Computation in sets with atoms

Mikołaj Bojańczyk

October 10, 2016

Contents

I Data words and their automata 2

1 Data words and register automata 3
1.1 Nondeterministic register automata. 3
1.2 Emptiness and universality for register automata 7

2 Alternating automata 9

Part I
Data words and their automata

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

1 Data words and register automata

Define a *data word* over a finite alphabet Σ to be a word where every position has a label in $\Sigma \times \mathbb{A}$, where $\mathbb{A}$ is a fixed infinite set. The first coordinate is called the *label* and the second coordinate is called the *data value*. The idea is that we can test labels explicitly by asking questions like

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

but we can only test the data value for equality e.g. ask

> Do the third and fifth letters have the same data value?

In the later parts of this book, we will try to formalise what it means to only test data values for equality, but for now the intuitive understanding should be sufficient.

Example 1. By abuse of notation, we assume that a word over the alphabet $\mathbb{A}$ is also a data word, which uses no labels. Here are some examples of languages of data words, in all of these examples we use no labels:

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

□

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

1.1 Nondeterministic register automata.

The syntax of a *nondeterministic register automaton* consists of:

- a finite alphabet Σ of *labels*;
- a finite set Q of *control states*;
- a finite set R of *register names*;
- an *initial state* $q_0 \in Q$ and a set of *accepting states* $F \subseteq Q$;
- a *transition relation*

$$\delta \subseteq \underbrace{Q \times (\mathbb{A} \cup \{\bot\})^R}_{\text{configurations}} \times \underbrace{\Sigma \times \mathbb{A}}_{\text{input}} \times \underbrace{Q \times (\mathbb{A} \cup \{\bot\})^R}_{\text{configurations}} \tag{1}$$

  subject to an equivariance condition described below.

The automaton is used to accept or reject data words where the alphabet is $\Sigma \times \mathbb{A}$. After processing part of the input, the automaton keeps track of a *configuration*, which is defined to be a control state plus a register valuation (i.e. a partial function from register names to data values). Initially, the configuration consists of the initial state and a completely undefined register valuation. The configuration is then updated according to the transition relation δ, and the automaton accepts if at the end of the word the control state belongs to the accepting set.

How to describe the transition relation? Since the space of configurations is infinite, the transition relation must satisfy some constraints, otherwise it cannot be represented in a finite way. We choose the following constraint, called *equivariance*: the transition relation can only compare data values with respect to equality. Equivariance can be formalized in two different ways below.

Semantic equivariance. A bijection $\pi : \mathbb{A} \to \mathbb{A}$ on the data values can be applied to configurations in the natural way, and therefore also to triples in the transition relation δ (the states and undefined values are not affected, only the data values). We say that δ is *semantically equivariant* if

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

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

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

1. the control state in the source (respectively, target) configuration is $q \in Q$;
2. the label in the input letter is $a \in \Sigma$;
3. the data value is undefined in register $r \in R$ of the source configuration (respectively, target configuration);
4. the data value in the input letter equals the contents of register $r \in R$ in the source configuration (respectively, target configuration);
5. the data value in register $r \in R$ of the source configuration (respectively, target configuration) equals the data value in register $s \in S$ of the source configuration (respectively, target configuration).

Lemma 1.1 *Semantics and syntactic equivariance are the same.*

Proof
It is not difficult to see that semantically equivariant subsets of the set (1) are closed

under boolean combinations. Since the bijections of data values do not affect satisfaction of the constraints 1-5 used in the definition of syntactic equivariance, it follows that syntactic equivariance implies semantic equivariance.

We now show that semantic implies syntactic. Define an *orbit of transitions* to be a subset of the set (1) which is semantically equivariant and which is minimal for that property with respect to inclusion.

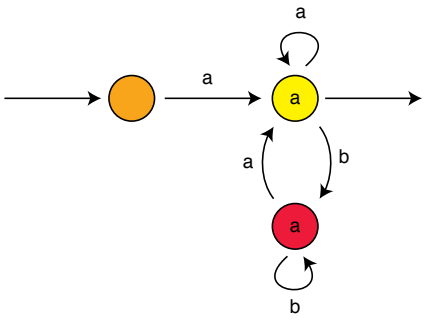
Claim 1.1.1 *Every orbit of transitions is syntactically equivariant.*

Proof (of Claim)
Because an orbit of transitions is uniquely defined by its states, which registers are undefined, and what is the equality type of the tuple of data values in the defined registers. All of this information can be expressed using the constraints 1-5 in the definition of syntactic equivariance. □

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

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

Example 2. Here is a deterministic register automaton which recognises language 1 from Example 1, i.e. the words in $\mathbb{A}^*$ where the first and last data values are equal. The automaton stores the first data value in its register, and then toggles between accepting or rejecting states depending on whether the input agrees with the register. Here is a picture:



The above picture should be interpreted as follows. There are three states, standing for the three colored circles, with initial and final states depicted by the dangling

arrow. Since there is one register, a configuration consists of a state and a possibly empty data value. Such configurations can be found in the picture above. For every pair of distinct atoms $a \neq b$, we add a transition from the above picture to the automaton. Note how every arrow in the picture corresponds to an orbit of transitions.

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

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

Exercise 2. For languages of data words one can also define the Myhill-Nerode relation, as used in minimisation of deterministic automata. Show a language of data words where every deterministic register automaton distinguishes (by its configuration) some two words which are Myhill-Nerode equivalent.

Exercise 3. Show there can be two noni

Exercise 4. Show that a nondeterministic register automaton can recognise language 2 from Example 1, but a deterministic one cannot.

Exercise 5. Call a nondeterministic register automaton *guessing* if there exists a transition $t \in \delta$ such that some data value in the target register valuation appears neither in the source register valuation nor in the input. Given an example of language that needs guessing to be recognised.

A corollary of the above two exercises is that:

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

Furthermore, the two nondeterministic variants are not closed under complementation, and the first two models are not closed under reverse.

Exercise 6. Consider the two-way variant of register automata, where the head of the automaton can move both ways. Show that a deterministic two-way register automaton can recognise the language:

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

Exercise 7. Possibly using unproved conjectures from complexity theory, show that two-way register automata cannot be determinised.

Exercise 8. Show that for two-way (even nondeterministic) register automaton $\mathcal{A}$ with one register, if the labels are Σ then the following language is regular:

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

1.2 Emptiness and universality for register automata

In this section we discuss two standard decision problems: emptiness (does the automaton accept at least one input word) and universality (does the automaton accept all input words). When talking about decidability, we assume that the transition function is represented according to the syntactic equivariance condition.

Theorem 1.2 *Emptiness is decidable for nondeterministic register automata.*

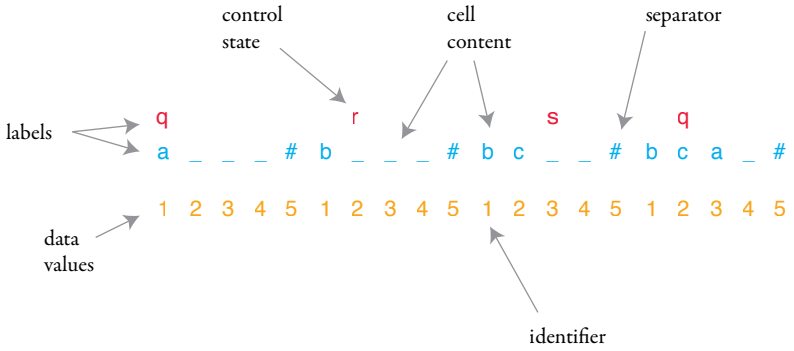
Proof
This proof just sketches the decidability argument, the complexity is discussed in Exercise 9. Define an *orbit of configurations* to be a set of configurations that is closed under bijections of data values. As in Lemma 1.1, an orbit of configurations can be defined by saying what is the states, which are the defined registers, and what is the equality type on the data values stored in the defined registers. Such a description takes finite space to store, and there are finitely many possible descriptions. The key observation that being in the same orbit of configurations is a congruence with respect to transitions, i.e. if two configurations are in the same orbit then both are reachable or both are unreachable. The algorithm for nonemptiness computes the orbits of reachable configurations. Initially, we have the equality type of the unique initial configuration, which can be easily computed. If we have the equality type of some configuration, we can easily compute the equality types of all configurations reachable from it in one step; thus finishing the description of the algorithm. □

Exercise 9. The complexity of the emptiness problem depends on how the size $|\mathcal{A}|$ of the input automaton is measured. Show that that emptiness is:

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

Theorem 1.3 *Universality is undecidable for nondeterministic register automata.*

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



Each letter encodes a single cell in a single configuration. The word represents a sequence of configurations, padded with blanks so that they all have the same length, and separated by a letter #. The labels are used to store the contents of the cell (blue), plus the control state (red) of the head if the head happens to be over that cell. Finally, each cell gets a unique identifier, a data value (orange). The following claim shows that the lahting problem reduces to universality of nondeterministic register automata, thus proving the theorem.

Claim 1.3.1 *There is a nondeterministic register automaton which accepts a data word if and only if it is* not *an encoding of an accepting run of the Turing machine.*

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

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

   The first condition can be tested using one register, the second condition using two registers.
2. There is a mistake between two consecutive configurations. Assuming the identifiers are chosen correctly, this can be tested using only one register, to tell which cells correspond to which ones in the following configuration.
3. The first configuration is not initial, or the last configuration is not accepting. For this, no registers are needed.

□ □

Exercise 10. The undecidability proof in Theorem 1.3 used automata with two register but no guessing (as in Exercise 5). Show that, in the presence of guessing, universality remains undecidable even with one register.

2 Alternating automata

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

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

We define the semantics of the automaton using *bags*, where a bag is defined to be a set (not necessarily finite) of configurations. For every input letter a (consisting of a label and a data value), we define a binary relation $\xrightarrow{a}$ on bags, such that $C \xrightarrow{a} D$ holds if:

- for every configuration $c \in C$ with an existential state, the bag D contains some configuration d such that (c, a, d) is a transition;
- for every configuration $c \in C$ with a universal state, the bag D contains all configurations d such that (c, a, d) is a transition.

A data word $a_1 \cdots a_n$ is accepted if there exists a run

$$\text{initial bag } = C_0 \xrightarrow{a_1} C_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} C_n \in \text{accepting bags}$$

where the initial bag is defined to be the singleton of the initial configuration, and an accepting bag is defined to be any bag that contains only configurations with accepting states. We define $\to$ to be the union of all relations $\xrightarrow{a}$, ranging over all letters a. In terms of this notation, an alternating automaton is nonempty if an accepting bag is reachable from the initial bag via a finite number of steps of $\to$.

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

Exercise 12. Find a language that is recognised by an alternating register automaton with guessing, but not by any alternating register automaton without guessing.

Exercise 13. Show that every two-way nondeterministic register automaton can be simulated by an alternating register automaton (with guessing and ϵ-transitions.)

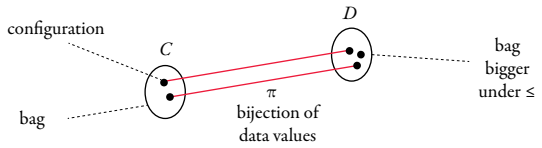
Exercise 14. Show that nondeterministic two-way register automata are strictly weaker than alternating register automata with guessing.

The emptiness problem. Nondeterministic register automata are the special case of alternating register automata where all states are existential. By Exercise 11, the emptiness and universality problems for alternating register automata are essentially the same problem, which is undecidable by Theorem 1.3. Furthermore, by the remarks in Exercise 10, this undecidability is true already for two registers and no guessing, or even one register with guessing. That is the limit of undecidability:

Theorem 2.1 *Emptiness is decidable for one register alternating automata without guessing.*

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

As in the definition of semantic equivariance from Section 1, bijections of data values can be applied to configurations and to bags of configurations. The following order on bags is the key to our proof: we write $C \leq D$ if there is some bijection of the data values π such that $C \subseteq \pi(D)$. Here is a picture:



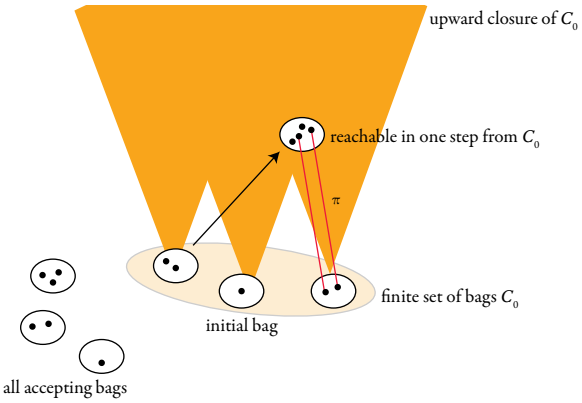
This is easily seen to be a quasi-ordering, i.e. a transitive and reflexive relation. Call a set of bags *upward closed* if it is upward closed with respect to this quasi-order. The *upward closure* of a set of bags is the least upward closed set of bags that contains it.

The following lemma gives the semi-decidability of emptiness.

Lemma 2.2 (Finite Emptiness Witness Lemma) *An alternating one register automaton without guessing accepts no words if and only if there exists some finite set of bags $\mathcal{C}_0$ which is a witness in the following sense:*

1. *$\mathcal{C}_0$ contains no accepting bags;*
2. *the initial bag is in $\mathcal{C}_0$;*
3. *if $C \to D$ and $C \in \mathcal{C}_0$, then D is in the upward closure of $\mathcal{C}_0$.*

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



The idea is that the orange area, i.e. the upward closure of C_0, is a trap in the sense that no transition can leave the orange area. It is straightforward to see that existence of a witness is semi-decidable: guess the set C_0 and check the three conditions. For the third condition, it is useful that there is no guessing, since the relation $\to$ has finite outdegree without guessing (without guessing, the condition would still be verifiable). Therefore, the Finite Emptiness Witness Lemma completes the proof of Theorem 2.1. It remains to prove the lemma, which we do in the rest of this section. Fix an alternating one register automaton.

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

Well quasi-order. We say that a quasi-order is a *well quasi-order* if it is well-founded (no infinite strictly decreasing chains) and has no infinite antichains. The technique of well quasi-orders, as used in the following proof, is one of the most common methods of proving decidable properties for systems with infinitely many configurations.

Exercise 15. Show that a quasi-order is a well-quasi-order if and only if every infinite sequence contains a monotone subsequence, i.e. one where $i \leq j$ implies $x_i \leq x_j$

Exercise 16. Show that for every dimension $d \in \{1, 2, \ldots\}$, the set $\mathbb{N}^d$ is a well quasi-order with respect to the coordinatewise ordering.

Lemma 2.3 *The relation $\leq$ on finite bags is a well quasi-order.*

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

- for each state $q \in Q$, does the bag contain a configuration with state q and an undefined register;

- for each set of states $P \subseteq Q$, what is the number of data values d such that the bag contains a configuration with state p and data value d if and only if $q \in P$.

A profile can be seen as a binary vector indexed by Q plus a vector of natural numbers indexed by subsets of Q. The profile mapping takes incomparable pairs (of bags under $\leq$) to incomparable pairs (of profiles seen as vectors ordered coordinatewise). Therefore, the profile mapping takes infinite antichains to infinite antichains. Since there are no infinite anitichains in the latter space by Exercise 16, it follows that there are no infinite antichains in the former space. □

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

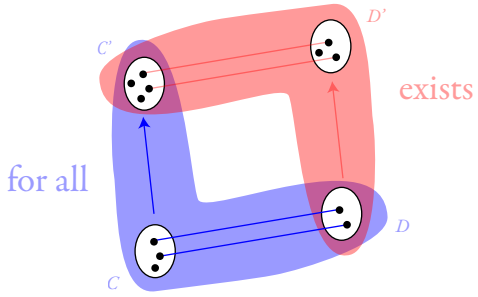
Lemma 2.4 *Every upward closed set of bags is the upward closure of a finite set of bags.*

Proof
By well-foundedness, every upward closed set is the upward closure of its minimal elements. The minimal elements form an antichain, and hence there can only be finitely many of them (up to renamings). □

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

Lemma 2.5 *The relation $\leq$ on bags is compatible with $\to$ in following sense: for every transition $C \to C'$ and every $D \leq C$ there exists some $D' \leq C'$ with $D \to D'$.*

Proof
Here is the picture of compatibility:



Because $\to$ is closed under permutations of data values, and also closed under making the first argument a smaller bag. □

Using compatibility, we can prove the right-to-left implication in the Finite Emptiness Witness lemma. Suppose then that $\mathcal{C}_0$ is a finite set of bags which satisfies the three conditions in the lemma. Let $\mathcal{C}$ be the upward closure of $\mathcal{C}_0$. Since accepting bags are downward closed, condition 1 in the lemma implies that $\mathcal{C}$ contains no accepting bags. Condition 2 implies that $\mathcal{C}$ contains the initial bag. Finally, compatibility ensures that if $C \in \mathcal{C}$ and $C \to C'$ then also $C' \in \mathcal{C}$. In other words, $\mathcal{C}$ is an invariant which witnesses that the initial bag cannot reach any accepting bag.

Finding the finite emptiness witness. To complete the proof of the Finite Emptiness Witness lemma, we need to prove the left-to-right implication, i.e. find the finite witness $\mathcal{C}_0$ in any alternating automaton that accepts no words.

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

Lemma 2.6 *$\mathcal{R}$ is downward closed.*

Proof
Take any finite path

$$C_n \to C_{n-1} \to \cdots \to C_1 \in \text{accepting bags.}$$

To prove the lemma, we prove by induction on n that if $D \leq C_n$ then $D \in \mathcal{R}$. The induction base is the fact that the set of accepting bags is downward closed. For the induction step, we use the compatibility established in Lemma 2.5. $\square$

Stated differently, the above lemma says that the complement of $\mathcal{R}$ is upward closed. Apply Lemma 2.4 to this complement, yielding a finite set of bags $\mathcal{C}_0$. We will prove that $\mathcal{C}_0$ is a witness in the sense of the Finite Emptiness Witness Lemma. By definition of $\mathcal{C}_0$, every bag C satisfies

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

To prove that $\mathcal{C}_0$ is a witness, let us check the three conditions from the Finite Emptiness Witness Lemma. Clearly there can be no accepting bags in $\mathcal{C}_0$, because an accepting bag can reach an accepting bag in zero steps. By assumption that the automaton is empty, the initial bag cannot reach an accepting bag, and hence the initial bag is in the upward closure of $\mathcal{C}_0$. Only the empty bag is smaller than the initial bag, and the empty bag is accepting, hence the initial bag must actually be in $\mathcal{C}_0$, and not only in its upward closure. Finally, let us prove the third condition. The upward closure of $\mathcal{C}_0$ is closed under taking a step of $\to$, since if C cannot reach an accepting bag, then the same is true for anything reachable from C. This implies the third condition. This completes the proof of the Finite Emptiness Witness Lemma and of Theorem 2.1.

Exercise 17. Define the Higman ordering on words to be the relation of not necessarily connected substrings. Show that there is an infinite antichain for the following order on $\mathbb{A}^*$:

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

Exercise 18. Show that there is a language $L \subseteq \mathbb{A}^*$ that is upward closed under the Higman order, but is not recognised by a nondeterministic register automaton.

References