Computation in sets with atoms

Mikołaj Bojańczyk

October 3, 2016

Contents

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. consecutive data values are different

□

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.

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

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

Exercise 3. 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.

In particular, deterministic register automata are strictly weaker than nondeterministic ones, and nondeterministic ones are not closed under complement.

Exercise 4. 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 5. Possibly using unproved conjectures from complexity theory, show that two-way register automata cannot be determinised.

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.*

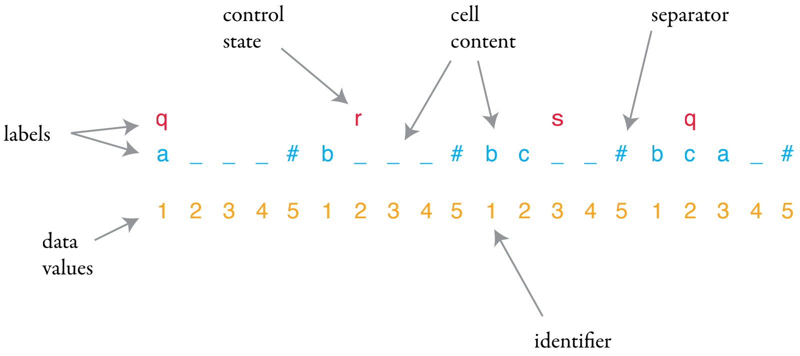
Proof
This proof just sketches the decidability argument, the complexity is discussed in Exercise 6. Define an *orbit of configurations* to be a set of configurations that is closed under bijections of data values. As in Lemma 1.1, an orbit of configurations can be defined by saying what is the states, 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 6. 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 7. The undecidability proof in Theorem 1.3 used automata with two register but no guessing (as in Exercise 3). Show that, in the presence of guessing, universality remains undecidable even with one register.