<table>
<thead>
<tr>
<th>Title</th>
<th>The algorithms for deciding some properties of finite convergent string rewriting systems (Theory and applications in computer algebra)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Mingchao, Guo; Lian, Li</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 1993, 848: 177-187</td>
</tr>
<tr>
<td>Issue Date</td>
<td>1993-09-01</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/83652">http://hdl.handle.net/2433/83652</a></td>
</tr>
<tr>
<td>Type</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td>Textversion</td>
<td>publisher</td>
</tr>
</tbody>
</table>

Kyoto University
The algorithms for deciding some properties of finite convergent string rewriting systems

Guo Mingchao, Li Lian
Lanzhou University, Gansu, China

Abstract. The following problems are decidable in $O(mn^2)$ time for convergent system $R$ on alphabet $A$, where $m = |A|$, $n$ is the length of $R$, i.e., the sum of all length of words appeared in $R$.

(1) Is the monoid presented by $R$ finite?
(2) How many elements in the monoid presented by $R$ if it is finite?
and a sofware is given to decide these properties and some other properties for such monoid.

1. String rewriting systems

Let $A$ be an alphabet, $R$ a subset of $A^* \times A^*$ which is called the set pf string rewriting rules. Elements in $R$ has the the form of $(x, y)$. Let $>$ be an order which satisfies that if $x > y$, then for arbitrary strings $w, z \in R$, $wxz > wyz$. A string rewriting system $RS$ is a double $(A, R)$ where $A$ is an alphabet and $R$ is a set of rewriting rules. An oriented $RS$ is triple $(A, R, >)$, for each rule $(x, y)$ in $R$, we have $x > y$ and denote as $x \rightarrow y$.

Some reduction relation on $A^*$ is defined as follows:

(a) $w_1 \rightarrow w_2$ iff there exists $z_1, z_2, (x, y) \in R$, such that $w_1 = z_1xz_2, w_2 = z_1yz_2$.

(b) $\rightarrow^*$ is the reflexive transitive closure of $\rightarrow$.

(c) $\leftrightarrow^*$ is the symmetric closure of $\rightarrow^*$.

A word $w$ is called irreducible if there is no word $z$ such that $w \rightarrow z$.

An $ORS$ is Noetherian if there is no infinite chain $w_1 \rightarrow w_2 \rightarrow \cdots \rightarrow w_n \rightarrow \cdots$.

An $ORS$ is confluent iff for arbitrary words $w_1, w_2, w_1 \leftrightarrow w_2$, there exists a word $z$ such that $w_1 \rightarrow^* z, w_2 \rightarrow^* z$.

An $ORS$ is convergent iff it is Noetherian and confluent.

$G$ is a generating relation of a monoid on alphabet $A$. Let $\rho$ be the minimum congruence containing $G$, then

$$M \cong A^*/\rho.$$ 

$\rho = \leftrightarrow^*$ when $G$ is reviewed as a set of rules, so we call $A^*/\leftrightarrow^*$ is the monoid defined by $ORS$. When $ORS$ is convergent, the following properties are decidable [2].
(1) Is the monoid presented by $R$ finite?

(2) How many elements are there the monoid presented by $R$ if it is finite?

(3) How is the multiplication table of the monoid presented by $R$ created?

(4) Is it a trivial monoid?

(5) Is it a group?

(6) Is it commutative?

(7) Is it a free monoid?

Whether an ORS is convergent is decidable [2]. If the ORS is not convergent, we can use the Knuth-Bendix convergent procedure to get an equivalent one in most case (But there indeed exists some ORS that has no convergent system). In the following discussion, we always suppose that ORS is convergent. About the properties of (4), (5), (6) and (7), F. Otto has given some algorithms in polynomial time [2, 4]. Now we turn our attention to the properties of (1), (2) and (3).

2. The decision of Properties (1), (2) and (3).

The number of equivalent classes is the number of elements in monoid when the ORS is convergent. And every class has exactly one irreducible element. The set of all irreducible words $IRR(R)$ is a regular language which can be accepted by a finite state automaton. The cardinal of $IRR(R)$ is equal to get the cardinal of monoid defined by the rewriting system. So we can count the cardinal to get the cardinal of the monoid. Some details are as follows.

Denote $\text{suffix}(x) := \{v | x = uv, u, v \in A^*\}$
$\text{dom}(R) := \{x | (x, y) \in R\}$
$\text{prefix}(R) := \{x | l = xy, y \in A^+, l \in \text{dom}(R)\}.$

We construct the automaton $FSA$ recognizing $IRR(R)$. The states of $FSA$ are those words which are proper prefixes of the left-side of the rules in $R$ where $\epsilon$ is the start state. All the states are final states. We denote the state set as $F$. The transitive function $\delta$ is constructed by following way.

$$\delta(w, a) = \begin{cases} \text{undefined} & \text{suffix}(wa) \cap \text{dom}(R) \neq \emptyset \\ s & \text{if } s \in L = \{x | x \in \text{suffix}(wa) \cap \text{prefix}(R)\}, \\ & \text{and } y \in L - \{s\}, |s| > |y| \end{cases}$$

Lemma 1: $IRR(R) = L(FSA)$ [5].
Theorem 2: Let $FSA(Q, A, \delta, e, F)$ be the automaton on alphabet $A$ constructed as above. Then for the state $w \in F$ and $a \in A$, to determine the next state $\delta(w, a)$ needs $O(n)$ time, where $n = |R|$.

Proof: The key to determine the next state $\delta(w, a)$ is to find the suffix of $wa$ such that $wa$ is exactly a prefix of the left-side of a rule in $R$.

Suppose the word $u$ is the left-side of a rule, $S(u) = \{ \text{The longest word in prefix of } u \text{ and suffix of } wa \}$; obviously, $S(u) \neq \emptyset$.

If $u$ itself is a suffix of $wa$, then $\delta(w, a)$ is undefined, otherwise for arbitrary $u$ in the left-side of the rule, the problem to find the longest word $S(u)$ is a substring recognizing problem [1]. It takes $O(|u|)$ time. When $u$ runs all the left of the rule in $R$, it needs $O(n)$ time.

Theorem 3: $FSA$ as above can be constructed in $O(mn^2)$ time, where $m = |A|$, $n = |R|$.

Proof: The automaton has been constructed if we have the transitive function. For every state $w$ and $a \in A$, the next state $\delta(w, a)$ can be determined in $O(n)$ steps. The number of states is less than $n$, so for all states and some $a \in A$, to determine the next states needs $O(n^2)$ time. When a runs through $A$, it takes $O(mn^2)$.

Theorem 4: The monoid presented by $R$ is infinite iff the automaton $FSA$ recognizing $IRR(R)$ has a circle.

Proof: If the monoid is infinite, then there is an element $w \in IRR(R)$, $|w| > n$, where $n$ is the number of the states of $FSA$. For $FSA$ has only $n$ states, so $w$ must pass some state twice at least, i.e., $FSA$ contains a circle. Conversely, if $FSA$ has a circle, there exist states $q, r$ and a word $w = xyz$, which satisfies

$$\delta(e, x) = q, \quad \delta(q, y) = q, \quad \delta(q, z) = r.$$ 

Then $FSA$ accepts all the words like $xy^kz$. The monoid presented by $R$ is infinite.

Algorithm 5:

INPUT: A finite automaton recognizing $IRR(R)$ on alphabet $A$.

begin $s_1 := \{ e \}$;
for $i := 1$ to $n$ do
begin $S_2 := \emptyset$;
begin $S_2 := \bigcup_{p \in S_1, a \in A} \delta(p, a)$;
$S_1 := S_2$;
end;
if $S_2 = \emptyset$ then OUTPUT : The monoid is finite
else OUTPUT : The monoid is infinite
end.
Theorem 6: The algorithm above needs \( O(mn) \) time to determine the existence of circle, where \( m = |R| \) and \( n \) is the number of states in automaton \( FSA \).

Proof: clearly.

If the monoid presented by \( R \) is finite, then \( FSA \) recognizing \( IRR(R) \) has no circle, denote the number of words accepted by state \( q \) as \( N(q) \), obviously, \( N(e) = 1 \).

\[
R(q) = \{ p \mid \exists a \in A : \delta(p, a) = q \}.
\]

\[
T(q, p) = | \{ a \in A \mid \delta(p, a) = q \} |.
\]

Theorem 7: Let \( FSA(Q, A, \delta, e, F) \) be a automaton recognizing \( IRR(R) \) which has no circle. Then it satisfies

(a) For each \( q \)

\[
N(q) = \begin{cases} 
1 & \text{if } q = e \\
\sum_{p \in R(q)} N(p)T(p, q) & \text{if } q \neq e
\end{cases}
\]

(b) \( |IRR(R)| = \sum_{q \in F} N(q) \)

Proof: (a). denote \( L(q) \) as the path length from start state to the state \( q \). Now let us induce on \( L(q) \).

\( L(q) = 0 \), then \( q \) is start state. It is obviously that \( N(q) = 1 \).

If \( L(q) < t \), the result is true. Let \( L(q) = t \), every path from \( e \) to \( q \) must pass uniquely a state \( p \) in \( R(q) \), by the induction, the number of word from \( e \) to \( p \) is \( N(p) \). So the number of words passing \( p \) from \( e \) to \( q \) is \( N(p)T(p, q) \), i.e., \( N(q) = \sum_{p \in R(q)} N(p)T(p, q) \).

(b) is clear.

Now we give the algorithm for calculating the order of the monoid.

Algorithm 8:

INPUT : Automaton \( FSA(Q, A, \delta, e, F) \) accepting \( IRR(R) \).

begin \( U := \{ e \} \);  
\( N\{ e \} := 1 \);  
\( S := \{ q \mid \exists p \in U, a \in A : \delta(p, a) = q \} \);  
While \( S \neq \emptyset \) do  
begin \( T := \emptyset \);  
for each \( q \in S \) do  
if \( R(q) \subseteq U \) then  
begin \( U := U \cup \{ q \} \);  
\( N(q) := \sum_{p \in R(q)} N(p)T(p, q) \);  
end
\[ T := T \cup \{ q \}; \]
\end{verbatim}

end;
\end{verbatim}

\begin{verbatim}
S := T;
end;
end.
\end{verbatim}

**OUTPUT**: \( \text{order} := \sum_{q \in F} N(q) \).

Theorem 9: The algorithm 8 takes \( O(mn^2) \) time where \( m = |A| \), \( n \) is the number of states of the automaton.

For constructing the multiplication table from a monoid, we can calculate the irreducible words of monoid at first. Denote \( W(p) := \{ x | \delta(e, x) := p \} \). The algorithm is similar with algorithm 8 and given as follows:

Algorithm 10:

**INPUT**: A automaton \( FSA(Q, A, \delta, e, F) \) recognizing \( IRR(R) \), where \( IRR(R) \) is finite.

\begin{verbatim}
begin W(e) := \{ 1 \};
U := \{ e \};
S := \{ q | \exists p \in U, a \in A : \delta(p, a) = q \};
While S \neq \emptyset do
begin T := \emptyset;
for each p \in S do
if R(p) \subseteq U then
begin U := U \cup \{ p \};
W(p) := \bigcup_{p \in R(p), a \in A} \{ xa | x \in W(q), \delta(q, a) = p \};
T := T \cup \{ p \};
end;
S := T;
end;
end.
\end{verbatim}

**OUTPUT**: \( IRR(R) := \bigcup_{p \in F} W(p) \).

Theorem 11: If monoid presented by \( R \) is finite, we can calculating \( IRR(R) \) in polynomial \( O(mn^2t) \) time, where \( m = |a| \), \( n \) is the number of states of \( FSA \), \( t = |IRR(R)| \).

If the order of elements is \( t \) and the length of longest element is \( n \), to calculate the irreducible words for all production of pairwise elements needs \( O(n) \) time. For there is a linear time algorithm to get the irreducible word from a given word with length \( 2n \) \[2\]. So we can create the multiplication table in \( O(nt^2) \) time.
If the order of the monoid to large, or it is infinite, to construct the whole multiplication table is impossible. But we can create a finite block of the table when the concrete elements is given. The following algorithm give finite elements which lengths are less than $K$ in a monoid.

Algorithm 12:

INPUT: $FSA(Q, A, \delta, e, F)$ recognizing $IRR(R)$, an integer $k > 0$.

begin
  $S := \{e\}$;
  $W := \{1\}$;
  While $(S \neq \emptyset)$ and ($loop < k$) do
  begin
    $T := \emptyset$;
    $loop := loop + 1$;
    For each $(x, q) \in S$ do
    begin
      $M_{(x, q)} := \{(xa, p) | \delta(q, a) = p, a \in A\}$
      If $M_{(x, q)} \neq \emptyset$ then
      begin
        $W := W \cup \{y | (y, p) \in M_{(x, q)}\}$;
        $T := T \cup M_{(x, q)}$;
      end;
      $S := T$;
    end;
  end;

OUTPUT: If $S = \emptyset$ then $IRR(R) := W$
else $W$ is a irreducible words set whose element's length is less than $k$.

When $|R| = 1$, then algorithm 12 takes $O(k)$ time. When $|A| > 1$, then it takes $O(\frac{m^{k+1}-1}{m-1})$ time.

3. Some examples about monoid presented by rewriting systems

Based on the discussion above and [2, 4]. A software is designed and works well. It can get a convergent system from a generated relation and decide the properties of 1 to 7. The following are examples running on the software.
1. 
Generated relation:
\[ R = \{ \text{aaaab} = \text{ba}, \quad \text{bbb} = 1, \quad \text{babb} = \text{aaaa}, \quad \text{aaaba} = \text{b}, \quad \text{baa} = \text{ab}, \quad \text{aaaba} = \text{babab} \}
\]

Lexicographical order
Convergent system:
\[ R = \{ \text{aabab} = \text{baba}, \quad \text{aabb} = \text{bba}, \quad \text{ababa} = \text{bb}, \quad \text{bbab} = \text{aa}, \quad \text{aaaaa} = \text{babab} \}
\]

Properties:
- trivial: NO
- finite: YES
- order: 21
- commutative: NO
- free monoid: NO
- group: YES

A finite multiplication table as following:

\[ \begin{array}{cccc}
\text{aab} & \text{abba} & \text{abab} & \text{aaaa} \\
\text{b} & \text{aab} & \text{aaaa} & \text{babab} & \text{aaab} \\
\text{ba} & \text{bba} & \text{aa} & \text{aaa} & \text{aaab} \\
\text{bb} & \text{babab} & \text{aba} & \text{aaab} & \text{abba} \\
\text{aa} & \text{aba} & \text{abab} & \text{bb} & 1 \\
\text{ab} & \text{baba} & \text{babab} & 1 & \text{b} \\
\text{aba} & \text{abba} & \text{aaa} & \text{aaaa} & \text{ba} \\
\text{abb} & 1 & \text{b} & \text{ba} & \text{bab} \\
\text{aaa} & \text{aaba} & \text{baba} & \text{abb} & \text{a} \\
\text{aab} & \text{bb} & 1 & \text{a} & \text{ab} \\
\text{bba} & \text{a} & \text{ab} & \text{aba} & \text{abab} \\
\text{bab} & \text{aaa} & \text{aba} & \text{aab} & \text{bba} \\
\text{baba} & \text{aaaa} & \text{aba} & \text{aab} & \text{bba} \\
\text{aaba} & \text{bab} & \text{aaaa} & \text{aaaaa} & \text{aba} \\
\text{aaaa} & \text{b} & \text{bb} & \text{bba} & \text{aa} \\
\text{aaab} & \text{abb} & \text{a} & \text{aa} & \text{aab} \\
\text{abba} & \text{aa} & \text{aab} & \text{aaba} & \text{baba} \\
\text{abab} & \text{aaaa} & \text{ba} & \text{ab} & \text{abb} \\
\text{aaaaa} & \text{ab} & \text{abb} & \text{abba} & \text{aa} \\
\text{babab} & \text{aab} & \text{bba} & \text{bab} & \text{aaaa} \\
\end{array} \]
2.
Generated relation:
\[ R = \{ \text{abb = aa, ababababab = b, ab = ba} \} \]

Lexicographical order
Convergent system:
\[ R = \{ \text{ba = ab, aaaaaaa = b, abb = aa, bbb = ab} \} \]

Properties:
- trivial: NO
- finite: YES
- order: 19
- commutative: YES
- free monoid: NO
- group: NO

A finite multiplication table as following:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>b</th>
<th>a</th>
<th>ab</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>b</td>
<td>a</td>
<td>ab</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>bb</td>
<td>ab</td>
<td>aa</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>ab</td>
<td>aa</td>
<td>aab</td>
</tr>
<tr>
<td>ab</td>
<td>ab</td>
<td>aa</td>
<td>.aab</td>
<td>aaa</td>
</tr>
<tr>
<td>aa</td>
<td>aa</td>
<td>aab</td>
<td>aaa</td>
<td>aaab</td>
</tr>
<tr>
<td>bb</td>
<td>bb</td>
<td>ab</td>
<td>aa</td>
<td>aab</td>
</tr>
<tr>
<td>aab</td>
<td>aab</td>
<td>aaa</td>
<td>aab</td>
<td>aaaa</td>
</tr>
<tr>
<td>aaa</td>
<td>aaa</td>
<td>aab</td>
<td>aaaa</td>
<td>aaaaab</td>
</tr>
<tr>
<td>aaab</td>
<td>aaab</td>
<td>aaaa</td>
<td>aaaaab</td>
<td>aaaaab</td>
</tr>
<tr>
<td>aaaa</td>
<td>aaaa</td>
<td>aaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
</tr>
<tr>
<td>aaaaa</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaab</td>
</tr>
<tr>
<td>aaaaaa</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>aaaaaaa</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>aaaaaaab</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>aaaaaaab</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>aaaaaaaab</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>aaaaaaaab</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>aaaaaaaba</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>aaaaaaab</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>aaaaaaaab</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
<tr>
<td>aaaaaaaba</td>
<td>aaaaaab</td>
<td>aaaaa</td>
<td>aaaaaaa</td>
<td>aaaaaaaa</td>
</tr>
</tbody>
</table>
3.
Generated relation:

\[ R = \{ baba = abab \quad \text{cbacbabc} = bc \text{abcba} \quad chc = bc \text{c}\quad ca = ac \quad aa = 1 \quad bb = 1 \quad cc = 1 \} \]

Lexicographical order
Convergent system:

\[ R \text{ is convergent.} \]

Properties:

<table>
<thead>
<tr>
<th>trivial:</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite:</td>
<td>NO</td>
</tr>
<tr>
<td>commutative:</td>
<td>NO</td>
</tr>
<tr>
<td>free monoid:</td>
<td>NO</td>
</tr>
<tr>
<td>group:</td>
<td>YES</td>
</tr>
</tbody>
</table>

A finite multiplication table as following:

<table>
<thead>
<tr>
<th></th>
<th>bab</th>
<th>bcb</th>
<th>abc</th>
<th>aba</th>
</tr>
</thead>
<tbody>
<tr>
<td>acbab</td>
<td>ac</td>
<td>acba</td>
<td>cbac</td>
<td>cb</td>
</tr>
<tr>
<td>acbabc</td>
<td>abcab</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>acbabc</td>
<td>abcb</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
<tr>
<td>abacbc</td>
<td>ababc</td>
<td>abcabc</td>
<td>cbabcba</td>
<td>cbabcba</td>
</tr>
</tbody>
</table>

Semigroup Multiplication Table
4.
Generated relation:
\[ R = \{ cc = 1 \quad bb = 1 \quad aa = 1 \quad ca = ac \quad bab = aba \quad cbacbacb = bcbacbabc \quad cbcbc = bcbcb \quad cbacb = bcbacb \quad cbacbaca = bcbacbabc \} \]

Lexicographical order
Convergent system:
\[ R \] is convergent.
Properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>trivial</td>
<td>NO</td>
</tr>
<tr>
<td>finite</td>
<td>YES</td>
</tr>
<tr>
<td>order</td>
<td>120</td>
</tr>
<tr>
<td>commutative</td>
<td>NO</td>
</tr>
<tr>
<td>free monoid</td>
<td>NO</td>
</tr>
<tr>
<td>group</td>
<td>YES</td>
</tr>
</tbody>
</table>

A finite multiplication table as following:

<table>
<thead>
<tr>
<th></th>
<th>cb</th>
<th>cbc</th>
<th>cba</th>
<th>bac</th>
</tr>
</thead>
<tbody>
<tr>
<td>bacbcba</td>
<td>abacbacba</td>
<td>abacbacbac</td>
<td>abacbacb</td>
<td>bacbacbc</td>
</tr>
<tr>
<td>bacbacb</td>
<td>bcbacbcb</td>
<td>bcbacbcb</td>
<td>bacbacbca</td>
<td>bacb</td>
</tr>
<tr>
<td>bcbacb</td>
<td>bcbacb</td>
<td>bcb</td>
<td>bcbb</td>
<td>bcbbacbca</td>
</tr>
<tr>
<td>bcacbca</td>
<td>bcbacbca</td>
<td>bcbacbca</td>
<td>bcbacbca</td>
<td>cbcb</td>
</tr>
<tr>
<td>abcacb</td>
<td>abacbac</td>
<td>abacbabc</td>
<td>abacbacb</td>
<td>abcbbb</td>
</tr>
<tr>
<td>abcacb</td>
<td>abacbabc</td>
<td>abacbac</td>
<td>abacbacb</td>
<td>abcbbb</td>
</tr>
<tr>
<td>abacbac</td>
<td>abcacb</td>
<td>abcacb</td>
<td>abcacb</td>
<td>abcacb</td>
</tr>
<tr>
<td>ababcab</td>
<td>babcabc</td>
<td>babcabc</td>
<td>babcabc</td>
<td>abcabcab</td>
</tr>
<tr>
<td>ababc</td>
<td>ababc</td>
<td>ababc</td>
<td>ababc</td>
<td>abcabcab</td>
</tr>
<tr>
<td>abcabc</td>
<td>abcabc</td>
<td>abcabc</td>
<td>abcabc</td>
<td>abcabcab</td>
</tr>
<tr>
<td>abcabc</td>
<td>abcabc</td>
<td>abcabc</td>
<td>abcabc</td>
<td>abcabcab</td>
</tr>
<tr>
<td>abcabc</td>
<td>abcabc</td>
<td>abcabc</td>
<td>abcabc</td>
<td>abcabcab</td>
</tr>
</tbody>
</table>
References


