A study of some easily parallellizable automata[1]

Sylvie Hamel

Département d'Informatique et de Recherche Opérationnelle
Université de Montréal, CP.6128 Succ. Centre-Ville
Montréal, Québec, Canada, H3C 3J7
phone:+1 514 343-6111(3504) fax:+1 514 343-5834
sylvie.hamel@umontreal.ca

Abstract

A vector algorithm is an algorithm that applies a bounded number of vector operations to an input vector, regardless of the length of the input. The allowable operations are usually restricted to bit-wise operations available in processors, including shifts and binary addition with carry. This paper studies automata for which it is *easy* to derive a vector algorithm to compute the sequence of states visited by the automata on a given input sequence. A special case of these automata, *shellable automata*, is used in pattern matching problems on biological sequences [2].

Keywords: automata, vector algorithms

1 Introduction

Given a deterministic, finite and complete automaton and an input sequence, we are interested in computing $\tau : \mathcal{A}^* \to Q^*$, the sequence of states visited by the automaton while reading this input. We will call this function τ a length-preserving *transduction*. Since executing one transition is usually considered to be a constant time operation, the output sequence can be obtained in $\mathcal{O}(m)$ time by simulating the run of the automaton on an input word of length m.

One way to accelerate the computations is to exploit the parallelism of bit-vector operations. For example, in [1] and [4], bit-vectors are used to code the set of states of a non-deterministic automaton. Another approach, developed in [7] and generalized in [2], uses bit-vectors to code both the input and output sequence, and computes the output with a bounded number of bit-wise operations on the input.

In [3], A. Bergeron and S. Hamel have shown that the automaton being aperiodic is a sufficient condition for the existence of a vector algorithm for the transduction problem. The fact that it is also a necessary condition was proved in [8] by O.Serre. However, for general aperiodic automata, the construction of a vector algorithm for the transduction problem relies on the cascade decomposition of the automaton, defined by Krohn and Rhodes in [5], yielding algorithms with exponential complexity in the size of the automaton. Drawbacks of this construction are that, given an automaton, there is no efficient way to obtain a cascade decomposition [6] and, moreover, deciding if an automaton is aperiodic is PSPACE-Hard [9].

In this article, we will investigate some *easily parallellizable* automata, meaning that, for these automata, the construction of a vector algorithm for the transduction problem will not rely on the cascade decomposition, but will be directly derived from the automaton. The paper is organized as follow. First we will talk about the basics of vector algorithms for the transduction problem. We will then recall how we can derive a vector algorithm for the transduction problem in a class of automaton, that was called *solvable* in [2], but that we will called *shellable* to avoid any confusion with solvable groups. We will then present vector algorithms for the transduction problem in two new classes of automata, the *k-shellable* automata and the *connected-k-shellable* automata.

2 The Basics of Vector Algorithms

2.1 Notation.

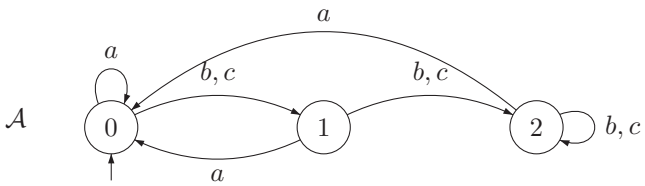
Most of our notation is standard. We will use bold letter to denote vectors. If $\mathbf{x} = x_1 \dots x_m$ and $\mathbf{y} = y_1 \dots y_m$ are two boolean vectors –or *bit vectors* – then $\mathbf{x} \vee \mathbf{y}$, $\mathbf{x} \wedge \mathbf{y}$, $\neg\mathbf{x}$ and $\mathbf{x} +_{\mathbf{b}} \mathbf{y}$ represent the bit-wise logical disjunction, conjunction, negation and binary addition with carry (that we perform from left to right, dropping the eventual last carry bit). We generalize this notation to arbitrary predicates and terms. For example, for a set S and a vector $\mathbf{e} = e_1 \dots e_m$, the expression $(\mathbf{e} \in \mathbf{S})$ is the bit vector $(e_1 \in S, \dots, e_m \in S)$, where $e_i \in S = 1$ or 0 depending on whether or

[1]with the support of NSERC and FQRNT

not the element e_i of the vector $\mathbf{e}$ is included in the set S. Finally, we need two more basic operations on vectors. First, the right *shift* by j is defined for $\mathbf{x} = x_1 \dots x_m$ to be $\uparrow_j \mathbf{x} = jx_1 \dots x_{m-1}$. Thus, the values of $\mathbf{x}$ are shifted to the right, and the first component is set to j. Second, we will need the operator $\rightarrow \mathbf{x}$ on a bit vector $\mathbf{x}$ defined as $\rightarrow \mathbf{x} = \mathbf{x} \vee [\neg(\mathbf{x} +_{\mathbf{b}} \mathbf{1})]$. This operator turns on (put 1's) in $\mathbf{x}$ all the positions to the right of the first 1. For example, if $\mathbf{x} = 00101110001$ then $\rightarrow \mathbf{x} = 00111111111$.

2.2 Vector Algorithms and the Transduction Problem.

It may be best to introduce the concept of vector algorithms for the transduction problem with an elementary example. Consider the following automaton. On input sequence $\boldsymbol{e} = bcaacbbacb$, it will generate the output $\boldsymbol{r} = 1200122012$.



We first compute the characteristic vector of each letter of the input alphabet. The *characteristic vector* of the letter a, denote $\boldsymbol{a}$, is simply the bit vector where the "on" bits indicate the positions of the letter a in the input. So given the input $\boldsymbol{e} = bcaacbbacb$, we have the following characteristic vectors; $\boldsymbol{a} = 0011000100$, $\boldsymbol{b} = 1000011001$, $\boldsymbol{c} = 0100100010$. Now, looking at the automaton $\mathcal{A}$, one sees that the output will be 0 if and only if the input letter is a, thus $(\boldsymbol{r} = \mathbf{0}) = \boldsymbol{a} = 0011000100$. Similarly, the output state is 1 if we were in state 0 and the input letter is either b or c, and so we have $(\boldsymbol{r} = \mathbf{1}) = \uparrow_1(\boldsymbol{r} = \mathbf{0}) \wedge(\boldsymbol{b} \vee \boldsymbol{c}) = 1000100010$. In all other cases, the output state is 2, and we have $(\boldsymbol{r} = \mathbf{2}) = \neg\,[(\boldsymbol{r} = \mathbf{0}) \vee (\boldsymbol{r} = \mathbf{1})] = 0100011001$.
If we assume that vector operations are done in parallel then, regardless of the length of the input sequence, the output can be computed here with only 5 vector operations (1 shift, 1 conjunction, 2 disjunctions and 1 negation). Although this example is simple, the output state depends on at most two input letters, it gives the flavor of the technique. In general, the output state will depend on arbitrarily "far" input letters and we will see in the next section how we can use the binary addition with carry bit as a memory of these past events for a special class of automata that have nice properties.

3 The Shellable Case

Let us recall the main features of the transduction problem in a *shellable* automaton. Since these results can be found in [2] all the proofs of this section are omitted. A *reset* in an automaton $\mathcal{A}$ is a transition event that induces a constant function on the states of $\mathcal{A}$. When a state s is reached only by resets than we will see that $(\mathbf{r} = \mathbf{s})$ (the positions where we are in state s given an input sequence) can easily be computed using only vector operations. We will also recall a generalization of this result that will give us a vector algorithm for the transduction problem in so called *shellable* automaton.

Definition 3.1 *Let $\mathcal{A}$ be a finite automaton on alphabet of events Σ, with states Q and transition function $F : Q \times \Sigma \longrightarrow Q$. A state s is said to be* **shellable** *if and only if for all events $x \in \Sigma$, the fact that there exists a state $s' \neq s$ such that $F(s', x) = s$ implies that x is a reset to s (for all states $q \in Q$ we have $F(q, x) = s$).*

Definition 3.2 *The* **indicator set** *I_s, of a shellable state s, is the set of resets to s.*

To lighten the notation, we will denote, given an input vector $\boldsymbol{e}$, the bit vector $\boldsymbol{e} \in \boldsymbol{I_s}$ by $\boldsymbol{I_s}$. With this notation in mind, we have the following proposition:

Proposition 3.1 **([2])** *If s is shellable, and if L_s is the set of event looping on s but not in I_s, then, if s is the initial state, the bit vector $(\boldsymbol{r} = \boldsymbol{s})$ is equal to*

$$\boldsymbol{I_s} \vee [\boldsymbol{L_s} \wedge (\neg \boldsymbol{I_s} +_{\boldsymbol{b}} \neg(\boldsymbol{I_s} \vee \boldsymbol{L_s}))]. \quad (a)$$

Otherwise, $(\boldsymbol{r} = \boldsymbol{s})$ is equal to

$$\boldsymbol{I_s} \vee [\boldsymbol{L_s} \wedge \neg(\boldsymbol{I_s} +_{\boldsymbol{b}} (\boldsymbol{I_s} \vee \boldsymbol{L_s}))]. \quad (b)$$ ■

Let $\mathcal{A}$ be a complete deterministic automaton and $\mathcal{A} \setminus \{s\}$ be the automaton obtained from $\mathcal{A}$ by removing state s, and all its pending arrows. Then if s is shellable, $\mathcal{A} \setminus \{s\}$ is still a complete automaton on the alphabet $\Sigma \setminus I_s$, since $F(r, a) \neq s$, if a is not in I_s. Note that if s is the initial state then $\mathcal{A} \setminus \{s\}$ will become a complete automaton without initial state.This is not a problem here since the initial state s will have been already taking care of by Formula (a) of Proposition 3.1.

Definition 3.3 *An automaton $\mathcal{A}$ is* **shellable** *if it has one state, or if it has one shellable state s, and $\mathcal{A} \setminus \{s\}$ is shellable.*

When an automaton $\mathcal{A}$ with d states is shellable, there is an induced ordering (not always unique) on its states, starting from the first shellable state, and then the next, and so on. We can thus relabel the states of $\mathcal{A}$ and assume that $Q = \{0, 1, \dots, d-1\}$.

Theorem 3.1 ([2]) *If $\mathcal{A}$ is a shellable automaton then we can easily derive from $\mathcal{A}$ a vector algorithm for the transduction problem, of complexity $\mathcal{O}(d|\Sigma|)$.* ■

4 k-shellable and connected-k-shellable automata

In this section, we will generalize the concept of shellable automata to k-shellable and connected-k-shellable automata. We will then derive vector algorithms for the transduction problem in these classes of automata.

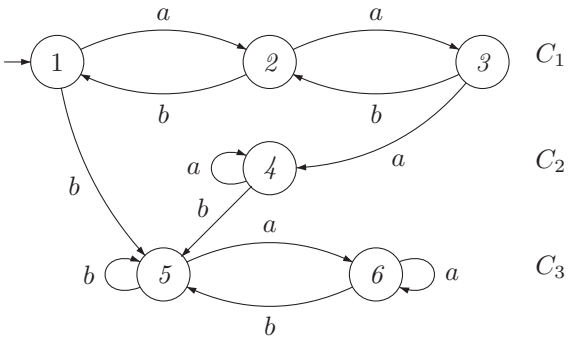
Definition 4.1 *Let $\mathcal{A}$ be a complete deterministic automaton on alphabet of events Σ with states Q and transition function $Q \times \Sigma \xrightarrow{F} Q$. A word $w \in \Sigma^*$ is said to be* **synchronizing** *in $\mathcal{A}$ if there exists a state $p \in Q$ such that for all states $q \in Q$, $q \cdot w = p$.*

Definition 4.2 *A* **state** *s is* **k-shellable** *iff for all words $x \in \Sigma^k$, the fact that there exist a state $s' \neq s$ such that $s' \cdot x = s$ implies that for all states $q \in Q$, $q \cdot x = s$. The* **indicator set** *I_s^k, of a k-shellable state s, is the set of synchronizing words w of length k reaching s. An* **automaton** *$\mathcal{A}$ is* **k-shellable** *if it has one state, or if it has one k-shellable state s, and $\mathcal{A} \setminus \{s\}$ is k-shellable.*

We will now define an even more general class of automata, called *connected-k-shellable*.

Definition 4.3 *An automaton $\mathcal{A}$ is* **connected-k-shellable** *if all of its strongly connected components C_i are t-shellable for a certain $t \leq k$, when considered as the complete automaton obtained by transforming all transitions going out of a state q_j of C_i, as a loop on q_j. If $k = 1$ we say that the automaton is* **connected-shellable**[a].

Example 4.1 *The following automaton is connected-2-shellable since its connected component C_1 is 2-shellable, while C_2 and C_3 are shellable:*



[a]The case $k = 1$ was studied in [8].

Given a connected-k-shellable automaton, there exist a partial ordering of its components described as follows. Let us call C_1 the connected component containing the initial state. Then, for $i < j$, we will say that a component C_i is visited before another component C_j, denoted $C_i \prec C_j$, if there exist a path from any state of C_i to any state of C_j. For example, in the figure above, we have $C_1 \prec C_2, C_3$ and $C_2 \prec C_3$.

4.1 Vector algorithms for k-shellable and connected-k-shellable automata

Theorem 4.1 *If $\mathcal{A}$ is a k-shellable automaton then we can easily derive from $\mathcal{A}$ a vector algorithm for the transduction problem, of complexity $\mathcal{O}(kd|\Sigma|^k)$.*

Proof. Here, we are given k, such that the automaton $\mathcal{A}$ is k-shellable. Suppose that we also have the generalized transition function $F^k : Q \times \Sigma^k \to Q$. Let $Q = \{0, 1, \ldots, d-1\}$ be an ordering of the states of $\mathcal{A}$ such that $\mathcal{A} \setminus \{0, 1, \ldots, \ell - 1\}$ is k-shellable for state ℓ, and let i be the initial state of $\mathcal{A}$. Given the input sequence $\mathbf{e} = e_1 e_2 \ldots e_m$, we will recursively compute the output $\mathbf{r} = r_1 r_2 \ldots r_m$ using intermediate computations of the bit vectors $(\mathbf{r} = \ell)$, for each ℓ in $\{0, 1, \ldots, d-1\}$, in k steps:

1. Starting with the initial state i of the automaton and reading the input with a window of length k, we can find the bits in position km, $m \geq 1$ of $(\mathbf{r} = \ell)$ using Theorem 3.1.
2. Beginning in state $F(i, e_1) = i_2$ of the automaton, and reading the input with a window of length k, starting at e_2, we can find the bits in position $(k+1)m$, $m \geq 1$ of $(\mathbf{r} = \ell)$ using Theorem 3.1. Set the first bit of the vector $(\mathbf{r} = \mathbf{i_2})$ to 1 (since after reading e_1 we will be in that state).

   ...

k. Finally, beginning in state $i \cdot e_1 e_2 \ldots e_{k-1} = i_k$ of the automaton and reading the input with a window of length k, starting at e_k, we can find the bits in position $(k + k - 1)m$, $m \geq 1$ of $(\mathbf{r} = \ell)$ using Theorem 3.1. Set the $k - 1^{th}$ bit of the vector $(\mathbf{r} = \mathbf{i_k})$ to 1.

Each step described here requires a constant number of vector operations, apart from the test $\mathbf{F^k}(\uparrow_{\mathbf{i}} \mathbf{r}, \mathbf{e}) = \ell$, which requires $|\Sigma|^k$ steps (once the vector $(\mathbf{r} = \ell)$ is known, we can form the con-

juction, for each word $w \in \Sigma^k$, $(\uparrow_{\mathbf{i}} \mathbf{r} = \ell) \wedge (\mathbf{e} = \mathbf{w})$, and then look-up the value of $F^k(\ell, w)$ in the transition table). Since we compute the d vectors $(\mathbf{r} = \ell)$ in k of these steps, the complexity of the algorithm is $\mathcal{O}(kd|\Sigma|^k)$. ■

Theorem 4.2 *If $\mathcal{A}$ is a connected-k-shellable automaton then we can easily derive from $\mathcal{A}$ a vector algorithm for the transduction problem, of complexity $\mathcal{O}(kd|\Sigma|^k)$.*[a]

Proof. First, let us partially order the connected components of $\mathcal{A}$ as described in section 4.2. Given this order, we will now show that we can compute, given an input $\mathbf{e} = e_1 \dots e_n$, when we are in each of the states of a component C_i given that we have already computed when we are in each of the states of the components visited before C_i. This intuitively comes from the fact that when we go out of a component we never come back to it and we always go to a "lower" component in the partial ordering.

Computing when we are in the states of component $\mathbf{C_1}$: Let $q_1, \dots q_{n_1}$ be the states in C_1. We want a vector algorithm that will compute the characteristic vectors $(\mathbf{r} = \mathbf{q_j})$, for $1 \leq j \leq n_1$, given an input $\mathbf{e}$. To do that, we first consider component C_1 as a complete t-shellable automaton $(t \leq k)$ by considering all transitions going out of a state q_j of C_1, as a loop on q_j. Using Theorem 4.1 we can compute with vector algorithms the characteristic vectors $(\mathbf{r_{C_1}} = \mathbf{q_j})$, for $1 \leq j \leq n_1$, and these vectors are equals to the vectors $(\mathbf{r} = \mathbf{q_j})$ iff on input $\mathbf{e}$ we never go out of component C_1. Now, if we do not stay in C_1 on input $\mathbf{e}$, there exists some transitions a_j and states $q_j \in C_1$ such that $q_j \cdot a_j \notin C_1$. Let the vector OUT_{C_1} be the disjunction of all these events:

$$OUT_{C_1} = \bigvee_{j=1}^{h} [(\uparrow_{\mathbf{i}} \mathbf{r_{C_1}} = \mathbf{q_j}) \wedge (\mathbf{e} = \mathbf{a_j})],$$

with $i = 1$, if q_j is the initial state, and 0 otherwise. Then the vector $\rightarrow OUT_{C_1}$ contains a sequence of 1's starting at the first position in $\mathbf{e}$, where we go out of C_1, up to the end of the vector, i.e in all positions we are out of C_1 on input $\mathbf{e}$. So, we can now compute the vectors $(\mathbf{r} = \mathbf{q_j})$, for $1 \leq j \leq n_1$ with the formula:

$$(\mathbf{r} = \mathbf{q_j}) = (\mathbf{r_{C_1}} = \mathbf{q_j}) \wedge \neg(\rightarrow OUT_{C_1}).$$

Computing when we are in the states of component $\mathbf{C_m}$ given that we know when we are in the states of components $\mathbf{C_1}, \dots \mathbf{C_{m-1}}$: The first thing to do here is to compute if, reading the input $\mathbf{e}$, we will enter component C_m are not. To do that, let us consider the following vector of all events a_j going in C_m:

$$IN_{C_m} = \bigvee_{j=1}^{h} [(\uparrow_{\mathbf{i}} \mathbf{r} = \mathbf{q_j}) \wedge (\mathbf{e} = \mathbf{a_j})].$$

All the vectors $(\mathbf{r} = \mathbf{q_j})$ appearing in this formula have already been computed since all the q_j are in one of the components $\mathbf{C_1}, \dots \mathbf{C_{m-1}}$ by the partial ordering of the components. If the vector IN_{C_m} is null than, on input $\mathbf{e}$, we never enter component C_m of the automaton and for each state s of C_m we have that the vector $(\mathbf{r} = \mathbf{s})$ is null.

Otherwise, let $p_1, \dots p_{n_m}$ be the states in C_m. Again, we consider component C_m as a complete automaton by considering, as for C_1, all the transitions going out of C_m, from a state p_j, as a loop on p_j. The initial state of C_m is the state where we enter C_m, while reading the input $\mathbf{e}$. Also, we want to begin the lecture of the input e where we left it entering C_m. To do that, we will introduce an "already read" marker $\mathcal{R}$, $\mathcal{R} \notin \Sigma$, and a loop labeled $\mathcal{R}$ on each state of the component C_m. (C_m is now a complete automaton on the alphabet $\Sigma \cup \mathcal{R}$.) We will then transform the input e to e', where $e'_j = \mathcal{R}$, in all positions j that are already read and $e'_j = e_j$, otherwise. Since the vector IN_{C_m} gives us the position where we go in C_m, we easily get the following formulas for e':

$$(\mathbf{e'} = \mathcal{R}) = \uparrow_1 \neg(\rightarrow IN_{C_m}) \text{ and}$$

$$(\mathbf{e'} = \mathbf{a}) = [(\mathbf{e} = \mathbf{a}) \wedge \uparrow_{\mathbf{0}} (\rightarrow IN_{C_m})], \forall a \in \Sigma.$$

With this e' and Theorem 4.1 we can then compute the characteristic vectors $(\mathbf{r_{C_m}} = \mathbf{p_j})$, for $1 \leq j \leq n_m$ and the vector OUT_{C_m}. It is easy to see that we have the following formulas for $(\mathbf{r} = \mathbf{p_j})$, for $1 \leq j \leq n_m$:

$$(\mathbf{r} = \mathbf{p_j}) = \rightarrow IN_{C_m} \wedge (\mathbf{r_{C_m}} = \mathbf{p_j}) \wedge \neg(\rightarrow OUT_{C_m}).$$

Now, let us talk about the complexity of the algorithm. Each state s of a connected-k-shellable automaton belongs to exactly one component C_i. To compute $(\mathbf{r} = \mathbf{s})$ we have to compute three bit vectors: IN_{C_i}, OUT_{C_i} and $(\mathbf{r_{C_i}} = \mathbf{s})$. Since IN_{C_i} and OUT_{C_i} can be computed with $\mathcal{O}(d|\Sigma|)$ operations and $(\mathbf{r_{C_i}} = \mathbf{s})$ is computed using Theorem 4.1, we have that the complexity of the algorithm is $\mathcal{O}(kd|\Sigma|^k)$. ■

Example 4.2 *Given the connected-2-shellable automaton $\mathcal{A}$ of Example 4.1 and the input $\mathbf{e} =$ abaababbabbaba we should find the output vector* $\mathbf{r}$=21232321215656. *The first thing to do is to consider C_1 as a complete 2-shellable automaton and compute the vectors $(\mathbf{r_{C_1}} = \mathbf{q_j})$ for all state $q_j \in C_1$. Using Theorem 5.1 we get that*

[a] The case $k = 1$ was proved in [8]. Here, we generalize this result to an arbitrary k.

$(\mathbf{r_{C_1}} = \mathbf{1})$=01000001011010, $(\mathbf{r_{C_1}} = \mathbf{2})$ = 10101010100101 *and* $(\mathbf{r_{C_1}} = \mathbf{3})$=00010100000000. *Now, there is two ways to go out of component* C_1*; we are in state 1 and follow the transition b or we are in state 3 and follow the transition a. These two events gives us the following vector:*

OUT_{C_1} =$[\uparrow_{\mathbf{1}} (\mathbf{r_{C_1}} = \mathbf{1}) \wedge (\mathbf{e} = \mathbf{b})]$
$\vee[\uparrow_{\mathbf{0}} (\mathbf{r_{C_1}} = \mathbf{3}) \wedge (\mathbf{e} = \mathbf{a})]$
= 00000000001000

Now, $\rightarrow OUT_{C_1}$ = 00000000001111*, which gives us the following vectors for state 1, 2 and 3:*

$(\mathbf{r} = \mathbf{1})$=$(\mathbf{r_{C_1}} = \mathbf{1}) \wedge \neg(\rightarrow OUT_{C_1})$ = 01000001010000
$(\mathbf{r} = \mathbf{2})$=$(\mathbf{r_{C_1}} = \mathbf{2}) \wedge \neg(\rightarrow OUT_{C_1})$ = 10101010100000
$(\mathbf{r} = \mathbf{3})$=$(\mathbf{r_{C_1}} = \mathbf{3}) \wedge \neg(\rightarrow OUT_{C_1})$=00010100000000.

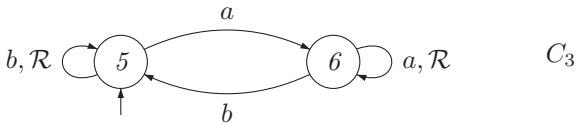
Now that we are done with all the states of component C_1*, let us see if we enter component* C_2 *while reading* $\mathbf{e}$*. The only transition that goes in* C_2 *from an other component is the transition a going from state 3 to state 4. That gives us the following vector:*

$$IN_{C_2} = (\uparrow_{\mathbf{0}} \mathbf{r} = \mathbf{3}) \wedge (\mathbf{e} = \mathbf{a}) = 00000000000000.$$

Since IN_{C_2} *is the null vector, we will never enter component* C_2*, while reading* $\mathbf{e}$*, and so we have* $(\mathbf{r} = \mathbf{4})$ = 00000000000000*. Finally, for the last component* C_3 *we can compute easily the vectors*

$$IN_{C_3} = [(\uparrow_{\mathbf{1}} \mathbf{r} = \mathbf{1}) \wedge (\mathbf{e} = \mathbf{b})] \vee [(\uparrow_{\mathbf{0}} \mathbf{r} = \mathbf{4}) \wedge (\mathbf{e} = \mathbf{b})]$$
$$= 00000000001000$$
$$OUT_{C_3} = 00000000000000.$$

What we have to do now is compute the vectors $(\mathbf{r_{C_3}} = \mathbf{5})$ *and* $(\mathbf{r_{C_3}} = \mathbf{6})$*, on input* $e' = \mathcal{R}RRRRRRRRRRaba$*, where* C_3 *is the following complete automaton on alphabet* $\Sigma \cup \mathcal{R}$*, with initial state 5 (we enter* C_3 *by state 5):*



Since, C_3 *is shellable, we use Theorem 3.1 and input* e' *to get the vectors* $(\mathbf{r_{C_3}} = \mathbf{5})$ = 11111111111010 *and* $(\mathbf{r_{C_3}} = \mathbf{6})$ = 00000000000101*. Finally, with the formula* $(\mathbf{r} = \mathbf{p_j}) = (\rightarrow \mathbf{IN_{C_3}}) \wedge (\mathbf{r_{C_3}} = \mathbf{p_j}) \wedge \neg(\rightarrow \mathbf{OUT_{C_3}})$ *we get* $(\mathbf{r} = \mathbf{5})$ = 00000000001010 *and* $(\mathbf{r} = \mathbf{6})$ = 00000000000101.

5 Conclusion

We have shown that k-shellable and connected-k-shellable automata are easily parallellizable, meaning that vector algorithms for the transduction problem in these automata can be directly derived from their transition table. The complexity of these algorithms is independent of the length of the input. We have also shown that it is possible to decide membership for both of these classes. In [2], A. Bergeron and the author have shown that, for some special shellable automata appearing in an approximate string matching algorithm, it was possible to accelerate the computation by exploiting certain arithmetic properties of the transition table of the automaton. This gave them a vector algorithm with complexity depending only on the number of states of the automaton. Recall that in the general case, the complexity of the algorithm depends on the number of states and transitions of the automaton. So, one interesting question here is the following. Can we describe formally the properties of the transition table of an automaton that will gives us an $\mathcal{O}(d)$ vector algorithm for the transduction problem in a shellable automaton with d states? If so, can we generalize these properties to words to get $\mathcal{O}(kd)$ vector algorithm for the transduction problem in k-shellable are connected k-shellable automata?

References

[1] R. A. Baeza-Yates and G. H. Gonnet, *A New Approach to Text Searching*, Communications of the ACM, 35, (1992), 74-82.

[2] A. Bergeron and S. Hamel, *Vector Algorithms for Approximate String Matching*, Inter. J. of Found. of Comp. Sc., 13-1, (2002), 53–66.

[3] A. Bergeron and S. Hamel, *From cascade decomposition to bit-vector algorithms*, Theoretical Computer Science, 313, (2004), 3–16.

[4] J. Holub and B. Melichar, *Implementation of Nondeterministic Finite Automata for Approximate Pattern Matching*, LNCS 1660, (1999), 92-99.

[5] K. Krohn and J. L. Rhodes, *Algebraic Theory of machines*, Trans. of the American Math. Soc., 116, (1965), 450-464.

[6] O. Maler and A. Pnueli, *Tight Bounds on the Complexity of Cascaded Decomposition Theorem*, Annual Symp. on Found. of Comp. Sc. IEEE, vol. II, (1990), 672-682.

[7] G. Myers, *A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming* , J. ACM, 46-3, (1999), 395-415.

[8] O. Serre, *Vectorial Languages and Linear Temporal Logic*, to appear in Theo.Comp. Sc.

[9] J. Stern, *Complexity of some Problems from the Theory of Automata*, Inform. and Control, 66, (1985), 163–176.