# Forms of Determinism for Automata*

Thomas Colcombet[1]

1 **Liafa / CNRS / Université Paris Diderot**
**Case 7014, F-75205 Paris Cedex 13, France**
thomas.colcombet@liafa.jussieu.fr

## Abstract

We survey in this paper some variants of the notion of determinism, refining the spectrum between non-determinism and determinism. We present unambiguous automata, strongly unambiguous automata, prophetic automata, guidable automata, and history-deterministic automata. We instantiate these various notions for finite words, infinite words, finite trees, infinite trees, data languages, and cost functions. The main results are underlined and some open problems proposed.

**1998 ACM Subject Classification** F.1.1 Models of Computation, F.1.2 Modes of Computation, F.4.3 Formal Languages

**Keywords and phrases** Automata, determinism, unambiguity, words, infinite trees

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2012.1

## 1 Introduction

The relationship between deterministic and non-deterministic machines plays a central role for many questions in computer science, in particular, when it comes to complexity. In this survey, we investigate these notions and related ones in the context of classical automata and more recent automata models.

The role of determinism for automata is different from its role for general Turing machines. The reason for that is that automata are more considered as data structures than as programs. Indeed, automata are meant to be compact representations of (possibly infinite) languages. They are meant to be transformed, composed and used in *decision procedures*. General Turing machines cannot play such a role since by the theorem of Rice only trivial properties concerning them can be decided. For this essential reason, determinism or non-determinism are considered under a very different angle for automata compared to, say, complexity theory.

Since the seminal works of Myhill [23] and Rabin and Scott [26] deterministic and non-deterministic finite-state automata are known to define the same classes of languages over finite words. It is also known that they have very different properties. For instance, complementing a deterministic automaton is a straighforward operation (linear time and space), while it is exponential for general non-deterministic automata in the sense that there exist languages accepted by an automaton which when complemented requires an exponentially bigger automaton for being accepted. For this reason it is meaningful to carefully distinguish the two models as far as decision procedures are involved.

As automata theory developed over time, new types of automata were introduced such as automata over infinite words, finite trees, infinite trees, etc... The relationship between

* This work has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement 259454.

deterministic and non-deterministic machines became richer at each step, reflecting the intricate mathematical principles underlying these objects.

In this survey, the following objects will be encountered:

- Standard word automata.
- Register automata (similar to the model of finite-memory automata of Kaminsky and Francez). Such automata read words over an infinite alphabet containing data values (such as natural numbers), can store such data values in a finite number of registers, and compare them for equality with the current data value.
A typical example is the language: 'some letter appears twice in the word'.
- Automata over infinite words (of length $\omega$). Due to the infinite length of the words, such automata do not use accepting states. The accepting runs are defined in terms of the states that appear infinitely often. Prominent examples are Büchi and Müller automata.
A typical example is the language: 'the letter $a$ appears finitely many times'.
- Automata over finite trees. Such automata can branch; thus computations proceed 'concurrently' in several subtrees of the tree.
A typical example the language is: 'every branch contains an occurrence of letter $b$'.
- Automata over infinite trees. As with finite trees such automata are branching, but additionaly the run should satisfy an accepting condition on each of its branches, similarly to the case of infinite words.
A typical example is the language: 'the letter $a$ occurs at infinitely many nodes'.
- Cost automata. These automata are used for recognising functions from words to non-negative integers. Such automata use several counters that can be incremented and reset. The values taken by the counters during all runs are aggregated into a value which is output.
A typical example is the function: 'the number of occurrences of letter $a$'.

We present in this survey different notions of determinism and non-determinism. It is important to understand that these variants are not tied to a specific form of automata. The different models are used to exhibit the differences between the notions, and also to illustrate why each of these notions is interesting in practice. Thus, we will introduce several restrictions to non-deterministic automata which are not as restrictive as determinism, but still allow us to derive some useful properties. There are several motivations for considering such variants.

*Complexity.* A first reason for preferring deterministic automata is that some operations are easier to perform with deterministic automata. This is the case for universality testing (PTIME against PSPACE in general over words), or complement (linear blowup against exponential blowup over words). What kind of notions, less restrictive than determinism yields the same complexity results? The notions of unambiguity and strong unambiguity provide some answers.

*Decidability.* A second reason is that one works with a class of automata which does not admit determinisation (i.e., deterministic automata are strictly weaker than non-deterministic ones) and that the non-deterministic automata do not enjoy good properties. This is the case with register automata. Non-deterministic register automata have an undecidable universality problem while deterministic ones have a decidable universality problem. One is interested in this situation in introducing forms of determinism, not as restrictive as the general determinism, and for which the universality problem remains decidable. It is relevant in this case to consider the class of strongly unambiguous automata as a class of automata of intermediate expressive power which retain good algorithmic properties.

*Structure.* Another reason for preferring deterministic automata to non-deterministic automata is that they have a sharper structure for some advanced decision procedures. This happens, for instance, when one is interested in the characterisation of some sub-classes of the regular languages, *i.e.* problems such as, 'is it possible to define a language given in input in some fixed fragment of logic?'. Deterministic automata, being more constrained, exhibit more behaviour in their structure, and for this reason are better indicated. We will see that prophetic automata over infinite words are good in this respect for characterising future temporal logics. In a similar way, guidable automata are suitable for analysing the fix-point structure of languages of infinite trees.

*Games.* A last reason is that one wants to use a specific property of deterministic automata, which is not enjoyed by non-deterministic automata. This is in particular the case for composing with games. The idea behind this is that in a game (say, two players, turn based), the first player is not aware of the future of the play since it depends of the choices of the opponent. Hence, he has to decide his moves solely based on the past of the play. The determinism of an automaton has a similar flavour since it means making decisions solely based on the current state, and hence without guessing any information concerning the future. This similarity can be used for safely composing games with automata. This is an important technique in automata theory. We will see that in this context the notion of history-determinism (also called 'good for solving games') can be used as a replacement to determinism. A motivating example is the theory of cost functions where deterministic automata are strictly weaker than general automata while history-deterministic ones are as expressive.

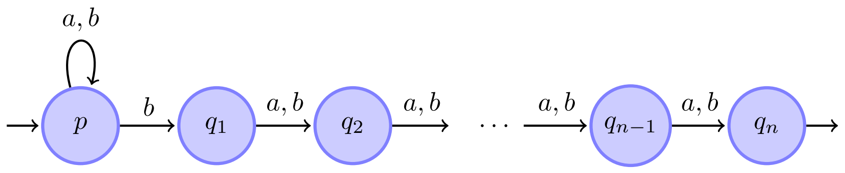
Through these motivations, many different variations around determinism have emerged: unambiguity, strong unambiguity, prophetic automata, guidability and history-determinism. Their presentation is organised as follows.

In Section 2 we recall the definitions of word automata as well as some standard results for them. In Section 3 we consider several notions of unambiguity. More specifically, we introduce unambiguity and strong unambiguity, and we study these notions over word automata, register automata, infinite word automata (in particular prophetic Büchi automata) and infinite tree automata (in particular the inherent ambiguity result). In Section 4 we present guidable automata, a notion related to top-down determinism over finite and infinite trees. In Section 5 we introduce history-deterministic automata, also described as those 'good for solving games'. We define the notion and give results over finite words. We then present the motivating example of cost functions. We also explain why these automata behave nicely in the context of games.

## 2 Word automata

A *(non-deterministic) (finite word) automaton* $(Q, \mathbb{A}, I, \Delta, F)$ consists of a finite set of *states* $Q$, an alphabet $\mathbb{A}$, a set of *initial states* $I \subseteq Q$, a *transition relation* $\Delta \subseteq Q \times \mathbb{A} \times Q$, and a set of *final states* $F \subseteq Q$. A *run* of the automaton over a word $a_1 \dots a_n$ is a sequence $q_0, \dots, q_n$ of states such that $(q_{i-1}, a_i, q_i) \in \Delta$ for all $i = 1 \dots n$. The run is *initial* if $q_0 \in I$ and it is final if $q_n \in F$. A run is *accepting* if it is both initial and final. A word is *accepted* if there is an accepting run of the automaton over it. The set of accepted words is called the *language* of the automaton. One also says that this is an automaton *for* the language. We will use the same terminology for extended forms of automata.

Consider for instance the (non-deterministic) automaton with input alphabet $\{a, b\}$ depicted as follows:



Following tradition, a transition $(p, a, q)$ is denoted by an edge from $p$ to $q$ labeled $a$. Multiple transitions going between the same states are denoted with commas separating letters. Initial states have an incoming arrow without origin, and final states have a similar outgoing arrow. The illustrated automaton is for the language 'the $n$th letter from the end is a $b$'.

An automaton is called *deterministic* if for all states $p$ and all letters $a$ there is at most one transition of the form $(p, a, q)$. Additionally it is called complete if for all states $p$ and letter $a$ there exists a transition of the form $(p, a, q)$. The symmetric notion is co-determinism. An automaton is *co-deterministic* if for all states $q$ and all letters $a$ there is at most one transition of the form $(p, a, q)$. The automaton above is co-deterministic, but not deterministic. The *size* of an automaton is number of its states.

▶ **Theorem 1.** *Word automata of size $n$ can be transformed into deterministic and complete automata of size at most $2^n$ for the same language. This bound is tight.*

In particular the automaton in the example above requires an exponential number of states to be made deterministic.

The closure properties of automata are of particular importance. Indeed languages accepted by automata are closed under union (using a disjoint union construction) and intersection (using a product construction). They are also closed under reversal. The *reversal* operation consists of inverting the order of letters in all words. This is obtained by reversing the transitions and exchanging initial and final states. All these operations are polynomial (in particular the resulting automaton has polynomial size). However, the complement operation requires an exponential blowup in the number of states; in general one cannot do better than putting the automaton in deterministic form and then complementing.

The situation slightly differs for deterministic (and complete) automata. Intersection and union are still polynomial operations (this time union requires a product construction rather than a disjoint union). This time complement is easy (one just has to complement the set of final states), but reversal requires an exponential blowup.

## 3 Unambiguous forms of automata

The notion guaranteeing the unicity of the runs for each input is classically called unambiguity. In this section, we develop several notions of unambiguity, classical unambiguity in Section 3.1, and then strong unambiguity in Section 3.2. We continue our study with the motivating example of register automata (also called finite-memory automata) in Section 3.3. We conclude our description of this notion with the study of unambiguous automata over infinite words, and in particular with prophetic Büchi automata in Section 3.4 and over infinite trees in Section 3.5.

### 3.1 Unambiguity

A key consequence of the notion of determinism is that if an input is accepted, then there exists one and one only run which witnesses this acceptance. This yields the first notion of unambiguity, which can be applied to many models of automata.

▶ **Definition 2.** A (non-deterministic) automaton is *unambiguous* if on every input there is at most one accepting run.

Of course all deterministic automata are unambiguous (in any model of computation). Thus, on every model of computation enjoying determinisation, unambiguous automata are not less expressive than non-deterministic automata. The converse is not true, and some unambiguous automata are not deterministic, even on finite words. Consider for instance the automaton from the previous section recognising the language of finite words such that 'the $n$'th letter from the end is a $b$'. This automaton is co-deterministic, and as a consequence unambiguous, but it is not deterministic.

One can also remark that it is easy to decide (in polynomial time) whether or not an automaton is unambiguous. For this, given a non-deterministic automaton $\mathcal{A}$, one constructs another non-deterministic automaton $\mathcal{B}$ which accepts an input if and only if the original automaton had two distinct accepting runs over this input. This new automaton is essentially obtained by a product construction. It simulates concurrently on the input two instances of the original automaton. An extra gadget bit is used to detect that the two simultaneous runs differ at some moment. Of course, this new automaton accepts some word if and only if the original automaton is ambiguous. Since emptiness is decidable, we obtain that unambiguity is decidable (even in polynomial time). This argument is generic, and can be applied to most forms of automata. This is in particular true for all models of automata encountered in this document, and even beyond.

As one may expect, unambiguous automata have some algorithmic advantages compared to general non-deterministic automata. For instance, the problem of universality (whether an automaton accepts all inputs), the problem of inclusion and the problem of equivalence are PSPACE complete for languages described by non-deterministic automata [29]. The situation is different for unambiguous automata.

▶ **Theorem 3** (Hunt and Stearns [28]). *Over finite words, the problems of universality, inclusion and equivalence of languages accepted by unambiguous automata are polynomial.*

Such results suggest that the notion of unambiguity could be used in decision procedures in replacement for determinism. However, even if succinct unambiguous automata exist (more succinct than equivalent deterministic automata), one does not know how to efficiently construct them:

▶ **Question 1.** *Is is possible to efficiently disambiguate non-deterministic automata? In other words, does there exist a construction which, given a non-deterministic automaton, produces an unambiguous automaton of minimal size (or close to the minimal) for the same language, in time polynomial (in the sum of sizes of the input and the output)?*

*Closure properties.* Some closure properties are elementary to perform on the languages accepted by unambiguous automata. This is, in particular, the case for closure under union and intersection. In order to achieve this, in either case one simply performs a product construction. In contrast to deterministic automata, the languages of unambiguous automata are closed under reversal in linear time (the procedure for non-deterministic automata naturally preserves unambiguity). Note, once more, that these approaches are very generic, and that these closure properties are enjoyed by most models of unambiguous automata, and in particular all the models of automata appearing in this survey.

Another interesting operation on automata is *the cascade* (it is an operation on automata, not on languages). The principle of cascading two automata is to run the first automaton on the input, and then run a second automaton with the run of the first automaton as input.

This operation is easy to perform using again a product construction. It can be used to perform composition of transducers, and also generalises the construction of intersection and union. It is very useful for things such as composing operators in a temporal logic. What is interesting is that the cascade of non-deterministic automata yields a non-deterministic automaton, the cascade of deterministic automata yields a deterministic automaton, and the cascade of unambiguous automata yields an unambiguous automaton. The cascade operation is a generic operation which can be applied to most models of automata, including all the models of automata appearing in this survey. Cascade products preserve unambiguity in all cases. Over words, this operation allows us to easily construct complex unambiguous automata by cascading deterministic and co-deterministic automata.

However, it is erroneous to think that unambiguous automata are 'easy' to complement in the manner of deterministic automata. Indeed, the notion of unambiguity does only refer to how the automaton accepts an input, and it says nothing about rejecting an input.

► **Question 2.** *What is the state complexity of complementing an unambiguous finite word automaton? In particular, is it the case that the complement of the language of an unambiguous word automaton is accepted by an unambigous word automaton of polynomial size?*

Remark in particular that determinising an unambiguous automaton may yield an exponential blowup of the number of states, and thus it does not answer Question 2.

## 3.2 Strong unambiguity

To circumvent the problem of closure under complement, another notion of unambiguity can be used. We call it *strong unambiguity.*

► **Definition 4.** A *strongly unambiguous* (and complete) automaton is an automaton which can be used both as an unambiguous automaton for the language and as an unambiguous automaton for its complement.

There are several ways to implement this definition. The simplest way is to consider the couple of two unambiguous automata, one for the language, the other for the complement. Some more compact presentations can be used, for instance using a single automaton and (in the case of finite word automata) two couples of sets of initial states and final states, one to be used for accepting the language, the other for accepting the complement. It is easy to see that in terms of size, the two codings are similar up to a linear factor in size.

Note that testing if a pair of automata represent a strongly unambiguous automaton is doable in polynomial time, at least over words. Indeed, it involves checking the unambiguity of both automata, the emptiness of their intersection, and the universality of the union. All these tests can be performed in polynomial time, the last one using Theorem 3.

*Closure properties.* Strongly unambiguous automata are naturally closed under union, intersection, and cascade (using product constructions), and under complement. Strongly unambiguous automata are also closed under reversal in an easy way. Once more these properties are generic, and apply to most models of automata, including all of those encountered in this survey.

A natural question is the relationship between strong unambiguity and unambiguity in the case of finite words, and in particular the possible 'equivalence' of the two notions. The question is equivalent to Question 2.

► **Question 3.** *Is it possible to transform an unambiguous automaton into a strongly unambiguous one of polynomial size?*

The two notions of unambiguity are interesting for finite word automata for complexity reasons. However, it is when dealing with other models of automata that the subtlety of the notion is revealed. We consider in the remainder of the section the case of register automata in Section 3.3, the case of Büchi automata over infinite words in Section 3.4 and finally the case of infinite trees in Section 3.5.

## 3.3 The motivating example of register automata

In this section, we show that notions of unambiguity, here strong unambiguity, can be used for defining classes of languages with nice properties. This requires a change of model, and going beyond classes of standard regular languages of finite words. We consider here the case of register automata. In this case, the notions of deterministic, unambiguous and non-deterministic automata have different expressive power. In particular the non-deterministic model is too strong (e.g., it is not closed under complement, and universality is undecidable). In this context it is interesting to consider the class of strongly unambiguous register automata; it generalises deterministic register automata, has good closure properties (in particular under complement and reversal contrary to deterministic automata), but does not inherit the bad decidability properties of the general class of non-deterministic register automata (universality is decidable).

A *data word* is a word over the alphabet $\mathbb{A} \times \mathbb{D}$ where $\mathbb{A}$ is a standard finite alphabet of *letters*, and $\mathbb{D}$ is an infinite alphabet of *data values*. A *register automaton* (variant of the model introduced by Kaminski and Francez [14] studied by Kaminski and Zeitlin [15] under the name 'finite-register automata with non-deterministic reassignement') has a finite number of *states*, and a finite number of *registers*, say $1, \dots, k$. Registers can store data values. A configuration of the automaton is a tuple $(p, d_1, \dots, d_k)$ consisting of a state and a data value for each register. At the beginning of the run, the automaton starts with any data value stored in its register (non-deterministic choice). Then the run proceeds as for a non-deterministic automaton whith reading the input, and at each time the transition describes how to use the data values of the register. Transitions are of the form $(p, a, u_1, \dots, u_k, q)$ where $p$ and $q$ are states, $a$ is a letter (from $\mathbb{A}$), and $u_1, \dots, u_k$ are sequences of actions taken from $\{=, \neq, \texttt{guess}\}$. Such a transition goes from state $p$ to state $q$ when reading letter $a$ and performs the sequence of actions on each counter as follows: action $=$ checks that the value of the register coincides with the data value read currently (otherwise it is not allowed to take the transition), the action $\neq$ is similar, but checks that the data values are different, and finally, the action `guess` possibly changes the value of the register, choosing the new value non-deterministically.

Let us give two examples of register automata. Here we assume that $\mathbb{A}$ contains only one letter, and we omit it wen drawing transitions. Furthermore, we consider only automata with one register. Thus transitions are simply labeled with sequence of actions.

$$\begin{tikzcd}
{} \arrow[r] & p \arrow[r, "\texttt{guess};="] & q \arrow[loop above, "\neq"] \arrow[r, "="] & r \arrow[r] & {}
\end{tikzcd}
\qquad
\begin{tikzcd}
{} \arrow[r] & p \arrow[loop above] \arrow[r, "\texttt{guess};="] & q \arrow[loop above] \arrow[r, "="] & r \arrow[loop above] \arrow[r] & {}
\end{tikzcd}$$

The first one guesses a value and immediately checks that the guess coincides with the left-most data value in the word. Then, it compares the value of this register with all other positions in the word until it finds another occurrence of this value. In order to reach the final state, the value stored in the register during the first step should be encountered a second time in the word. It accepts the language 'the first data value reappears'. The second automaton chooses non-deterministically a position in the word, stores its value, then chooses non-deterministically a second position in the word, and checks that the value is equal to its register. It accepts the language 'some data value appears twice'. The reader can check easily that the complement of the first language, namely the language 'the first data value does not occur elsewhere' is also accepted by a register automaton (set the set of final states to $\{p, q\}$). However, it is also well known that the complement of the second language, namely the language 'all data values are different' is not accepted by any register automaton.

Using direct adaptations of the constructions for classical word automata, one can show that the languages accepted by register automata are closed under union (using disjoint union of the automata) and intersection (using a product construction, with a set of registers containing all registers of the automata), and that their emptiness is decidable. The register automata are also closed under reversal of languages (this is not the case for the original model of Kaminski and Francez). However, on the negative side, register automata are not closed under complement, and their universality problem is undecidable. Overall, these automata seem to be too expressive, and it is relevant to search for sub-models which would enjoy a better balance between expressiveness and decidability.

A first worthwhile restriction to consider is deterministic register automata. A *deterministic register automaton* is a register automaton for which it is possible to construct in a unique way a run, step by step, while reading the input from left to right. For this, one requires that (a) there is exactly one initial state, (b) for each pair of distinct transitions reading the same letter from the same state, there exists a register whose action starts with $=$ in one of the transitions, and starts with $\neq$ in the other transition, (c) each `guess` action is immediately followed with an $=$ action, and (d) by convention the register share a common data value at the beginning of the run[1], say $\perp \notin \mathbb{D}$. The sequence of actions '`guess`; $=$' is called `store` since it amounts to collect the current data value, and store it in the register[2]. For instance, the first example of a register automaton above is in fact a deterministic register automaton. Deterministic register automata are closed under union, intersection, complement and cascade, and emptiness and universality are decidable. However, these automata lack the closure under reversal.

The notion of a *strongly unambiguous register automaton* is the one presented for finite word automata, instantiated for register automata. As in the usual case, the notion of strong unambiguity is semantic (in terms of existence of runs), but it is nevertheless a decidable property using the argument from the previous sections. Also, using the generic constructions described above, one easily shows that strongly unambiguous register automata are closed under intersection, union, complement and reversal. Furthermore, emptiness and universality are decidable. Hence, this model seems a good compromise between expressivity (it is more general than deterministic and co-deterministic automata) and decidability (most key properties of closure and decidability hold).

However, there is another way to define a class with similar properties. Consider the class of languages that are accepted by a register automaton and such that the complement

[1] Strictly speaking, this is not a restriction of the syntax of non-deterministic register automata, nevertheless, one easily shows that such deterministic register automata, up to minor modifications, can be seen as special cases of non-deterministic ones.

[2] The original finite-memory automata of Kaminski and Francez are allowed to use the action `store` but not the action `guess`. This restriction yields a weaker model, which is in particular not closed under language reversal. For instance the language 'the last data value does not appear before' cannot be recognised even with a non-deterministic finite-memory automaton. Kaminski and Zeitlin have studied more recently the model of 'finite-memory automata with non-deterministic reassignment' [15]. This model corresponds to register automata, where 'non-deterministic reassignment' stands for '`guess`'.

is also accepted by a register automaton. Since non-deterministic register automata are closed under intersection, union and reversal, the same holds for this class. Furthermore, the emptiness and universality of this class are decidable. However, this class has one important defect: it is not possible to decide if a pair $(\mathcal{A}, \mathcal{B})$ of register automata represents a valid language of the class. Indeed, it would require to decide if the union of the languages of $\mathcal{A}$ and $\mathcal{B}$ contains all words. This is the universality problem, which is undecidable for register automata.

The conjecture is that the two classes coincide.

▶ **Conjecture 4.** *The class of data languages that are accepted by register automata and whose complements are accepted by register automata coincide with the class of data languages accepted by strongly unambiguous automata.*

One direction is straightforward. Indeed, since the class of strongly unambiguous data languages is closed under complement, every language accepted by a strongly unambiguous automaton is accepted by a register automaton as well as its complement.

Something which may seem surprising is that the above conjecture may even be effective. This may happen in particular if it is proved as a consequence of the following conjecture.

▶ **Conjecture 5.** *Given two data languages $K, L$ accepted by register automata and of empty intersection, there exists effectively a language $U$ accepted by a strongly unambiguous register automaton such that $K \subseteq U$ and $L \subseteq \complement U$.*

This conjecture would imply Conjecture 4. Indeed, if $K$ and $L$ are complement of each other and both are accepted by a register automaton, then $U = K$ where $U$ is the strongly unambiguous language from Conjecture 5. However, even if $U$ is known, this would not help for deciding whether $K = \complement L$.

Another conjecture regarding this class reads as follows.

▶ **Conjecture 6.** *Strongly unambiguous register automata are equivalent to cascades of deterministic and co-deterministic register automata.*

Let us remark that the notion of cascade requires a bit more precision in this case. Indeed, cascading automata means executing a second automaton with a run of the first automaton as input. However, the question arises whether the second automaton, when reading the run of the first automaton, has access to the content of the registers of the first automaton during its run (as if these values were data-values written on the word). In the above conjecture, we assume that cascade allows this.

We terminate with register automata by remarking that Conjectures 4 and 5 make sense for the natural extension register automata to finite trees, while Conjecture 6 would have no obvious meaning in this context.

## 3.4 The case of infinite words: prophetic Büchi automata

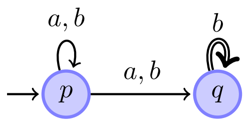
We have seen that it was straightforward for (finite) word automata to transform them into unambiguous ones. Indeed, it is sufficient to determinise. The situation for infinite words is different. We are dealing here with infinite words of length $\omega$; *i.e.* words of the form $a_1a_2\ldots$, with $a_1, a_2, \ldots$ in a given finite alphabet $\mathbb{A}$. The set of words over the alphabet $\mathbb{A}$ is denoted $\mathbb{A}^\omega$.

A (non-deterministic) *automaton over infinite words*[3] $(Q, \mathbb{A}, \Delta, \mathcal{W})$ has a set of initial states $I$, an input alphabet $\mathbb{A}$, a transition relation $\Delta$, and an accepting condition $\mathcal{W}$. The *accepting condition* $(\mathbb{C}, W)$ consists of an alphabet $\mathbb{C}$ and a language of infinite words $W \subseteq \mathbb{A}C^{\omega}$. The *transition relation* is a subset of $Q \times \mathbb{A} \times \mathbb{C} \times Q$. A *run* of such an automaton over a word $a_1 a_2 \dots$ is a sequence:

$$(q_0, a_1, c_1, q_1), (q_1, a_2, c_2, q_2), \dots$$

of transitions in $\Delta$. It is *accepting* if $q_0 \in I$ and $c_1 c_2 \cdots \in W$. A word is *accepted* if there exists an accepting run over it. The set of accepted words is the *language of the automaton.* An automaton over infinite words is called *deterministic* (and complete) if for all input letter $a$ and all states $p$, there exists one and only one transition of the form $(p, a, c, q)$.

Automata over infinite words are classified according to their accepting condition. A *Büchi automaton* is an automaton over finite words using the accepting condition $\mathcal{B} = (\{1, 2\}, W_B)$ where $W_B$ is the language of infinite words that contain infinitely many occurrences of letter 2. Transitions labelled with the condition letter 2 are called *Büchi transitions.* Said differently, a run in a Büchi automaton is accepting if it visits infinitely often some Büchi transition. Consider for instance the following Büchi automaton (where we take the convention that Büchi transitions are drawn as double arrows):



This automaton accepts the language of infinite words over $\{a, b\}$ that contain finitely many occurrences of the letter $a$. Indeed, for this automaton to accept an infinite word, the accepting run has to witness infinitely many occurrences of the Büchi transition from $q$ to $q$. This means that the word has to eventually contain only $b$'s. Conversely, given an infinite word with finitely many $b$'s, one can construct an accepting run by starting in state $p$, and going to $q$ when the last occurrence of letter $a$ has been seen. This Büchi automaton is not deterministic. It is well known that this language is not accepted by any deterministic Büchi automaton (one interesting aspect of Büchi's seminal work was to be able to complement Büchi automata without determinising them [3]).

What McNaughton has later shown is that, using a stronger acceptance condition, it is possible to determinise Büchi automata. A *Müller accepting* condition $(C, M)$ is such that $M$ is a Boolean combination of properties of the form 'the letter $c$ occurs infinitely often' [22].

▶ **Theorem 5** (McNaughton [20]). *A language of infinite words is accepted by a Büchi automaton if and only if it is accepted by a deterministic Müller automaton.*

A consequence of Theorem 5 is that Müller automata can be made deterministic (one has to remark that Müller automata are easy to transform into Büchi automata), and hence unambiguous. However, the question remains for Büchi automata: is it possible to transform Büchi automata into equivalent unambiguous ones? Prophetic automata positively answer this question. Indeed, every Büchi automaton can be made prophetic (Theorem 7), and every prophetic automaton is strongly unambiguous.

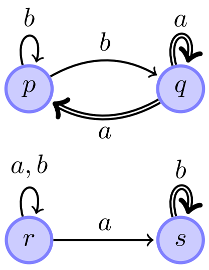
[3] We adopt here a terminology where accepting conditions label transitions rather than states. This is not the standard definition. However the various models are equivalent, and all the content of this section would be valid for the other model.

Prophetic automata are a strong form of co-deterministic automata; *i.e.*, automata such that for all states $q$ and all letters $a$ there exists at most one transition of the form $(p, a, c, q)$. However, co-determinism (as studied in [1]) is not sufficient for guaranteeing unambiguity. Indeed, since the word is infinite, there can be several infinite runs over the same input even if it is co-deterministic (the proof that co-determinism entails unambiguity for finite words involves an induction on the length of the word which cannot be performed for infinite words).

▶ **Definition 6.** A Büchi automaton is *prophetic*[4] if it is universal and unambiguous when all its states are set to be initial.

So in particular if the prophetic automaton is such that all states are the origin of an accepting run over some input (useless states are removed), the unambiguity assumption entails that the automaton is co-deterministic. By definition, prophetic automata are strongly unambiguous.

For instance, consider the following Büchi automaton (without initial states):



For all words (over $\{a, b\}$) this automaton has exactly one accepting run (assuming all states are initial). Indeed, if a word has infinitely many $a$'s, it is accepted only from state $p$ if the first letter is $b$, and only from state $q$ if its first letter is $a$. The word $b^\omega$ is accepted only from state $s$, and the words with finitely many $a$'s (at least one) are accepted only from state $r$. Hence this automaton is prophetic. If one sets $\{r, s\}$ to be the set of initial states, the automaton accepts the language of words with finitely many occurrences of letter $a$. As explained above, this language is accepted by a non-deterministic Büchi automaton, but by no deterministic Büchi automaton. This prophetic automaton shows that the language is nevertheless accepted by a strongly unambiguous Büchi automaton.

Though the subject of this survey is not to dig further in this notion of prophetic automata, let us show that it has some quite intriguing properties. In particular, these automata are in some sense 'minimal in the long run'. This can be illustrated by the following remark: 'there is exactly one prophetic automaton (that has only productive states) for the full langage over the unary alphabet $\{a\}$, and this automaton has only one state'. This property is very different from what we are used to with deterministic automata. Indeed, there are infinitely many deterministic Büchi automata for this language.

The key theorem concerning prophetic Büchi automata is that such automata exist for all regular languages of infinite words.

▶ **Theorem 7** (Michel and Carton [5]). *For all regular languages of infinite words, there exists an equivalent prophetic automaton.*

[4] The reader should be aware that prophetic are called 'unambiguous automata' in [5]. We avoid this terminology here which would be strongly ambiguous.

A direct consequence of this result is that, in particular, Büchi automata can be made strongly unambiguous.

This is not the only interest of prophetic automata. Another motivation is the use of prophetic automata for the characterisation and decidability of classes of regular languages. In general, one is interested in questions of the form:

Does a given regular language $L$ belong to a fixed sub-class of regular languages?

This kind of questions was initiated with the famous seminal work of McNaughton-Papert-Schützenberger [27, 21] characterising in an effective way the languages of finite words that can be defined in first-order logic. There is now a rich variety of results of this form. The decision procedure for such results usually starts by assuming that the language $L$ is represented in a specific suitable form. For instance, to characterise first-order logic, one should start with a minimal deterministic automaton, or with the syntactic monoid. For deciding the star-height one starts with another form of automata [16, 7]. We will also see in Section 4 guidable automata which were introduced for attacking the Mostowski hierarchy problem. In general, it is an interesting high level question to be able to relate the properties of the class of regular languages one aims at with the suitable form of representation needed for the language.

Prophetic automata enter this picture when one is interested in characterising the regular languages of infinite words corresponding to some temporal logics, and more specifically, if one wants to answer the following question:

Is it possible to define a regular language of infinite words $L$ in temporal logic using only the operator `neXt` (*resp.* `Until`)?

This problem is decidable [24], and the representation of the language used in the proof and in the decision procedure is a prophetic automaton for $L$.

There is an intuitive reason for that: the temporal logics in consideration only use future modalities, and thus nicely cohabit with prophetic automata. One way to witness this nice cohabitation is to remark that a formula of such a temporal logic is naturally encoded as a cascade of elementary prophetic automata representing the operators of the logic. This would not hold if the logic could use past operators. Nor if automata would be deterministic rather than prophetic.

## 3.5 The case of infinite trees and inherent ambiguity

We have only encountered word automata so far. The theory of automata over trees is also very rich (see e.g., [10] for an extensive presentation).

For simplicity, we will use a slightly restrictive notion of trees. This is solely for simplifying the technical aspects of the discussion. We will consider binary *tree*s, with labels (from a finite alphabet, say $\mathbb{A}$) only on inner nodes (*i.e.*, nodes that are not leaves). Each node has either two children, or it is a leaf. One denotes by $\varepsilon$ the root node of the tree, and given an inner node $x$, one denotes by $x0$ its left child and by $x1$ its right child. The *label* at inner node $x$ in tree $t$ is denoted $t(x)$. One finally denotes by *nodes*$(t)$ the set of nodes of the tree $t$. A tree is finite if it has finitely many nodes, otherwise it is called infinite. We call the *infinite binary tree* the only infinite tree without leaves over a unary alphabet.

A *finite tree automaton* $(Q, \mathbb{A}, I, \Delta, F)$ has a finite set of *states* $Q$, an *input alphabet* $\mathbb{A}$, a set of initial states $I$, a *transition relation* $\Delta \subseteq Q \times \mathbb{A} \times Q \times Q$, and a set of *final states* $F$. A *run* $(\rho, \delta)$ over a tree $t$ is a mapping $\rho$ from nodes of $t$ to $Q$ and a mapping $\delta$ from inner nodes to $\Delta$ such that for every inner node $x$, $\delta(x) = (\rho(x), t(a), \rho(x0), \rho(x1))$. The run is *initial* if $\rho(\varepsilon)$ is initial. The run is *final* if $t(x) \in F$ for all leaves $x$. The run is *accepting* if it is both initial and final. Note that $\rho$ is sufficient by itself for defining the run. The mapping $\delta$ will become necessary when we consider infinite trees.

Over finite words, determinism and co-determinism played a symmetric role, and so had exactly the same properties. This is not the case over trees, due to the absence of similar symmetry. For this reason, one distinguishes two different notions of determinism. A tree automaton is *bottom up deterministic* (and complete) if (a) it has exactly only one final state and (b) for all states $q, r$ and letter $a$, there is exactly one transition of the form $(p, a, q, r)$. A tree automaton is *top-down deterministic* (and complete) if (a) it has only one initial state, and (b) for all state $p$ and all letters $a$ there exists exactly one transition of the form $(p, a, q, r)$. A top-down deterministic automaton has exactly one initial run on each input. A bottom-up deterministic automaton has exactly one final run on each *finite* input (there can be several runs over infinite trees).

▶ **Theorem 8.** *Automata over finite trees can be made bottom-up deterministic, but not top-down deterministic in general.*

The fact that every tree automaton can be made bottom-up deterministic is obtained by a powerset-construction similar to the determinisation of word automata. For the impossibility of obtaining top-down deterministic automata it is witnessed by the following very simple tree language:

$$\left\{ \begin{array}{ccc} & \mathrm{a} & \\ / & & \backslash \\ \mathrm{a} & & \mathrm{b} \end{array} , \begin{array}{ccc} & \mathrm{a} & \\ / & & \backslash \\ \mathrm{b} & & \mathrm{a} \end{array} \right\} .$$

Indeed, a top-down deterministic automaton, when processing the tree, has to guess whether the $b$-subtree will occur in the left or the right sub-tree. This contradicts determinism. Formally, the proof shows that if a top-down deterministic automaton accepts the above trees, then it accepts also the 'three-$a$'s' tree.

Over finite trees it is possible to provide a strongly unambiguous automaton by using bottom-up deterministic automata according to Theorem 8. But what about infinite trees?

Formally an infinite tree automaton is defined as for finite trees, but is further enhanced with an accepting condition $\mathcal{W} = (\mathbb{C}, W)$ (as defined in Section 3.4), and each transition is of the form $(p, a, c, q, r)$ where $c$ is an extra component from $\mathbb{C}$. A run $\rho$ over an infinite tree is now called *final* if all its leaves belong to $F$, and furthermore for every infinite branch $x_0, x_1, \dots$, the word $\rho(x_0)\rho(x_1)\cdots \in W$. A run is *accepting* if it is both initial and final. A Müller automaton for infinite trees is such an automaton which uses a Müller acceptance condition. A language of infinite trees is called *regular* if it is the language of some Müller automaton over infinite trees. Regular languages of infinite trees are known to be closed under union, intersection, and complement (a famous result of Rabin [25]).

For instance, consider the Büchi automaton for infinite trees over the alphabet $\{a, b\}$ that has states $\{p, q\}$, initial state $p$, final state $q$, and transitions:

$$\{\, (p, b, 1, p, \top), (p, b, 1, \top, p), (p, a, 1, \top, \top), (\top, a, 2, \top, \top), (\top, b, 2, \top, \top) \,\} .$$

The language of this automaton is the language 'there is an occurrence of the letter $a$'. Observe first that all trees are accepted from state $\top$. The state $p$ should be understood as 'searching for letter $a$'. Given an input, the automaton has to start at root position in state $p$; *i.e.*, it 'searches for letter $a$'. Then the state $p$ is propagated as long as a letter $b$ is encountered (since only the two first transitions can be used from $p$ while reading $b$). Each time the automaton has to decide to pursue its search to the left subtree (first transition) or to the second subtree (second transition). When finding an occurrence of $a$, there is nothing more to check and the third transition is used. Finally, the Büchi condition guarantees that the two first transitions are not used infinitely often along an infinite branch; *i.e.*, it prevents the automaton from searching, never finding, but still accepting the input infinite tree.

▶ **Theorem 9** (Carayol, Löding, Niwinski and Walukiewicz [4]). *The language of infinite trees 'there is an occurrence of the letter $a$' is* intrinsically ambiguous*; i.e., there exists no unambiguous automaton for this language.*

The proof of this result is by reduction to the result of the non existence of a regular choice function over the infinite binary tree.

In order to state the result concerning choice one needs to define the regular relations over trees. Firstly, given $U_1, \dots, U_k$ subsets of the nodes of $t$, denote by $t[U_1, \dots, U_k]$ the tree $t$ in which the label of node $x$ is set to $(t(x), b_1(x), \dots, b_k(x))$ where $b_i(x) = 1$ if $x \in U_i$ and 0 otherwise, for all $i$. In other word $t[U_1, \dots, U_k]$ is the tree $t$ enhanced with the characteristic functions of the sets $U_1, \dots, U_k$ as extra labelling information. Call now a relation $R \subseteq (2^{nodes(t)})^k$ *regular* if there is an automaton which accepts $t[U_1, \dots, U_k]$ if and only if $(U_1, \dots, U_k) \in R$. In other words, there exists an automaton which, when given the various input sets as extra labelling of the tree, is able to check whether the relation holds between them. We will use this definition for a function from sets of nodes to nodes. Such a function can be seen in the usual way as a relation between sets and singleton sets.

A *choice function* on a tree $t$ is a function which maps each non-empty subset of nodes of $t$ to one of its elements; it chooses an element in the set given as input.

▶ **Theorem 10** (Gurevich and Shelah [12], see [4] for a simple proof). *There is no regular choice function on the infinite binary tree*[5].

The phenomenon raised by this theorem is that an automaton is able to test that a set of nodes is non-empty, but it is unable to pinpoint in a unique way an element witnessing this non-emptiness. The reduction essentially says that if an unambiguous automaton for the language 'there exists an occurrence of letter $a$' existed, then this automaton would have to identify in some way a specific occurrence of the letter $a$ as witness[6]. Thus it could be seen as choosing some occurrence of $a$ inside the set of occurrences of $a$. In other word, this automaton could be used, after some slight modifications, as an implementation of a choice function. This would contradict Theorem 10.

## 4 Guidable automata

A guidable automaton is an automaton over finite or infinite trees which has some properties that resemble those of a top-down deterministic automaton. Recall that top-down deterministic automata are not as expressive as general automata, already over finite trees. The notion was introduced in [8] and is deeply studied in [17].

Intuitively, a guidable automaton is an automaton which requires the minimum quantity of information for resolving its non-determinism. This informal definition is stated as 'given any

[5] The theorem of Gurevich and Shelah is stated in terms of definability in monadic second-order logic. Thanks to the result of Rabin [25], the statements are equivalent.

[6] For instance, the above automaton for the language 'there is an occurrence of letter $a$' witnesses letter $a$, namely when it uses the transition $(p, a, \top, \top)$. But this automaton is ambiguous, and for this reason there may be several positions on which this transition could be used.

accepting run of any automaton for the same language, it contains sufficient information for resolving the non-determinism.' Another supporting intuition is that a guidable automaton is able to 'simulate' the behaviour of any automaton for this language.

To implement this, let us first describe what is a *guide* from an automaton $\mathcal{A}$ to an automaton $\mathcal{B}$. A guide is an application:

$$g: \quad Q_\mathcal{B} \times \Delta_\mathcal{A} \to \Delta_\mathcal{B} \ ,$$

where $Q_\mathcal{B}$ are the state of $\mathcal{B}$, and $\Delta_\mathcal{A}$ and $\Delta_\mathcal{B}$ are the set of transitions of the automata $\mathcal{A}$ and $\mathcal{B}$ respectively.

Let us show now how to extend a guide $g$ into a function $\tilde{g}$ from the runs of $\mathcal{A}$ to runs of $\mathcal{B}$. Assume that the automaton $\mathcal{B}$ has a unique initial state $q_0$, and consider some run $(\rho_\mathcal{A}, \delta_\mathcal{A})$ of $\mathcal{A}$ over some input tree $t$. One constructs a run for $\mathcal{B}$ inductively as if $g$ was defining a top-down deterministic transducer of states $Q_\mathcal{B}$, reading the run of $\mathcal{A}$. Then there is at most one run $(\rho, \delta)$ of $\mathcal{B}$ such that:

$$\rho(\varepsilon) = q_0 \qquad\qquad \text{and} \quad \delta(x) = g(\rho(x), \delta_\mathcal{A}(x)) \quad \text{for all inner nodes } x \ .$$

This initial run, if it exists, is called the run of $\mathcal{B}$ driven by $(\rho_\mathcal{A}, \delta_\mathcal{A})$.

An infinite tree automaton $\mathcal{A}$ *guides* an infinite tree automaton $\mathcal{B}$ if there exists a guide $g$ such that for every accepting run $\rho$ of $\mathcal{A}$, $\tilde{g}((\rho, \delta))$ is an accepting run of $\mathcal{B}$ over the same infinite tree. A direct consequence of this is that the language of $\mathcal{A}$ is included in the language of $\mathcal{B}$. In fact, the guide $g$ can be understood as an 'easy to verify' certificate for this inclusion.

▶ **Definition 11.** An infinite tree parity[7] automaton $\mathcal{B}$ is *guidable* if for all infinite tree parity automata $\mathcal{A}$ generating a sub-language of $\mathcal{B}$, $\mathcal{A}$ guides $\mathcal{B}$.

Thus, an automaton is guidable if for all automaton generating a sub-language there is an easy certificate for the inclusion of the languages.

You can remark easily that every top-down deterministic automaton is guidable. Indeed, the guide uses the input transition from $\Delta_\mathcal{A}$ just for determinining the letter, and has no choice concerning the transition to take. The above infinite tree Büchi automaton for the language 'there is an occurrence of letter $a$' is guidable. Unfortunately, this would be too technical to establish here.

An important property of guidable automata is that they always exist.

▶ **Theorem 12** ([8]). *Every regular language of infinite trees is accepted by a guidable parity automaton.*

The construction requires a doubly exponential blowup, and this is tight [17].

This theorem was used for an attempt to deciding the Mostowski hierarchy for non-deterministic automata; *i.e.* the hierarchy induced by the number of priorities necessary for a parity automaton to accept a language of infinite trees. It seems that guidable automata are the correct form of automata to analyse in a procedure for deciding this hierarchy. For the moment the problem has only been reduced to another problem, the existence of a bound on the function computed by some forms of automata [8].

Löding has proposed a method for deciding if a given infinite tree parity automaton is guidable [17]. The key ingredient is to able to construct, given two infinite tree parity

[7] The parity conditions are special cases of Müller conditions: the letters are integers, and the accepting language contains those words such that the maximal integer encountered infinitely often is even.

automata $\mathcal{A}$ and $\mathcal{B}$, a finite Rabin game which is won by the first player if and only if $\mathcal{A}$ guides $\mathcal{B}$ (and in this case the positional winning strategy induces the guide). The algorithm for deciding if a given infinite tree parity automaton $\mathcal{B}$ is guidable is then to first transform $\mathcal{B}$ into a guidable automaton $\mathcal{A}$ (this requires a doubly exponential blowup in the number of states) and then check that $\mathcal{A}$ guides $\mathcal{B}$. This is a doubly exponential procedure.

▶ **Question 7.** *Can we decide efficiently if a infinite tree parity automaton is guidable? If the answer is no, does there exist small certificates that an infinite tree automaton is guidable? What is the state complexity of complementing a guidable automaton?*

## 5 History-determinism

### 5.1 Principle

History-determinism is a notion suitable for automata running over words (finite or infinite). It has been introduced for its use in the context of games of infinite duration by Henzinger and Piterman [13] under the name *'good for solving games'*. The same notion was introduced independently in [6] for cost functions, and used together with Löding in [9] for developing the theory of cost functions over finite trees.

Given an infinite word automaton $\mathcal{A} = (Q, \mathbb{A}, I, \Delta, \mathcal{W})$ (the automaton may have infinitely many states) and a mapping $\tau$ from $Q$ to some set $Q'$, denote by $\tau(\mathcal{A})$ the *homomorphic image* of $\mathcal{A}$ by $\tau$, that is the automaton $(Q', \tau(I), \tau(\Delta), \mathcal{W})$ where $\tau(\Delta)$ is the transition relation

$$\{(\tau(p), a, c, \tau(q)) \ : \ (p, a, c, q) \in \Delta\} \ .$$

The homomorphic image of an automaton over finite words is defined in a similar way, omitting the accepting condition part, and replacing the final states $F$ by $\tau(F)$.

Let us remark that $\mathcal{A}$ may be deterministic and $\tau(\mathcal{A})$ not be deterministic. Note also that the language of the homomorphic image is always a superset of the language of the original automaton. This simply comes from the fact that accepting runs in $\mathcal{A}$ have an accepting homomorphic image by $\tau$. Note finally that it can be a strict superset.

▶ **Definition 13.** An automaton is *history-deterministic* if it is the homomorphic image of a (possibly infinite) deterministic automaton for the same language.

The idea behind the notion of history-determinism is that an automaton can be non-deterministic, but that one can use the deterministic automaton from which it is the homomorphic image as an oracle for resolving the non-deterministic choices. If an automaton is history-deterministic, and if adding a new transition or a new initial state to it does not change the language, then the obtained automaton is again history-deterministic.

Consider a finite or infinite word history-deterministic automaton $\mathcal{H}$ which the homomorphic image $\tau(\mathcal{A})$ of a deterministic automaton for the same language. For $p$ a state of $\mathcal{A}$, call $A_p$ the language accepted from state $p$ by $\mathcal{A}$. For $q$ a state of $\mathcal{H}$, define in the same way $H_q$ the language accepted by $\mathcal{H}$ from state $q$. We claim that for all states $p$ of $\mathcal{A}$ reachable from the initial state, $A_p = H_{\tau(p)}$. One direction is straightforward. Indeed, any accepting run of $\mathcal{A}$ from state $p$ is send by $\tau$ to an accepting run over the same input of $\mathcal{H}$ from sate $\tau(p)$. Conversely, assume a word $u$ is accepted by $\mathcal{H}$ from state $\tau(p)$. Let $w$ be a run such that $\mathcal{A}$ reaches state $p$ after reading it (reachability assumption). This means that $wu$ is accepted by $\mathcal{H}$. Thus by the history-determinism assumption, $wu$ is accepted by $\mathcal{A}$. It follows, since $\mathcal{A}$ is deterministic, that $u$ is accepted by $\mathcal{A}$ from state $p$. This proves the claim.

We directly deduce from it the following characterisation of history-deterministic automata over finite words.

▶ **Proposition 14.** *An automaton over finite words is history deterministic if and only if it contains a deterministic sub-automaton for the same language.*

Indeed the construction is as follows, where we reuse the above notations for $\mathcal{A}$ and $\mathcal{H}$. First of all, one can consider that all states of $\mathcal{A}$ are reachable. Otherwise one removes the non-reachable states, and consider that $\mathcal{H}$ is the homorphic image of this restricted automaton. Then one chooses as unique initial state $\tau(q_0)$. Finally one removes sufficiently many transitions from $\mathcal{H}$ for making it deterministic without creating a dead-end; *i.e.* without ever removing the last transition issued from a given state reading a given letter. One then checks that none of these edge removals change the language accepted by $\mathcal{H}$. This is obvious since whenever a transition has been removed from $\mathcal{H}$, say $(p, a, q)$, then another transitions $(p, a, q')$ was existing. But, by the above claim, (and since we have only kept the states which are homomorphic images of reachable states of $\mathcal{A}$), $H_q = H_{q'}$. It follows that removing $(p, a, q)$ or $(p, a, q')$ (but not both) from the automaton does not change its language.

It is natural to wonder whether this result can be extended.

▶ **Conjecture 8.** *A parity automaton is history-deterministic if and only if it contains a deterministic sub-automaton for the same language.*

Let us remark that in the case of finite words, history-determinism has an efficient decision procedure.

▶ **Theorem 15** (Löding [18]). *One can decide in polynomial time if an automaton has a deterministic sub-automaton for the same language.*

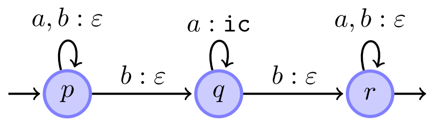
## 5.2 The motivating example of cost functions

Proposition 14 and Conjecture 8 tend to make us think that the notion is not very interesting. Indeed it implies that history-deterministic automata are always of bigger size than deterministic ones for the same language. So why would we be interested in using such forms of automata?

An enlightening example is to consider the case of cost functions. In this model deterministic automata are strictly weaker than non-deterministic ones, but history-deterministic automata are as expressive as non-deterministic automata.

Let us describe this model of automata through examples. Consider the following automaton:

$\mathcal{B} =$



This automaton is a *one counter B-automaton* ([6], following ideas from [16] and [2]). We do not want to enter a full description of this object. Let us simply describe its semantics. The interesting reader can refer to [9] for more precise definitions. In our case, the automaton possesses one counter, which ranges over non-negative integers. At the beginning of the run, the counter assumes value 0. Three actions can be performed on the counter:

- it can be left unchanged, using action $\varepsilon$,
- it can be incremented by 1, using action $\mathtt{ic}$,

- it can be reset to 0, using action $\mathtt{r}$.

Each transition is labelled $a_1, \cdots : \alpha$, where $a_1, \ldots$ are the letters read by the transition for being fired, and $\alpha$ is the action performed on the counter. A run of the automaton is defined as usual. A run is said *$n$-accepting* if it starts in an initial state, end in a final state and the counter never exceeds value $n$ along the run (this can be seen as a run under a resource consumption constraint ). The *$n$-language* accepted by the automaton is the set of words which are $n$-accepted. This first automaton $n$-accepts a word if and only if the input word contains a factor of the form $ba^m b$ for some $m \le n$. [8]

One needs to adapt the notion of history-determinism to B-automata. A B-automaton is *history-deterministic*[9] if for all $n$, there is a deterministic B-automaton of same $n$-language, the homomorphic image of which is $\mathcal{A}$. In other words, the automaton is unique, but should be uniformly history-deterministic for all acceptance conditions.

Let us give an intuitive meaning why the above automaton is not history-deterministic. Fix yourself some $n$, for instance 1. Now imagine that you feed this automaton with a word over $\{a, b\}$, letters by letters, yielding a word of value $n$. Your goal is to construct step by step an $n$-accepting run of the automaton. At the beginning of the run, you are in state $p$. While receiving each letter, you have to decide what transition to take. The non-determinism occurs when in state $p$, while reading letter $b$. Your problem is that you want to go to state $q$ only when a segment of $a$'s of length $n$ is about to arrive next. Whatever is your computational power, if you do not know the letters to arrive next, you are unable to decide if should proceed to $p$ or to $q$. This informal argument is the reason why this first B-automaton is not history-deterministic: it is impossible to resolve the non-determinism without looking the input ahead. This corresponds to the above notion of history-determinism since, assuming history-determinism, the automaton from which you are the homomorphic image is deterministic, and hence it has all the computation power possible (there is no size constraint, not even finiteness), but because of its determinism it has no access to the future of the input.

Consider now the following automaton:

$$\mathcal{H} = \begin{tikzcd} {} \arrow[r] & p \arrow[loop above, "a:\varepsilon"] \arrow[r, bend left, "b:\mathtt{r}"] & q \arrow[loop above, "a:\mathtt{ic}"] \arrow[l, bend left, "a:\varepsilon"] \arrow[r, "b:\varepsilon"] & r \arrow[loop above, "{a,b:\varepsilon}"] \arrow[r] & {} \end{tikzcd}$$

Consider an $n$-accepting run of this automaton over the input, then it has to read a $p$ at some point, and enters state $q$, then spends at most $n$ steps in state $p$, and then reads again a $b$ allowing to proceed to $r$ which is the only accepting state. This means that the word contains a factor $ba^m b$ for $m \le n$. Let us show conversely how to construct 'in a deterministic way' an $n$-accepting run over a word containing a factor $ba^m b$ for $m \le n$. The run starts in state $p$. As long as a letter is met, and there is no non-deterministic choice, the run is prolongated using the only transition available. The only case of non-determinism is when the run is in state $q$ and the input letter is an $a$. In this case, one has to choose to either stay in $p$–call this choice '*continue*'–or to go to $p$–call this choice '*skip*'–. One resolves this non-determinism by choosing to continue as long as the counter has a value smaller than $n$,

[8] One often sees B-automata as defining functions. The *function computed by a B-automaton* is the least $n$ such that there exists an $n$-accepting run, or infinity if there is no such run. In this case, the automaton computes the least size of a block of consecutive $a$'s surrounded by $b$'s.

[9] The exact definition requires this modulo an approximation [6]. We do not enter these details here.

and to skip otherwise. Using this construction, if the input word contains a factor $ba^mb$ for $m < n$, then the word has to terminate in state $r$ which is accepting. This strategy of constructing the run can be implemented by a deterministic automaton which would know (a) what state the automaton $\mathcal{H}$ is in and (b) the current value of the counter. In this case this bigger deterministic automaton (it has $3(n+1)$ states) is sent by homomorphic image to $\mathcal{H}$ (by projecting out the counter value), and accepts the same $n$-language. This means that $\mathcal{H}$ is history-deterministic. Furthermore it is equivalent to $\mathcal{B}$.

In the above example, it is clear that it is required to know the current value of the counter. And in fact, it would be impossible to provide a determinisitic or even an unambiguous B-automaton which would accept the same $n$-language for all $n$. In this case, the notions of determinism and history-determinism completely differ in expressiveness.

The general theory is a bit more involved, since automata need be considered modulo an approximation $\approx$ in order to be made history determinisitic. We do not define it here. Nevertheless, the following statement should sustain the interest of the notion of history-determinism.

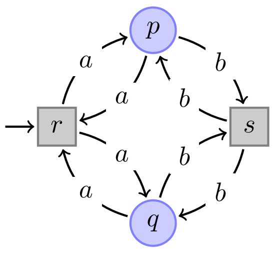
▶ **Theorem 16** ([6]). *B-automata cannot be made deterministic nor unambiguous, even up to $\approx$. Every B-automaton is equivalent to an history-deterministic one, up to $\approx$.*

In the last section, we explain why history-deterministic automata are good for solving games. This has been used in particular for proving results concerning cost functions in [9].

## 5.3 History-determinism is good for solving games

As mentioned above the key motivation for introducing history-determinism is its use for solving games. We will explain this for automata over infinite words, though the main application of the technique is for cost-functions.

We start by describing what is a game. A *game* $(A, \mathcal{W})$ consists of a *winning condition* $\mathcal{W} = (\mathbb{C}, W)$ (corresponding to an accepting condition in Section 3.4), and an *arena* $A$ which is a (possibly infinite) directed graph with edges labeled by $\mathbb{C}$, and such that the vertices (called *positions*) are partitioned into positions for *Eva* and positions for *Adam*. Furthermore, there is a distinguished *starting position* $p_0$. Traditionally, the positions of Eva are denoted by circles, and the positions of Adam by squares. The following example



The game is played by two opponents, Eva and Adam. It starts by putting a token on the starting position position (here $r$). Then the token is moved by the owner of the position. Here Adam for the first move. The player chooses the edges along which the token should move. Here, either to $p$ or to $q$. Then the game proceeds with the new position. In the above case, this is the turn of Eva to play. In the end, the two players will have constructed an infinite path in the arena, labeled with some infinite word $w$ over $\mathbb{C}$. This play is *winning for Eva* if $w \in W$. Otherwise it is *winning for Adam*. A player is winning if he has a strategy specifying what he should play such that, whatever are the choice of the opponent, the resulting play is winning for him. Formally, a *strategy* is defined as a mapping from partial plays (the prefix of a play), ending in a state of the player to edges, the transition taken by the strategy after having seen this prefix. In the above case, if the winning condition is the set of languages that contains infinitely many occurrences of both $a$ and $b$, then Eva (circle) wins the game by alternatively playing left and right.

A winning condition is called *determined* if for all games using this winning condition, one of the players has a winning strategy. It is possible to construct winning conditions which are not determined. However, thanks to Martin's theorem [19], every reasonable accepting condition, and in particular every regular winning condition is determined. This is an important property since it allows us to reason with arguments such as if there is no strategy for one player, then the other has a winning strategy: it allows to complement an existential property 'there is a strategy for one player', and obtain again an existential property 'there is a strategy for the other player' ($\star$).

Games play an important role in automata theory since the seminal work of Gurevich and Harrington [11], and in particular in the proof of closure under complementation of the regular languages of infinite trees. More specifically, complementing a regular language means negating a property of the form 'there exists a run of the first automaton', and obtain a property of the form 'there exists a run of the complement automaton'. This is exactly what games are good for (cf. $\star$). But for this approach to work, the automata must be able to 'manipulate' the strategies. The issue is that the strategy is a very complex object. Indeed it tells what should be the next arrow to take knowing the full history of the play so far. This is too much information for an automaton. For this reason, the simpler *finite memory* strategies play an important role. In the above example, player Eva requires one bit of memory in order to switch between left and right, and be sure to win: this is a memory 2 strategy. In fact, the condition 'infinitely many $a$'s and infinitely many $b$'s' requires memory at most 2 in every game. Said differently, on every game with this winning condition, if Eva wins, then she needs only one bit of memory for implementing a winning strategy. More generally, every winning condition which is a regular language of infinite words requires only a finite amount of memory to implement the winner's strategy.

It would be too long to explain in mode detail why finite memory strategy are important for complementing infinite tree automata. Another advantage of the notion is that it helps for deciding the winner in a finite game. Indeed, it is sufficient to guess a finite memory strategy (and this is an object of linear size), and then check that this strategy is winning (often a polynomial task). This is how one proves that deciding the winner of a game over a fixed regular winning condition is in NP$\cap$coNP (it is more complex if the winning strategy is part of the input).

Now, consider a game $\mathcal{G}$ of winning condition $\mathcal{L} = (\mathbb{B}, L)$, and assume you have a deterministic infinite word automaton $\mathcal{A}$ for the language $L$ of accepting condition $\mathcal{W} = (\mathbb{C}, W)$. Then, a natural thing to do is to compose the two objects. This yields a new game $\mathcal{G} \otimes \mathcal{A}$, the position of which are ordered pairs of a position in $\mathcal{G}$ and a state of $\mathcal{A}$, and of winning condition $\mathcal{W}$. At each turn of the game, the player chooses to move the token according to the first game, and the second component is updated according to the transitions of the automaton, outputting a letter from $\mathbb{C}$. Formally, a new position $(p, q)$ belongs to the same player as $p$ in $\mathcal{G}$, and there is an edge from $(p, q)$ to $(p', q')$ labelled $c$ if there is an edge from $p$ to $p'$ labelled $b$ in $\mathcal{G}$, and a transition of the form $(q, b, c, q')$ in $\mathcal{A}$. The starting position is the pair of the starting position in $\mathcal{G}$ and the initial state from $\mathcal{A}$. It is quite clear that playing this new game amounts to play the original game, and update at the same time the state of the automaton. Since the automaton accepts the winning condition $\mathcal{L}$, the winner is preserved.

▶ **Theorem 17.** *The games $\mathcal{G}$ and $\mathcal{G} \otimes \mathcal{A}$ have same winner. Furthermore, if the winner of $\mathcal{G} \otimes \mathcal{A}$ can win with memory $k$, then he/she can win with memory $k|\mathcal{A}|$ the game $\mathcal{G}$.*

The memory part of this statement is also natural: in order to translate a winning strategy from $\mathcal{G} \otimes \mathcal{A}$ to a winning strategy for $\mathcal{G}$, it is necessary to maintain (a) sufficient memory for winning the game $\mathcal{G} \otimes \mathcal{A}$, and furthermore (b) sufficient memory for keeping track of the state the automaton $\mathcal{A}$ should be in. Hence an upper bound of $k|\mathcal{A}|$.

A similar construction can be done, with $\mathcal{A}$ not deterministic. In this case, in the product game $\mathcal{G} \otimes \mathcal{A}$, player Eva is furthermore in charge of constructing the run of the automaton (this is implemented in the precise definition of the game). But, if $\mathcal{A}$ is not deterministic, what remains true is that if Adam wins $\mathcal{G}$, then Adam wins $\mathcal{G} \otimes \mathcal{A}$. The converse does not necessarily hold. Indeed, the game $\mathcal{G} \otimes \mathcal{A}$ requires Eva to resolve the non-determinism of $\mathcal{A}$. But, when Eva makes such a choice, Adam can take advantage of this information for winning the game. History-determinism is here the right notion, and this is why the notion is also called 'good for solving games' in [13]:

▶ **Theorem 18.** *If $\mathcal{H}$ is history-deterministic, then $\mathcal{G}$ and $\mathcal{G} \otimes \mathcal{H}$ have same winner.*

What differs here compared to the deterministic case is that, when translating the winning strategy for Eva in the game $\mathcal{G} \otimes \mathcal{H}$ to $\mathcal{G}$, one uses the automaton $\mathcal{A}$ from which $\mathcal{H}$ is the homomorphic image as second component.

The intriguing consequence of this is that the memory needed to win the game $\mathcal{G}$ is now $k|\mathcal{A}|$ and not $k|\mathcal{H}|$ as if $\mathcal{H}$ was deterministic. This is interesting since the automaton $\mathcal{A}$ may a priori be very large compared to $\mathcal{H}$. Using this construction, one reduces a game $\mathcal{G}$ which may require a lot of memory to a slightly larger game $\mathcal{G} \otimes \mathcal{H}$ of same winner which uses much less memory.

In the context of regular winning conditions, assuming Conjecture 8 holds, we cannot hope to really win something, since this means that one could choose the automata $\mathcal{H}$ and $\mathcal{A}$ of same size. In [9] the technique is used for cost functions. In this context, it allows one to transform games that require an unbounded quantity of memory into games which require no memory. History-deterministic automata play a crucial role in the theory of cost-functions for this reason.

## Acknowledgment

I am very grateful to Christopher Broadbent, Olivier Carton, Christof Löding, Gabriele Puppis and Thomas Wilke for their discussion and their help during the preparation of this document.

## References

1 Danièle Beauquier and Dominique Perrin. Codeterministic automata on infinite words. *Inf. Process. Lett.*, 20(2):95–98, 1985.

2 Mikolaj Bojańczyk and Thomas Colcombet. Bounds in $\omega$-regularity. In *LICS 06*, pages 285–296, 2006.

3 J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford Univ. Press, 1962.

4 Arnaud Carayol, Christof Löding, Damian Niwinski, and Igor Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Central European Journal of Mathematics*, 8(4):662–682, 2010.

5 Olivier Carton and Max Michel. Unambiguous büchi automata. *Theor. Comput. Sci.*, 297(1-3):37–81, 2003.

6 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150. Springer, Berlin, 2009.

7 Thomas Colcombet and Christof Löding. The nesting-depth of disjunctive $\mu$-calculus for tree languages and the limitedness problem. In *Computer science logic*, volume 5213 of *Lecture Notes in Comput. Sci.*, pages 416–430. Springer, Berlin, 2008.

8 Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *Automata, languages and programming. Part II*, volume 5126 of *Lecture Notes in Comput. Sci.*, pages 398–409. Springer, Berlin, 2008.

9 Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79, 2010.

10 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007.

11 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *STOC*, pages 60–65, 1982.

12 Yuri Gurevich and Saharon Shelah. Rabin's uniformization problem. *J. Symb. Log.*, 48(4):1105–1119, 1983.

13 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2006.

14 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

15 Michael Kaminski and Daniel Zeitlin. Extending finite-memory automata with non-deterministic reassignment (extended abstract). In *AFL*, pages 195–207, 2008.

16 Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO*, 3(39):455–509, 2005.

17 Christof Löding. Logic and automata over infinite trees. Habilitation thesis, Aachen, 2009.

18 Christof Löding. Finding determinisitic subautomata in polynomial time. Personal communication, 2011.

19 D. A. Martin. Borel determinacy. *Ann. Math.*, 102:363–371, 1975.

20 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

21 Robert McNaughton and Seymour Papert. *Counter-free Automata.* MIT Press, 1971.

22 David Muller. Infinite sequences and finite machines. In *Switching Theory and Logical Design, Proc. Fourth Annual Symp. IEEE*, pages 3–16. IEEE, 1963.

23 J. Myhill. Finite automata and representation of events. *Fund. Concepts in the Theory of Systems*, pages 57–624, 1957.

24 Sebastian Preugschat and Thomas Wilke. Characterizing fragments of ltl using cuba's. In *FoSSaCS*, 2012.

25 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. soc.*, 141:1–35, 1969.

26 Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM J. Res. and Develop.*, 3:114–125, April 1959.

27 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

**28** Richard Edwin Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.*, 14(3):598–611, 1985.

**29** Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.