An Introduction to Suffix Automaton (Survey Report)

Shimi Zhang

Nov. 2, 2013

Chapter 1

Prerequisites

Some definitions:

1. ε = an empty word.
2. $\mathcal{A}$ = a finite set called the *alphabet*.
3. $\mathcal{A}^*$ = the set of words over $\mathcal{A}$ and ε.
4. $\mathcal{A}^+ = \mathcal{A}^* - \{\varepsilon\}$
5. $|w|$ = the length of word w.
6. Suff(y) = a set of all suffixes of word y.
7. Fact(y) = a set of all factors (substring) of word y.

The suffix automaton is denoted by $\mathfrak{A}(y)$. Its states are classes of the (right) syntactic equivalence associated with Suff(y), that is, are the sets of factors of y having the same right context within y. This states are in one-to-one correspondence with the (right) contexts of the factors of y in y itself. Formally, we call the (right) context of a word u is $\mathcal{R}(u) = u^{-1}\text{Suff}(y)$.

We denote by $\equiv_y$ the syntactic congruence which is defined, for $u, v \in \mathcal{A}^*$, by

$$u \equiv_y v$$

if and only if

$$\mathcal{R}_y(u) = \mathcal{R}_y(v).$$

One can also identify the states of $\mathfrak{A}(y)$ to sets of indices on y which are end positions of occurrences of equivalent factors. For any factor u of y, one denotes by

$$\textit{end-pos}(u) = \min\{|wu| \mid wu \text{ is a prefix of } y\} - 1,$$

the right position of the first occurrence of u in y. Note that $\textit{end-pos}(\varepsilon) = -1$.

Lemma 1.1. *Let $u, v \in$ Fact(y) with $|u| \leq |v|$. Then,*

$$u \textit{ is a suffix of } v \textit{ implies } \mathcal{R}_y(v) \subseteq \mathcal{R}_y(u)$$

and

$$\mathcal{R}_y(u) = \mathcal{R}_y(v) \textit{ implies both } \textit{end-pos}(u) = \textit{end-pos}(v) \textit{ and } u \textit{ is a suffix of } v.$$

Proof. Let us suppose that u is a suffix of v, Let $z \in \mathcal{R}_y(v)$. By definition, vz is a suffix of y and, since u is a suffix of v, the word uz is also a suffix of y. Thus, $z \in \mathcal{R}_y(u)$, which proves the first implication.
Let us now suppose $\mathcal{R}_y(u) = \mathcal{R}_y(v)$. let w, z be such that $y = w \cdot z$ with $|w| = \textit{end-pos}(u) + 1$. By definition of *end-pos*, u is a suffix of w. Therefore, z is the longest word in $\mathcal{R}_y(v)$, which yields $|w| = \textit{end-pos}(v) + 1$. The words u and v are thus both suffixes of w, and as u is shorter than v one obtains that u is a suffix of v. This finishes the proof of the second implication and the whole proof. □

Lemma 1.2. *Let $u, v, w \in \textit{Fact}(y)$. If u is a suffix of v, v is a suffix of w and $u \equiv_y w$, then $u \equiv_y v \equiv_y w$,.*

Proof. By Lemma 1.1, the assumption implies

$$\mathcal{R}_y(w) \subseteq \mathcal{R}_y(v) \subseteq \mathcal{R}_y(u).$$

Then, the equivalence $u \equiv_y w$ which means $\mathcal{R}_y(u) = \mathcal{R}_y(w)$ gives the conclusion. □

Corollary 1.3. *Let $u, v \in \mathcal{A}^*$. Then, the contexts of u and v are comparable for inclusion or are disjoint, that is, at least one of the three following conditions is satisfied:*

1. $\mathcal{R}_y(u) \subseteq \mathcal{R}_y(v)$,
2. $\mathcal{R}_y(v) \subseteq \mathcal{R}_y(u)$,
3. $\mathcal{R}_y(u) \cap \mathcal{R}_y(v) = \emptyset$.

Proof. One proves the property by showing that the condition

$$\mathcal{R}_y(u) \cup \mathcal{R}_y(v) \neq \emptyset$$

implies

$$\mathcal{R}_y(u) \subseteq \mathcal{R}_y(v) \text{ or } \mathcal{R}_y(v) \subseteq \mathcal{R}_y(u).$$

Let $z \in \mathcal{R}_y(u) \cup \mathcal{R}_y(v)$. Then, uz, vz are suffixes of y, and u, v are suffixes of yz^{-1}. Consequently, among u and v one is a suffix of the other. One obtains finally the conclusion by Lemma 1.1. □

Suffix function
On the set $\text{Fact}(y)$ we consider the function s_y called the *suffix function* of y. It is defined, for all $v \in \text{Fact}(y) \setminus \{\varepsilon\}$, by

$$s_y(v) = \text{ longest suffix } u \text{ of } v \text{ such that } u \not\equiv_y v.$$

After Lemma 1.1, one deduces the equivalent definition:

$$s_y(v) = \text{longest suffix } u \text{ of } v \text{ such that } \mathcal{R}_y(v) \subset \mathcal{R}_y(u).$$

Note that, by definition, $s_y(v)$ is a proper suffix of v, i.e., $|s_y(v)| < |v|$. The following lemma shows that the suffix function s_y induces a failure function on state of $\mathfrak{A}(y)$.

Lemma 1.4. *Let $u, v \in \textit{Fact}(y) \setminus \{\varepsilon\}$. If $u \equiv_y v$, then $s_y(u) = s_y(v)$.*

Proof. By Lemma 1.1 one can suppose without loss of generality that $|u| \leq |v|$, so u is a suffix of v. The word u cannot be a suffix of $s_y(v)$ because Lemma 1.2 would imply $s_y(v) \equiv_y v$, which contradicts the definition of $s_y(v)$. Consequently, $s_y(v)$ is a suffix of u. Since, by definition, $s_y(v)$ is the longest suffix of v which is not equivalent to itself, it is also $s_y(u)$. Thus, $s_y(u) = s_y(v)$. □

Lemma 1.5. *Let $y \in \mathcal{A}^+$. The word $s_y(y)$ is the longest suffix of y that appears at least twice in y itself.*

Proof. The context $\mathcal{R}_y(y)$ is $\{\varepsilon\}$. As y and $s_y(y)$ are not equivalent, $\mathcal{R}_y(s_y(y))$ contains some nonempty word z. Then $s_y(y)z$ and $s_y(y)$ are suffixes of y, which shows that $s_y(y)$ appears twices at least in y. Any suffix w of y, longer than $s_y(y)$, is equivalent to y by definition of $s_y(y)$. It thus satisfies $\mathcal{R}_y(w) = \mathcal{R}_y(y) = \{\varepsilon\}$. Which shows that w appears only once in y and finishes the proof. □

Lemma 1.6. *Let $u \in Fact(y)\setminus\{\varepsilon\}$. Then, any word equivalent to $s_y(u)$ is a suffix of $s_y(u)$.*

Proof. Let $w = s_y(u)$ and $v \equiv_y w$. We show that v is a suffix of w. The word w is a proper suffix of u. If the conclusion of the statement is false, according to Lemma 1.1 one obtains that w is a proper suffix of v. Then let $z \in \mathcal{R}_y(u)$. As w is a suffix of u equivalent to v, we have $z \in \mathcal{R}_y(w) = \mathcal{R}_y(v)$. Then, u and v are both suffixes of yz^{-1}, which implies that one is a suffix of the other. But this contradicts either the definition of $w = s_y(u)$ or the conclusion of Lemma 1.2, and proves that v is a suffix of $w = s_y(u)$. □

The on-line construction of suffix automaton relies on the relationship between $\equiv_{wa}$ and $\equiv_w$ which we examine here. By doing this, we consider that the generic word y is equal to wa for some letter a. The properties detailed below are also used to derie precise bounds on the size of the automaton.

Lemma 1.7. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. The congruence $\equiv_{wa}$ is a refinement of $\equiv_w$, that is for all words $u, v \in \mathcal{A}^*$, $u \equiv_{wa} v$ implies $u \equiv_w v$.*

Proof. Let us assume that $u \equiv_{wa} v$, that is, $\mathcal{R}_{wa}(u) = \mathcal{R}_{wa}(v)$, and show that $u \equiv_w v$, that is, $\mathcal{R}_w(u) = \mathcal{R}_w(v)$. We show $\mathcal{R}_w(u) \subseteq \mathcal{R}_w(v)$ only because the opposite inclusion results by symmetry.
If $\mathcal{R}_w(u) = \emptyset$ the inclusion is clear. If not, let $z \in \mathcal{R}_w(u)$. Then uz is a suffix of w, which implies that uza is a suffix of wa. The assumption gives that vza is a suffix of wa, and thus vz is a suffix of w, or $z \in \mathcal{R}_w(v)$, which finishes the proof. □

Lemma 1.8. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. Let z be the longest suffix of wa that appears in w. If u is a suffix of wa strictly longer than z, then the equivalence $u \equiv_{wa} wa$ holds.*

Proof. It is a direct consequence of Lemma 1.5 because z occurs at least twice in wa. □

Before going to the main theorem we state an additional relation concerning right contexts.

Lemma 1.9. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. Then, for each word $u \in \mathcal{A}^*$,*

$$\mathcal{R}_{wa}(u) = \begin{cases} \{\varepsilon\} \cup \mathcal{R}_w(u)a & \text{if } u \text{ is a suffix of } wa, \\ \mathcal{R}_w(u)a & \text{otherwise.} \end{cases}$$

Proof. First notice that $\varepsilon \in \mathcal{R}_{wa}(u)$ is equivalent to: u is a suffix of wa. It is thus enough to show $\mathcal{R}_{wa}(u)\setminus\{\varepsilon\} = \mathcal{R}_w(u)a$.
Let z be a nonempty word of $\mathcal{R}_{wa}(u)$. We get that uz is a suffix of wa. The word uz can be written $uz'a$ with uz' a suffix of w. Consequently, $z' \in \mathcal{R}_w(u)$ and thus $z \in \mathcal{R}_w(u)a$.
Conversely, let z be a (nonempty) word in $\mathcal{R}_w(u)a$. It can be written $z'a$ for $z' \in \mathcal{R}_w(u)$. Thus, uz' is a suffix of w, which implies that $uz = uz'a$ is a suffix of wa, that is, $z \in \mathcal{R}_{wa}(u)$. This proves the converse statement and ends the proof. □

Theorem 1.10. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. Let z be the longest suffix of wa that appears in w. Let z' be the longest factor of w for which $z' \equiv_w z$. Then for each $u, v \in Fact(w)$,*

$$u \equiv_w v \text{ and } u \not\equiv_w z \text{ imply } u \equiv_{wa} v.$$

Moreover, for each word u such as $u \equiv_w z$,

$$u \equiv_{wa} \begin{cases} z & if |u| \leq |z|, \\ z' & otherwise. \end{cases}$$

Proof. Let $u, v \in \mathrm{Fact}(w)$ be such that $u \equiv_w v$. By definition of the equivalence we get $\mathcal{R}_w(u) = \mathcal{R}_w(v)$. We suppose first that $u \not\equiv_w z$ and show that $\mathcal{R}_{wa}(u) = \mathcal{R}_{wa}(v)$, which is $u \equiv_{wa} v$.
According to Lemma 1.9, we have just to show that u is a suffix of wa if and only if v is a suffix of wa. Indeed, it is enough to show that if u is a suffix of wa then v is a suffix of wa since the opposite implication results by symmetry.
So let us suppose that u is a suffix of wa. We deduce from the fact that u is a factor of w and the definition of z that u is a suffix of z. We can thus consider the greatest integer $j \geq 0$ for which $|u| \leq |s_w^j(z)|$. Let us note that $s_w^j(z)$ is a suffix of wa (like z is), and the Lemma 1.2 ensures that $u \equiv_w s_w^j(z)$. From which we get $v \equiv_w s_w^j(z)$ by transitivity.
Since $u \not\equiv_w z$, we have $j > 0$. Lemma 1.6 implies that v is a suffix of $s_w^j(z)$, and thus also of wa as wished. This proves the first part of the statement.
Let us consider now a word u such as $u \equiv_w z$.
When $|u| \leq |z|$, to show $u \equiv_w z$ by using the above argument, we have only to check that u is a suffix of wa because z is a suffix of wa. This, in fact, is a simple consequence of Lemma 1.1.
Let us suppose $|u| > |z|$. The existence of such a word u implies $z' \neq z$ and $|z'| > |z|$ (z is a proper suffix of z'). Consequently, by the definition of z, u and z' are not suffixes of wa. Using the above argument again, this proves $u \equiv_{wa} z'$ and finishes the proof. □

Corollary 1.11. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. Let z be the longest suffix of wa that appears in w. Let z' be the longest word such as $z' \equiv_w z$. Let us suppose $z' = z$. Then, for each $u, v \in Fact(w)$,*

$$u \equiv_w v \text{ implies } u \equiv_{wa} v.$$

Proof. Let $u, v \in \mathrm{Fact}(w)$ be such that $u \equiv_w v$. We prove the equivalence $u \equiv_{wa} v$. The conclusion comes directly from Theorem 1.10 if $u \not\equiv_w z$. Else, $u \equiv_{wa} z$; by the assumption made on z and Lemma 3.1, we get $|u| \leq |z|$. Finally, Theorem 1.10 gives the same conclusion. □

Corollary 1.12. *Let $w \in \mathcal{A}^*$ and $a \in \mathcal{A}$. If the letter a does not appear in w, then, for each $u, v \in Fact(w)$,*

$$u \equiv_w v \text{ implies } u \equiv_{wa} v.$$

Proof. Since a does not appear in w the word z of Corollary 1.11 is the empty word. It is of course the longest of its class, which makes it possible to apply Corollary 1.11 and gives the same conclusion. □

Chapter 2

Suffix automaton

The *suffix automaton* of a word y is the minimal automaton that accepts the set of suffixes of y. It is denoted by $\mathfrak{A}(y)$. The most surprising property of the automaton is that its size is linear in the length of y although the number of factor of y can be quadratic. The construction of the automaton also takes a linear time on a fixed alphabet.

2.1 Size of suffix automata

The size of an automaton is expressed both by the number of its states and the number of its edges. We show that $\mathfrak{A}(y)$ has less than $2|y|$ states and less than $3|y|$ edges, for a total size $O(|y|)$. This result is based on Theorem 1.10.

Proposition 2.1. *Let $y \in \mathcal{A}^*$ be a word of length n and let $st(y)$ be the number of states of $\mathfrak{y}$. For $n = 0$, we have $st(y) = 1$; for $n = 1$, we have $st(y) = 2$; for $n > 1$ finally, we have*

$$n + 1 \leq st(y) \leq 2n - 1,$$

and the upper bound is reached if and only if y is of the form ab^{n-1}, for two distinct letters a, b.

Proof. The equalities concerning short words can be checked direclty including $st(y) = 3$ when $|y| = 2$. Let us suppose $n > 2$. The minimal number of states of $\mathfrak{A}$ is obviously $n + 1$ (otherwise the path labelled by y would contain a cycle yielding an infinite number of words recognized by the automaton); the minimal value is reached with $y = a^n (a \in \mathcal{A})$.
Let us show the upper bound. By Theorem 1.10, each letter $y[i], 2 \leq i \leq n-1$, increases by at most two the number of states of $\mathfrak{A}(y[0..i-1])$. As the number of states of $\mathfrak{A}(y[0]y[1])$ is 3, it follows that

$$\begin{aligned} st(y) \leq & \quad 3 + 2(n-2) \\ = & \quad 2n - 1, \end{aligned}$$

as announced.
The construction of a word of length n whose suffix automaton has $2n - 1$ states is still a simple application of Theorem 1.10 by noting that each letter $y[2], y[3], ..., y[n-1]$ must effectively lead to the creation of two states during the construction. Notice that after the choice of the first two letters, which must be different, there is no choice for the other letters. This produces the only possible form given in the statement. □

Lemma 2.2. *Let $y \in \mathcal{A}^+$ and let $ed(y)$ be the number of edges of $\mathfrak{A}(y)$. Then,*

$$ed(y) \leq st(y) + |y| - 2$$

.

Proof. Let us call q_0 the initial state of $\mathfrak{A}(y)$, and consider the spanning tree of longest paths starting at q_0 in $\mathfrak{A}(y)$. The tree contains $st(y) - 1$ edges of $\mathfrak{A}(y)$ because it arrives exactly at one edge on each state except on the initial state.
With each other edge(p, a, q) we associate the suffix uav of y defined as follows: u is the label of the path starting at q_0 and ending at p; v is the label of the longest path from q arriving on a terminal state. Doing so, we get an injection from the set of concerned edges to the set Suff(y). The suffixes y and ε are not concerned because they are labels of paths in the spanning tree. This shows that there is at most Card(Suff$(y)\setminus\{y, \varepsilon\}) = |y| - 1$ additional edges. Summing up the numbers of edges of the two types, we get at maxium of $st(y) + |y| - 2$ edges in $\mathfrak{A}(y)$. □

2.2 Suffix links and suffix paths

Theorem 1.10 and its two consecutive corollaries provide the frame of the on-line construction of the suffix automaton $\mathfrak{A}(y)$. The algorithm controls the conditions which appear in these statements by means of function defined on the states of the automaton, the suffix link function, and of a classification of the edges into solid and non-solid edges. We define these two concepts hereafter. Let p be a state of $\mathfrak{A}(y)$, different from the inital state. State p is a class of factors of y that are equivalent with respect to equivalence $\equiv_y$. Let u be any word in the class ($u \neq \varepsilon$ because p is not the initial state). We define the suffix link of p, denoted by $f_y(p)$, as the congruence class of $s_y(u)$. The function f_y is called the suffix link function of the automaton. According to Lemma 1.4 the value of $s_y(u)$ is independent of the word u chosen in the class of p, which makes the definition coherent. The suffix link function is also called a failure function and used with this meaning in pattern matching machine. For a state p of $\mathfrak{A}(y)$, we denote by $lg_y(p)$ the maxmimum length of words u in the congruence class of p. It is also the length of the longest path starting from the initial state and ending at p. The longest paths starting at the initial state form a spanning tree for $\mathfrak{A}(y)$ (a consequence of Lemma 1.1). Edges which belong to this tree are qualified as *solid*. In an equivalent way,

$$edge(p, a, q) \text{ is solid}$$

if and only if

$$lg_y(q) = lg_y(p) + 1.$$

This notion is used in the construction of the automaton. Suffix links induce by iteration what we call suffix paths in $\mathfrak{A}(y)$. One can note that

$$q = f_y(p) implies lg_y(q) < lg_y(p).$$

So, the sequence

$$\langle p, f_y(p), f_y^2(p), ...\rangle$$

is finite and ends at the initial state (which does not have a suffix link). It is called the suffix path of p in $\mathfrak{A}(y)$, and is denoted by $SP(p)$. Let *last* be the state of $\mathfrak{A}(y)$ that is the class of word y itself. This state is characterized by the fact that it is not the origin of any edge. The suffix path of *last* plays an important part in the on-line construction. It is used to effectively test conditions of Theorem 1.10 and its corollaries.
We denote by δ the transition function of $\mathfrak{A}(y)$.

Proposition 2.3. *Let $u \in Fact(y)\setminus\{\varepsilon\}$ and let $p = \delta(q_0, u)$. Then, for each integer $j \geq 0$ for which $s_y^j(u)$ is defined,*

$$f_y^j(p) = \delta(q_0, s_y^j(u)).$$

Proof. We prove the result by recurrence on j. If $j = 0$, $f_y^j(p) = p$ and $s_y^j(u) = u$, therefore the equality is satisfied by assumption.
Let $j > 0$ such as $s_y^j(u)$ is defined and suppose by recurrence assumption that $f_y^{j-1}(p) = \delta(i, s_y^{j-1}(u))$. By definition of f_y, $f_y(f_y^{j-1}(p))$ is the congruence class of the word $s_y(s_y^{j-1}(u))$. Consequently, $f_y^j(p) = \delta(q_0, s_y^j(u))$, which completes the recurrence and the proof. □

Corollary 2.4. *The terminal states of $\mathfrak{A}(y)$ are the states of the suffix path of last, SP(last).*

Proof. First, we prove that states of the path suffix are terminal. Let p be any state of *SP(last)*. One has $p = f_y^j(last)$ for some $j \geq 0$. Because $last = \delta(q_0, y)$, Proposition 2.3 implies $p = \delta(q_0, s_y^j(y))$; and as $s_y^j(y)$ is a suffix of y, p is a terminal state.
Conversely, let p be a terminal state of $\mathfrak{A}(y)$. Let then u be a suffix of y such that $p = \delta(q_0, u)$. Since u is a suffix of y, we can consider the greatest integer $j \geq 0$ for which $|u| \leq |s_y^j(y)|$. By Lemma 1.2 one obtains $u \equiv_y s_y^j(y)$. Thus, $p = \delta(q_0, s_y^j(y))$ by definition of $\mathfrak{A}(y)$. Therefore, Proposition 2.3 applied to y implies $p = f_y^j(last)$, which proves that p appears in *SP(last)*. This ends the proof. □

2.3 On-line construction

We present an on-line construction algorithm works in linear space with an execution time $O(|y| \times \log(\mathrm{Card}\mathcal{A}))$. The algorithm treats the prefixes of y from ε to y itself. At each stage, just after having treated prefix w, the following information is available:

- The suffix automaton $\mathfrak{A}(w)$ with its transition function δ.
- The table f, defined on the states of $\mathfrak{A}(w)$, which implements the suffix function f_w.
- The table L, defined on the states of $\mathfrak{A}(w)$, which implements the function length lg_w.
- The state *last*.

Terminal states of $\mathfrak{A}(w)$ are not explicitly marked, they are given implicitly by the suffix path of *last*.

```
function SuffixAutomaton(y, n)
    M ← NewAutomaton()
    L[initial(M)] ← 0
    last[M] ← initial(M)
    for each letter a of y, sequentially do
        Extension(a)
    p ← last[M]
    Do
        terminal(p) ← TRUE
        p ← f[p]
    While p is defined
    return M
```

function EXTENSION(a)
$new \leftarrow$ NEWSTATE()
$L[new] \leftarrow L[last[M]] + 1$
$p \leftarrow last[M]$
Do
$adj[p] \leftarrow adj[p] \cup \{(a, new)\}$
$p \leftarrow f[p]$
While p is defined and TARGET(p, a) is undefined
if p is undefined **then**
$f[new] \leftarrow initial(M)$
else
$q \leftarrow$ TARGET(p, a)
if (p, a, q) is a solid edge, that is, $L[p] + 1 = L[q]$ **then**
$f[new] \leftarrow q$
else
$f[new] \leftarrow$ CLONE(p, a, q)
$last[M] \leftarrow new$

function CLONE(p, a, q)
$clone \leftarrow$ NEWSTATE()
$L[clone] \leftarrow L[p] + 1$
$adj[clone] \leftarrow adj[q]$
$f[clone] \leftarrow f[q], f[q] \leftarrow clone$
Do
$adj[p] \leftarrow adj[p] \setminus \{(a, q)\}$
$adj[p] \leftarrow adj[p] \cup \{(a, clone)\}$
$p \leftarrow f[p]$
While p is defined and TARGET(p, a)
return *clone*

Theorem 2.5. *Algorithm* SUFFIXAUTOMATON *builds a suffix automaton, that is* SUFFIXAUTOMATON(y) *is the automaton* $\mathfrak{y}$*, for* $y \in \mathcal{A}^*$.

When the first loop stops, three disjoint cases arise:

1. p is not defined,

2. (p, a, q) is a solid edge,

3. (p, a, q) is a nonsolid edge.

The following four figures are described those three difference cases.

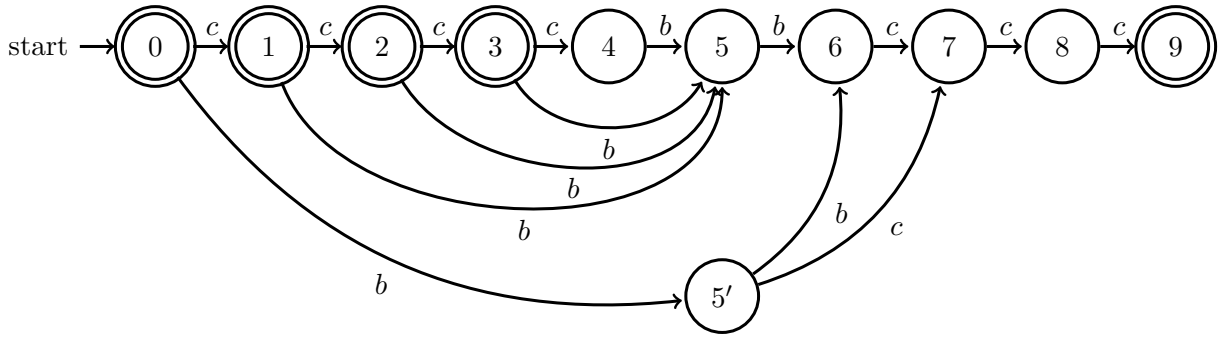

Figure 2.1: Suffix automaton $\mathfrak{A}(ccccbbccc)$ on which is illustrated in Figure 2.2, Figure 2.3 and Figure 2.4 the procedure Extension(a) according to three cases.

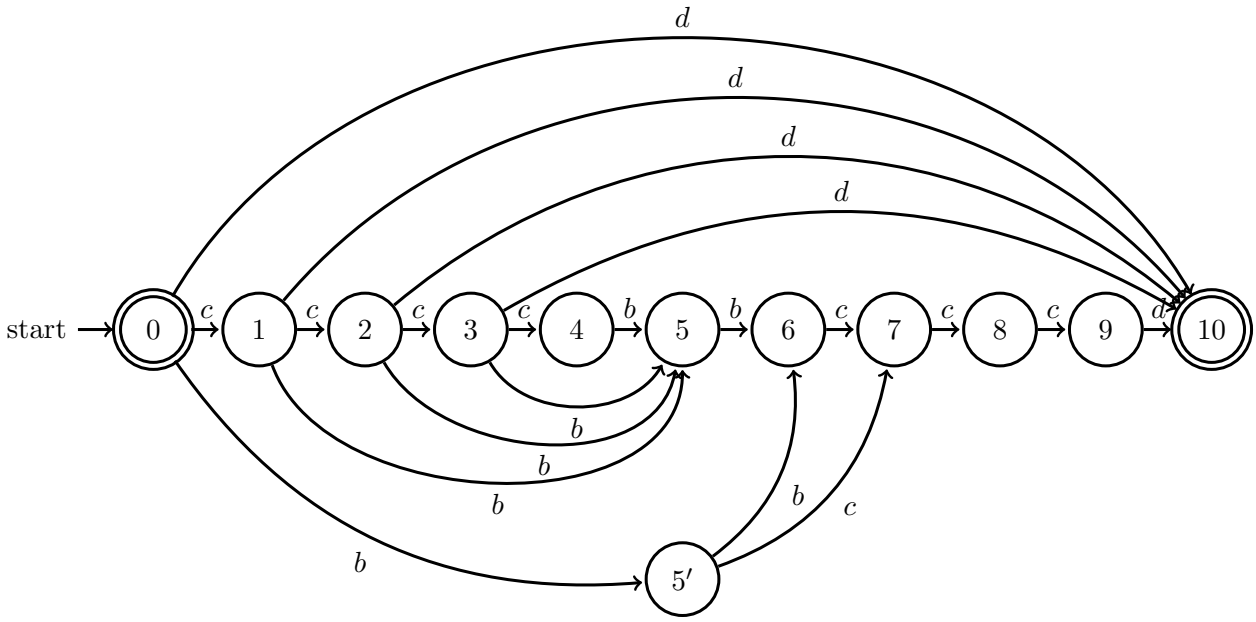

Figure 2.2: Suffix automaton $\mathfrak{A}(ccccbbcccd)$ obtained by extending $\mathfrak{A}(ccccbbccc)$ of Figure 2.1 by letter d. During the execution of the first loop of Extension(d), state p traverses the suffix path $\langle 9, 3, 2, 1, 0\rangle$ At the same time, edges labelled by letter d are created, starting from these states and leading to 10, the last created state. The loop stops at the initial state. This situation corresponds to Corollary 1.12.

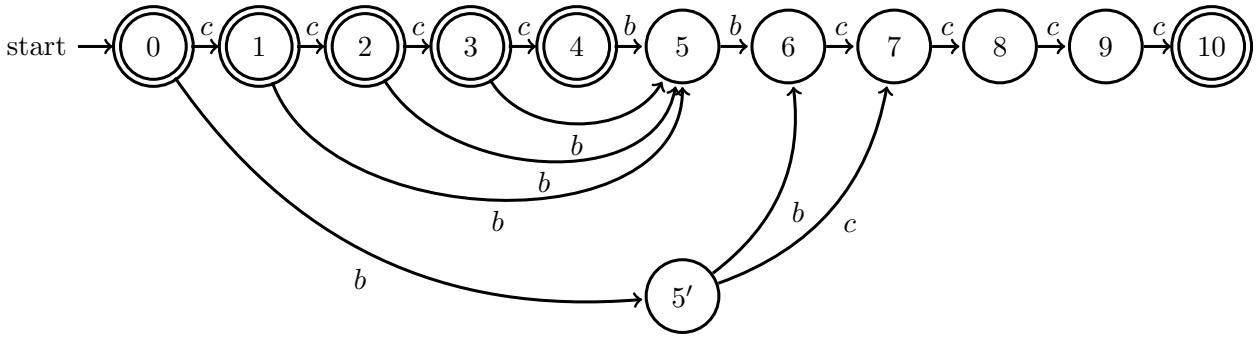

Figure 2.3: Suffix automaton $\mathfrak{A}(ccccbbcccc)$ obtained by extending $\mathfrak{A}(ccccbbccc)$ of Figure 2.1 by letter c. The first loop of the procedure EXTENSION(c) stops at state $3 = f[9]$ because an edge labelled by c starts from this state. Moreover, the edge $(3, c, 4)$ is solid. We obtain directly the suffix link of the new state created: $f[10] = \delta(3, c) = 4$. There is nothing else to do according to Corollary 1.11.

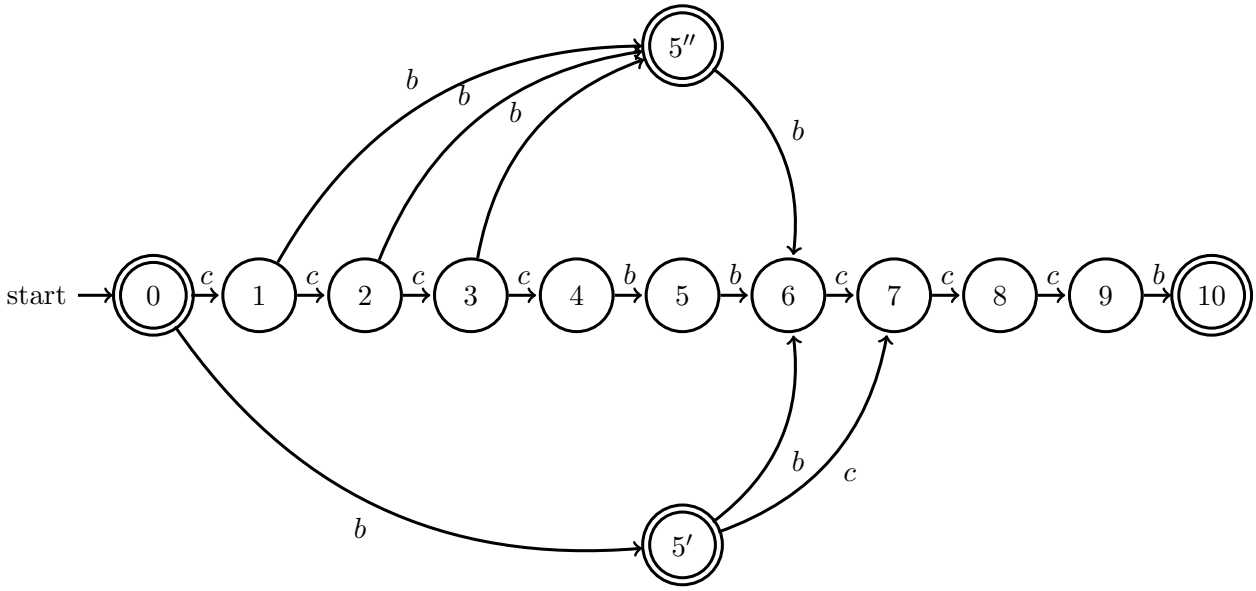

Figure 2.4: Suffix automaton $\mathfrak{A}(ccccbbcccb)$ obtained by extending $\mathfrak{A}(ccccbbccc)$ of Figure 2.1 by letter b. The first loop of the procedure EXTENSION(c) stops at state $3 = f[9]$ because an edge labelled by b starts from this state. In the automaton $\mathfrak{A}(ccccbbccc)$ edge$(3, b, 5)$ is not solid. The word $cccb$ is a suffix of $ccccbbcccb$ but $ccccb$ is not, although they both lead to state 5. This state is duplicated into the final state 5" that is the class of factors $cccb, ccb$ and cb. Edges $(3, b, 5), (2, b, 5)$ and $(1, b, 5)$ of $\mathfrak{A}(ccccbbccc)$ are redirected onto 5" according to Theorem 1.10.