Orbit-Finite Sets and Their Algorithms[*][†]

Mikołaj Bojańczyk

University of Warsaw, Warsaw, Poland
bojan@mimuw.edu.pl

Abstract

An introduction to *orbit-finite sets*, which are a type of sets that are infinite enough to describe interesting examples, and finite enough to have algorithms running on them. The notion of orbit-finiteness is illustrated on the example of register automata, an automaton model dealing with infinite alphabets.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Orbit-finite sets, sets with atoms, data words, register automata

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.1

Category Invited Talk

1 Introduction

An orbit-finite set is a set that is constructed using some infinite logical structure, such as $(\mathbb{N}, =)$ or $(\mathbb{Q}, <)$, and which is finite up to automorphisms of that structure. For example, if the structure is $(\mathbb{N}, =)$, then

$$\{X : X \subseteq \mathbb{N} \text{ and } |X| \leq 3\}$$

is an orbit-finite set, because automorphisms of the structure (i.e. permutations of $\mathbb{N}$) can be used to map any subset $X \subseteq \mathbb{N}$ to any other subset of same cardinality, and therefore the set has four elements (cardinalities 0, 1, 2, 3) up to automorphisms.

The goal of this paper is to give a gentle introduction to orbit-finite sets, in particular to explain how they can be represented and manipulated using algorithms. As a running example we use register automata over infinite alphabets. A more detailed description can be found in the lecture notes [5].

2 The running example: register automata

As our running example for describing orbit-finite sets, we consider register automata over data words, and their associated decision problems such as emptiness or minimisation.

Typically, formal language theory uses finite alphabets. Here is an example which uses an infinite alphabet.

▶ **Running Example.** *Let $\mathbb{A}$ be some infinite set. As our running example, consider the language*

$$\{w \in \mathbb{A}^* : \text{ at most two distinct letters appear in } w\}. \tag{1}$$

[*] This paper is part of LIPA, a project funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 683080).
[†] I would like to thank Thomas Colcombet and Szymon Toruńczyk for comments on a first draft.

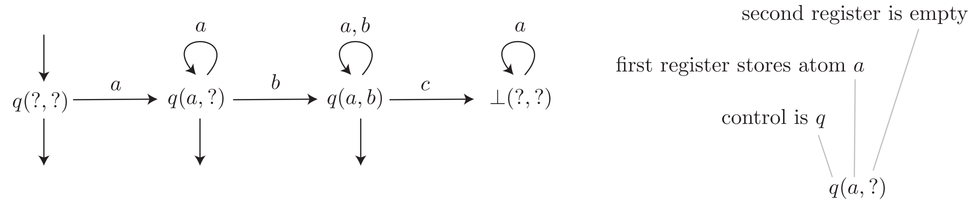


Figure 1 There are two possible values for the control state, q and $\bot$. A configuration of the automaton consists of a value for the control plus a valuation of the registers. Dangling arrows indicate initial and accepting configurations. The variables a, b, c range over distinct atoms, i.e. to get the automaton one should instantiate the picture for every triple $(a, b, c) \in \mathbb{A}^3$ of distinct atoms.

In our running example, the letters are only compared with respect to equality. Words as in the example are called languages of *data words*. The most common type of alphabet for data words is of the form $\Sigma \times \mathbb{A}$, where Σ is a finite set and $\mathbb{A}$ is an infinite set whose elements can only be compared for equality. We will use the name *atoms* for $\mathbb{A}$; to underline that they have no structure except for equality. Automata that process data words (and more complicated objects, such as data trees) are a popular model in database theory (e.g. an XML document is conveniently described as a type of data tree) or in the theory of verification. See the survey [25] for more on such automata.

One of the most basic automata models for data words is register automata (introduced in [17] under the name of *finite memory automata*). This is a type of automaton which uses finitely many registers to store some of the atoms that have been seen so far. We will use register automata as our running example of orbit-finite sets.

▶ **Running Example.** *This language in the running example, namely "at most two letters (atoms) appear" is recognised by a register automaton with two registers. The registers are used to store the at most two distinct atoms in the input. Once the two registers are filled up and a fresh third atom appears, the automaton enters a rejecting sink state. The automaton is shown in Figure 1.*

In general, the space of configurations of a register automaton is defined by giving a finite set of control states and a set of register names. A configuration is then a pair: (control state, partial function from register names to atoms). For the precise syntax of the transition relation (and notions of initial and final configurations), we refer to [17]; for the purposes of this paper it suffices to say that the syntax is designed so that transitions depend on the atoms in a way which only uses equality. Register automata, and especially deterministic register automata, are one of the simplest automaton models for data words, e.g. they are not expressive enough to recognise the language "all input letters are different". For more expressive models, see [25].

3 Mild extensions of register automata

To motivate the introduction of orbit-finite and definable sets, which are the topic of this paper, we present three mild requirements for extending the model of register automata. We will then show that these requirements can be met by automata with definable (or equivalently, orbit-finite) descriptions.

3.1 Minimisation

One natural thing to do with (deterministic) register automata is to minimise them. As we will see, the register mechanism is not well suited to this task.

▶ **Running Example.** *Consider the automaton from Figure 1, which uses two registers to recognise words with at most two distinct atoms. This automaton has different configurations after reading inputs ab and ba, because its configuration implicitly remembers which letter was read first. A minimal automaton, on the other hand, should have the same configuration after reading these two inputs, because the language is commutative. In a minimal automaton, the set of reachable configurations should be something like:*

$$\underbrace{\{\epsilon, \perp\}}_{\textit{initial state and rejecting sink}} \quad \cup \quad \underbrace{\{a : a \in \mathbb{A}\}}_{\textit{words that have exactly one atom}} \quad \cup \quad \underbrace{\{\{a,b\} : a \neq b \in \mathbb{A}\}}_{\textit{word that have exactly two atoms}} \quad .$$

Such a configuration set cannot be achieved with the register mechanism.

A workaround for the problems described above would be to allow registers which store unordered sets of atoms. Here is another example language, where the workaround with unordered sets of atoms fails.

▶ **Example 1.** Consider the following language

$$\{wv : w, v \in \mathbb{A}^3 \text{ are equal up to cyclic shift}\}. \tag{2}$$

For this language, the set of configurations of a minimal automaton reachable after reading three letters should be

$$\{ \underbrace{\{abc, bca, cab\}}_{\substack{\text{equivalence class of a}\\ \text{three letter word up}\\ \text{to cyclic shifts}}} : a, b, c \in \mathbb{A}\}.$$

Example 1 and the running example essentially exhaust the possible problems with miminisation: to minimise register automata it suffices to consider a model where each control state has a varying number of registers, and there might be some group acting on the registers (e.g. so that the registers form an unordered set, or can be shifted cyclically, etc.), see [8, Section 6]. The idea to use groups acting on registers dates back to [23].

Why is it so important to minimise automata? A minimal automaton is not just small – with the obvious efficiency advantages – but more importantly it is a canonical representation of its recognised language.

One use for canonical automata is that for the correctness of certain algorithms, it is useful to assume that the input is a canonical automaton. An example is learning algorithms, whose running time bounds and correctness proofs are based on minimal automata, see e.g. [20] for a variant of the Angluin algorithm for register automata. Another example is algorithms which input an automaton and decide if its recognised language is definable in some logic, see [2, 6]; here non-definability is typically equivalent to some kind of forbidden pattern in the canonical automaton.

Another use for canonical automata is that they can be useful when trying to better understand "regularity" for languages over infinite an alphabet. There is a multitude of automata models for infinite alphabets, see [22, 25], most of them with different expressive powers, and one naturally asks [4]: which model defines the "regular languages"? Over finite alphabets, the Myhill-Nerode theorem gives a convincing machine independent characterisation of regular languages in terms of minimisation: a language is regular if and only if it has finitely many equivalence classes in their syntactic congruence (and these equivalence classes form the states of the canonical minimal automaton). Therefore, a natural idea is to study the syntactic congruences of languages such as the one in the running example, and try to find devices which store the information from the syntactic congruence and nothing else. This idea was pursued for monoids (corresponding to a two-sided syntactic congruence) in [6], for deterministic automata (corresponding to a one-sided syntactic congruence, which is different from the two-side one in the presence of infinite alphabets) in [8], and for deterministic timed automata in [3, 10]. In all of these cases the straightforward register mechanism (without group actions and other features) is insufficient to allow minimisation. See also [21] for algorithmic aspects of minimising register automata.

3.2 More general input alphabets

In data words and register automata, the input alphabet is typically assumed to be of the form $\Sigma \times \mathbb{A}$ for some finite set Σ. Why not allow slightly more general input alphabets, such as pairs of atoms $\mathbb{A}^2$ or unordered pairs of atoms $\binom{\mathbb{A}}{2}$? At first sight this seems like a needless generalisation, but it turns out that some interesting theoretical issues appear only for the more general alphabets. As an example [18, Example 2.5], which admittedly goes far beyond register automata because it uses Turing machines, consider the following set:

$$\{\{\{a, b, c\}, \{d, e, f\}\} : a, b, c, d, e, f \in \mathbb{A} \text{ are pairwise different}\}.$$

Suppose that the input alphabet Σ is the above set, and consider the language:

$$\{wv : w, v \in \Sigma^* \text{ are such that } \pi(w) = v \text{ for some permutation of the atoms } \pi\}.$$

In [18] it is shown that the above language witnesses that deterministic and nondeterministic Turing machines (in a suitable generalisation for infinite alphabets using atoms) have different expressive powers, and simpler alphabets (e.g. alphabets which talk about less than six atoms) do not witness this. This result builds on [9], which in turn builds on the seminal Cai-Fürer-Immermann [27] construction. The readers familiar with [27] will recognise the use of six atoms in the alphabet Σ.

3.3 Atoms with more structure than just equality

In our definition of data words and register automata, the atoms $\mathbb{A}$ had equality as the only available structure. Why not allow for more structure? Data words with additional structure, such as a total order, have long been present in the literature on data words. For example, [13] shows that emptiness is decidable for register automata where the input alphabet is $(\mathbb{N}, <)$ and the register operations can compare letters with respect to the order. Another important example is timed automata [1], which can be viewed as a special kind of register automata where the atoms are $(\mathbb{Q}, <, +1)$, see [10] and [14, 15] for the case of pushdown automata. Other examples come from modelling programs interacting with a database, see e.g. [26, 11]; in these applications the atoms might have e.g. an arbitrary graph structure.

4 Definable sets

In Section 2 we described register automata, and in Section 3 we discussed three requirements for a more general model: it should minimise (Section 3.1); it should allow more general input alphabets than only the atoms (Section 3.2); and it should allow more structure on the atoms than just equality (Section 3.3). In this section we present such a model, using the notion of definable sets. The general idea is that definable sets are those that can be constructed using set-builder notation. Before giving the precise definition, consider some examples.

▶ **Example 2.** Here are some of the sets that we have seen so far:

atoms	$\mathbb{A}$
ordered pairs of atoms	$\mathbb{A}^2$
unordered pairs of atoms	$\{\{a,b\} : a,b \in \mathbb{A}\}$
states in the minimal automaton from the running example	$\{\epsilon, \bot\} \cup \{a : a \in \mathbb{A}\} \cup \{\{a,b\} : a \neq b \in \mathbb{A}\}$
triples of atoms modulo cyclic shift	$\{\{abc, bca, cab\} : a,b,c \in \mathbb{A}\}$
input alphabet for a language which witnesses that Turing machines with atoms do not determinise	$\{\{\{a,b,c\},\{d,e,f\} : a,b,c,d,e,f \in \mathbb{A} \text{ are distinct}\}$

The above examples used only equality. In the spirit of Section 3.3, let us consider some examples which assume that the atoms are equipped with structure other than equality.

▶ **Example 3.** Assume that the atoms are equipped with a total order, and assume that 5 is one of the atoms. Here are some examples of sets defined using set builder notation:

atoms smaller than 5	$\{a : a \in \mathbb{A} \text{ with } a < 5\}$
all closed intervals	$\{\{c : c \in \mathbb{A} \text{ with } a \leq c \leq b\} : a,b \in \mathbb{A} \text{ with } a < b\}$

We now give a more formal description of set-builder notation and sets defined by it. A parameter is an underlying logical structure $\mathbb{A}$, i.e. a universe equipped with some relations and functions (equality is for free), which will be referred to as the *atoms*.

▶ **Example 4.** Here are some examples of logical structures that we will use as atoms:

$$\underbrace{\mathbb{A} = (\mathbb{N}, =)}_{\text{equality only}} \qquad \underbrace{\mathbb{A} = (\mathbb{Q}, <)}_{\text{ordered rationals}} \qquad \underbrace{\mathbb{A} = (\mathbb{N}, +)}_{\text{Presburger arithmetic}} \qquad \underbrace{\mathbb{A} = (\mathbb{N}, +, \times)}_{\text{arithmetic}}$$

In the first structure, we write the equality symbol (which is implicitly assumed to be always available) just to underline that it is the only structure available.

Given a logical structure $\mathbb{A}$, we define *set-builder expressions over* $\mathbb{A}$ by induction as follows. Fix some infinite set of variables $\{a, b, c, \ldots\}$, which will be used in the set-builder expressions, and which are intended to range over atoms, i.e. elements of the universe of $\mathbb{A}$. Subexpressions in set-builder expression can have free variables, but in the end we are interested in an outermost expression with no free variables. There are four constructions: each element of $\mathbb{A}$ is an expression (constant expression); each variable is an expression (variable expression); and expressions can be combined using binary union and set comprehension. These constructions are illustrated in Figure 2.

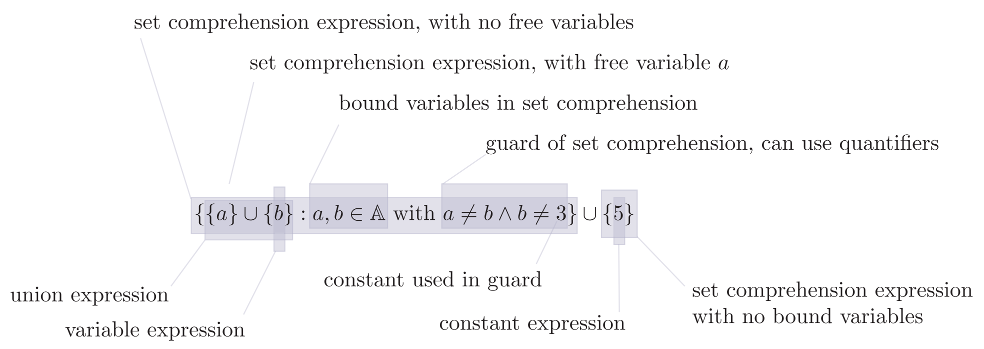

set comprehension expression, with no free variables
set comprehension expression, with free variable a
bound variables in set comprehension
guard of set comprehension, can use quantifiers
$\{\{a\} \cup \{b\} : a, b \in \mathbb{A} \text{ with } a \neq b \wedge b \neq 3\} \cup \{5\}$
union expression
variable expression
constant used in guard
constant expression
set comprehension expression
with no bound variables

Figure 2 A set-builder expression over $\mathbb{A} = (\mathbb{N}, =)$.

The semantics of a set-builder expression is a function which inputs a valuation of its free variables (i.e. a function from the free variables to the universe of $\mathbb{A}$) and outputs an atom, a set, a set of sets, etc. defined in the obvious way. A *definable set over* $\mathbb{A}$ is the semantics of a set-builder expression without free variables. The reader will readily see that all sets mentioned in Example 2 are definable regardless of the choice of $\mathbb{A}$ (see the running example below for an explanation of how pairing and elements such as $\perp$ are encoded) and the sets mentioned in Example 3 are definable as long as the vocabulary of $\mathbb{A}$ contains a binary relation $<$ and the universe contains an element 5.

Our proposed solution to the requirements raised in Section 3 is to consider automata where all components (the input alphabet, the states, the transition relation, the initial and accepting states) are definable over some structure $\mathbb{A}$. Call these *definable automata* over some structure $\mathbb{A}$ [8, 11].

▶ **Running Example.** *Assume that the atoms are $\mathbb{A} = (\mathbb{N}, =)$, i.e. some countably infinite set with equality only. Here is a deterministic automaton which recognises the language from our running example, i.e. words in $\mathbb{A}^*$ with at most two distinct letters. The input alphabet is $\mathbb{A}$, which is clearly a definable set, as defined by the set-builder expression $\{a : a \in \mathbb{A}\}$. The set of states Q is*

$$\{\emptyset, \perp\} \cup \{a : a \in \mathbb{A}\} \cup \{\{a, b\} : a \neq b \in \mathbb{A}\},$$

which is also definable as long as we assume that $\perp$ is syntactic sugar for some definable set like $\{\emptyset\}$. The initial state is $\emptyset$, which is clearly definable, and all states states are final except $\perp$. Here is the set of transitions, which happens to be a total function of type $Q \times \mathbb{A} \to Q$ as required in a deterministic automaton:

$$\begin{aligned}
&\{(\emptyset, a, \{a\}) : a \in \mathbb{A}\} \cup \\
&\{(\{a\}, b, \{a, b\}) : a, b \in \mathbb{A}\} \cup \\
&\{(\{a, b\}, a, \{a, b\}) : a, b \in \mathbb{A}\} \cup \\
&\{(\{a, b\}, c, \perp) : a, b, c \in \mathbb{A} \textit{ with } a \neq b \wedge a \neq c \wedge b \neq c\} \cup \\
&\{(\perp, a, \perp) : a \in \mathbb{A}\}
\end{aligned}$$

The above description uses ordered triples, which are formally not part of the syntax of set-builder expressions. However, triples and other tuples can encoded in sets using Kuratowski pairing:

$$(x, y) \overset{\text{def}}{=} \{x, \{x, y\}\} \qquad (x, y, z) \overset{\text{def}}{=} ((x, y), z).$$

Under the above encoding of triples as sets, it is clear that the transition relation defined above is an example of a definable set.

Based on the above example, it is easy to see that definable automata generalise register automata. What is more, the added flexibility of definable automata is sufficient to satisfy the requirements mentioned in Section 3, in particular minimisation, i.e. for every deterministic definable automaton there exists a unique (up to isomorphism of automata) minimal deterministic definable automaton which recognises the same language [8, Theorem 3.8]. (This minimisation requires an additional assumption on the atoms, called oligomorphism, which will be discussed in Section 6.)

Without further restrictions on the atoms $\mathbb{A}$, definable automata are too general to be useful, as shown in the following example.

▶ **Example 5.** Assume that the atoms are Presburger arithmetic $\mathbb{A} = (\mathbb{N}, +)$. Consider an automaton where the input alphabet has one letter only, say the input alphabet is $\{\emptyset\}$, and the set of states is the definable set of all atoms (i.e. natural numbers). Assume that there is only one final state, namely the natural number 0. Since there is only one input letter, the transition function can be viewed as a function $\mathbb{A} \to \mathbb{A}$. As the transition function, consider the function

$$a \mapsto \begin{cases} a/2 & \text{if } a \text{ is even} \\ 3a+1 & \text{otherwise} \end{cases}$$

which is a definable set as witnessed by the following set-builder expression for its graph:

$$\{(a,b) : a, b \in \mathbb{A} \text{ with } a = b + b\} \quad \cup$$
$$\{(a,b) : a, b \in \mathbb{A} \text{ with } \neg(\exists c\ a = c + c) \wedge b = a + a + a + 1\}.$$

The transition function is based on the famous Collatz conjecture, which in the terminology of this example says: for every choice of initial state, at least one input word is accepted. In fact, all decision problems, such as emptiness or equivalence, are going to be undecidable for this particular choice of atoms (Presburger arithmetic, or even $(\mathbb{N}, <)$ or $(\mathbb{N}, +1)$), which follows from the fact that Minsky machines are a special case of definable automata.

▶ **Example 6.** Assume that the atoms are arithmetic $\mathbb{A} = (\mathbb{N}, +, \times)$, and consider a set-builder expression of the form

$$\{a : a \in \mathbb{A} \text{ with } \varphi(a)\}.$$

The above set is empty if and only if $\varphi(a)$ is unsatisfiable. Since satisfiability in arithmetic is undecidable, it follows that one cannot even decide if a set-builder expression describes an empty set. (This is in contrast with Presburger arithmetic from Example 5, where at least emptiness for set-builder expressions is definable, by virtue of Presburger arithmetic having a decidable theory, see [19, Proposition 2].) Other problems, such as nonemptiness or equivalence for automata are clearly also going to be undecidable when the atoms are arithmetic.

5 Graph reachability for definable sets

In the previous section, we introduced definable sets, and discussed definable automata, i.e. automata where all components are definable sets. In Examples 5 and 6 we argued

```
input V, E, s, t given by set-builder expressions

R := {s}
repeat
 for v in R do
  for w in V do
    if {v,w} in E then
      R += {w}
until R does not grow

if t in R then
 print "reachable"
else
 print "unreachable"
```

Figure 3 A naive algorithm for reachability.

that emptiness for automata is undecidable when the atoms are $(\mathbb{N}, +)$ or $(\mathbb{N}, +, \times)$. In this section, we discuss algorithmic problems like emptiness of automata in more detail. Instead of automata emptiness, we will discuss the essentially equivalent but more fundamental problem of graph reachability. We formalise the problem and present a condition on the atom structure $\mathbb{A}$ which guarantees the graph reachability problem is decidable. This condition is going to be violated for atoms like $(\mathbb{N}, <)$, $(\mathbb{N}, +)$ and $(\mathbb{N}, +, \times)$ but it is going to be satisfied for atoms like $(\mathbb{N}, =)$ or $(\mathbb{Q}, <)$.

The graph reachability problem

For a logical structure $\mathbb{A}$, define *reachability for definable graphs over* $\mathbb{A}$ to be the following decision problem:

- **Input.** A graph (V, E) where V, E are definable sets and two vertices $s, t \in V$.
- **Question.** Is there a path from s to t?

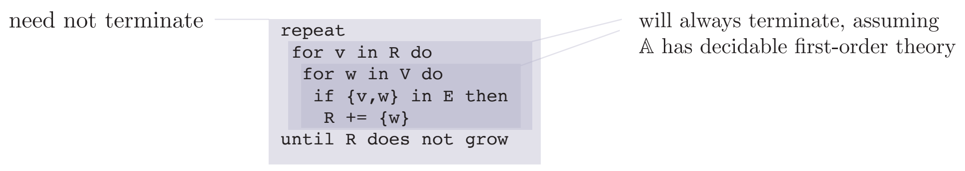
In the decision problem above, the inputs are represented by set-builder expressions. This representation assumes that there is some way of representing the universe of $\mathbb{A}$, which is the case for all atom structures we have discussed so far, where the universe consists of natural or rational numbers. We are mainly interested in decidability and not in the precise complexity of the decision problem.

▶ **Example 7.** Assume that the atoms $\mathbb{A}$ are $(\mathbb{N}, +)$. A valid input for the reachability problem for definable graphs over $\mathbb{A}$ is

$$V = \mathbb{N} \quad E = \{(a, b) : a, b \in \mathbb{A} \text{ with } a = b + b \vee a + 1 = b\} \quad s = 7 \quad t = 2.$$

For this particular input the answer is "yes" because one can go from 7 to 2 by doing several steps of the form "divide by two or add one". For the same reason as discussed in Example 5, i.e. encoding Minsky machines, reachability is undecidable for definable graphs over Presburger arithmetic $(\mathbb{N}, +)$. Actually, the undecidability holds already for atoms $(\mathbb{N}, <)$, since Minsky machines use only increments and decrements on the counters, and these can be defined in first-order logic using only the order. Note that to get undecidability for $(\mathbb{N}, <)$ it is important that formulas in the guards of definable sets are allowed to use quantifiers. If only quantifier-free formulas would be allowed in the guards, then graph reachability for $(\mathbb{N}, <)$ would be decidable [13].

Example 7 shows that the reachability problem is undecidable when the atoms are natural numbers with order, or any other richer structure such as addition or multiplication. What about definable graphs over atoms with equality only, or atoms such as $(\mathbb{Q}, <)$? It turns out that for these atoms, reachability is decidable, and the algorithm is quite straightforward. Define R_n to be the vertices which can be reached from the source by path of at most n edges. A naive algorithm to solve graph reachability (see Figure 3) would be to simply compute the sequence $R_0 \subseteq R_1 \subseteq R_2 \subseteq \cdots$ until it stabilises, and then test if the target vertex is in the stable set. Here is a key property of the program in Figure 3. Assuming that the atoms have decidable first-order theory (which assumption is true Presburger arithmetic $(\mathbb{N}, +)$ but not for general arithmetic $(\mathbb{N}, +, \times)$), then at least each step (both for loops) of the naive algorithm can be carried out in finite time, but the number of steps (repeat loop) might be unbounded:


need not terminate
will always terminate, assuming
$\mathbb{A}$ has decidable first-order theory

```
repeat
 for v in R do
  for w in V do
   if {v,w} in E then
    R += {w}
until R does not grow
```

A more formal statement is in the following lemma.

▶ **Lemma 8.** *Let $\mathbb{A}$ be a logical structure. Suppose that $E \subseteq V \times V$ and $R \subseteq V$ are given by set-builder expressions over $\mathbb{A}$. Then one can compute a set-builder expression representing*

$$R \cup \{v \in V : \{v, w\} \in E \text{ for some } w \in R\}.$$

Assume furthermore that $\mathbb{A}$ has decidable first-order theory. Then one can decide, given R' and R represented using set-builder expressions, if $R' \subseteq R$.

The above lemma can be shown using [19, Proposition 2]; but it is mainly interesting as part of a more general framework, namely programming languages that deal with definable sets. There are currently two programming languages: a functional one [7], with an implementation as a Haskell library [20]; and an imperative one [12] with an implementation as a C++ library [19]. The example reachability program in Figure 3 is based on the imperative approach. The original version of the imperative programming language [12] assumed that the atoms were oligomorphic (see below), but the version from [19] relaxed this assumption to having a decidable first-order theory (which captures additional examples such as Presburger arithmetic). The semantics of both languages are designed so that one does not need to prove results like Lemma 8 by hand; but one can simply use more general principles like the following: every program without `repeat` can be simulated in finite time, assuming the inputs (i.e. program state before) and outputs (i.e. program state after) are represented using set-builder expressions.

As follows from Example 7, decidability of the first-order theory of $\mathbb{A}$ alone does not guarantee that the repeat loop in the program from Firgure 3 will terminate in finitely many steps. Remarkably, when the atoms have equality only, or they are $(\mathbb{Q}, <)$, then the repeat loop necessarily terminates. The reason is that atoms with equality only or $(\mathbb{Q}, <)$ are examples of *oligomorphic* structures. In the next two sections, we discuss oligomorphic structures and prove this termination (Theorem 13). The assumption that the atoms are oligomorphic is what makes the theory of definable sets robustly well behaved.

6 Oligomorphism and orbit-finiteness

In this section we describe what it means for a logical structure to be oligomorphic. The main use of this assumption is that it allows us to use relaxed notion of finiteness for sets, called *orbit-finiteness*; in particular all definable sets will turn out to be orbit-finite.

Definition of oligomorphism

If $\mathbb{A}$ is a logical structure, then an *automorphism* is defined to be any permutation of its universe which is consistent with all the relations and functions in the vocabulary of $\mathbb{A}$. For example, when $\mathbb{A}$ has only equality in its signature then an automorphism is any permutation; while if $\mathbb{A}$ is $(\mathbb{Q}, <)$ then an automorphism is any order-preserving permutation. The structures $(\mathbb{N}, <)$ and $(\mathbb{N}, +)$ have no automorphisms, while $(\mathbb{Z}, <)$ has only translations as automorphisms.

▶ **Definition 9** (Oligomorphism)**.** Consider a logical structure $\mathbb{A}$. We say that two tuples $\bar{a}, \bar{b} \in \mathbb{A}^k$ are *in the same orbit* if there exists an automorphism of $\mathbb{A}$ which takes $\bar{a}$ to $\bar{b}$ componentwise. The structure $\mathbb{A}$ is called *oligomorphic* if for every k, the "same orbit" equivalence relation on $\mathbb{A}^k$ has finitely many equivalence classes.

The notion of oligomorphism made its appearance in model theory in 1959, thanks to a theorem proved independently by Engeler, Ryll-Nardzewski and Svenonius: for a countable structure, being oligomorphic is equivalent to having an ω-categorical theory, see [16, Theorem 7.3.1]. One important corollary of this theorem (and its proof) is that in an oligomorphic structure, every orbit of $\mathbb{A}^k$ can be defined by a formula of first-order logic with k free variables; we will use this property later on.

▶ **Example 10.** Here are some examples and non-examples of oligormophic structures.

- Assume that $\mathbb{A} = (\mathbb{N}, <)$. This structure has no automorphisms, and therefore the "same orbit" equivalence relation on $\mathbb{A}$ has infinitely many equivalence classes (which are singletons). Therefore $\mathbb{A}$ is not oligomorphic.
- Assume that $\mathbb{A} = (\mathbb{Z}, <)$. The automorphisms are translations, and hence the "same orbit" equivalence relation has one equivalence class on $\mathbb{A}$. However, there are infinitely many equivalence classes for $\mathbb{A}^2$, because

  $$(a_1, a_2), (b_1, b_2) \text{ are in the same orbit} \qquad \text{iff} \qquad a_1 - a_2 = b_1 - b_2.$$

  Therefore $\mathbb{A}$ is not oligomorphic.
- Assume that $\mathbb{A} = (\mathbb{N}, =)$, i.e. a countably infinite set with equality only. For every k, tuples in $\mathbb{A}^k$ are in the same orbit if and only if they have the same equality types, and there are finitely many equality types. Therefore, $\mathbb{A}$ is oligomorphic.
- Assume that $\mathbb{A} = (\mathbb{Q}, <)$. It is not difficult to see that for every k, tuples $\bar{a}, \bar{b}$ satisfy $\bar{a} \sim \bar{b}$ if and only if they have the same order types, and there are finitely many order types. Therefore, $\mathbb{A}$ is oligomorphic.
- Every structure over a finite vocabulary without functions that is homogeneous is oligomorphic. This covers Fraïssé limits of classes of finite relational structures, such as the previous two items, or the countable random graph. For more on homogeneous structures, the Fraïssé limit and the random graph, see [16, Section 7].

Orbit-finiteness

Oligomorphic structures turn out to be exceptionally well-behaved with respect to definable sets. The reason for this is that in an oligomorphic structure one can distinguish a relaxed notion of finiteness, called *orbit-finiteness*, which is well behaved and can be used to prove results such as the termination of the algorithm Figure 3.

To define orbit-finiteness, we need to introduce a little set terminology. Fix a logical structure $\mathbb{A}$. Define the *cumulative hierarchy over* $\mathbb{A}$ to be objects that are built using atoms (i.e. elements of $\mathbb{A}$) and set brackets in a well-founded way. More precisely, for an ordinal number we define the rank α objects of the the cumulative hierarchy as follows: for $\alpha = 0$ the rank α objects are the empty set and every atom; for $\alpha > 0$ the rank α objects sets whose elements are objects of rank $< \alpha$. The cumulative hierarchy is the union of all ranks. For example, every definable set over $\mathbb{A}$ is in the cumulative hierarchy, but there are many more sets in the cumulative hierarchy (e.g. the cumulative hierarchy is closed under taking arbitrary subsets, unlike definable sets).

If π is an automorphism of $\mathbb{A}$ (actually, any function of type $\mathbb{A} \to \mathbb{A}$), then one can apply π to an object x in the cumulative hierarchy, by simply applying to the atoms that are used in x; the result is a new object $\pi(x)$ in the cumulative hierarchy with the same rank.

▶ **Definition 11** (Finite support and orbit-finiteness)**.** Let $\mathbb{A}$ be a logical structure. For x in the cumulative hierarchy over $\mathbb{A}$, we say that x is *finitely supported* if there exists a tuple of atoms $\bar{a}$ (which is called a *support* of x) such that

$$\pi(\bar{a}) = \sigma(\bar{a}) \quad \text{implies} \quad \pi(y) = \sigma(y) \qquad \text{for every automorphisms } \pi, \sigma \text{ of } \mathbb{A}.$$

We say that x is *orbit-finite* if the following equivalence relation on elements of x has finitely many equivalence classes:

$$y \sim z \quad \text{if } y = \pi(z) \qquad \text{for some automorphism } \pi \text{ of } \mathbb{A}.$$

The notion of finite supports is fundamental to set theories such as Fraenkel-Mostowski, and more recently to the theory of nominal sets [24]. To the author's best knowledge, the notion of orbit-finiteness was first explicitly proposed in [6]. In the terminology of the above definition, $\mathbb{A}$ is oligomorphic if and only if $\mathbb{A}^k$ is orbit-finite for every k. The following theorem shows that, under the assumption that the atoms are oligomorphic, then the notion of orbit-finiteness is well behaved and definable sets are the same as sets which are hereditarily orbit-finite. For a proof of the following theorem, see [5].

▶ **Theorem 12.** *Let $\mathbb{A}$ be a logical structure which is oligomorphic, and let x be in the cumulative hierarchy over $\mathbb{A}$.*

1. *x is orbit-finite if and only if for every tuple of atoms $\bar{a}$, the following equivalence relation has finitely many equivalence classes:*

$$y \sim_{\bar{a}} z \quad \textit{if } y = \pi(z) \textit{ for some automorphism } \pi \textit{ of } \mathbb{A} \textit{ which satisfies } \pi(\bar{a}) = \bar{a}.$$

2. *x is definable if and only if it is hereditarily orbit-finite (i.e. x finitely supported and orbit-finite, and both these properties also hold for elements of x, their elements, and so on recursively).*

The equivalence in item 2 of the above theorem is very useful for computation: in some cases it is more convenient to use definable sets (e.g. to represent sets in a finite way), and in some cases it is more convenient to use orbit-finite sets (e.g. in termination proofs). We now show an example of this usefulness.

Termination of the reachability algorithm

In Figure 3 from Section 5, we presented a naive reachability algorithm for definable graphs. We also remarked that, as long as the atoms have decidable first-order theory, then each iteration of the repeat loop could be evaluated in finite time. We are now ready to show that, as long as the atoms are oligomorphic, then the repeat loop will be be performed finitely often.

▶ **Theorem 13.** *Assume that $\mathbb{A}$ is oligomorphic. Then the reachability algorithm in Figure 3 terminates on every input.*

Proof. Assume that the input to is a graph with distinguished source and target, and that each part of the input (vertices, edges, source and target) is a definable set represented by a set-builder expression. By item 2 in Theorem 12, each part of the input has a finite support. By combining these finite supports into a single tuple, we can conclude there is some tuple of atoms $\bar{a}$ which supports all parts of the input, i.e. $\bar{a}$ supports the vertices, the edges and the source and target vertices. For $n \in \{0, 1, \ldots\}$ define R_n to be the vertices which are reachable from the source vertex by a path with at most n edges. Recall the equivalence relation $\sim_{\bar{a}}$ mentioned in item 1 of Theorem 12. Using induction on n and the assumption that $\bar{a}$ supports the source vertex and the edges, we get the following observation:

(*) each set R_n is a union of equivalence classes of the $\sim_{\bar{a}}$.

By item 1 of Theorem 12, there are finitely many equivalence classes. By (*) each step of the reachability algorithm adds some new equivalence classes, and therefore the algorithm must terminate in a finite number of steps. ◀

The goal of the above proof was to illustrate the interplay between definability (as a way of representing infinite inputs) and orbit-finiteness (as a way of proving termination for algorithms). Other examples of this interplay include algorithms which uses a fixpoint computation, such as the standard algorithm for emptiness of a context-free grammar (which works if the grammar is definable over oligomorphic atoms) or the Moore minimisation algorithm for deterministic automata (which works for definable automata over oligomorphic atoms).

References

1 Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

2 Michael Benedikt, Clemens Ley, and Gabriele Puppis. Automata vs. logics on data words. In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, pages 110–124, 2010. doi:10.1007/978-3-642-15205-4_12.

3 Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. In *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, Buenos Aires, Argentina, May 17-20, 2010*, 2010. URL: http://ceur-ws.org/Vol-619/paper11.pdf.

4 Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5):702–715, 2010. doi:10.1016/j.tcs.2009.10.009.

5 Mikołaj Bojańczyk. Atom book [online]. https://www.mimuw.edu.pl/~bojan/paper/atom-book.

6 Mikołaj Bojańczyk. Nominal monoids. *Theory Comput. Syst.*, 53(2):194–222, 2013. `doi:10.1007/s00224-013-9464-1`.

7 Mikołaj Bojańczyk, Laurent Braud, Bartek Klin, and Slawomir Lasota. Towards nominal computation. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 401–412, 2012. `doi:10.1145/2103656.2103704`.

8 Mikołaj Bojańczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3), 2014. `doi:10.2168/LMCS-10(3:4)2014`.

9 Mikołaj Bojańczyk, Bartek Klin, Slawomir Lasota, and Szymon Toruńczyk. Turing machines with atoms. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 183–192, 2013. `doi:10.1109/LICS.2013.24`.

10 Mikołaj Bojańczyk and Slawomir Lasota. A machine-independent characterization of timed languages. In *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 92–103, 2012. `doi:10.1007/978-3-642-31585-5_12`.

11 Mikołaj Bojańczyk, Luc Segoufin, and Szymon Toruńczyk. Verification of database-driven systems via amalgamation. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA – June 22-27, 2013*, pages 63–74, 2013. `doi:10.1145/2463664.2465228`.

12 Mikołaj Bojańczyk and Szymon Toruńczyk. Imperative programming in sets with atoms. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, pages 4–15, 2012. `doi:10.4230/LIPIcs.FSTTCS.2012.4`.

13 Karlis Cerans. Deciding properties of integral relational automata. In *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*, pages 35–46, 1994. `doi:10.1007/3-540-58201-0_56`.

14 Lorenzo Clemente and Slawomir Lasota. Reachability analysis of first-order definable pushdown systems. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, pages 244–259, 2015. `doi:10.4230/LIPIcs.CSL.2015.244`.

15 Lorenzo Clemente and Slawomir Lasota. Timed pushdown automata revisited. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 738–749, 2015. `doi:10.1109/LICS.2015.73`.

16 W. Hodges. *Model Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1993.

17 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. `doi:10.1016/0304-3975(94)90242-9`.

18 Bartek Klin, Slawomir Lasota, Joanna Ochremiak, and Szymon Toruńczyk. Turing machines with atoms, constraint satisfaction problems, and descriptive complexity. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS'14, Vienna, Austria, July 14-18, 2014*, pages 58:1–58:10, 2014. `doi:10.1145/2603088.2603135`.

19 Eryk Kopczynski and Szymon Toruńczyk. LOIS: syntax and semantics. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 586–598, 2017. URL: `http://dl.acm.org/citation.cfm?id=3009876`.

20 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In *Proceedings of the 44th ACM SIGPLAN Symposium on*

Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017, pages 613–625, 2017. URL: `http://dl.acm.org/citation.cfm?id=3009879`.

21 Andrzej S. Murawski, Steven J. Ramsay, and Nikos Tzevelekos. Bisimilarity in fresh-register automata. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 156–167, 2015. `doi:10.1109/LICS.2015.24`.

22 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004. `doi:10.1145/1013560.1013562`.

23 M. Pistore. *History Dependent Automata.* PhD thesis, University of Pisa, 1999.

24 A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2013.

25 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, pages 41–57, 2006. `doi:10.1007/11874683_3`.

26 Victor Vianu. Automatic verification of database-driven systems: a new frontier. In *Intl. Conf. on Database Theory (ICDT)*, pages 1–13, 2009. `doi:10.1145/1514894.1514896`.

27 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. `doi:10.1007/BF01305232`.