Computation in sets with atoms

November 30, 2013

Contents

1 Introduction **4**

I Introduction to Sets with Atoms **5**

2 Definition of sets with atoms **6**
2.1 Legal sets with atoms 6

3 Orbit-finite sets with atoms **14**
3.1 Definition of orbit-finite sets 14

4 Oligomorphic atoms **18**
4.1 Consequences of oligomorphism 18
4.2 Finitely supported is the same as first-order definable 22

5 Hereditarily definable sets **23**
5.1 Hereditarily orbit-finite sets and algorithms on them 25

6 Homogeneous atoms **31**
6.1 Finitely supported is the same as quantifier-free definable 33
6.2 The Fraïssé limit 34

7 What kind of definition is orbit-finiteness? **40**

II The Chomsky Hierarchy **43**

8 Finite automata with atoms **44**
8.1 Finite automata 44
8.2 Register automata 49
8.3 Other models of finite automata 51

9 Context-free languages with atoms **52**
9.1 Pushdown automata. 52
9.2 Context-free grammars. 56

10 Turing machines with atoms **57**

III Programming Languages 60

11 Imperative programming with atoms **61**
11.1 Definition of the imperative language 61
11.2 Executing the programs without using atoms 64
11.3 Example programs 65

12 Functional programming with atoms **70**
12.1 λ-calculus with orbit-finite sets 70
12.2 Example programs 74

IV The Church-Turing Thesis 75

13 Computability modulo representations **75**
13.1 Turing machines 76
13.2 Imperative and functional programs 79

14 Least supports (planned) **81**

15 Deterministic Turing machines **81**
15.0.1 The language 82
15.0.2 Algebras as a model of local computation 83
15.0.3 Algebras do not recognise L 85

16 For bit vector atoms, P$\neq$NP **88**

V Logics 92

17 The compactness theorem (planned) **92**

18 Eliminating orbit-finite disjunction (planned) **92**

VI Automata, continued 93

19 Other models of finite automata (planned) **93**
19.1 Alternating automata (planned) 93
19.2 Two-way automata (planned) 93

20 Learning of finite automata **94**

21 Database-driven systems (planned) **98**

22 Monoids (planned) **98**

VII Solutions to the exercises 99

1 Introduction

Part I
Introduction to Sets with Atoms

In this part, we define sets with atoms. The main focus is on the *orbit-finite sets with atoms*, which play the role of finite sets. We argue that certain infinite sets that appear in computer science are orbit-finite sets with atoms.

Our definition of sets with atoms is parametrized by a relational structure for the atoms. We show that certain properties of the atom structure, as studied in model theory, guarantee that the orbit-finite sets are well behaved. In particular, sets with atoms are well behaved when the atoms are an oligomorphic structure, and even better behaved when the atoms are homogeneous structure over a finite vocabulary.

2 Definition of sets with atoms

Normal sets, such as $\emptyset$ or $\{\emptyset, \{\emptyset\}\}$, are built out of the empty set and brackets, although the structure of brackets might be complicated, for instance in the real numbers. In sets with atoms, one postulates the existence of an infinite set of atoms, and sets can contain those atoms as well. The atoms are modelled as a logical structure: a universe together with some relations and function. Examples of 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
$(\mathbb{Z}, +1)$	the integers with the unary successor function

The three kinds of atoms listed above will be called, respectively, the *equality atoms*, the *total order atoms*[1], and the *integer atoms*. The choice of atoms is a parameter of the notion of sets with atoms[2].

2.1 Legal sets with atoms

Not every set built out of atoms and set brackets is legal. The intuitive idea behind a legal set with atoms is that it is a set built out of atoms, whose definition is allowed to only use the structure given by the atoms (i.e. relations and functions from the vocabulary of the atoms), and finitely many constants that refer to specific atoms. We begin with some examples.

- For every choice of atoms, a set that does not use atoms, such as $\emptyset$, or $\{\emptyset, \{\emptyset\}\}$, or the set of real numbers, is a legal set with atoms. The definitions of such sets refer to finitely many atoms, because they do not refer to any atoms.
- Regardless of the choice of atoms, the set of all atoms is a legal set with atoms. The definition of this set does not refer to any atoms, or any structure of the atoms.
- Consider the equality atoms. We denote equality atoms by underlined natural numbers, such as $\underline{1}$ or $\underline{2}$, to underline that atoms do not share the structure of natural numbers, such as a successor function. Any finite set of atoms, such as $\{\underline{1}, \underline{2}, \underline{3}\}$, is a legal set with atoms. The definition of a finite set refers only to the atoms in the set. Likewise for cofinite sets, such as all atoms except for $\{\underline{2}, \underline{4}\}$.

[1] There are many totally ordered sets, but the rational numbers are special. They are the unique totally ordered set that is homogeneous. Being homogeneous turns out to be important for sets with atoms, see Section 6.

[2] We distinguish between a set with atoms, and a set of atoms. A set of atoms is a subset of the universe of the atom structure. A set with atoms is a more general concept, which covers also sets of sets of atoms, and other more complicated sets.

- Consider the equality atoms. The set of even numbered atoms $\{\underline{0}, \underline{2}, \underline{4}, \ldots\}$ is *not* a legal set with atoms. A definition of this set would need to either explicitly mention infinitely many atoms, or refer to the notion of "even-numbered" atoms, which does not exist, since we assume that atoms have no structure. However, the set of even-numbered atoms is legal in the integers atoms, because it can be defined as "anything obtained from $\underline{0}$ by applying the successor function an even number of times".

Below we define legal sets with atoms more formally.

Definition of legal sets with atoms. Fix a structure for the atoms. Consider first the *cumulative hierarchy* of sets with atoms, which is a hierarchy of sets indexed by ordinal numbers and defined as follows. The empty set is the unique set of rank 0. A set of rank α is any set whose elements are sets of rank smaller than α, or atoms. The cumulative hierarchy contains too many sets, including illegal sets such as the set of "even-numbered" atoms mentioned above.

The legal sets with atoms are a obtained by restricting the cumulative hierarchy to sets which satisfy the finite support condition, which is defined in terms of automorphisms. An automorphism of the atoms is any bijection of the atoms which preserves the relations and functions that are part of the atoms structure. If an automorphism furthermore preserves all atoms $a_1, \ldots, a_n$, then it is called a $(a_1, \ldots, a_n)$-automorphism. An automorphism can be applied to a set in the cumulative hierarchy, by renaming its elements (if the set happens to contain atoms), elements of its elements, and so on recursively. The resulting of applying an automoprhism π to a set X is denoted by $\pi(X)$, and it is a set with the same rank. If the definition of a set X uses only the vocabulary and atoms $a_1, \ldots, a_n$, then under any reasonable notion of "definition", the set X should satisfy $\pi(X) = X$ for every $(a_1, \ldots, a_n)$-automorphism of atoms. This motivates the following definition

Definition 2.1 (support) *A tuple $\bar{a}$ of atoms is called a* support *of a set X in the cumulative hierarchy if $\pi(X) = X$ holds for every $\bar{a}$-automorphism π. A set is called* finitely supported *if it has some finite support.*

Note that the order or repetition of atoms in the tuple is not relevant for the support, i.e. only the set of atoms that appear in the tuple matters[3]. The support of a set with atoms is not unique, e.g. supports are closed under adding atoms. (In Section 14, we will show that in some cases, a canonical least support can be found, though.) A set with empty support is called *equivariant*.

Example 1. [Supports] Consider the equality atoms, and the set X of all atoms except $\underline{2}$. This set is supported by the atom $\underline{2}$, because any any $\underline{2}$-automorphism

[3]For this reason, many authors use a set of atoms as a support, instead of a tuple of atoms. We use tuples so that we can be convenient 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.

will preserve X as a set, but it might rearrange its elements. The set X is not equivariant, so $\underline{2}$ is a minimal support, actually it is a least support. □

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. In particular, an equivariant set is one which can be defined without referring to any specific atoms. Since a set can have many definitions, there might not be a canonical support. For instance, when the atoms are the integers with the successor function, then the set $\{2\}$ is supported by 2, but it is also supported by 1 because it can be defined by "the singleton of the successor of 1".

Definition 2.2 (Legal set with atoms) *A* legal set with atoms *is a set in the cumulative hierarchy which is hereditarily finitely supported, i.e. it is finitely supported, its elements are finitely supported, and so on.*

In this section, Section 2.1, we will pay attention to the distinction between legal and illegal sets with atoms. In subsequent sections, we implicitly assume that all sets with atoms are legal, unless stated otherwise.

Unions, intersections, etc. In many respects, legal sets with atoms behave like normal sets. For instance, if X, Y are legal sets with atoms, then $X \times Y$, $X \cup Y$, X^* and the finite powerset of X are all legal sets with atoms. When talking about pairs, we use any set-theoretic pairing function, e.g. the Kuratowski pair:

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

Subsets. A subset of a legal set with atoms might no longer be legal.

Example 2. [Legal sets of equality atoms] Consider the equality atoms. The set of all atoms is legal, as we have remarked above. Which subsets of the set of all atoms are legal? Equivalently, which sets of atoms are finitely supported? We claim that the finitely supported sets of atoms are exactly the finite and co-finite sets. It is not difficult to see that the finite co-finite sets are finitely supported. For the converse implication, consider a set X of atoms that is neither finite, nor co-finite. We will show that X cannot have finite support. Suppose then that $a_1, \ldots, a_n$ is a candidate for a finite support. Since neither X nor its complement are infinite, they cannot be included in the finite set $\{a_1, \ldots, a_n\}$, and therefore there must be atoms $a \in X$ and $b \notin X$ that are not in $a_1, \ldots, a_n$. Let π be the permutation of atoms which swaps a and b, and is the identity on other atoms. This permutation is an $(a_1, \ldots, a_n)$-automorphism, so it should fix X, but it does not. □

Example 3. [Legal sets of total order atoms] Consider the total order atoms. 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 $b \in X$. □

Observe that in both examples above, the finitely supported sets of atoms coincide with sets of atoms definable by quantifier-free formulas which can use constants from the atoms. The reason is that both examples of atoms are *homogeneous* structures, which are discussed in Section 6. In general, when the atoms are not homogeneous, finitely supported sets are not the same thing as quantifier-free definable sets, as shown in the following example.

Example 4. [Legal sets of integer atoms] Consider the integer atoms. What are the legal subsets of atoms? All subsets of the atoms are legal. More generally, any set built out of atoms is legal. This is because under the integer atoms, the finite support condition is trivially true. Every set is finitely supported, namely supported by any nonempty tuple, such as 3 or $(-4, 6)$. This is because if an automorphism of the integers is necessarily a translation, and if a translation has at least one fix-point, then it is the identity. On the face of it, this sounds like good news, e.g. legal sets under the integer atoms are closed under arbitrary subsets, unlike legal sets with atoms under most other atoms. However, there is a price to pay, as we shall see in later sections. □

In legal sets with atoms there is a different notion of powerset, the *finitely supported powerset*, which contains only the finitely supported subsets of a given set. As shown in Examples 2 and 3, the finitely supported powerset can be smaller than the standard powerset.

Relations. Since legal sets with atoms are closed under (finite) products, we can talk about relations (of finite arity). For instance, a binary legal relation on sets X and Y is any finitely supported subset of $X \times Y$.

Example 5. [Binary equivariant relations on equality atoms] Consider the equality atoms. 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:

$$\emptyset \qquad \text{Atoms} \times \text{Atoms} \qquad \{(a, a) : a \in \text{Atoms}\} \qquad \{(a, b) : a \neq b \in \text{Atoms}\}.$$

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. □

Example 6. [Binary equivariant relations on total order atoms] Consider the total order atoms. What are the equivariant binary relations? All of the four equivariant relations mentioned in Example 5 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:

$$\{(a,b) : a < b\} \qquad \{(a,b) : a \leq b\} \qquad \{(a,b) : a > b\} \qquad \{(a,b) : a \geq b\}.$$

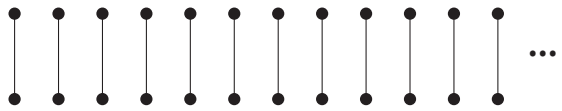
Observe again these are exactly the binary relations that can be defined by quantifier-free formulas. □

Functions. A function is the special case of a binary relation which contains one pair for each value of the first coordinate. In other words, a legal function with atoms is any function whose graph is a legal set with atoms.

Example 7. [Equivariant functions] In the equality atoms, there is only one equivariant function from atoms to atoms, namely the identity, which follows from Example 5. Also in the total order atoms, the identity is the only equivariant function from atoms to atoms.

In the equality atoms, there are only two equivariant functions from pairs of atoms to atoms, namely the projections. In the total order atoms, there is an additional function, namely max. □

Example 8. Intuitively speaking, an equivariant function should only output atoms that it is given on the input. Although essentially correct, this can be false when there are dependencies between atoms. Suppose that the atoms are a graph with infinitely many edges that do not share any nodes.



The function which flips the structure by mapping one side of an edge to the other side of an edge is equivariant. □

Another way of looking at legal functions with atoms is that the functions should commute with automorphisms of atoms, as stated in the following fact.

Fact 2.3 *A function $f : X \to Y$ is supported by a tuple of atoms $\bar{a}$ if and only if*

$$f(\pi(x)) = \pi(f(x)) \qquad \text{for every } x \in X \text{ and every } \bar{a}\text{-automorphism } \pi. \quad (1)$$

Proof
By unraveling the definition, the equality in condition (1) is equivalent to

$$(\pi(x), \pi(f(x))) \in f,$$

which is equivalent to

$$(x, f(x)) \in \pi^{-1}(f),$$

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 $\bar{a}$. □

For those who like commuting diagrams, condition (1) can be expressed by

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ \downarrow\pi & & \downarrow\pi \\ X & \xrightarrow{f} & Y \end{array} \qquad \text{for every } \bar{a}\text{-automorphism } \pi. \tag{2}$$

In particular, an equivariant function is a function which commutes with every automorphism of atoms.

Historical background. The historical origins of legal sets with atoms are in set theory. In 1922, Abraham Fraenkel showed that, when the atoms have equality only, the legal sets with atoms:

- fail the axiom of choice, as shown in Example 9 below, but
- satisfy a system of axioms which is similar to ZF[4].

As mentioned above, the axioms satisfied by legal sets with atoms are not the real ZF 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 ZF axioms had to wait for Cohen.

Example 9. [Axiom of choice fails] This example shows that the axiom of choice fails in legal sets with atoms, under the equality atoms. Consider the family of two-element subsets of Atoms, call it $\mathsf{P}_2(\text{Atoms})$. The axiom of choice would say that there is a function

$$f : \mathsf{P}_2(\text{Atoms}) \to \text{Atoms}$$

which maps every two-element set to one of its elements. There is no such legal function. (Under atoms with total order, there is such a function, namely max.) Toward a contradiction, suppose that f is such a function, and has finite support. Choose a two element set $\{a, b\}$ of atoms that do not appear in the support. Without loss of generality, assume that $f(\{a, b\}) = a$. Let π be the transposition which swaps a with b, and is the identity on all other atoms, in particular on the support of f. We get a contradiction:

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

□

[4]One difference between ZF and the axioms satisfied by sets with atoms is that the latter does not have full extensionality (objects with the same elements are equal), because an atom has the same elements as the empty set.

Exercise 1. Consider the equality atoms. Show a finitely supported graph, which admits an illegal two-coloring, but does not admit any legal two-coloring.

Example 10. [No finitely supported total order] In normal sets, one can impose a total ordering, even a well-ordering, on every set. This is a consequence of the axiom of choice, but it fails in legal sets with equality atoms. (In sets with total order atoms, a legal total order on atoms can be imposed, namely the underlying order, however a well-ordering cannot be imposed.) We prove a stronger statement: for every finitely supported *partial* order $<$ on the equality atoms, all atoms outside the support are incomparable. Indeed, 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. □

Sets with atoms were further developped by Andrzej Mostowski, which is why they are sometimes called Fraenkel-Mostowski sets. The following example, which uses atoms with order, comes from Mostowski.

Example 11. [No finitely supported well-founded total order] Consider the total order atoms, i.e. the rational numbers with order. The set of atoms clearly has a linear order, namely the built-in one. However, there is no finitely supported well-founded total order. Indeed, 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.

□

There is an even more general definition of sets with atoms, which is due to Specker, and not used in this text. In the more general definition, the parameter is a set of atoms whose structure is given not by relations and predicates as in this text, but by a filter of subgroups of the permutation group of the atoms. Instead of requiring a set to have finite support, we require that its stabiliser (a subgroup of the permutations of the atoms) belongs to the filter.

For more on how sets with atoms are used for independence proofs in set theory, see the book [4]. A more recent source of interest in sets with atoms comes from the semantics community. Sets with atoms, known as *nominal sets*, turn out to be a good foundation to the study of name binding in syntax, e.g. in the syntax of the λ-calculus. This application is illustrated in the following example. For more on nominal sets, see the book [5].

Example 12. [Nominal sets and λ-terms] Consider the set of λ-terms. For the moment, we distinguish terms even if they differ only in the bound variables,

so $\lambda x.x$ and $\lambda y.y$ are different terms. The set of λ-terms is the least solution to the equation

$$X \quad = \quad \text{Variables} \quad \uplus \quad X \times X \quad \uplus \quad \text{Variables} \times X, \tag{3}$$

where the middle component in the disjoint union stands for application, and the last component stands for λ-abstraction. Formally speaking, the equation above involves three operators which map sets to sets: namely the constant Variables, as well as $X \mapsto X \times X$ and $X \mapsto \text{Variables} \times X$.

What about λ-terms modulo renaming bound variables? Can this set be seen as a least solution to some equation which involves set to set operators? It turns out that to write such an equation, it is convenient to use sets with atoms (under the equality atoms).

The general ideas is to use atoms as the variables, with supports playing the role of free variables. For instance, the identity term can be seen as the equivalence class

$$\{\lambda a.a : a \text{ is an atom}\}.$$

The identity term has empty support, which corresponds to the fact that it has no free variables. The constant function which maps every input to the variable $\underline{1}$ is the equivalence class

$$\{\lambda a.1 : a \text{ is an atom different than } \underline{1}\}.$$

When seen this way, the set of λ-terms modulo renaming bound variables is the least solution to a equation similar to (3), but where instead of the operation $X \mapsto \text{Variables} \times X$, a different operation called *name abstraction* is used. This operation is defined below.

We say that an atom is *fresh* in an object if the object has some support that does not contain this atom. For instance the atom $\underline{5}$ is fresh for the pair $(\underline{1}, \underline{7})$, also $\underline{5}$ is fresh for the set of all atoms. The notion of freshness is best studied for the equality atoms, which is the case in this discussion of λ-terms. For a set with atoms X, define an equivalence relation $\sim_X$ on $\text{Atoms} \times X$ which considers two pairs (a, x) and (b, y) to be equivalent if there is some atom c that is fresh in both of the pairs such that $\pi(x) = \sigma(y)$ holds, assuming that π is the automorphism which swaps a with c, and σ is the automorphism which swaps b with c. The intuition is that y is obtained from x by replacing a with b while not changing other atoms in the definition of x. For instance, when X is Atoms^2, then

$$(\underline{5}, (\underline{1}, \underline{5})) \quad \sim_X \quad (\underline{6}, (\underline{1}, \underline{6})).$$

Another example: if X is the cofinite sets of atoms, then

$$(\underline{5}, \text{Atoms} - \{\underline{1}, \underline{3}, \underline{5}\}) \quad \sim_X \quad (\underline{6}, \text{Atoms} - \{\underline{1}, \underline{3}, \underline{6}\}).$$

One can show that $\sim_X$ is an equivalence relation which is supported by whatever supports X, and therefore the quotient

$$\text{Atoms} \times X_{/\sim_X}$$

is a legal set with atoms, called the *name abstraction of* X. For instance, the name abstraction of the atoms contains the equivalence class

$$\{(a, a) : a \text{ is an atom}\}$$

and, for every atom a, the equivalence class

$$\{(a, b) : b \text{ is an atom different than } a\}.$$

It is not difficult to show that if in equation (3), one replaces Variables $\times X$ by the name abstraction of X, the least solution will become the set of λ-terms modulo renaming bound variables. □

3 Orbit-finite sets with atoms

In the previous section, we discussed two applications of sets with atoms: independence proofs in set theory (the original motivation behind sets with atoms), and a study of name binding (nominal sets). In this section we describe a different reason for considering sets with atoms, which is the motivation behind this text: in sets with atoms there is a different, more relaxed, notion of finiteness.

3.1 Definition of orbit-finite sets

From now on, we assume that all sets with atoms are legal, unless otherwise stated. Suppose that x is an atom or a set with atoms, and $\bar{a}$ is a tuple of atoms. Define the *$\bar{a}$-orbit of x* to be the set

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

The $\bar{a}$-orbit is supported by $\bar{a}$, and therefore is a legal set with atoms. A set with atoms is supported by $\bar{a}$ if and only if it is a union of $\bar{a}$-orbits. The union can be infinite. The general idea behind orbit-finite sets is that they are finite unions of orbits. This idea can be formalised in several different definitions.

The first definition is called a *multi-support orbit-finite set*, which is a union of finitely many orbits sets, but the orbits can use different supports, i.e. a set of the form:

$$\bigcup_{i \in I} \{\pi(x_i) : \pi \text{ is an } \bar{a}_i\text{-automorphism}\}, \tag{4}$$

for some finite indexing set I, some legal sets $\{x_i\}_{i \in I}$ called the *orbit representatives*, and some possibly different supporting tuples $\{\bar{a}_i\}_{i \in I}$. A more restrictive definition requires all of the supporting tuples $\bar{a}_i$ to be the same tuple $\bar{a}$, in

which case we call the set *single-support orbit-finite*. In other words, a set is single-support orbit-finite if it decomposes into finitely many $\bar{a}$-orbits for some tuple $\bar{a}$ which supports it. Finally, in a most restrictive definition, a set is called *every-support orbit-finite* if it decomposes into finitely many $\bar{a}$-orbits for every tuple $\bar{a}$ which supports it.

As we will show in Theorem 4.1, all three definitions are equivalent for a wide variety of atom structures, and we can simply call a set *orbit-finite*. In those cases where the definitions are not equivalent, we will need to specify which one we use.

Observe that for an equivariant set, there is no distinction between single-support and multi-support orbit-finiteness. Indeed, an equivariant set is partitioned uniquely into orbits under the action of all automorphisms; if there a finitely many such orbits then it is single-support orbit-finite, otherwise it is not even multi-support orbit-finite.

Most of this text is devoted to studying orbit-finite sets, under various kinds of atoms.

Example 13. [The set of all atoms is single-support orbit-finite] In all of the atom structures that we have discussed, the atoms are single-support orbit-finite. This is because in the structures involved (namely, the empty structure, the countably infinite structure with empty vocabulary, the rational numbers with order, the integers with successor), every element can be mapped to every other element by an automorphism. □

Example 14. [Total order atoms are all-support orbit-finite] Consider the total order atoms. As we have already remarked, the set of all atoms is a single-support single-orbit set. We show that the set of all atoms is even all-support orbit-finite. Since the set of all atoms has empty support, showing that it is all-support orbit-finite would require showing that for every tuple $\bar{a}$, the atoms split into finitely many $\bar{a}$-orbits. This is indeed true, because when the support $\bar{a}$ contains atoms $a_1 < \cdots < a_n$, then the $\bar{a}$-orbits are the singletons $\{a_1\}, \ldots, \{a_n\}$ and the intervals

$$(-\infty; a_1) \quad (a_1; a_2) \quad \cdots \quad (a_{n-1}, a_n) \quad (a_n; +\infty).$$

□

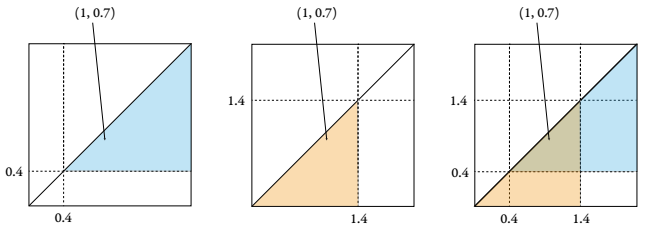
Example 15. [In the integer atoms, all notions of orbit-finiteness are distinct] Consider the integer atoms. The set

$$\mathbb{Z} \cup \{(1, 1)\}$$

is multi-support orbit-finite, because it is the union of two single-support orbit finite sets, namely $\mathbb{Z}$ and $\{(1, 1)\}$. However, this set is not single-support orbit-finite. If the single support were to be nonempty, then $\mathbb{Z}$ falls apart into infinitely many orbits. If the single support were to be empty, then the set would also need contain the pair $(2, 2)$.

The set $\mathbb{Z}$ of all atoms is an example of a set that is single-support orbit finite, but not strongly orbit-finite. This is because every nonempty support causes $\mathbb{Z}$ to fall apart into infinitely many orbits. □

Example 16. [Different decompositions of multi-support orbit-finite sets] Consider the total order atoms and the 0.4-orbit of the ordered pair $(1, 0.7)$ plus the 1.4-orbit of the same ordered pair. This set union is illustrated below:



The set can be decomposed into non-intersecting orbits, e.g. under the action of $(0.4, 1.4)$-automorphisms, but then 12 orbits are needed. □

We now study which set operators preserve orbit-finiteness. It is not difficult to see that union preserves multi-support orbit-finiteness, and also all-support orbit-finiteness. As shown in Example 15, in the integer atoms single-support orbit-finiteness is not preserved under union, when the arguments of the union use different supports. Closure under powerset, product and subset is also problematic. In Section 6, we will impose additional conditions on the atoms which ensure that orbit-finiteness is preserved under products and subsets. Powerset is a lost cause.

Powerset. Recall Example 2, which says that under the equality atoms, not all subsets of a legal set with atoms are legal, and therefore the appropriate notion of powerset for legal sets with atoms is the finitely supported powerset. The following example shows, however, that orbit-finite sets are not even closed under the finitely supported powerset. The example only considers the equality atoms, but the situation is the same in any atoms where orbit-finite sets are not the same thing as finite sets.

Example 17. [Orbit-finiteness is not preserved by powerset] Consider the equality atoms, which are orbit-finite in the strongest all-orbit sense. The finitely supported powerset of the atoms contains all the finite sets of atoms. Sets of different sizes cannot be in the same orbit, since an automorphism preserves the size of a set. It follows that the finitely supported powerset of the atoms is not orbit-finite even in the weakest, multi-support, sense. □

Some constructions in computer science, such as determinisation of finite automata, depend on the powerset preserving finiteness. These constructions will fail with atoms; in particular, as we shall see in Section 8, nondeterministic and deterministic automata are not equivalent in the presence of atoms.

Product. In the integer atoms, the product of two single-support orbit-finite sets might even not be multi-support orbit-finite.

Example 18. [Single-support orbit-finite sets not closed under products, for integers] Consider the integer atoms. The set of atoms, namely the integers $\mathbb{Z}$, is single-support orbit-finite. Already the square of this set, namely $\mathbb{Z} \times \mathbb{Z}$, is not single-support orbit-finite, or even multi-support orbit-finite. The orbits of this set, under the action of all integer automorphisms, are the diagonals, i.e. sets of the form

$$\{(x, x + y) : x \in \mathbb{Z}\} \qquad \text{for } y \in \mathbb{Z}.$$

There are infinitely many such diagonals. □

Example 19. [Number of orbits in X^2 can be arbitrarily big, even when X has one orbit] Even when the product has finitely many orbits, the number of orbits can grow a lot. Consider the equality atoms. Let $n \in \mathbb{N}$. We denote the set of non-repeating n-tuples of atoms by $\text{Atoms}^{(n)}$. 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. The square of this set,

$$\text{Atoms}^{(n)} \times \text{Atoms}^{(n)}$$

has a number of equivariant orbits that is exponential in n. More precisely, the set has one orbit for every partial bijection between $\{1, \ldots, n\}$ and $\{1, \ldots, n\}$, because there are different possible ways in which the coordinates of the first tuple might be equal to the coordinates of the second tuple. In particular, the number of orbits of X^2 can be arbitrarily big, even when X has one orbit, and the number of orbits can depend on an additional parameter like the dimension n. □

Subset. In some pathological cases, single-support orbit-finite have finitely supported subsets that are not even multi-support orbit-finite.

Example 20. Consider the integer atoms. The set of atoms, namely the integers, is single-support orbit-finite. Consider an arbitrary infinite subset of integers, e.g. the even integers $2\mathbb{Z}$. This set is legal, because, as we remarked in Example 4, every subset of the integers is finitely supported, e.g. supported by 0. We claim every proper infinite subset of the integers is not even multi-support orbit-finite. This is because a multi-support subset of the integers contains either finitely many integers or is equivariant, in which case it is not a proper subset. □

4 Oligomorphic atoms

A logical structure is called oligomorphic if for every n, the set of n-tuples of atoms has finitely many equivariant orbits. The notion of oligomorphism structures is important in model model theory. For instance, a theorem of Ryll-Nardzewski says that a structure is oligomorphic if and only if it is a model of an ω-categorical theory.

Example 21. [The equality and order atoms are oligomorphic.] In the equality atoms, the equivariant orbit of a tuple of atoms $(a_1, \ldots, a_n)$ is uniquely determined by its equality type, i.e. the information about which pairs of coordinates carry equal atoms. There are finitely many equality types for a given dimension n. For the total order atoms, one needs a bit more information, but still of finite size: which says which coordinates are bigger than which other coordinates. □

Example 22. The integers are not oligomorphic, because pairs of atoms have infinitely many orbits, as shown in Example 18. □

In this section, we show that when the atoms are oligomorphic, then sets with atoms are well behaved:

- all three notions of orbit-finiteness are equivalent (Theorem 4.1);
- orbit-finite sets are closed under finite products (Theorem 4.1);
- orbit-finite sets are closed under finitely supported subsets (Exercise 2);
- a set of n-tuples of atoms is finitely supported if and only if it is definable by a first-order logic formula with n free variables and constants from the atoms (Theorem 4.5);

Actually, not only are the properties above implied by oligomorphism, but also each of the four properties alone implies oligomorphism. In other words, oligomorphism of the atoms can be seen exactly the property required for good behaviour of orbit-finite sets.

Furthermore, assuming the first-order theory of the atoms is decidable, hereditarily orbit-finite sets can be represented in a finite way (without atoms) and natural operations (e.g. union or equality test) are decidable in terms of these representations (Theorem 5.4).

4.1 Consequences of oligomorphism

In this section, we show the following theorem.

Theorem 4.1 *Assume that the atoms are multi-support orbit-finite. The following conditions are equivalent.*

1. *For every n, the set of n-tuples of atoms is multi-support orbit-finite;*
2. *The atoms are an oligomorphic structure;*
3. *The set of atoms is all-support orbit-finite;*
4. *For every n, the set of n-tuples of atoms is all-support orbit-finite;*
5. *Single-support orbit-finite sets are closed under finite products.*

When the conditions above are satisfied, then the notions of single-support orbit-finite, multi-support orbit-finite, and strongly orbit-finite are all equivalent.

Thanks to the above theorem, when the atoms are oligomorphic, then we can simply call a set *orbit-finite*, without specifying which variant of orbit-finiteness is meant.

1 implies 2. As we have observed before, for an equivariant set, being multi-support orbit-finite is the same thing as consisting of finitely many equivariant orbits. The set of n-tuples of atoms is an equivariant set.

2 implies 3. To prove condition 3, we need to show that for every tuple of atoms $\bar{a}$, there is a finite set A of atoms such that every atom is in the same $\bar{a}$-orbit as some atom in A. Let n be the dimension of the tuple $\bar{a}$. By condition 2, the set

$$X = \{\bar{a}a : a \in \text{Atoms}\} \subseteq \text{Atoms}^{n+1}$$

interesects finitely many orbits of Atoms^{n+1} under the action of all automorphisms. In other words, there is a finite set $Y \subseteq X$ such that every element of X can be obtained from some element of Y by applying an atom automorphism. Let A be the projection of Y onto the last coordintate. By assumption on Y, for every atom b, there is some $a \in A$ such that $\bar{a}a$ and $\bar{a}b$ are related by an atom automorphism, call it π, which is necessarily a $\bar{a}$-automorphism.

3 implies 4. Condition 4 says that for every $n \in \mathbb{N}$ and every tuple of atoms $\bar{a}$, the set of n-tuples of atoms has finitely many $\bar{a}$-orbits. We show this by induction on n. The induction base of $n = 1$ is our assumption, namely condition 3. For the induction step, let A be a finite set of atoms representing each $\bar{a}$-orbit of atoms. Let $\bar{b}$ be a tuple which contains $\bar{a}$ and all atoms in A. By the induction assumption, there are finitely many $\bar{b}$-orbits of $(n-1)$-tuples of atoms, let X be a finite set of $(n-1)$-tuples representing these $\bar{b}$-orbits. We claim that every n-tuple of atoms is in the same $\bar{a}$-orbit as some tuple in the finite set $A \times X$. Let then $(a_1, \ldots, a_n)$ be an n-tuple of atoms. By assumption on A, there is some $\bar{a}$-automorphism π such that $\pi(a_1) \in A$. By assumption on X, there is some $\bar{b}$-automorphism σ such that $\sigma(\pi(a_2, \ldots, a_n)) \in X$. It follows that applying first π and then σ maps $(a_1, \ldots, a_n)$ to a tuple in $A \times X$. Since both π and σ are $\bar{a}$-automorphisms (actually, σ is even a $\bar{b}$-automorphism), we obtain the desired result.

4 implies 5 We want to show that the product of two multi-support orbit-finite sets is also multi-support orbit-finite.

Lemma 4.2 *A set is multi-support orbit-finite if and only if it can be presented as*

$$\bigcup_{i \in I} f_i(\{\pi(\bar{b}_i) : \pi \text{ is a } \bar{a}_i\text{-automorphism}\})$$

where I is finite, and for every i, $\bar{a}_i, \bar{b}_i$ are tuples of atoms, and f_i is an equivariant function. Likewise for single-support orbit-finite sets, only all of the tuples $\bar{a}_i$ must be equal.

Proof
Consider first the right-to-left implication. For every finite support $\bar{a}$, the image of a $\bar{a}$-orbit of x under an equivariant function f is the same as the $\bar{a}$-orbit of $f(x)$. This gives the right-to-left implication, and also shows that multi-support (respectively, single-support) orbit-finite sets are closed under images of equivariant functions. Note that equivariance is important, for instance in the integers, any countable set can obtained as an image of $\mathbb{Z}$ under a finitely supported function, because every function has finite support.

For the left-to-right implication, consider a multi-support orbit-finite set, which by definition is of the form

$$\bigcup_{i \in I} \{\pi(x_i) : \pi \text{ is a } \bar{a}_i\text{-automorphism}\}. \tag{5}$$

For each x_i, choose a finite support, call it $\bar{b}_i$. Consider the relation

$$\{(\pi(\bar{b}_i), \pi(x_i)) : \pi \text{ is an atom automorphism}\}.$$

By the assumption that $\bar{b}_i$ supports x_i, this relation is actually a function, call it f_i, which is furthermore equivariant. By equivariance, the set in the statement of the lemma is equal to (5). The same proof gives the single-support case of the lemma. □

By the above lemma, it suffices to show that a product of two sets as in the statement of the lemma is multi-support orbit-finite. Because multi-support orbit-finite sets are closed under finite products and images of equivariant functions, it suffices to show that for every tuples of atoms $\bar{a}, \bar{b}, \bar{c}, \bar{d}$, the product

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

is multi-support orbit-finite. But the above is a set of atom tuples (whose dimension is the sum of dimensions of $\bar{b}$ and $\bar{d}$) which is supported by the atoms in $\bar{a}$ and $\bar{c}$. Therefore, by condition 4, it is single-support orbit-finite.

5 implies 1 Immediate.

Corollary 4.3 *When the atoms are oligomorphic, then for every tuple of atoms $\bar{a}$, every orbit-finite set X contains finitely many $\bar{a}$-supported elements.*

Proof
An element $x \in X$ is supported by $\bar{a}$ if and only if $\{x\}$ is an $\bar{a}$-orbit. By Theorem 4.1, there are finitely many $\bar{a}$-orbits in X. □

The converse of Corollary 4.3 is false, e.g. any infinite set without atoms, such as the natural numbers, contains infinitely many equivariant elements.

Equivalence of variants of orbit-finiteness. To finish the proof of Theorem 4.1, we show that if the conditions in the theorem are satisfied, then every multi-support orbit-finite (the weakest variant of orbit-finiteness) set is strongly orbit-finite (the strongest variant of orbit-finiteness). (The converse implication is also true: if all-support orbit-finiteness is the same as multi-support orbit-finiteness, then the assumption that the atoms are multi-support orbit-finite implies condition 3 of the theorem.) All-support orbit-finite sets are closed under finite unions and images of equivariant functions. The former is easy to see, for the latter we use that fact that for any support $\bar{a}$, sets which have finitely many $\bar{a}$-orbits are closed under images of equivariant functions. By Lemma 4.2, every every multi-support orbit-finite set is a finite union of equivariant images of finitely supported sets of tuples of fixed dimension, and by condition 4 of the theorem, finitely supported sets of tuples of fixed dimension are strongly orbit-finite.

Exercise 2. Show that the conditions in Theorem 4.1 are equivalent to "single-support orbit-finite sets are closed under finitely-supported subsets".

Example 23. [Examples of single-orbit sets in the equality atoms] Consider the equality atoms. For $n \in \mathbb{N}$, a permutation of $\{1, \ldots, n\}$ can be applied to an n-tuple of atoms, by permuting its coordinates. For a subgroup H of permuations of $\{1, \ldots, n\}$, consider two n-tuples of atoms H-equivalent if there is some permutation in H which maps one tuple to the other. Define

$$\text{Atoms}^{(n)}/H$$

to be the set of non-repeating n-tuples of atoms, modulo H-equivalence. For various choices of H we get various single-orbit equivariant sets. For instance, when H contains only the identity permutation, we get the non-repeating n-tuples, and when H contains all permutations we get the sets of size n. Interesting single-orbit sets are obtained for intermediate choices of H, e.g. when H is a cyclic group. One can show that every equivariant orbit is of the form discussed above, modulo an equivariant bijection. □

Exercise 3. Consider the total order atoms. Show that there are uncountably many single-orbit sets, which are different even up to finitely supported bijections.

Exercise 4. Do the previous exercise, but for equality atoms.

Exercise 5. Show that when the atoms are oligomorphic, orbit-finite sets are closed under images of finitely supported functions.

4.2 Finitely supported is the same as first-order definable

In this section, we show another benefit of oligomorphic atoms: finitely supported sets of tuples are exactly those that can be defined in first-order logic. This is essentially part of the Ryll-Nardzewski theorem, which says that a structure is oligomorphic if and only if it is ω-categorical.

Lemma 4.4 *In an oligomorphic structure, every equivariant set of n-tuples is first-order definable.*

Proof
Consider the following equivalence relations on tuples of equal length: a) they satisfy the same formulas of first-order logic with n-free variables, of given quantifier rank at most k; and b) they are in the same equivariant orbit. Each of these equivalence relations is characterised by an appropriate Ehrenfeucht-Fraïssé game, which differ in the number of rounds that is played: in the first kind of game k rounds are played, in the second kind of game infinitely many rounds are played. We claim that when the structure is oligomorphic, then Spoiler wins the infinite round game if and only if for some k, he wins the k-rounds game. In other words, for every two tuples in different orbits, there is a formula of first-order logic that distinguishes them. This claim proves the lemma, since there are finitely many equivariant orbits of given dimension, and therefore each is definable in first-order logic. Other equivariant sets of n-tuples are finite unions of orbits.

To prove the claim, consider the situation in the infinite round game 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 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 Koenig lemma to show that Spoiler wins the infinite round game if and only if for some k he winds the k-round game. □

Theorem 4.5 *Suppose that the atoms are an oligomorphic structure and $\bar{a}$ is a tuple of atoms. A set of n-tuples of atoms is supported by $\bar{a}$ if and only if it is definable by a first-order formula with n free variables and constants from $\bar{a}$.*

Proof
The right-to-left implication is immediate: the truth value of first-order formulas with constants from $\bar{a}$ is invariant under $\bar{a}$-automorphisms. For the left-to-right implication, consider a set X of n-tuples 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 4.4 it is definable by a formula of first-order logic φ. A tuple $\bar{b}$ belongs to X if and only if it satisfies $\varphi(\bar{a}\bar{b})$. □

5 Hereditarily definable sets

Consider an arbitrary logical structure for the atoms, not necessarily oligomorphic. We define the notion of a *hereditarily definable set* below. Fix some infinite set of variables, which are meant to range over atoms. If $\bar{x}$ is a tuple of variables, then an *$\bar{x}$-valuation* is a function that maps each variable in the tuple to an atom. A hereditarily orbit-finite set will be defined by an expression, whose syntax is given below, plus a valuation of the variables in that expression.

- **Atom expression.** A variable is an expression, called an *atom expression*.
- **Empty set.** There is a constant $\emptyset$ that represents the empty set.
- **Set expression.** Let $\bar{x}, \bar{y}$ be disjoint tuples of variables and let α be an already defined expression with free variables included in $\bar{x}\bar{y}$. Let φ be a first-order formula with free variables $\bar{x}\bar{y}$, which does not use any atom constants and therefore defines an equivariant set of atom tuples. Then

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

  is an expression, called a *set expression*. The free variables are $\bar{x}$, and the variables $\bar{y}$ are bound.
- **Union expression.** If $\alpha_1, \ldots, \alpha_n$ are set expressions, then so is $\alpha_1 \cup \cdots \cup \alpha_n$. Such an expression is called a *union expression*.

To represent a hereditarily definable set, we use an expression α together a valuation of its free variables, which is a tuple of atoms $\bar{a}$.

Algorithms on hereditarily definable sets. We say that a logical structure is *effective* if one can represent its universe (e.g. there exists an enumeration of its universe) so that one can decide, given a formula of first-order logic and a valuation of its variables (using the given representation), if the formula is true.

Example 24. Presburger arithmetic, i.e. the natural numbers equipped with addition, is an effective structure which is not oligomorphic. □

To represent a hereditarily definable set, we need some way of representing the valuation of free variables. The assumption that the atoms are effective gives a way of representing the valuation, and therefore it makes sense to talk about algorithms which input or output hereditarily definable sets.

Theorem 5.1 *[Representation of hereditarily orbit-finite sets] Suppose that the atoms are effective, but not necessarily oligomorphic. Then there are algorithms implementing the following operations:*

1. **Singleton.** *Given X, which is a hereditarily definable set, compute $\{X\}$;*
2. **Binary union.** *Given hereditarily definable sets X, Y, compute $X \cup Y$;*
3. **General union.** *Given a hereditarily definable set, compute $\bigcup X$.*
4. **Support.** *Given a hereditarily definable set X, compute some finite support;*
5. **Choice.** *Given a hereditarily definable set, produce an element or say it has no elements;*
6. **Inclusion.** *Given hereditarily orbit-finite sets X, Y, decide if $X \subseteq Y$.*

Lemma 5.2 *For every expression α one can compute an expression $\cup\alpha$ with the same free variables such that*

$$\cup\alpha(\bar{a}) = \bigcup_{x \in \alpha(\bar{a})} x \qquad \text{for every } \bar{a}$$

Proof
The proof is unfolding the definition of set union. The interesting case is when α is a set expression

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

Suppose that β is a finite union of several expressions. Those expressions in the union that are atoms do not contribute anything, so we may assume without loss of generality that β is a finite union

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

Therefore, the union $\bigcup \alpha(\bar{a})$ is equal to

$$\bigcup_{i \in I} \{\gamma_i(\bar{x}\bar{y}\bar{z}_i) : \text{ for } \bar{y}\bar{z}_i \text{ such that } \varphi_i(\bar{x}\bar{y}\bar{z}_i) \wedge \varphi(\bar{x}\bar{y})\}.$$

□

Fact 5.3 *Let α, β be expressions whose free variables are included in $\bar{x}$. One can compute a formula of first-order logic which selects exactly those valuations $\bar{a}$ which make the inclusion $\alpha(\bar{a}) \subseteq \beta(\bar{a})$ hold. Likewise for membership $\in$.*

Proof
The proof is done in parallel for the operations $\in$ and $\subseteq$, by induction on the combined size of the expressions. Consider the following cases, depending on the operation.

$\subseteq$ If the left side is an atom, then the inclusion formula is false, since an atom is not a subset of anything. If the left side is a finite union, then all of its components must be a subset of the right side, and therefore a finite conjunction of inductively obtained formulas is used. The remaining case is when the left side is a set expression

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

In this case inclusion boils down to membership for smaller expressions: for every $\bar{y}$ satisfying $\tau(\bar{x}\bar{y})$, the expression $\alpha(\bar{x}, \bar{y})$ must be a member of the right side.

$\in$ If the right side is an atom, then the formula is simply false, because an atom has no elements. If the right side is a union, then we need to check if the left side is a member of some set on the right side, and therefore we use a finite disjunction of inductively obtained formulas. The remaining case is when the right side is a set expression

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

To test if the left side is a member of the above, we need to test if there is some valuation of $\bar{y}$ which satisfies τ, such that the left side is equal to $\beta(\bar{x}, \bar{y})$. When the left side is an atom, then equality is built into first-order logic, otherwise equality is inclusion both ways, which can be described using the induction assumption.

□

The atoms in the tuple $\bar{a}$ are represented by the data structures described in Exercise 9. An expression together with a valuation is denoted by $\alpha(\bar{a})$, the semantics are defined in the obvious way.

A *set builder expression* is defined

5.1 Hereditarily orbit-finite sets and algorithms on them

In the previous section, we have shown that when the are oligomorphic then, like finite sets, they are closed under finite products, images of functions and subsets. In this section we show that, when the atoms are oligomorphic and have a decidable first-order theory, then sets which are hereditarily orbit-fintie

(they are orbit-finite, their elements are orbit-finite and so on until an atom or the empty set is reached) can be represented in a finite way and manipulated by algorithms.

The point is to be able to state and prove results such as: "it is decidable if an automaton with atoms is nonempty" or "one can effectively transform a deterministic automaton with atoms into an equivalent minimal one". If the automaton is defined to be a special case of a hereditarily orbit-finite set, as it will be in Section 8, and if hereditarily orbit-finite set can be represented by a data structures without atoms, as they will be in Theorem 5.4, then the standard notion of decidability can be applied to decide properties of automata with atoms.

Hereditarily orbit-finite sets are essentially the only orbit-finite sets, up to equivariant bijections.

Exercise 6. Show that when the atoms are oligomorphic, then every orbit-finite set is the image of a hereditarily orbit-finite set under an equivariant bijection.

Assumptions on the atoms. We say that a structure is *effective* if one can represent its universe (e.g. there exists an enumeration of its universe) so that one can decide, given a formula of first-order logic and a valuation of its variables (using the given representation), if the formula is true.

Example 25. Presburger arithmetic, i.e. the natural numbers equipped with addition, is an effective structure which is not oligomorphic. □

Exercise 7. Consider an oligomorphic structure with a decidable first-order theory. Show that the following conditions are equivalent:

1. given n, one can compute the number of equivariant orbits of n-tuples;
2. given n, one can compute a first-order formula in $2n$ variables which defines the relation "two n-tuples of atoms are in the same orbit".

The toolkit for hereditarily orbit-finite sets. We now show what operations can be implemented on finite representations of hereditarily orbit-finite sets.

Theorem 5.4 *[Representation of hereditarily orbit-finite sets] Suppose that the atoms are effectively oligomorphic. Then there are data structures for representing atoms and hereditarily orbit-finite sets, which admit the following operations:*

1. **Singleton.** *Given X, which is a hereditarily orbit-finite set or an atom, compute $\{X\}$;*
2. **Binary union.** *Given hereditarily orbit-finite sets X, Y, compute $X \cup Y$;*

3. **Support.** *Given a hereditarily orbit-finite set X, compute some finite support;* [5]

4. **Intersecting orbits.** *Given a hereditarily orbit-finite set X and a finite tuple $\bar{a}$ of atoms, produce all hereditarily orbit-finite sets that intersect X and are $\bar{a}$-orbits;* [6]

5. **Choice.** *Given a nonempty hereditarily orbit-finite set, produce an element;*

6. **Inclusion.** *Given hereditarily orbit-finite sets X, Y, decide if $X \subseteq Y$.*

Exercise 8. Show that the following operations can be implemented using the representations from Theorem 5.4.

1. Given a function f and an argument x, compute $f(x)$;
2. Given a function f and a set of arguments X, compute the image $f(X)$.

The rest of this section is devoted to showing Theorem 5.4.

The data structure for the atoms. The following exercise shows that the atoms can be represented so that the truth value of first-order formulas with constants from the atoms can be decided. The representation in the solution of the exercise is very inefficient, so a more efficient implementation of the toolkit in Theorem 5.4 would need also to input, as an additional parameter, a more efficient implementation of the atoms and their first-order decision procedure.

Exercise 9. Consider an effectively oligomorphic structure. Then one can represent every element of the universe by a finite data structure, say a string of bits, such that one can also decide which first-order formulas with free variables are true, given a representation of the valuation of the variables.

Exercise 10. Assume that the atoms are oligomorphic and infinite. Let f be a representation as in the previous exercise, seen as a function from bit strings to atoms. Show that f is not finitely supported.

The data structure for the sets. We now define the data structure for representing hereditarily orbit-finite sets. Essentially, a set is represented by an expression in set builder notation, which uses formulas of first-order logic over the atoms, possibly with some constants from the atoms.

Fix some infinite set of variables, which are meant to range over atoms. If $\bar{x}$ is a tuple of variables, then an *$\bar{x}$-valuation* is a function that maps each variable

[5] This finite support is represented as a list of atoms. There is no requirement that this support is optimal in any sense, e.g. least with respect to inclusion.

[6] There are finitely many $\bar{a}$-orbits that intersect X, so they can be given as list of sets.

in the tuple to an atom. A hereditarily orbit-finite set will be defined by an expression, whose syntax is given below, plus a valuation of the variables in that expression.

- **Atom expression.** A variable is an expression, called an *atom expression*.
- **Empty set.** There is a constant $\emptyset$ that represents the empty set.
- **Set expression.** Let $\bar{x}, \bar{y}$ be disjoint tuples of variables and let α be an already defined expression with free variables included in $\bar{x}\bar{y}$. Let φ be a first-order formula with free variables $\bar{x}\bar{y}$, which does not use any atom constants and therefore defines an equivariant set of atom tuples. Then

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

  is an expression, called a *set expression*. The free variables are $\bar{x}$, and the variables $\bar{y}$ are bound.
- **Union expression.** If $\alpha_1, \ldots, \alpha_n$ are set expressions, then so is $\alpha_1 \cup \cdots \cup \alpha_n$. Such an expression is called a *union expression*.

To represent an atom or a hereditarily orbit-finite set, we use an expression α together a valuation of its free variables, which is a tuple of atoms $\bar{a}$. The atoms in the tuple $\bar{a}$ are represented by the data structures described in Exercise 9. An expression together with a valuation is denoted by $\alpha(\bar{a})$, the semantics are defined in the obvious way.

Example 26. We can treat $\{\alpha\}$ as syntactic sugar for

$$\{\alpha : \text{for } \epsilon \text{ such that true}\}$$

where ϵ is the empty tuple. Using union and singleton, one can encode any hereditarily finite (not just orbit-finite) set. In particular, ordered pairs under the Kuratowski definition can be encoded, and therefore every finitely supported set of atom tuples can be encoded by an expression. □

Exercise 11. Show that expressions together with valuations define exactly the atoms and hereditarily orbit-finite sets.

Manipulating expressions. We implement the operations in Theorem 5.4.

- **Support.** Clearly $\alpha(\bar{a})$ is supported by $\bar{a}$, although this might not be the least support (even if such a thing exists). Therefore, a correct implementation of the support operation is the function that inputs an expression together with a valuation, and returns the valuation.
- **Binary union.** Included in the syntax of expressions.
- **Singleton.** See Example 26.

- **Intersecting orbits.** Suppose that we want all $\bar{c}$-orbits that intersect a set represented by $\alpha(\bar{a})$. The interesting case is when α is a set expression

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

  The union of $\bar{c}$-orbits intersection $\alpha(\bar{a})$ is the set

$$\{\pi(\beta(\bar{a}\bar{b})) : \text{ for } \bar{b} \text{ and } \pi \text{ such that } \varphi(\bar{a}\bar{b}) \text{ and } \pi \text{ is a } \bar{c}\text{-automorphism}\}.$$

  Because the function $\bar{a}\bar{b} \mapsto \beta(\bar{a}\bar{b})$ is equivariant, the set above is equal to

$$\{\beta(\pi(\bar{a}\bar{b})) : \text{ for } \bar{b} \text{ and } \pi \text{ such that } \varphi(\bar{a}\bar{b}) \text{ and } \pi \text{ is a } \bar{c}\text{-automorphism}\}.$$

  The set above can be restated as

$$\{\beta(\bar{a}'\bar{b}') : \text{ for } \bar{a}'\bar{b}'\bar{b} \text{ such that } \varphi(\bar{a}\bar{b}) \text{ and } \bar{a}\bar{b} \text{ is in the same } \bar{c}\text{-orbit as } \bar{a}'\bar{b}'\}.$$

  By the assumption that the atoms are effectively oligomorphic, the condition that two tuples $\bar{a}\bar{b}$ and $\bar{a}'\bar{b}'$ are in the same $\bar{c}$-orbit is definable in first-order logic, call this formula ψ. Therefore, the above is equal the following set, which is clearly represented by an expression, with the free variables valued to $\bar{a}\bar{c}$.

$$\{\beta(\bar{a}'\bar{b}') : \text{ for } \bar{a}'\bar{b}'\bar{b} \text{ such that } \varphi(\bar{a}\bar{b}) \wedge \psi(\bar{c}\bar{a}\bar{b}\bar{a}'\bar{b}')\}$$

- **Inclusion.** The following fact reduces inclusion and membership to deciding the truth value of first-order formulas over the atom structure, which is decidable by the effectivity assumption.

  Fact 5.5 *Let α, β be expressions whose free variables are included in $\bar{x}$. One can compute a formula of first-order logic over the atoms $\varphi(\bar{x})$ which selects exactly those valuations $\bar{a}$ which make the inclusion $\alpha(\bar{a}) \subseteq \beta(\bar{a})$ hold. Likewise for membership $\in$.*

  Proof
  The proof is done in parallel for the operations $\in$ and $\subseteq$, by induction on the combined size of the expressions. Consider the following cases, depending on the operation.

  $\subseteq$ If the left side is an atom, then the inclusion formula is false, since an atom is not a subset of anything. If the left side is a finite union, then all of its components must be a subset of the right side, and therefore a finite conjunction of inductively obtained formulas is used. The remaining case is when the left side is a set expression

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

  In this case inclusion boils down to membership for smaller expressions: for every $\bar{y}$ satisfying $\tau(\bar{x}\bar{y})$, the expression $\alpha(\bar{x}, \bar{y})$ must be a member of the right side.

$\in$ If the right side is an atom, then the formula is simply false, because an atom has no elements. If the right side is a union, then we need to check if the left side is a member of some set on the right side, and therefore we use a finite disjunction of inductively obtained formulas. The remaining case is when the right side is a set expression

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

To test if the left side is a member of the above, we need to test if there is some valuation of $\bar{y}$ which satisfies τ, such that the left side is equal to $\beta(\bar{x}, \bar{y})$. When the left side is an atom, then equality is built into first-order logic, otherwise equality is inclusion both ways, which can be described using the induction assumption.

□

This completes the proof of Theorem 5.4. Note that for all operations, apart from intersecting orbits, we only used the weaker assumption that the atoms can be represented so that truth value of first-order formulas can be decided. Only in the intersecting orbits operation we used the fact that there is a first-order formula defining the relation "tuples are in the same orbit". The weaker assumption holds for a wide variety of structures that are not effectively oligomorphic, e.g. Presburger arithmetic $(\mathbb{N}, +)$ or integers with multiplication $(\mathbb{N}, \cdot)$.

Uniqueness of representations. Suppose that ρ is a surjective function from bit strings to hereditarily orbit-finite sets and atoms. Such a function is called a *representation function* if it supports all the operations in Theorem 5.4, i.e. one can decide if two bit strings represent the same set, one can compute a bit string representing the union of two sets represented by bit strings, etc.

Theorem 5.6 *If ρ_1, ρ_2 are representation functions, then there is a computable function f from bit strings to bit strings and an automorphism π of the atoms such that the following diagram commutes:*

$$\begin{array}{ccc} \textit{bit strings} & \xrightarrow{f} & \textit{bit strings} \\ \downarrow \rho_1 & & \downarrow \rho_2 \\ \textit{hof sets and atoms} & \xrightarrow{\pi} & \textit{hof sets and atoms} \end{array}$$

The proof of the theorem is in the following exercises.

Exercise 12. If ρ is a representation function, then given a first-order formula and a valuation represented by ρ, one can test if the formula is true under the valuation.

Exercise 13. One can define f on the ρ_1-representations of atoms such that for some atom automorphism π, the diagram in Theorem 5.6 commutes for f restricted to ρ_1-representations of atoms.

Exercise 14. Let the partially defined f and the automorphism π be as in the previous exercise. Then f can be extended to ρ_1-representations of hereditarily orbit-finite sets, so that the diagram in Theorem 5.6 commutes for π.

6 Homogeneous atoms

In this section, we study atoms which are homogeneous. Recall that in an oligomorphic structure, every finitely supported property of tuples of atoms is definable by a formula of first-order logic. In a homogeneous structure, quantifier-free formulas are enough. Another benefits of homogeneity is that every homogeneous structure is obtained by a taking a limit of a class of finite structures closed under amalgamation. This gives a way of producing homogeneous structures, taking the limit of classes such as: the class of all finite total orders, the class of all directed graphs, the class of all equivalence relations, etc.

Finitely generated substructures and blowups. An *(induced) substructure* of a structure $\mathfrak{A}$ is defined to be a structure obtained from $\mathfrak{A}$ 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 that contains the subset. A *finitely generated* substructure is defined to be one generated by a finite subset of the universe. The *blowup* of a structure is the function which maps $n \in \mathbb{N}$ to the biggest size of a substructure that is generated by n elements. A structure is said to have *unbounded blowup* if the blowup maps some n to ∞.

Example 27. [Blowups] Every structure without functions has blowup $n \mapsto n$, because only functions can add elements. The blowup of $(\mathbb{Z}, +1)$ maps 1 to ∞, because every integer generates an infinite set. Consider the powerset of the natural numbers

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

with the union function. The blowup is $n \mapsto 2^n$, because the sets

$$\{1\}, \{2\}, \ldots, \{n\}$$

generate the powerset of $\{1, \ldots, n\}$. □

Example 28. [Bit vector atoms] Consider the following structure. We use the name *bit vector* for an infinite sequence of zeros and ones which has finitely

many ones. By ignoring the trailing zeros, 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 binary function for addition modulo two.

The bit vectors are actually a linear 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 consists of bit vectors which have a 1 on all the first n coordinates: $1, 11, 111, 1111, \ldots$.

The substructure generated by a set of bit vectors is all possible linear combinations of those vectors. Since a vector can be used 0 or 1 times in a linear combination, it follows that the blowup for the bit vector structure is $n \mapsto 2^n$. In particular, the bit vector structure has (exponentially) bounded blowup. □

Homogeneous structures. A structure is called *homogeneous* if every isomorphism between finitely generated substructures extends to a full automorphism. In this section we study atom structures that are homogeneous over finite vocabularies, possibly including functions.

Example 29. The empty set of atoms, the equality atoms, and the total order atoms are all homogeneous structures over finite vocabularies, which explains why they are oligomorphic. The powerset of the natural numbers with inclusion is not homogeneous, because $\emptyset \mapsto \{1\}$ is a finite partial automorphism which does not extend to a full automorphism. □

Example 30. [Integers are or are not homogeneous, depending on vocabulary] 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 $(\mathbb{Z}, \leq)$ and $(\mathbb{Z}, +1)$, with $+1$ being the successor function. The two structures have the same automorphism group, namely the translations. The first structure is not homogeneous, because the function that maps 0 to 0 and 1 to 2 is an isomorphism between finitely generated substructures. The second structure is homogeneous, because finitely generated substructures are half-intervals of the form $\{i, i+1, \ldots\}$. □

Example 31. [Bit vector atoms are homogeneous] We show that 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 finite partial automorphism? 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. □

6.1 Finitely supported is the same as quantifier-free definable

In this section, we show that, in a homogeneous structure with finite vocabulary and bounded blowup, finitely supported relations on atoms are exactly the same thing as relations definable by quantifier-free formulas.

Lemma 6.1 *Assume that the atoms are homogeneous. Two tuples of atoms 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. □

Lemma 6.2 *Assume that the atoms are homogeneous, over a finite vocabulary, and have bounded blowup. Then the atoms are oligomorphic. Furthermore, a set of n-tuples of atoms is $\bar{a}$-supported if and only if it is definable by a quantifier-free formula with constants from $\bar{a}$.*

Proof
Let us first show oligomorphism. From the assumption on bounded blowup, there is some k such that substructures of the atoms with n generators have size at most k. It follows by a pumping argument that 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 6.1, there are finitely many equivariant orbits of n-tuples, which proves oligomorphism.

Consider now the "Furthermore" part. The right-to-left implication is immediate. For the left-to-right implication, we use oligomorpism to show that a $\bar{a}$-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 6.1. □

Exercise 15. Show that if the atoms are oligomorphic, then they have bounded blowup.

A further corollary of Lemma 6.2 is that in structures which satisfy its assumptions, every formula of first-order logic is equivalent to a quantifier-free formula. This is because the truth value of formulas of first-order logic is preserved under full automorphisms, and therefore the set of tuples defined by a

formula is equivariant, and therefore quantifier-free definable by Lemma 6.2. The same proof works also for richer logics, such as higher-order logics, and for formulas which contain certain atoms as constants.

Example 32. Consider the equality atoms. We redo Example 5, only using Corollary 6.2. By the corollary, a binary relation on atoms is supported by the empty set (i.e. equivariant) if and only if it is definable by a quantifier-free formula. Up to logical equivalence, there are only four quantifier-free formulas in two variables, namely

$$\bot \qquad \top \qquad x = y \qquad x \neq y.$$

These formulas correspond exactly to the relations discussed in Example 5. □

6.2 The Fraïssé limit

We have seen some examples of homogeneous atoms, such as in the equality atoms or the total order atoms, or even the bit vector atoms. In this section we present a construction, called the *Fraïssé limit*, which inputs a class of finitely generated structures, and produces a single (usually infinite) homogeneous structure, called its Fraïssé limit. Actually, the Fraïssé limit is the only possible way of producing a countable homogeneous structure, since we shall see that every homogeneous structure is the Fraïssé limit of its finitely generated substructures.

The age of a homogeneous structure. The *age* of a relational structure is defined to be the class of finitely generated structures that embed into it. The following theorem shows that a countable homogeneous structure is uniquely determined by its age.

Theorem 6.3 *If two countable homogeneous structures have the same age, then they are isomorphic.*

We begin with a lemma, which gives the only property of homogeneous structures that is used in the proof of the theorem.

Lemma 6.4 *Let $\mathfrak{A}$ be in the age of a homogeneous structure $\mathfrak{H}$. Every partial isomorphism between substructures of $\mathfrak{A}$ and $\mathfrak{H}$ extends to an embedding of $\mathfrak{A}$ in $\mathfrak{H}$.*

Proof
Let f be the partial isomorphism, and let $\mathfrak{B}$ be the substructure of $\mathfrak{A}$ on which the partial isomorphism is defined. Since $\mathfrak{A}$ is in the age, there is some embedding $g : \mathfrak{A} \to \mathfrak{H}$. These functions are illustrated in the following diagram:

$$\begin{array}{ccc} \mathfrak{B} & \xrightarrow{id} & \mathfrak{A} \\ {\scriptstyle f}\downarrow & & {\scriptstyle g}\downarrow \\ \mathfrak{H} & & \mathfrak{H} \end{array}$$

By following f^{-1} and then g, we get a partial automorphism of $\mathfrak{H}$. By homogeneity, this partial automorphism extends to a full automorphism, which makes the following diagram commute.

$$\begin{array}{ccc} \mathfrak{B} & \xrightarrow{id} & \mathfrak{A} \\ {\scriptstyle f}\downarrow & & {\scriptstyle g}\downarrow \\ \mathfrak{H} & \xrightarrow{\pi} & \mathfrak{H} \end{array}$$

The extension required by the lemma is then g followed by π^{-1}. □

Example 33. Consider a countably infinite directed graph, where every edge is chosen independently with equal probability one half. With probability one, every partial isomorphism between a finite graph and this graph extends to an embedding. In other words, with probability one, this graph has the property from Lemma 6.4. □

Theorem 6.3 follows from the following lemma.

Lemma 6.5 *Let $\mathfrak{H}_1, \mathfrak{H}_2$ be countable structures with the same age, which both have the property from Lemma 6.4. Then every partial isomorphism between finitely generated substructures of $\mathfrak{H}_1$ and $\mathfrak{H}_2$ extends to a full isomorphism.*

Proof
We prove the following claim:

> Consider a partial isomorphism f between finitely generated substructures of $\mathfrak{H}_1$ and $\mathfrak{H}_2$. For every element a of either $\mathfrak{H}_1$ or $\mathfrak{H}_2$, the partial isomorphism can be extended to be defined also on a.

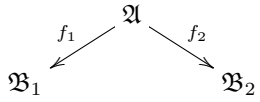
Let f and a be as in the claim. Without loss of generality, assume that a is in $\mathfrak{H}_1$. Let $\mathfrak{A}$ be the substructure of $\mathfrak{H}_1$ generated by a plus all the elements where f is defined. Since $\mathfrak{A}$ is in the age of $\mathfrak{H}_1$, it is also in the age of $\mathfrak{H}_2$. The function f is a partial isomorphism between $\mathfrak{A}$ and $\mathfrak{H}_2$, and therefore it extends by Lemma 6.4 to an embedding of $\mathfrak{A}$ in $\mathfrak{H}_2$, which is the extension required by the claim.

The lemma follows from the claim by a zig-zag construction. Define inductively a sequence of partial isomorphisms between finite substructures of $\mathfrak{H}_1$ and $\mathfrak{H}_2$, 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. □

This completes the proof of Theorem 6.3.

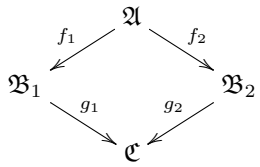
Example 34. Consider the random graph described in Example 33. Because the property from Lemma 6.4 holds with probability one, then by Lemma 6.5 there is one graph that we get with probability one, up to isomorphism. This graph is called the *random directed graph*. By applying Lemma 6.5 with both $\mathfrak{H}_1$ and $\mathfrak{H}_2$ being the random directed graph, we see that the random directed graph is homogeneous. The age is the set of all finite directed graphs. □

Since a countable homogeneous structure is uniquely identified by its age, it is natural to ask: which classes of finite structures are ages of countable homogeneous structures? The special property which distinguishes the age of a homogeneous structure, among other classes of finitely generated structures, is amalgamation, which is described below. An *instance of amalgamation* consists of two embeddings with a common source:



(8)

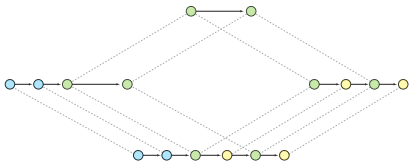
A *solution* of the instance consists of a structure $\mathfrak{C}$ and two embeddings g_1, g_2 such that the following diagram commutes



(9)

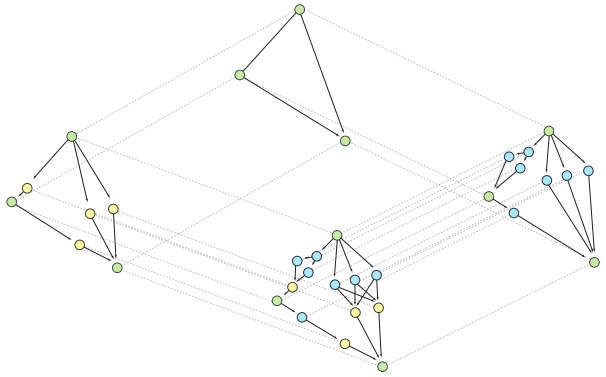
We say a class of structures is *closed under amalgamation* if every instance of amalgamation which uses structures from the class has a solution which also uses a structure from the class.

Example 35. Consider the class of finite total orders. This class is closed under amalgamation. Here is an example of an instance and solution of amalgamation:



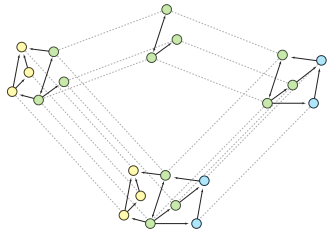
□

Example 36. Consider the class of all finite partial orders, not necessarily total orders. This class is closed under amalgamation. Here is an example of an instance and solution of amalgamation:



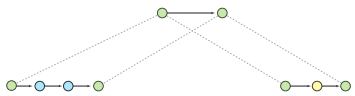
One way of amalgamating two partial orders, which is illustrated in the picture above, is to put the elements of the left (yellow) order after the elements of the right (blue) order, as long as they have the same relationship with the common (green) elements. Other strategies lead to other amalgamations.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

Example 37. Consider the class of all finite directed graphs. This class is closed under amalgamation. An instance of amalgamation is basically two directed graphs that have a common induced subgraph. One way of amalgamating them looks like this:



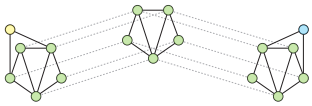
The amalgamation above is not unique; one could also add edges between the blue and yellow nodes. This shows that, for instance, the class of all cliques is closed under amalgamation. □

Example 38. Consider the class of finite directed graphs, 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 39. Consider the class of (undirected) 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:



□

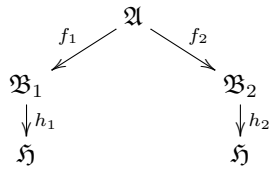
The following theorem shows that amalgamation characterises uniquely the ages of countable homogeneous structures.

Theorem 6.6 *Let $\mathscr{A}$ be a class of finitely generated structures over a common vocabulary, which is closed under isomorphism and substructures. Then $\mathscr{A}$ is the age of a countable homogeneous structure if and only if it is closed under amalgamation.*

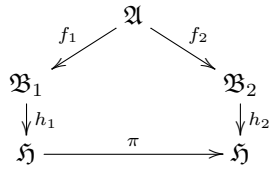
The two implications of the theorem are proved in Lemmas 6.7 and 6.8.

Lemma 6.7 *The age of a countable homogeneous structure is closed under amalgamation.*

Proof
Let $\mathfrak{H}$ be a homogeneous structure. Consider an instance of amalgamation as in (8). Because the structures are in the age of $\mathfrak{H}$, there are embeddings h_1, h_2 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 $\mathfrak{A}$ in $\mathfrak{H}$. However, the images of both of these embeddings are isomorphic substructures of $\mathfrak{H}$; and by homogeneity there is a full automorphism π which extends this partial automorphism. In other words, the following diagram commutes:

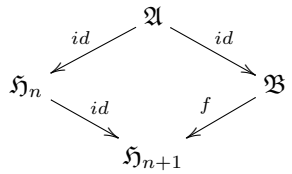


The above diagram shows a solution of amalgamation. □

Lemma 6.8 *If a class of finitely generated structures is closed under isomorphism, substructures and amalgamation, then it is the age of a countable homogeneous structure.*

Proof
Let $\mathscr{A}$ be a class of structures satisfying the assumptions of the lemma. We define a sequence of finitely generated structures $\mathfrak{H}_1, \mathfrak{H}_2, \ldots \in \mathscr{A}$ with the following property:

- For every $n \in \mathbb{N}$, the structure $\mathfrak{H}_n$ is a substructure of $\mathfrak{H}_{n+1}$.
- Let $n \in \mathbb{N}$. Suppose that $\mathfrak{A}$ is a substructure of both $\mathfrak{H}_n$ and some $\mathfrak{B} \in \mathscr{A}$. If $\mathfrak{B}$ has size at most n, then $\mathfrak{B}$ embeds into $\mathfrak{H}_{n+1}$ in a way consistent with $\mathfrak{A}$, i.e. there is some f such that following diagram commutes:



The sequence is defined by induction. To obtain $\mathfrak{H}_{n+1}$ from $\mathfrak{H}_n$, we apply amalgamation to all possible embeddings of substructures of $\mathfrak{H}_n$ into structures from $\mathscr{A}$ of size at most n.

Define $\mathfrak{H}$ to be the limit (i.e. union) of the sequence $\mathfrak{H}_1, \mathfrak{H}_2, \ldots$. We claim that $\mathfrak{H}$ is homogeneous. The age of $\mathfrak{H}$ is clearly $\mathscr{A}$, because every structure of size n embeds into $\mathfrak{H}_{n+1}$. By definition $\mathfrak{H}$ satisfies the property from Lemma 6.4. By Lemma 6.5 applied to the case of $\mathfrak{H}_1 = \mathfrak{H}_2 = \mathfrak{H}$, it follows that every finite partial automorphism of $\mathfrak{H}$ extends to a full isomorphism. □

This completes the proof of Theorem 6.6.

The countable homogeneous structure which is constructed in Lemma 6.8 is called the *Fraïssé limit*. The Fraïssé limit is the inverse operation of the age. Applying the Fraïssé limit to various classes from Examples 35, 36 and 37, we get several homogeneous structures.

- The Fraïssé limit of all directed total orders is the rational numbers with their order. This structure we know very well, it is the total order atoms.
- 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. We can use the Fraïssé limit of partial orders as a notion of atoms; we call these atoms the *partial order atoms*.
- There is a Fraïssé limit of all directed graphs. This directed graph is not easy draw, but it is the random directed graph from Example 34. Again, we can use the Fraïssé limit of directed graphs as a notion of atoms; we call these atoms the *directed graph atoms*.

For the partial order atoms and the directed graph atoms, the whole theory of sets with atoms works. In particular, since the vocabulary is finite and the Fraïssé limit is a homogeneous structure, we get all the good properties of orbit-finite sets, such as orbit refinement and closure under products and subsets. Finally, we can also use the expressions for hereditarily orbit-finite sets defined by Theorem 5.4. Recall that Theorem 5.4 assumed that the atoms are also effectively oligomorphic. This is the case for all the examples (equality, total order, partial order, directed graph), thanks to the following observation:

Lemma 6.9 *Assume that $\mathscr{A}$ is a class of finite structures that is closed under amalgamation, substructures and isomorphism (these assumptions guarantee a Fraïssé limit). If $\mathscr{A}$ has bounded blowup, the vocabulary is finite, and membership in $\mathscr{A}$ is decidable, then the Fraïssé limit of $\mathscr{A}$ is effectively oligomorphic.*

Exercise 16. Show that the property from Exercise 20 holds also for Fraïssé limit of all undirected graphs. (The same is true for directed graphs.)

7 What kind of definition is orbit-finiteness?

It would be nice if orbit-finiteness would correspond to some standard definition of finiteness, only interpreted in the sets with atoms. Let us consider three standard definitions of finiteness.

1. A finite set admits a bijection with $\{1, \ldots, n\}$ for some $n \in \mathbb{N}$.
2. A finite set does not admit a bijection with a proper subset of itself.
3. For a finite set, every chain of subsets has a maximal element.

The second property is called *Dedekind finiteness*. We will show that the first two definitions do not correspond to orbit-finiteness, but the third one does for some choices of atom structures.

The first definition, when interpreted in sets with atoms, gives the sets that are finite in the standard sense.

The following exercises show that, in the equality atoms, Dedekind finiteness covers all orbit-finite sets (Exercise 17), but also some sets that are not orbit-finite (Exercise 18).

Exercise 17. Show that in the equality atoms (actually, under any oligomorphic atoms), every orbit-finite is Dedekind finite.

Exercise 18. Show that in the equality atoms, there is a set that is not orbit-finite, but Dedekind finite.

Actually 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 7.1 *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.*

The third definition of finiteness looks most promising. First we note that the following alternative definition is too restrictive:

3'. For a finite set, every directed family of subsets has a maximal element.

(A partially ordered set is called directed if every two elements have a common upper bound. In the special case of a family of subsets, this means that any two sets from the family are included in some third set from the family.) Definition 3' is satisfied only by sets that are finite in the standard sense. Regardless of the choice of atoms, if X is an infinite, but possibly orbit-finite, set, then the family of finite subsets of X is a directed family, but has no maximal element.

Somehow magically, restricting to chains, as is the case in Definition 3, makes the definition work for a wide variety of atom structures. To explain this, we introduce yet another definition of finiteness, which talks about uniformly supported chains (or directed families). A set (in this case a chain of subsets or a directed family of subsets) is called *uniformly supported* if there is some finite set of atoms which supports all elements of the set. The following lemma shows that, when restricted to uniform sets, both Definitions 3 and 3' capture orbit-finiteness.

Lemma 7.2 *For oligomorphic atoms, a set X is orbit-finite if and only if it satisfies either of the following equivalent conditions:*

3u. Every uniform chain of subsets of X has a maximal element;

3u'. Every uniform directed family of subsets of X has a maximal element;

Proof
The implication from 3u' to 3u is immediate. For the implication from 3u 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 of X to 3u', suppose that $\mathcal{X}$ is a uniformly supported directed family of subsets of 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. □

Let us now come back to Definition 3. It turns out that for some choices of atoms, Definitions 3 and 3u are equivalent, and therefore Definition 3 captures orbit-finiteness. Examples include the equality atoms (Exercise 20) and the Fraïssé limit of graphs (Exercise 16). For other choices, the definitions are not equivalent, as seen in the following exercise.

Exercise 19. Show that in the total order atoms, Definitions 3 and 3u are not equivalent.

Exercise 20. Consider the equality atoms. Let X be a set which admits a finitely supported total order (e.g. a chain of subsets of some set, ordered by inclusion). Show that X is uniform.

Part II
The Chomsky Hierarchy

In this part of the paper, we discuss what happens to the Chomsky hierarchy when sets with atoms are used. The general theme is as follows: we take a classical definition, such as the definition of a nondeterministic finite automaton, and then replace "finite set" by "orbit-finite set", while all other objects, such as functions or relations which are not necessarily orbit-finite, are required to be legal sets with atoms. We then see which constructions work, and which ones fail.

8 Finite automata with atoms

In this section, we discuss finite automata in sets with atoms. Unless explicitly noted, we always consider the case when the atoms are effectively oligomorphic, with the ensuing good properties of orbit-finite states. Under these assumptions, a construction will work if it only uses unions, subsets and products (such as equivalence of pushdown automata and context-free grammars), but a construction might fail if it uses the powerset operation (such as determinization of finite automata).

8.1 Finite automata

In this section, we discuss the atom versions of nondeterministic and deterministic finite automata. We prove that the two models are not equivalent, and that the deterministic version admits a Myhill-Nerode characterization.

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". In other words, a *nondeterministic orbit-finite automaton* consists of an orbit-finite input alphabet A, an orbit-finite set of states Q, finitely supported subsets of initial and accepting states, and finitely supported set of transitions $\delta \subseteq Q \times A \times Q$. An automaton is called *deterministic* if it has one initial state, and δ is a function from $Q \times A$ to Q.

By the assumption that the atoms are oligomorphic, the initial and accepting states, as well as the transitions, are all orbit-finite sets, as finitely supported subsets of products of orbit-finite sets. Acceptance is defined in the usual way. The language recognized by an automaton is the set of words it accepts.

Atoms which support an automaton will also support the language recognised by the automaton. In particular, if an automaton is equivariant, then also its language is equivariant. is because the definition of the language recognised by an automaton is defined in set theory, using only the membership predicate. The following lemma shows that any such definition will preserve supports.

Lemma 8.1 *Suppose that $\varphi(x, y)$ is a formula of first-order logic which only uses the set membership $\in$ predicate. Suppose that the interpretation of φ defines a function from x to y in the model of (legal) sets with atoms. If $\varphi(x, y)$ holds in sets with atoms, then any support for x is also a support for y.*

Proof
Because the function defined by φ is equivariant, as a function from sets with atoms to sets with atoms. □

Example 40. Consider the equality atoms. Let the alphabet be

$$A = \{\{a, b\} : a \neq b \in \text{Atoms}\},$$

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.

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

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

$$Q = \text{Atoms}.$$

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

$$\delta = \{(a, \{a, b\}, a) : a \neq b \in \text{Atoms}\}.$$

□

Example 41. The automaton from the previous example, unlike some automata, can be determinized. 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 modeled 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 $\emptyset$. □

Example 42. Consider the graph atoms, which are the Fraïssé limit of directed graphs. In these atoms, a finite set of atoms induces a finite graph; all finite directed graphs can be obtained this way. Therefore, an automaton can read a sequence of atoms $a_1 \cdots a_n$ and recognize properties such as: "the sequence is a path" or "the graph induced by the sequence contains a clique of size three" or "the graph induced by the sequence is not a clique". This leads to the following question: for which graph properties X, is the following language recognised by a nondeterministic automaton:

$$L_X = \{a_1 \cdots a_n : \text{ the graph induced by } a_1, \ldots, a_n \text{ satisfies } X\}$$

To recognise L_X, the automaton should be prepared for an arbitrary enumeration of the vertices of the graph, possibly with repetitions. 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? □

Example 43. 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.

We claim that, unlike Minsky machines, this weaker model has decidable emptiness. The argument for decidability is that 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. □

Example 44. We illustrate importance of the assumption on oligomorphic atoms. Consider the integer atoms, which are not oligomorphic. Let $D \subseteq \mathbb{Z}$ be an arbitrary set of integers. Consider the language of integers sequences where every two consecutive letter have a difference in D:

$$L_D = \{a_1 \cdots a_n \in \mathbb{Z} : a_i - a_{i-1} \in D \text{ for every } i \in \{2, \ldots, n\}\}.$$

This language is recognized by a deteterministic automaton, which keeps in its state the last letter seen, and enters an error state if the difference is outside D:

$$\delta(q, a) = \begin{cases} a & \text{if } a - q \in D \\ \bot & \text{otherwise} \end{cases}.$$

The transition function is equivariant. Since the set D can be any set of integers, e.g. the prime numbers or some undecidable set, it follows that orbit-finite automata over the integers are too powerful to be interesting. □

Lemma 8.2 *Assume that the atoms are oligomorphic. The expressive power of nondeterministic orbit-finite automata is not changed if ϵ-transitions are allowed.*

Proof
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. □

It is easy to see that languages recognized by nondeterministic orbit-finite automata are closed under union (because orbit-finite sets are closed under disjoint union), intersection (because orbit-finite sets are closed under product) and concatenation (disjoint unions again, and using Lemma 8.2). They also contain all orbit finite subsets of A^*. This raises the question of regular expressions and a Kleene Theorem, but we do not discuss this question.

Example 45. Closure under intersection can fail for atoms that are not oligomorphic. Consider the integer atoms. Let the input alphabet be integers. Consider the languages: "the letters on even-numbered positions are all equal" and "the letters on odd-numbered positions are all equal". Both languages are recognised by deterministic orbit-finite automata, but their intersection is not, since it would require storing two numbers in the state. (Nondeterminism would not help.) □

What about closure under complementation? The standard proof relies on determinization, which relies on the subset construction. The subset construction fails in sets with atoms, because the powerset of a orbit-finite need not be orbit-finite. The following fact shows that complementation is actually impossible.

Lemma 8.3 *Languages recognized by nondeterministic orbit-finite automata are not closed under complementation.*

Proof
We do the proof in the equality symmetry, but after minor variations it works in other symmetries. The proof follows that same lines as the proof that finite memory automata of Francez and Kaminsky are not closed under complementation. Let L be the words over the alphabet of atoms, where some atom appears twice. This language is recognised by a nondeterministic orbit-finite automaton which guesses the atom that appears twice.

We claim that the complement, namely the non-repeating sequences of atoms, is not recognized by an nondeterministic orbit-finite automaton. The idea is that after reading sufficiently many atoms, the automaton must remove one of them from memory, and then that atom can appear again. Toward a contradiction, suppose that there is such an automaton, with a finite support. Let k be a number such that every state has a support of size at most n. Such a number exists, because the states are orbit-finite, and the size of supports is the same for all elements in the same orbit.

Consider an input word where all letters are different, of sufficiently large even length. This word should be accepted by the automaton, so there should be an accepting run. Let q be the state of the run after reading the first half of the input word. If n is big enough, then there is a letter a in the first half, and a letter a in the second half, such that both a, b do not appear in the support of q or the automaton. Therefore, the automorphism π which swaps a with b

preserves the support of both q and the automaton. Apply π to the input word and the accepting run, yielding a new input word and a new run. Since π fixes the support of the automaton, the new run is a run on the new input word. Since π fixes the support of q, the original and new runs have the same states after reading the first half of the input. Therefore, we can combine the first half of the original input word with the second half of the new word, and still get an accepting word. However, this hybrid word has a in both the first and second half. □

Corollary 8.4 *Deterministic and nondeterministic orbit-finite automata have different expressive powers.*

Example 46. Consider the equality atoms, and the following language

$$\{w^n : n \in \mathbb{N} \text{ and } w \in \text{Atoms}^* \text{ is a word where all letters are different}\}$$

Consider an automaton that checks if

- the word has infixes ab and ac, for some distinct atoms a, b, c; or
- the word begins with a, ends with b, and has an infix bc for some $c \neq a$.

This automaton rejects precisely the words from the language. Using the same idea, we can encode runs of a Turing machine as words over an orbit-finite alphabet, and write a nondeterministic orbit-finite automaton which rejects precisely encodings of accepting runs. Therefore, the universality problem, namely the question "does a given nondeterministic orbit-finite automaton reject some input", is undecidable. □

Minimization of deterministic automata As observed above, determinization fails. Something that does work, and which justifies the usefulness of deterministic orbit-finite automata, is the Myhill-Nerode theorem.

Example 47. [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

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

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 even 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. 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. □

We begin by recalling the standard definition of Myhill-Nerode equivalence. Suppose that $L \subseteq A^*$ is a language. We define two words $w, w' \in A^*$ to be L-equivalent if for every word $v \in A^*$, 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, 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. Because the syntactic automaton is defined using the language of set theory, Lemma 8.1 says that any support the language is a support of the syntactic automaton.

Theorem 8.5 *A language is recognized 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 recognizing deterministic automaton, by applying a finitely supported function. Since finitely supported functions preserve orbit-finite sets, it follows that the syntactic automaton must have an orbit finite state space, if the original automaton did. □

8.2 Register automata

The automata discussed so far in this section are rather abstract, in the sense that the state space is an arbitrary orbit-finite set with atoms. There are advantages to the abstract approach, e.g. it makes the Myhill-Nerode theorem possible. In this section, we discuss a more concrete model of automata, which uses registers to store atoms. This model is essentially the same thing as the *finite memory automata* of Francez and Kaminsky.

Register sets. Assume that the atoms are a homogeneous structure over a finite vocabulary. Define an *n-dimensional basic register set* to be a finitely supported set of n-tuples of atoms. By Lemma 6.2, this is the same as a set of n-tuples defined by a quantifier-free formula, possibly with constants from the atoms. Giving the dimension and the quantifier-free formula is called a *quantifier-free representation* of a the basic register set. Since quantifier-free

formulas can be equivalent, this representation is not unique. This representation can be exponentially more succinct than explicitly describing all orbits. For instance, even when the atoms have equality only, the set of all tuples of n-atoms is a basic register set, but it contains one orbit for every equality type.

A *register set* is a finite disjoint union of basic register sets, called its *components*, possibly of different dimensions. Its quantifier-free representation is the list of the quantifier-free representations of the underlying basic register sets. We consider two parameters of a quantifier-free representation of a register set: its *dimension* is the maximal dimension of a basic register set contained in it, and its *size* is the combined size of quantifier-free formulas in the underying basic register sets. Observe that the product of finitely many register sets is also a register set, and therefore it makes sense to talk about relations and functions on register sets which are register sets as well.

A register set is of course a special case of an orbit-finite set.

Register automata. Define a register automaton to be the special case of an orbit-finite automaton where all of the components (the states, the input alphabet, the initial and final states, and the transition relation) are register sets. Such an automaton can always be represented using quantifier-free formulas.

When representing an automaton, we can require the shape formulas for the input alphabet and the state space to be either types, or arbitrary quantifier-free formulas. In the same way, the transition formulas in an automaton can also be either types, or arbitrary quantifier-free formulas. These choices influence the complexity of decision problems, as illustrated by the following result.

Theorem 8.6 *Consider equivariant register automata over the equality atoms, given by register representations. The emptiness problem is:*

1. PSPACE*-complete if the size of the input is measured as the size of the register representation of both the states and the transitions.*
2. NP*-complete if the size of the input is measured as the number of orbits in the state space plus the size of the register representation of the transitions.*
3. *in* PTIME *if the size of the input is measured as the number of orbits in the state space plus the number of orbits in the transitions.*

Proof (sketch)

1.
   - PSPACE membership. A nondeterministic PSPACE can guess the accepted word.
   - PSPACE hardness. The problem is already hard for automata with a one letter alphabet (therefore the 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. The problem is already hard for a automata with a one letter input alphabet. Consider an automaton with where the state space has two orbits: two disjoint copies of non-repeating n-tuples of atoms. The first orbit is initial, and the second orbit is accepting. Therefore, whether or not the automaton is nonempty boils down to the question whether there exist nonrepeating n-tuples $\bar{a}$ and $\bar{b}$ such that the transition relation (a quantifier-free formula) is satisfied by $\bar{a}\bar{b}$. 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).
   - NP membership. Consider a graph, where the vertices are orbits of the state space, 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
     (a) 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.
     (b) 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 accpeting 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.

□

8.3 Other models of finite automata

So far, we have discussed only deterministic and nondeterministic left-to-right automata. In normal sets, there are many equivalent models for recognising regular languages, such as monoids, two-way automata, alternating automata, or monadic second-order logic. All of these models can be defined in sets with atoms, and basically only the trivial inclusions are known to be true. Figure 1 illustrates some of the models, and some counterexamples for inclusions. The figure (and our knowledge so far) is incomplete, e.g. it does not discuss two-way automata that are alternating or nondeterministic, and it does not include

counterexamples for some inclusions (these inclusions might not be strict, as far as we know). For a more detailed discussion, including precise definitions of the models, see [**?**].

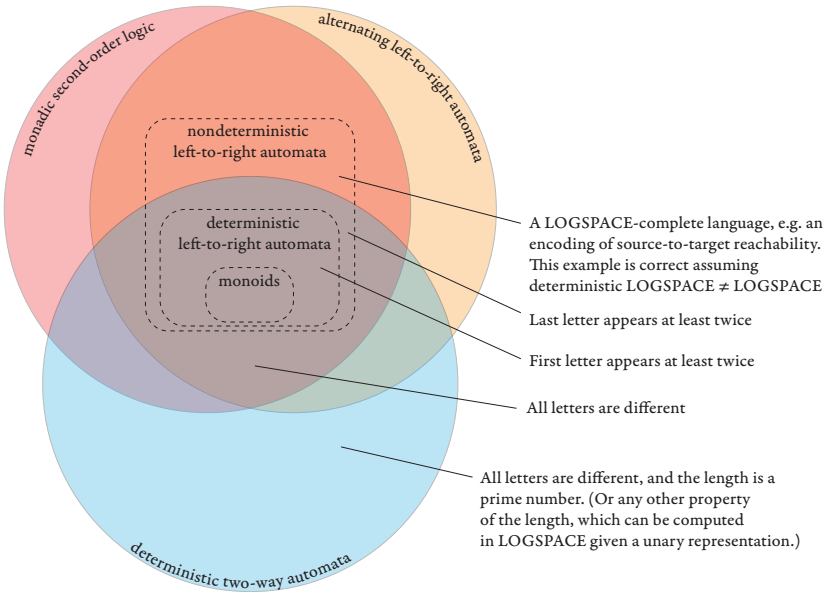


Figure 1: Inclusions between various candidates for "regular" languages, for atoms with equality.

9 Context-free languages with atoms

In this section, we discuss context-free languages with atoms. For the rest of Section 9, fix some oligomorphic structure for the atoms. We will show that pushdown automata are equivalent to context-free grammars. In a nutshell, the standard proof works, because it does not use the subset construction.

9.1 Pushdown automata.

An orbit-finite pushdown automaton is defined the same way as normal pushdown automaton, except the word "finite" is replaced by "orbit finite", and all components are required to be sets with atoms.

A *orbit-finite pushdown automaton* $\mathcal{A}$ consists of orbit-finite input and stack alphabets A, Γ, distinguished initial states and stack symbols, an orbit-fintie

state space Q, and a finitely supported set of transitions

$$\delta \subseteq Q \times \Gamma \times (A \cup \epsilon) \times Q \times \Gamma^*.$$

The semantics of the pushdown automaton are defined in the standard way (we use acceptance through the empty stack). Because the semantics of an automaton are defined in the language of set theory, Lemma 8.1 implies that the language is supported by any atoms that support the automaton.

Example 48. [Pushdown automaton for palindromes.] For an alphabet A, consider the language of palindromes:

$$P = \{a_1 a_2 \cdots a_n a_n \cdots a_2 a_1, a_1 a_2 \cdots a_n \cdots a_2 a_1 : a_1, \ldots, a_n \in A\} \subseteq A^*$$

This language is recognized 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 A. For instance, in the case when $A =$ Atoms, the automaton keeps a stack of data values 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 49. [Pushdown automaton for modified palindromes.] The automaton in Example 48 had two control states. In some cases, it might be useful to have a set Q of control states that is orbit-finite, but not finite. Consider for example the set of odd-length palindromes where the middle letter is equal to the first letter:

$$P' = \{a_1 a_2 \cdots a_n a_1 a_n \cdots a_2 a_1 : a_1, \ldots, a_n \in A\} \subseteq A^*.$$

A natural automaton recognizing 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 $\Gamma = A \times A$, 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. □

Example 50. [A pushdown automaton with freshness] We describe an extension of orbit-finite pushdown automata, which is taken from a paper by

Murawski adn Tzevelekos []. Consider the equality atoms, although the example works for other atoms such as the total order atoms or the partial order atoms. Two objects x and y (which can be either atoms or sets with atoms) are called *fresh* with respect to each other if there exists disjoint sets S and T that support x and y, respectively. E.g. the atom $\underline{1}$ is fresh with respect to the set $\{\underline{2}, \underline{3}\}$, while the pair $(\underline{1}, \underline{2})$ is not fresh with respect to the pair $(\underline{2}, \underline{3})$. Consider a model of orbit-finite pushdown as defined in this section, with one new kind of transition:

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

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 precondition that a is fresh with respect to every letter on the stack and the current state q. If the precondition is not satisfied, then the transition is not enabled (we are working with a nondeterministic automaton).

We claim that emptiness is decidable for the extended model described above. The reason is that we can replace the new kind of transition by a transition with a weaker precondition: the letter a is required to be fresh only with respect to the current state and the topmost symbol on the stack. This weaker precondition can be expressed using the transitions of the basic model of orbit-finite pushdown automata. The automaton with the weaker precondition does not accept the same words, but it is nonempty if and only if the original automaton was nonempty. □

Example 51. [Modelling boolean recursive programs with atoms] Pushdown automata without atoms are sometimes used to model the behavior of recursive boolean programs. This modelling can be extended with atoms, which means that we can also model programs that have variables for atoms. Consider the total order atoms. Consider a recursive function such as the following one. (This program does not do anything smart.)

```
function f(a: atom)
begin
  b:=read() // read an atom from the input
  if b = a then
    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 sophisticted examples: many mutually recursive functions, boolean variables, other homogeneous datatypes for the atoms. As we shall see later on, many questions about pushdown automata remain decidable in sets with atoms, such as "does a nondeterministic orbit-finite pushdown automaton terminate on some input?" or "does a deterministic orbit-finite pushdown automaton terminate on all inputs?", or "can the stack reach arbitrarily large height?". This yields decidability for certain questions about the behavior of recursive programs. □

Example 52. [Higher-order pushdown automata have undecidable emptiness] We show that the algorithm described in Example 51 stops to work with higher-order pushdown automata, which are a natural model for higher-order recursive programs. A higher-order pushdown automaton is like a pushdown automaton, except that it can have stacks of stacks, or stacks of stacks of stacks, etc. We show that, in the presence of atoms, the model has undecidable emptiness already for order two (stacks of stacks).

To get undecidable emptiness, we do not use the full power of order two pushdown automata. All that we need is the following model. At any moment of the computation, the automaton has one or two stacks. Push and pop can only be executed on the later stack, i.e. if two stacks are available, then push and pop can only be executed on the second stack. If at some moment the automaton has only one stack, it can execute a "duplicate" command, which creates a second stack with the same contents as the first stack. This model is a special case of an order two pushdown automaton (where the second order stack is limited to height two), so it has decidable emptiness without atoms.

We now argue that the above model has undecidable emptiness in the presence of atoms. We only show that such an automaton can recognise

$$L = \{(w\#)^n : w \in \text{Atoms}^* \text{ has no repetitions and } n \in \mathbb{N}\}$$

over the alphabet $\text{Atoms} \cup \{\#\}$. 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.

One 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? □

9.2 Context-free grammars.

We now turn to context-free grammars. We hope that by now, the reader can fluently generalize any definition to sets with atoms, such as this one: an *orbit-finite context-free grammar* $\mathcal{G}$ consists of an orbit-finite input alphabet, an orbit-finite set of nonterminals $\mathcal{N}$, and a finitely supported set of productions

$$\mathcal{P} \subseteq \mathcal{N} \times (\mathcal{N} \cup A)^*.$$

Example 53. [Grammar for palindromes.] Consider the palindrome language P from Example 48. This language is generated by the following grammar. The grammar has one nonterminal N. The rules are

$$\begin{aligned} N &\to aNa && \text{for every } a \in A. \\ N &\to \epsilon \end{aligned}$$

□

Example 54. [Grammar for modified palindromes.] In the previous example, the grammar had just one nonterminal. Sometimes, it is useful to have an orbit-finite, but infinite, set of nonterminals. Consider the language P' from Example 49. For this language, we need different set of nonterminals, with a starting nonterminal N as well as one nonterminal N_a for every $a \in A$. The rules of the grammar are:

$$\begin{aligned} N &\to aN_aa && \text{for every } a \in A. \\ N_a &\to bN_ab && \text{for every } b \in A. \\ N_a &\to a && \text{for every } b \in A. \end{aligned}$$

One can show that the language P' cannot be recognized by a context-free grammar with atoms which has a finite set of nonterminals. Otherwise, the language P' would have the following property, which it does not have:

For every sufficiently long word $w \in P'$, there is a decomposition $w = w_1 w_2 w_3$, with w_2 and $w_1 w_3$ nonempty such that

$$w_1(\pi \cdot w_2)w_3 \in P' \qquad \text{for every } \pi \in G.$$

□

Equivalence of pushown automata and context-free grammars. In the examples so far, the languages P and P' could be recognized by orbit-finite pushdown automata, and generated by orbit-finite context-free grammars. This is no coincidence, since the two models are equivalent, as stated in the following exercise.

Exercise 21. Show that orbit-finite pushdown automata and orbit-fintie context-free grammars define the same language classes.

10 Turing machines with atoms

In this section, we talk about Turing machines[7]. An orbit-finite Turing machine is defined in the spirit as orbit-finite automata, or orbit-finite pushdown automata. The definition is the same as in a standard Turing machine, except that all components (states, input and work alphabets, transitions) are required to be orbit-finite sets with atoms. The tape is as usual: a sequence of cells, each storing a symbol of the work alphabet or a blank symbol. The head moves as usual: left, right, or stay. A configuration consists of the tape contents, the position of the head, and the control state. A one-step successor of a configurations is defined in the usual way, this successor need not be unique in machines that are not deterministic. A computation is a sequence of configurations, whose length is a normal natural number without atoms, hence one can talk about things like polynomial time computation. Maybe one could imagine a model where the structure of the tape is not discrete, but somehow orbit-finite, but we do not pursue this here.

Many of the standard results, such as equivalence of single-tape and multi-tape machines, work with the standard proofs. (In this section, we use single-tape machines.)

Example 55. [A Turing machine checking that all letters are different] Consider the equality atoms. Assume that the input alphabet is Atoms. We show a deterministic Turing machine which accepts words where all letters are distinct, and the atom $\underline{5}$ does not appear. (This is the language from Lemma 8.3, with the additional condition on $\underline{5}$ thrown in to illustrate non-equivariant machines.) The idea is that the machine iterates the following procedure until the tape is

[7] We skipped the context-sensitive level of the Chomsky hierarchy, because we have nothing interesting to say about it.

empty: if the first letter on the tape is different than $\underline{5}$, replace it by a blank and load it into the state, scan the word to check that the just-erased letter does not appear again, and go back to the beginning of the tape. □

Example 56. [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 sequence of numbers $i_1 \cdots i_n$ such that i_k is the first position where atom a_k appears. For instance, the deatomisation of $\underline{211291} \in \text{Atoms}^*$ is $122132 \in \{1,2,3\}^*$.

The point of deatomisation is that it stores all information about the input word. More precisely, two input words have the same deatomisation if and only if the are in the same equivariant orbit. Furthermore, it is not difficult to write a deterministic Turing machine, which transforms a sequence of atoms into its deatomisation (to achieve a finite output alphabet, the numbers in the deatomisation are encoded in binary, and separated by #). 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.

One can also show that, when the input alphabet is the atoms, this is the only thing possible: every equivariant Turing machine with atoms, deterministic or nondeterministic, can be converted into one which works as follows: first compute the deatomisation of the input, then remove the input and do the remaining work on the deatomisation.

□

Example 57. [A non-straight input alphabet] Consider the equality atoms. Let the input alphabet be sets of atoms of size at most ten, which is not a straight alphabet. 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 even number of positions in the input word.

We have actually applied something similar to the deatomisation technique from Example 56. In the input alphabet from this example, the deatomisation is defined as follows: we assign identifiers to equivalence classes (of the above-defined equivalence) and say how many elements belong to each equivalence class. This stores complete information, in the sense that two input words are in the same equivariant orbit if and only if they have the same deatomisation. □

Alternating machines. Recall that in an alternating machine, the control states are partitioned into existential and universal states. The semantics of the machine is defined via a game, which played by two players: the existential player and the universal player. The game begins in the initial configuration, with the input on the tape and the head over the first tape call in the initial state. In a given configuration, the player who owns the control state chooses the successor configuration. The existential player wins if the a configuration with an accepting state is reached, the universal player wins otherwise (the other cases are when the play is infinite, or reaches a configuration in a non-accepting state without successor configurations).

The machine accepts an input if the existential player has a winning strategy in the game described above. Formally speaking, a strategy is a tree whose nodes are labelled by configurations, such that nodes that use existential control states have one child with a successor configuration, and nodes that use universal control states have all possible successor configurations as children. To be winning for the existential player, all maximal paths must be finite and end in configurations with accepting states. Such a winning strategy is called a computation tree.

Exercise 22. Show that if an alternating machine has computation tree, then it has one where the set of nodes is orbit-finite, and the depth is bounded by a natural number.

Part III
Programming Languages

So far, we have not concentrated on algorithms. Typically, we would like to have algorithms for

1. Decide if a word in $w \in \text{Atoms}^*$ is a palindrome.
2. Decide if a nondeterministic orbit-finite automaton is nonempty.
3. Transform an orbit-finite pushdown automaton into a grammar.

Admittedly, we have already defined one notion of algorithm, namely Turing machines with atoms. This notion is good for the first problem on the list, deciding palindromes.

For the second and third problems, however, Turing machines are not the best choice. The problem is how to represent the input (also the output, in the third problem). A Turing machine with atoms has a fixed input alphabet, while both the second and third problems involve dealing with arbitrary alphabets, state spaces and so on. Therefore, in order to implement algorithms for the second and third problems on a Turing machine, we would need some form of representation for the input and output, which would use a single alphabet for all inputs and outputs. Also, it is difficult to imagine how the presence of atoms in such a fixed alphabet would be helpful to such a representation, so it seems that the generalised atom version of Turing machines would be useless in this context.

Of course the issue of representing inputs and outputs appears already without atoms. However the coding without atoms and orbit-finite sets is simple enough that it can usually be left implicit. For sets with atoms, the representation is much more involved.

Instead of dealing with the representation issues in every algorithm, we propose a programming language (actually, two programming languages: an imperative one, and a functional one), which deals directly with sets with atoms. The representation issues are tackled once, when designing the programming language; leaving the programmer free to concentrate on the issue at hand. 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 the three problems described above.

11 Imperative programming with atoms

In this section, we present an imperative programming language with atoms. The language extends while programs with two types: one for storing atoms and one for storing hereditarily orbit-finite sets. To deal with an orbit-finite set, the language has a loop construction, which is executed in parallel for all elements of an orbit-finite set.

11.1 Definition of the imperative language

The language is called *while programs with atoms*. The definition of the language depends on the choice of atoms (but not too much). We assume that the atoms are an effective oligomorphic structure, as in the assumptions of Theorem 5.4.

The datatype. We only have two datatypes in our language: `atom` and `set`. A variable of type `atom` stores an atom or is *undefined*. A variable of type `set` stores a hereditarily orbit-finite set (and not an atom). To have a minimal language, we encode other types inside `set`, using standard set-theoretic encodings. For instance, the booleans are encoded by

$$\texttt{false} \stackrel{\text{def}}{=} \emptyset \qquad \texttt{true} \stackrel{\text{def}}{=} \{\emptyset\}$$

and natural numbers are encoded by

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

We use such encodings to have a minimal language, at the cost of efficiency. Of course, in a real-life language, we would have real integer types with built in operations on them.

Syntax. The language contains the following constructions:

- **Constants.** There are infinitely many constants of type `atom`: one constant for every atom. (These constants depend on the choice of atom structure, e.g. there will be different constants for the equality atoms and different constants for the total order atoms.) There are constants $\emptyset$ and `Atoms` of type `set`, representing the empty set and the set of all atoms.
- **Expressions.** Expressions (which have values in the type `atom` or `set`), can be built out of variables and constants, using the following operations:
    1. Variables and constants are expressions. We assume that the types of the variables are declared in a designated preamble to the program; variables of type `atom` are initially undefined, while variables of type `set` are initially set to $\emptyset$.

2. For every symbol σ in the vocabulary of the atom structure, if σ has arity n and $\mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_n$ are expressions of type `atom`, then

$$\sigma(\mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_n)$$

is an expression which evaluates to `true` or `false` (such an expression is of type `set`). For instance, when the atom structure is the total order atoms, and `x` and `y` are variables of type atom, then $\mathtt{x} \leq \mathtt{y}$ is an expression (we write $\leq$ using infix style).

3. Comparisons $\mathtt{e} \in \mathtt{f}$ and $\mathtt{e} = \mathtt{f}$, which evaluate to `true` or `false`. In these comparisons, `e` and `f` can be either of type `atom` or `set`.
4. Union $\mathtt{e} \cup \mathtt{f}$, intersection $\mathtt{e} \cap \mathtt{f}$ and set difference $\mathtt{e} - \mathtt{f}$, for `e` and `f` of type `set`.
5. A unary singleton operation which adds one set bracket $\{\mathtt{e}\}$. Here, `e` can be either of type `atom` or `set`.
6. An operation which extracts the unique atom from a set:

$$\mathtt{theunique}(\mathtt{e}) = \begin{cases} \mathtt{f} & \text{when } \mathtt{e} = \{\mathtt{f}\} \text{ for some } \mathtt{f} \text{ of type } \mathtt{atom} \\ \mathtt{undefined} & \text{otherwise.} \end{cases}$$

- Values of expressions can be assigned to variables using the instruction $\mathtt{x} := \mathtt{e}$, provided that the types match.
- **Programming constructions.**
  1. A conditional `if e then I else J`. If the value of expression `e` is $\mathtt{true} \stackrel{\text{def}}{=} \{\emptyset\}$, then program `I` is executed, otherwise program `J` is executed.
  2. A (sequential) while loop `while e do I`, which executes the program `I`, while the value of expression `e` is `true`.
  3. A (parallel) for loop `for x in X do I`. Here `X` is an expression of type `set` and `x` is a variable of either type. The general idea is that the instruction `I` is executed, in parallel, with one thread for every element `x` (of appropriate type) of the set `X`. The question is: how are the results of the threads combined? We answer this question in more detail below.

Semantics. We now sketch a semantics for the language. In a given program, a finite number of variables is used. A state of the program is a valuation ν which assigns atoms to variables of type `atom` and hereditarily definable sets to variables of type `set`. Essentially, a valuation is a (finite length) tuple containing atoms and hereditarily definable sets, and therefore the set of all valuations is a legal set with atoms (but not orbit-finite, because of the values of variables of type `set`). A state of the program can be represented in a finite way when the atoms are an effective structure.

The semantics of a program `I` is a partial function, which maps a valuation ν of the free variables of `I` (before executing the program) to another valuation of the variables these free variables (after executing the program), denoted by $\mathtt{I}[\nu]$. The function is partial, because for some valuations, the program might not terminate. Given a valuation of the variables ν, we also define the running time, which intuitively stands for the maximal number of executed instructions.

The partial function $(\mathtt{I}, \nu) \mapsto \mathtt{I}[\nu]$ is going to be equivariant, because its definition (see below) does not refer to any particular atoms. In particular, for any given program `I`, the function $\nu \mapsto \mathtt{I}[\nu]$ is supported by the atoms that appear in the code of `I`.

We only explain the semantics for programs of the form

```
for x in X do I,
```

the other semantics are defined in the standard way. Suppose that we want to execute the program on a valuation ν. Two cases need be considered: when `x` is a variable of type `set` or when `x` is variable of type `atom`. The set $\nu(\mathtt{X})$ might store elements of both types `set` and `atom`. We say that an element $x \in \nu(\mathtt{X})$ is *appropriate* if it matches the type of the variable `x`. Define ν_x to be the same as ν, but with variable `x` set to x. For every appropriate $x \in \nu(\mathtt{X})$, execute the instruction `I` on the valuation ν_x. We require that

1. for every appropriate x, the program `I` terminates,

otherwise the `for` loop does not terminate. Assuming the above condition, for every appropriate x, there is a valuation $\mathtt{I}[\nu_x]$ after executing `I`. We now want to aggregate these valuations into a single valuation μ, which will be the result of executing the `for` program. To aggregate the valuations on variables of type `atom`, we require that

2. for every variable `y` of type `atom`, all valuations $\mathtt{I}[\nu_x]$ agree on `y`.

If the condition above fails the `for` loop does not terminate. Otherwise, we can define $\mu\mathtt{y}$ to be the unique value used on `y` by all valuations $\mathtt{I}[\nu_x]$. If `y` is a variable of type `atom`, then in order for $\mu(\mathtt{y})$ to be defined, we require that all valuations agree on the value of `y`, and value is the one used by μ. Set variables are aggregated using set union, i.e. a variable `y` of type `set` is assigned the value

$$\mu(\mathtt{y}) \stackrel{\text{def}}{=} \bigcup_{\text{appropriate } x \in \nu(\mathtt{X})} \mathtt{I}[\nu_x](\mathtt{y}). \tag{10}$$

We will later show that the above actually defines a valuation, i.e. the set above is hereditarily definable and can be computed. For this to be true, we place a final restriction on the `for` loop, namely

3. there is a finite upper bound on the running times of the program `I`, ranging over possible values of x.

Summing up, for the `for` loop to terminate, we require all three conditions 1,2,3 above.

The definition of the language and its semantics is now complete.

11.2 Executing the programs without using atoms

In this section, we show that the programs can be simulated without atoms, in the following way. To represent the code of a program, as well as a valuation of the variables, we use hereditarily definable sets.

In a memory state, each program variable contains a hereditarily definable set, which may be an atom in the case of variables of type atom. Recall that a hereditarily definable set consists of a set-builder expression plus a valuation of its free variables. Define a *parametrised memory state* to be like a memory state, except that it contains only the set-builder expressions without valuations of their free variables. We call the *parameters* of a parametrised memory state to be all the free variables of the set-builder expressions it contains. If ν is a parametrised memory state with parameters $\bar{x}$, and $\bar{a}$ is a tuple of atoms of same length as $\bar{x}$, then by applying $\bar{a}$ to all set-builder expressions in ν, we get an actual memory state, denoted by $\nu_{\bar{a}}$.

For instance, in the total order atoms, a parametrised memory state with parameters x, y and program variables $\mathtt{X}$ and $\mathtt{Y}$, could be

$$\mathtt{X} \mapsto x \qquad \mathtt{Y} \mapsto \{z : \text{ for } z \text{ such that } x < z < y\}$$

together with the valuation $x \mapsto 0, y \mapsto 2$. We define a *parametrised memory state* to be a Consider a tuple $\bar{x}$ of variables, which are meant to range over atoms.

definable sets such as but they are ampped

for program variables $\bar{\mathtt{X}} = \mathtt{X}_1, \ldots, \mathtt{X}_n$ and atom parameters $\bar{x} = x_1, \ldots, x_k$ to be a function which maps each variable $\mathtt{X}_i$ to a set-builder expression with free variables included in $\bar{x}$. Therefore, a valuation template is like a memory state used in the semantics of the programs, except that it is missing particular tuple of atoms $\bar{a}$ for the variables in $\bar{x}$. Once such a tuple $\bar{a}$ is provided, a valuation template ν turns into a memory state denoted by $\nu_{\bar{a}}$.

Theorem 11.1 *Assume that the atoms are effective, i.e. there is an enumeration of the atoms so that the truth value of first-order formulas can be decided. There is a normal program (without) atoms* $\mathtt{P}$*, which inputs:*

- *the code of a while program with atoms* $\mathtt{I}$*;*
- *a formula* φ *of first-order logic with free variables* $\bar{x}$*;*
- *a valuation template* ν *for* $\mathtt{I}$ *with parameters* $\bar{x}$*;*

and does the following:

- *if for some* $\bar{a}$ *satisfying* $\varphi(\bar{a})$*, the program* $\mathtt{I}$ *does not terminate when starting in* $\nu_{\bar{a}}$*, then also* $\mathtt{P}$ *does not terminate;*
- *otherwise,* $\mathtt{P}$ *terminates and produces a valuation template* μ *with parameters* $\bar{x}$*, such that for every* $\bar{a}$ *satisfying* $\varphi(\bar{a})$*, the result of executing* $\mathtt{I}$ *on* $\nu_{\bar{a}}$ *is* $\mu_{\bar{a}}$*.*

$$\mathtt{I}[\nu_{\bar{a}}] = \mu_{\bar{a}} \qquad \textit{for every } \bar{a} \textit{ satisfying } \varphi(\bar{a}).$$

The proof of the theorem is by induction on the size of the code of `I`.

Since the representations do not use atoms, they can be seen as standard bit strings, i.e. words over the alphabet $\{0, 1\}$. Therefore `P` can be modelled as a Turing machine, which inputs two bit strings and outputs a single bit string (and possibly does not terminate). Theorem 11.1 follows from the observation that the semantics can be evaluated on representations, using the operations on hereditarily orbit-finite sets given in Theorem 5.4. The only non-obvious step is making Lemma **??** effective. An inspection of the proof of that lemma shows that if the function f and the set X are hereditarily orbit-finite sets given by representations, then $\bigcup_{x \in X} f(x)$ is also hereditarily orbit-finite, and its representation can be computed.

11.3 Example programs

Before writing example programs, we introduce some syntactic sugar which makes programming easier.

Notational conventions Like in Python, we use indentation to distinguish blocks in programs. We write `{x,y}` instead of `{{x}∪{y}}`. 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. All the variables are assumed to be of type `set`.

```
function first(p)
  for a in p do
    for b in p do
      for x in a do
        for y in b 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 58. [The diagonal] As a warmup, we write a program that produces a specific set, namely

$$\{(a, a) : a \in \text{Atoms}\}.$$

The following program calculates this set in variable `X`.

```
for x in Atoms do X := X ∪ {(x,x)}
```

The same effect would be achieved by the following program.

```
for x in Atoms do X := {(x,x)}
```

□

Example 59. [Programs that use order on atoms] In the same spirit, we can produce sets that refer to some structure on the atoms.

Consider the total order atoms. Recall that there is an expression `x` $\leq$ `y` that says if the atom stored in variable `x` is smaller than the atom stored in variable `y`. For instance, the following program generates the growing triples of atoms.

```
for x in Atoms do
 for y in Atoms do
  for z in Atoms do
    if (x ≤ y) and (y ≤ z) then X := {(x,y,z)}
```

□

Example 60. [Program that produces closed intervals] Consider the total order atoms. The following program produces in variable `Y` the family of all closed intervals.

```
for (x,y) in Atoms do
   for z in Atoms do
      if (x ≤ z) and (z ≤ y) then X := X ∪ {z}
   Y := Y ∪ {X}
```

□

The following lemma generalizes the previous three examples.

Lemma 11.2 *Assume that the atoms are homogeneous and over a finite vocabulary. Every hereditarily orbit-finite set can be produced by some program.*

Proof
By Exercise 11, every hereditarily orbit-finite set can be defined by an expression as in the proof of Theorem 5.4, i.e. is of the form $\alpha(\bar{a})$ for some expression (whose syntax does not involve atoms), and some tuple of atoms $\bar{a}$ which gives values

to all free variables in α. By induction on the size of α, one can show that there is a program that inputs $\bar{a}$ and outputs the value of $\alpha(\bar{a})$. □

Example 61. [Programs with integer atoms do not work] We remark how the assumption on homogeneous atoms over a finite vocabulary is necessary. Suppose that the atoms are integers. We begin by converting the successor, which is given in the programming language as a relation, into a function. To do this, we first write a function `f(x)`, which maps `x` to $\{\texttt{x} + 1\}$:

```
function f(x)
  for y in Atoms do
      if succ(x,y) then
          ret:={z}
  return ret
```

Using the desingleton operation, we get the successor function $\texttt{x} \mapsto \texttt{x+1}$. Using a while loop, we can now write a program `natural(x)` which returns true if and only if `x` is a natural number. After evaluating the following program, the variable `X` stores the natural numbers, which is not an orbit-finite set.

```
y:=0
for x in ℤ do
  if natural(x) then X:=x
```

In a similar spirit, we could write a program which produces the set of prime numbers, or any decidable set of numbers, or even undecidable sets of numbers. This also means that the number of threads that is run in parallel is not orbit-finite. The reason for this problem is that the semantics strongly depends on oligomorphism. □

Example 62. [Reachability] We write 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. Why is the program still correct in the presence of atoms?

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 S-orbits. Let $X_1, X_2, \ldots, X_k$ denote the S-orbits of X. Therefore,

$$X = X_1 \cup X_2 \cup \cdots \cup X_k. \tag{11}$$

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 (11). In each iteration of the `repeat` we add some orbits, and therefore the loop can be iterated at most k times. □

Example 63. [Automaton emptiness] Following [2], we define an *orbit-finite nondeterministic automaton* the same way as a nondeterministic automaton, with the difference that all of the components (input alphabet, states, initial states, final states, transitions) are required to be hereditarily orbit-finite sets. Using reachability, it is straightforward to write an emptiness check for nondeterministic orbit-finite automata:

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

□

Example 64. [Monoid aperiodicity] An *orbit-finite monoid* is a monoid where the carrier is a hereditarily orbit-finite set, and the graph of the monoid operation is a finitely supported. (It follows that the graph of the monoid operation is a hereditarily orbit-finite set, because hereditarily orbit-finite sets are closed under finitely supported subsets.) Such a monoid is called *aperiodic* if for every element m of the monoid, there is a natural number n such that

$$m^n = m^{n+1}. \tag{12}$$

In [1] it was shown that an orbit-finite monoid is aperiodic if and only if all of the languages it recognises are definable in first-order logic. 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 tests the identity (12) for every element in the carrier.

```
function aperiodic (Carrier,Monop)
  for m in Carrier  do
    X:=∅
    new:=m
    repeat
       old:=new
```

```
      X:=X ∪ {old}
      new := Monop(old,m)
    until new ∈ X
    if new = old then ret := true else ret := false
  return ret
```

In the program above, the line `new := mult(old,m)` is actually syntactic sugar for a subroutine, which examines the graph of the multiplication operation `mult`, and finds the unique element `new` which satisfies `(old,m,new)` $\in$ `mult`.

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. This is shown in [1].

Finally, the program relies on the particular encoding of the booleans: `true` is the nonempty set $\{\emptyset\}$, while `false` is $\emptyset$. If the for loop sets `ret` to `true` for some `m` in the carrier of the monoid, then the whole program will return `true`, since the aggregation operation is union, which behaves like $\vee$ for booleans. $\square$

Example 65. [Automaton minimisation] The programs for automaton emptiness and monoid aperiodicity were for yes/no questions. We now present a program that minimizes deterministic automata. The program is a function

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

which inputs an orbit-finite deterministic automaton and returns the minimal automaton[8]. We assume that all states are reachable, the non-reachable states can be discarded using the emptiness procedure described above. We will also assume that all states are sets (and not atoms), so all variables in the code below are declared as `set`. 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.

```
for p in Q do
 for q in Q do
   for a in A do
      R := R ∪ {((delta(p,a),delta(q,a)),(p,q))}

base := (F × (Q-F)) ∪ ((Q-F) × F)
equiv := (Q × Q) - reach(R,base)
return ∅ = reach(R,q0) ∩ F
```

(The code above uses $\times$, which is implemented using `for`.) For the states of the minimal automaton, we need the equivalence classes of the relation `equiv`, which are produced by the following code.

[8]This program is particularly interesting when comparing the imperative programming language from this paper with the functional programming language from the next section. In that language, we are unable to correctly type a program for minimisation.

```
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. □

12 Functional programming with atoms

In the previous section, we gave an imperative style programming language. In this section, we discuss functional programming language in sets with atoms.

12.1 λ-calculus with orbit-finite sets

We begin by defining a core of the language, which is an extension of the typed λ-calculus, to which we add the ability to store atoms and orbit-finite sets of terms. We assume that the reader is familiar with the very basics of λ-calculus, such as the definition of terms, or the idea of β-reduction.

Terms. The set of terms is defined as usual (with variables, application and λ-abstraction) with the following new features. There are five constants $\emptyset$, `add`, `map`, `union` and `orbit`, as well as a constant for every atom. Furthermore, there is one new term constructor: every orbit-finite set of terms is also a term. Observe that orbit-finite sets of terms can be nested with the other constructions of λ-calculus, e.g. we can have an orbit-finite set of applications which involve orbit-finite sets of terms. This means that the syntax tree of a term has orbit-finite branching, like the syntax tree of a hereditarily orbit-finite set.

We adopt the convention that terms which are orbit-finite sets (call them *set terms*) are written using curly letters, such as $\mathcal{M}$ and $\mathcal{N}$, while the other terms are written using normal letters, such as M and N.

Representing terms. Assume that every variable is encoded as a hereditarily orbit-finite set, e.g. the variables are natural numbers

$$x_0 \stackrel{\text{def}}{=} \emptyset \qquad x_1 \stackrel{\text{def}}{=} \{\emptyset\} \qquad x_2 \stackrel{\text{def}}{=} \{\emptyset, \{\emptyset\}\} \qquad \cdots .$$

Assume that the constructors of the calculus (application, λ-abstraction, and orbit-finite sets) are encoded unambiguously in the language of set theory in some way, e.g. using Kuratowski pairing. Under any natural encoding, the set corresponding to a term is a hereditarily orbit-finite set. Therefore, we can treat terms as special cases of hereditarily orbit-finite sets, and use Theorem 5.4 to represent terms.

β-reduction. Computation in λ-calculus is performed by β-reduction. We keep the standard β-reduction rule of λ-calculus:

$$(\lambda x.M)N \quad \to_\beta \quad M[x := N],$$

where $M[x := N]$ is a capture-avoiding substitution. To manipulate terms with sets, we have the following constants:

- $\emptyset$ represents the empty set of terms.
- `add` adds one term to a set of terms, as defined by the β-reduction rule

  $$\texttt{add}\ M\ \mathcal{N} \qquad \to_\beta \qquad \mathcal{N} \cup \{M\},$$

  which can only be applied when $\mathcal{N}$ is a set term.
- `map` applies one term to all elements of a set of terms, as defined by the β-reduction rule

  $$\texttt{map}\ M\ \mathcal{N} \qquad \to_\beta \qquad \{MN : N \in \mathcal{N}\}.$$

  which can only be applied when $\mathcal{N}$ is a set term.
- `union` returns the union of a set, as defined by the β-reduction rule

  $$\texttt{union}\ \mathcal{M} \qquad \to_\beta \qquad \bigcup \mathcal{M} = \{N : N \in \mathcal{N} \text{ for some } \mathcal{N} \in \mathcal{M}\},$$

  which can only be applied when $\mathcal{M}$ is a set term, which contains set terms.
- `orbit` returns the orbit of a term with respect to a given support, as defined by the β-reduction rule

  $$\texttt{orbit}\ \{a_1, \ldots, a_n\}\ M \qquad \to_\beta \qquad \{\pi(M) : \pi \text{ is a } (a_1, \ldots, a_n)\text{-automorphism}\},$$

  which can only be applied when the first argument is a finite set of atoms. Observe that the notion of an $(a_1, \ldots, a_n)$-automorphism does not depend on the order of the sequence $(a_1, \ldots, a_n)$, so the reduction rule is well defined.

Finally, $\to_\beta$ needs some structural rules, which say how to apply β-reduction inside subterms. These structural rules are obvious for all term constructors, except for the set term. For set terms, the structural induction rule is defined as follows.

- Suppose that $\mathcal{M}$ is an orbit-finite set of terms. A *reduction mapping* is an function f, which is not the identity, whose domain contains $\mathcal{M}$, and which maps every term $M \in \mathcal{M}$ to a term $f(M)$ such that

  $$M = f(M) \qquad \text{or} \qquad M \to_\beta f(M).$$

  If f is a reduction mapping, then $\mathcal{M} \to_\beta f(\mathcal{M})$.

We allow the reduction mapping to map some terms to themselves, because we want to present β-reductions in smaller steps that are easier to understand. The system would be equivalent if we would require reduction mappings to map every term not in a normal form (i.e. every term which admits some β-reduction) to a different term.

Example 66. [Reduction mapping needs to be equivariant] The assumption that the reduction mapping is equivariant in the structural rule for set terms might seem unnatural. One could imagine a more relaxed requirement, with finitely supported reduction mappings. This would lead to two problems.

The first problem is that, with finitely supported reduction mappings, a step of β-reduction could introduce atoms that were not present in a term before. As an example of this phenomenon, consider the equality atoms. Choose some term with an atom that can β-reduce in two different ways, say

$$\underbrace{a((\lambda x.x)a)}_{N_a} \quad {}_\beta\!\leftarrow \quad \underbrace{((\lambda x.x)a)((\lambda x.x)a)}_{M_a} \quad \rightarrow_\beta \quad \underbrace{((\lambda x.x)a)a}_{K_a}.$$

Consider the set term $\{M_a : a \in \text{Atoms}\}$. One could use a reduction mapping that reduces $M_{\underline{1}}$ to $N_{\underline{1}}$, but reduces the remaining terms M_a to K_a. Under this reduction mapping, the equivariant term $\{M_a : a \in \text{Atoms}\}$ β-reduces to

$$\{N_{\underline{1}}\} \cup \{K_a : a \in \text{Atoms} - \{\underline{1}\}\},$$

which is not an equivariant term, because it needs $\underline{1}$ in its support.

The second problem is that finitely supported reduction mappings can lead to infinite sequences of reductions[9]. Consider the set term

$$\{(\lambda x.x)a : a \in \text{Atoms}\}.$$

Under a reduction mapping which β-reduces only the term with atom $\underline{1}$, we get

$$\{(\lambda x.x)a : a \in \text{Atoms} - \{\underline{1}\}\} \cup \{\underline{1}\}.$$

We can repeat this process, applying the β-reduction rule only to the term with atom $\underline{2}$, yielding

$$\{(\lambda x.x)a : a \in \text{Atoms} - \{\underline{1}, \underline{2}\}\} \cup \{\underline{1}, \underline{2}\}.$$

This process can be repeated indefinitely, yielding an infinite sequence of β-reductions. □

As in the section on imperative programming, we need to show that the calculus is well defined, in the sense that applying a β-reduction rule to a term yields a term. In other words, we have to argue that all sets stay orbit-finite

[9]The second problem goes away if require reduction mappings to map every term to a different term.

after a β-reduction rule is applied. There are two possible problems: the `union` term, and the `map` term. As shown in the previous section, the `union` term cannot create an orbit-infinite set, thanks to the assumption that the atoms are oligomorphic. For the `map` term, we need the following property: if X is an orbit-finite set, and x is any element, then $\{x\} \times X$ is orbit finite. This property, is a weaker version of closure of orbit-finite sets under product, and therefore it holds under the assumption that the atoms are oligmorphic. The property fails in the integer atoms, e.g. $\{2\} \times \mathbb{Z}$ is a set that is not orbit-finite.

Types. We now describe a type system for the language. The types of the language are the same as in the standard of λ-calculus, with two addition of two typing constructors, P_{of} and P_{fin}. A term of type $\mathsf{P}_{\text{of}}\alpha$ represents an orbit-finite set of terms of type α, while a term of type $\mathsf{P}_{\text{fin}}\alpha$ represents a finite set of terms of type α.

The typing rules are inherited form the standard λ-calculus, with the operations $\emptyset$, `add` and `map` treated as polymorphic constants, which can assume the following types for every type α:

$$\begin{array}{rcl} \emptyset & : & \mathsf{P}_{\text{of}}\alpha \\ \texttt{add} & : & \mathsf{P}_{\text{of}}\alpha \to \alpha \to \mathsf{P}_{\text{of}}\alpha \\ \texttt{map} & : & (\alpha \to \beta) \to \mathsf{P}_{\text{of}}\alpha \to \mathsf{P}_{\text{of}}\beta \\ \texttt{union} & : & \mathsf{P}_{\text{of}}\mathsf{P}_{\text{of}}\alpha \to \mathsf{P}_{\text{of}}\alpha \end{array}$$

The same typing rules as above, but with P_{of} replaced by P_{fin} are also allowed. The only place in the language where there is a difference between P_{of} and P_{fin} is the `orbit` term, which has type:

$$\texttt{orbit} : \mathsf{P}_{\text{fin}}\text{Atoms} \to \alpha \to \mathsf{P}_{\text{of}}\alpha.$$

Since the `orbit` term only needs P_{fin} applied to atoms, we would get an equivalent language if we removed the P_{fin} type constructor, and added a type for finite sets of atoms.

Confluence and weak normalisation. A term is said to be *in normal* form if it cannot β-reduce to a different term. Confluence and weak normalisation are two basic properties of β-reduction: weak normalisation says that terms have normal forms, and confluence says that these normal forms are unique. We show that these properties hold for our extension of the λ-calculus.

Proposition 12.1 *The reduction relation $\to_\beta$ is confluent. On typed terms it is weakly normalising.*

Proof
In the proof, by reduces we mean "reduces in possibly several steps".

We begin with confluence. The definition of confluence is that for every term M which can be reduce to terms N_1 and N_2, there exists a term N such that

both N_1 and N_2 can be reduced to N. A reduction relation is said to be weakly normalising on a term M if the term can be reduced (possibly, in many steps) to a term in normal form (a term where no reductions can be applied). $\square$

In λ-calculus without atoms, strong normalisation says that there is no infinite sequence of β-reductions:

$$M_1 \to_\beta M_2 \to_\beta M_3 \to_\beta \cdots .$$

A classical result (citation needed) says that simply typed terms have strong normalisation. This is not true in λ-calculus with atoms.

In the presence of atoms, as we have observed, there can be

Theorem 12.2 *There is a normal program (without) atoms, which inputs a typed term represented using the data structures of Theorem 5.4, and outputs a representation of its normal form.*

12.2 Example programs

The previous section discussed how orbit-finite sets can be added to a functional programming language, one the example of simply typed λ-calculus without recursion. In this section we show examples of functional programming with orbit-finite sets. For the examples, we use a full ML style programming language, with the same extensions as in the previous section, i.e. type constructors P_{of} and P_{fin}, together with the terms $\emptyset$, `add`, `union` and `orbit`.

```
binaryunion x y = union (add x (add y ∅))
```

Theorem 12.3 *ss*

Part IV
The Church-Turing Thesis

13 Computability modulo representations

What is a decidable language, or a computable function, in the presence of atoms? Are languages decided by Turing machines, as defined in Section 10, all languages that "should" be decidable? What about functions computed by while programs with atoms, or functional programs with atoms? In other words, what is the Church-Turing thesis with atoms?

Computability modulo representations. One approach to this question is to represent sets with atoms by bit strings, and use the notion of computability over bit strings. On bit strings, the notion of computability is very robust, and it does not matter if one uses deterministic, nondeterministic, or alternating Turing machines, and the Church-Turing thesis says that all other models give the same notion as well. To fix notation, we say that a partial function (from words over some input alphabet to words over some output alphabet, both without atoms) is *computable* if there is a Turing machine without atoms which takes finite time to output the result on arguments where the function is defined, and which does not terminate on arguments where the function is undefined.

Suppose that ρ is a surjective function from bit strings to hereditarily orbit-finite sets, which is a representation as discussed at the end of Section 5.1. Suppose that f is a partial function which transforms hereditarily orbit-finite sets into hereditarily orbit-finite sets. We say that f is *computable modulo* ρ if there is some partial computable function f' from bit strings to bit strings such that the following diagram commutes:

$$\begin{array}{ccc} \text{bit strings} & \xrightarrow{f'} & \text{bit strings} \\ \downarrow\rho & & \downarrow\rho \\ \text{hof sets} & \xrightarrow{f} & \text{hof sets} \end{array}$$

We say that f is *computable modulo representations* if it is computable modulo ρ for some representation function ρ. As the following lemma shows, in the definition, we could replace "some representation" by "every representation".

Lemma 13.1 *If ρ_1 and ρ_2 are different representations, then a function f is computable modulo ρ_1 if and only if it is computable modulo ρ_2.*

Proof
Suppose that $\bar{a}$ is a support of f. By Theorem 5.6, if ρ_1 and ρ_2 are different representations, then we can find a function τ from bit-strings to bit-strings,

such that for some atom automorphism, the following diagram commutes:

$$\begin{array}{ccc} \text{bit strings} & \xrightarrow{\tau} & \text{bit strings} \\ \downarrow{\scriptstyle \rho_1} & & \downarrow{\scriptstyle \rho_2} \\ \text{hof sets} & \xrightarrow{\pi} & \text{hof sets} \end{array}$$

By inspecting the proof of Theorem 5.6, one can ensure that τ is surjective, and that π is a $\bar{a}$-automorphism. Since τ is surjective, there is a computable function τ^{-1} such that $\tau^{-1} \circ \tau$ is the identity. The result follows by composing diagrams:

$$\begin{array}{ccccccc} \text{bit strings} & \xrightarrow{\tau} & \text{bit strings} & \xrightarrow{f'} & \text{bit strings} & \xrightarrow{\tau^{-1}} & \text{bit strings} \\ \downarrow{\scriptstyle \rho_2} & & \downarrow{\scriptstyle \rho_1} & & \downarrow{\scriptstyle \rho_1} & & \downarrow{\scriptstyle \rho_1} \\ \text{hof sets} & \xrightarrow{\pi} & \text{hof sets} & \xrightarrow{f} & \text{hof sets} & \xrightarrow{\pi^{-1}} & \text{hof sets} \end{array}$$

□

13.1 Turing machines

We begin by showing that alternating Turing machines characterise exactly those languages which are computable modulo representations. To formalise this, we interpret a language as a characteristic function. There are two ways to do this, suitable for modelling semi-decidability and decidability, respectively:

- the *partial characteristic function of* L maps words from L to 1 and is undefined on other words;
- the *complete characteristic function of* L maps words from L to 1 and maps other words to 0.

The notion of computability modulo representations can be applied to these functions, because if the input alphabet A is hereditarily orbit-finite, then every word in A^* is a hereditarily orbit-finite set (recall how finite sequences are encoded as sets). The following theorem characterises semi-decidability.

Theorem 13.2 *Assume that the atoms are effectively oligomorphic and over a finite vocabulary. Let A be a hereditarily orbit-finite alphabet, and $L \subseteq A^*$ a finitely supported language. Then the following conditions are equivalent:*

1. *The partial characteristic function is computable modulo representations.*
2. *The language is recognised by an alternating Turing machine with atoms.*

If furthermore the atoms are homogeneous, then the items above are also equivalent to:

3. *The language is recognised by a nondeterministic Turing machine with atoms.*

The theorem implies an analogous characterisation for decidable languages: the complete characteristic function is computable modulo representations if and only if L and its complement are recognised by alternating Turing machines with atoms. As we will show in Section 15, deterministic machines are strictly weaker than nondeterministic ones, already in the case of atoms with equality only.

The following example shows that when the vocabulary of the atoms is infinite, then there are languages with decidable representations which are not recognised by any Turing machine.

Example 67. [Problems with Turing decidability for atoms with an infinite vocabulary] Consider an infinite vocabulary, with one n-ary relation R_n for every possible arity n. Consider the class of all finite structures over this vocabulary, with the restriction that for every n, the relation R_n is interpreted as some family of sets of size n, i.e. it only selects tuples of n distinct elements, and is closed under permuting coordinates. This class of structures is closed under amalgamation, and therefore it has a Fraïssé limit. Call this limit the *homogeneous hypergraph*. For every n, the homogeneous hypergraph has finitely many non-equivalent quantifier-free formulas in n variables, because it does not make sense to use the relations R_m with $m > n$. Therefore, the homogeneous hypergraph is oligomorphic; one can also show that it is effectively oligomorphic. Therefore Theorem 5.4 can be applied to the homogeneous hypergraph. Consider now the following language over the alphabet of the atoms:

$$L = \{a_1 \cdots a_n : a_1, \ldots, a_n \text{ are atoms with } R_n(a_1, \ldots, a_n)\}.$$

It is easy to see that $\mathrm{rep}(L)$ is a decidable language without atoms. However, the language L itself, is not recognised by any Turing machine, be it deterministic, nondeterministic, or even alternating. The intuitive reason is that in each step, a Turing machine makes its decision based only on its control state and the tape symbol under the head. These symbols can store only a bounded number of atoms, and therefore the information about relations R_n of large arity is unavailable. A more precise proof is left as an exercise to the reader. □

Implication from 2) to 1) A configuration of a Turing machine (contents of the tape, sate, head position) is also a hereditarily orbit-finite set. The toolkit from Theorem 5.4 allows us to algorithmically manipulate (encodings of) configurations, e.g. answering questions like: how many cells are used? what is the content of sixth cell on the tape? Therefore, the following question is decidable without atoms for a given Turing machine with atoms: given representations of configurations c and d, decide if the machine go in one step from c to d? A computation tree of an alternating machine, by Exercise 22, can be assumed to be a hereditarily orbit-finite set. Again, the computation trees can also be manipulated using the tookit, and therefore given a representation, one can decide if it represents a computation tree. Therefore, a Turing machine can search for a representation of a computation tree.

Implication from 1) to 2) A formula of first-order logic in the vocabulary of the atoms is said to *define the orbit* of a tuple of atoms $\bar{a}$ if it has one free variable for every coordinate in the tuple, and the formula is true in exactly the tuples in the same equivariant orbit as $\bar{a}$. By Lemma 4.4, for every tuple there is a formula which describes its orbit. Such a formula can be written down using a finite alphabet, say bit strings, and therefore can be input by a machine.

Lemma 13.3 *There is an alternating Turing machine which inputs a tuple of atoms and a formula of first-order logic, and accepts if and only if the formula defines the orbit of the tuple. If furthermore the atoms are homogeneous, then a deterministic machine is enough.*

Proof
By the assumption on effective oligomorphism, we can compute if a formula of first-order logic defines an orbit, i.e. it does not admit two satisfying valuations in different orbits. This is a property only of the formula, so atoms are not needed for this part of the computation. Once we have checked that the formula defines an orbit, we need to evaluate it on the tuple of atoms given on the input.

If the atoms are not homogeneous, then we use alternation to simply evaluate the formula, with the existential player choosing existentially quantified atoms and the universal player choosing universally quantified atoms.

If the atoms are furthermore homogeneous, then every formula is effectively equivalent to a quantifier-free one. Therefore, we can compute a quantifier-free version of the formula, and then deterministically check if it holds in the input tuple. If the quantifier-free formula contains functions, the deterministic machine will need to compute the values of these functions; however, due to finiteness of the vocabulary each function can be hard-coded into the machine. □

Lemma 13.4 *Let $\bar{a}$ be a finite tuple of atoms, and A an $\bar{a}$-supported hereditarily orbit-finite alphabet. There is an alternating automaton recognising the language*

$$\{w\#d : w \in A^* \text{ and } d \text{ is a data structure representing the } \bar{a}\text{-orbit of } w\}.$$

If furthermore, the atoms are homogeneous over some finite vocabulary (possibly including functions), then a nondeterministic machine is enough.

Proof
Every hereditarily orbit-finite set A is a surjective image of tuples of atoms. This means that there is some $k \in \mathbb{N}$ and a surjective function

$$\alpha : \text{Atoms}^k \to A.$$

Let $\alpha^{(n)}$ denote the coordinate-wise lifting of this function to $k \cdot n$ tuples of atoms, whose range is A^n. Given n, one can compute a representation of $\alpha^{(n)}$, whose graph is a hereditarily orbit-finite set.

We now describe the machine from the statement of the lemma. Given on input a word $w \in A^*$ of length n, the machine uses nondeterminism to guess word v over the atoms such that $\alpha^{(n)}(v) = w$, and stores v on the tape. Using Lemma 13.3, we can compute a formula φ of first-order logic such that $\varphi(\bar{a}u)$ holds if and only if u is in the same $\bar{a}$-orbit as v. This yields a data structure representing the $\bar{a}$-orbit of v. Using Exercise 8, we can compute the image of this orbit under $\alpha^{(n)}$; this image is the $\bar{a}$-orbit of the input word w. □

Proof (of implication 1) to 2) in Theorem **??**)
Let $L \subseteq A^*$ be a language with atoms such that $\mathrm{rep}(L)$ is recognised by a normal Turing machine M without atoms. Suppose that $\bar{a}$ supports the language L and its input alphabet. The alternating Turing machine recognising L inputs $w \in A^*$, and uses Lemma 13.4 to compute a representation of the $\bar{a}$-orbit of w. It then uses 5 of Theorem 5.4 to get a representation of some word v in this $\bar{a}$-orbit, and runs the machine M on v. The result of M is the same on w and v, because L is $\bar{a}$-supported. □

13.2 Imperative and functional programs

We say that a family X of hereditarily orbit-finite sets is computed by a while program with atoms if there is a while program with atoms which inputs a hereditarily orbit-finite set and outputs yes/no depending on whether the set belongs to X. The family X itself need not be hereditarily orbit-finite, e.g. X could be all hereditarily orbit-finite sets that represent natural numbers (in particular, they use no atoms).

Theorem 13.5 *Assume that the atoms are effectively oligomorphic. Then for a partial function f from hereditarily orbit-finite sets to hereditarily orbit-finite sets, the following conditions are equivalent:*

1. *f is finitely supported and computable modulo representations;*
2. *f is computed by some functional program with atoms;*
3. *f is computed by some while program with atoms.*

The implication from 1) to 2) follows from Theorem 11.1. We now prove the converse implication. The general idea is similar to the one in Theorem **??**, which was the Church-Turing thesis for Turing machines: namely a while program can transform a hereditarily orbit-finite set into a data structure representing it, possibly with a little loss of precision.

Recall that in Theorem 5.4, a hereditarily orbit-finite set or atom was represented by an expression α together with a valuation $\bar{a}$ of its free variables. The best we could hope for would be a program that inputs a set x and outputs an expression α and a tuple of atoms $\bar{a}$ such that $x = \alpha(\bar{a})$. Unfortunately, this might be impossible in most cases. The reason is that the tuple $\bar{a}$ is necessarily a support of x, and for many atom structures there is no finitely supported

function which maps a sets to tuples supporting them. For instance, this is the case in the equality atoms: given on input an unordered set $\{a, b\}$, computing a representation of the form $\alpha(\bar{a})$ would require imposing an order on the set $\{a, b\}$, which is impossible.

When proving Theorem **??**, we will therefore need to use a different representation. Suppose that α is an expression with n free variables, and A is a set of n-tuples of atoms. Let x be an atom or hereditarily orbit-finite set. We say that (α, A) is a *weak representation* of x if A is nonempty and for every tuple $\bar{a} \in A$, x is equal to $\alpha(\bar{a})$. A weak representation is itself a hereditarily orbit-finite set, and therefore weak representations can be input and output by while programs with atoms.

Lemma 13.6 *There is a while program with atoms which inputs n and a set $A \subseteq \text{Atoms}^n$, and outputs some k and a first-order formula φ in $n+k$ variables such that*

$$A = \{\bar{a} : \varphi(\bar{a}\bar{b})\} \qquad \text{for some } \bar{b} \in \text{Atoms}^k.$$

Proof
By enumerating all possible first-order formulas. □

Lemma 13.7 *There is a while program with atoms which inputs a hereditarily orbit-finite set or atom and outputs some weak representation of it.*

Proof
The program works by induction on the nesting depth of the input set. When it inputs an atom a, it outputs a weak description $(x, \{x \mapsto a\})$. Suppose that the input is a set X. By recursively doing calling the program for simpler sets, and then doing a `for` loop, we can compute a family

$$\{(\alpha_x, A_x)\}_{x \in X}$$

such that for every $x \in X$, (α_x, A_x) is a weak description of x. There are finitely many possible values of α_x, since these are objects with atoms. Furthermore, expressions α admit a computable total ordering, because they can be represented as bit strings, and therefore one can compute a list $\alpha_1, \ldots, \alpha_n$ of all possible values of α_x. It is not difficult to see that X is equal to

$$\bigcup_{i \in \{1,\ldots,n\}} \{\alpha_i(\bar{a}) : \text{for some } x \in X,\ \alpha_x = \alpha_i \text{ and } \bar{a} \in A_x\}.$$

From weak descrpitions of a finite number of sets one can compute a weak descrpition of their union. Therefore, it suffices to show that for every $i \in \{1, \ldots, n\}$, we can compute a weak description of the set

$$X_i \stackrel{\text{def}}{=} \{\alpha_i(\bar{a}) : \bar{a} \in B_i\} \quad \text{where} \quad B_i \stackrel{\text{def}}{=} \{\bar{a} : \text{for some } x \in X,\ \alpha_x = \alpha_i \text{ and } \bar{a} \in A_x\}.$$

The set B_i is a set of tuples of atoms of the same dimension, namely the number of free variables in α_i, call this dimension n_i. The set B_i can be computed based on i. By Lemma 13.6, we can compute number k_i and a formula φ_i with $n_i + k_i$ free variables such that

$$B_i = \{\bar{b} : \varphi_i(\bar{b}\bar{c})\} \qquad \text{for some } \bar{c} \in \text{Atoms}^{k_i}.$$

Define C_i to be the set of tuples $\bar{c}$ which make the above equality true. Once φ_i is known, the set C_i can be computed using a `for` loop ranging over all k_i-tuples of atoms. We now define the weak expression for the set X_i. Let $\bar{x}$ be a tuple of variables of the same length as $\bar{a}$, and let $\bar{y}$ be a tuple of variables of the same length as $\bar{y}$. Consider the expression

$$\beta_i \stackrel{\text{def}}{=} \{\alpha_i(\bar{x}) : \text{ for } \bar{x} \text{ such that } \varphi_i(\bar{x}\bar{y})\}.$$

We have shown that for every $\bar{c} \in C_i$, the expression $\beta_i(\bar{c})$ is equal to X_i. Therefore, (β_i, C_i) is a weak description of X_i.

□

Lemma 13.8 *Suppose that X is a family of hereditarily orbit-finite sets which is supported by a k-tuple of atoms $\bar{a}$, and such that* $\text{rep}(X)$ *is decidable. For every expression α with n free variables, one can compute a first-order formula φ_α with $n + k$ variables such that*

$$\alpha(\bar{b}) \in X \quad \textit{iff} \quad \varphi_\alpha(\bar{a}\bar{b}) \qquad \textit{for every } n\textit{-tuple of atoms } \bar{b}$$

Suppose that we are given a hereditarily orbit-fintie set x. Using Lemma 13.7, we compute a weak description (α, B) for x. Using Lemma 13.8, we can compute a formula φ_α such that x belongs to X if and only if there is some $\bar{b} \in B$ such that $\varphi_\alpha(\bar{a}\bar{b})$ holds. The latter condition can be tested by a while program with atoms.

Suppose that α is an expression as in Theorem 5.4 and X

14 Least supports (planned)

15 Deterministic Turing machines

In this section we show that, already with the equality atoms, deterministic and nondeterministic Turing machines are not equivalent. What is more, already nondeterministic polynomial time is not included in deterministic semidecidable. This illustrates the weakness of the deterministic model.

Theorem 15.1 *Consider the equality atoms. There is a language that is decidable by a nondeterministic polynomial time Turing machine, but not decidable by any deterministic Turing machine (without any time bounds).*

A consequence of the theorem is that, with atoms, P is not equal to NP. However the theorem is too strong to make the P vs NP connection interesting: it shows that computation with atoms is so different from computation without atoms, that results on the power of nondeterminism in the presence of atoms are unlikely to shed new light on the power of nondeterminism without atoms.

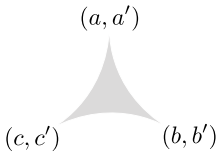
The rest of Section 15 is devoted to proving Theorem 15.1. In Section 15.0.1, we define a language L that witnesses the difference between NP and deterministic decidability; we also show that L is in NP. Then we prove that L is not deterministically semi-decidable. The proof is in two steps. In Section 15.0.2, we define orbit-finite algebras and show that every deterministic Turing machine can be simulated by an orbit-finite algebra. In Section 15.0.3 we show that no orbit-finite algebra can recognise L.

15.0.1 The language

The input alphabet We begin by defining the input alphabet. We use the name *straight triangle* for an six-tuple of distinct atoms. We think of a straight triangle as three pairs of atoms:

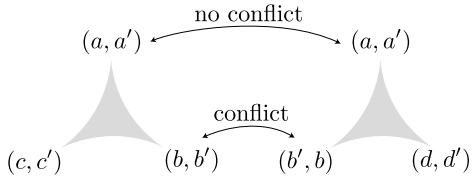
$$(a, a'), (b, b'), (c, c')$$

We define the *side sets* of the straight triangle to be the unordered pairs $\{a, a'\}$, $\{b, b'\}$ and $\{c, c'\}$. We denote straight triangles by τ. The set of all triangles is a one-orbit set. We visualise a straight triangle as a hyperedge that connects orientations of its side sets:



The hyperedge is ordered (i.e. there is a notion of first, second, and third pair of atoms), but this will not play a role in the construction.

Suppose that we have several straight triangles. A *conflict* is a side set which appears in two triangles in different orientations. For instance, the following two straight triangles have one conflict:



Straight triangles $\tau_1, \ldots, \tau_n$ are called *nonconflicting* if they have no conflicts. Consider a triangle which has a side set $\{a, a'\}$. The *swap on* $\{a, a'\}$ changes the orientation of the side, i.e. changes (a, a') to (a', a) and vice versa. Note

that doing a swap (on any side set) does not change the set of side sets. Swaps are a way of resolving conflicts. For instance, in the picture above, doing a swap on the side set $\{b, b'\}$ in the left (or right, but not both) triangle will remove the conflict.

We say that two straight triangles are *parity equivalent* if one can go from one to the other by an even number of swaps (i.e. swapping zero or two side sets). We use the name *parity triangle* for a parity equivalence class of straight triangles; each such a class contains exactly four straight triangles (these are called *straightenings* of the parity triangle). Doing a single swap changes the parity equivalence class of a straight triangle, doing another swap comes back to the original class. Therefore, when the side sets are fixed, there are exactly two parity triangles with these side sets. These two parity triangles are said to be *dual*, and changing a parity triangle to its dual is called *flipping*. Note that the set of all parity triangles is a one-orbit set.

The language We now define a language that witnesses Theorem 15.1. The input alphabet is parity triangles. A sequence of parity triangles belongs to the language if they can be straightened so that there are no conflicts.

Observe that membership in the language does not depend on the order or repetition of letters, and therefore it makes sense to talk about a set of parity triangles belonging to the language.

We will show that the language is a witness for Theorem 15.1: it is in NP but not deterministically decidable. Membership in NP is straightforward: the machine has a work alphabet that contains straight triangles, and uses nondeterminism to straighten the input. Once the input has been straightened, the conflicts can be counted in deterministic polynomial time. Each parity triangle can be straightened in four ways, so there is a finite (exponential) number of nondterministic choices. However, a deterministic Turing machine cannot go through all these choices one by one, since it would need some canonical way of ordering them, which turns out to be impossible.

Remark: The language L is a variant of the Cai-Fürer-Immerman (CFI) construction [6] from descriptive complexity theory. There, it is used to show that a certain logic $\mathcal{C}^{\omega}_{\infty\omega}$ cannot express a property of (unordered) graphs which is, however, decidable in polynomial time. That result can also be deduced from Theorem 15.3 below.

15.0.2 Algebras as a model of local computation

The reason why a deterministic Turing machine cannot recognise the language L is that it has only a local view of the computation: the decision for the next step is taken based on the state of the machine, and one cell of the tape. In particular, the decision depends only on the small set of atoms that is found in the state and one cell; the size of this set is fixed by the machine, and does not depend on the input. Our proof will show that any computation model of this kind will not recognise the language L. To model locally based decisions, we

use the notion of algebras and terms (similar to circuits). Terms in an algebra are evaluated in a local fashion: the result of a bigger term depends only on a single operation applied to its subterms. By using terms and algebra, our proof will not need to depend on the technical details of Turing machines such as end-of-tape markers, the position of the head, etc.

An *orbit-finite algebra* consists of:

- an orbit-finite *universe* A,
- a finite set of finitely supported *operations* of finite arity:

$$f_1 : A^{n_1} \to A \qquad , \ldots , \qquad f_k : A^{n_k} \to A.$$

We require the set of operations to be finite, although an orbit-finite set of operations would also be natural. We use algebras only as a technical tool, and we choose a truly finite set of operations for technical convenience.

A term in an algebra is defined as usual: it is a finite tree where internal nodes are operations, and the leaves are variables or constant operations. Given a term ϕ, and a valuation *val* which maps its variables to the universe of the algebra, we write $\phi[val] \in A$ for the value of the term under the valuation.

If there is some implicit order $x_1, \ldots, x_n$ on the variables of a term, then we can also evaluate the term in a word $w \in A^n$, by using the valuation that maps the i-th variable to the i-th position. We denote this value by $\phi[w]$.

Recognising a language We now define what it means for an algebra to recognise a language $L \subseteq A^*$. To input a word from A^*, we require the universe of the algebra to contain the input alphabet, but it can also contain some other elements, which can be seen as a work alphabet. Finally, we require the universe to contain the Boolean values *true* and *false*, so that it can say when a word belongs to the language. We say that such an algebra (non-uniformly) recognises L if for every input length n there is a term ϕ_n with n variables such that

$$\phi_n[w] = \begin{cases} \mathit{true} & \text{if } w \in L \\ \mathit{false} & \text{if } w \notin L \end{cases} \qquad \text{for every } w \in A^n.$$

Lemma 15.2 *For every deterministic Turing machine, there is an algebra that recognises its language.*

Proof
By unraveling the definition of a Turing machine. □

The lemma does not require the machine to be total: it is allowed to have non-terminating computations on non-accepted words. In other words, the theorem says that (deterministically) semi-decidable languages are recognised by algebras.

15.0.3 Algebras do not recognise L

By Lemma 15.2, in order to show that L is not deterministically semi-decidable, it suffices to show the following:

Theorem 15.3 *No orbit-finite algebra recognises L.*

The rest of this section is devoted to proving this theorem.

Triangulations and parity. A set of (straight or parity) triangles is called *triangulated* if

- side sets are either disjoint or equal,
- every side set appears in exactly two triangles.

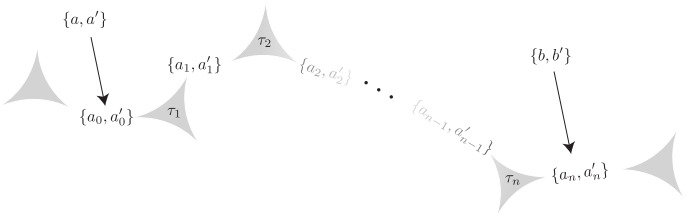
Lemma 15.4 *Let $\mathcal{T}$ be triangulated set of parity triangles. All straightenings of $\mathcal{T}$ have the same parity of the number of conflicts.*

Proof
Since every side set appears in exactly two triangles, doing a swap on a single straight triangle changes the number of conflicts by one: either it adds one or removes one. To go from one straightening to another, one needs to do an even number of swaps. □

We say that two (straight or parity) triangles are *neighbouring* if they share a side set. A set of triangles is called *connected* if every triangle can be reached from every other via a sequence of neighbouring triangles.

Lemma 15.5 *A connected and triangulated set of parity triangles belongs to L if and only if at least one (equivalently, all) of its straightenings has an even number of conflicts.*

Proof
The left to right implication is from the definition of the language: zero conflicts is an even number of conflicts. Consider the right to left implication, which uses the assumption on connectedness. Suppose that $\mathcal{T}$ is a connected, triangulated set of parity triangles which has a straightening $\mathcal{S}$ with an even (but nonzero) number of conflicts. We will find another straightening of $\mathcal{T}$ that has two conflicts less. By iterating this procedure, we eventually reach a straightening with zero conflicts. Let then $\{a, a'\}$ and $\{b, b'\}$ be two conflicting sides of two straight triangles in $\mathcal{S}$. By the assumption on connectedness, there are neighbouring straight triangles $\tau_1, \ldots, \tau_n \in \mathcal{S}$ that connect $\{a, a'\}$ with $\{b, b'\}$ as in the following picture:

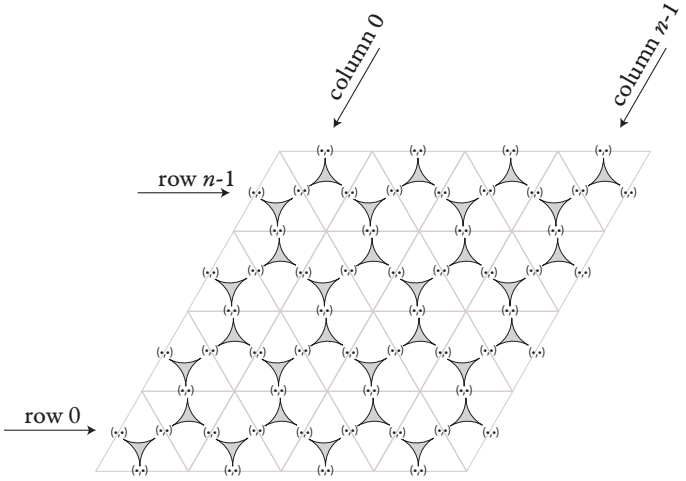


For each $i \in \{0, \ldots, n\}$, let σ_i be the triangle obtained from τ_i by swapping the side sets $\{a_{i-1}, a'_{i-1}\}$ and $\{a_i, a'_i\}$. Let $\mathcal{S}'$ be the set of triangles obtained from $\mathcal{T}$ by replacing $\tau_1, \ldots, \tau_n$ with $\sigma_1, \ldots, \sigma_n$. Since two or zero swaps are done on each straight triangle, we see that $\mathcal{S}'$ is also straightening of $\mathcal{T}$. We claim that $\mathcal{S}'$ has two less conflicts than $\mathcal{S}$. Indeed, the side set $\{a, a'\}$ is no longer a conflict, since we swapped the atoms a and a' on one of the triangles incident to it, namely the triangle τ_0, but not on the other triangle. The same argument holds for the side $\{b, b'\}$. On the other hand, for each $i \in \{1, \ldots, n-1\}$, the side $\{a_i, a'_i\}$ is a conflict in $\mathcal{T}$ if and only if it was a conflict in $\mathcal{S}$, because we swapped the atoms a_i and a'_i on both triangles that contain the edge set. □

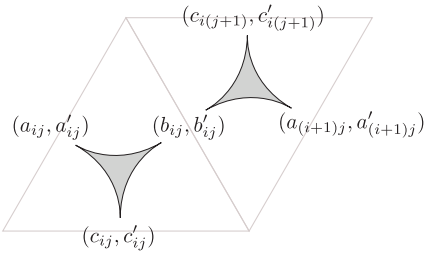
Toruses To construct an input that will confuse an algebra, we place parity triangles in a torus-like arrangement. Let $n \in \mathbb{N}$. Let us begin with $3n^2$ side sets

$$\{a_{ij}, a'_{ij}\} \qquad \{b_{ij}, b'_{ij}\} \qquad \{c_{ij}, c'_{ij}\} \qquad \text{for } i, j \in \{0, \ldots, n-1\}.$$

We connect these side sets using $2n^2$ straight triangles, as depicted below:



The nonconflicting straight n torus



The tile in column i and row j:

We adopt the convention that coordinates are counted modulo n, and therefore in each row, the side set on the right edge of the picture is equal to the side set on the left edge of the picture. The same goes for columns. In other words, the neighborhood graph of the triangles has the shape of a torus.

We use the name *straight nonconflicting n-torus* for the above set of straight triangles. A *straight n-torus* is obtained from the above by doing any number of swaps on any of the triangles. Finally, a *parity n-torus* is any set of parity triangles that can be straightened to a straight n-torus.

We now complete the proof of Theorem 15.3. We will show that, when a term from a fixed algebra is evaluated in a sufficiently large parity torus, then the result is insensitive to flipping almost all triangles. Let us formalise this notion. Let $\mathcal{A}$ be an orbit-finite algebra, ϕ be a term of $\mathcal{A}$, and $\mathcal{T}$ a parity n-torus. For a valuation $val : variables(\phi) \to \mathcal{T}$ and a parity triangle $\tau \in \mathcal{T}$, we say that ϕ and val *ignore* τ, if

$$\phi[val] = \phi[val_\tau],$$

where val_τ is defined like val, but with the parity triangle τ flipped to its dual.

By Lemma 15.5, flipping a single parity triangle affects membership in L. Therefore, a term used to recognise L (for inputs of given size) cannot ignore a single parity triangle. The following lemma shows that for sufficiently large toruses, almost all parity triangles will be ignored, thus showing that L cannot be recognised by an algebra.

Lemma 15.6 *Let $\mathcal{A}$ be an orbit-finite algebra. There is some $k \in \mathbb{N}$ such that for every sufficiently large n, every parity n-torus $\mathcal{T}$, every term ϕ in $\mathcal{A}$ and every valuation val with values in $\mathcal{T}$, ϕ and val ignore all but at most k elements of $\mathcal{T}$.*

Proof
Let r be the maximal arity of all operations in $\mathcal{A}$. The proof of Lemma 15.6 proceeds by induction on the size of the term ϕ. The base case is when the term is a variable, and a variable ignores all $\tau \in \mathcal{T}$ except for one. For the induction step, fix some val. The topmost operation in ϕ has arity at most r, so by the inductive assumption, there are at most $k \cdot r$ elements $\tau \in \mathcal{T}$ which are not ignored by ϕ and val. We need to show, however, that there are actually only at most k such elements τ. The argument has a geometric flavor, and builds on the following easy observation that it is hard to decompose a torus into small pieces:

Fact 15.7 *After removing m triangles in an n-torus, there remains a connected component of at least $2n^2 - m^2$ triangles.*

Define $m = 2(k_1 + k_2)$, where k_1 is the size of the least support of all (finitely many) operations in the algebra $\mathcal{A}$ and k_2 is the maximal size of a least support of an element of the universe of $\mathcal{A}$. We now reveal the value of k; we put $k = m^2$.

Let S be a set which supports all operations in the algebra $\mathcal{A}$ and the value $\phi[val]$. Because S supports all operations in the algebra, it follows that

$$\phi[\pi(val)] = \pi(\phi[val]) \qquad \text{for every } S\text{-automorphism } \pi.$$

Because S also supports $\phi[val]$, it follows that

$$\phi[\pi(val)] = \phi[val] \qquad \text{for every } S\text{-automorphism } \pi. \tag{13}$$

Without losing generality, S can be chosen so that it has at most k_1+k_2 elements. Because every atom appears in at most two parity triangles, there are at most m elements in $\mathcal{T}$ which use at least one atoms from S.

Assume now that n is sufficiently large; specifically, we need that $2n^2 > k \cdot r + k$. By Fact 15.7, there is a connected subset $C \subseteq \mathcal{T}$ such all elements of C have least supports disjoint with S, and the size of C is at least $2n^2 - k$, so it is bigger than $k \cdot r$. By the inductive assumption, some $\tau \in C$ is ignored by ϕ and val. For the proof of Lemma 15.6 it is enough to prove that *every* $\tau \in C$ is ignored by ϕ and val; indeed, there are at most k elements outside of C in the torus. To this end, since C is connected, it is now enough to show:

Lemma 15.8 *If some $\tau \in C$ is ignored by ϕ and val, then every neighbor $\tau' \in C$ of τ also is.*

To prove this, note that applying the atom automorphism π that swaps the atoms in the side set shared by τ and τ', has exactly the same effect on the torus $\mathcal{T}$ as flipping both τ and τ'. (For this we use the assumption that these atoms do not appear elsewhere in $\mathcal{T}$.) In consequence, flipping τ' has the same effect as applying π and then flipping τ. As the side set is disjoint from S, π is an S-automorphism so, by (13), it does not change the value of $\phi[val]$. As a result, flipping τ' has the same effect on $\phi[val]$ as flipping τ. □

16 For bit vector atoms, P≠NP

Recall the bit vector atoms that were introduced in Section **??**. The main result of this section is stated in the following theorem.

Theorem 16.1 *Consider the bit-vector atoms. Even when the input alphabet is the atoms,* P≠NP.

The problem that separates the classes is testing linear dependence of vectors, i.e. the following language:

$$D \stackrel{\text{def}}{=} \{a_1 \cdots a_n \in \text{Atoms} : a_1, \ldots, a_n \text{ are linearly dependent}\}.$$

Membership in NP is straightforward. The machine uses nondeterminism to guess a linear combination that witnesses dependence:

$$a_1 \cdot a_1 + \cdots + a_n \cdot a_n = 0$$

The coefficients are $a_1, \ldots, a_n$ are guessed as the machine moves through the input tape, after i steps the machine stores in its state the partial sum of a subset of the first i vectors. (To do this, the state space Q of the machine needs to be built using atoms.)

The rest of this section is devoted to showing that linear dependence cannot be done in deterministic polynomial time. The proof has two steps.

1. The input alphabet is the set of atoms, which is a straight set. One could imagine a Turing machine that quickly recognises dependence by using a non-straight work alphabet or state space, such as the Grassmannian from Example **??**. The first step shows that this is not the case: every deterministic machine can be made straight (i.e. have a straight work alphabet and state space), without affecting the running time.

2. In the second step, we show that a straight deterministic machine needs exponential time to reject a sequence of linearly independent vectors[10].

Step 1: Reduction to straight machines. The reduction to straight machines breaks down into two further substeps. We first show that the bit vector atoms have a property called *least closed supports*. Then, we show that when the atoms have least closed supports, then every deterministic machine over a straight input alphabet can be made straight without affecting the running time.

Consider an arbitrary atom structure, not necessarily the bit vector atoms. The *closure* of a set S of atoms is the set of all atoms that are supported by S. Of course the closure contains S, because every atom is supported by itself. The closure of a finite set is finite, although it can be much bigger, as established by the following lemma.

Lemma 16.2 *When the atoms are oligomorphic, the closure of a finite set is finite.*

A set of atoms is called *closed* if it is its own closure. We say that an atom structure has *least closed supports* if for every object x in sets over these atoms, the following condition holds: there is a finite closed support of x that is contained in all other finite closed supports.

Lemma 16.3 *Bit vector atoms have least closed supports.*

Proof
Under the vector space atoms, the closure of a set S of atoms is the linear space spanned by S, i.e. the set of vectors of the form

$$\sum_{v \in T} v \qquad \text{for some } T \subseteq S.$$

[10] For every length of the input, there is actually only one rejecting run, up to automorphism. A rejected input is a tuple of independent vectors. All independent n-tuples of vectors are in the same equivariant orbit. If the transition function is equivariant, then all runs over independent n-tuples of vectors are in the same orbit. In particular all of these runs have the same behaviour of the head (when it moves left or right, and when it produces a zero vectors).

Lemma 16.4 *If an element x of a set with atoms is supported by closed sets S and by T, then it is also supported by $S \cap T$.*

Proof
(TODO complete) □ □

To complete step 1, we will show the following result.

Lemma 16.5 *Assume that the atoms have least closed supports. Every deterministic machine M over a straight alphabet can be transformed into a straight deterministic machine M', with the same running times.*

To prove the above lemma, we show that least closed supports allow us to represent non-straight sets in a canonical way by straight sets. A *canonical straight representation* for an equivariant set Y is defined to be a surjective function $r : X \to Y$ such that X is a straight set, and the function preserves and reflects supports: for every $x \in X$, both x and $r(x)$ have the same supports.

Lemma 16.6 *Assume that the atoms have least closed supports. Every equivariant set has a canonical straight representation.*

Proof
It suffices to prove the lemma for single-orbit equivariant sets. Let X be a single-orbit equivariant set. Choose some $x \in X$, and let $\{a_1, \ldots, a_n\}$ be the least closed support of x. The function mapping $(a_1, \ldots, a_n) \mapsto x$ extends uniquely to an equivariant function

$$r : \mathrm{G} \cdot (a_1, \ldots, a_n) \to \mathrm{G} \cdot x = X.$$

The function r is a canonical straight represenation, because both $(a_1, \ldots, a_n)$ and x have the same least closed support, and this extends to the whole orbit by equivariance. □

The following lemma shows that an equivariant function can be lifted to canonical straight representations.

Lemma 16.7 *Let $r_1 : Y_1 \to X_1$ and $r_2 : Y_2 \to X_2$ be canonical straight representations, and let $f : X_1 \to X_2$ be an equivariant function. There is an equivariant function $g : X_1 \to X_2$ which makes the following diagram commute*

$$\begin{array}{ccc} X_1 & \xrightarrow{g} & X_2 \\ \downarrow{\scriptstyle r_1} & & \downarrow{\scriptstyle r_2} \\ Y_1 & \xrightarrow{f} & Y_2 \end{array}$$

Proof
We define the function g separately for each orbit of X_1. Choose some x_1 in some orbit of X_1. Because r_2 is surjective, there must be some $x_2 \in X_1$ with

$$r_2(x_2) = f(r_1(x_1)).$$

Because r_2 is a canonical straight representation, x_2 has the same support as the $f(r_1(x_1))$. Because r_1 is preserves supports, and f is equivariant, it follows that the least closed support of $f(r_1(x_1))$ is included in the least closed support of x_1. Summing up, the least support of x_2 is included in the least closed support of x_1. Therefore, the mapping $x_1 \mapsto x_2$ can be extended to an equivariant function on the orbit of x_1. We do the same for the other orbits. □

Proof (of Lemma 16.5)
By applying Lemma 16.6 to the work alphabet and the state space, and then applying Lemma 16.7 to the transition function. □

Step 2. In this step, we show that deterministic straight machines need exponential time to recognise if vectors are linearly independent.

Lemma 16.8 *When run on independent vectors $a_1 \cdots a_n$, a deterministic normal form machine for the language D must make exponentially many steps.*

Before proving Lemma 16.8, we prove the following sublemma.

Lemma 16.9 *For every nonempty $I \subseteq \{1, \ldots, n\}$ there is a set $b_1, \ldots, b_n$ of linearly dependent vectors such that*

$$\sum_{i \in J} b_i \neq 0 \qquad \text{for every } J \subseteq \{1, \ldots, n\} \text{ with } J \neq I$$

Proof (of Lemma 16.9)
Without loss of generality, we assume that $n \in I$. Choose some independent vectors $b_1, \ldots, b_{n-1}$. Define

$$b_n = b_k + b_{k+1} + \cdots + b_{n-1},$$

which guarantees linear dependency. To prove the statement of the lemma, consider a set J that is different from I. If J contains some $j \notin I$ then the sum from the statement of the lemma cannot be zero, because b_j is linearly independent from all the remaining vectors. Otherwise, J is a proper subset of I. If J does not contain n, then the sum from the statement of the lemma cannot be zero, because the vectors $I - \{n\}$ are linearly independent. Finally if J contains n but omits some $i \in I$, then the sum from the statement of the lemma will be nonzero on coordinate a_i. □

Proof (of Lemma 16.8)

Consider a run of deterministic normal form machine on an input $a_1 \cdots a_n$ which consists of independent vectors. Since the vectors are independent, the machine should reject.

As usual, a computation can be visualised as a rectangular grid, where each tile of the grid gets a colour which consists of a symbol of the work alphabet and a possibly a state (if the tile coincides with the head). We say that an atom is

two-supported by a computation if it is supported by the atoms that appear in the colours of two neighbouring tiles. The number of atoms in the colours of two tiles is bounded by the Turing machine. The number of colours supported by these atoms is therefore also bounded by the machine, if exponentially bigger. It follows that the number of atoms that are two-supported by a computation is quadratic in the length of the run, assuming that the machine is fixed.

The transition function of the machine is equivariant, so everything it produces has smaller or equal support to its arguments. It follows that every atom which appears in the computation, or is two-supported by it, is already supported by the input, and therefore each of these atoms is a linear combination of the input vectors. Since the computation is not exponentially long, then some linear combination must be missing, i.e. there must be some $I \subseteq \{1, \ldots, n\}$ such that the atom

$$\sum_{i \in I} a_i$$

is not two-supported by the computation.

Apply Lemma 16.9, to the set I, yielding a tuple of linearly dependent atoms $b_1, \ldots, b_n$. We will show that this input is also rejected by the machine, which contradicts the assumption that the machine accepts all dependent tuples. Consider the computations of the machine on inputs $a_1 \cdots a_n$ and $b_1 \cdots b_n$, encoded as coloured grids, call them ρ and σ.

Claim 16.9.1 *For every two tiles x, y the pairs of colours*

$$(\rho(x), \rho(y)) \qquad (\sigma(x), \sigma(y))$$

are in the same equivariant orbit.

Proof
By induction on the distance of the tiles from the top row of the grid, which contains the input. □

By taking $x = y$, the claim implies that for every individual tile, the colors in both grids are in the same equivariant orbit. Whether or not a state is accepting is an equivariant property, and therefore one computation contains an accepting state if and only if the other computation contains an accepting state. □

Part V
Logics

17 The compactness theorem (planned)

18 Eliminating orbit-finite disjunction (planned)

Part VI
Automata, continued

19 Other models of finite automata (planned)

19.1 Alternating automata (planned)

Exercise 23. Show that for the equality atoms, the Higman order on Atoms^* admits orbit-infinite chains.

Exercise 24. Show that for the equality atoms, there is a language that is upward closed under the Higman order, but is not recognised by a nondeterministic orbit-finite automaton.

19.2 Two-way automata (planned)

20 Learning of finite automata

Learning deterministic orbit-finite automata. Consider the following setup. There are two players: learner and teacher. An input alphabet is fixed, and known to both learner and teacher. Teacher chooses an equivariant language over the input alphabet, which is recognised by a deterministic orbit-finite automaton. Learner wants to learn the teacher's language, asking a smallest amount of questions. Learner can ask two kinds of questions to teacher:

- *Membership.* Learner asks if a given word belongs to the teacher's language.
- *Equivalence.* Learner proposes a deterministic orbit-finite automaton, and asks if it recognises the teacher's language. If it does not recognise the teacher's language, teacher gives an example of a word to which the learner's automaton gives the wrong answer.

Theorem 20.1 *Assume that the atoms admit least supports. Learner has an algorithm, which learns the teacher's language, and asks a number of questions that is exponential in the number of equivariant orbits in the syntactic automaton of teacher's language.*

Without atoms, the algorithm runs in polynomial time. The reason for the exponential is that the number of equivariant orbits in a product $X \times Y$ can be much greater than the product of the numbers of equivariant orbits in X and Y. For example, if X is the set of nonrepeating n-tuples of equality atoms, which is a one orbit set, then the product $X \times X$ has an exponential number of orbits, namely one for every partial bijection between $\{1, \ldots, n\}$ and $\{1, \ldots, n\}$.

The rest of this section is devoted to describing learner's algorithm. Let us fix teacher's language, call it L. This language is initially unknown to the learner.

If D is a set of words, we say that words w and w' can be D-distinguished if there is some $u \in D$ such that only one of the words wu and $w'u$ is in the language L. If words cannot be D-distinguished, then they are called D-indistinguishable. The Myhill-Nerode equivalence of the language L is the same as A^*-indistinguishability. The general idea is that learner will approximate A^*-indistinguishability by D-indistinguishability, for growing orbit-finite sets D.

An *approximation* consists of two equivariant orbit-finite sets of words, the *state reaching words* R and the *distinguishers* D, subject to the following conditions. An approximation is called *closed* if for every word $w \in R$ and every letter $a \in A$, there is some word in R that is D-equivalent to wa. An approximation is called *congruent* if D-equivalence

$$w \sim_D w' \qquad \text{implies} \qquad wa \sim_D w'a$$

A *partial equivalence relation* is a relation that is symmetric and transitive, but not necessarily reflexive. An approximation (R, D) defines a partial equivalence relation on Q as follows. The domain is the states reachable via words

from R. Two states in the domain are considered equivalent if they cannot be D-distinguished. Since D and R are equivariant, (R, D)-equivalence is also equivariant.

Let $\sim$ and $\approx$ be two partial equivalence relations on the same set. We say that $\approx$ is more exact than $\sim$ if every equivalence class of $\sim$ is a union of equivalence classes of $\approx$. In other words, the domain (elements which satisfy $x \approx x$) of $\approx$ is at least as big as the domain of $\sim$, and on the domain of $\sim$ the relation $\approx$ is equal to $\sim$ or more refined.

The algorithm. The algorithm maintains an approximation (R, D). In each iteration of the main loop, the (R, D)-equivalence becomes more exact. As we will show in Lemma 20.3, a partial equivalence relation on Q can become more exact at most a polynomial number of times in the number of orbits in Q.

Initially, R contains only the empty word and D is empty. It then repeats the following steps, until teacher's language is learned.

1. **Main loop.** Asking membership questions, learner finds the answer to every question of the form

   $$wav \stackrel{?}{\in} L \qquad \text{for } w \in R,\ a \in A \cup \{\epsilon\}, \text{ and } v \in D$$

   Since the language is known to be equivariant, it is enough to ask one membership question per equivariant orbit of the set

   $$R \times D \qquad \cup \qquad R \times A \times D.$$

   Unfortunately, the number of such orbits is exponential in the number of orbits of R, A and D.

   If the approximation is not closed, then do step 1a, otherwise if it is not congruent then do step 1b, and otherwise (if it is both closed and congruent), then do step 1c.

   (a) **Not closed.** Because the approximation is not closed, then there is a word $w \in R$ and a letter $a \in A$, such that wa is D-distinguishable from every word in R. Add the orbit of wa to R, and return to step 1.

   (b) **Closed, but not congruent.** Because the approximation is not congruent, then there are words $w, w' \in R$ and a letter $a \in A$, such that w and w' cannot be D-distinguished, but can be aD-distinguised. Find the word $u \in D$ such that au distinguishes w and w'. Add the orbit of au to D, and return to step 1.

   (c) **Closed and congruent.** If this step is reached, then the approximation is closed and congruent. Consider the following automaton, called the (R, D)-automaton. The states are D-equivalence classes of words in R. The transition function is defined by

   $$([w]_{\sim_D}, a) \mapsto [v]_{\sim_D} \qquad \text{where } v \in R \text{ is } D\text{-equivalent to } wa.$$

The word v is guaranteed to exist if the approximation is closed. The D-equivalence class of v does not depend on the choice of w from its D-equivalence class, because the approximation is congruent. Summing up, the (R, D)-automaton is deterministic. Learner asks if the (R, D)-automaton recognises teacher's language. If yes, finish the algorithm. If not, let $a_1 \cdots a_m$ be teacher's example of a word where the (R, D)-automaton gives the wrong answer.

Claim 20.1.1 *There is some $i \in \{1, \ldots, m\}$ and $u \in D$ such that*

$$a_1 \cdots a_{i-1} \not\sim_{D \cup \{a_i u\}} v \qquad \text{for every } v \in R.$$

Proof
Let $v_i \in R$ be the state of the (R, D)-automaton after reading the prefix $a_1 \cdots a_i$. Choose the first $i \in \{1, \ldots, m\}$ such that v_i and $a_1 \cdots a_i$ are D-distinguished by some word, call it $u \in D$. This i must exist, since for $i = 0$ the two words are equal and for $i = m$ the two words are distinguished by the empty word.
By choice of i, the word v_{i-1} and $a_1 \cdots a_{i-1}$ are D-equivalent. We now prove the claim. Consider some $v \in R$. If v is D-distinguishable from v_{i-1}, then it is also D-distinguishable from $a_1 \cdots a_{i-1}$. The more interesting case is when v is not D-distinguishable from v_{i-1}. By congruence, va_i is not D-distinguishable from $v_{i-1}a_i$, which is not D-distinguishable from v_i by definition of the (R, D)-automaton. It follows that u also distinguishes va_i from $a_1 \cdots a_i$, and therefore $a_i u$ distinguishes v from $a_1 \cdots a_{i-1}$. □

The algorithm finds i and u by asking membership queries, and adds the orbit of $a_1 \cdots a_{i-1}$ to R, and the orbit of $a_i u$ to D.

In Lemma 20.2, we show that each time the main loop is executed, the partial equivalence relation induced by (R, D) becomes more exact. In Lemma 20.3, we show that a partial equivalence relation on Q can become more exact at most a polynomial number of times (in the number of orbits in Q). The proof of Lemma 20.2 is the same as without atoms, while Lemma 20.3 needs an additional observation, because even single-orbit sets can admit nontrivial equivalence relations.

Lemma 20.2 *Each time the main loop is executed, the partial equivalence relation induced by (R, D) becomes more exact.*

Proof
In step 1a, the domain grows. In step 1b, the domain stays the same, but the equivalence becomes more refined. In step 1c, the domain grows. □

The *dimension* of a finitely supported set X is defined to be

$$\sup_{x \in X} |\text{least support of } x|$$

The above assumes existence of least supports.

Lemma 20.3 *Assume that the atoms admit least supports. Let Q be an equivariant set of dimension d and with k orbits. Let*

$$\sim_1, \ldots, \sim_m$$

be more and more exact equivariant partial equivalence relations on Q. Then m is at most $k \cdot d \cdot \log d$.

Proof
The quotient is defined as usual: it is the set of equivalence classes. Consider the quotients

$$Q/_{\sim_1}, \ldots, Q/_{\sim_m}$$

The number of orbits in the quotient grows as the relation becomes more exact. The number of orbits cannot grow more than k times. Therefore, we can assume without loss of generality that all the quotients have the same number of orbits. Choose some orbit of $Q/_{\sim_1}$, call it X. The sequence

$$X/_{\sim_1}, \ldots, X/_{\sim_m}$$

is a sequence of (complete) equivalence relations on a single orbit set. We will prove that such a sequence can have length at most $d \cdot \log d$. By the representation theorem, $X/_{\sim_1}$ is isomorphic to nonrepeating d-tuples of atoms modulo some subgroup of $Sym(d)$. A quotient under an equivariant equivalence relation corresponds to either: making the dimension d smaller, or making the subgroup bigger. The dimension can be made smaller at most d times. The Lagrange theorem says that a subgroup of a finite group can have at most half of the elements of the containing group. Therefore, when the dimension stays the same, say d, then the subgroup can grow at most $\log(d!)$ times before it covers all $d!$ permutations. Since $\log(d!)$ is around $d \log d$, the statement follows. □

Conjecture 20.4 *Consider the equality atoms. Learner has an algorithm, which learns the teacher's language, and asks at most*

$$exp(d) \cdot poly(n)$$

questions, where the numbers d and n are obtained from the syntactic automaton of teacher's language as follows:

- *d is the dimension of the state space.*
- *n is the number of equivariant orbits in the state space.*

21 Database-driven systems (planned)

22 Monoids (planned)

Part VII
Solutions to the exercises

Solution to Exercise 1.
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-coloring. A legal two-coloring, 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 2.
For the right-to-left implication, we show that if condition 3 in the theorem fails, then by a cardinality argument there is a finitely supported subset of the atoms which is not single-support orbit-finite. If condition 3 fails, then there is a support $\bar{a}$ such that the atoms have infinitely many $\bar{a}$-orbits. Any union of these orbits is an $\bar{a}$-supported subset of the atoms, and therefore the atoms have uncountably many finitely-supported subsets. On the other hand, there are only countably many single-support orbit-finite subsets of the atoms, because such a subset can be described in a finite way by giving the supporting atoms and a finite set of atoms, one per each orbit. The same argument shows a stronger statement: if condition 3 fails, then there is a single-support orbit-finite set (the atoms) which has a finitely supported subset that is not even multi-support orbit-finite.

For the left-to-right implication, suppose that conditions the theorem hold, and consider a single-support orbit-finite set X, which admits a representation

$$X = \bigcup_{i \in I} f_i(X_i) \qquad \text{where } X_i = \{\pi(\bar{b}_i) : \pi \text{ is a } \bar{a}_i\text{-automorphism}\}$$

by Lemma **??**. Consider a subset $Y \subseteq X$, which is supported by atoms $\bar{c}$. Then

$$Y = \bigcup_{i \in I} f_i(X_i \cap f_i^{-1}(Y_i)).$$

Since $\bar{c}$-supported sets are closed under preimages of equivariant functions, for every i the set

$$X_i \cap f_i^{-1}(Y_i)$$

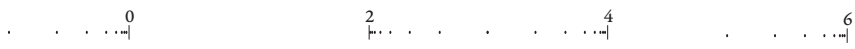
is a $\bar{c}$-supported set of tuples of atoms. By condition 4, it has finitely many $\bar{c}$-orbits, and therefore the whole set Y is a single-support orbit-finite.

Solution to Exercise 3.
Let X be any subset of the atoms, not necessary legal. The equivariant orbit $\mathrm{G} \cdot X$ is a single-orbit set, although it might be illegal. Although different

choices of X might lead to the same orbit $\mathrm{G} \cdot X$, there are still uncountably many possibilities for $\mathrm{G} \cdot X$. (This is not true in the equality atoms, where $\mathrm{G} \cdot X$ is uniquely determined by the sizes of X and its complement. That is why the construction needs to be more complicated.) For instance, given a set A of natural numbers, consider the set

$$X_A = \bigcup_{i \in \mathbb{N}} I_i \qquad \text{where } I_i = \begin{cases} \{2i + \frac{1}{n} : \text{ for } n \in \mathbb{N}_{>0}\} & \text{when } i \in A \\ \{2i - \frac{1}{n} : \text{ for } n \in \mathbb{N}_{>0}\} & \text{when } i \in A. \end{cases}$$

This set is illegal regardless of A, because it is not a finite union of intervals. When A contains 1 but none of $\{0, 2, 3\}$, then the beginning of the set X_A looks like this:



When A and B are different sets of natural numbers, then the orbits $\mathrm{G} \cdot X_A$ and $\mathrm{G} \cdot X_B$ are different, even disjoint. This is because applying an order automorphism π of the rational numbers will preserve the notion of left and right limit points. What is more, there is no finitely supported bijection

$$f : \mathrm{G} \cdot X_A \to \mathrm{G} \cdot X_B. \tag{6}$$

Suppose that f is a finitely supported function as in (6), and that it maps a set Y to a set Z. Let S be a finite support of f. It is not difficult to see that the sets Y and Z must disagree on infinitely many rational numbers. Therefore, there must be a rational number $x \notin S$ which belongs to exactly one of Y or Z. Since all points in the set $S \cup Y \cup Z$ are isolated, there must be a small enough open interval I which contains x, but no other elements of $S \cup Y \cup Z$. Let π be an automorphism of the rational numbers which moves x but is the identity outside I. Because S supports f, the graph of f is invariant under π. Because I contains only x from the set $Y \cup Z$, it follows that π modifies the set among Y, Z which contains x, but does not do anything to the set which does not. It follows that f is not a bijection, which contradicts our assumption.

Solution to Exercise 4.
A similar construction.

Solution to Exercise 5.
Suppose that X is an orbit-finite set, and f is a finitely supported function. Let $\bar{a}$ be a support of both X and f. Because X is all-support orbit-finite, it has finitely many $\bar{a}$-orbits. The image of a $\bar{a}$-orbit under a $\bar{a}$-supported function is also a $\bar{a}$-orbit:

$$f(\{\pi(x) : \pi \text{ is a } \bar{a}\text{-automorphism}\}) = \{\pi(f(x)) : \pi \text{ is a } \bar{a}\text{-automorphism}\}.$$

Therefore, the image $f(X)$ is a finite union of $\bar{a}$-orbits.

Solution to Exercise 6.
Follows from Lemma 4.2 and the fact that the set of n-tuples of atoms is a hereditarily orbit-finite set (recall how tuples are coded as sets).

Solution to Exercise 7.
Implication from 1 to 2. By assumption on effectiveness, for n we can compute the number of orbits of n-tuples, say k. We start enumerating all finite sets of formulas in n free variables, until we find a set Γ of k formulas which define nonempty but disjoint sets of n tuples of atoms. Such a set must exist, and its correctness can be determined using the first-order theory of the atoms. Two n-tuples are in the same orbit if and only if they satisfy the same (unique) formula from Γ.

Implication from 2 to 1. Using the formula for having the same orbit of n-tuples, for every k one can write a first-order sentence which says that there exist k representative n-tuples which are in pairwise different orbits, and every other n-tuple is in the same orbit as one of these representatives. The unique k for which this formula is true is the number of orbits of n-tuples.

Solution to Exercise 8.

1. Using item 3, we find a tuple of atoms $\bar{a}$ that supports both f and x. Using item 4, we decompose f, seen as a binary relation, into $\bar{a}$-orbits

$$f = \bigcup_{i \in I} f_i.$$

   We can look through all the orbits f_i, and for each one use item 5 to find an element, call it y_i. One of the orbits f_i contains the pair $(x, f(x))$, and since the $\bar{a}$-orbit of this pair is a singleton, it follows that some y_i is equal to $(x, f(x))$. Therefore, the second coordinate of this y_i stores the desired result.

2. Using item 3, we find a tuple of atoms $\bar{a}$ that supports X. Using items 4 and 5, we compute representations of elements $x_1, \ldots, x_n \in X$ which cover all $\bar{a}$-orbits. Using the previous item, we compute the values f iin these elements. Using item 4, we compute all $\bar{a}$-orbits that intersect

$$\{f(x_1), \ldots, f(x_n)\},$$

   and we return the union of these orbits.

Solution to Exercise 9.

Claim 5.4.1 *One can compute a sequence of first-order formulas $\varphi_1, \varphi_2, \ldots$ defining orbits of nonrepeating tuples such that the following property holds. Let*

$\bar{a}$ be a tuple in the orbit φ_i. Then this tuple can be extended to a tuple $\bar{a}\bar{b}$ in the orbit φ_{i+1} such that for every a in $\mathfrak{A}$ but not in $\bar{a}$, there is some b that appears in the tuple b such that a and b are in the same $\bar{a}$-orbit.

From the claim above it follows that there is an infinite sequence $b_1, b_2, \ldots$ of elements in $\mathfrak{A}$, such for every i, some finite prefix of this sequence is in the orbit φ_i. Let $\mathfrak{B}$ be the structure $\mathfrak{A}$ with the universe restricted to elements appearing in the infinite sequence. (We assume without loss of generality that the vocabulary contains no functions.) From the claim above it follows that Duplicator can win the infinite round Ehrenfeucht-Fraïssé game between $\mathfrak{A}$ and $\mathfrak{B}$, and therefore the two structures are isomorphic.

Clearly the universe of $\mathfrak{B}$ can be represented: the element b_i is represented by the number i. We claim that under this representation, one can decide the truth value of formulas with free variables, assuming the valuation of the free variables is represented by the numbering. By renaming variables, we can assume that formula and valuation on the input of the problem are such that the free variables of the formula are included in $x_1, x_2, \ldots$ and the valuation is such that variable x_i is mapped to b_i. In this case, if the formula uses n variables, then its truth value is uniquely determined by the orbit of the tuple $(b_1, \ldots, b_n)$, and this orbit is known from the sequence of formulas $\{\varphi_i\}_i$ in the claim.

Solution to Exercise 10.
Suppose that f is supported by atoms $\bar{a}$. Every input to f, being a bit string, has empty support. It follows that every output of f is supported by $\bar{a}$, and therefore every atom is supported by $\bar{a}$. By Corollary 4.3, there are finitely many $\bar{a}$-supported atoms.

Solution to Exercise 11.
Consider the left-to-right inclusion. The proof is by induction on the expression. The only interesting step is a set expression

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

The value of this expression (for some given $\bar{x}$-valuation) is a subset of the image of τ under the equivariant function, which maps a $\bar{x}\bar{y}$-valuation η to $[\alpha]_\eta$. Orbit-finite sets are preserved under images and subsets.

Consider the right-to-left inclusion. The proof is by induction on the rank of a hereditarily orbit-finite set. The atom expressions define all atoms. Expressions are closed under finite unions. The only step is showing that if a set is defined by a set expression, then so is its S-orbit. This follows from items 1 and 4 in Theorem 5.4, because the S-orbit of a set x is the same as the union of S-orbits that intersect the set $\{x\}$.

Solution to Exercise 12.

Solution to Exercise 13.
One can decide if a bit-string is a ρ_1-representation of an atom, by testing if it has no elements, and that it is not equal to the empty set. Consider some computable enumeration $w_1, w_2, \ldots$ of ρ_1-representations of atoms, e.g. bit-strings are sorted first by length and then lexicographically. Likewise, consider an enumeration $v_1, v_2, \ldots$ of ρ_2-representations of atoms. By induction on n we compute finite sets of bit-strings

$$\emptyset = Y_0 \subseteq X_1 \subseteq Y_1 \subseteq X_2 \subseteq Y_2 \subseteq \cdots$$

and values of f on bit strings from these sets such that for every $n \geq 1$:

- The set X_n contains $w_1, \ldots, w_n$;
- The image $f(Y_n)$ contains $v_1, \ldots, v_n$;
- There is an atom automorphism π which makes this diagram commute

$$\begin{array}{ccc} X_n & \xrightarrow{f} & f(X_n) \\ \downarrow{\scriptstyle \rho_1} & & \downarrow{\scriptstyle \rho_2} \\ \text{atoms} & \xrightarrow{\pi} & \text{atoms} \end{array}$$

Since both ρ_1 and ρ_2 are surjective, it follows that in the limit we get the desired result. We show how to compute X_n based on Y_{n-1}; the step from X_n to Y_n is done analogously. Let

$$Y_{n-1} = \{y_1, \ldots, y_k\}.$$

If w_n is already in Y_{n-1}, we do nothing. Otherwise, we add w_n to Y_{n-1}. We need to define f on w_n. If $\rho_1(w_n)$ is equal to $f(y_i)$ for some $i \in \{1, \ldots, k\}$, which can be determined using the equality test, then we define $f(w_n)$ to be $f(y_i)$. In the remaining case, we start enumerating all formulas of first-order logic in $k+1$ variables, until we find a formula φ which defines the orbit of the tuple

$$(\rho_1(y_1), \ldots, \rho_1(y_k), \rho_1(w_n)). \tag{7}$$

This formula can be found using Exercise 12. Then we start enumerating all bit strings, until we find a string v such that the tuple

$$(\rho_2(f(y_1)), \ldots, \rho_2(f(y_k)), \rho_2(v)).$$

also satisfies the formula φ, and is therefore in the same orbit as (7). We define $f(w_n)$ to be v.

Solution to Exercise 14.
Given a bit-string w which ρ_1-represents a set X, we proceed as follows, by induction on the nesting of set brackets in X. We compute a ρ_1-representation of

a support $\bar{a}$ of X. By the previous exercise, we can compute a ρ_2-representation of $\pi(\bar{a})$.

We know that X is a finite union of its $\bar{a}$-orbits, let this decomposition be

$$X = \bigcup_{i \in I} X_i.$$

We compute ρ_1-representations of the $\bar{a}$-orbits X_i. For each orbit X_i, we compute a ρ_1-representation of an element $x_i \in X_i$. By applying the induction assumption, we compute a ρ_2-representation of $\pi(x_i)$. We compute a ρ_2-representation of $\pi(\{x_i\})$. We then compute a ρ_2-representation of the $\pi(\bar{a})$-orbit intersecting $\pi(\{x_i\})$, which is the same as the $\pi(\bar{a})$-orbit of $\pi(x_i)$, which is equal to the image under π of the $\bar{a}$-orbit of x_i. Summing up, we have computed a ρ_2-representation of $\pi(X_i)$ for each i, which leads to a ρ_2-representation of $\pi(X)$.

Solution to Exercise 15.
If two tuples of atoms $\bar{a}$ and $\bar{b}$ are in the same orbit, then the sizes of their generated substructures are the same.

Solution to Exercise 16.
Let us revisit the proof in Exercise 20. The proof uses property (*), which fails in the universal equivalence relation or in the random graph. However, the proof only needs a weaker property, namely:

> (**) Every finite partial automorphism f of the atoms can be extended to a complete automorphism π, such that for some $n \in \mathbb{N}$:
>
> $$\pi^n(a) = a \qquad \text{for every } a \text{ in the domain of } f.$$

We show that property (**) holds in the Fraïssé limit of undirected graphs. We use a theorem of Hrushovski [3]:

Theorem 6.10 *Every finite (undirected) graph G embeds in some finite graph H such that every partial automorphism of G extends to a complete automorphism of H.*

To prove property (**), let G be the subgraph of the random graph induced by the domain and co-domain of f. Apply Theorem 6.10, yielding some H. By the universality property of the random graph, we may assume that H is a finite induced subgraph of the random graph. By the theorem, f extends to some full automorphism of H, and that automorphism extends to a full automorphism π of the random graph, which preserves H as a set. It follows that there is some n such that π^n is the identity when restricted to H.

Solution to Exercise 17.
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 18.
Consider the equality atoms and the set

$$\text{Atoms}^{(*)} \quad \stackrel{\text{def}}{=} \quad \bigcup_n \text{Atoms}^{(n)},$$

i.e. the set of nonrepeating 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

$$f : \text{Atoms}^{(*)} \to \text{Atoms}^{(*)}$$

is a bijection. Suppose that f is supported by a finite tuple of atoms $\bar{a}$. For a tuple in $\text{Atoms}^{(*)}$ 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 7.0.1 *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 $\text{Atoms}^{(*)}$ 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 $\text{Atoms}^{(*)}$ 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 19.
Let X be the set of atoms, and let $\mathcal{X}$ be the chain of half-open intervals:

$$\mathcal{X} = \{(-\infty; x) : x \in \text{Atoms}\}.$$

This is an equivariant chain, which sums up to all the atoms, but which does not contain the set of all the atoms. It is, however, not uniformly supported, because each interval is supported by its upper bound x.

Solution to Exercise 20.
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.

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 21.
The proof is the same as the standard proof for normal sets.

- When transforming a context-free grammar into a pushdown automaton, we do not need any assumptions on the atoms. The classical construction works. 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. Here it is useful to assume that the grammar is in Chomsky normal form.

- When transforming a pushdown automaton into a context-free grammar, we use the assumption that the atoms are homogeneous over a finite vocabulary, since we need closure of orbit-finite sets under products. Suppose that the pushdown automaton is as in Definition **??**.

  The nonterminals are going to be

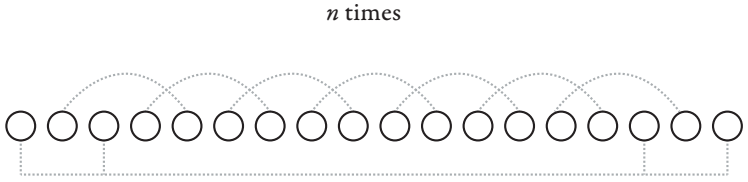
  $$\mathcal{N} = Q \times \Gamma \times Q.$$

  (Here we need the assumption on the atoms, since $\mathcal{N}$ is a product of orbit-finite sets.) 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 in the classical construction; it is easy to see that the set of rules is orbit-finite.

Solution to Exercise 22.
Since the computation tree is well-founded, we can use induction to show that for every computation tree, there is one which satisfies the statement of the lemma, and where the label of the root is the same. The interesting case is the induction step, when the root of a computation tree t is labelled by a configuration c which uses universal control state. Let $\bar{a}$ be a support of both the root of t and the alternating machine. The set of successor configurations of the root label is orbit-finite, and therefore it has finitely many $\bar{a}$-orbits. Choose configurations $c_1, \ldots, c_n$ which represent all these $\bar{a}$-orbits. Choose child subtrees of the root $t_1, \ldots, t_n$ which begin in the configurations $c_1, \ldots, c_n$. By induction assumption, these smaller computation trees can be converted into computation trees $s_1, \ldots, s_n$ of orbit-fintie size and finite depth, such that the roots of these trees are $c_1, \ldots, c_n$. If π is a $\bar{a}$-automorphism and $i \in \{1, \ldots, n\}$, then $\pi(s_i)$ is a computation tree whose root is a successor configuration of c. It follows we get a computation tree if we use label the root by c and have children of the form $\pi(s_i)$ ranging over $\bar{a}$-automorphisms π and $i \in \{1, \ldots, n\}$. The depth of this new tree is bounded by one plus the maximal depth of the trees $s_1, \ldots, s_n$. It has an orbit-finite set of nodes, because its set of nodes is an orbit-finite union of orbit-finite sets of nodes.

Solution to Exercise 23.
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 atoms, 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 a not in the same orbit 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 atoms. 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 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 24.
Consider the set W of words in the solution to Exercise 23. Let $W_P \subseteq W$ be the subset of words that have a prime number of different atoms. Finally, let L be the upward closure of W_P under the Higman order. We claim that this language is not recognised by a nondeterministic orbit-finite 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.

References

[1] Mikolaj Bojanczyk. Data monoids. In *STACS*, pages 105–116, 2011.

[2] Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata with group actions. In *LICS*, pages 355–364, 2011.

[3] Ehud Hrushovski. Extending partial isomorphisms of graphs. *Combinatorica*, 12(4):411–416, 1992.

[4] T.J. Jech. *The Axiom of Choice*. Studies in Logic and the Foundations of Mathematics Series. North-Holland Publ., 1973.

[5] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.

[6] Jin yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.