Bibliographic notes for Section 3. 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 5.10). To do this, one needed a representation of the state space which would: a) be 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-finite automata. The computational complexity of automata minimisation are studied in Murawski et al. (2015). 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.

Context-free languages for infinite alphabets were originally introduced in Cheng and Kaminski (1998), which contains a proof of 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). Higher-order pushdown automata for infinite alphabets, as in Exercise 76, come from (Murawski et al., 2014, Section 6).

5.4 Graph homomorphisms

In this case study, we consider graph homomorphisms[3]. One of the purposes of this case study is to introduce the following effectivity assumption on the atoms.

Definition 5.15. 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 $\mathbb{A}^n$.

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

Example 31. In the equality atoms, the number of equivariant orbits is the number of equality types, which can easily 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. However, using forcing, one can construct an oligomorphic structure which has a decidable first-order theory, but where the Ryll-Nardzewski function cannot be computed, see Schmerl (1978).

The assumption on a computable Ryll-Nardzewski function turns out to be useful when we want to compute the partition of a set into orbits. This will be the case below, when studying a decision problem about homomorphisms. We begin by defining the problem.

Definition 5.16 (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:

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

In this section, we the decision problem of determining if a homomorphism exists, given two hereditarily orbit-finite graphs. There are three variants of this decision problem, depending on the requirements for the support of the homomorphism.

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

(1) supported by a given tuple $\bar{a}$?
(2) finitely supported?
(3) not necessarily finitely supported?

The three variants are indeed different, as illustrated in the following examples. We will show in this section that Variant (1) is decidable. Variants (2) and (3) are undecidable[4]. If we modify Variant (3) to ask for a not necessarily finitely supported *isomorphism*, we get an open problem.

Example 32. [Variants (1) and (2) have different answers] Consider the equality atoms. We given a example where there exists a finitely supported homomorphism, but there is no homomorphisms supported by a given tuple $\bar{a}$. For

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

this example, $\bar{a}$ is 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 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 33. [Variants (2) and (3) have different answers] Consider the equality atoms. We given a example where there exists a homomorphism, but not any finitely supported one. Both graphs G_1 and G_2 are cliques in the sense that every two distinct edges are connected by edges in both ways, but there are no self-loops. The vertices of G_1 are $\mathbb{A}^2$ and the vertices of G_2 are $\mathbb{A}$. The homomorphisms between two graphs are exactly the injective functions

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

because mapping to 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 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 question – 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. 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 it h is supported by $\bar{a}$, the it is a union of $\bar{a}$-orbits. By oligomorphism, there are finitely many $\bar{a}$-orbits. The subtle points is that we also need to compute these orbits. This is done using the following lemma.

Lemma 5.17. *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 $V_1 \times V_n$, yielding a list of orbits. Using the Symbol Pushing Lemma, filter that list by keeping only those orbits which are contained in $V_1 \times V_2$. For every h which is a union of these orbits, use the Definable Relation Lemma to check if h is a homomorphism. It remains to prove the lemma.

Proof of Lemma 5.17 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 that the Ryll-Nardzewski function is computable.

Claim 5.18. *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 constants 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 by assumption on the atoms. It remains to find the first-order formulas. By Lemma 4.7, such formulas exist. Therefore we can use exhaustive enumeration until we find k formulas without constants which partition $\mathbb{A}^n$ into exactly k parts.
- 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. □

Let X be a hereditarily orbit-finite set. If X is given by a union set-builder expression, then we can compute the lists 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 on the list are equal. We can therefore assume that X is of the form

$$\{\beta(\bar{y}\bar{b}) : \text{ for } y_1, \ldots, y_n \in \mathbb{A} \text{ such that } \varphi(\bar{y}\bar{b})\}$$

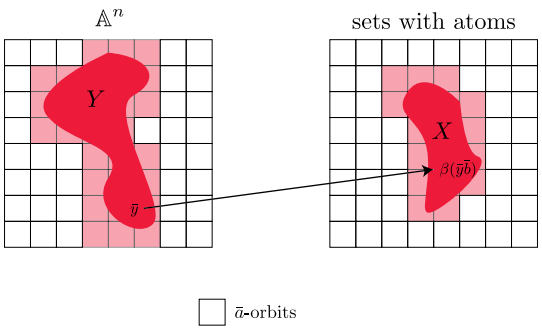
for some atom tuple $\bar{b}$ and some set-builder expression β. Define $Y \subseteq \mathbb{A}^n$ to be the set of tuples $\bar{y}$ which satisfy $\varphi(\bar{y}\bar{b})$. The set X is the image of Y under the function

$$\bar{y} \quad \mapsto \quad \beta(\bar{y}\bar{b}). \tag{5.2}$$

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 (5.2). These are clearly hereditarily definable sets. □

Exercises

Exercise 78. Let $\mathbb{A}$ be a countable structure, which has a decidable first-order theory (without constants) and a computable Ryll-Nardzewski function. Show that $\mathbb{A}$ has a decidable first-order theory with constants.

Exercise 79. 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 first-order formula defining the "same equivariant orbit" on $\mathbb{A}^n$;
(2) the Ryll-Nardzewski function is computable.

Exercise 80. A *Büchi game* has the same syntax as alternating reachability from the previous exercise. 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 hereditarily definable Büchi game. Hint: use memoryless determinacy of Büchi games without atoms, see (Thomas, 1990, Theorem 6.4).

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 3.7 about atom tuples modulo partial equivalence. As an application of the stronger representation theorem, we prove that deterministic register automata have the same expressive power as deterministic orbit-finite automata, for input alphabets of the form

$$\text{finite set} \times \mathbb{A}.$$

6.1 Least supports

We 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.

Theorem 6.1 (Least Support Theorem)**.** [1] *Consider the equality atoms. For every set with atoms x there is a least, with respect to inclusion, finite set of atoms supporting x.*

Fix the equality atoms for the rest of this section. We say that an atom automorphism fixes a set if it is the identity when restricted to that set.

[1] The Least Support Theorem was 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).

Lemma 6.2. *Let S, T be finite sets of atoms. Every atom automorphism π which fixes $S \cap T$ can be presented as a composition*

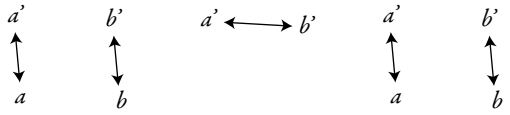
$$\pi = \pi_1 \circ \cdots \circ \pi_n$$

such that each π_i is an atom automorphism that fixes either S or T.

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

Proof of Lemma 6.2 Let us prove the lemma 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.

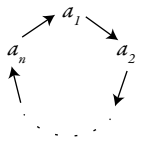
(2) *Finite permutations.* An automorphism (i.e. permutation) of the atoms is called finite if it moves finitely many atoms. Every finite permutation is a finite composition of transpositions, and thus the previous item implies that the conclusion of the lemma is also true when π is a finite permutation.

(3) *Infinite cycles.* Suppose that π is an infinite cycle, as in the following picture:

$$\cdots \longrightarrow a_{-1} \longrightarrow a_0 \longrightarrow a_1 \longrightarrow a_2 \longrightarrow \cdots$$

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 $S \cup T$.

□

Exercises

Exercise 81. Show that the atoms $(\mathbb{Q}, <)$ also have least supports.

Exercise 82. Show an example of oligomorphic atoms without least supports.

Exercise 83. Consider the equality atoms. Show that if a group is orbit-finite, then it is finite.

6.2 A representation theorem for equality atoms

We use the Least Support Theorem to get a classification of equivariant orbit-finite sets in the equality atoms, up to equivariant bijections. Let us write $\mathbb{A}^{(n)}$ for the set of non-repeating n-tuples of atoms. This is an equivariant single-orbit set. To a tuple in $\mathbb{A}^{(n)}$ we can apply a permutation

$$g : \{1, \ldots, n\} \to \{1, \ldots, n\}$$

of the coordinates as follows

$$g(a_1, \ldots, a_n) = (a_{g(1)}, \ldots, a_{g(n)}).$$

Note that we have two groups acting on $\mathbb{A}^{(n)}$: atom automorphisms and permutations of the coordinates. These actions commute with each other: if π is an atom automorphism and g is a permutation of the coordinates then

$$g(\pi(a_1, \ldots, a_n)) = \pi(g(a_1, \ldots, a_n)).$$

Let G be a subgroup of all permutations of the coordinates $\{1, \ldots, n\}$. Define

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

to be $\mathbb{A}^{(n)}$ modulo the equivalence relation which identifies two tuples if they are in the same orbit with respect to the action of G.

Example 34. 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. □

The following theorem shows that quotienting under permutations of coordinates is essentially the only possible construction for equivariant one-orbit sets in the equality atoms.

Theorem 6.3 (Classification of orbit-finite sets)**.** [2] *Consider the equality atoms. Every equivariant one-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 $\{1, \ldots, n\}$.

Proof Let then X be one equivariant orbit. Choose some $x \in X$. By the Least Support Theorem, there is some least support of x, let it be $\{a_1, \ldots, a_n\}$. Consider the relation

$$\{(\pi(a_1, \ldots, a_n), \pi(x)) : \pi \text{ is an atom automorphism}\} \subseteq \mathbb{A}^{(n)} \times X,$$

which is equivariant by definition. Because $a_1, \ldots, a_n$ supports x, the above is actually an equivariant function, call it

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

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

Define G to be the set of permutations g of the coordinates $\{1, \ldots, n\}$ which satisfy

$$f(g(\bar{a})) = f(\bar{a}).$$

The set G is easily seen to be closed under composition and inverse, i.e. it is a subgroup of all permutations. We show below that every tuples $\bar{b}, \bar{c} \in \mathbb{A}^{(n)}$ satisfy

$$f(\bar{b}) = f(\bar{c}) \qquad \text{iff} \qquad \bar{c} = g(\bar{b}) \text{ for some } g \in G \tag{6.3}$$

The above equivalence implies that X is isomorphic to $\mathbb{A}^{(n)}/_G$ and thus finishes the proof.

Let us first show the right-to-left implication in (6.3). Assume that $\bar{c} = g(\bar{b})$ holds for some $g \in G$. Let π be some permutation of the atoms mapping $\bar{a}$ to $\bar{b}$, which exists because there is only one orbit of non-repeating tuples of given dimension.

$$\begin{aligned}
f(\bar{b}) &= \text{(choice of } \pi\text{)} \\
f(\pi(\bar{a})) &= \text{(equivariance of } f\text{)} \\
\pi(f(\bar{a})) &= \text{(definition of } G\text{)} \\
\pi(f(g(\bar{a}))) &= \text{(equivariance of } f\text{)} \\
f(\pi(g(\bar{a}))) &= \text{(automorphisms and } G \text{ commute)} \\
f(g(\pi(\bar{a}))) &= \text{(choice of } \pi\text{)} \\
f(g(\bar{b})) &= \text{(choice of } g\text{)} \\
f(\bar{c})
\end{aligned}$$

We now show the left-to-right implication in (6.3). Assume that $f(\bar{b}) = f(\bar{c})$. Let π be some permutation of the atoms which maps $\bar{b}$ to $\bar{a}$. By equivariance of f,

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

Since f is equivariant, every input supports its output, and therefore the tuple $\pi(\bar{c})$ supports $f(\bar{a})$. By the Least Support Theorem, the atoms that appear in the tuple $\pi(\bar{c})$ must be $\{a_1, \ldots, a_n\}$, i.e. there must be some permutation g of coordinates such that

$$\pi(\bar{c}) = g(\bar{a})$$

Since $\pi(\bar{c})$ has the same image under f as $\bar{a}$, it follows that $g \in G$. Therefore,

$$\bar{c} = \pi^{-1}(g(\bar{a})) = g(\pi^{-1}(\bar{a})) = g(\bar{b})$$

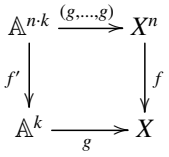
which completes proof of (6.3). □

Exercises

Exercise 84. 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



6.3 Extended example: deterministic automata

In the case study on orbit-finite automata, we showed in Theorem 5.9 that for nondeterministic register automata have the same expressive power as nondeterministic orbit-finite automata, for alphabets where the two notions could be compared. Using the representation result from Theorem 6.3, we prove a similar result for deterministic automata.

Theorem 6.5. *Consider the equality atoms $\mathbb{A} = (\mathbb{N}, =)$. 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.*

The rest of Section 6.3 is devoted to proving the above theorem. In the proof, we use an intermediate automaton model, based on the following definition.

Definition 6.6 (Straight set)**.** A *straight set* is a set which admits an equivariant bijection with a set of the form

$$\mathbb{A}^{(n_1)} + \cdots + \mathbb{A}^{(n_k)} \qquad \text{for some } k, n_1, \ldots, n_k \in \{0, 1, \ldots\}.$$

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 9.

We prove Theorem 6.5 in two steps, as described below:

deterministic orbit-finite equivariant automata $\overset{\text{Lemma 6.8}}{=}$ deterministic orbit-finite equivariant automata with straight states $\overset{\text{Lemma 6.7}}{=}$ deterministic register automata

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. We care about automata which are equivariant, i.e. all of the functions described above 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:

$$\begin{array}{ccc} 1 & \xrightarrow{\iota_{\mathcal{B}}} & P \\ & {\scriptstyle \iota_{\mathcal{A}}} \searrow & \downarrow {\scriptstyle h} \\ & & Q \end{array} \qquad \begin{array}{ccc} P \times \Sigma & \xrightarrow{\delta_{\mathcal{B}}} & P \\ {\scriptstyle (h,id)} \downarrow & & \downarrow {\scriptstyle h} \\ Q \times \Sigma & \xrightarrow[\delta_{\mathcal{A}}]{} & Q \end{array} \qquad \begin{array}{ccc} P & & \\ {\scriptstyle h} \downarrow & \searrow^{F_{\mathcal{B}}} & \\ Q & \xrightarrow[F_{\mathcal{A}}]{} & \{\text{yes,no}\} \end{array} .$$

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 will use homomorphisms to prove that deterministic automata recognise the same languages in Lemmas 6.7 and 6.8 below, which complete the proof of Theorem 6.5.

Lemma 6.7. *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-fintie automaton into a straight set. This is done using the representation theorem from the previous section.

Lemma 6.8. *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

$$\mathbb{A}^{(n_1)}_{/G_1} + \cdots + \mathbb{A}^{(n_k)}_{/G_k}$$

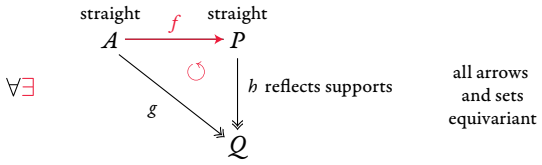
for some dimensions n_i and groups G_i. Define P to be the straight set

$$\mathbb{A}^{(n_1)} + \cdots + \mathbb{A}^{(n_k)}$$

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 6.7, 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 6.9.



Proof It is enough to consider the case when A and P have one orbit each, i.e. these are sets of non-repeating tuples of certain dimensions:

$$A = \mathbb{A}^{(n)} \quad P = \mathbb{A}^{(k)} \qquad \text{for some } n, k \in \{0, 1, \ldots\}.$$

Choose some $\bar{a} \in A$. Because h is surjective, one can choose some $\bar{b} \in P$ such that

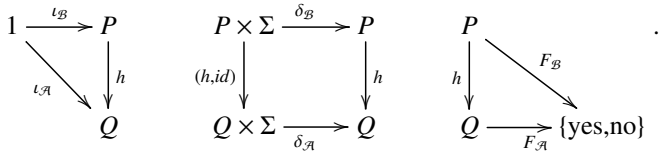
$$g(\bar{a}) = h(\bar{b}).$$

Because h reflects supports, and $\bar{a}$ supports $g(\bar{a})$ by equivariance of g, it follows that $\bar{a}$ supports $\bar{b}$. This means that $\bar{b}$ is a tuple consisting of some of the atoms from $\bar{a}$, possibly in a different order. It follows that $\bar{b}$ is obtained from $\bar{a}$ by applying a function of the form

$$(a_1, \ldots, a_n) \mapsto (a_{i_1}, \ldots, a_{i_k}) \qquad \text{for some distinct } i_1, \ldots, i_k \in \{1, \ldots, n\}.$$

This is the function f from the statement of the claim. □

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 6.9, while acceptance is defined as the composition $F_{\mathcal{A}} \circ h$. □

7

Homogeneous atoms

To define orbit-finiteness, we have assumed that the atoms are oligomorphic. How does one get oligomorphic structures?

In this chapter we show one way of getting oligomorphic structures, called the Fraïssé limit. Actually, the Fraïssé limit will produce structures which are not just oligomorphic, but satisfy a stronger property, called *homogenity*. In homogeneous structures, first-order logic collapses to its quantifier-free fragment. 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.

The Fraïssé limit and homogeneous structures are basic notions in model theory. For further information, see e.g. (Hodges, 1993, Section 7).

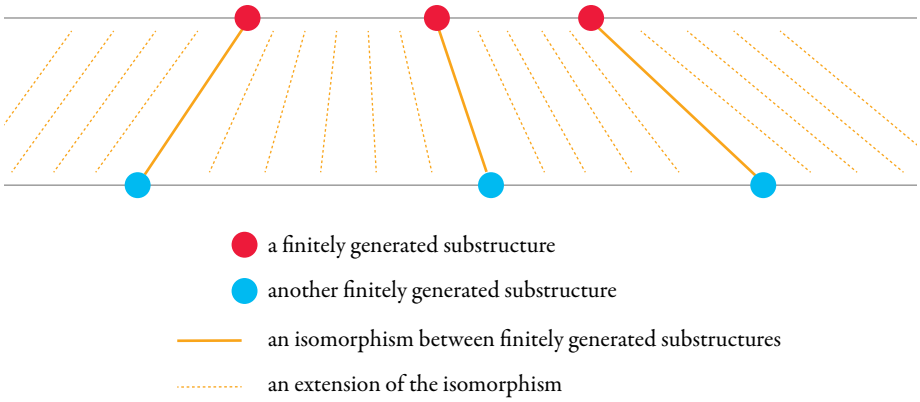
7.1 Homogeneous structures

In this section we study atom structures over finite vocabularies, possibly including functions. An *(induced) substructure* of a structure $\mathbb{A}$ is defined to be a structure obtained from $\mathbb{A}$ by restricting the universe to some subset that is closed under applying functions from the vocabulary. Since we only talk about induced substructures, we simply omit the word induced. The substructure *generated* by a subset of the universe is defined to be the smallest substructure which contains the subset. A *finitely generated* substructure is one generated by a finite subset of the universe. When the vocabulary has no function symbols, then a finitely generated substructure is obtained by restricting the universe to an arbitrary finite subset.

Definition 7.1 (Homogeneous structure)**.** A structure is called *homogeneous*

if every isomorphism between finitely generated substructures extends to a full automorphism.

Example 35. The empty set of atoms, the equality atoms, and $(\mathbb{Q}, <)$ are all homogeneous structures over finite vocabularies. The proof for $(\mathbb{Q}, <)$ is in this picture



In Theorem 7.3, we show that if a structure is homogeneous, and satisfies a further condition that is true whenever the vocabulary uses no functions, then it is oligomorphic. This explains why the structures mentioned above are oligomorphic. □

Example 36. Consider the powerset of the natural numbers

$$(\mathsf{P}(\mathbb{N}), \cup),$$

with the 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 37. [Integers are or are not homogeneous, depending on vocabulary] Unlike for oligomorphism, whether or not a structure is homogeneous depends on the choice of predicates in the vocabulary, and not just the automorphism group. Consider the structures

$$\underbrace{(\mathbb{Z}, <)}_{\text{integers with an order relation}} \qquad \underbrace{(\mathbb{Z}, +1)}_{\text{integers with a successor function}} .$$

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. 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 10. □

The structure from the following example will be discussed in more detail in Chapter 10 about Turing machines.

Example 38. [Bit vectors] We use the name *bit vector* for an infinite sequence of zeroes and ones which has finitely many ones. By ignoring trailing zeroes, a bit vector can be represented as a finite sequence such as 00101001. Define the *bit vector structure* to be the bit vectors equipped with a function for coordinatewise addition modulo two:

$$0101 + 11001 = 10011.$$

The bit vectors are a vector space over the two element field. The dimension of this linear space is countably infinite, 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 $1, 11, 111, 1111, \ldots$.

The bit vector structure is homogeneous. What is a finitely generated substructure? This is a substructure generated by a finite set of linearly independent bit vectors. What is a partial automorphism between finitely generated substructures? This is an arbitrary bijection between two finite sets of linearly independent bit vectors, extended homomorphically to their linear combinations. What is a full automorphism? This is an arbitrary bijection between two bases, extended homomorphically to their linear combinations. It follows that every finite partial automorphism extends to a full automorphism: use the Steinitz exchange lemma to extend the bijection of two finite linearly independent sets to a bijection of bases. □

Quantifier elimination. In an oligomorphic structure, equivariant sets of tuples of atoms are necessarily definable in first-order logic, see Lemma 4.7. For homogeneous structures, quantifier-free formulas are enough.

Lemma 7.2. *Let $\mathbb{A}$ be homogeneous. 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. For example, in the homogeneous structure $(\mathbb{Z}, +1)$, the property $a = b + 10$ can be expressed in a quantifier-free way, likewise for integers other than 10. In such examples, the quantifier-free type cannot be expressed by a single quantifier-free formula.

Relation to oligomorphism. Not all oligomorphic structures are homogeneous, as witnessed by the structure $(\mathbb{Z}, +1)$ from Example 37. The following theorem explains which homogeneous are oligomorphic, at least under the assumption that the vocabulary is finite (but can include functions).

Theorem 7.3. *Let $\mathbb{A}$ be a homogeneous structure over a finite vocabulary. 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 atoms, then it is generated by a term of height at most k. 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. By Lemma 7.2, 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. 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 is definable by a quantifier free formula thanks to Lemma 7.2.

□

Example 39. The structure $(\mathbb{Z}, +1)$ violates condition (*) from Theorem 7.3, because every integer generates an infinite set. In the powerset of the natural numbers from Example 36, condition (*) is satisfied, because n elements generate at most 2^n sets. The same bound is true for the bit vector atoms from Example 38. □

A corollary of Theorem 7.3 is that in structures which satisfy 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 certain atoms as constants.

7.2 The Fraïssé limit

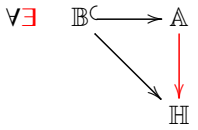
We have seen examples of homogeneous atoms, such as the equality atoms, $(\mathbb{Q}, <)$ or the bit vector atoms. In this section we present a construction, which inputs a class of finitely generated structures, and produces a single (usually infinite) homogeneous structure, called its *Fraïssé limit*. The Fraïssé limit is the only possible way of producing a countable homogeneous structure, because every homogeneous structure is the Fraïssé limit of its finitely generated substructures.

We begin by showing that a countable homogeneous structure is uniquely determined by its finitely generated substructures.

Theorem 7.4. *For a countable structure $\mathbb{H}$, the following are equivalent:*

(1) *$\mathbb{H}$ is homogeneous;*
(2) *If $\mathbb{B} \subseteq \mathbb{A}$ are finitely generated substructures of $\mathbb{H}$ then every embedding of*

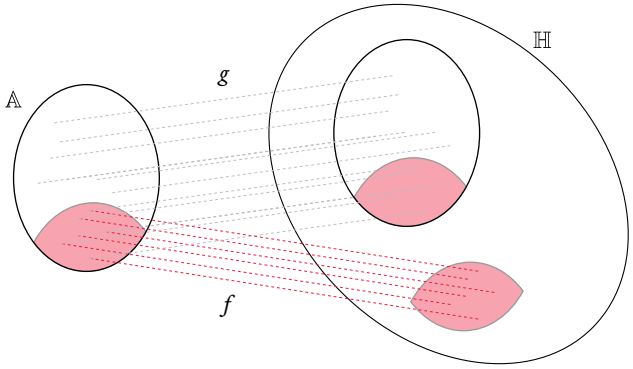
$\mathbb{B}$ *into* $\mathbb{H}$ *extends to an embedding of* $\mathbb{A}$*. In a diagram:*



Furthermore, countable homogeneous structures with the same finitely generated substructures are isomorphic.

Proof

- (1) ⇒ (2). Let f be an embedding of $\mathbb{B}$ into $\mathbb{H}$ as in the assumption of (2), and let g 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 f^{-1} and then g, we get a partial automorphism between two finitely generated substructures (the two red parts) of $\mathbb{H}$. By homogeneity, this partial automorphism extends to a full automorphism. The function g composed with the inverse of that automorphism is the desired embedding.
- (2) ⇒ (1). The following claim, in the case of $\mathbb{H} = \mathbb{H}_1 = \mathbb{H}_2$ shows that $\mathbb{H}$ is homogeneous.

  Claim 7.5. *Let* $\mathbb{H}_1, \mathbb{H}_2$ *be countable structures with the same finitely generated substructures. If both satisfy* (2)*, 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 either $\mathbb{H}_1$. Let $\mathbb{A}$ be the substructure of $\mathbb{H}_1$ generated by a plus all the domain of f. This is a finitely generated structure, and it must also be a substructure of $\mathbb{H}_2$ by the assumption of the

claim. By (2), 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:

(*) Let f be an isomorphism between finitely generated substructures of $\mathbb{H}_1$ and $\mathbb{H}_2$. For 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, every element of both structures appears eventually in the source or target of the partial isomorphisms. The full isomorphism is then the limit of these partial isomorphisms. □

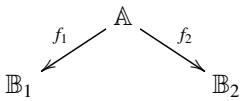
- By Claim 7.5 applied to an empty partial isomorphism between $\mathbb{H}_1, \mathbb{H}_2$, we see that homogeneous are uniquely determined by their countably generated substructures.

□

Amalgamation

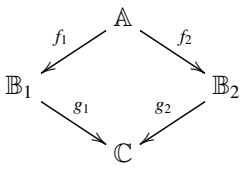
By Theorem 7.4, a countable homogeneous structure is uniquely identified by its finitely generated substructures. Which classes of finitely generated structures are obtained by taking substructures of a homogeneous structure? The special property which distinguishes these classes is amalgamation, which is described below.

An *instance of amalgamation* is two embeddings with a common source:



(7.1)

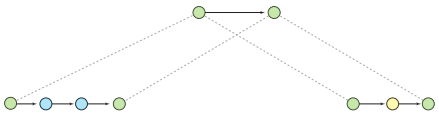
A *solution* of the instance is a structure $\mathbb{C}$ and two embeddings g_1, g_2 such that the following diagram commutes:



(7.2)

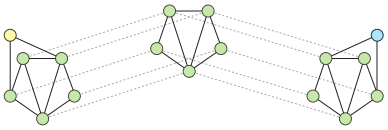
We say that a class of structures is *closed under amalgamation* if every instance of amalgamation which uses structures from the class has a solution which also uses a structure from the class.

Example 40. Consider the class of finite directed graphs (viewed as structures with one binary relation), where the edge relation is a partial successor, i.e. all 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:



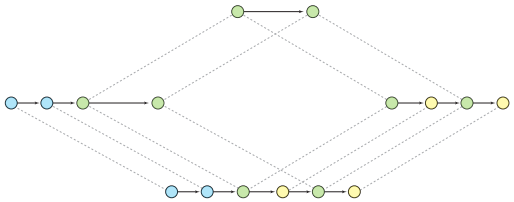
□

Example 41. Consider the class of (undirected) finite planar graphs. Undirected edges are modelled by saying that the binary predicate is symmetric. The class is not closed under amalgamation, here is an instance without a solution:



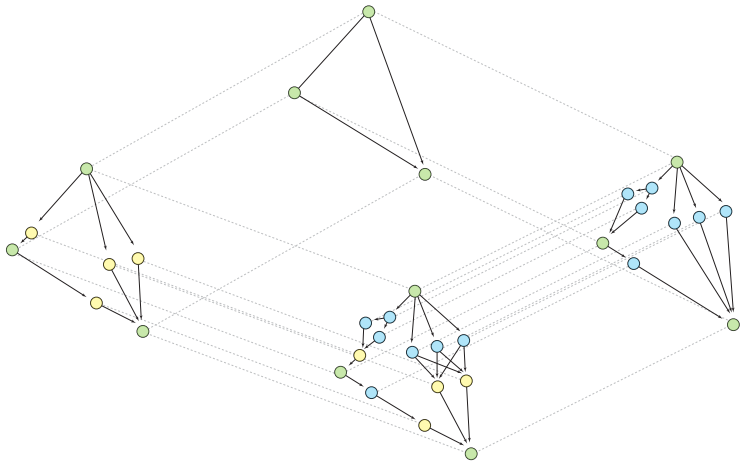
□

Example 42. Consider the class of finite total orders. This class is closed under amalgamation. Here is an example of an instance and solution of amalgamation:



□

Example 43. Consider the class of all finite partial orders, i.e. binary relations that are reflexive and transitive. This class is closed under amalgamation. Here is an example of an instance and solution of amalgamation:



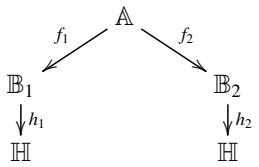
One way of amalgamating two partial orders, which is illustrated in the picture above, is to put the elements of the left (yellow) order after the elements of the right (blue) order, as long as they have the same relationship with the common (green) elements. Other strategies lead to other amalgamations. □

Theorem 7.6. *The following conditions are equivalent for every class $\mathscr{A}$ of finitely generated structures over a common vocabulary, which is closed under isomorphism and substructures:*

(1) *There is a countable homogeneous structure $\mathbb{H}$ such that $\mathscr{A}$ is equal to the set of finitely generated structures that embed into $\mathbb{H}$.*
(2) *$\mathscr{A}$ is closed under amalgamation.*

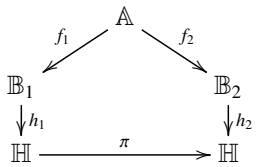
Proof

(1) ⇒ (2) Let $\mathbb{H}$ be a homogeneous structure, and let $\mathscr{A}$ be the finitely generated structures that embed into it. We need to show that $\mathscr{A}$ is closed under amalgamation. Consider an instance of amalgamation which uses structures that embed into $\mathbb{H}$, as in the following diagram:



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 substructures of $\mathbb{H}$; and by homogeneity there is a full automorphism π which extends this partial automorphism. In other words, the following diagram commutes:



The above diagram shows a solution of amalgamation.

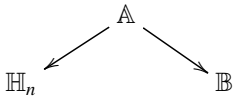
(2) $\Rightarrow$ (1) Let $\mathscr{A}$ be a class of finitely generated structures that is closed under amalgamation. Let $\mathscr{E}$ be all embeddings

$$f : \mathbb{A} \to \mathbb{B} \qquad \text{with } \mathbb{A}, \mathbb{B} \in \mathscr{A}.$$

Call two embeddings in $\mathscr{E}$ equivalent if they are the same, up to isomorphisms of the source and target structures $\mathbb{A}, \mathbb{B}$. Up to this equivalence, there are countably many embeddings in $\mathscr{E}$, therefore one can choose an enumeration $f_1, f_2, \ldots \in \mathscr{E}$ which contains all embeddings up to equivalence. Using the assumption that $\mathscr{A}$ is closed under amalgamation, we can define a sequence

$$\mathbb{H}_1 \subseteq \mathbb{H}_2 \subseteq, \cdots \quad \in \mathscr{A}$$

such that for every $n \in \{1, 2, \ldots\}$ and every instance of amalagamation



that uses embeddings from $\{f_1, \ldots, f_n\}$ (up to equivalence), there is a solution whose target is $\mathbb{H}_{n+1}$. Define $\mathbb{H}$ to be the limit (i.e. union) of the sequence $\mathbb{H}_1, \mathbb{H}_2, \ldots$. By construction, $\mathbb{H}$ has all structures from $\mathscr{A}$ as substructures, and no others. Since $\mathbb{H}$ satisfies condition (2) from Theorem 7.4, it is homogeneous.

□

The countable homogeneous structure which is constructed in the above theorem is called the *Fraïssé limit* of the class $\mathscr{A}$. By Theorem 7.4, this limit is unique up to isomorphism. Applying the Fraïssé limit to the classes from Examples 42 and 43, we get homogeneous structures:

- The Fraïssé limit of all directed total orders is $(\mathbb{Q}, <)$.

- There is a Fraïssé limit of all partial orders. This partial order is not easy draw. It contains as isomorphic copies the rational numbers, infinite antichains, and the infinite binary tree.

Exercises

Exercise 85. Assume a finite relational vocabulary. Suppose that $\mathscr{A}$ is a class of structures that satisfies the assumptions of Theorem 7.6, and let $\mathbb{A}$ be its Fraïssé limit. Show that if membership in $\mathscr{A}$ is decidable, $\mathbb{A}$ is an effective structure.

Exercise 86. 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[1], 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 87. Define a *weak tree* to be a partially ordered set where for every x, the set $\{y : y < x\}$ is totally ordered. Show that the class of finite trees is not closed under amalgamation.

Exercise 88. Define a *tree* to be a weak tree in the sense of Exercise 87, together with a binary function which maps two nodes to their closest common ancestor. Show that the class of trees is closed under amalgamation. We use the name *random tree* for the Fraïssé limit of this class.

Exercise 89. Assume that the atoms is the random tree described in Exercise 88. Find a finitely supported equivalence relation on the atoms which has infinitely many infinite equivalence classes.

Exercise 90. Assume that the atoms is the random tree described in Exercise 88. 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.

[1] For an introduction to MSO and Rabin's Theorem, see (Thomas, 1990, Theorem 6.8).

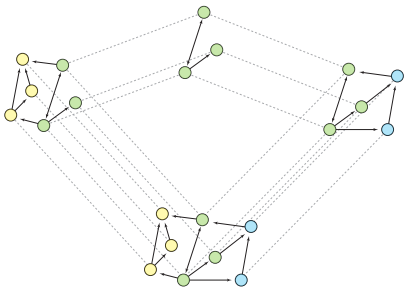
Exercise 91. Show that the random tree from Exercise 88 has decidable MSO theory.

Exercise 92. [2] 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 Extended example: graph atoms

This section describes an extended example of a homogeneous structure, namely the limit of all graphs. For the rest of this section, by graph we mean an undirected graphs, although the same results hold for directed graphs. A graph is modelled as a structure with a single binary predicate.

The class of finite graphs is closed under amalgamation. An instance of amalgamation is basically two graphs that have a common induced subgraph. One way of amalgamating them looks like this:



The amalgamation above is not unique; one could also add edges between the blue and yellow nodes. The same reasoning shows that, for instance, the class of all cliques is closed under amalgamation.

Since it is closed under amalgamation, the class of finite graphs has a Fraïssé limit by Theorem 7.6. Call this limit the *random graph*. The name is justified by the following observation.

[2] This exercise is essentially (Bojańczyk et al., 2013b, Proposition 2).

Theorem 7.7. *Consider a countably infinite graph, where each edge is chosen independently with equal probability one half. 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 has probability one, because failing the event would require infinitely many coin tosses that go wrong. Since there are countably many choices of $F \subseteq G$ and functions h, up to isomorphism, it follows that with probably one the graph $\mathbb{H}$ satisfies condition (2) of Theorem 7.4, and therefore it is isomorphic to the random graph. □

Since the random graph has only one relation in its vocabulary and no functions, it is oligomorphic by Theorem 7.3. Therefore, we can talk about orbit-finiteness when the atoms are the random graph. Also, constructions that involve the random graph are effective:

Lemma 7.8. *The random graph has a decidable first-order theory with constants.*

Proof By Exercise 78, it is enough to show that the random graph has decidable first-order theory (without constants). This proof uses no special properties of the random graph, only that the set of its substructures (i.e. finite graphs) has decidable membership.

We show that first-order logic has effective quantifier elimination. Define an *n-pointed graph* to be a finite graph G with a surjective valuation

$$\{1, \ldots, n\} \to \text{vertices of } G.$$

For every $a_1, \ldots, a_n \in \mathbb{A}^n$, the n-pointed graph of the tuple $(a_1, \ldots, a_n)$ is defined by taking the subgraph of $\mathbb{A}$ induced by $\{a_1, \ldots, a_n\}$ together with the valuation $i \mapsto a_i$. The isomorphism type of the n-pointed graph is essentially the same thing as a quantifier-free type. Since the random graph is homogeneous, the truth value

$$\mathbb{A} \models \varphi(\bar{a}) \qquad \text{for } \bar{a} \in \mathbb{A}^n$$

depends only on the isomorphism type of the n-pointed graph of $\bar{a}$. Define the *representation* of a first-order formula $\varphi(x_1, \ldots, x_n)$ to be the set of (isomorphism types) of n-pointed graphs which are consistent with that formula. This set is necessarily finite, since there are finitely many n-pointed graphs modulo isomorphism. The representation can be computed from the formula φ, by

induction, in a straightforwad manner. To compute the representation of

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

we give all n-pointed graphs that can be extended, by adding one vertex, to an $(n+1)$-pointed graph that belongs to the representation of ψ. Since the class of finite undirected graphs has decidable membership, this representation can be computed. □

Example 44. 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 stores the last seen vertex in its state.

□

Example 7.9 (Path decompositions)**.** For $k \in \{1, 2, \ldots\}$, define a *width k path decomposition* to be a sequence

$$\underbrace{V_1, \ldots, V_n}_{\text{bags}} \subseteq \mathbb{A} \qquad \text{where } |V_1|, \ldots, |V_n| \leq k + 1$$

which satisfies the following conditions: (a) for every atom, the indexes of bags where it appears is a connected interval in $\{1, \ldots, n\}$; and (b) if atoms appear in the decomposition (possibly in different bags) and are connected by an edge, then they appear together in some bag.

The set of path decompositions can be viewed as a language over the orbit-finite alphabet Σ_k which consists of subsets of $\mathbb{A}$ that have size at most $k + 1$. This language is not recognised by an orbit-finite automaton.

Exercises

Exercise 93. 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 94. 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 95. 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 96. Assume that the atoms are the random graph. Show that there is no finitely supported total order on the random graph.

Exercise 97. 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.

Bibliographic notes for Section 3. The idea to use a homogeneous structure as the atoms, and to the consider orbit-finite sets over those atoms, is from Bojańczyk et al. (2014, 2013b).

PART THREE

COMPUTATION WITH ATOMS

In part II, we defined orbit-finite sets, and showed how orbit-finite sets can behave similarly to finite ones. We also gave some algorithms that computed properties of 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

In this section we discuss how algorithms can manipulate orbit-finite sets. For example, in Section 5, we have introduced various models of automata where the states, transitions, etc. are orbit-finite. But what is the point of studying automata if one cannot run algorithms on them, such as testing for emptiness or minimisation? The goal of this chapter is to define what it means for an operation on definable sets to be computable. In this chapter, we only care about computability in finite time – which is already non-trivial for infinite sets. In chapter 9, we discuss an orbit-finite version of polynomial time.

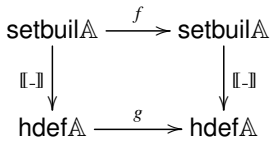
8.1 Definable while programs

If X is a set whose elements can be represented in a finite way, then a partial function $f : X \to X$ is called *computable* if there is a Turing machine which inputs $x \in X$ and: (a) halts and outputs $f(x)$ if its is defined; (b) does not terminate otherwise. Our notion of computability for definable sets is straightforward: an operation on definable sets is computable if it can be computed assuming that definable sets are represented by set builder expressions.

Definition 8.1 (Computable function on definable sets)**.** Assume that the atoms $\mathbb{A}$ are have decidable frist-order theory with atom constants. We write $\mathsf{setbuil}\mathbb{A}$ for expression of the form $\alpha(\bar{a})$ where α is a set-builder expression and $\bar{a}$ is an atom tuple which evaluates the free variables. A partial function

$$\mathsf{set}\mathbb{A} \xrightarrow{f} \mathsf{set}\mathbb{A}$$

is called *computable* if there if the following diagram commutes



for some computable partial function f on set builder expressions.

In the spirit of the above definition, we can also talk about computable yes/no properties of definable sets (such as emptiness), or about computable operations from pairs of definable sets to definable sets (such as intersection or set difference). Formally speaking, these are not new notions: a yes/not output can be represented by viewing the Booleans as definable sets via

$$\text{no} = \emptyset \qquad \text{yes} = \{\emptyset\},$$

while a pair of definable sets can be viewed as a single definable set via Kuratowski pairing. In the next section, we show that the most basic operations like testing for emptiness, Boolean operations, taking pairs, projecting pairs, etc. are all computable.

Definable while programs

In Definition 8.1, we proposed a notion of computability for functions from definable sets to definable sets. A problem with that definition is that it deals with the representation using set builder expressions at each step; leading to cumbersome constructions. The goal of this section is to present a programming language, which deals directly with sets with atoms. The representation issues are tackled once, when designing the programming language. An added bonus of this approach is that sometimes, when the issue at hand is simple enough, the resulting code in sets with atoms is exactly the same as in normal sets. This is the case, for instance, with graph reachability and the automata problems that were discussed in Section 5, such as minimisation or converting a pushdown automaton to a grammar.

Our programming language is called *definable while programs*. We use two assumptions on the atoms:

(1) they are an effective structure in the sense of Definition **??**;
(2) they are effectively oligomorphic in the sense of Exercise 78.

For some results we only need the first assumption, in those cases we can use e.g. Presburger arithmetic for the atoms. Before defining the programming

language, consider the following example, which uses the atoms $(\mathbb{Q}, <)$. These atoms satisfy both assumptions, although formally speaking they are effective only up to isomorphism. Consider the definable set of all bounded open intervals:

$$I = \{\{z : \text{ for } z \text{ such that } x < z < y\} : \text{for } x, y \text{ such that } x < y\}.$$

The following program (comments are in blue) computes all bounded open intervals which contain 0:

```
load some definable sets or atoms into  variables
X := I
Y := ∅
z := 0

run some code in parallel for every element of X
for x in X do
  if z ∈ x then
    Y := Y ∪ {x}
```

After executing the program above, variable `Y` stores all bounded open intervals which contain 0. The above example illustrates the two key properties of the programming language: variables store definable sets, and one can run a for loop in parallel for all elements in a definable set.

Syntax. We assume that there is a countably infinite set of variable names. The language is untyped: every variable stores a definable set, which might be an atom. Although cumbersome, one can encode other data structures using definable sets, e.g. the natural numbers can be encoded by

$$0 \stackrel{\text{def}}{=} \emptyset \qquad 1 \stackrel{\text{def}}{=} \{\emptyset\} \qquad \cdots \qquad n \stackrel{\text{def}}{=} \{0, 1, \ldots, n-1\}.$$

Of course a reasonable implementation would have more features, such as booleans or integer arithmetic. We present the programming language using a minimal syntax, so as to concentrate on the aspects of the language that deal with atoms. Below we describe the possible instructions in the language.

- **Assignment of definable sets.** For every definable set x, which might be an atom, and every variable `x`, one can write an assignment `x :=` x. This part of the syntax depends on the atom structure at hand, since the notion of definable set depends on the atoms. In order for the syntax to be effective, one needs an assumption that definable sets can be represented, e.g. this is

true for effectively oligomorphic structures, or for structures over a finite vocabulary where the universe is $\{0, 1, \ldots\}$.

- **Adding to sets.** If `x`, `y` are variables one can write an instruction `x` := `x` $\cup$ {`y`}, which adds the content of `y` to the set stored in `x`. If `x` does not store a set but an atom, then the instruction aborts with failure.
- **Sequential composition** If `I` and `J` are already defined programs, then also `I;J` is a program which first executes `I` and then `J`.
- **Control flow.** Suppose that `x` and `y` are variables, and `I` is an already defined program, and δ is one of $\in$ or $=$. Then

  `if x` δ `y then I` `while x` δ `y do I` `for x in y do I`

  are all programs. The nonstandard construct is the `for` loop. The general idea is that the instruction `I` is executed, in parallel, with one thread for every element `x` of the set `y`. The semantics are described in more detail below, in particular we explain how the results of the parallel threads are combined in a `for` loop.

Semantics. We now sketch an operational semantics for the language. Define a *program state* to be a function from program variables to definable sets (which might be an atom) which uses finitely many variables, i.e. it assigns the empty set to all but finitely many variables. If γ is a program state and `I` is a while program, then we write γ`I` for the program state obtained by executing `I` when starting in γ. The mapping $\gamma \mapsto \gamma$`I` is a partial function, because the result is undefined when the program does not terminate. In the semantics, we also talk about the running time of a program, which intuitively stands for the maximal number of sequentially executed instructions.

We only explain the semantics for programs of the form

`for x in X do I`,

the other constructions being handled in the standard way. Suppose that we want to execute the for loop above on a program state γ. If the variable `X` stores an atom, then the for loop does nothing, i.e. it does not modify the program state γ. Assume otherwise. For every element x of the set stored in `X`, define γ_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 γ_x, or it might terminate in a finite number of steps n_x. If `I` does not terminate on some γ_x, or the numbers n_x are unbounded, then the `for` loop does not terminate. Otherwise it terminates, and its running time is one plus the maximal number n_x. Apart from terminating, what does the `for` loop do? The idea is to run the body of the loop on every program state γ_x and then aggregate the resulting program

states into a single one. We use the following aggregation function. For a set Γ of program states, define its *aggregation* to be the program state which stores on variable $\mathtt{x}$:

- a definable set x (possibly an atom) if all program states in Γ have x in $\mathtt{x}$;
- 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 $\mathtt{x}$. If that happens, the atom will be lost in the aggregation, because an atom has no elements and it therefore gets ignored when taking a union. Using this notion of aggregation, we define the result of evaluating the for loop to be the aggregation of the set

$$\{\gamma_x \mathtt{I} : x \text{ is an element of the set stored in } \mathtt{X}\}$$

This completes the definition of the semantics of the for loop.

Example 45. The following program implements the mapping $\{x\} \mapsto x$. More precisely, if the variable $\mathtt{x}$ 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
```

In the above, testing if $\mathtt{x} = \{\mathtt{y}\}$ is syntactic sugar for first setting a fresh variable $\mathtt{z}$ to $\{\mathtt{y}\}$ and then testing if $\mathtt{x}$ and $\mathtt{z}$ have the same value. □

Example programs We end this section with some example programs. Like in Python, we use indentation to distinguish blocks in programs. We first extend the syntax with some features. We allow substitutions like

```
x := {y,{y,z}}
```

which are simulated by longer pieces of code like

```
x := ∅
x := x ∪ {y}
u := x ∪ {z}
x := x ∪ {u}
```

We extend the syntax with functions (with the usual semantics); the syntax of functions is illustrated on the Kuratowski pairing function

```
function pair(x,y)
  return {{x},{x,y}}
```

We write `(a,b)` instead of `pair(a,b)`. Here is the function which projects a Kuratowski pair of sets into its first coordinate, and returns $\emptyset$ if its argument is not a Kuratowski pair of sets.

```
function first(p)
  ret := ∅
  for a in p do
    for x in a do
      for y in p do
        if p = {x,{x,y}} then ret:=x;
  return ret
```

The second coordinate of a pair is extracted the same way. Similarly, we could write functions for projections of pairs storing atoms, or pairs storing one atom and one set. Using the projections, we can extend the language with a pattern-matching construction

```
for (x,y) in X do I
```

which ranges over all elements of `X` that are pairs of elements of appropriate types. We use a similar convention for tuples of length greater than two.

Example 46. [Programs that use order on atoms] Consider the atoms $(\mathbb{Q}, <)$. The relation $\leq$ on atoms is a definable set of pairs, and is therefore a constant in the language. Therefore, we can write `x ≤ y` in our programs to compare atoms for order, which is technically speaking syntactic sugar for testing membership of `(x,y)` in $\leq$. For instance, the following program (which uses boolean operations in conditionals, which can be easily simulated in the language) generates the set of growing triples of atoms.

```
X := (ℚ, <)
T := ∅
for x in X do
 for y in X do
  for z in X do
    if (x ≤ y) and (y ≤ z) then T := {(x,y,z)}
```

Since the set of growing triples of atoms is a definable set, we could also just directly load it to `T` with one instruction. □

Example 47. Assume that the atoms are Presburger arithmetic, i.e. $(\mathbb{N}, +)$. One

could easily write a multiplication function, by using a while loop to implement multiplication in terms of iterated addition. One could also write a primality test. The following program looks like it computes the set of all primes:

```
P := ∅
for x in ℕ
  if prime(x) then P := P ∪ {x}
```

Actually, the program does not terminate, because the body of the for loop has unbounded running time, and the semantics says that such loops do not terminate. □

Example 48. [Reachability] We write below a program which inputs a binary relation R and a set of source elements S, and returns all elements reachable (in zero or more steps) from elements in S via the relation R. The program is written using `until`, which is implemented by `while` in the obvious way.

```
function reach (R,S)
  New := S
  repeat
   Old := New
   for (x,y) in R do
     if x ∈ Old then New := Old ∪ {y}
  until Old = New
```

The program above is the standard one for reachability, without any modifications for the setting with atoms.

The program can be run for any atoms, including non-oligomorphic ones. Sometimes, it might even terminate (and thus give the correct result). For example, if atoms are Presburger arithmetic and the relation R is the definable set of pairs of natural numbers which disagree modulo 3, and S is $\{0\}$, then the program will terminate in two iterations of the until loop and return the set of all natural numbers. However, if R would represent the successor relation (or transition function of a nonterminating Minsky machine), then the program might not terminate.

If the atoms are an oligomorphic structure, then the program will always terminate in a finite number of steps. The argument was given already in Exercise 46, but we repeat it here. Suppose that $\bar{a}$ is a tuple of atoms that supports both the relation R and the source set S. Let X be the set that contains S and every element that appears on either the first or second coordinate of a pair from R. The set X is easily seen to be supported by $\bar{a}$ and to have finitely many

$\bar{a}$-orbits. Let $X_1, X_2, \ldots, X_k$ denote the $\bar{a}$-orbits of X. Therefore,

$$X = X_1 \cup X_2 \cup \cdots \cup X_k. \tag{8.1}$$

It is easy to see that after every iteration of the `repeat` loop, the values of both variables `New` and `Old` are subsets of X that are supported by $\bar{a}$. Therefore the values of these variables are obtained by selecting some of the orbits listed in (8.1). In each iteration of the `repeat` we add some orbits, and therefore the loop can be iterated at most k times. □

Example 49. [Automaton emptiness] Using reachability from the previous example, it is straightforward to write an emptiness check for nondeterministic definable automata (recall them from Exercise **??**):

```
function emptyautomaton(A,Q,I,F,delta)
  R := ∅
  for (p,a,q) in delta do
    R := R ∪ {(p,q)}
  return ∅ = (reach(R,I) ∩ F)
```

The program will always terminate if the atoms are oligomorphic. □

Example 50. [Automaton minimisation] The program for automaton emptiness answered a yes/no question. We now present a program that minimizes deterministic automata. The program is a function

```
function minimize(A,Q,q0,F,delta)
```

which inputs a definable deterministic automaton and returns the minimal automaton. (We assume that the atoms are oligomorphic.) We assume that all states are reachable, the non-reachable states can be discarded using the emptiness procedure described above. The code is a standard implementation of Moore's minimisation algorithm. The only point of writing it down here is that the reader can follow the code and see that it works with atoms.

In a first step, we compute in the variable `equiv` the equivalence relation which identifies states that recognise the same languages. To do this, we compute the complement of the equivalence relation. We first the non-equivalence relation to states that are distinguished by the empty word:

```
nonequiv := (F × (Q-F)) ∪ ((Q-F) × F)
```

(The code above uses $\times$, 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 in Q do
  for p' in Q do
    for a in A do
      for q in Q do
        for q' in Q
          if (p,a,q) ∈ delta and (p',a,q') ∈ delta and (q,q') ∈ R
             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) - reach(R,base)
```

For the states of the minimal automaton, we use the equivalence classes of the relation `equiv`, which are produced by the following code.

```
function classes (equiv)
  for (a,b) in equiv do
    for (c,d) in equiv do
       if a=c then class := class ∪ {c}
    ret := ret ∪ {class}
  return ret
```

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. □

Example 51. Call a monoid aperiodic if for every element m, the sequence $m^1, m^2, \ldots$ is ultimately constant. Assume that the atoms are oligomorphic. The following program inputs a monoid (its carrier and the graph of the monoid operation) and returns true if and only if the monoid is aperiodic. The program simply executes the definition of aperiodicity.

```
function aperiodic (Carrier,Monop)
  counterexamples := ∅
    for m in Carrier  do
      X := ∅
      power := m
      while power ∉ X
         X :=X ∪ {power}
         power := Monop(power,m)
      if power  ≠ Monop(power,m) then
```

```
        counterexamples := counterexamples ∪ {m}
    if counterexamples = ∅ then return true else return false
```

In the program above, `Monop(power,m)` is actually syntactic sugar for a subroutine which examines the graph of the multiplication operation `Monop`, and finds the unique element `x` which satisfies `(power,m,x) ∈ Monop`.

In the program, the set `X` is used to collect consecutive powers $\mathtt{m}, \mathtt{m}^2, \mathtt{m}^3, \ldots$. To prove termination, one needs to show that this set is always finite, even if the monoid in question is not aperiodic. Furthermore, there is a fixed upper bound on the size of such sets. Every power of `m` is supported by whatever supports `m` and the multiplication operation in the monoid. Since the carrier of the monoid is definable, it is also orbit-finite, by Theorem 4.6. In an orbit-finite set, there are finitely many elements with a given support, as shown in Exercise 48. It follows that for every `m`, 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 definable while programs

In this section, we show how definable while programs can be simulated by normal programs (e.g. Turing machines). The idea is to describe the parallel execution in a symbolic way.

Definition 8.2 (Function computed by a definable while program)**.** A partial function

$$f : \mathsf{hdef}\mathbb{A} \to \mathsf{hdef}\mathbb{A}$$

is said to be *computed by a definable while program* if there is a while program `I` with a designated interface variable, such that for every definable set x, if the program `I` is executed in the program state where the interface variable stores x and all other variables are empty, then it `I` terminates if and only if $f(x)$ is defined, and if it terminates then the interface variable stores $f(x)$.

Theorem 8.3. *Assume that the atoms are an effective structure. If a partial function*

$$f : \mathsf{hdef}\mathbb{A} \to \mathsf{hdef}\mathbb{A}$$

is computed by a definable while program (in the sense of Definition 8.2) then it is computable (in the sense of Defintion 8.1).

To prove the above theorem, we show how the semantics of definable while programs can be simulated by manipulating definable sets. A program state γ can be viewed as a set

$$\{(\mathtt{x}, \gamma(\mathtt{x})) : \mathtt{x} \text{ is a program variable used by } \gamma\ \}.$$

Assuming that the program variables are special cases of definable sets, e.g. they are natural numbers, a program state is a special case of a definable set. In particular, it makes sense to speak of definable sets of program states. The Interpreter Theorem follows as a special case of the following lemma, by using definable sets of program states that are singletons.

Lemma 8.4 (Interpreter Lemma)**.** *Assume that the atoms are an effective structure. There is a computable function (call it an interpreter) which inputs a definable while program* $\mathtt{I}$ *and a definable set* Γ *of program states, and does the following:*

- *if for every* $n \in \{0, 1, \ldots\}$ *there is some program state* $\gamma \in \Gamma$ *such that* $\mathtt{I}$ *does not terminate on* γ *in at most n steps, then the interpreter does not terminate.*
- *Otherwise, the interpreter terminates and outputs* $\{(\gamma, \gamma\mathtt{I}) : \gamma \in \Gamma\}$.

Proof Note that it follows implicitly from the statement of the lemma that the output is also a definable set, since computable functions on definable sets can only output definable sets.

In the proof of the lemma, it will be convenient to use the following data structure. For definable sets $X_1, \ldots, X_n$, define their *union-member structure* to be the relational structure whose universe is the union-member closure (as defined in Exercise **??**) of $X_1 \cup \cdots \cup X_n$ and which is equipped with unary predicates for the sets $X_1, \ldots, X_n$ and a binary predicate $\in$ for set membership. Using Exercises **??** and **??**, it follows that if $X_1, \ldots, X_n$ are definable sets, then their union-member structure is also definable set and can be computed.

The interpreter works by structural induction on the text of the program $\mathtt{I}$. The proof essentially says that the operational semantics can be made effective. We only do the proof for `if`, `while` and `for`.

- **If.** Suppose that we want to compute the graph of the function

$$\gamma \in \Gamma \qquad \mapsto \qquad \gamma(\mathtt{if\ x}\ \delta\ \mathtt{y\ then\ I}) \tag{8.2}$$

where δ is one of $=, \in$. Consider the two sets

$$\underbrace{\{(\gamma, \gamma(\mathtt{x})) : \gamma \in \Gamma\}}_{X} \qquad \underbrace{\{(\gamma, \gamma(\mathtt{y})) : \gamma \in \Gamma\}}_{Y}$$

which are definable by the assumption that Γ is definable. Let $\mathfrak{A}$ be the union-member structure of these two sets. From Exercise **??** it follows that any relation that can be defined in first-order logic inside $\mathfrak{A}$ is also definable, which means that we can quantify over elements of $X \cup Y$, elements of elements of $X \cup Y$, etc. and also compare them for equality or membership. Note that the Kuratowski definition of pairs can be formalised in first-order logic inside the union-member structure. As an application of this principle, let $\Gamma_0 \subseteq \Gamma$ be the program states which satisfy the condition in the `if` statement:

$$\Gamma_0 = \{\gamma : \exists x \exists y \underbrace{(\gamma, x) \in X}_{\gamma(\mathtt{x})=x} \wedge \underbrace{(\gamma, y) \in Y}_{\gamma(\mathtt{y})=y} \wedge x \; \delta \; y\}.$$

By Exercise **??**, the above set is definable and can be computed based on X and Y. We will use this type of reasoning several times in the proof: whenever something can be defined using first-order logic based on previously known definable things, then it is also definable and can be computed. Apply the induction assumption to compute the graph of

$$\{(\gamma, \gamma\mathtt{I}) : \gamma \in \Gamma_0\}$$

Using first-order logic in the union-member structure of Γ, Γ_0 and the above relation, we define the graph of the function from (8.2), using the formula

$$\{(\gamma, \mu) : (\gamma \in \Gamma_0 \wedge \gamma\mathtt{I} = \mu) \vee (\gamma \in \Gamma - \Gamma_0 \wedge \gamma = \mu)\},$$

and therefore the graph itself is definable.

- **While loop**. Consider a definable program of the form

  ```
  while x = y do J
  ```

  Let Γ be a definable 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 case of `if`, for each n we can compute Γ_n and also the semantics of the while loop with domain restricted to Γ_n. We try all n until $\Gamma_n = \Gamma$. If no such n exists, then the interpreter does not terminate.
- **For loop**. Assume that the input program `I` is

  ```
  for x in X do I.
  ```

  It is not difficult to show that the following ternary relation is definable

  $$\{(\gamma, x, \gamma_x) : \gamma \in \Gamma, x \in \gamma(\mathtt{X})\}$$

Using the induction assumption, the following binary relation is definable

$$\{(\gamma, \gamma_x \mathtt{I}) : \gamma \in \Gamma, x \in \gamma(\mathtt{X})\}$$

By Exercise **??**, the graph of the function

$$\gamma \in \Gamma \mapsto \{\gamma_x \mathtt{I} : x \in \gamma \mathtt{X}\}$$

is a definable set. The semantics of the entire for loop is the composition of the above function with the aggregation function used in the semantics of a `for` loop. Since functions with definable graphs are closed under composition, it suffices to show the following lemma.

Lemma 8.5. *If Δ is a definable family of definable sets of memory states, then the following partial function has a definable graph, which can be computed from Δ:*

$$\Gamma \in \Delta \quad \mapsto \quad \textit{the aggregation of } \Gamma.$$

Proof We show that for every program variable `x` the function

$$\Gamma \in \Delta \quad \mapsto \quad \text{value of } \mathtt{x} \text{ in the aggregation of } \Gamma. \tag{8.3}$$

has a definable graph. Consider the union-member structure corresponding to Δ. Define Δ_0 to be those $\Gamma \in \Delta$ where all program states agree on variable `x`, this set is definable as it can be defined in first-order logic in the union-member structure of Δ. For the same reason, the function from (8.3) can be defined using first-order logic in the same structure. □

□

In Theorem 8.4, we showed that definable while programs describe computable functions on definable sets. Do they describe all computable functions? Did we miss some feature in the programming language? In this section we show that definable while programs are computationally complete in a sense that is described below.

Theorem 8.6 (Computational completeness of definable while programs)**.** *Assume that the atoms $\mathbb{A}$ are an effective structure which is furthermore effectively oligomorphic in the sense of Exercise 78. Then for every partial function*

$$f : \mathsf{hdef}\mathbb{A} \to \mathsf{hdef}\mathbb{A}$$

the following conditions are equivalent:

(1) *f is computed by a definable while program;*
(2) *f is finitely supported and computable.*

The implication from 1 to 2 is a corollary of Theorem 8.4 and the observation that the semantics of a definable while program are invariant under atom automorphisms which fix those atoms that appear in the code of the program. The implication from 1 to 2 does not use effective oligomorphism or even oligomorphism alone.

The rest of Section 8.2 is devoted to proving the implication from 2 to 1. Let f be as in item 2. To show item 1, we use the following lemma, which says that a while program can convert a definable set into a set builder expression defining it, at least up to automorphisms.

Lemma 8.7. *Assume that the atoms are effectively oligomorphic. Let $\bar{a}$ be a tuple of atoms. There is a definable while program which inputs a definable set x and outputs a set builder expression α and a tuple of atoms $\bar{b}$ evaluating its free variables such that*

$$\pi(x) = [\![\alpha]\!](\bar{b}) \qquad \textit{for some } \bar{a}\textit{-automorphism } \pi.$$

Proof Suppose that the input set is x. The program enumerates through all possible set builder expressions α. For each set builder expression α, say with free variables $\bar{y}$, it does a for loop across all $\bar{y}$-tuples of atoms to compute the set

$$B_\alpha = \{\bar{b} : \bar{b} \text{ is a } \bar{y}\text{-tuple of atoms such that } x = [\![\alpha]\!](\bar{b})\}.$$

To compute B_α, we need to show that a while program can compute $[\![\alpha]\!](\bar{b})$ given α and $\bar{b}$; this is not difficult to do by structural induction on the set builder expression α. If the set B_α is empty, then the program proceeds to the next set builder expression α. By definition of definable sets, eventually a set builder expression α will be found so that B_α is nonempty. Suppose then that α is such that B_α is nonempty, and let n be the number of free variables in α, which means that $B_\alpha \subseteq \mathbb{A}^n$.

Let k be the dimension of the tuple $\bar{a}$. Using Exercise 79 and the assumption that the atoms are effectively oligomorphic, one can compute first-order formulas $\varphi_1, \ldots, \varphi_m$ in $k + n$ free variables which define all orbits of $\mathbb{A}^{k+n}$. Every $\bar{b} \in B_\alpha$ is in some $\bar{a}$-orbit, which means that there must be some i such that $\varphi_i(\bar{a}\bar{b})$ holds. Therefore, in particular there must be some i such that some tuple $\bar{b} \in B_\alpha$ satisfies $\varphi_i(\bar{a}\bar{b})$, and this i can be computed. (First-order formulas can be evaluated in the program, by using for loops to simulate quantifiers; this observation was already used in evaluating set-builder expressions.) A tuple of atoms $\bar{b}$ satisfies $\varphi_i(\bar{a}\bar{b})$ if and only if

$$\pi(x) = [\![\alpha]\!](\bar{b}) \qquad \text{for some } \bar{a}\text{-automorphism } \pi.$$

The program uses decidability of the first-order theory of $\mathbb{A}$ to enumerate all possible tuples $\bar{b}$ until it finds one which maps which makes $\varphi_i(\bar{a}\bar{b})$ true, and this is the output. □

We now complete the proof of the implication from 2 to 1 in Theorem 8.6. Suppose that f is a function from definable sets to definable sets which is supported by a tuple of atoms $\bar{a}$, and assume that f is computable. We present below a definable while program which computes f. Assume that on input we have a definable set x. By Lemma 8.7, a definable while program can compute α and $\bar{b}$ such that

$$\pi(x) = [\![\alpha]\!](\bar{b}) \qquad \text{for some } \bar{a}\text{-automorphism } \pi. \tag{8.4}$$

By the assumption that f is computable, a while program can compute a set builder expression β and an atom tuple $\bar{c}$ evaluating its free variables such that

$$f([\![\alpha]\!](\bar{a})) = [\![\beta]\!](\bar{c}) \tag{8.5}$$

Using the same ideas as in Lemma 8.7, a definable while program can compute the set $\bar{a}$-orbit of the tuple $\bar{b}\bar{c}$, i.e. the set

$$\{\pi(\bar{b}\bar{c}) : \pi \text{ is an } \bar{a}\text{-automorphism}\}$$

From the above, we can compute the set

$$\begin{aligned} f_0 \stackrel{\text{def}}{=} \{([\![\alpha]\!](\pi(\bar{b})), [\![\beta]\!](\pi(\bar{c}))) : \pi \text{ is an } \bar{a}\text{-automorphism}\} \stackrel{(8.5)}{=} \\ \{\pi([\![\alpha]\!](\bar{b}), f([\![\alpha]\!](\bar{b}))) : \pi \text{ is an } \bar{a}\text{-automorphism}\}. \end{aligned}$$

Since f is a function supported by $\bar{a}$, the set f_0 is a subset of the graph of f. Since the partial function f_0 is supported by $\bar{a}$, and its domain contains $[\![\alpha]\!](\bar{b})$, it follows from (8.4) that the domain of f_0 also contains x. Therefore, we can simply evaluate f_0 in x to get $f(x)$. This completes the implication from 2 to 1 in Theorem 8.6.

Bibliographic notes for Section 2. The programming language of while programs with atoms is based on the language from Bojańczyk and Toruńczyk (2012). This language was preceded by Bojanczyk et al. (2012), which proposed a programming language in a functional paradigm (see also Moerman et al. (2017) for an implementation of the functional programming language together with an application to learning register automata). Both languages (definable while programs and the functional language) have the same expressive power. Definition 8.1 and Theorem 8.6 on computational completeness of while programs are inspired by (Bojańczyk et al., 2013a, Theorems IV.1 and IV.2), although Theorem 8.6 is more general because it applies to computation on sets and not just objects that can be encoded as strings (see Section 10 for limitations of the second approach).

The programming languages from Bojanczyk et al. (2012); Bojańczyk and Toruńczyk (2012) and their semantics were written under the assumption that the atoms are oligomorphic. The idea to extend the semantics to atoms that are not necessarily oligomorphic, but have decidable first-order theory, is inspired by the programming language LOIS (*Looping over Infinite Sets*) Kopczynski and Toruńczyk (2016, 2017). The Symbol Pushing Lemma is inspired by LOIS.

9

Polynomial time

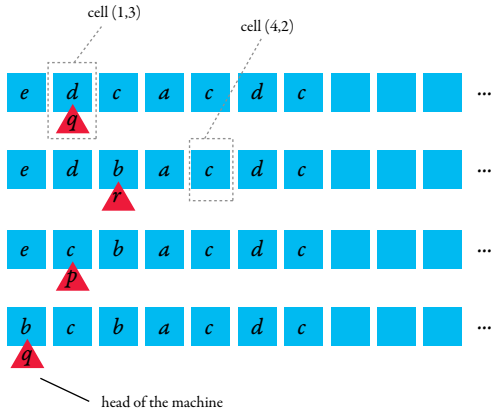
10
Turing machines

In this section, we talk about definable Turing machines. A definable 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 one tape, and where the transition relation is a subset of:

$$\underbrace{(\overbrace{\Gamma}^{\text{work alphabet}} \cup \{\text{blank}\}) \times \overbrace{Q}^{\text{states}}}_{\text{what the machine sees}} \times \underbrace{(\{\text{accept, reject}\} \cup (\overbrace{\Gamma \times Q \times \{\text{left, stay, right}\}}^{\text{write a symbol and move the head}}))}_{\text{what the machine does}}.$$

We assume for the rest of this section that the atoms are oligomorphic and have decidable first-order model checking. By the assumption on oligomorphism, we definable is the same thing as hereditarily orbit-finite.

The tape in a Turing machine is a sequence of cells indexed by natural numbers (we use tapes that are infinite to the right only), each cell storing a symbol of the work alphabet or a blank symbol (and possibly also the state of the machine, if head happens to be in that cel). Here is a picture of a computation:



A finite computation of a nondeterministic Turing machine is defined to be a function (for some n, namely the computation time)

$$\rho : \underbrace{\mathbb{N}}_{\text{space}} \times \underbrace{\{1, \ldots, n\}}_{\text{time}} \to \underbrace{\Gamma \cup \{\text{blank}\}}_{\text{cells without the head}} \cup \underbrace{(\Gamma \cup \{\text{blank}\}) \times Q}_{\text{cells with the head}} \tag{10.1}$$

which is consistent with the transition function of the Turing machine in the usual way. We assume that the first row (i.e. corresponding to the initial step of the computation) consists of some input word padded by an infinite sequence of blanks, with the head having the initial state and positioned over the first input letter.

Many of the standard results for standard Turing machines remain true for definable Turing machines, such as equivalence of single-tape and multi-tape machines, using the standard proofs. What does not work, however, is determinisation, and this is the focus of this section.

Example 52. [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 is 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 go back to the beginning of the tape. If the first non-bnak letter on the tape is $\underline{5}$, then reject immediately. The set of states is

$$\underbrace{\{q_0\}}_{\text{initial state}} \cup \underbrace{\mathbb{A} - \{\underline{5}\}}_{\text{scan to the right if this atom reappears}} \cup \underbrace{\{q_1\}}_{\text{return to the beginning}}$$

□

Example 53. [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

$$i_1 \# i_2 \# \cdots \# i_n \in \{0, 1, \#\}^*$$

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 property that is decidable in terms of the deatomisation, such as "a prime number of distinct atoms, each one appearing a non-prime number of times", can be computed by a Turing machine with atoms.

The idea of deatomisation is quite specific to sequences of atoms. For sequences over other definable alphabets, the notion is less clear. As we will later see in Theorem 10.7, a deterministic deatomisation procedure is not going to be possible for some definable alhabets. □

Example 54. [A more fancy input alphabet] Consider the equality atoms. Let the input alphabet be sets of atoms of size at most ten. We show a deterministic Turing machine that 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. However, a deterministic Turing machine cannot "indicate" the atom $\underline{1}$, since it has the same properties as the atom $\underline{2}$.

Here is a solution to the problem. When an input word is fixed, consider two atoms equivalent if they appear in exactly the same letters of the word. In the example above, the equivalence classes are

$$\{\underline{1}, \underline{2}\}\ \{\underline{3}\}\ \{\underline{4}\}.$$

(We ignore the infinite equivalence class that contains all atoms, which do not appear in the input word.) A Turing machine for the language works as follows. First, it computes the equivalence classes (each equivalence class can be stored in a cell of the tape, since it is a set of at most ten atoms). Then it nondeterministically chooses one of these equivalence classes, loads it into its state (to have a deterministic machine, each one can be tried), and sees if it appears in an odd number of positions in the input word. □

Exercise 98. 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 (i.e. semi-decidable) 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 99. 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 100. 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 101. 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 $\mathbb{A}$.

10.1 Computational completeness of alternating machines

Recall Theorem 8.6, which described two equivalent characterisations of computable functions from definable sets to definable sets. Let us specialise this concept to the case of languages over a definable alphabet. Assume that the input alphabet Σ is a definable set. It is not difficult to show that every word $w \in \Sigma^*$ is a definable set, since a sequence of definable letters is also definable. View a language as its characteristic function

$$L : \Sigma^* \to \{\text{yes}, \text{no}\}$$

and extend this characteristic function to all definable sets by giving a "no" answer to every definable set which is not in Σ^*. We say that $L \subseteq \Sigma^*$ is *computable* if the function obtained this way is computable in either of the two equivalent senses from Theorem 8.6. Before continuing, let us give an alternative definition of computability which is equivalent, but more adapted to the setting of words and Turing machines.

Recall that by definition, a definable alphabet Σ is given by a set builder expression $\alpha(\bar{x})$ together with a tuple of atoms $\bar{a}$ that instantiates its free variables $\bar{x}$. The expression α cannot be an atom, since otherwise the alphabet has no letters. Therefore, it must be a finite union of set expressions:

$$\alpha(\bar{x}) = \bigcup_{i \in I} \{\alpha_i(\bar{x}\bar{y}_i) : \text{for } \bar{y}_i \text{ such that } \varphi_i(\bar{x}\bar{y}_i)\}.$$

In order to provide a letter in Σ, one needs to indicate $i \in I$ and a tuple of atoms $\bar{b}$ that evaluates the tuple of variables $\bar{y}_i$; the letter is then equal to $\alpha_i(\bar{a}\bar{b})$. Note that the length of the tuple $\bar{b}$ might depend on i. Thanks to the assumption that the atoms are effective, a tuple of atoms can be written down as a bit string. Define a *representation* of a word $w \in \Sigma^*$ to be a bit string describing the sequence of representations of letters as defined above.

Example 55. Consider the equality atoms, represented as bit strings, and the alphabet

$$\underbrace{\{\{y_1, y_2\} : \text{for } y_1, y_2 \text{ such that } y_1 \neq y_2\}}_{\text{expression 1}} \cup \underbrace{\{\underline{2}, \underline{3}\}}_{\text{expression 2}}$$

One possible source of ambiguity in representations is that an element might be captured by two different expressins, e.g. $\{\underline{1}, \underline{2}\}$ in the above example. This problem could be avoided by rewriting the expressions. Another problem, is that even if the expression is known, then two different valuations of the free variables might give the same result. For example, a letter $\{\underline{2}, \underline{5}\}$ is encoded by writing down first the expression number – necessarily expression 1, in this particular example – then giving a valuation of the variables which yields this letter. Note that there are two possible valuations, namely

$$(y_1, y_2) \mapsto (\underline{2}, \underline{5}) \quad (y_1, y_2) \mapsto (\underline{5}, \underline{2})$$

This source of ambiguity could be eliminated by choosing the lexicographically least valuation. Since ambiguity of representation will not play a role in our discussion, we will not make an effort to eliminate it. □

Lemma 10.3. *Assume that the atoms are effectively oligormorphic. If* Σ *is a definable alphabet, then the following conditions are equivalent for every language* $L \subseteq \Sigma^*$*:*

(1) *L is finitely supported and* $\{w \in 2^* : w \text{ represents a word from } L\}$ *is semi-decidable in the usual sense without atoms; and*
(2) *L is computed by a definable while program.*

Proof Using Theorem 8.6. □

The main goal of this section is to give a third equivalent item in the above lemma, namely recognised by a Turing machine. However, in our proof (at least for the general case of arbitrary effective atoms structures), we will need to use *alternating* Turing machines. Already for the simplest equality atoms a deterministic machine will not be enough.

Alternating machines. We begin by describing the alternating model, which we call *definable alternating Turing machines*. The syntax 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. (By Exercise 60, a run is well-founded if and only if there is some finite bound $n \in \mathbb{N}$ on the length of all paths; this bound is called the *depth* of the run.) An accepting run over an input word is a run where the root has the initial configuration (i.e. the 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 words which admit at least one accepting run.

The main result in this section is the following theorem, which shows that definable alternating Turing machines are computationally complete for languages over definable alphabets. Furthermore, nondeterministic machines are enough when the atoms have quantifier elimination, i.e. every formula of first-order logic is equivalent to a quantifier-free one.

Theorem 10.4. *Assume that the atoms $\mathbb{A}$ are effectively oligomorphic. Then the following conditions are equivalent for every $L \subseteq \Sigma^*$ where Σ is definable:*

(1) *L is finitely supported and $\{w \in 2^* : w$ represents a word from $L\}$ is semi-decidable in the usual sense without atoms; and*
(2) *L is computed by a definable while program.*
(3) *L is recognised by a definable alternating Turing machine.*

If $\mathbb{A}$ has quantifier elimination, then the above conditions are also equivalent to

4. L is recognised by a definable nondeterministic Turing machine.

The theorem implies that for atoms such as $(\mathbb{N}, =)$ or $(\mathbb{Q}, <)$, nondeterministic Turing machines are computationally complete for languages over definable alphabets. Actually, the assumption on quantifier elimination can be relaxed to obtain item 4, see Exercise 102. A scenario consistent with the author's knowledge, although unlikely, is that item 4 is equivalent to the other items for all effectively oligomorphic structures.

The rest of Section 10.1 is devoted to proving Theorem 10.4. The equivalence of items 1 and 2 is Lemma 10.3. The implication from 3 to 2 is straightforward, by using a definable while program to enumerate candidates for accepting runs. It remains to prove the implication from 1 to 3 (and the implication from 1 to 4 under the additional assumption on quantifier elimination.)

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 (which is part of the definition of being effective), then the following language over alphabet* $\mathbb{A} + \{0, 1\}$ *is recognised by an alternating Turing machine:*

$$\{a_1 \cdots a_n \underline{\varphi} : a_1, \ldots, a_n \in \mathbb{A},\ \varphi \text{ has } n \text{ free variables, and } \mathbb{A} \models \varphi(a_1, \ldots, a_n)\ \}.$$

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 Exercise 53, 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 $\mathbb{A}$.

Lemma 10.6. *Assume that the atoms are effective. Let* $L \subseteq \mathbb{A}^*$ *satisfies item 1 in Theorem 10.4 then it is recognised by a definable alternating Turing machine. If* $\mathbb{A}$ *has has quantifier elimination, then a deterministic Turing machine is enough.*

Proof The assumption that L satisfies item 1 says that L is finitely supported, say by some atom tuple $\bar{a}$, and there is a Turing machine M without atoms which recognises the set $\{w \in 2^* : w \text{ represents a word from } L\}$. We claim that an alternating Turing machine can reverse the representation up to $\bar{a}$-automorphisms. More precisely, we claim that an alternating Turing machine can compute a function

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

such that for every $w \in \mathbb{A}^*$, the bit string $f(w)$ represents some word in $\mathbb{A}^*$ that is in the same $\bar{a}$-orbit as w. Once we have proved this, we can compose f with M to get a machine recognising L (this works by the assumption that the language L is invariant under $\bar{a}$-automorphisms).

To compute the function f, we use the assumption that the atoms are effectively oligomorphic. Suppose that the input to f is $\bar{b} \in \mathbb{A}^*$. Let n be the length of the tuple $\bar{a}\bar{b}$. Using the assumption on effective oligomorphism, we can compute finitely many formulas with n variables which describe all orbits

of n-tuples. Furthermore, if the atoms admit quantifier elimination, then these formulas are quantifier free. Using Lemma 10.5, we can determine which formula φ describes the orbit of $\bar{a}\bar{b}$. Finally, using the assumption that the atoms have decidable first-order model checking, we can find the first bit string that represents some $\bar{c}$ such that $\varphi(\bar{a}\bar{c})$ holds; this word bit string is the value of the function f on input $\bar{b}$. □

We are now ready to complete the proof of Theorem 10.4. Let $L \subseteq \Sigma^*$ be a language which satisfies condition 1. Like for any definable set Σ, there exists some k and a surjective function

$$f : \mathbb{A}^k \to \Sigma$$

whose graph is a definable set. 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^* is also computable. 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.

Exercise 102. Call two structures *equivalent* if they have the same universe and the same automorphism group. Show that following conditions are equivalent for every effectively oligomorphic structure $\mathbb{A}$:

(1) nondeterministic Turing machines recognise the same languages as alternating ones;
(2) $\mathbb{A}$ is equivalent to 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 equality atoms, Turing machines do not determinise

In Section 10.1 we showed that alternating Turing machines are computationally complete. When the atoms admit quantifier elimination, which is the case e.g. for equality atoms, then already nondeterministic Turing machines are

computationally complete. In this section we show that in the equality atoms, deterministic and nondeterministic Turing machines have different expressive power.

Theorem 10.7. *Assume that the atoms are* $(\mathbb{N}, =)$*. There is a definable alphabet* Σ *and a language* $L \subseteq \Sigma^*$ *that is recognised by a nondeterministic definable Turing machine, but which is not recognised by any deterministic definable Turing machine.*

Recall that our notion of recognition is the one corresponding to semi-decidable languages, i.e. the Turing machines are not required to terminate on rejected inputs. In the proof of the theorem, we will see that the nondeterministic machine which recognises L runs in polynomial time, i.e. on every accepted input it has some accepting run with a polynomial number of computation steps. A consequence of the theorem is that, in the equality atoms, NP is not contained in the class of deterministic semi-decidable languages, and hence also NP≠P. However, since computation with atoms is so different from computation without atoms, Theorem 10.7 is unlikely to shed new light on the power of nondeterminism without atoms.

The rest of Section 10.2 is devoted to proving Theorem 10.7.

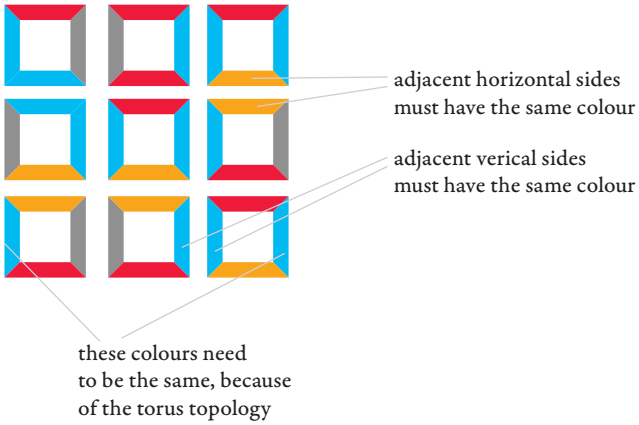
Tilings. In our proof we a type of tiling problem, and therefore we begin by defining terminology for tilings. If C is a set of colours, then a *tile over colours* C is an element of C^4, which is visualised as a square with each side coloured by a colour from C:

We arrange tiles in square grids with a torus topology, i.e. with the last row adjacent to the first row, and likewise for columns. For $n \in \mathbb{N}$, let us define

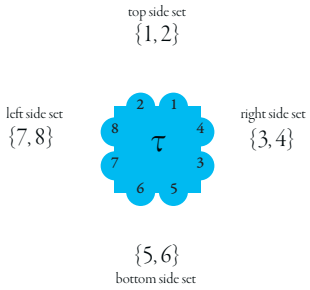
$$n \times n \stackrel{\text{def}}{=} \underbrace{\{0, 1, \ldots, n-1\}}_{\text{columns}} \times \underbrace{\{0, 1, \ldots, n-1\}}_{\text{rows}}.$$

The adjacency relation is defined using arithmetic modulo n, i.e. the left neighbour of a grid position is obtained by decrementing the first coordinate modulo n, likewise for right, upper and lower neighbours. If C is a set of colours, then an $n \times n$ *tiling over colours* C is defined to be any function from $n \times n$ to tiles

over colours C. A tiling is called *consistent* if adjacent pairs of tiles have the same colour on the shared side, as explained in the following picture for 3×3:



The separating language. For the proof of Theorem 10.7, we will discuss tilings where the set of colours is the set $\mathbb{A}^{(2)}$ of ordered pairs of distinct atoms. To define the separating language, the key is the following equivalence relation on tiles. For a tile τ over colours $\mathbb{A}^{(2)}$, define a *side set* to be a set of two atoms that can be found on one of its four sides, as explained in the following picture:

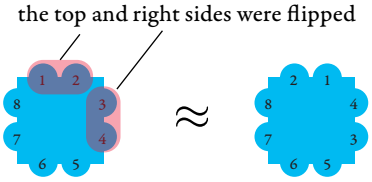


Define a *flip* of τ to be one of the four tiles that can be obtained from τ by choosing some side set and applying the atom automorphism which swaps the pair of atoms from this side set. We will only apply flips when τ belongs to

$$\Sigma \quad \stackrel{\text{def}}{=} \quad \text{tiles over colours } \mathbb{A}^{(2)} \text{where all four side sets are pairwise disjoint.}$$

and therefore a flip can be seen as affecting only one of the sides. Consider two tiles from Σ to be $\approx$-equivalent if one can go from one to the other by doing an

even number of flips. Here is a picture of equivalent tiles, where two flips were used:



An equivalence class of $\approx$ class consists of 8 tiles: 0 flips can be done in 1 way, 2 flips can be done in 6 ways and 4 flips can be done in 1 way. Define $\Sigma_{/\approx}$ to be the set of $\approx$-tiles; this is a definable set. We are now ready to define the separating language. The reason for the name CFI is explained in the bibliographic notes at the end of the section.

Definition 10.8 (CFI property)**.** Define an $n \times n$ $\approx$-tiling to be a function

$$\mathcal{T} : n \times n \to \Sigma_{/\approx}.$$

We say that $\mathcal{T}$ satisfies the CFI property if there exists a consistent tiling

$$\mathcal{S} : n \times n \to \Sigma$$

such that $\mathcal{T} = \mathcal{S}/_{\approx}$, where is defined by $x \mapsto [\mathcal{S}(x)]_{\approx}$.

Formally speaking, the separating language required for Theorem 10.7 should be a set of words, and not $\approx$-tilings. Therefore we assume some convention on linearly ordering the tiles in an $\approx$-tiling, e.g. the tiles are ordered lexicographically, 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 $\Sigma_{/\approx}$. The separating language from Theorem 10.7 is defined to be the set of all encodings of $\approx$-tilings that satisfy the CFI property.

The language L is clearly recognised by a nondeterministic Turing machine. The work alphabet of the machine is $\Sigma_{/\approx} \cup \Sigma$ plus some 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.

L is not recognised by any deterministic Turing machine. To finish the proof of Theorem 10.7, it remains to show that L is not recognised by any deterministic Turing machine. We begin by discussing a natural doubt the reader

might have at this point. Given an input representing an $\approx$-tiling $\mathcal{T}$, there are only finitely many (even if exponentially many) possibilities for choosing the witness $\mathcal{S}$ as in Definition 10.8. Why not then 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 there are only eight elements in an equivalence class $\tau \in \Sigma_{/\approx}$, 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 language L is not recognised by any deterministic Turing machine. This will be a consequence of Lemma 10.10 below, which says that a deterministic Turing machine, unlike the CFI property, is insensitive to flipping tiles in an $\approx$-tiling.

We begin by giving the definitions that are used in lemma.

For tile $\sigma \in \Sigma$, define $[\sigma]$ to be the tile over colours $\binom{\mathbb{A}}{2}$ which is obtained from σ by forgetting the order information about the pair on each side. It is not difficult to see that $\sigma \approx \sigma'$ implies $[\sigma] = [\sigma']$, and hence it is meaningful to define $[\tau]$ for an $\approx$-tile τ by

$$[\tau] \stackrel{\text{def}}{=} [\sigma] \quad \text{for some, equivalently every, } \sigma \text{ in the equivalence class } \tau$$

We call a $\approx$-tiling $\mathcal{T}$ *weakly consistent* if the tiling

$$[\mathcal{T}] : n \times n \to \text{tiles over colours } \binom{\mathbb{A}}{2} \quad \text{defined by } x \mapsto [\mathcal{T}(x)]$$

is a consistent tiling over colours $\binom{\mathbb{A}}{2}$, and furthermore each atom appears in at most two tiles. If $\mathcal{T}$ is weakly consistent then every atom either does not appear in $\mathcal{T}$, or appears exactly twice, in adjoining side sets corresponding to some edge.

Define the *flip* of an $\approx$-tile $\tau \in \Sigma_{/\approx}$ to be the set

$$\bar{\tau} \stackrel{\text{def}}{=} \{\sigma \in \Sigma : \text{doing an odd number of flips on } \sigma \text{ yields a tile in } \tau\}.$$

It is not difficult to see that the above set is also an $\approx$-tile, i.e. flipping can be viewed as an operation on $\approx$-tiles. This operation is an involution, since flipping twice has no effect. The following lemma formalises the statement that the CFI property is sensitive to flips.

Lemma 10.9. *If $\mathcal{T}$ is an $n \times n$ weakly consistent $\approx$-tiling, then for every $x \in n \times n$, the following $\approx$-tiling is weakly consistent and violates the* CFI *property:*

$$\mathcal{T}_x(y) = \begin{cases} \overline{\mathcal{T}(y)} & \text{if } y = x \\ \mathcal{T}(y) & \text{otherwise} \end{cases}$$

Proof Flips do not modify side sets, and since weak consistency is defined entirely in terms of sides sets, it follows that doing a flip preserves the property of being weakly consistent. To show that a flip results in a violation of the CFI property, we use a parity argument. For $\mathcal{S} : n \times n \to \Sigma$ define the *conflict set* to be the set of edges e in the $n \times n$ grid such that the colours of the two sides adjoining on e are different. Using this terminology, an $\approx$-tiling $\mathcal{T}$ satisfies the CFI property if and 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 L is not recognised by any definable deterministic Turing machine. We use the following notation: if $\mathcal{T}$ is an $\approx$-tiling and M is a definable deterministic Turing machine whose input alphabet $\Sigma_{/\approx}$, then we write $M_{\mathcal{T}}$ for the unique computation of M on the word representing $\mathcal{T}$. We use the formalisation of computations from (10.1) at the beginning of Section 10, i.e. a computation is a square array of cells, each one containing a letter of the work alphabet, or a pair (letter of the work alphabet, state of the machine). The following lemma, together with Lemma 10.9 immediately implies that no deterministic definable Turing machine can recognise the language L, thus completing the proof of Theorem 10.7.

Lemma 10.10. *For every definable deterministic Turing machine M with input alphabet $\Sigma_{/\approx}$ there exists $k \in \mathbb{N}$ with the following property. Let*

$$\mathcal{T} : n \times n \to \Sigma_{/\approx} \qquad \text{with } n > 2k$$

be weakly consistent. Assuming the notation $\mathcal{T}_x$ defined in Lemma 10.9, the following holds for every $i, j \in \mathbb{N}$:

$$M_{\mathcal{T}}(i, j) = M_{\mathcal{T}_x}(i, j) \qquad \text{for all } x \in n \times n \text{ with at most } k^2 \text{ exceptions} \qquad (*)$$

Proof Let Γ be the work alphabet of the machine, and let Q be its states. Choose k such that for every element s of

$$\Gamma \cup \{\text{blank}\} \cup (\Gamma \cup \{\text{blank}\}) \times Q \qquad (10.2)$$

there exists a tuple of at most $k/2$ atoms which supports both s and the Turing machine M. Such a number can be chosen, because M has some fixed finite support, while s ranges over an orbit finite set, and hence there is a common upper bound for the size of supports needed for s.

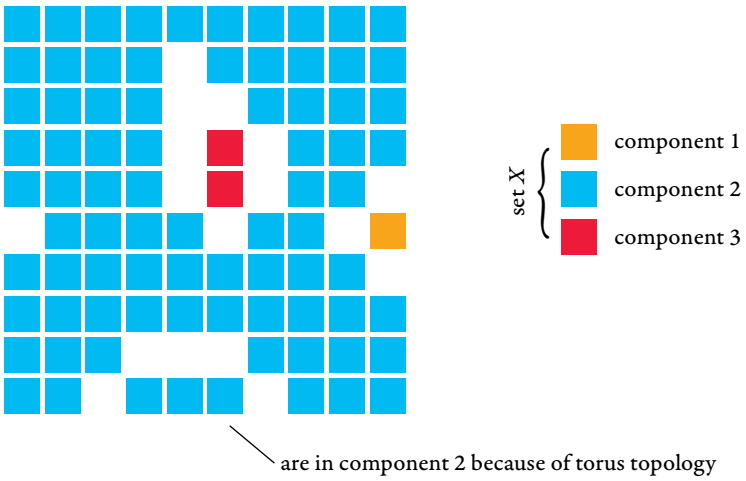
We prove (*) by induction on i. For the induction base of $i = 0$, we observe

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:

$$M_{\mathcal{T}}(i, j) = M_{\mathcal{T}_x}(i, j) \qquad \text{for all } x \in n \times n \text{ with at most } 2k^2 \text{ exceptions} \quad (**)$$

In the rest of this proof, we bring down the number of exceptions to k^2. To do this, we will talk about connected components in the square grid after removing some grid positions. (The square grid refers to the tiling, and not the computation of the machine.) To make these notions precise, we view $n \times n$ as a graph, where the vertices are grid positions, and two grid positions are connected by an edge if they are adjacent in the (torus) grid topology. For a subset $X \subseteq n \times n$, define its connected components to be the connected components in the subgraph of $n \times n$ 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 (*). By choice of k, we can find a tuple of atoms $\bar{a}$ which has size at most $k/2$ and supports both the Turing machine M and $M_{\mathcal{T}}(i, j)$, the latter element belonging to (10.2). Define

the following set of grid positions:

$$Z = \{z \in n \times n : \text{some atom from } \bar{a} \text{ appears in } \mathcal{T}(z)\}.$$

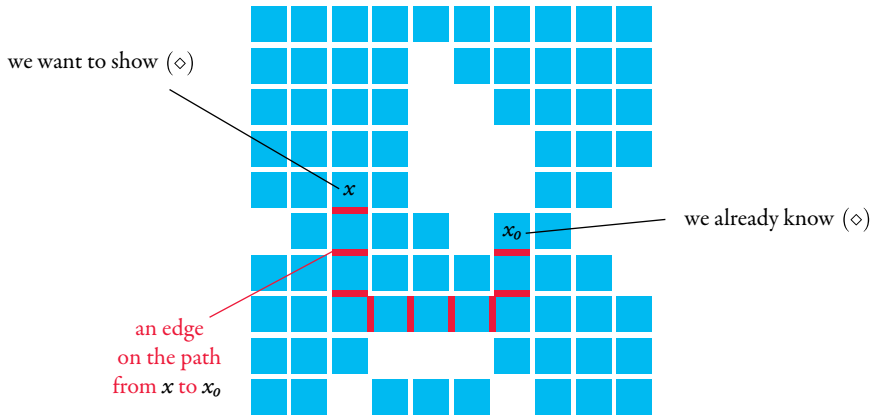
The set Z has at most k grid positions, by the assumption that every atom appears in at most two grid positions from $\mathcal{T}$. From the assumption that $n > 2k$ (actually $n > k$ is enough for this part of the argument) and a straightforward analysis of connectivity in an $n \times n$ grid, one can conclude that the graph corresponding to $n \times n - Z$ has a connected component which contains all grid positions from $n \times n$ with at most k^2 exceptions. Let X be this connected component. Because X omits at most k^2 grid positions, to prove (*), it will suffice to show that

$$M_{\mathcal{T}}(i, j) = M_{\mathcal{T}_x}(i, j) \tag{$\diamond$}$$

holds for every $x \in X$. Observe first a weaker existential version of the above: there exists some $x_0 \in X$ which satisfies ($\diamond$). This is because we have

$$\underbrace{k^2}_{\text{number of exceptions in (**)}} \quad < \quad \underbrace{n^2 - k^2}_{\text{size of } X}$$

which in turn follows from the assumption that $n > 2k$. Let us now show that ($\diamond$) holds for every $x \in X$, not necessarily equal to x_0. 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:



By the assumption that $\mathcal{T}$ is weakly consistent, to every edge e of $n \times n$ we can associate an unordered set of atoms $S_e \in \binom{\mathbb{A}}{2}$, such that the sets S_e are disjoint for different edges. Define π to be the atom automorphism which flips each unordered set S_e, with e ranging over edges on the path from x to x_0. For

each tile except those corresponding to x, x_0, the automorphism flips an even number of side sets, and hence we have:

$$\mathcal{T}_x = \pi(\mathcal{T}_{x_0}). \tag{10.3}$$

Recall the tuple of atoms $\bar{a}$. The path from x to x_0 was chosen so that all of the sets S_e are disjoint with $\bar{a}$, and therefore π is a $\bar{a}$-automorphism. Because $\bar{a}$ was chosen so that it supports both the machine M and the value $M_{\mathcal{T}}(i, j)$, we have

$$\begin{aligned} M_{\mathcal{T}_x}(i, j) &= (\text{by (10.3)}) \\ M_{\pi(\mathcal{T}_{x_0})}(i, j) &= (\bar{a} \text{ supports } M) \\ \pi(M_{\mathcal{T}_{x_0}}(i, j)) &= (x_0 \text{ satisfies } (\diamond)) \\ \pi(M_{\mathcal{T}}(i, j)) &= (\bar{a} \text{ supports } M_{\mathcal{T}}(i, j)) \\ M_{\mathcal{T}}(i, j) & \end{aligned}$$

which completes the proof of $(\diamond)$ for x and thus also proves the lemma. □

Exercise 103. In the proof of Theorem 8.6, 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.

Exercise 104. 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 105. Assume the equality atoms and consider the alphabet

$$\{\{\{a, b, c\}, \{d, e, f\}\} : a, b, c, d, e, f \text{ are distinct atoms}\}.$$

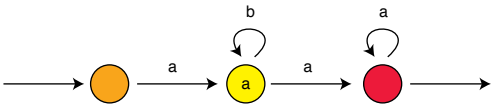
Show that Turing machines over this input alphabet do not determinise.

Bibliographic notes for Section 2. Turing machines with atoms were introduced in Bojańczyk et al. (2013a), which also contained the proof of Theorem 10.7. The separating language in Theorem 10.7 is a variant of the Cai-Fürer-Immerman construction yi Cai et al. (1992), it is also motivated by a construction from model theory (Cherlin and Lachlan, 1985, example on p. 819). A further study of determinisation of Turing machines was done in Klin et al. (2014); in particular Section 5.1 of that paper gives a solution for the generalisation of Exercise 104 to the optimal value of $k = 5$. Theorem 10.4 on the computational universality of alternating Turing machines is new in these notes.

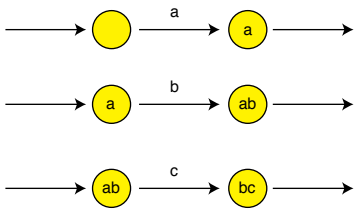
PART FOUR

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 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.

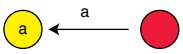
Solution to Exercise 3.
Consider the language

$$\{ab(c^n) : a, b, c \in \mathbb{A} \text{ are distinct and } n \in \mathbb{N}\}$$

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 4.
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 5.

Solution to Exercise 5.
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 6.
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 7.

(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.

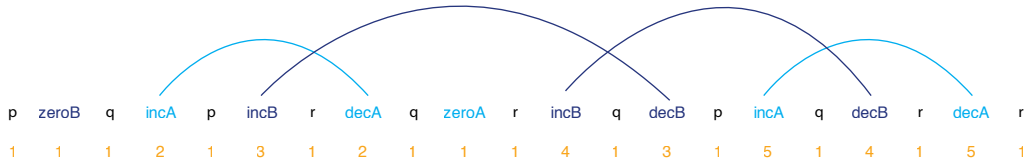
(ii) For every state q_1 in orbit Q_1 there exists a state q_2 in orbit Q_2 such that some transition leads from q_1 to q_2 in one step.

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 8.
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 1.7. 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 9.
One can write a formula which is true exactly in the encodings of runs of Turing machines as used in Theorem 1.7. Alternatively, one can write a formula which is true exactly in the encodings of runs of Minsky machines as used in Exercise 8.

Solution to Exercise 10.
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 11.
The right-to-left implication is immediate, because infinite antichains and in-

finite 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 12.
Using Exercise 11 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

$$x_1, x_2, \ldots \in \mathbb{N}^{d+1}.$$

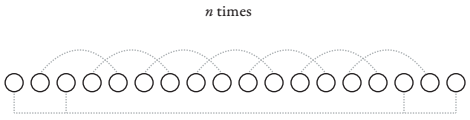
By the induction assumption, there is an infinite subsequence such that the projection onto the first coordinate is monotone. By induction assumption again, that subsequence has an infinite subsequence where the projection onto the remaining coordinates is monotone, and the result follows.

Solution to Exercise 13.
See (Schmitz and Schnoebelen, 2012, Exercise 1.10)

Solution to Exercise 14.
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 1.11, 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 15.
Using Theorem 1.15 and the Higman ordering on configurations.

Solution to Exercise 16.
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 17.
Consider the set W of words in the solution to Exercise 16. 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 lan-

guage 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 18.
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 19.
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.

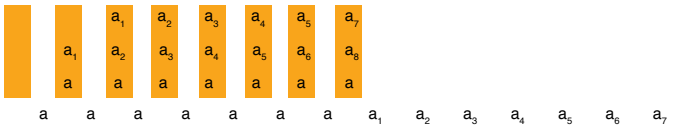
Solution to Exercise 20.
This solution comes from the Master's thesis of Tomasz Wysocki Wysocki (2013). Consider the following language over $\mathbb{A}$:

$$\{a^n a_1 \cdots a_n : n \in \mathbb{N} \text{ and } a, a_1, \ldots, a_n \in \mathbb{A} \text{ are all distinct}\}$$

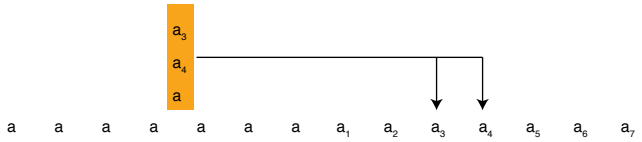
Let us first argue that an alternating register automaton without guessing cannot recognise the language. After reading a prefix of the form a^n, the bag can only have a in its registers. Since there are finitely many possibilities for such bags, there must be some $n < m$ such that the set of reachable 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):



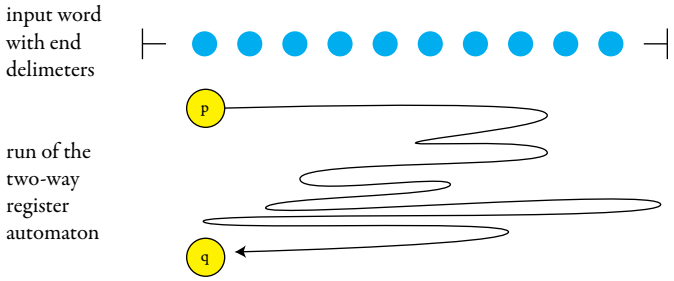
The contents of the registers are above are guessed, but they are verified using alternation: the initial state is universal, and in each step it spawns off a parallel thread that checks if the current configuration corresponds to two consecutive data values in the future, as in this picture:



Solution to Exercise 21.
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

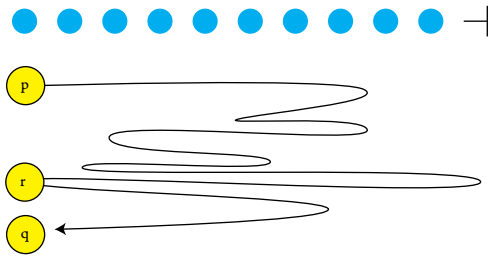
$$p, q \in \mathsf{Loc} \times \text{register valuations}$$

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 $\vdash$.



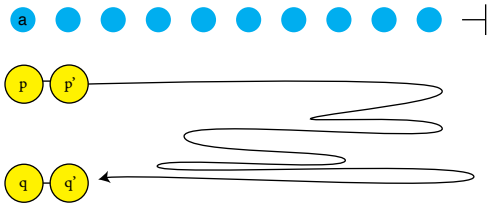
The crucial point is to recognise loops: we will sketch that there is an alternating register automaton, such that if is initialised in a state that stores both p and q, then it accepts if and only if there is a (p, q)-loop. Once loops are recognised, it is not difficult to simulate the two-way automaton (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 22.
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 23.
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

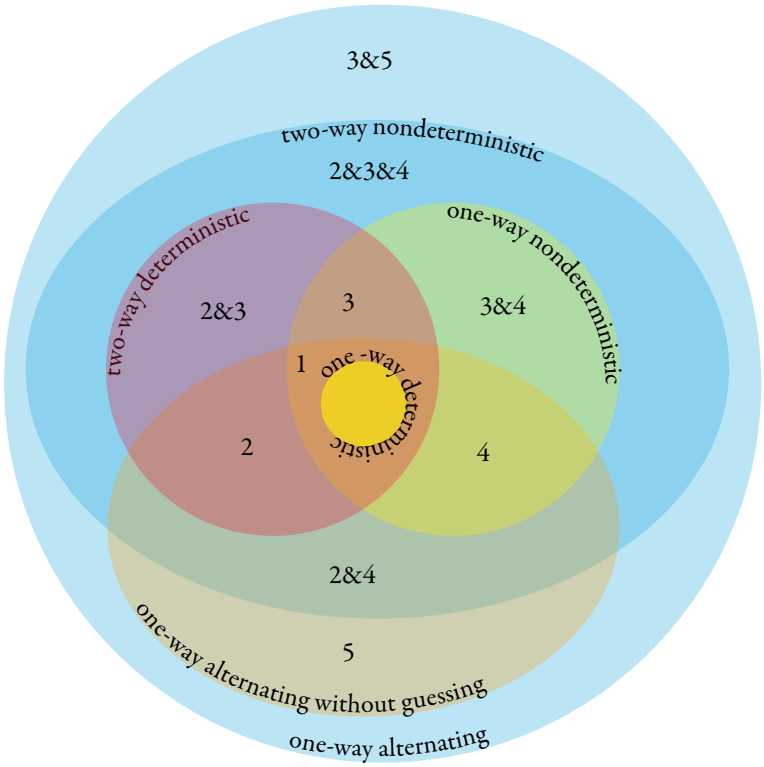
$$a_1 b_1 \cdots a_n b_n \in \mathbb{A}^*$$

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 24.
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 23, 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 25.
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 26.
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 8. 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 27.
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 Pressburger arithmetic, which is decidable.

Solution to Exercise 28.
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 2.7 to get the desired result.

Solution to Exercise 29.
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 28 and a labelling of every position by its offset from the beginning modulo k.

Solution to Exercise 30.
With the more powerful model from this exercise, one can recognise computations of counter machines as used in Exercise 8. This would contradict Theorem 2.6 that emptiness is decidable for data automata.

Solution to Exercise 31.
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 2.10. *A data word belongs to the language in the exercise if and only if:*

(1) *every class string satisfies the following expression:*

$$((\underbrace{open}_{\text{opening but not closing}} \overbrace{middle^*}^{\text{remaining cases}} \underbrace{close)}_{\text{closing but not opening}} + \underbrace{clopen}_{\text{opening and closing}} + \underbrace{\bot}_{\text{not chosen}})^*$$

(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.

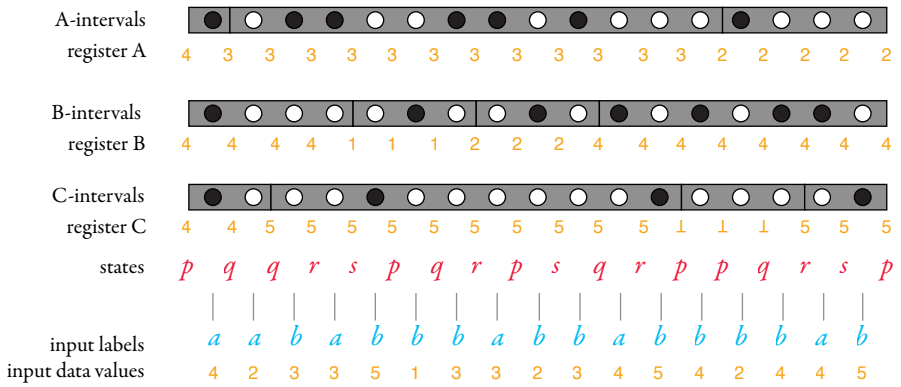
We now show that top-down implication. Condition 1 is easy to see, so we focus on condition 2. We say that two intervals I and J are in conflict if I contains the class predecessor (i.e. previous position in the same class) of the

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). □

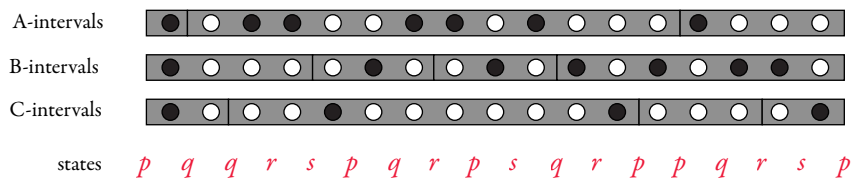
Solution to Exercise 32.
For the sake of this exercise, we consider a model of register automaton where undefined registers are not allowed. The initial configuration has the initial state and the first data value in all the registers. It is not difficult to see that this model is equivalent to the original 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 6. 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 31, 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 33.
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 34.
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 35.
For undecidability, we could extend the idea from Exercise 34 to recognise use the encoding of Turing machine computations from Theorem 1.7. 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 Theorem 2.11.

Solution to Exercise 36.
We refer to the two logics in the exercise as "two variable logic" and "modal logic". Let us only do the more interesting direction from two variable logic to modal logic, the opposite translation is simply a formalisation of the definition. We claim that for every formula of two variable logic $\varphi(x)$ with one free variable there is a formula of the modal logic that selects the same positions. Once this claim has been proved, the result follows, since a sentence (i.e. formula without free variables) of two variable is of the form $\exists x\, \varphi(x)$, and therefore the corresponding formula of modal logic is

$$\varphi' \vee \bigvee_m \langle m \rangle \varphi'$$

where φ' is the formula of modal logic equivalent to $\varphi(x)$ and m ranges over all modalities. The proof of the claim is by induction on formula size. The interesting case is when $\varphi(x)$ uses a quantifier, say

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

The formula $\psi(x, y)$ is a Boolean combination of formulas that are predicates, negations of predicates, or begin with a quantifier. By converting this Boolean combination to DNF, and distributing disjunction across $\exists$ we can see that $\varphi(x)$

is equivalent to a disjunction of formulas of the form

$$\exists y \bigwedge_{i \in I} \psi_i(x, y)$$

where each ψ_i is either a predicate or its negation, or it begins with a quantifier. Let $J \subseteq I$ be those indices where ψ_i has two free variables, this corresponds to the binary predicates and their negations. Without loss of generality we can assume that if $i \in I - J$, then the free variable of φ_i is y, since otherwise we could move φ_i out of the scope of the quantifier $\exists y$. Therefore we have rewritten the formula as

$$\exists y \quad \bigwedge_{j \in J} \psi_j(x, y) \ \wedge \ \bigwedge_{i \in I - J} \psi_i(y)$$

To the second conjunction we can apply the induction assumption, yielding an equivalent formula of the modal logic, call it ψ. If the first conjunction is inconsistent, then the whole formula can be replaced by some unsatisfiable formula of the modal logic. If the first conjunction is consistent, then it is equal to a disjunction of modalities $m_1 \vee \cdots \vee m_n$, in which case the entire formula is equivalent to

$$\bigvee_{i \in \{1,\ldots,n\}} \langle m_i \rangle \psi$$

Solution to Exercise 37.
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 38.
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 39.
All of the four equivariant relations mentioned in Example 38 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 40.
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 41.
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 42.
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 43.
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 44.
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 45.
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 46.
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 3.8 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 47.
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 3.4 are equivalent, it follows that $\mathbb{A}^{n+1}$ has finitely many $\bar{a}$-orbits for every tuple of atoms.

Solution to Exercise 48.
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 49.
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 50.
We begin with the following observation.

Claim 3.9. *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 $\bar{a}\bar{b} \in R$

depeonds 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 51.
Consider the total order atoms, and the relation $a < b$ contained in $\mathbb{A}^2$. There is no finitely supported function which maps each atom to a strictly bigger one.

Solution to Exercise 52.
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 48. 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 48 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 53.
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 3.8 it follows that the range of the function is orbit-finite.

Solution to Exercise 54.
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 53.

Solution to Exercise 55.
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 56.
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 3.10. *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 3.11. *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 57.
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 58.
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 59.
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 58 and therefore it is equivalent to orbit-finiteness. Actually, we show a stronger property.

Lemma 3.12. *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 60.
Same proof as for the standard lemma, plus this observation: the depth of a subtree is invariant under applying atom automorphisms.

Solution to Exercise 61.
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 62.

- *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 59, 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 63.

(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 78.)

(2) *One can reach a finite cycle that intersects T.* Condition (2) is checked using the set structure from Lemma 5.5.

Solution to Exercise 64.
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 65.
First observe that the transitive closure of a hereditarily definable binary relation is not necessarily hereditarily definable. As an example, consider the successor relation $\{(n, n+1) : n \in \mathbb{N}\}$. This relation is clearly hereditarily definable, but its transitive closure is the order relation, which is not hereditarily definable.

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 66.
The usual proof works. It is important that transitive closure preserves orbit-finiteness.

Solution to Exercise 67.
Determinism can be formalised in first-order logic and then checked using the Definable Relation Lemma. 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 68.
Consider a nondeterministic definable 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 69.
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 70.
The standard proof works. This is because the standard proof does not change the state space, only it adds transitions, initial states and final states. The new bigger set of transitions is still finitely supported.

Solution to Exercise 71.
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 54.

Solution to Exercise 72.
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 1.7, is also seen to be an orbit-finite intersection of languages recognised by orbit-finite deterministic automata.

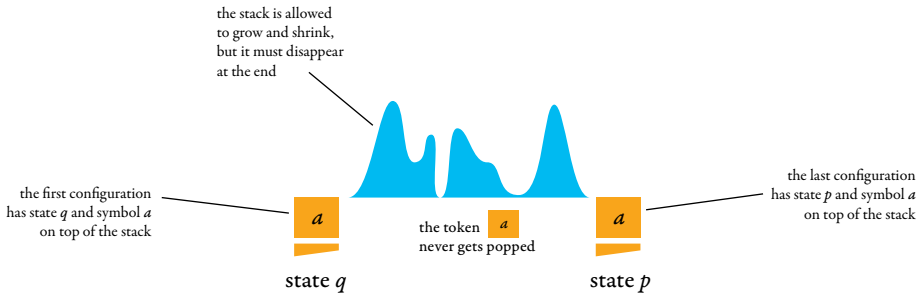
Solution to Exercise 73.
Intuitively speaking, the problem is that intersection corresponds to product on automata, and we cannot do orbit-finite products. Here is the counterexample. For every $a \in \mathbb{A}$, the language "a appears at most once" is recognised by a (deterministic) orbit-finite automaton. If we could intersect all these languages, then we would get a nondeterministic automaton for the language "all letters are distinct". By Exercises **??**, this would mean that "all letters are distinct" could be recognised by a register automaton, which is not the case.

Solution to Exercise 74.
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 75.
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 5.14. *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*

$$q \xrightarrow{read(b)} p \quad \text{implies} \quad q \overset{a}{\Rightarrow} b$$

$$q \xrightarrow{fresh(b)} p \quad \text{implies} \quad q \overset{a}{\Rightarrow} b \qquad \text{if } b \text{ is fresh with respect to } q, a, \bar{c}$$

$$q \xrightarrow{push(a')} q' \overset{a'}{\Rightarrow} p' \xrightarrow{pop(a')} p \quad \text{implies} \quad q \overset{a}{\Rightarrow} b$$

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 76.

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 77.
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 $w = w_1w_2w_3$, with w_2 and w_1w_3

nonempty such that

$$w_1(\pi \cdot w_2)w_3$$

is a palindrome for every $\bar{a}$-automorphism π.

Solution to Exercise 78.
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 5.19. *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 79.
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 4.7, 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 80.
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

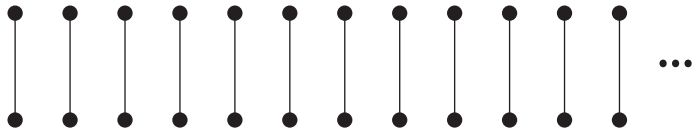
$$S_i \subseteq (V_i \times V) \cap E$$

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 81.
See (Bojańczyk et al., 2014, Corollary 9.5).

Solution to Exercise 82.
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 83.
This exercise is based on (Colcombet et al., 2015, Lemma 2.14). Let $\bar{a}$ be the least support of the group, i.e. the tuple consisting of theorbit finiteerse, the multiplication operation and the inverse function $g \mapsto g^{-1}$. (Actually,orbit finite supports the multiplication operation, then it supports the universe and the inverse.) For an element g of the group, define $[g]$ to be its least support minus the atoms that appear in $\bar{a}$. It is not difficult to see every elements g, h of the group satisfy

$$[gh] \subseteq [g] \cup [h] \qquad [g^{-1}] \subseteq [g] \tag{6.1}$$

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 orbit finiteomorphism π so that

$$\pi([g]) \cap [g] = \emptyset \tag{6.2}$$

We have

$$g = \pi(g)\pi(g)^{-1}g.$$

Combining this with (6.1) we get

$$[g] \subseteq [\pi(g)] \cup [\pi(g)^{-1}g]$$

Combining this with (6.2), we get

$$[g] \subseteq [\pi(g)^{-1}g]$$

By maximality of $[g]$ the above is actually an equality. Using a similar reasoning applied to

$$\pi(g)^{-1} = g^{-1}\pi(g)\pi(g)^{-1}$$

we conclude that

$$[\pi(g)] = [\pi(g)^{-1}] \subseteq [g^{-1}\pi(g)] = [\pi(g)^{-1}g] = [g].$$

where the first and second equalities hold because taking the inverse does not affect the value of $[\_]$. From (6.2) 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 48). Therefore the group is finite.

The same proof would work for some other atoms, e.g. $(\mathbb{Q}, <)$. I do not know if it works for all oligomorphic atoms.

Solution to Exercise 84.
Using the same ideas as in Theorem 6.3, we get the following result.

Claim 6.4. *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 6.4, 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

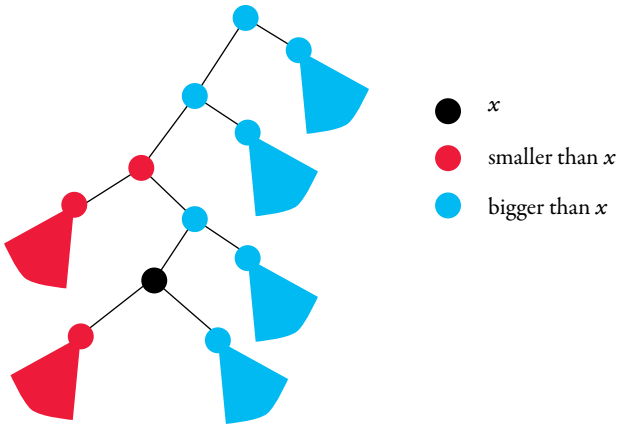
$$f' \stackrel{\text{def}}{=} \{\pi(\bar{a}, \bar{b}) : \pi \text{ is a } \bar{c}\text{-automorphism}\}$$

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.

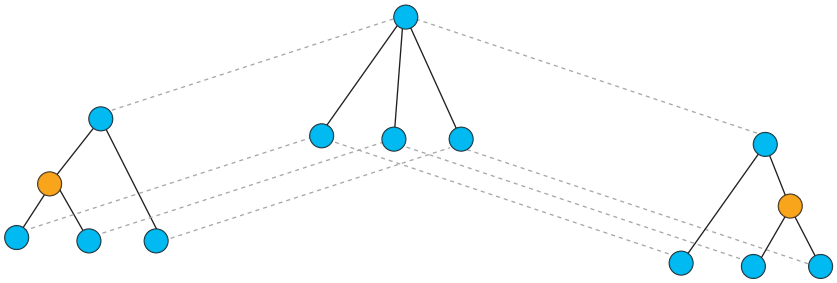
Solution to Exercise 85.
In a homogeneous structure, two tuples are in the same orbit if they satisfy the same quantifier-free formulas. By the assumption that the vocabulary is relational (i.e. has no function symbols) and finite, up to logical equivalence there are finitely many quantifier-free formulas 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 78, 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 78 and is thus an effective structure.

Solution to Exercise 86.
A rational number can be viewed as node in Rabin's tree $\{0, 1\}^*$ as follows

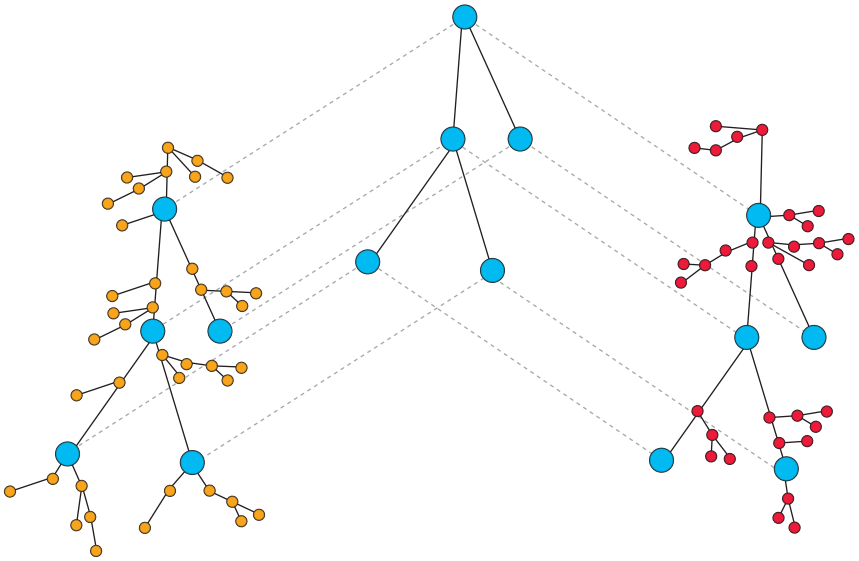


Solution to Exercise 87.
Here is the instance of amalgamation which has no solution:

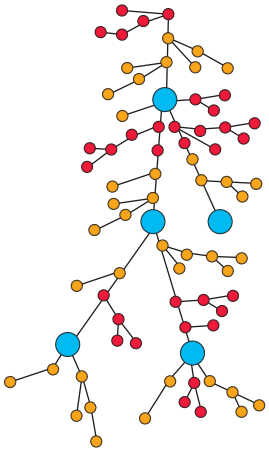


Solution to Exercise 88.
Proof by picture:

an instance of amalgamation



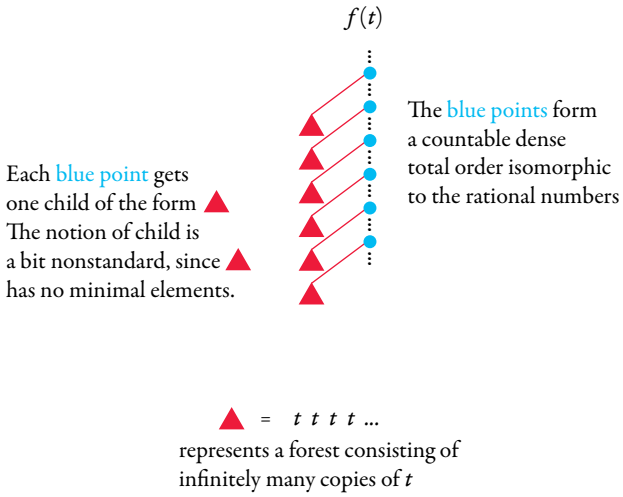
its solution



Solution to Exercise 89.
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 90.

Solution to Exercise 91.
We will show how this random 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 random 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 86 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 92.
See (Bojańczyk et al., 2013b, Proposition 2).

Solution to Exercise 93.
By going through the proof of Theorem 7.6.

Solution to Exercise 94.
No. The vertices might come in the wrong order.

Solution to Exercise 95.
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 96.
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 homogenity, it extends to a $\bar{a}$-automorphism, which preserves the total order, but swaps b with c

Solution to Exercise 97.
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 85 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 4.9. 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 98.
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

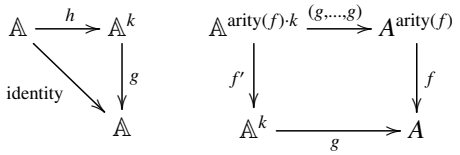
$$t_{n,k,i}(a_1, \ldots, a_k)$$

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 99.
The right-to-left implication is shown the same way as in Exercise 98. For the left-to-right implication, we need a stronger version of Exercise 98, 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 84 (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

$$g(t(h(a_1), \dots, h(a_n))) \in F.$$

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 100.
One could use Exercises 98 and 99, 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 10.1. *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 10.2. *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 10.1 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 10.2. 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 10.2 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 101.
In this case, conditions (**) from Exercise 99 and (***) from Exercise 100 are the same.

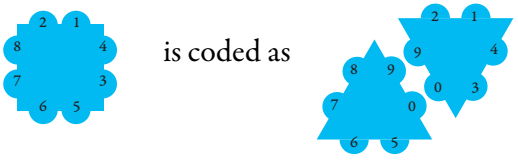
Solution to Exercise 102.
By the proof of Theorem 10.4, item 1 from the exercise is equivalent to the following property:

(*) 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 100 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 100 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 103.



Solution to Exercise 104.
By looking at the finitely many possible equivariant definable relations, and then doing a deatomisation procedure in each case.

Solution to Exercise 105.
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, 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*.

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.

Bojanczyk, Mikolaj. 2011. Data Monoids. Pages 105–116 of: *STACS*.

Bojańczyk, Mikołaj. 2013. Nominal Monoids. *Theory Comput. Syst.*, **53**(2), 194–222.

Bojańczyk, Mikołaj, and Lasota, Slawomir. 2012a. An extension of data automata that captures XPath. *Logical Methods in Computer Science*, **8**(1).

Bojańczyk, Mikołaj, and Lasota, Slawomir. 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, 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.

Bojanczyk, Mikolaj, Klin, Bartek, and Lasota, Slawomir. 2011. Automata with Group Actions. Pages 355–364 of: *LICS*.

Bojańczyk, Mikołaj, David, Claire, Muscholl, Anca, Schwentick, Thomas, and Segoufin, Luc. 2011. Two-variable logic on data words. *ACM Trans. Comput. Log.*, **12**(4), 27:1–27:26.

Bojanczyk, Mikolaj, Braud, Laurent, Klin, Bartek, and Lasota, Slawomir. 2012. Towards nominal computation. Pages 401–412 of: *POPL*.

Bojańczyk, Mikołaj, Klin, Bartek, Lasota, Slawomir, 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, Slawomir. 2014. Automata theory in nominal sets. *Logical Methods in Computer Science*, **10**(3).

Cheng, Edward Y. C., and Kaminski, Michael. 1998. Context-Free Languages over Infinite Alphabets. *Acta Inf.*, **35**(3), 245–267.

Cherlin, G., and Lachlan, A.H. 1985. Stable finitely homogeneous structures. *Trans. AMS*, **296**(2), 815–850.

Clemente, Lorenzo, and Lasota, Slawomir. 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, Slawomir. 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, Ley, Clemens, and Puppis, Gabriele. 2015. Logics with rigidly guarded data tests. *Logical Methods in Computer Science*, **11**(3).

Demri, Stéphane, and Lazic, 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, Lazic, 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*.

Hodges, W. 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).

Jurdzinski, Marcin, and Lazic, 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, Lasota, Slawomir, 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, Lasota, Slawomir, Ochremiak, Joanna, and Torunczyk, Szymon. 2016. Homomorphism problems for first-order definable structures. In: *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 65. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Kopczynski, 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.*

Kopczynski, 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*.

Leroux, Jérôme. 2010. The General Vector Addition System Reachability Problem by Presburger Inductive Invariants. *Logical Methods in Computer Science*, **6**(3).

Mac Lane, S., and Moerdijk, I. 1992. *Sheaves in geometry and logic: a first introduction to topos theory*. Springer.

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 pi-Calculus. Pages 255–269 of: *TACAS*.

Montanari, Ugo, and Pistore, Marco. 2005. History-Dependent Automata: An Introduction. Pages 1–28 of: *SFM*.

Murawski, Andrzej S., Ramsay, Steven J., 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 S., Ramsay, Steven J., 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, A. M. 2013. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science, vol. 57. Cambridge University Press.

Presburger, Mojzesz. 1929. Uber die Vollstandigkeiteines gewissen Systems der Arithmetik ganzer Zahlen, in welchen die Addition als einzige Operation hervortritt. In: *Comptes-Rendus du ler Congres des Mathematiciens des Pays Slavs*.

Ryll-Nardzewski, Cz. 1959. On the categoricity in power 0. *Bull. Acad. Polon. Sci. Sér. Sci. Math. Astr. Phys*, **7**, 545–548.

Schmerl, James. 1978. A decidable $_0$-categorical theory with a non-recursive Ryll-Nardzewski function. *Fundamenta Mathematicae*, **98**(2), 121–125.

Schmitz, Sylvain, and Schnoebelen, Philippe. 2012 (Aug.). *Algorithmic Aspects of WQO Theory*. Lecture.

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)*.

Wysocki, Tomasz. 2013. *Automaty alternujace z rejestrami na skończonych słowach*. M.Phil. thesis, University of Warsaw.

yi Cai, Jin, Fürer, Martin, and Immerman, Neil. 1992. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, **12**(4), 389–410.

Author index

Subject index