Information and Computation 209 (2011) 692–704

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Reactive automata

Maxime Crochemore [a,c,*], Dov M. Gabbay [b,c]

[a] Université Paris-Est, France
[b] Bar Ilan University, Israel
[c] King's College London, London WC2R 2LS, UK

ARTICLE INFO

Article history:
Received 10 October 2008
Revised 23 May 2010
Available online 13 January 2011

Keywords:
Automaton
Language representation
Reactivity

ABSTRACT

A reactive automaton has extra links whose role is to change the behaviour of the automaton. We show that these links do not increase the expressiveness of finite automata but that they can be used to reduce dramatically their state number both in the deterministic case and the non-deterministic case.

Typical examples of regular expressions associated with deterministic automata of exponential size according to the length of the expression show that reactive links provide an alternative representation of total linear size for the language.

1. Introduction and background

This paper offers a reactive point of view for automata and shows that using reactive arrows can reduce the number of states of an automaton. Within the framework of reactive automata we provide some examples where such a reduction is striking.

Membership for regular languages, i.e., the problem of testing if a word belongs to the language, is traditionally done with the help of a finite automaton representing the language, which is equivalent due to Kleene's Theorem (see for example [11]). When the automaton is deterministic the acceptance of a word of length m can be tested in linear time using $O(kn)$ space, where k is the size of the alphabet and n the number of states of the automaton. If the automaton is non-deterministic the alternative is to simulate an equivalent deterministic automaton or to transform it into a deterministic automaton. The first option leads to $O(mn)$ membership time with space proportional to that of the non-deterministic automaton. The second option yields linear membership but at the cost of the determinisation which can be time and space exponential in the number of states. For related algorithms, see [1] or [12] and references therein. Both solutions are indeed implemented in the many variant of `grep` software aimed at locating regular motifs in texts.

The possible exponential blow up of automata when they are determinised is a major concern for applications. In this paper we outline a solution based on extra arcs called reactive links in so-called reactive automata. Their size seems to be restrained even with a small number of reactive links.

The notion of reactive automata was introduced in [4,6], as part of a general reactive methodology (see also [2,5,7–10]). The basic idea is that a reactive system is a system that changes when it is used, that is, it dynamically changes during its execution as a reaction to the manner it is being utilised. A practical example of that is any machine whose components change under stress or overuse. To give some such examples, consider:

1. an old bridge that might collapse if used by heavy lorries,
2. a fuse,
3. a computer that crashes if overheated.

A reactive system has to be distinguished from a time-dependent system as it is not dependent on an objective clock.

* Corresponding author at: King's College London, London WC2R 2LS, UK
E-mail address: maxime.crochemore@kcl.ac.uk (M. Crochemore).

0890-5401/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2011.01.002

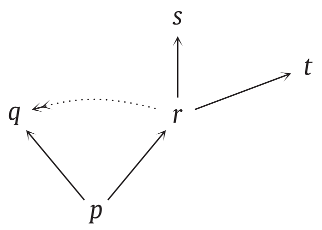

Fig. 1. Police blocks: the signal along the dotted double-head arrow from point *r* tells point *q* that it needs not block the road at *q*.

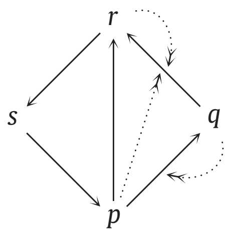

Fig. 2. A reactive graph.

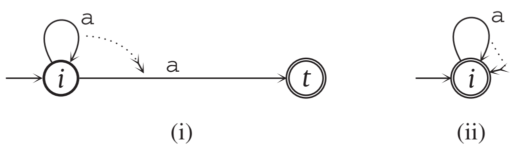

Fig. 3. Two reactive automata accepting the language $\mathtt{a}^{2n+1}$. All arcs are initially active. (i) The automaton uses a reactive edge-to-edge link, which cancels or activates the arc from the initial state *i* to the terminal state *t*. (ii) The automaton uses a reactive edge-to-state link, which flips the status of state *i* alternately to terminal and non-terminal.

1.1. Police blocks

A bank robbery took place at 5 pm just before bank closure time. The police institutes road blocks to catch the gateaway car. If a car is spotted crashing through block *r*, a signal is sent to other road blocks along the route to remain alert, while another signal is sent to road blocks not along the route to be lifted (see Fig. 1).

1.2. Reactive graph

The simplest theoretical example of a reactive system is the reactive graph of Fig. 2. The single-head arrows are directional links from vertices. The double-head arrows are links from vertices to single arrows. The expression "using the graph" can mean in this context "traversing the graph." Let us start from node *p* and move along the single-head arrows to other nodes. As we exit *p* to go either to *q* or to *r* along the single-head arrows the double-head arrow originated at *p* fires and disconnects the single-head arrow (q, r) it is pointing at. Thus, by exiting node *p* to go to node *q*, the arrow from *q* to *r* is cancelled. Upon continuing from *r* to *s* a signal is again sent to the arrow from *q* to *r* that is then reactivated. Here, "use" means "walking through the graph" and the "change" or "reaction" is executed by the double-head arrows, switching single-head arrows *on* and *off*.

Algebraically the graph can be described as a set of nodes $\{p, q, r, s\}$, with the single-head arrows written as pairs $\{(p, q), (p, r), (q, r), (r, s), (s, p)\}$ and the double-head arrows as $\{(p, (q, r)), (q, (p, q)), (r, (r, q))\}$.

1.3. Reactive automaton

Fig. 3 displays an example of a reactive automaton. Consider the automaton (ii), which has one state *i*. The transition is indicated as above by the single-head arrow and reactivity by the double-head arrow. For simplicity we assume we have only one letter $\mathtt{a}$. Assume *i* is the initial state as well as the terminal state. Upon receiving a letter $\mathtt{a}$ the machine stays at *i*, and can accept or continue. The status of *i* as a terminal state is then cancelled by the reactive arrow. Upon receiving a second letter $\mathtt{a}$ the machine cannot accept anymore the word $\mathtt{aa}$ as *i* is no longer a terminal state. Instead if the machine receives a third letter $\mathtt{a}$ state *i* is reactivated as a terminal state and the machine can accept the word $\mathtt{a}^3$. Obviously this reactive automaton accepts words of the form $\mathtt{a}^{2n+1}$, $n \geq 0$, only. To implement such an acceptor without reactivity we would need more states (at least two for this example). We shall see in Section 3 that any reactive automaton can be simulated by a non-deterministic automaton. However the saving we get through reactivity is in the complexity of states. Further sections of the article consolidate this idea.

The above examples show that there are several ways of defining reactive links: from state to state (Fig. 1), state to edge (Fig. 2), edge to state (Fig. 3(ii)), or edge to edge (Fig. 3(i)); in the rest, we choose the most general reactivity: edge-to-edge reaction.

The value of our approach requires some clarification and discussion. Suppose we have an algorithm A involving some parameters π. If we offer a variant algorithm A' which reduces the number of parameters in π but increases the number of some new parameters π', then we might question the value of the new A', especially if the overall complexity remains the same. A reasonable objection can be made that no gain was demonstrated. This surely might be a valid objection in an abstract setting: if A and A' have no intuitive meaning and are mere combinatorial variations of each other, then we have indeed gained nothing. Instead, in our case the reactive arrows have a meaning, are part of a general methodology with meaningful applications in many areas, and have a human intuitive appeal. So just on the grounds of meaning and intuition our contribution stands. But there is more in favour of the approach.

When an automaton is used in some applied area, its states mean something in the application. The transitions, in response to input, mirror some natural behaviour in the application. Our reactive arrows are fully compatible with the automaton transitions. All they say is similar to: if you make transition 1, then do not execute transition 2 and here is transition 3 instead! So this compatibility implies that reactive arrows can carry something that has a natural meaning in the application and therefore one may be able to devise a new simpler reactive automaton that does the same job with less complexity. It is true that when we prove a general reduction theorem we cannot perhaps show a real reduction in size, but in many cases of the application area and due to the nature of the job to be done reality allows us to simplify. This is because the reactive arrows have a natural meaning beyond their mere combinatorial use.

Take for example the reduction from k^n to kn states (see Section 6). Our formal proof embeds the k^n states into the paths of a $k \times n$ matrix. Formally this matrix is arbitrary and meaningless. But in an application it may present itself naturally, and because of that serious simplifications may be possible.

This type of competition between algorithms A and A' mentioned above is not new. Look for example at the classical propositional calculus. This is an NP-complete system and many problems in diverse areas are translated into it. It has several methodologies for theorem proving. Consider two of them: resolution, which is disjunction-negation based, and say tableaux or logic programming type, which is implication-conjunction based. Which one is to be used depends on the application. For colouring problems or Sudoku, problem resolution is better, but for rule based problems tableaux is better. We have elsewhere developed reactive tableaux [7] and their use can simplify the matter. If we realise during the computation that certain rules are no longer needed we can block them. So, an algorithm for reducing the number of disjunctions in favour of implications can be useful in an appropriate application.

In the next section we define the notion of a reactive automaton and prove in Section 3 that the expressive power of automata is unchanged by adding reactive links. In Section 4 we show how to reduce the number of states of an ordinary non-deterministic automaton by adding reactive links. Section 6 is the analogue for deterministic automata and shows an exponential decrease in the number of states provided enough reactive links of higher level introduced in Section 5 are added to the structure. Examples of reactive automata given in Section 7 do not have this drawback because they have only a linear number of reactive links while they are logarithmically smaller than the minimal automata associated with their accepted language. Some remarks and open questions are stated in the conclusion.

2. Reactive automata

We first explain our reactive concepts, then go to the formal definition.

2.1. Concepts

Let us fix the letter σ. A transition (by σ) from state a to state b can be “on” or can be “off”. We write it as (on, a, b) or as (off, a, b), or just as (a, b), leaving the on/off status to context. We also have reactive arrows, pictured as dotted arrows with a double head. The typical situations are described in Fig. 4.

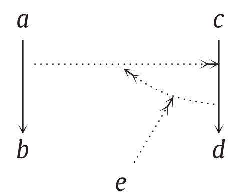

Fig. 4. Different types of reactive arrows.

When the automaton of Fig. 4 moves from a to b it sends a signal to the arrow (c, d). This signal can be of three types:

1. $(+, (a, b), (c, d))$ switches (c, d) to *on* if it is *off* and leaves it *on* if it is *on*.
2. $(-, (a, b), (c, d))$ switches (c, d) to *off* if it is *on* and leaves it *off* it is *off*.
3. $(\pm, (a, b), (c, d))$ is a proper switch: if (c, d) is *off* it changes it to *on*, and if it is *on* it changes it to *off*.

The reactive arrow itself (from (a, b) to (c, d)) can be on or off, which is denoted by $(\text{on}, (?, (a, b), (c, d))$ or $(\text{off}, (?, (a, b), (c, c)))$, where ? can be $+$ or $-$, or $\pm$.

Similarly we have a reactive arrow from (c, d) to the reactive arrow from (a, b) to (c, d). It can also be $+$, $-$, or $\pm$, which for example is denoted by $(+, (c, d), (+, (a, b)(c, d)))$.

A reactive arrow can emanate from a state. For example in the diagram $(+, e, (+, (c, d), (+, (a, b), (c, d))))$ is emanating from e.

An ordinary non-deterministic automaton $\mathcal{A} = (S, i, \Sigma, F, R)$ includes a set of states, S, an initial state $i \in S$, and a non-empty subset $F \subseteq S$ of terminal states. The automaton receives words written on the alphabet Σ. For every $\sigma \in \Sigma$ and state $p \in S$ the relation $R(p, \sigma, q)$ indicates that if the automaton is in state p and receives the letter σ then it can non-deterministically shift to state q.

Definition 2.1 (Switch reactive transformation). Let $R \subseteq S \times \Sigma \times S$ be the transition relation of an (ordinary) automaton $\mathcal{A}$. Let $\mathbf{T}^+, \mathbf{T}^-$ be two subsets of $(S \times \Sigma \times S) \times (S \times \Sigma \times S)$; they are composed of pairs of the form $((p, \sigma, q), (r, \tau, s))$ where $\sigma, \tau \in \Sigma$, $p, q, r, s \in S$, and $(p, \sigma, q) \in R$.

We define a transformation $(p, \sigma, q) \longrightarrow R^{(p,\sigma,q)}$ for $(p, \sigma, q) \in R$ using the sets $\mathbf{T}^+$ and $\mathbf{T}^-$ as follows:

$$\begin{aligned} R^{(p,\sigma,q)} &= (R \setminus \{(r, \tau, s) \mid (r, \tau, s) \in R \text{ and } ((p, \sigma, q), (r, \tau, s)) \in \mathbf{T}^-\}) \\ &\quad \cup \{(r, \tau, s) \mid (r, \tau, s) \in R \text{ and } ((p, \sigma, q), (r, \tau, s)) \in \mathbf{T}^+\} \end{aligned}$$

Definition 2.2 (Switch reactive automaton)

1. A reactive automaton is an ordinary non-deterministic automaton with a switch reactive transformation, i.e., a triple $\mathcal{R} = (\mathcal{A}, \mathbf{T}^+, \mathbf{T}^-)$ which defines the switch reactive transformation above.
2. Let $\sigma_1\sigma_2 \ldots \sigma_n$ be a word on the alphabet Σ. We define the notion of a (non-deterministic) run of $\mathcal{R}$ over $\sigma_1\sigma_2 \ldots \sigma_n$. The run is a sequence of pairs (p_k, R_k), $k = 0, \ldots, n$, defined as follows:
   Step 0. We start with the pair $(p_0, R_0) = (i, R)$ from the automaton $\mathcal{A} = (S, i, \Sigma, F, R)$.
   Step $k > 0$. Assume pairs $(p_0, R_0), (p_1, R_1), \ldots, (p_{k-1}, R_{k-1})$ have been defined as the result of a run over $\sigma_1\sigma_2 \ldots \sigma_{k-1}$. Then, state p_k is such that $(p_{k-1}, \sigma_k, p_k) \in R_{k-1}$ and $R_k = R_{k-1}^{(p_{k-1},\sigma_k,p_k)}$.
3. We say that the reactive automaton $\mathcal{R}$ accepts the word $\sigma_1\sigma_2 \ldots \sigma_n$ if there is a run of the automaton over this word that ends with $p_n \in F$.

Fig. 5 shows a reactive automaton that changes its arcs going to the unique terminal state. The automaton is deterministic and has linear size, $O(n)$. The automaton accepts the language $A^*\texttt{a}A^n\texttt{c}$ over the alphabet $A = \{\texttt{a}, \texttt{b}, \texttt{c}\}$. It is known that the minimal deterministic (ordinary) automaton accepting the same language has $2^{n+1} + 1$ states (see [11]).

Fig. 6 shows another reactive automaton which changes its terminal states only and accepts a language similar to that of Fig. 5. The definition of a reactive automaton changing its terminal states is a simple adaptation of the above definition. Changing terminal states can be simulated in the model of switch reactive automata as follows without altering the possible

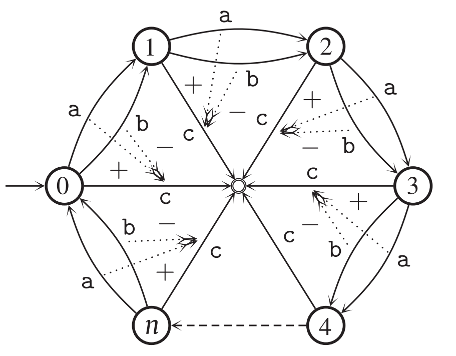

Fig. 5. Reactive automaton on the alphabet $A = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ accepting the language $A^*\texttt{a}A^n\texttt{c}$. The automaton is deterministic. Any a-transition activates a c-arc from its origin state to the centre, which is the only terminal state. Any b-transition switches off such an arc. If a run over a word stops in the terminal state (the last step is a c-transition), the $(n + 1)$th letter before the end of the word must be a, so the word belongs to the language, and conversely.

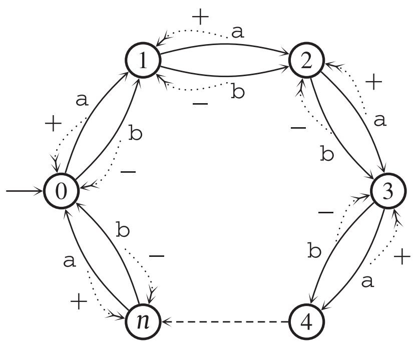

Fig. 6. Reactive deterministic automaton on the alphabet $A = \{\mathtt{a}, \mathtt{b}\}$ accepting the language $A^*\mathtt{a}A^n$ similar to the language of the automaton of Fig. 5. Initially, the automaton has no terminal state and it uses edge-to-state reaction. Any a-transition makes its starting state a terminal state. Any b-transition transforms its starting state into a non-terminal state. If the run over a word stops in a terminal state, since the state has been set as a terminal state by an a-transition on the nth letter before, the word belongs to the language, and conversely.

determinism: all potential terminal states are made non-terminal and linked by an ε-arc to a unique terminal node; reactive links to states are redirected to the new ε-arcs.

In the following we also say that arcs are activated or inactivated. It means that activated arcs are those in R, the others are in its complement.

We introduce in Section 5 below a higher level notion of reactive links that we use in Section 6.

3. Reactivity and non-reactivity

We focus on switch reactive automata and show that their expressive power is identical to the one of ordinary automata. The proof can be adapted to automata with the various types of reactive links described in previous sections.

Theorem 3.1. *Any switch reactive deterministic or non-deterministic automaton is equivalent to an (ordinary) deterministic or non-deterministic automaton, respectively.*

Proof. Let $\mathcal{R} = (\mathcal{A}, \mathbf{T}^+, \mathbf{T}^-)$. We define the automaton $\mathcal{B} = (\hat{S}, (i, R), \Sigma, F, \hat{R})$ whose set of states $\hat{S}$ is composed of pairs of the form (x, R') where $x \in S$ and R' is related to R via the switch reactive transformation using $\mathbf{T}^+$ and $\mathbf{T}^-$. The transition relation $\hat{R}$ of $\mathcal{B}$ is defined using $\mathbf{T}^+$ and $\mathbf{T}^-$ as follows: $((x_1, R_1), \sigma, (x_2, R_2)) \in \hat{R}$ iff $(x_1, \sigma, x_2) \in R_1$ and $R_2 = R_1^{(x_1,\sigma,x_2)}$. Then $\mathcal{B} = (\hat{S}, i, \Sigma, F, \hat{R})$.

It is straightforward to see that a run of $\mathcal{R}$ on $\sigma_1\sigma_2 \ldots \sigma_n$ corresponds to a run of $\mathcal{B}$ on the same word and vice versa. □

Remark. We saw that a switch reactive automaton actually starts as an ordinary automaton $\mathcal{A} = (S, i, \Sigma, F, R)$ and then changes into different automata while receiving the input. The proof extends to reactive automata with state-to-state or edge-to-state reactive links. Therefore, changes can either affect the transition relation or the terminal states as shown on the previous examples.

4. Saving states of non-deterministic automata using reactive links

Reactive links can be used to reduce the number of states of (ordinary) automata. In the section we show how it works in general for non-deterministic (deterministic automata are considered in the next section). It shows the power of reactive links provided we add enough of them to the new structure. Bounding linearly the number of reactive arcs to get the same result is an open question.

Let $\mathcal{A}$ be a non-deterministic automaton with kn states on the alphabet Σ. The aim is to reduce the number of states to $k + n$ by introducing reactive links.

Let us denote the states of $\mathcal{A}$ as pairs (s_i, e_j), $i = 1, \ldots, k, j = 1, \ldots, n$. Let $S = \{s_1, \ldots, s_k, e_1, \ldots, e_n\}$. We define a reactive automaton $\mathcal{R}(\mathcal{A})$ on the set of states S that has the same behaviour as $\mathcal{A}$.

4.1. Basic idea

We first explain the basic idea. Let σ be a fixed letter and let (s, e) be a fixed state. Record the σ-transitions from (s, e) as in Fig. 7.

We organise the non-deterministic targets according to the lexicographic order of pairs (s, e). We “break” the non-deterministic choice of edges by first choosing the s component and then the corresponding e component of pairs.

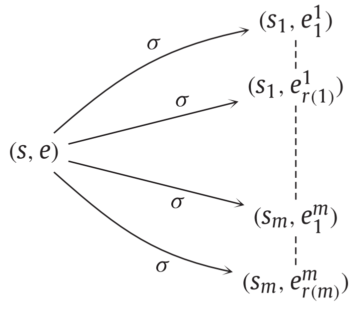

Fig. 7. Edges outgoing state (s, e) in the non-deterministic automaton $\mathcal{A}$, and their target states.

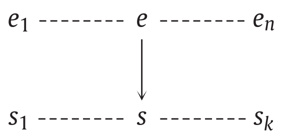

Fig. 8. An arrow from e to s in the reactive automaton $\mathcal{R}(\mathcal{A})$ represents the state (s, e) of the automaton $\mathcal{A}$.

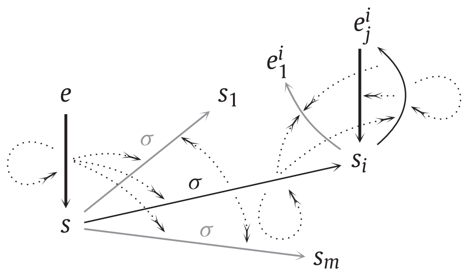

Fig. 9. Simulation of a transition in the automaton $\mathcal{A}$ from state (s, e) to state of (s_i, e_j^i) by arcs and reactive edges of the reactive automaton $\mathcal{R}(\mathcal{A})$: from state e to state e_j^i.

Fig. 8 illustrates the states of $\mathcal{R}(\mathcal{A})$. The arrow from e to s represents the pair (s, e). The automaton $\mathcal{A}$ being in state (s, e) is represented by the automaton $\mathcal{R}(\mathcal{A})$ being in state e with the arrow from e to s being the only active arrow.

Suppose we see the letter σ and want to move from (s, e) to one of its targets, say (s_i, e_j^i), $1 \leq j \leq r(i)$. We want to end up in state e_j^i of $\mathcal{R}(\mathcal{A})$ with the only active arrow being the arrow from e_j^i to s_i. To do that, we first move along the arrow from e to s. This move activates all (non-deterministic) connections from s to s_i, $i = 1, \ldots, m$ (see Fig. 9). We also inactivate the arrow from e to s. We then move non-deterministically along one of the newly activated links labelled σ, say from s to s_i. This is a choice of the $(s_i, .)$ group. Immediately as we move along the arrow from s to s_i we activate (connect) the links to $e_1^i, \ldots, e_{r(i)}^i$ emanating from s_i and disconnect all arrows emanating from s including the arrow from s to s_i itself. Now we are at s_i with arrows leading to e_j^i, $1 \leq j \leq r(i)$. We move non-deterministically along one of them to reach, say e_j^i. Immediately as we move along the arrow from s_i to e_j^i we disconnect all arrows emanating from s_i and activate the arrow from e_j^i to s_i. Which leads to state e_j^i with the only activated link being from e_j^i to s_i. Fig. 9 shows the connections we need.

Note that the same procedure works for the deterministic case as well. But we show later that a better result can be obtained in the deterministic case. Also note that the move in $\mathcal{A}$ from (s, e) to (s_i, e_j^i) is simulated by three moves in $\mathcal{R}(\mathcal{A})$

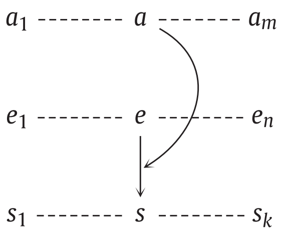

Fig. 10. Possible representation of state (s, e, a), among the knm states of the non-deterministic automaton $\mathcal{A}$, by arrows in the reactive automaton $\mathcal{R}(\mathcal{A})$.

and that we use only reactive arrows of second level, i.e., $(0 \to 0) \to (0 \to 0)$. For the result of Section 5 we need higher level arrows.

With the above procedure the simulation requires only a switch automaton.

The reactive automaton $\mathcal{R}(\mathcal{A})$ is defined by the states and connections as defined above for each σ and each (s, e). For each edge $((s, e), \sigma, (s_i, e^i_j))$ of $\mathcal{A}$, $1 \le j \le r(i)$, the automaton $\mathcal{R}(\mathcal{A})$ contains the following links:

(e, s)	active at start
(s, e)	non-active
$((e, s), (e, s))$	active
(s, s_i)	not active, $1 \le i \le m$
$((e, s), (s, s_i))$	active
$((s, s_i), (s, s_\ell))$	active, $1 \le i, \ell \le m$
$((s, s_i), (s_i, e^i_j))$	active, $1 \le i \le m$, $1 \le j \le r(i)$
(s_i, e^i_j)	not active, $1 \le i \le m$, $1 \le j \le r(i)$
(e^i_j, s_i)	not active, $1 \le i \le m$, $1 \le j \le r(i)$
$((s_i, e^i_j), (s_i, e^i_\ell))$	active, $1 \le j, \ell \le r(i)$
$((s_i, e^i_j), (e^i_j, s_i))$	active, $1 \le j, \ell \le r(i)$.

The above transformation of the non-deterministic automaton $\mathcal{A}$ to the switch automaton $\mathcal{R}(\mathcal{A})$ proves the next statement.

Lemma 4.1. *Associate the double condition $\mathcal{R}(\mathcal{A})$ is in state e and (e, s) and the only active arrow in $\mathcal{R}(\mathcal{A})$ with the state (s, e) of $\mathcal{A}$. Set the arrow (e, s) as terminal in $\mathcal{R}(\mathcal{A})$ whenever state (s, e) is terminal in $\mathcal{A}$. Then the automaton $\mathcal{A}$ accepts w iff $\mathcal{R}(\mathcal{A})$ accepts it.*

Corollary 4.2. *If w is recognised by $\mathcal{A}$ with kn states then it is recognised by $\mathcal{R}(\mathcal{A})$ with $k + n$ states. If $\mathcal{A}$ is deterministic, so is $\mathcal{R}(\mathcal{A})$.*

4.2. Can the reduction process be repeated?

Let $\mathcal{A}$ be a deterministic automaton having *knm* states. Can we collapse it to a reactive automaton with $k + n + m$ states?

Consider the states $\{s_1, \dots, s_k, e_1, \dots, e_n, a_1, \dots, a_m\}$. We can represent the states of $\mathcal{A}$ as triples (s, e, a). The question is, if we use a reactive automaton representing the "(s, e)" part as before, then by the same spirit we need to represent $((s, e), a)$ as in Fig. 10.

We need an arrow from a to the arrow (e, s). This is *not* an arrow to e but really to the arrow (e, s) because the arrow is associated with the pair (s, e). However, this cannot be done as these arrows must move from state to state. Indeed, there is another solution to reduce the number of states when the automaton $\mathcal{A}$ is deterministic as shown in the next section.

5. Higher level reactive links

For the results of Section 6 we need to use higher order links. These arrows emanating from arrows to arrows, etc., are inductively defined to any level. The concepts we define now.

Definition 5.1. Let S be the states and Σ be the alphabet of the automaton.

1. A level $0 \longrightarrow 0$ arrow has the form (s_1, σ, s_2) where $s_1, s_2 \in S$ and $\sigma \in \Sigma$.
2. Let α and β be arrows of levels ℓ_α and ℓ_β, respectively. Then $\alpha \longrightarrow \beta$ is an arrow of level $\ell_\alpha \longrightarrow \ell_\beta$.

Fig. 11 shows a reactive automaton having reactive arrows of different levels. Arrow 1 has level $\ell_1 = [(0 \longrightarrow 0) \longrightarrow (0 \longrightarrow 0)]$. Arrow 3 is of level $\ell_2 = [((0 \longrightarrow 0) \longrightarrow (0 \longrightarrow 0)) \longrightarrow ((0 \longrightarrow 0) \longrightarrow (0 \longrightarrow 0))]$. Arrow 4 has level $\ell_1 \longrightarrow \ell_2$.

When we move from a to b we pass through the connection $a \longrightarrow b$. So arrow 1 is "asked to act" and sends a signal to the arrow $b \longrightarrow c$. Since arrow 1 was asked to act, in turn it asks arrow 4 to act and send a signal to arrow 3. Thus, the effect of using the arrow $a \longrightarrow b$ is transmitted to arrows 1, 4, and 3.

To describe reactions as in the above example, we need to give a formal inductive definition.

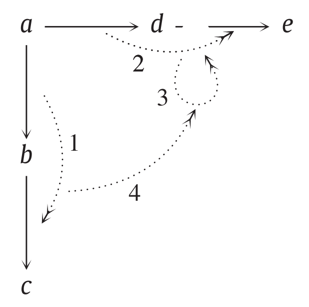

Fig. 11. Reactive arrows of different levels.

Definition 5.2. Let S be a set of states, Σ the alphabet, and T a set of higher level transitions as defined in Definition 5.1. Assume that if $\alpha \longrightarrow \beta \in T$ then $\alpha \in T$ and $\beta \in T$. Let τ be a function from T to $\{0, 1\}$: $\tau(\alpha) = 1$ says that α is on (or live), and $\tau(\alpha) = 0$ says that α is off (or not on). (Note that being "on" is not equivalent to being "asked to act".) Assume $(s, \sigma, t) \in T$ is on. We now calculate by induction the cumulative effect of the transition from s to t that changes the τ value of some arrows.

5.1. The inductive algorithm

Assume (s, σ, t) is on.

Step 0. Make a move along the arc from s to t. This starts a chain reaction as follows. Let $\gamma = [(s, \sigma, t) \longrightarrow \beta]$. We say that γ is "asked to act" at step 0. Two cases: γ may be on or off. If it is off nothing happens. If it is on it will act. So let at this step 0 $\tau^0 = \tau$ and let the arrows which are asked to act at step 0 all be arrows γ of the form $[(s, \sigma, t) \longrightarrow \beta]$.

Step 1. Let $\gamma = [(s, \sigma, t) \longrightarrow \beta]$ be any arrow that was asked to act at step 0 such that it is on. So γ is now ready to act at step 1. If β is on according to τ^0, then make β off according to τ^1, that is, set $\tau(\beta) = 0$. If β is off according to τ^0, then make β on according to τ^1. This defines τ^1.

Also if $\epsilon = (\gamma \longrightarrow \delta)$ then since γ is acting at step 1, ϵ is asked to act at step 1. It will act at stage 2 if ϵ is on at step 2. The above gives us τ^1 and gives us a list of arrows that are asked to act at step 1. We can now go to step 2.

Step *n* + 1. We assume we have defined τ^n at step n and have identified at stage n those arrows $\gamma = \beta \longrightarrow \delta$ which were "asked to act" at step n. We now start the process of stage $n + 1$. For each $\gamma = \beta \longrightarrow \delta$ which was asked to act at step n and which is on according to τ^n, we look at β. If β is on according to τ^n it is turned off for τ^{n+1} and if it is off it is turned on for τ^{n+1}. We also look at arrows η of the form $\gamma \longrightarrow \epsilon$ and ask them to act at step $n + 1$. We now have τ^{n+1} and a list of arrows asked to act at step $n + 1$, so we can carry on to step $n + 2$. Note that the level of the arrows η which we ask to act goes up at every stage, and so this will prevent looping — at some step no arrow will be asked to act. At some step m there will be no more arrow η being asked to act unless the graph is infinite. Note that we may have loops in the reactivity arrows but whenever an arrow is asked to act it must emanate from another arrow acting already and so its level is higher. So after m steps the process will stop and then let $\tau_{s,\sigma,t} = \tau^m$, for the first value m for which there is no change, that is, $\tau^m = \tau^{m+1}$. Thus we defined the following notion by induction: given τ and (s, σ, τ) which is on according to τ we defined $\tau_{(s,\sigma,\tau)}$.

6. Reducing the number of states of deterministic automata

On deterministic automata reactive links are even more powerful to reduce their number of states. This section shows that a deterministic automaton with k^n states can be reduced to an equivalent deterministic reactive automaton with $k \cdot n$ states by adding enough reactive links. Bounding linearly the number of them to get the same result, as in the previous section, is an open question. Although the construction of this section adapts to non-deterministic automata, we only present it for deterministic automata for reasons of explanatory exposition and to avoid unnecessary complicated presentation.

Let $\mathcal{A}$ be a deterministic automaton. Let σ be a fixed letter. Let f_σ be the transition table associated with σ ($f_\sigma(x) = y$ means that state y is the next state after x under the action of the input σ).

Assume $m = k^n$. We represent the states x of $\mathcal{A}$ as vectors $x = (x_1, \ldots, x_n)$, where $x_i \in \{a^i_1, \ldots, a^i_k\}$ for each i. The $k \times n$-matrix $\mathbb{A} = \{a^i_j, i = 1, \ldots, k, j = 1, \ldots, n\}$ comprises pairwise distinct elements.

We now define a deterministic reactive automaton $\mathcal{R}(\mathcal{A})$ from $\mathcal{A}$. The states of $\mathcal{R}(\mathcal{A})$ are the $k \times n$ elements

$$a^i_j, \quad i = 1, \ldots, n, j = 1, \ldots, k.$$

Let R be the set of connections in the automaton $\mathcal{R}(\mathcal{A})$. We write $(x, y) \in R$ to mean that there is a connection from x to y. It may be *on* or it may be *off*. This is not indicated since it changes as we move from state to state. Double arrows are written as $(?, x, y)$ as mentioned above. Again these may be *on* or *off*, depending on the context.

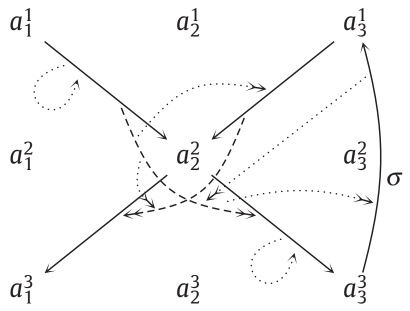

Fig. 12. Two paths in a (3×3)-matrix representing two states of a deterministic automaton with 3^3 states with the help of the dashed reactive arrows.

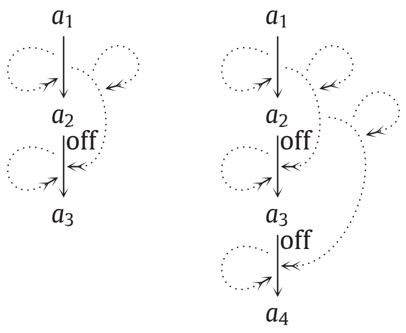

Fig. 13. Representing a triplet (left) and a 4-tuple (right). The bottom dotted arrow encodes the triplet (a_1, a_2, a_3) or the four element vector (a_1, a_2, a_3, a_4), respectively. Only inactivated edges are indicated as off.

We need some more notation. Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ be two vectors $(x_i, y_i \in \{a_1^i, \ldots, a_k^i\})$ for which $f_\sigma(x) = y$. We need to represent the move from x to y in the reactive automaton with states $a_j^i, i = 1, \ldots, n, j = 1, \ldots, k$. To do this, we have to encode states $x = (x_1, \ldots, x_n)$ in the reactive automaton in some way. We use paths in the $k \times n$-matrix $\mathbb{A}$. A path through the matrix (from top to bottom) corresponds to a vector $x = (x_1, \ldots, x_n)$ for which $x_i \in \{a_1^i, \ldots, a_k^i\}$. We can represent paths by connecting arrows.

Fig. 12 shows two paths for a 3×3-matrix. Let $x = (a_1^1, a_2^2, a_3^3)$ and $y = (a_3^1, a_2^2, a_1^3)$. The active arrows are (a_1^1, a_2^2), (a_2^2, a_3^3), (a_3^1, a_2^2), and (a_2^2, a_1^3). If we rely on active arrows only to represent triplets, then we are also activating two more paths, namely (a_1^1, a_2^2, a_1^3) and (a_3^1, a_2^2, a_3^3). So we have to do it differently.

On Fig. 12 the triplet (a_1^1, a_2^2, a_3^3) is characterised both by the edge (a_2^1, a_2^2) and the reactive arrow $((a_2^1, a_2^2), (a_2^2, a_3^3))$; (a_3^1, a_2^2, a_1^3) is characterised both by (a_3^1, a_2^2) and $((a_3^1, a_2^2), (a_2^2, a_1^3))$. In fact, the two dashed arrows alone contain all the necessary information. Each dashed arrow represents a unique triplet.

Now suppose $x = (a_1^1, a_2^2, a_3^3)$ represents a state in a deterministic automaton with 3^3 states. Suppose the transition from x by σ goes to $y = (a_3^1, a_2^2, a_1^3)$. How can we represent that? We start with the active edge (a_1^1, a_2^2) and the active dashed arrow $((a_1^1, a_2^2), (a_2^2, a_3^3))$. This means that the automaton $\mathcal{R}(\mathcal{A})$ is in state x and we are at a_1^1 in the matrix. Deterministically, you can move only to a_2^2. This activates the edge (a_2^2, a_3^3) via the dashed arrow that uniquely represents the triplet $x = (a_1^1, a_2^2, a_3^3)$. Now we can activate $y = (a_3^1, a_2^2, a_1^3)$ as follows. We send two signals to activate the edge (a_3^1, a_2^2) and the arrow $((a_3^1, a_2^2), (a_2^2, a_1^3))$. We also send a signal to activate the σ-edge. Once we get to a_3^3 we follow the edge labelled by σ to a_3^1 (first element of y) and inactivate x, that is, inactivate (a_1^1, a_2^2), (a_2^2, a_3^3), and $((a_1^1, a_2^2), (a_2^2, a_3^3))$.

The basic idea is that we encoded a triplet (a_1, a_2, a_3) by edges and arrows as shown on Fig. 13 (left). To encode a 4-tuple (a_1, a_2, a_3, a_4) we do it as on Fig. 13 (right). To encode a n-tuple, we cascade the construction of previous figure, which is displayed on Fig. 14 as well as a σ-transition.

The above construction reduces the number of states of the initial automaton from k^n to kn. It replaces them by reactive arrows but sending signal through arrows may be cheaper than having states. It is obvious that an equivalent presentation of a problem maybe cheaper than another.[1]

The conclusion of the section is in the next statement.

[1] A typical example is polynomial evaluation:

$$f(x) = 1 + 2x + 3x^2 + x^3 = 1 + x(2 + x(3 + x)),$$

for which the second expression is cheaper to evaluate.

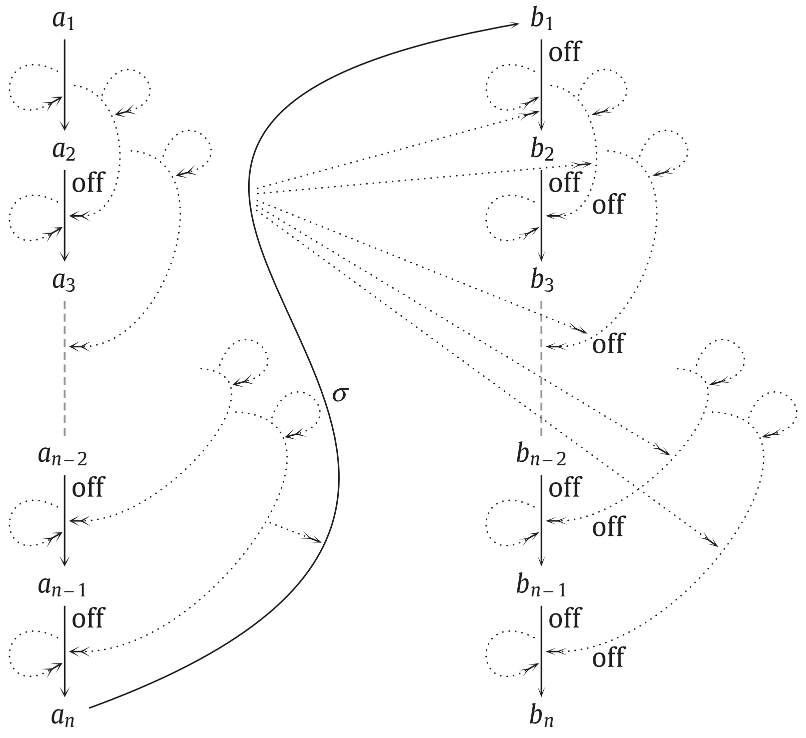

Fig. 14. Encoding of n-tuples and transition from the n-tuple $(a_1, a_2, \ldots, a_n)$ to the n-tuple $(b_1, b_2, \ldots, b_n)$. Only inactivated edges and arrows are indicated as off. Loops are permanently on.

Proposition 6.1. *If $\mathcal{A}$ is deterministic automaton with k^n states, it has an equivalent reactive automaton $\mathcal{R}(\mathcal{A})$ with $k \cdot n$ states.*

7. Examples in size reduction

The use of reactive links in automata can dramatically reduce the size of a deterministic automaton accepting a given regular language. The two previous sections show that this is possible for deterministic as well as non-deterministic automata but with a large number of reactive arcs. Instead, in the next examples, reactivity keeps the total size of automata as small as the size of their non-deterministic equivalent but without loosing determinism. In these examples determinism without reactivity leads to an exponential blow up of the number of states and then of the total size of automata.

The first example corresponds to the finite language of words in which each letter of the alphabet appears at most once (see Fig. 15). Its principle is that loops on the initial state are cancelled by a reactive link immediately after being used. States of a deterministic automaton for the language have to store the set of letters already treated and therefore the minimal automaton has at least an exponential number of states. Instead the total size of the reactive automaton accepting the language is $O(k)$ on a k-letter alphabet.

The automaton of Fig. 16 accepts all the $k!$ permutations of letters. To do that we add a path of length k from the initial state to the unique terminal state. Compared with the automaton of the previous example, the aim is to count the number of letters treated on the initial state. Each loop on the initial state has an additional reactive link that activates its associated ε-arc on the path. So, the terminal state can be reached only if all ε-arc are activated. We view the automaton as deterministic because its light non-determinism due to ε-arcs can be remove by considering a special symbol marking the end of words. The size of the reactive automaton is $O(k)$, which contrasts with the $O(2^k)$ size of the minimal automaton accepting the language.

The language $A^*\mathtt{a}A^n$ is accepted by the non-deterministic automaton of Fig. 17 that has $n + 2$ states and $O(n)$ total size. The non-determinism appears on the initial state only. It is known that the minimal deterministic automaton accepting the same language has 2^{n+1} states (see [11]) and then also $O(2^n)$ total size, while the equivalent reactive automaton of Fig. 5 has only also $n + 2$ states. It is noticeable that it has $O(n)$ total size despite the addition of reactive links.

The last remarkable example concerns the language $A^{\leq n}(\mathtt{a}A^n)^*$ on the alphabet $A = \{\mathtt{a}, \mathtt{b}\}$. Fig. 18 displays a non-deterministic automaton accepting it. It is non-deterministic because all its $n + 1$ states are initial states. Fig. 19 shows an equivalent reactive automaton, which, as above, may be considered as deterministic since ε-arcs are useful only at the end of the input word. Reactive links are from $\mathtt{b}$-transitions and cancel their associated ε-arc to the terminal state. The number of states is $n + 2$ and the total size is $O(n)$. This is to be compared with the result of Béal et al. [3], which shows that the minimal deterministic automaton for this language has an exponential number of states.

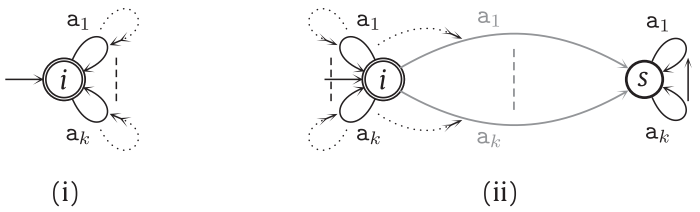

Fig. 15. Two deterministic reactive automata accepting the set of strings in which each letter of the alphabet $\{\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_k\}$ appears at most once. All loop arcs are initially active. Loops on state *i* are made inactive after their first use. (i) Incomplete version. (ii) Complete version: only arcs from *i* to *s* are initially inactive and become active after the first use of their corresponding loop on state *i*.

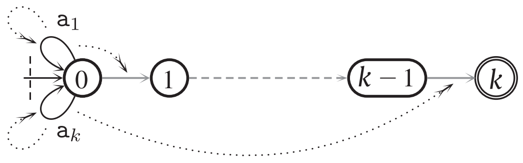

Fig. 16. A deterministic reactive automaton accepting the set of strings that are permutations of the letters $\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_k$. All loops on the initial state are initially active and other ε-arcs are inactive. One reactive link for letter $\mathtt{a}_i$ cancels its respective loop while the second activates its associated ε-arc.

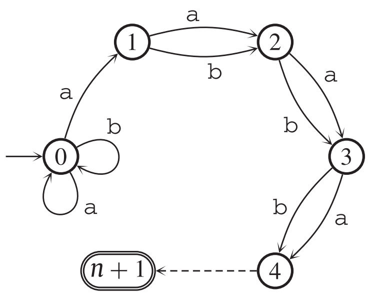

Fig. 17. Non-deterministic automaton on the alphabet $A = \{\mathtt{a}, \mathtt{b}\}$ accepting the language $A^*\mathtt{a}A^n$. Its equivalent reactive automata of Figs. 5 and 6 have sizes of the same order.

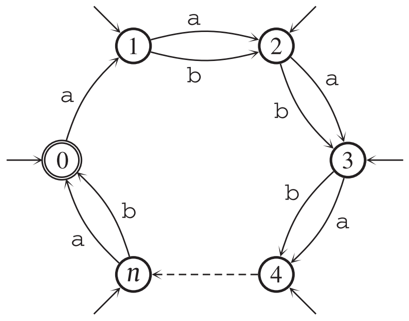

Fig. 18. Non-deterministic automaton on the alphabet $A = \{\mathtt{a}, \mathtt{b}\}$ accepting the language $A^{\leq n}(\mathtt{a}A^n)^*$. All states are initial states.

8. Conclusion

The strength of reactive links in automata comes essentially from the reduction of the size of their implementation. Designing a non-deterministic automaton to solve a Pattern Matching question leads to slow algorithms requiring extra work to be implemented. Instead, the use of reactive links can turn a non-deterministic automaton into a deterministic Pattern Matching machine. This has two simultaneous advantages: little effort at implementation and efficient running time since the basic operation required when parsing a stream of data with an automaton is table lookup to change state, which avoids any other more time demanding low or high level instruction.

The usefulness of reactive links is obvious when they are used to send signals to part of the machine that is to be used in the rest of the parsing. The signals change the behaviour of the machine in a very smooth way but with deep consequences on the future analysis of the stream.

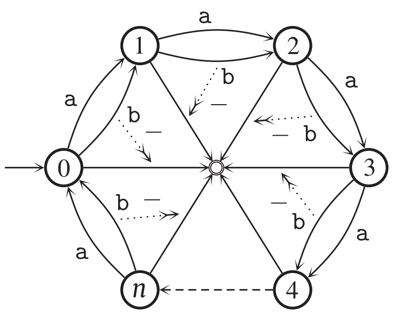

Fig. 19. Reactive deterministic automaton on the alphabet $A = \{\mathtt{a}, \mathtt{b}\}$ accepting the same language $A^{\leq n}(\mathtt{a}A^n)^*$ as the automaton of Fig. 18. It has only one terminal state in the center. During a run, at least one ε-arc remains if some positions of letter a in the input word form a non-extendible arithmetic progression of period n. The automaton has only one more state and twice as many arcs as the automaton of Fig. 18.

8.1. Gene signals

We describe here a possible use of a reactive automaton for fast recognition of coding sequences (potential genes) in genomes or chromosomes. An essential part of the information of these molecules is stored in data bases in the form of DNA sequences drawn on the four-letter alphabet of nucleotides {A, C, G, T}. Some segments of a DNA sequence, called coding genes, encode protein molecules and have specific features. As an example and to simplify the demonstration, we consider only the two following properties: a coding gene is a sequence of codons (words of length 3) that starts at position 0 with the codon ATG; upstream, around position −16 from the position of A, the DNA sequence contains a promoter segment called the tata-box composed of a sequence similar to TATAAT.

The idea of a reactive automaton for the detection of a start codon based on the simplified description is analogue to what is used in previous examples. The automaton is composed of two parts. The first part detects occurrences of the promoter sequence at each position i modulo 12, the distance between the end of the promoter supposedly of length 6 and the end of the start codon. The second part detects occurrences of the start codon at each position j modulo 12. Terminal states in the second part are initially inactive. When an occurrence of the promoter is found by the first part of the automaton, a signal is sent to activate the corresponding terminal state in the second part of the automaton. When the current state overpasses this state it is again inactivated. During a run of the automaton, the current state consists of a pair of states from each part. The state is terminal if its second component is.

Fig. 20 shows how the second part of the automaton can be designed. The first part is built in the same way. Reactive links are then added as described above.

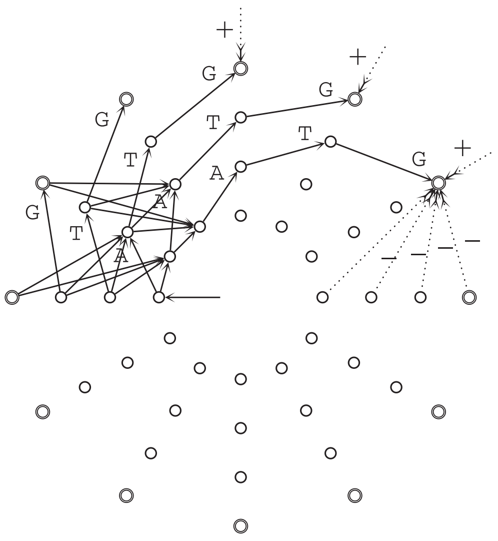

Fig. 20. Deterministic reactive automaton accepting the start codon ATG at positions i modulo 12. The automaton is partially drawn. It is composed of four circles of states, the outer circle contains states that are terminal when activated by a reactive link coming from another part of the whole machine. They are inactivated when going to the next spoke. From the inner circle, circles correspond respectively to the prefixes ε, A, AT, and ATG of the codon. Although not shown on the picture all edges have a label that corresponds to the circle of their target. For example, from states of the second outer circle, there is a G-edge that goes to a state of the larger circle, an A-edge that goes to a state of the smaller circle, and all other edges go to a state of the inner circle. The next state is on the next spoke of the wheel. Edges and reactive links from states on two different spokes of the wheel are analogue.

8.2. Open question

In some sense, reactivity competes with non-determinism to get small automata accepting a given language, although they are not antagonist concepts. Because despite results of Sections 4 and 6 showing that reactivity reduces significantly the number of states of automata, the solution requires in general a large number of extra links. But the significant examples of Section 7 raise our hope that this number can indeed be fairly small.

This article leaves open the question of whether, given a language described by a regular expression of size r, there is a reactive deterministic automaton of size $O(r)$ accepting it.

Further questions concerns other standard questions on automata, like determinisation, equivalence and implementation for example.

Acknowledgments

We are grateful to the referees for their very detailed and helpful comments on the previous version of the article.

References

[1] A.V. Aho, Algorithms for finding patterns in strings, in: Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A), Elsevier and MIT Press, 1990, pp. 255–300, ISBN 0-444-88071-2.
[2] H. Barringer, D.M. Gabbay, D. Rydeheard, Reactive grammars, in: N. Dershowitz, E. Nissan (Eds.), Language, Culture, Computation, LNCS, Springer, in press (in honour of Yakov Choueka).
[3] M.-P. Béal, M. Crochemore, F. Mignosi, A. Restivo, M. Sciortino, Computing forbidden words of regular languages, Fundamenta Informaticae 56 (1,2) (2003) 121–135.
[4] D.M. Gabbay, Reactive Kripke semantics and arc accessibility, in: W. Carnielli, F.M. Dionesio, P. Mateus (Eds.), Combinatorial Logic, Centre of Logic and Computation, University of Lisbon, 2004, pp. 7–20.
[5] D.M. Gabbay, Reactive Kripke models and contrary-to-duty obligations, in: R. van der Meyden, L. van der Torre (Eds.), Deontic Logic in Computer Science, LNAI, vol. 5076, Springer, 2008, pp. 155–173.
[6] D.M. Gabbay, Reactive Kripke semantics and arc accessibility, in: A. Avron, N. Dershowitz, A. Rabinovitch (Eds.), Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of his 85th Birthday, LNCS, vol. 4800, Springer-Verlag, Berlin, 2008, pp. 292–341.
[7] D.M. Gabbay, Reactive intuitionistic Tableaux, Synthese (special issue in honour of E.W. Beth, 2010), doi:10.1007/s11229-010-9781-8.
[8] D.M. Gabbay, A. Bochman, Reactive causation, Draft.
[9] D.M. Gabbay, A. Garcez, Modes of attack in argumentation networks, Studia Logica 93 (2–3) (2009) 199–230.
[10] D.M. Gabbay, K. Schlechta, Reactive preferential structures and nonmonotonic consequence, Review of Symbolic Logic 2 (2) (2009) 414–450.
[11] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, third ed., Addison-Wesley, 2006.
[12] M. Lothaire (Ed.), Applied Combinatorics on Words, Cambridge University Press, 2005.