Combinatorial Characterization of the Language Recognized by Factor and Suffix Oracles

Alban Mancheron and Christophe Moan

L.I.N.A., Université de Nantes, 2, Rue de la Houssinière, B.P. 92208,
44322 Nantes Cedex 3, France
e-mail: {Mancheron, Moan}@lina.univ-nantes.fr

Abstract. Sequence Analysis requires to elaborate data structures which allow both an efficient storage and use. Among these, we can cite Tries [1], Suffix Automata [1, 2], Suffix Trees [1, 3]. Cyril Allauzen, Maxime Crochemore and Mathieu Raffinot introduced [4, 5, 6] a new data structure, linear on the size of the represented word both in time and space, having the smallest number of states, and allowing to accept at least all the substrings of the represented word. They called such a structure a *Factor Oracle*. On the basis of this structure, they developed another one having the same properties excepting the accordance of all the suffix of the represented word. They called it *Suffix Oracle*.
The characterization of the language recognized by the Factor/Suffix Oracle of a word is an open problem for which we provide a solution.

Keywords: Factor Oracle, Suffix Oracle, automata, language, characterization.

1 Introduction

Within text indexation, several structures were developed. The objective of these methods is to represent a text or a word s, *ie.* a succession of symbols taken in an arbitrary alphabet denoted by Σ, in order to "quickly" determine whether this word contains some specific sub-word. In which case, we call this sub-word a *factor* of s.

Cyril Allauzen, Maxime Crochemore and Mathieu Raffinot described a method allowing to build an *acyclic* automaton, accepting at **least** the factors of s, having as few states as possible ($|s|+1$), and being *linear* in the number of transitions ($2\,|s|-1$). They named such an automaton a *Factor Oracle*.

In this automaton, each state is final. Using the same automaton, but only keeping "particular" states as final, one obtains a *Suffix Oracle*.

This structure has several advantages. First of all, the construction algorithm is easy to understand and implement; this is not the case of the most efficient algorithm for building Suffix Tree's. Next, Oracles are homogeneous automata (*ie.* all the transitions going to the same state are labeled with the same symbol). That means that we do not need to label edges. This makes this structure very sparing in memory (much more than Suffix Trees or Tries). Indeed, methods based upon this structure obtain good results. Thus, Lefebvre & *al.* [7, 8, 9] use it for repeated motifs discovery over large genomic data, and obtain results similar to the one obtained using thousands of BLASTn requests, but in a few seconds. They also use the Factor Oracle in text compression [10], and in some cases they have compression ratio comparable to bzip2 (which is one of the most efficient compression algorithm).

Nevertheless, at least two problems linked to these Oracles are still opened: the first one is the characterization of the language recognized by Oracles; the second one is: does there exist an algorithm, linear in time and space, to build an automaton accepting at least the factors/suffixes of a word s being minimal in number of transitions?

The first open problem is really important. Currently, the main difficulty when using Oracles is to distinguish true positives from false positives. That is why we are interested in the first problem. In the following section, we provide several definitions relating to the construction of Oracles. Then we give the characterization of the language recognized by this structure. To conclude, we show some results about the Oracles.

2 Definitions

Subsequently, we use the notations hereafter (some of them are issued from [4, p. 2]): we denote by $Fact(s)$ (resp. $Suff(s)$ and $Pref(s)$) the set of the factors (resp. suffixes and prefixes) of $s \in \Sigma^+$, by $Pref_s(i)$ the prefix of s having length $i \geq 0$. Given $x \in Fact(s)$, we denote by $Nb_s(x)$ the number of occurrences of x in s, and we say that x is *repeated* if $Nb_s(x) \geq 2$.

Definition 2.1 Given a word $s \in \Sigma^+$ and x a factor of s, we define the function Pos as the position of the first occurrence of x in $s = uxv$ ($u, v \in \Sigma^*$) such that x is not repeated in ux): $Pos_s(x) = |u| + 1$. We also define the function *poccur* such that $poccur_s(x) = |u| + |x| = Pos_s(x) + |x| - 1$ (denoted by $poccur(x, s)$ in [4, p. 2]).

In the following, we define the Oracles, then we give some notations and definitions peculiar to factors, as well as properties about the newly defined objects. Finally, in order to characterize the language recognized by Oracles, we define particular factors and then operations linked to them.

2.1 Oracles

We give below the algorithm of Allauzen & *al.* [4] which describes the Oracle construction (*cf.* algorithm 1). In the same paper, authors give another algorithm which allows to build the same automaton in linear time on the size of s. Nevertheless, because we are only interested in the properties of the Oracle, we do not give it in this paper.

Definition 2.2 [4, pp. 2, 10] Given a word $s \in \Sigma^*$, we define the *Factor Oracle* of s as the automaton obtained by the algorithm 1 (p. 141), where all the states are final. It is denoted by $FO(s)$. We define the *Suffix Oracle* of s as the automaton obtained by the same algorithm, where are final only the states such that there exists a path from the initial state recognizing a suffix of s. It is denoted by $SO(s)$.

Notation 2.1 Given a word $s \in \Sigma^*$, we use the term *Oracle* to indifferently indicating $SO(s)$ or $FO(s)$, and we denote it by $O(s)$.

Algorithm 1: Construction of the Factor Oracle of a word[1]

```
1  Input: Σ % Alphabet (supposed minimal) %
2         s ∈ Σ* % The word to process %
3  Output: Oracle % Factor Oracle of s %
4
5  Begin
6    Create the initial state labeled by e_0
7
8    For i from 1 to |s| Do
9      Create a state labeled by e_i
10     Build a transition from the state e_{i-1} to the state e_i labeled by s[i]
11   End For
12
13   For i from 0 to |s|-1 Do
14     Let u be a word of minimal length recognized in the state e_i
15     For All α ∈ Σ \ {s[i+1]} Do
16       If uα ∈ Fact(s[i-|u|+1..|s|]) Then
17         j ← poccur_{s[i-|u|+1..|s|]}(uα) - |u|
18         Build a transition from the state e_i to e_{i+j} labeled by α
19       End If
20     End For All
21   End For
22 End
```

We have an order relation between states in these Oracles. Indeed, if we have two states e_i and e_j such that $i \leq j$, we can say that $e_i \leq e_j$.

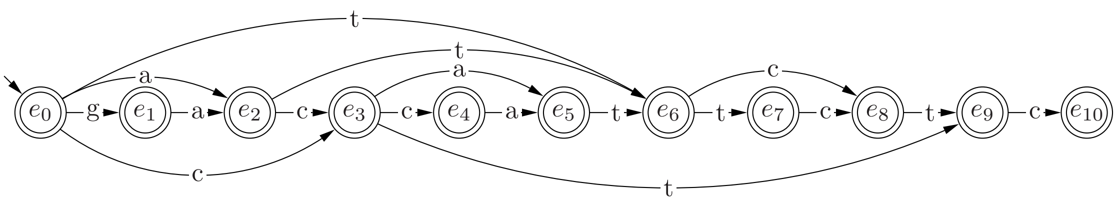

Figure 1: Factor Oracle of the word *gaccattctc*.

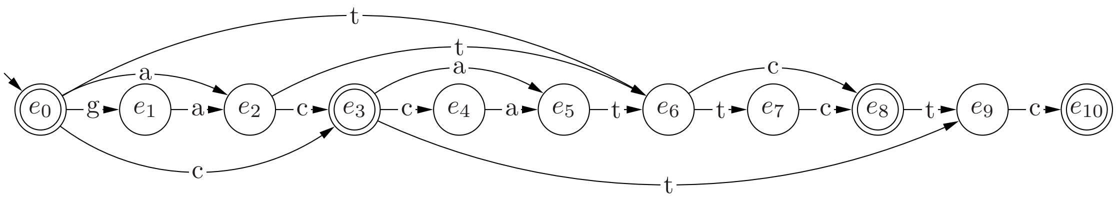

Figure 2: Suffix Oracle of the word *gaccattctc*.

[1]As mentioned in [11], the term $-|u|$ (line 17) is unfortunately missing in the original algorithm.

Definition 2.3 Given a word $s \in \Sigma^*$ and a word x accepted in the state e_i $(0 \leq i \leq |s|)$ by the Oracle of s, we define the function *State* as $State(x) = e_i$.

Lemma 2.1 *[4, pp. 2, 3] Given a word $s \in \Sigma^*$ and its Oracle, there is a unique word having minimal length accepted at each state e_i $(0 \leq i \leq |s|)$ of $O(s)$. It is denote it by $min(e_i)$.*

Lemma 2.2 *[4, pp. 2, 3] Given a word $s \in \Sigma^*$, its Oracle and an integer i $(0 \leq i \leq |s|)$, then $min(e_i) \in Fact(s)$ and $i = poccur_s(min(e_i))$.*

Notation 2.2 Given a word $s \in \Sigma^*$, we denote by $\#_{in}(e_i)$ (resp. $\#_{out}(e_i)$) the number of ingoing (resp. outgoing) transitions in the state e_i $(0 \leq i \leq |s|)$ of the Oracle of s.

2.2 Canonical Factors & Contraction Operation

We first introduce some definitions about particular factors from a given word. We use such factors for defining the contraction operation, as well as properties peculiar to this operation. We next define the sets of words we obtain applying this operation. At the end of this section, all that we need to characterize the language of Oracles will be defined.

Definition 2.4 Given a word $s \in \Sigma^*$ and its Oracle, we define the set of *Canonical Factors* of s as following:

$$\mathcal{F}_s = \{min(e_i) \mid 1 \leq i \leq |s| \ \wedge \ (\#_{out}(e_i) > 1 \ \vee \ \#_{in}(e_i) > 1)\}$$

Given a suffix t of s and a Canonical Factor f of s, we say that f is a *conserved* Canonical Factor of s in t if the first occurrence of f in s is contained in t. We denote by $\mathcal{F}_{s,t}$ the set of conserved Canonical Factors of s in t (thus $\mathcal{F}_{s,t} \subseteq \mathcal{F}_s$).

These particular factors enable us to define a set of couple of specific positions in the word s. Those will be used in order to derive new words from s.

Definition 2.5 Given a word $s \in \Sigma^*$ and a Canonical Factor f of s such that:

$$\begin{cases} s & = & ufv & (u, v \in \Sigma^*) \\ fv & = & wfx & (w \in \Sigma^+, x \in \Sigma^*) \\ Pos_s(f) & = & |u| + 1 & \end{cases}$$

then we call the pair $(|u| + 1, |uw| + 1)$ a *contraction* of s by f, and $s' = ufx$ is the result of this contraction.

Notation 2.3 Given a word $s \in \Sigma^*$ and a Canonical Factor $f \in \mathcal{F}_s$, we denote by $\mathcal{C}_s^f$ the set of the contractions of s by f. We denote the set of all the contractions we can operate on s by $\mathcal{C}_s^*$ $(\equiv \bigcup_{f \in \mathcal{F}_s} \mathcal{C}_s^f)$. Let t be a suffix of $s = t't$ $(t' \in \Sigma^*)$, we denote by $\mathcal{C}_{s,t}^*$ the subset of $\mathcal{C}_s^*$ such that $\mathcal{C}_{s,t}^* = \{(p', q') \mid (p, q) \in \mathcal{C}_s^* \ \wedge \ p > |t'| \ \wedge \ (p', q') = (p - |t'|, q - |t'|)\}$.

Since contractions will be used to produce new words, we only need to consider a subset of the set of contractions.

Definition 2.6 A set $\mathcal{C}$ of contractions is *coherent* if and only if it does not contain two contractions (i_1, j_1), (i_2, j_2) such that: $i_1 < i_2 < j_1 < j_2$. Furthermore, we say that $\mathcal{C}$ is *minimal* if and only if it does not contain two contractions (i_1, j_1) and (i_2, j_2) such that $i_1 \leq i_2 < j_2 \leq j_1$ or such that $i_1 < j_1 = i_2 < j_2$.

Now we can define the operation that, given a word, allows us to build some new specific words.

Definition 2.7 Given a word $s \in \Sigma^*$ and a coherent and minimal set of contractions $\mathcal{C} = \{(p_1, q_1), \ldots, (p_k, q_k)\}$ (associated to the set of canonical factors $\{f_1, \ldots, f_k\}$), then we define the function $Word$ as following:

$$\begin{aligned} Word(s, \mathcal{C}) &= s[1..p_1 - 1]\, s[q_1..p_2 - 1] \ldots s[q_{k-1}..p_k - 1]\, s[q_k..|s|] \\ &= s[1..p_1 - 1]\, f_1\, s[q_1 + |f_1|..p_2 - 1] \ldots\, f_k\, s[q_k + |f_k|..|s|] \end{aligned}$$

We call this sequence *the result of the contractions from* $\mathcal{C}$ *applied to* s.

From now, we only consider coherent and minimal sets of contractions (since we are interested in the results of contractions, it is easy to see why other sets don't need to be considered anymore). Let us notice that whatever the order of contraction, the obtained word remains the same.

Definition 2.8 We define $\mathcal{E}(s) = \bigcup\limits_{\mathcal{C} \subseteq \mathcal{C}^*_s} Word(s, \mathcal{C})$, and we call this set the *closure* of s.

To illustrate the various definitions given above, we take the example *gaccattctc* (*cf.* figures 1 and 2). Then the set of Canonical Factors is $\mathcal{F}_{gaccattctc} = \{a, c, ca, t, tc, ct\}$, and $\mathcal{C}^*_{gaccattctc} = \{(2,5), (3,4), (3,8), (3,10), (6,7), (6,9), (7,9)\}$. Let $\mathcal{C} = \{(2,5), (7,9)\}$ ($\mathcal{C} \subseteq \mathcal{C}^*_{gaccattctc}$). Hence $Word(gaccattctc, \mathcal{C}) = g\cancel{\boldsymbol{a}}\cancel{c}\cancel{c}\boldsymbol{a}t\cancel{\boldsymbol{t}}\cancel{\boldsymbol{c}}\boldsymbol{tc} = gattc$. The closure of *gaccattctc* is:

$$\mathcal{E}(gaccattctc) = \left\{ \begin{array}{l} gac, gacatc, gacatctc, gacattc, gacattctc, gaccatc, gaccatctc, \\ gaccattc, gaccattctc, gactc, gatc, gatctc, \boldsymbol{gattc}, gattctc \end{array} \right\}$$

3 Characterization of the language recognized by Oracles

Given a word $s \in \Sigma^*$, we saw how to build the corresponding Factor (resp. Suffix) Oracle. This Oracle allows to recognize at least all the factors (resp. suffixes) of s. Nevertheless, it accepts a certain number of additional words too. For example the word *atc* is accepted by the Factor (resp. Suffix) Oracle of *gaccattctc* (*cf.* figures 1 and 2), whereas it is either a factor nor a suffix of *gaccattctc*. We defined above the set $\mathcal{E}(s)$. In this part, we show that the Suffix Oracle exactly recognizes all the suffixes of the words from $\mathcal{E}(s)$. Then, we use this result to show that the Factor Oracle recognizes exactly all the factors of the words from $\mathcal{E}(s)$.

We first recall some useful lemmas of [4].

Lemma 3.1 *[4, p. 3] Given a word $s \in \Sigma^*$ and an integer i $(0 \leq i \leq |s|)$, then $min(e_i)$ is suffix of all word recognized in the state e_i of the Oracle of s.*

Lemma 3.2 *[4, p. 4] Given a word $s \in \Sigma^*$ and a factor w of s, then w is recognized in the state e_i $(1 \leq i \leq poccur_s(w))$ of the Oracle of s.*

Lemma 3.3 *[4, p. 4] Given a word $s \in \Sigma^*$ and an integer i $(0 \leq i \leq |s|)$, then every path ending by $min(e_i)$ in the Oracle of s leads to a state e_j such that $j \geq i$.*

Lemma 3.4 *[4, p. 5] Given a word $s \in \Sigma^*$ and $w \in \Sigma^*$ a word accepted by the Oracle of s in state e_i, then every suffix of w is also recognized by the Oracle in state e_j such that $j \leq i$.*

The proof of this last Lemma is given in [4] only for the Factor Oracle. We need to extend this result for the Suffix Oracle.

Proof (Lemma 3.4)
If we denote by x a suffix of w, the original Lemma gives us that $State(x) \leq State(w)$. We need to prove that if $State(w)$ is final, then $State(x)$ is final. In order to do this, we have to consider two cases:

Case 1: $|x| \geq |min(e_i)|$
That means that $min(e_i) \in Suff(x)$, thus according to Lemma 3.3, we can conclude that $State(x) \geq State(min(e_i))$, and since $State(min(e_i)) = e_i = State(w)$, then $State(x) = State(w)$.

Case 2: $|x| < |min(e_i)|$
The state e_i being final means that there exists a suffix t of s such that $State(t) = e_i$. According to Lemma 3.1, we deduce that $min(e_i) \in Suff(t) \subseteq Suff(s)$. Since x and $min(e_i)$ are suffixes of w, then $|x| < |min(e_i)| \Rightarrow x \in Suff(min(e_i))$. So x is also suffix of s and, by Definition of the Suffix Oracle, $State(x)$ is final. □

Before tackle demonstrations, we present two lemmas dealing with properties linked to Canonical Factors.

Lemma 3.5 *Given a word $s \in \Sigma^*$, a Canonical Factor $f \in \mathcal{F}_s$ such that $s = ufv$ $(u, v \in \Sigma^*)$ and f is not repeated in uf, and $\mathcal{C} \in \mathcal{C}^*$ a set of contractions. If there exists $w \in \Sigma^*$ such that $Word(uf, \mathcal{C}) = wf$ then wf and f are recognized in the same state in the Oracle of s.*

Proof (Lemma 3.5)
We denote by $\mathcal{C}_i \subseteq \mathcal{C}^*_s$ a set of contractions having cardinality i. In the same way, we denote by $w_i f$ the word obtained applying contractions $\mathcal{C}_i$ to uf (warning: $w_i f = Word(uf, \mathcal{C}_i) \not\Rightarrow w_i = Word(u, \mathcal{C}_i)$). Let us show by induction on the size of $\mathcal{C}_i$ that $State(Word(uf, \mathcal{C}_i)) = State(f)$ $(\forall\ \mathcal{C}_i \in \mathcal{C}^*_s)$.
Let $e_x = State(f)$ ($f = min(e_x)$ by Definition of f) and $e_{x'_i} = State(Word(uf, \mathcal{C}_i))$. If we consider $\mathcal{C}_0$, then $Word(uf, \mathcal{C}_0) = uf$. According to Lemma 3.3, $x'_0 \geq x$. Furthermore, according to Lemma 3.2 applied to uf, we have $x'_0 \leq poccur_s(uf)$. However by Definition of f, $poccur_s(f) = |uf| = poccur_s(uf)$. This implies $x'_0 \leq x$, and finally $x'_0 = x$.

Let us show now that if this lemma is true for a set of contractions $\mathcal{C}_i \subset \mathcal{C}_s^*$, then it is true for a set $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{(p,q)\}$. We assume without loss of generality that (p,q) is the last contraction (by ascending order over the positions) in $\mathcal{C}_{i+1}$. Let b the Canonical Factor used by this contraction. We can write $uf = s[1..p-1]\, s[p..q-1]\, s[q..|uf|]$. Since we choose (p,q) being the last contraction, all the contractions in $\mathcal{C}_i$ are applicable to $s[1..p-1]$. So there exists $a, c \in \Sigma^*$ such that $w_i f = a\, s[p..|uf|] = abc$, and $d \in \Sigma^*$ such that $w_{i+1} f = a\, s[q..|uf|] = abd$. We also could write $ab = Word(s[1..p-1]\, b, \mathcal{C}_i)$ (the opposite would mean that the contraction (p,q) can't be operate from b), and according to the induction hypothesis, we have $State(ab) = State(b)$. From this, we deduce that $State(abc) = State(bc)$ and $State(abd) = State(bd)$. Since $bd(= s[q..|uf|])$ is a suffix of $bc(= s[p..|uf|])$, according to the Lemma 3.4:

$$\begin{array}{rlcl} & State(bd) & \leq & State(bc) \\ \Leftrightarrow & State(abd) & \leq & State(abc) \\ \Leftrightarrow & State(w_{i+1}f) & \leq & State(w_i f) \\ \Leftrightarrow & State(w_{i+1}f) & \leq & State(f) \end{array}$$

But, according to Lemma 3.3, we have $State(w_{i+1}f) \geq State(f)$, consequently we obtain $State(w_{i+1}f) = State(f)$. So, this lemma is true for all $\mathcal{C}_i \subseteq \mathcal{C}_s^*$. □

Lemma 3.6 *Let s be word in Σ^*, $O(s)$ be its Oracle, and e_i be a state of $O(s)$ such that $u = min(e_i)$ and $u \in \mathcal{F}_s$. Let p be a transition issued from e_i labeled by α to a state e_{i+j} $(j > 1)$. Then there exists at the position $(i + j - |u|)$ of s an occurrence of $u\alpha$. Moreover, we have the contraction $(i - |u| + 1, i + j - |u|)$ of s by u.*

Proof (Lemma 3.6)
By construction (*cf.* algorithm 1), the transition p from e_i to e_{i+j} is added because there exists a position j in $s[i - |u| + 1..|s|]$ such that: $j = poccur_{s[i-|u|+1..|s|]}(u\alpha) - |u|$. We also have $u\alpha \in Fact(s)$ since $u\alpha \in Fact(s[i - |u| + 1|s|])$. Cleophas & *al.* [11] have proved that since $u = min(e_i)$ and $u\alpha \in Fact(s)$, then $i - |u| + poccur_{s[i-|u|+1..|s|]}(u\alpha) = poccur_s(u\alpha)$. Hence, we have $i + j = poccur_s(u\alpha)$, and finally $s[i + j - |u|, i + j] = u\alpha$. □

Algorithm 2: Obtaining the contractions generating w starting from t in the Oracle of s

Initialization: $S^0 = t$, $S^0_w = w$, $\mathcal{C}_0 = \emptyset$, $sdec = |s| - |t|$ % t is a suffix of s %

Input: $S^i \in \Sigma^*$ % A suffix of s that can still be ''contracted'' %
$S^i_w \in \Sigma^*$ % The word to process %
$\mathcal{C}_i$ % Set of contractions %
Output: a set of contractions

Begin
$p_i \leftarrow$ longest common prefix between S^i and S^i_w (Property 3.1, item 1)
$e_{r_i} \leftarrow State(p_i)$ (Property 3.1, item 2)
$f_i \leftarrow min(e_{r_i})$
If $(|p_i| < |S^i_w|)$ **Then**
$e_{r'_i} \leftarrow$ Transition$(e_{r_i}, S^i_w[|p_i| + 1])$ (Property 3.1, item 4)
$\mathcal{C}_{i+1} \leftarrow \mathcal{C}_i \cup \{c_i\}$, $c_i = (r_i - |f_i| + 1 - sdec, r'_i - |f_i| - sdec)$ (Property 3.2, item 2)

```
15        S_w^{i+1} ← S_w^i[|p_i| − |f_i| + 1..|S_w^i|]  (Property 3.1, item 3)
16        S^{i+1} ← t[r'_i − |f_i| − sdec..|t|]  (Property 3.1, item 3)
17        Return Contractor(S^{i+1}, S_w^{i+1}, C_{i+1})
18     Else
19        If (|S^i| > |S_w^i|) Then
20           C_{i+1} ← C_i ∪ {c_i},  c_i = (r_i − |f_i| + 1 − sdec, |s| − |f_i| + 1 − sdec)  (Property 3.3)
21        Else
22           C_{i+1} ← C_i  (Property 3.3)
23        End If
24        Return C_{i+1}
25     End If
26  End
```

Our goal in this part is to give a characterization of the language accepted by the Oracle of a word s. To do that, we use the algorithm *Contractor* (*cf.* algorithm 2). Given a word $s \in \Sigma^*$ and its Suffix Oracle $SO(s)$, *Contractor* needs a word w accepted by $SO(s)$ and a suffix t of s chosen such that $\begin{cases} w[1] & = t[1] \\ |t| & \text{maximal} \end{cases}$. The result of *Contractor* is a set $\mathcal{C}$ of contractions such that $w = Word(t, \mathcal{C})$. After a first brief presentation of *Contractor*, we will introduce the notations of the algorithm.

We saw (in the Definition) that $Word(t, C)$, for a set of contractions C, is a concatenation of substrings of t. We can see these sub-words as prefixes of suffixes of t. A jump from one substring to the next one is a contraction. The question is now how to find the correct suffixes and their prefixes. The answer is *Contractor*. This is a recursive algorithm that finds all the contractions used to contract t in w, by searching the suffixes of t which we talk about. The main idea of *Contractor* is to read the words t and w from left to right, and when the one-to-one characters differ, to use a contraction in t to reach a further position in order to allows the reading of the same characters than w.

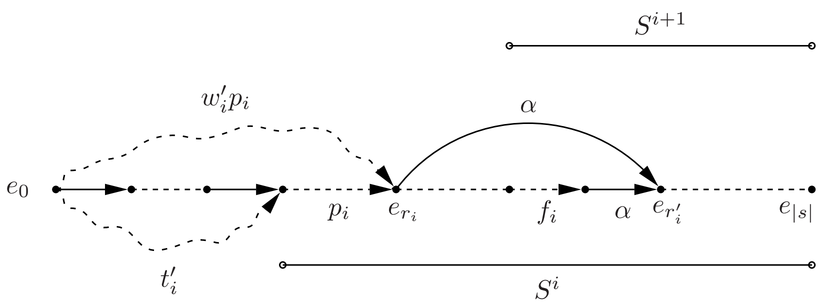

Figure 3: Illustration of a step in the algorithm *Contractor* ($\alpha = S_w^i[|p_i| + 1]$).

The inputs are words S^i and S_w^i ($i \geq 0$), and $\mathcal{C}_i$ a set of contractions. Initially, we have $S^0 = t$, $S_w^0 = w$ and $\mathcal{C}_0 = \emptyset$. We denote by p_i (line 9) the longest prefix of S^i and S_w^i. So, we can write:

$$\begin{cases} S^i & = & p_i S'^i \\ S_w^i & = & p_i S'^i_w \end{cases} \tag{3.1}$$

Let $e_{r_i} = State(p_i)$ (line 10) and $f_i = min(e_{r_i})$ (line 11). Due to Lemma 3.1, we have:

$$p_i = p'_i f_i \ (p'_i \in \Sigma^*) \tag{3.2}$$

About the other variables, $e_{r'_i}$ (line 13) is the state reached by the transition from e_{r_i} and labeled by $\alpha = S^i_w[|p_i|+1] = S'^i_w[1]$, $\mathcal{C}_{i+1}$ is a set of contractions (which has cardinality $i+1$). We need to use the variable $sdec = |s| - |t|$ to translate the indexes of each contraction. Indeed, the positions for a contraction are computed using the indexes of the states (each state e_i is linked to the i^{th} character of s, not to the character $(i - |s| + |t|)$ of t). Thus, a contraction would be correct for s, but not for t Hence, we proceed as for the Definition of $\mathcal{C}^*_{s,t}$, *ie.* we remove $|s| - |t|$.

The figures 3 and 4 illustrates *Contractor*, and are useful to understand the properties below. The following Property 3.1 claims some interesting characteristics of the variables used by *Contractor*.

Property 3.1
For all $i \geq 0$, the following assertions are true:

1. $f_i\alpha \in Pref(p_{i+1})$.
2. $S^i = t[r_i - |p_i| + 1 - sdec..|t|]$.
3. S^{i+1} and S^{i+1}_w are respectively suffixes of S^i and S^i_w; S^i and S^i_w $(i \geq 0)$ are respectively suffixes of t and w.
4. The transition from e_{r_i} to $e_{r'_i}$ and labeled by α always exists.

Proof (Property 3.1)

1. Since $f_i = min(e_{r_i})$, and according to Lemma 3.6, we can write $s[r'_i - |f_i|..r'_i] = t[r'_i - |f_i| - sdec..r'_i - sdec] = f_i\alpha$. So S^{i+1} begins with $f_i\alpha$, and S^{i+1}_w too (line 15).
2. For $i = 0$ (initialization case), $S^0 = t$ and t is the longest suffix of s beginning by $w[1]$. Then we can easily see that if $e_q = State(S^0[1])(q > 0)$, then $t[q - sdec..|t|] = S^0$ and $State(p_0) = q + |p_0| - 1 = e_{r_i}$. Thus $S^0 = s[r_0 - |p_0| + 1 - sdec..|s|]$.
   Now, let us see the recursive case. We have $S^{i+1} = t[r'_i - |f_i| - sdec..|t|]$ (*Contractor*, line 16). Since S^{i+1}_w begins by $f_i\alpha$ (*cf.* item 1), $r_{i+1} = r'_i + |p_{i+1}| - |f_i| - 1$. Finally
   $$S^{i+1} = t[r'_i - |f_i| - sdec..|t|] = t[r_{i+1} - |p_{i+1}| + 1 - sdec..|t|].$$
3. This is obvious for S^i_w because S^{i+1}_w is suffix of S^i_w by construction (line 15) and $S^0_w = w$. Concerning S^i, we have $S^0 = t$ thus the property is true for $i = 0$. Let us suppose that S^i is suffix of t, and show it for $i + 1$. We prove now that S^{i+1} is suffix of S^i. From the preceding point (item 2), we have $S^i = t[r_i - |p_i| + 1 - sdec..|t|]$. In *Contractor*, we have $S^{i+1} = t[r'_i - |f_i| - sdec..|t|]$ (line 16). According to equality 3.2, $r_i - |p_i| = r_i - |p'_i| - |f_i|$. Because $|p'_i| \geq 0$, we obtain $r_i - |f_i| \geq r_i - |p_i|$. Furthermore $r'_i > r_i$. Finally $r'_i - |f_i| > r_i - |p_i|$ and S^{i+1} is a suffix of S^i.
4. According to item 3 in this Property, S^i_w is suffix of w. Then S^i_w is recognized by $O(s)$ (Lemma 3.4). According to equality 3.1 with $S'^i_w[1] = \alpha$, the transition must exists. That implies that $\#_{out}(e_{r_i}) \geq 2$, and then, by Definition of the Canonical Factors, we deduce that $f_i = min(e_{r_i}) \in \mathcal{F}_s$.

□

From equality 3.1 and the above Property 3.1 (item 4), we can write:

$$\begin{cases} t &= t'_iS^i & & (t'_i \in \Sigma^*) \\ w &= w'_iS^i_w &= w'_ip'_iS^{i+1}_w & (w'_i \in \Sigma^*) \end{cases} \tag{3.3}$$

Before giving more explanations about *Contractor*, we need to prove the items of the following property.

Property 3.2
For all $i \geq 0$:

1. $State(w'_ip_i) = State(t'_ip_i) = State(p_i) = e_{r_i}$.
2. c_i is a contraction of t'_iS^i (resp. w'_iS^i) by f_i. The result of this contraction is $t'_ip'_iS^{i+1}$ (resp. $w'_ip'_iS^{i+1} = w'_{i+1}S^{i+1}$).

Proof (Property 3.2)

1. This is obvious for $i = 0$ because $t'_i = w'_i = \epsilon$. Let us suppose the property is true for i, and prove this is true for $i+1$. From Property 3.1 (item 2), we deduce that the word read in $O(s)$ starting from $e_{r_i-|p_i|}$ to $e_{|s|}$ by using only "main" transitions (*ie.* transitions of type $e_j \to e_{j+1}$) is S^i. According to Property 3.1 (item 3) we deduce:
$$S^i = uS^{i+1} \ (u \in \Sigma^*) \tag{3.4}$$
So, there exists the state e_q ($q > r_i - |p_i|$) such that the word read from e_q to $e_{|s|}$ using only "main" transitions is S^{i+1}. In particular, $q = r'_i - |f_i| - 1$. We have $t'_{i+1} = t'_iu$ (*cf.* equality 3.3 and 3.4) and $State(t'_iu) = e_q$. Then, since $f_i = min(e_{r_i})$ and since there exists a transition from e_{r_i} to $e_{r'_i}$ labeled by α (*cf.* Property 3.1, item 4), we have $State(t'_{i+1}f_i\alpha) = State(t'_iuf_i\alpha) = State(f_i\alpha) = e_{r'_i}$. Furthermore $p_{i+1} = f_i\alpha v$ ($v \in \Sigma^*$). So we can deduce that $State(t'_{i+1}f_i\alpha v) = State(t'_{i+1}p_{i+1}) = State(p_{i+1})$.
2. From the equalities 3.1, 3.2 and 3.3, we deduce that:
$$t = t'_iS^i = t'_ip'_if_iS'^i \tag{3.5}$$
Since $S^{i+1} \in Suff(S^i)$, we have $S^i = uS^{i+1}$ ($u \in \Sigma^+$). Hence, we deduce from equality 3.5 that $t'_ip'_if_iS'^i = t'_iuS^{i+1}$. According to the Property 3.1 (item 1), we have $t'_ip'_if_iS'^i = t'_iuf_i\alpha u'$ ($u' \in \Sigma^*$). Because we have $State(t'_ip'_if_i) = State(f_i)$ and $|u| > |p'_i|$ ($S'^i[1] \neq \alpha$), we can contract t'_iS^i by f_i; the result is:
$$t'_ip'_if_i\alpha u' = t'_ip'_iS^{i+1} \tag{3.6}$$
Since $State(w'_ip_i) = State(t'_ip_i)$, we can deduce that $w'_iS^i = w'_ip_iS'^i$ is contracted by f_i in $w'_ip'_iS^{i+1}$. According to equality 3.3, we deduce that $w'_{i+1} = w'_ip'_i$. Then $w'_ip'_iS^{i+1} = w'_{i+1}S^{i+1}$.

□

The Property 3.2 shows us that c_i (a contraction of $t'_i S^i$ by f_i) is a contraction for t and, more interesting, for $w'_i S^i$. Before concluding about these contractions, we need to examine the termination of *Contractor* and its final case. For all $i \geq 0$, we have either $|S^i_w| > |S^{i+1}_w|$, nor $|S^i_w| = |S^{i+1}_w|$ and $|p_{i+1}| > |p_i|$ (if $f_i = p_i$). Since $p_i > 0$, we deduce that we finally obtain $p_j = S^j_w$ ($j > i$). The following property concerns the final cases of *Contractor*.

Property 3.3
Let $j \geq 0$ be such that $p_j = S^j_w$. If $|S^j_w| \neq |S^j|$, then t needs a last contraction. Else $\mathcal{C}_j$ is the final set.

Proof (Property 3.3)
The word obtained up to now with the contraction of $\mathcal{C}_j$ is $w'_j p_j S'^j$ (*cf.* Property 3.2, item 2). If $S^j_w = S^j$, then $S'^j = \epsilon$ and $\mathcal{C}_j$ is complete (line 20). According to Property 3.2 (item 1), we have $State(w'_j p_j) = e_{r_j}$. Thus $min(e_{r_j}) \in Suff(w)$ (Lemma 3.1) and $min(e_{r_j}) \in Suff(t)$ (by Definition of the final state in a Suffix Oracle). Then a last contraction completes the set of contractions (line 22). □

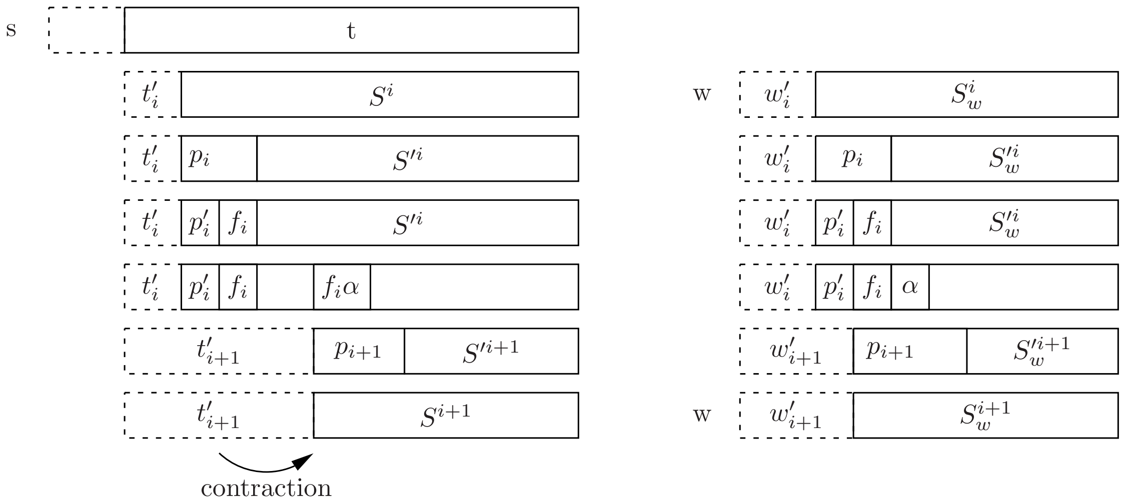

Figure 4: Visualization of *Contractor* on S^i and S^i_w.

Now, let us see how a step of *Contractor* works. We consider the i^{th} call of *Contractor*, whose inputs are $S^i = p_i S'^i$, $S^i_w = p_i S'^i_w$ and $\mathcal{C}_i$. The contractions already used to contract the beginning of t (*ie.* t'_i) into the beginning of w (*ie.* w'_i) are in $\mathcal{C}_i$. At this point we consider the longest common prefix (denoted by p_i) of S^i and S^i_w (p_i is both a factor of t and w, Property 3.1). The algorithm has two cases. If $|p_i| = |S^i_w|$, we are in a final case we described above. Else, the prefix p_i is not S^i_w, and then we need at least one other contraction until $|p_i| = |S^i_w|$. Thus we search for another suffix S^{i+1} of t with which we can continue to contract. From Property 3.2, we have the contraction is the right one, and we continue with the suffix S^{i+1}. When we reach the end of the process (*ie.* the end of w), we return the last up-to-date set $\mathcal{C}_{i+1}$ and $w = Word(t, \mathcal{C}_{i+1})$.

We can notice that:

1. $\mathcal{C}$ is not always minimal. The algorithm could be modified but would become more difficult to understand. However, the minimality is not an objective here.

2. $\mathcal{C}$ is coherent. Let (a,b) and (c,d) be two contractions added successively to $\mathcal{C}$. We have $a < b$ and $c < d$ because $r'_i > r_i$ and $|s| > r_i$ (*cf.* lines 14 and 20). Next, either $e_{r_{i+1}} = State(p_{i+1}) = e_{r'_i}$ and so $b = c$, or $e_{r_{i+1}} > e_{r'_i}$ (because we can have $p_{i+1} = f_i\alpha v$ ($\alpha = S^i_w[|p_i|+1]$) where $v \neq \epsilon$, and thus $b < c$.

Lemma 3.7 *Given a word $s \in \Sigma^*$, its Suffix Oracle, a word $w \in \Sigma^*$ accepted by $SO(s)$, and t being the longest suffix of s such that $w[1] = t[1]$, then* Contractor$(t, w, \emptyset)$ *returns a set $\mathcal{C}$ such that $w = Word(t, C)$.*

Proof (Lemma 3.7)
Let $j \geq 0$ such that $S^{j+1}_w = p_{j+1}$. Then, according to Property 3.2, we deduce that $\mathcal{C}_{j+1}$ is a coherent set of contractions of t. Then, we have:

$$Word(t, \mathcal{C}_{j+1}) = w'_{j+1}S^{j+1} = w'_{j+1}S^{j+1}_w u = w'_{j+1}p_{j+1}u \quad (u \in \Sigma^*)$$

because p_{j+1} is prefix of S^{j+1}. If $u = \epsilon$, we have $Word(t, \mathcal{C}_{j+1}) = w$ (equality 3.3). Else (*cf.* Property 3.3) a ultimate contraction c_{j+1} contracts $w'_{j+1}S^{j+1}_w u$ by f_{j+1} in $w'_{j+1}S^{j+1}_w = w = Word(t, \mathcal{C}_{j+1} \cup \{c_{j+1}\})$.
Finally *Contractor* provide a set $\mathcal{C}$ such that $w = Word(t, \mathcal{C})$. □

The following two theorems are the main purpose of this paper.

Theorem 3.1 *Exactly all the suffixes of the words from $\mathcal{E}(s)$ are recognized by the Suffix Oracle of s .*

Proof (Theorem 3.1)
'$\Rightarrow$': *Each suffix of a word from $\mathcal{E}(s)$ is recognized by the Suffix Oracle of s.*
According to Lemma 3.4, if w is accepted by $SO(s)$, then each suffix of w is also accepted by $SO(s)$, so we only need to prove that each word from $\mathcal{E}(s)$ is accepted by $SO(s)$.
Let $\mathcal{C} \in \mathcal{C}^*_s$ be a set of contractions applicable to s. Let us build $w = Word(s, \mathcal{C})$, and show that w is accepted by $SO(s)$. Let $\mathcal{C}_i$ be the set of the first i contractions of $\mathcal{C}$ (chosen without loss of generality by ascending order over the positions), (x_j, y_j) be the j^{th} contraction, and $f_j \in \mathcal{F}_s$ the Canonical Factor used by (x_j, y_j) $(1 \leq j \leq i)$. We note $w_j = Word(s, \mathcal{C}_j)$. The property (P) to check is that w_i is accepted by $SO(s)$. Because $w_0 = s$, the property (P) is true for $i = 0$. Let us suppose that it is true for i, and show that (P) is true for $i+1$. We have:

$$\left\{\begin{array}{lcl} w_i & = & s[1..x_1-1]\,s[y_1..x_2-1]\ldots s[y_i..|s|] \\ s[y_i..y_i+|f_i|-1] & = & f_i \end{array}\right.$$

By Definition of the Canonical Factors, f_{i+1} does not occur in s before the position x_{i+1} ($x_{i+1} > y_i$). Thus we can write, in particular, w_i and w_{i+1} as:

$$\left\{\begin{array}{lcl} w_i & = & v'f_{i+1}u \\ w_{i+1} & = & v'f_{i+1}u' \end{array}\right. \text{with} \left\{\begin{array}{lcll} v' & = & s[1..x_1-1]s[y_1..x_2-1]\ldots s[y_i..x_{i+1}-1] & \\ f_{i+1}u & = & u''f_{i+1}u' & (u'' \in \Sigma^+) \end{array}\right.$$

Because the contraction concerns (x_{i+1}, y_{i+1}), then we also have $|s|-|f_{i+1}u|+1 = x_{i+1}$ and $|s|-|f_{i+1}u'|+1 = y_{i+1}$. This is true because the contractions are in ascending order, so the word s is not yet modified after the positions x_{i+1} and y_{i+1} (hence $f_{i+1}u$ and $f_{i+1}u'$ are suffixes of s). Let q be the state of $SO(s)$ such that $q = State(f_{i+1})$. According to the Lemma 3.5:

$$State(v'f_{i+1}) = q \tag{3.7}$$

Furthermore, $f_{i+1}u'$ is a suffix of s, so it is necessary recognized by $SO(s)$. This requires to pass through the state q when the word $f_{i+1}u'$ is read in $SO(s)$. Thus, starting from q, we can read u', and reach a final state. So, according to equality 3.7, $w_{i+1} = v'f_{i+1}u'$ is accepted by $SO(s)$. To conclude, we just showed that w_i is recognized by $SO(s)$, for all $i \leq |\mathcal{C}|$. Finally, Lemma 3.4 allows to conclude that each suffix of a word of $\mathcal{E}(s)$ is recognized by $SO(s)$.

'$\Leftarrow$': *Each word recognized by the Suffix Oracle of s is suffix of a word from $\mathcal{E}(s)$.*
Let w be a word accepted by the Suffix Oracle of s, and t be the longest suffix of s ($s = s't$) such that $w[1] = t[1]$. Then there exists a set $\mathcal{C}$ of contractions such that $w = Word(t, \mathcal{C})$ (Lemma 3.7). Since $\mathcal{C} \subseteq \mathcal{C}^*_{s,t}$, there exists a set $\mathcal{C}' \subseteq \mathcal{C}^*_s$, obtained by translating the indexes of $\mathcal{C}$ with $sdec$, such that $s'w = Word(s't, \mathcal{C}')$. Because $s'w \in \mathcal{E}(s)$, we can conclude that each word accepted by $SO(s)$ is a suffix of a word from $\mathcal{E}(s)$. □

On the basis of this previous result, we can give a similar theorem, which is available for the Factor Oracle instead of being available for the Suffix Oracle.

Theorem 3.2 *Exactly all the factors of the words from $\mathcal{E}(s)$ are recognized by the Factor Oracle of s .*

Proof (Theorem 3.2)
'$\Rightarrow$': *Each factor m of a word from $\mathcal{E}(s)$ is recognized by the Factor Oracle of s.*
Let $SO(s)$ be the Suffix Oracle of s, and $u \in \mathcal{E}(s)$ be such that m is prefix of a suffix of u, denoted by mv ($v \in \Sigma^*$). Then mv is accepted by $SO(s)$ (*cf.* Theorem 3.1), thus there exists a single path $(e_0 \rightarrow e_{x_1} \rightarrow \ldots \rightarrow e_{x_{|m|+|v|}})$ in $SO(s)$ that recognizes mv. Therefore, there exists a path $(e_0 \rightarrow e_{x_1} \rightarrow \ldots \rightarrow e_{x_{|m|}})$ (with $e_{x_{|m|}}$ final) that recognizes m.

'$\Leftarrow$': *Each word m recognized by the Factor Oracle of s is factor of a word from $\mathcal{E}(s)$.*
Let $SO(s)$ be the Suffix Oracle of s. If m is recognized by $SO(s)$ then m is a suffix of a word of $\mathcal{E}(s)$ (*cf.* Theorem 3.1). Let us suppose that m is not recognized by $SO(s)$, then m is recognized by $FO(s)$ in the state $e_{x_{|m|}}$ (not final in $SO(s)$). By construction, $e_{x_{|m|}} \in \{e_k | 0 \leq k \leq |s|\}$, the set of the states of $FO(s)$, with $(e_0 \rightarrow e_1 \rightarrow \ldots \rightarrow e_{|s|})$ the path that accepts the word s itself (with $e_{|s|}$, among others, final in $SO(s)$). Thus, there exits a path from $e_{x_{|m|}}$ to $e_{|s|}$ in $SO(s)$. So, m is prefix of a word recognized by $SO(s)$. That implies that m is prefix of a suffix of some $u \in \mathcal{E}(s)$. Therefore, m is a factor of a word of $\mathcal{E}(s)$. □

4 Properties upon Oracles & Future Works

In the conclusion of their article, Cleophas & *al.* [11] show that the Oracle is not minimal in number of transitions among the set of homogeneous automata.

Furthermore, if we consider the set of homogeneous automata recognizing at least all the factors (resp. suffixes) of s, having the same number of states and at most the same number of transitions than the Factor (resp. Suffix) Oracle, we show that the Oracle is not minimal on the number of accepted words. We can see that the Oracle of *axttyabcdeatzattwu* (*cf.* figure 5) has 35 transitions. The Factor Oracle accepts 247 words and the Suffix Oracle accepts 39 words, though there exists another homogeneous automaton (*cf.* figure 6) recognizing at least all the factors (resp. suffixes) of *axttyabcdeatzattwu*, and having only 34 transitions. The "Factor" version of this automaton recognizes only 236 words and its "Suffix" version accepts only 30 words. This example shows that the Oracle is not minimal in number of accepted words among the set of homogeneous automata having the same number of states and *less* transitions.

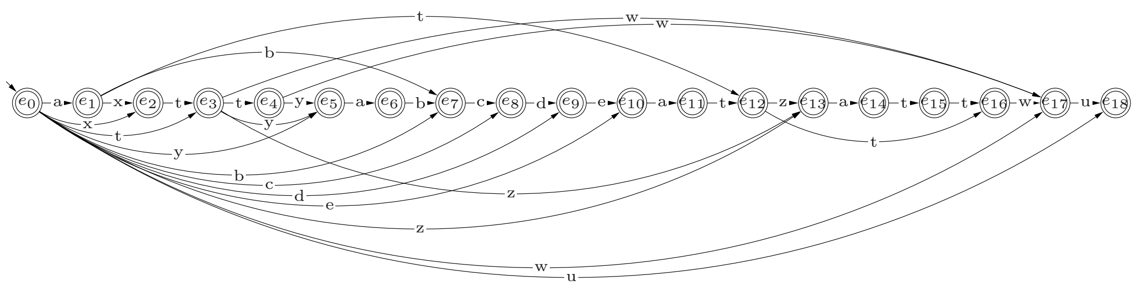

Figure 5: Factor Oracle of the word *axttyabcdeatzattwu*.

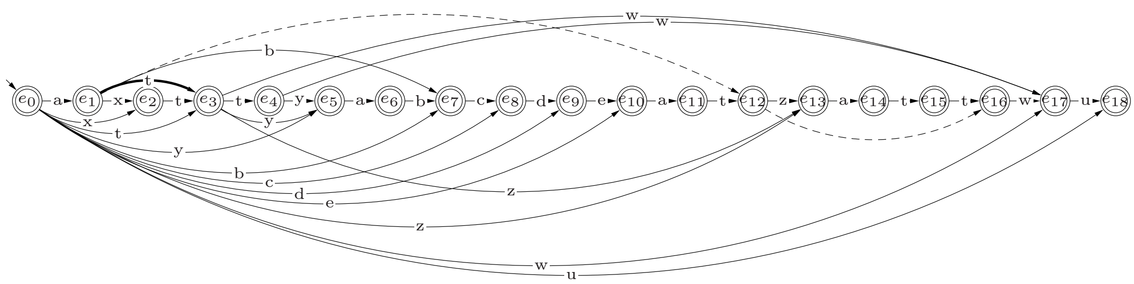

Figure 6: This automaton (considering only the continuous lines) accepts at least all the factors of the word *axttyabcdeatzattwu*. The bold transition is the only one which is not present in the Factor Oracle of this word (*cf.* figure 5) though the two dotted ones are present in the Factor Oracle, but not in this automaton.

We observe that, in some cases, the number of words accepted by Oracles does not allow to give confidence to this structure when it is used for detect factors or suffixes of a word. Because, even if the number of false positive can sometimes be null (*eg.* *aaaaaa*...), it can also be exponential. Indeed, we can build a word s such that each subset of $\mathcal{C}_s^*$ is coherent and minimal. For example: $s = aabbccddee\ldots$. The set $\mathcal{C}_s^*$ of contractions which are available on such a word is $\{(1,2),(3,4),(5,6),\ldots,(|s|-1,|s|)\}$. If we consider any (non-empty) subset $\mathcal{C} \subseteq (\mathcal{C}_s^* \setminus \{(1,2)\})$ of contractions, it is easy to notice that $Word(s,\mathcal{C}) \notin Fact(s)$. Besides, all the words obtained from such subsets are pairwise different.

The number of these subsets is:

$$\sum_{i=1}^{|\mathcal{C}_s^*|-1} \binom{|\mathcal{C}_s^*|-1}{i} = \sum_{i=1}^{\frac{|s|}{2}-1} \binom{\frac{|s|}{2}-1}{i} = 2^{\frac{|s|}{2}-1} - 1$$

So the number of words that will be accepted by the Oracles but are not factor/suffix of s is $\mathcal{O}\big(2^{|s|}\big)$.

In order to better benefit from this structure, it has to be improved, or to be slightly modified. However, it could be useful for future works to improve the knowledges about the Oracle structure. Effectively, it could be interesting to have either an empirical nor a statistical estimation of the accuracy (time and quality of the results) of the Oracle when substituted to Tries or Suffix Trees in algorithms.

References

[1] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: computer science and computational biology.* Cambridge University Press, 1997.

[2] Anselm Blumer, Janet Blumer, David Haussler, Andrzej Ehrenfeucht, M. T. Chen, and Joel Seiferas. The smallest automaton recognizing the subwords of a text. *Theorical Computer Science*, 40(1):31–55, 1985.

[3] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[4] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Oracle des facteurs, Oracle des Suffixes. Technical Report 99-08, Institut Gaspard-Monge, Université de Marne-la-Vallée, 1999.

[5] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Factor Oracle: A New Structure for Pattern Matching. In *Conference on Current Trends in Theory and Practice of Informatics*, pages 295–310, 1999.

[6] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Efficient Experimental String Matching by Weak Factor Recognition. In *Proceedings of the 12th conference on Combinatorial Pattern Matching*, volume 2089 of *Lecture Notes in Computer Science*, pages 51–72. Springer-Verlag, 2001.

[7] Arnaud Lefebvre and Thierry Lecroq. Computing repeated factors with a factor oracle. In L. Brankovic and J. Ryan, editors, *Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms*, pages 145–158, Hunter Valley, Australia, 2000.

[8] Arnaud Lefebvre and Thierry Lecroq. A heuristic for computing repeats with a factor oracle: application to biological sequences. *International Journal of Computer Mathematics*, 79(12):1303–1315, 2002.

[9] Arnaud Lefebvre, Thierry Lecroq, Hélène Dauchel, and Joël Alexandre. FORRepeats: detects repeats on entire chromosomes and between genomes. *Bioinformatics*, 19(3):319–326, 2003.

[10] Arnaud Lefebvre and Thierry Lecroq. Compror: on-line lossless data compression with a factor oracle. *Information Processing Letters*, 83(1):1–6, 2002.

[11] Loek Cleophas, Gerard Zwaan, and Bruce Watson. Constructing Factor Oracles. In *Proceedings of the 3rd Prague Stringology Conference*, 2003.