Summer 2011

Learning Author's Writing Pattern System By Automata

Qun Yu
San Jose State University

Learning Author's Writing Pattern System By Automata

A Writing Report

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Qun Yu

JUNE 2011

SAN JOSÉ STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Titled

LEARNING AUTHOR'S WRITING PATTERN SYSTEM BY AUTOMATA

By

Qun Yu

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Prof. Tsau Young Lin, Department of Computer Science	Date
Prof. Robert Chun, Department of Computer Science	Date
Prof. Soon Tee Teoh, Department of Computer Science	Date

APPROVED FOR THE UNIVERSITY

Associate Dean	Date

ABSTRACT

LEARNING AUTHOR'S WRITING PATTERN SYSTEM BY AUTOMATA
By
Qun Yu

The purpose of the report is to document our project's theory, implementation and test results. The project works on an automata-based learning system which models authors' writing characters with automatons. Since there were pervious works done by Dr. T.Y. Lin and Ms. S.X. Zhang, we continue on ALERGIA algorithm analysis and initial common pattern study in this project.

Although every author has his/her own writing style, such as sentence length and word frequency etc, there are always some similarities in writing style. We hypothesize that common strings fogged the expected test result, just like the noise in radio wave. This report gives the design and implementation of finding common pattern, as well as testing results.

This report also describes the implementation of ALERGIA algorithm based on paper of Learning Stochastic Regular Grammars by Means of a State Merging Method by Rafael C. Carrasco and Jose Oncina [2]. The coding is done in Java 6 on Eclipse Helios version.

ACKNOWLEDGEMENT

First of all, I am heartily thankful to my advisor, Dr. Tsau Young Lin. Without his inspiring guidance, I would not stay clear on my research subject and complete my project.

Moreover, I would like to thank Dr. Robert Chun and Dr. Soon Tee Teoh for being my report committee.

Also, I owe my deepest gratitude to my husband, son and parents who supported me in every possible respect during my study. I am grateful for the encouragement from friends and classmates, here I give special thanks to Shuang Wang for her helpful discussions on automata.

Finally, It is a pleasure to give my regards and blessings to all of those who helped me during my project work.

Table of Contents

List of Figures

List of Tables

1. Introduction

In 2005, Dr. Tsau Young Lin and P. Baliga performed research on applying automata to Intrusion Detection and obtained promising results [8], in which, an automaton is used to detect patterns, i.e., the sequences of system calls in a program. Three years later, in [1], Lin and S. X. Zhang published a paper for the text analysis based on automaton. In paper [1], the concept of pattern was switched to sequences of function words in a sentence. This writing report continues this research direction in [1]. There are two major works done in this project, ALERGIA algorithm implementation based on paper of *Learning Stochastic Regular Grammars by Means of a State Merging Method* by Rafael C. Carrasco and Jose Oncina[2], and common pattern analysis.

To setup an automata-based text analysis system, first, we transform a text file, such as a novel, into a sample file which contains only digits 0 to 4. In this step, it is critical to choose and divide function words into proper category. Second, we build an FPTA tree from the sample file. Then, we minimize the FPTA tree to an automaton using ALERGIA algorithm. Eventually, we input another sample file to the automaton for acceptance testing.

We take a further step on authors' writing analysis by introducing common pattern concept. We believe that certain percentage of accepted strings are common to every author, i.e., these strings are not specific strings of the test author.

After the short introduction, this report illustrates the details in later sections and the content divided into seven sections. In section 2, we discuss what function words are and why we use function words. In section 3, we introduce stochastic finite automata in brief. In section 4, we review Regular Language and ALERGIA algorithm, which is the theory to build an automaton from a set of regular language strings. In section 5, we describe how we apply the automata theories to our specific research target. Also, we present the implementation in details. Section 6 compares the ALERGIA algorithm implementation with previous work in [5]. In section 7, we focus on the method to build common pattern automaton, verify common pattern automaton and the experimental results. At last, in section 8, we give the conclusions of our project result and hypothesis on possible improvement in the implementation.

2. Function Words

There is a saying in Chinese: The writing mirrors the writer. Undoubtedly writers have their own writing habits and personality. But what can we rely on to distinguish the works from different writers? In linguistics, words are classified into open-class and closed class. Function words belong to closed-class, while content words belong to open-class. Typically the content words are nouns, verbs, adjectives and some adverbs, while function words are pronouns, conjunctions, prepositions, auxiliary verbs, and some adverbs [4]. According to the definition in [3] “Function words are words that have little lexical meaning or have ambiguous meaning, but instead serve to express grammatical relationships with other words within a sentence, or specify the attitude or mood of the speaker.”

Let’s discuss why we use function words to analyze authorship.

First, from the theory of Linguistics, function words in a language are unlikely to increase or change, and are elements of grammar. On the other hand, content words have specific lexical meaning but are open to change. For example, new content words are created for new emerging disease. Although function words have no special meaning defined in dictionary, they are in an important role of providing the grammatical sentence structure and help content words create meaningful sentences. Therefore, it is meaningful to model a writer’s writing character with his/her function words in works.

Second, usually we pay only attention on content words and tend to ignore the function words during reading; as a result, we tend to learn and mimic the usage of content words, but not function words. We conclude that writers have their own consistent habits in using function words.

In 1964, [12] Mosteller and Wallace identified 70 function words and used statistical inference to analyze authorship. They solved a historical dispute about the authorship of the Federalist successfully.

Previously, many of the approaches to analyze writing style stated in recent research literatures are based on statistic measurement of particular function

words frequency. In [1], Lin proposed and started the research on automata based authorship identification.

3. Finite State Automata

In this paper, “state” and “node” have equivalent meaning.

3.1 Finite State Automata

In the theory of computation, a Finite State Automata(FSA) is a model of pattern recognizer described with a five-tuple < Q, A , δ, q_0, F>, where:

- Q is a finite non-empty set of states.
- A is a finite non-empty set of input symbols.
- δ is a transition function from $Q \times A$ to Q.
- q_0 is the initial state which belongs to Q.
- F is the final states which is a subset of Q, F can be a empty set.

An FSA with the above definition has functions as following:

- The automaton always starts from the initial state q_0.
- The automaton read one input symbol at a time, and transits from the current state to next state according to the transition function δ. To be specific, let $q_{curretnt}$ be the current state and a the symbol just read, the automaton transits to the state given by $q_{next} = \delta(q_{curretnt}, a)$.
- When the automaton reaches the end of the input string, if the current state belongs to F, the input string is considered accepted by this FSA.

FSA is also known as Deterministic Finite State Automata (DFSA), which means that for every state ($q_{curretnt} \in Q$) and a particular transition function (δ), there is one and only one next state ($q_{next} \in Q$).

Often, we use a transition diagram to represent a DFSA (in brief, DFA). For example: Q ={0,1,2}, A={a}, F={1}, q_0={0}, and transition functions (δ) are defined as the following table:

State (q)	Input Symbol (a)	Transition Function ($\delta(q,a)$)
0	a	1
1	a	2

2	a	1

Table 1: Simple DFA Example Transition Functions

Figure 1 shows a state transition diagram for this DFA:

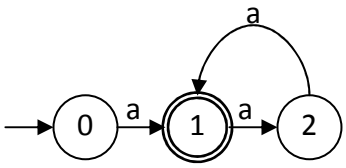


Figure 1: Simple DFA Example

From above transition diagram, we can see input string “a” and “aaa” can be accepted by this DFA.

A set of all strings in A^* that can be accepted by a DFA is considered as a regular language. A regular language may have an infinite number of strings and be represented with a DFA.

3.2 Stochastic Finite State Automata

The transition from current state q to next state might be random if there is more than one input symbols. For example, with two input symbol a_1 and a_2 from current state, the transition may go through either $\delta\ (q, a_1)$ or $\delta\ (q, a_2)$ randomly. Therefore, SFA comes into a role to represent a DFA with probabilities on transition functions.

Corresponding with Deterministic Finite Automata, another two terms are Deterministic Probabilistic Finite Automata (DPFA) and Deterministic Frequency Finite Automata (DFFA). In our research practice, we adopted DFFA, which indicates the frequency number for a transition between a pair of states. DPFA records the transition possibility with probabilistic data.

A Stochastic Regular Language contains transition probabilities on top of Regular Language.

4. ALERGIA Algorithm

ALERGIA algorithm is a state-merging algorithm from the probabilistic view. In this section, we review the ALERGIA algorithm described in [2] by Rafael C.

Carrasco and Jose Oncina. Figures 2,3 and 4 are from [2]. Figure 2 gives the top-level logic of ALERGIA algorithm. Figure 3 is the logic of finding compatible states. Figure 4 shows the logic of checking difference between two nodes.

```
algorithm ALERGIA
Input:
        S: sample set of strings
        α: 1 – confidence level
Output:
        SFA
Begin
        A = stochastic prefix tree acceptor from S
        Do (for j = successor(first node(A)  to last node(A))
                Do (for i = firstnode(A) to j)
                        If compatible(i,j)
                                Merge(A,i,j)
                                Determinize(A)
                                Exit (i loop)
                        End if
                End for
        End for
        Return A
End algorithm
```

Figure 2 : ALERGIA algorithm [2]

In the first step, ALERGIA algorithm takes a set of string (*S*) as input in order to construct a prefix tree (*T*). At every state, outgoing frequency by every single input symbol is calculated. We have the following notions for *T*:

- n_i is the number of strings passing state q_i
- $f_i(a)$ is the number of strings outgoing from state q_i to next state by transition function $\delta(q_i, a)$
- $f_i(\#)$ is the number of strings ending at state q_i

Two states (q_i, q_i) are considered equivalent only when their probabilities are equal, that is, $p_{if=} f_i(\#)/n_i$ and $p_{jf=} f_j(\#)/n_j$ are equal, $p_i(a)= f_i(a)/ n_i$ and $p_{jf=} f_j(a)/n_j$ are equivalent, as well as recursively their successors are equivalent.

Two states are considered as compatible states if the difference between their probabilities is less than a confidence range, and recursively their successors are compatible. ALERGIA algorithm will return false for equivalence check if any of the difference is larger than a confidence check. Formula for comparing two states q and q' are shown as following.

$$\left|\frac{f}{n} - \frac{f'}{n'}\right| > \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n}} + \frac{1}{\sqrt{n'}}\right).$$

```
algorithm compatible
Boolean compatible(i,j)
Input:
        i, j nodes
Output:
        Boolean
Begin
        If different(ni,fi(#),nj,fj(#))
                Return false
        End if
        Do (∀a ∈ A)
                If different(ni,fi(a),nj,fj(a))
                        Return false
```

```
                End if
                If not compatible(δ(i,a),δ(j,a))
                        Return false
                End if
        End do
        Return true
End algorithm
```

Figure 3: Algorithm compatible check [2]

```
algorithm different
Boolean different(n, f, n', f')
Input:
        n, n': number of strings arriving at each node
        f, f': number of strings ending/following a given arc
Output:
        Boolean
Begin
        Return
```

$$\left|\frac{f}{n}-\frac{f'}{n'}\right| > \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n}}+\frac{1}{\sqrt{n'}}\right)$$

End algorithm

Figure 4: Algorithm different check [2]

If two states are compatible, they are legible to merge. We use flag RED and BLUE to indicate state q_i and state q_j in A in algorithm ALERGIA (Figure 2), always merge BLUE state to RED state. We initialize RED state set with root state of FPTA tree, and add all the successors of root state to BLUE state. ALERGIA fetches a BLUE state at a time and loop through RED state set to find a RED state to merge. If no merge is found, this BLUE state is promoted to RED state. Continuously promote non-RED successors of each RED node to BLUE. Refer to Figure 5.

Note: The logic in Figure 5 and Figure 6 are our understanding of method calls in Figure 2.

```
algorithm merge

void Merge(A,i,j)

Input:
        A: FPTA tree
        i, j: nodes in A
Output:
        none
Begin
        k = getPredecessor(j);
    a = getInputSymbol(k,j);
    // redirect δ(k,a) from state j to state i
    addChild(k,i,a);
End algorithm
```

Figure 5: Algorithm merge

After removing the BLUE state q_j and its successors from FPTA tree in Algorithm merge, we need to fold the successors of BLUE state q_j to the successors of RED state. This step is the method call, Determinize(A),in Algorithm ALERGIA. In this method, first update RED node *i*'s $f_i(a)$ *and* $f_i(\#)$. And then recursively fold successors if both RED and BLUE states have next state by input symbol *a*. on the other hand, if RED states have no next state by input symbol *a*, we just redirect the BLUE state's successor to RED state. It is important to update FPTA tree's $f_i(a)$ *and* $f_i(\#)$ after every merge. See Figure 6 for Determinize algorithm.

```
algorithm Determinize

void Determinize(i,j)

Input:
                i, j: nodes merged in method Merge
Output:
                none
Begin
                UpdateFrequencyInfo(i);
      For(first(successor of j)to last(successor of j)
          If δ(i,a) is existing
            Determinize(δ(i,a), δ(j,a))
          Else
            // redirect δ(j,a)'s predecessor to i
             AddSuccessor(i, δ(j,a));
             UpdateFrequencyInfo(i);
           End If
       End For
End algorithm
```

Figure 6: Algorithm determinize

5. Project Implementation

As we discussed in section 2, Function Words are the elements of language grammar. They connect Content Words to make sentence meaningful. In our project, we classify Function Words into 5 categories which are indicated with digits 0 to 4. We extract and substitute function words in a text file with corresponding digits 0 to 4. The result file is so called sample file which contains only number strings. Once we get the sample file, first, build a FPTA tree from a sample file with input symbols (0,1,2,3,4) and a sentence as an input string. Then, minimize FPTA tree by applying ALERGIA algorithm to get an automaton, which represents this text file's writing character. Now, input a sample file to this automaton for acceptance percentage result, which shows the similarity between two text files.

Our implementations are done in Java 6.0. Since we need a large amount of experiments to find common pattern, the implementations are divided into four executable files: CreateSampleDataFile.jar , BuildAutomaton.jar , CheckAcceptance.jar and CheckAcceptanceWithFilterCmnptn.jar.

All the test jobs are performed in Cygwin with batch files which are created in bash scripting language. Building automaton is a time-consuming process because merge algorithm needs recursive comparisons to all successors for a pair of nodes. By dividing the process steps, we are able to separate the training (building automatons) time from acceptance test. Also, by saving automatons into a text file makes that the time-consuming job execute only once.

Since the aim of our implementation is not for end user purpose, there is no user interface provided.

5.1 Step1: Create Sample File from Text File

In the first step, we collect as many as possible writings of a particular author and extract Function Words from writings to build a sample file.

We use same Function Words as [1]. The detail information about Function Words can be found in [4]. The Function Words are classified into five categories:

- Adverbs other than manner adverbs and omitting the funny legal ones,

- Auxiliary verbs (including contractions),
- Prepositions/Conjunctions (one category since there is some overlap),
- Determiners/Pronouns (omitting archaic THOU, THEE, etc),
- Numbers.

There are 320+ Function Words in total. Refer to Appendix B for details.

According to the above Stop Words list, our program "`CreateSampleDataFile.java`" extracts Stop Word one by one and substitutes it with the category digit (0-4) to create a new file, i.e. a sample file.

An example of part of a sample file is shown as following:

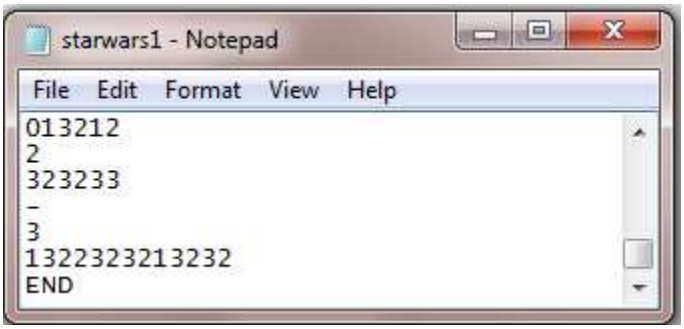


Figure 7: Step1 – Sample File

In Figure 7, each line is corresponds to StopWords structure of a sentence in text file and will be an input string for FPTA tree. The symbol,"-", represents an empty string. "END" indicates the end of sample file.

5.2 Step2: Build Automaton from Sample File

5.2.1 Step2.1: Build FPTA Tree

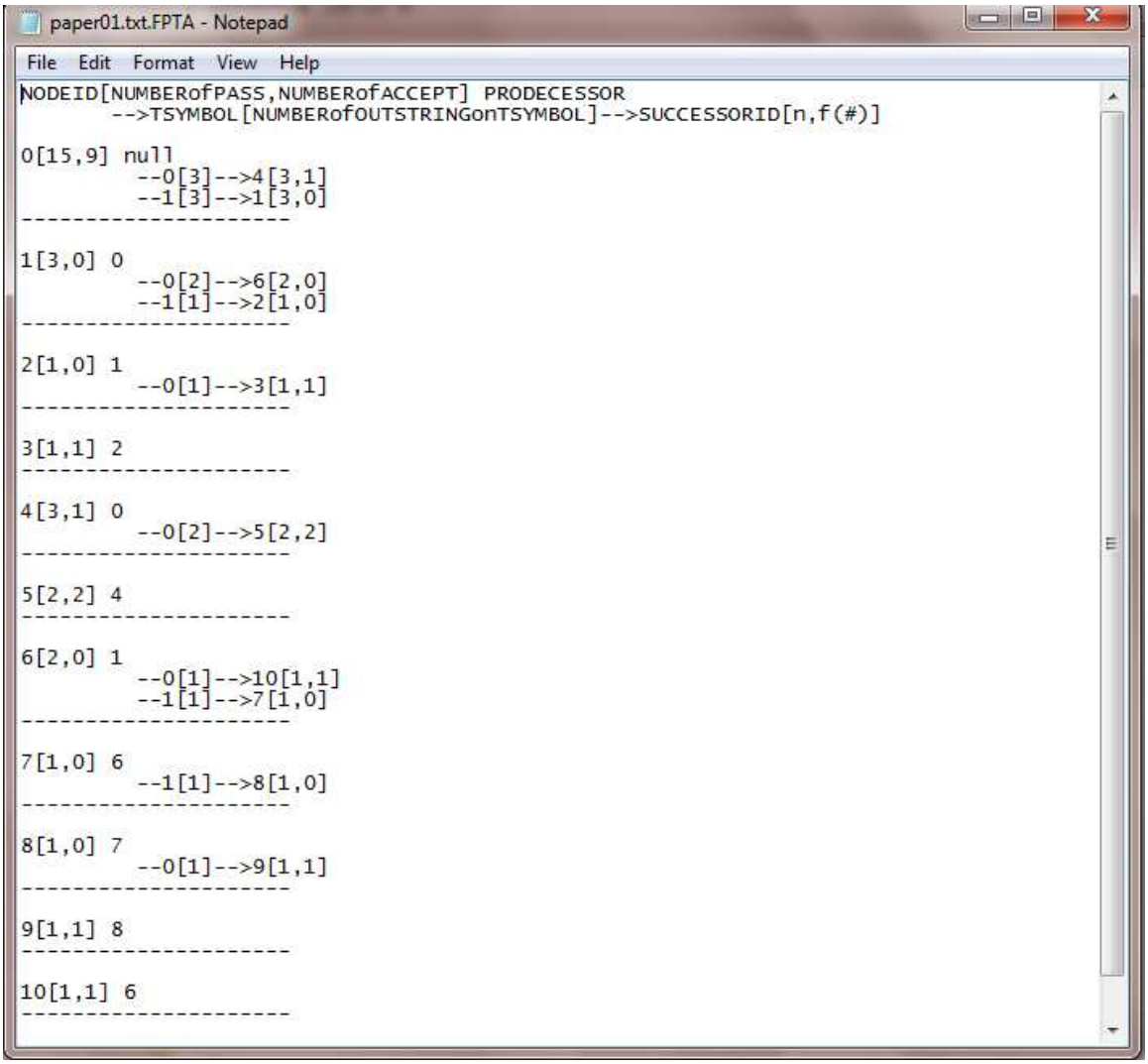
The method "`BuildFPTA`" in program "`BuildAutomaton.java`" takes a sample file as input. Process each line as an input string to build a FPTA tree. The following sample file *S* is from [2]. We use *S* as input file to build an automaton and illustrate the process with the data in our log/result file.

Strings in sample file *S* = {110, -,-,-,0,-,00,00,-,-,-,10110,-,-,100}

Initial FPTA tree with state (node) 0 : → (0)

For every input string, enters FPTA from node 0, with input symbol *a,* if $\delta(q_{curretnt}, a)$ is not existing, a new node is created. If the string ends at node $q_{curretnt}$, the current node is marked as accept state (node) with a double circle: 

The FPTA tree are built as following (log file of FPTA tree) from *S:*



According to the log file above, we can draw the corresponding FPTA tree diagram as following:

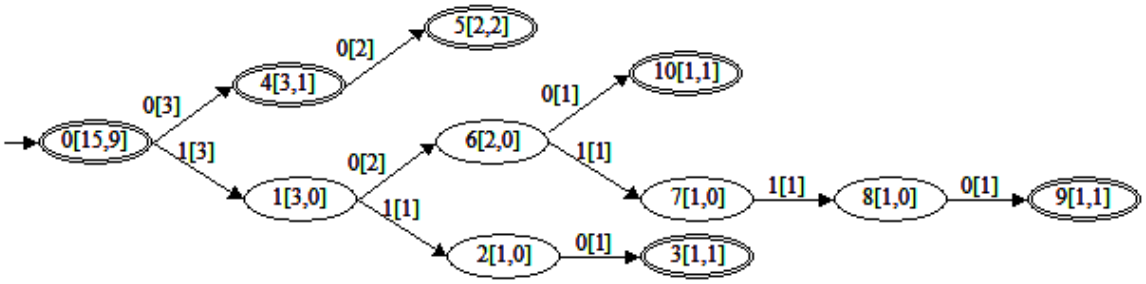


Figure 8: Step2 – FPTA tree

In Figure 8, "0[15,9]" indicates the information of node, i.e. nodeId[*number of strings pass the node(**n**), number of strings end at the node(**f(#)**)*]; 0[3] indicates the information about outgoing strings, i.e. input symbol[*number of strings leave from here(**f(i)**)*]. We can see the relationship of these information as **n = f(#) + sum(f(i))**

In order to explain our implementation well, the following is the Node and FPTA tree/Automaton declaration:

```
class Node { // define Node info
                    private int id;
                    private int numPass; // how many Strings pass this node
                    private int numAccept; // how many Strings ends at this node
                    private HashMap<Integer,Node> children; // <transition symbol, the child Node> ,
currently transition symbol is 0-4
                    private HashMap<Integer,Integer> numOut; // <transition symbol, the number of strings
follow the transition symbol from current node to child node>
                    private Node predecessor;
                    private int tSymbol;

                    public Node(int id){
                              this.setId(id);
                              this.setNumPass(0);
                              this.setNumAccept(0);
                              this.setChildren(new HashMap<Integer,Node>());
                              this.setNumOut(new HashMap<Integer,Integer>());
                              this.setPredecessor(null);
                              this.settSymbol(-1);
                    }

// get and set methods for private instance variable
Note: the methods are not shown for brevity
}
class DFFA{ //Deterministic Frequency Finite Automaton
          private Node root;
          private ArrayList<Node> nodes; // include root
          private ArrayList<Node> treeEnds; // didn't use
          private ArrayList<Node> red;
          private LinkedList<Node> blue;
```

```
        final static int tSymbolType = 5;

        public DFFA(Node root)
        {
                this.setRoot(root);
                this.setNodes(new ArrayList<Node>());
                this.addNode(root);
                this.setTreeEnds(new ArrayList<Node>());
                this.setRed(new ArrayList<Node>());
                this.setBlue(new LinkedList<Node>());
        }
// get and set methods for private instance variable

Note: the methods are not shown for brevity
}
```

5.2.2 Step2.2: Minimize FPTA tree

The method “minimizeFPTA” in program “BuildAutomaton.java” applies ALERGIA algorithm to merge compatible nodes and finally output a Deterministic Frequency Finite Automaton (DFFA).

The FPTA tree example in 5.2.1 becomes a DFFA after three merges (alpha=*0.8)*:

From the debug log file of our program, the result after the first merge is

```
0 and 4 is compatible
0  [20,12]
        --0[5]-->0[20,12]
        --1[3]-->1[3,0]
1  [3,0]
        --0[2]-->6[2,0]
        --1[1]-->2[1,0]
2  [1,0]
        --0[1]-->3[1,1]
3  [1,1]
4  [3,1]
        --0[2]-->5[2,2]
5  [2,2]
6  [2,0]
        --0[1]-->10[1,1]
        --1[1]-->7[1,0]
7  [1,0]
        --1[1]-->8[1,0]
8  [1,0]
        --0[1]-->9[1,1]
9  [1,1]
10  [1,1]
```

The transition diagram is shown as following:

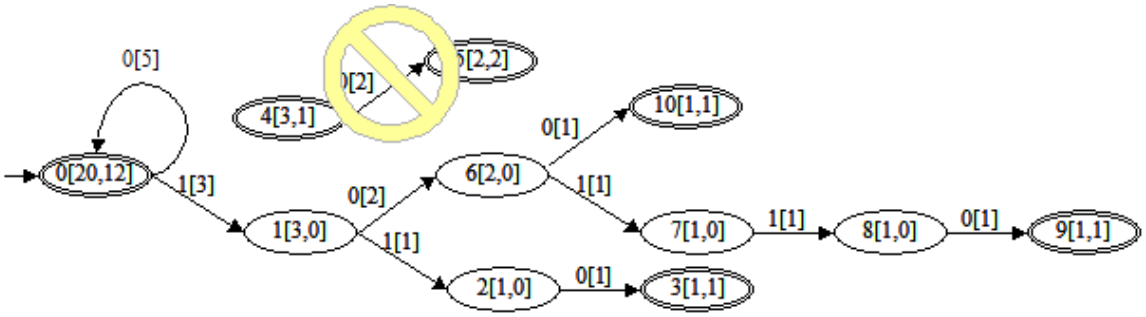


Figure 9: Step2 – FPTA after merging node 0 and 4

The result after the second merge is

```
0 and 6 is compatible
0   [23,13]
        --0[6]-->0[23,13]
        --1[4]-->1[4,0]
1   [4,0]
        --0[2]-->0[23,13]
        --1[2]-->2[2,0]
2   [2,0]
        --0[2]-->3[2,2]
3   [2,2]
4   [3,1]
        --0[2]-->5[2,2]
5   [2,2]
6   [2,0]
        --0[1]-->10[1,1]
        --1[1]-->7[1,0]
7   [1,0]
        --1[1]-->8[1,0]
8   [1,0]
        --0[1]-->9[1,1]
9   [1,1]
10   [1,1]
```

The transition diagram are shown as following:

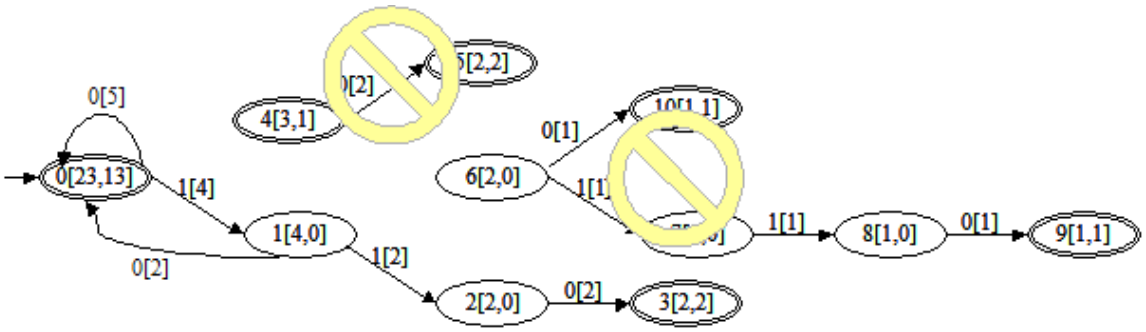


Figure 10: Step2 – FPTA after merging node 0 and 6

The final result after the third merge is

```
1 and 2 is compatible
0   [25,15]
        --0[6]-->0[25,15]
        --1[4]-->1[6,0]
1   [6,0]
        --0[4]-->0[25,15]
        --1[2]-->1[6,0]
2   [2,0]
        --0[2]-->3[2,2]
3   [2,2]
4   [3,1]
        --0[2]-->5[2,2]
5   [2,2]
6   [2,0]
        --0[1]-->10[1,1]
        --1[1]-->7[1,0]
7   [1,0]
        --1[1]-->8[1,0]
8   [1,0]
        --0[1]-->9[1,1]
9   [1,1]
10   [1,1]
```

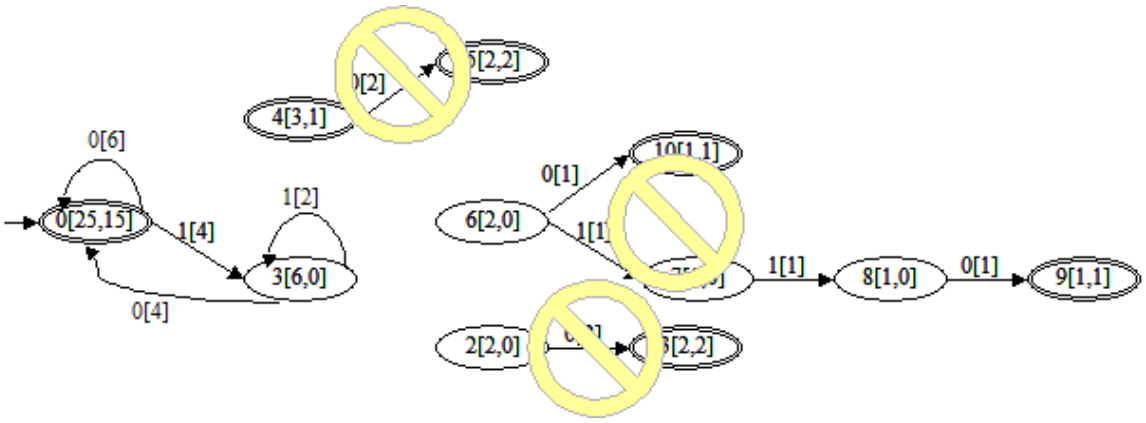


Figure 11: Step2 – FPTA after merging node 1 and 2

`minimizeFPTA` is the key method of ALERGIA algorithm. The following is part of source code of method `minimizeFPTA`:

```
        boolean promote = true;
        // initialize red and blue set
        // put 1st node in red
        Node root = theTree.getRoot();
        theTree.addRed(root.getId());

        // put 1st node's successors in blue
        HashMap<Integer, Integer> children = root.getChildrenIds();
        root=null;
        Iterator<Integer> iterator = children.keySet().iterator();
while(iterator.hasNext()){
        int key = (Integer) iterator.next();
        int value = children.get(key);
        if(value!=0) // don't have to add root itself to blue
                theTree.addBlue(value); //value is child node index in DFFA
}

// implement Alergia on FPTA to minimize it
ArrayList<Integer> red = theTree.getRed();
LinkedList<Integer> blue = theTree.getBlue();

while(!blue.isEmpty()){

        Node newRedNode = null;
      // for each node in blue to compare with each node in red
        int blueNodeId = blue.removeFirst();
                promote = true;
                Node blueNode = theTree.getNode(blueNodeId);
                for(int redNodeId : red ){
                        Node redNode = theTree.getNode(redNodeId);
                        if(isCompatible(redNodeId,blueNodeId,theTree)){
                                merge(redNode,blueNode,theTree);
                                determinize(redNode,blueNode,theTree);
                                promote = false;
```

```
                                        break;
                                }
                        }

                        if(promote){
                                // currently no merge is possible, promote current blue node to red
                                red.add(blueNodeId);
                                newRedNode = theTree.getNode(blueNodeId);

                                // promote no color successors of newRedNode to blue
                                children = newRedNode.getChildrenIds();
                           iterator = children.keySet().iterator();
                           while(iterator.hasNext()){
                                int key = (Integer) iterator.next();
                                int value = children.get(key);
                                //Node theNode = theTree.getNode(value);
                                if(theTree.getRed().contains(value) || theTree.getBlue().contains(value))
// this child has color red or blue
                                        continue;
                                else
                                        blue.add(value);
                           } //while
                        }
                   } //while
                theTree.setRed(red);
                theTree.setBlue(blue);
```

The final automaton is stored in `ArrayList<node> red` in class DFFA. Methods `saveAutomaton` and `readAutomaton` is a pair of functions which deal with writing/reading automaton to/from a text file.

Method `isDiffer` implements algorithm *difference check* in Figure 4.

```
private boolean isDiffer(double n1, double n2, double f1, double f2) {
        double d1 = Math.abs(f1/n1 - f2/n2);
double d2 =(Math.sqrt(1/n1) +   Math.sqrt(1/n2))*Math.sqrt(Math.log(2/alpha)/2);

        return d1 > d2;
}
```

Method `isCompatible` implements the algorithm *compatible check* in Figure 3. It takes two nodes, a red node and a blue node, as input, then returns true if these two nodes are compatible.

```
        private  boolean isCompatible(int redNodeId, int blueNodeId, DFFA theTree) {

                if ((redNodeId >= 0) && (blueNodeId >= 0)) {
                        Node redNode = theTree.getNode(redNodeId);
                        Node blueNode = theTree.getNode(blueNodeId);
                        int redPass = redNode.getNumPass();
                        int bluePass = blueNode.getNumPass();
```

```
                        if (isDiffer(redPass, bluePass, redNode.getNumAccept(),
blueNode.getNumAccept()))
                        {
                                return false; // didn't pass Alergia Function test ( n,f(#) )
                        }
                        HashMap<Integer,Integer> redChildrenIds = redNode.getChildrenIds();
                        HashMap<Integer,Integer> blueChildrenIds = blueNode.getChildrenIds();

                        for (int i = 0; i < tSymbolType; i++) {
                                if (isDiffer(redPass, bluePass,
redNode.getaNumOut(i),blueNode.getaNumOut(i)))
                                {
                                        return false; // didn't pass Alergia Function test ( n, f(i) ), i =
0~4 in this case
                                }
                                int redChildId = -1;

                                if(redChildrenIds.containsKey(i))
                                                redChildId = redNode.getChildId(i);

                                int blueChildId = -1;
                                if(blueChildrenIds.containsKey(i))
                                                blueChildId =
blueNode.getChildId(i);
                                if (!isCompatible(redChildId, blueChildId,theTree)) // recursive test for
successors
                                {
                                        redNode=null;
                                        blueNode=null;
                                        return false; // one pair of children are not Compatible
                                }
                        }
                        redNode=null;
                        blueNode=null;
                        return true;
                }
                else
                        return true;
        }
```

Method `merge` implements the algorithm in Figure 5. It takes two nodes, a red node and a blue node, as input; then removes blue node from its predecessor and add red node to blue node's predecessor.

```
        // merge a blue node to a red node
        private void merge(Node redNode, Node blueNode, DFFA theTree){
                // get the only predecessor of the blue node
                int thePredecessorId = blueNode.getPredecessorId();
                // redirect the relationship from the predecessor->blueNode to predecessor->redNode
           int tSymbol = blueNode.gettSymbol(); // tSymbol is the transition symbol which lead to blueNode
                Node thePredecessor = theTree.getNode(thePredecessorId);
           if(thePredecessor.getChildId(tSymbol) == blueNode.getId()){// make sure the correct node
                        thePredecessor.getChildrenIds().put(tSymbol, redNode.getId()); // blueNode lost
its connection with thePredecessor
                        blueNode.setPredecessorId(-1);
                }
                thePredecessor=null;
```

```
                redNode=null;
                blueNode=null;
        }
```

Method determinize implements the algorithm in Figure 6. It takes two nodes, a red node and a blue node, as input, then fold blue node's successors to red node's successors recursively if any.

```
//After merging, it's necessary to fold the isolated subtree into
//original DFFA, this method recursively folds the subtree into DFFA
private void determinize(Node redNode, Node blueNode){
// fold blueNode to redNode
// update pass and accept frequency on redNode
redNode.setNumPass( redNode.getNumPass() + blueNode.getNumPass());
redNode.setNumAccept( redNode.getNumAccept() + blueNode.getNumAccept());

HashMap<Integer, Integer> bluehm = blueNode.getNumOut();
HashMap<Integer, Integer> redhm = redNode.getNumOut();

if(!bluehm.isEmpty()){
                Iterator iterator = bluehm.keySet().iterator();
                while(iterator.hasNext()){
                        int key = (Integer) iterator.next();
                        int value = (Integer)bluehm.get(key);
                        if(redhm.containsKey(key))
                        value = value + (Integer)redhm.get(key);
                        redhm.put(key, value); // duplicated key will lead into over write the old value
                }
        }

        // fold successors
        // update frequency on redNode's successors
        HashMap<Integer, Node> redChildren = redNode.getChildren();
        HashMap<Integer, Node> blueChildren = blueNode.getChildren();

        if(!blueChildren.isEmpty()){ // blueNode has children needed to fold onto redNode
                for(int key=0; key<tSymbolType; key++){
                        Node blueChild = null;
                        if(blueChildren.containsKey(key)){
                                blueChild = (Node)blueChildren.get(key);
                                if(redChildren.containsKey(key)){ // redNode has child by tSymbol "key"
                                        Node redChild = redChildren.get(key);
                                        determinize(redChild,blueChild); // recursive call to fold all
successors
                                }else{ // redNode has no child by tSymbol "key", redirect blueChild-
>blueNode to blueChild->redNode
///*** possible need to add the child of blue to blue because it is a child of red now
        if(theTree.getRed().contains(redNode.getId()))
                if(!theTree.getRed().contains(blueChild.getId()) &&
!theTree.getBlue().contains(blueChild.getId()))
        theTree.getBlue().add(blueChildId);
        redNode.getChildren().put(key, blueChild);
                                        // blueChild split with old predecessor
                                        Node blueNodeOldPredecessor = blueChild.getPredecessor();

blueNodeOldPredecessor.getChildren().remove(key);
                                        blueChild.setPredecessor(redNode);
```

```
                                        redNode.getNumOut().put(key, blueChild.getNumPass());
                                        redNode.setNumPass(redNode.getNumPass() +
blueChild.getNumPass());
                                    }
                                    }//if
                                }//for
                        }//if

}//method
```

5.3 Step3: Check Acceptance for a sample file by an automaton

The `CheckAcceptance` method in program `CheckAcceptance.java` read a sample file one line at a time. As we stated in previous section, a line is an input string.

Empty string is accepted at start state, i.e. root node:

```
if(line.equals("-")){ // empty string => accepted string
        acceptedNum++;
        filewr.write(line);
        filewr.newLine();
        continue; // continue while loop to read next line
}
```

The input string is accepted only when it ends at an accepting state in automaton. The following is source code to check if the string is acceptable:

```
int l = line.length();
for(int i=0;i<l;i++){
        int tSymbol = Integer.parseInt(line.substring(i, i+1));
        Node theChild = theNode.getChild(tSymbol);
        if(theChild!=null){//yes, Can go through, this tSymbol transition is existing in Automaton
          if(i==l-1){//wow, sentence ends here, let's check if theChild is an accepting Node.
          if(theChild.getNumAccept()>0){//Good! accepted!
              acceptedNum++;
              filewr.write(line);
                  filewr.newLine();
            }else{//oops, theChild is not an accepting Node, rejected
                  rejectedNum++;
            }
        }else{//ok, string is not end yet, keep on going
            theNode = theChild;
                  }
        else{//Nope, Can't go through, rejected
             rejectedNum++;
        }
}//for
```

We have another method called `CheckAcceptanceWithFilterCmnptn`, which adds a constraint for accepting an input string. In a word, the method only accepts

non-common strings. We will discuss it further for the algorithm on finding common pattern.

6. Result

6.1 Automaton Built Verification

A SFA built from a set of strings should accepts that set of strings at any alpha setting. The following table shows our implementation obeys the rule.

	Test Book: Jane Eyre (26)	Test Book: Jane Eyre (30)
Alpha setting	Self test acceptance %	Self test acceptance %
0.1	100%	100%
0.2	100%	100%
0.3	100%	100%
0.4	100%	100%
0.5	100%	100%
0.6	100%	100%
0.7	100%	100%
0.8	100%	100%
0.9	100%	100%
1.0	100%	100%
1.1	100%	100%
1.2	100%	100%
1.3	100%	100%
1.4	100%	100%
1.5	100%	100%
1.6	100%	100%
1.7	100%	100%
1.8	100%	100%
1.9	100%	100%
2.0	100%	100%

Table 2: Automaton Built Verification

6.2 Discerning Ability Test

We use Harry Potter Book 5 as test automaton, and Harry Potter 3 (same author) and One Hundred Years of Solitude (different author) as sample files, to do discerning ability test.

The result looks slightly different from [1] because we pre-processed text file to eliminate misleading ". " and "?" generated by converting process from .pdf files to .txt file.

alpha setting	Harry Potter book 5 ***(self test)***	Harry Potter book 3	One Hundred Years Of Solitude	Difference
0.1	98	98	95	3
0.2	97	95	89	6
0.3	93	88	74	14
0.4	89	77	49	28
0.5	93	70	38	32
0.6	98	64	31	33
0.7	100	57	25	32
0.8	100	57	25	32
0.9	100	57	25	32
1.0	100	57	25	32
1.1	100	57	25	32
1.2	100	57	25	32
1.3	100	57	25	32
1.4	100	57	25	32
1.5	100	57	25	32
1.6	100	57	25	32
1.7	100	57	25	32
1.8	100	57	25	32
1.9	100	57	25	32
2.0	100	57	25	32

Table 3: Discerning Ability Test of Previous Work's Implementation [5]

From above table, we can see the flaw of this implementation is that automaton's self test couldn't be 100% when alpha is less than 0.7; also we can see the best alpha point is 0.6 or 0.7, and there is a big improvement when alpha from 0.3 to 0.4.

alpha setting	Harry Potter book 5 ***(self test)***	Harry Potter book 3	One Hundred Years Of Solitude	Difference
0.1	100	100	100	0
0.2	100	99.99	99.96	0.03
0.3	100	99.98	99.68	0.3
0.4	100	99.89	99.56	0.33
0.5	100	99.72	98.95	0.77
0.6	100	99.95	98.4	1.55
0.7	100	99.19	94.45	4.74
0.8	100	97.95	88.73	9.22
0.9	100	96.35	81.56	14.79
1.0	100	94.29	75.23	19.06

1.1	100	87.25	52.68	34.57
1.2	100	84.1	47.61	36.49
1.3	100	72.51	28.74	43.77
1.4	100	72.1	28.07	44.03
1.5	100	70.95	27.19	43.76
1.6	100	70.65	26.68	43.97
1.7	100	69.77	25.59	44.18
1.8	100	69.69	25.47	44.22
1.9	100	69.67	25.43	44.24
2.0	100	69.67	25.43	44.24
FPTA	100	69.67	25.43	44.24

Table 4: Discerning Ability Test of Our Implementation

Our Implementation shows better result at FPTA tree contrasting to [5] implementation at alpha 2.0, the reason maybe we did small changes in Function Words categories. That is, move "*one*" from category 3 to category 4, "*but*" and "*that*" from category 2 to category 3. Since Function Words has wide impact on input string and hence FPTA tree, small Function Words category changes bring great test result changes.

6.3 Performance

The following table shows the average time used to build an Automaton for 2 implementations on same computer (AMD Turion TL60):

Merge implementation	Average number of State in FPTA tree	Alpha setting	Running seconds
Our Implementation	52952	1.5	450
Previous Implementation [5]		0.7	83

Table 5: Running Time Compare (Build Automaton)

From the above table, we can see the speed advantage of [5] implementation because it only merges the states on branches and therefore it doesn't compare every pair of states in FPTA tree.

However, Table 14 shows the process time of acceptance test is longer as a trade-off. Automaton generated from [5] implementation has much more states because it only merges states on individual branches. As a result, it spends longer in traversing automaton for acceptance check.

Merge implementation	Number of test files	Running seconds
Our Implementation	49	27
Previous Implementation		477

Table 6: Running Time Compare (Check Acceptance)

7. Common Pattern

As it is named, Common Strings are those strings every author tends to use. In our perspective, a Common String is a string representing a sentence structure in a sample file, which could be accepted by every automaton.

7.1 Why Common Pattern

Although writers have their distinct writing characters, they couldn't be completely different and must be something in common more or less. Lin deducts there are some strings that can be accepted by every automaton. It is interesting to know: 1) is there a large amount of common strings? 2) How to find common pattern? 3) How common strings affect acceptance percentage?

7.2 How to Find Common Pattern

A Common string is an input string that every automaton accepts. Finding common pattern is a challenge and interesting topic. First, a complete set of common string between two automatons means intersection of them. Just as mentioned previously, automaton intersection is another challenge topic. Second, There are huge amount of authors in the world, it is not possible to ensure that a string can be accepted by every author's automaton. Therefore, we worked on 49 authors and performed a closed world experiment. Instead of finding intersection among automatons, we extract all common strings from 49 authors' sample files and then build an automaton representing common strings characters. Sample Files information refers to Appendix A.

We find such strings by the following steps:

Currently, we collected 49 authors' works and created 49 sample files, eventually built 49 automatons.

- First, choose one sample file and one automaton to obtain the first accepted strings in a text file.
- Second, use the accepted strings as input sample file and choose next automaton in the list to obtain another text file of accepted strings.
- Then, continue the second step until all the automatons are tested through.

- Last, the last accepted strings could be accepted by all 49 automatons.

However, these strings are not a complete set of Common Strings among these 49 authors' sample files. Therefore, we repeat the above four steps for 49 times by picking one sample file at step one at each time. As a result, there are 49 files generated as common string files. We combine these 49 common string files into a single file and then built an automaton from it.

```
Algorithm finding common strings

Input:
        f_in: 49 sample files from 49 authors

        a_in: 49 automatons represents 49 authors

Output:

        a_out: an automaton represents Common Pattern

Begin

        For each sample file in f_in Do

                Input_samplefile = sample file

                For each automaton in a_in Do

                        Write accepted string into file: accepted_file

                        Input_samplefile = accepted_file

                End For

        End For

        Combine 49 final accepted_files into a single file: cmnptn_file

        Build an automaton with cmnptn_file

End algorithm
```

Figure 12: Algorithm finding common pattern

7.3 Implementation prepare

The common pattern testing is a tedious and time-consuming work. Therefore, We need batch scripting for this purpose. Cygwin is installed to run bash script files in our project. The batch files are designed to process files based on directory setting.

The 1st batch file "step01_createsamplefiles.sh" is to create sample files from text files, i.e. convert all text files in a directory into sample files and save them in a specified directory.

The 2nd batch file "step02_buildautomaton.sh" is to build automatons from sample files in a directory and save the results (automatons) into a specified directory.

The 3rd batch file "step03_findcmnptn.sh" is to find common strings which can be accepted by all the automatons in a specified directory.

Besides the above 3 batch files, there are several more batch files created for testing. However, batch files are not critical to our test results, therefore, we don't give detail for them here.

7.4 Finding Common Pattern

7.4.1 Notions used in Common Pattern Testing

- In-world sample files
  49 sample files used to find Common Pattern
- In-world automatons
  49 automatons built from 49 in-world sample files
- Out-world sample files
  Sample files are not from 49 in-world authors. 25 sample files are used for verify common pattern test case 03.
- "pure" sample files
  Sample files after filter common strings
- "pure" automatons files
  Automatons built from "pure" sample files

7.4.2 Data Analysis of Common Pattern

The following table shows the result of finding common strings by 49 authors' writings:

49 in-world automaton built at *alpha=0.8*

Total strings in 49 sample files	629012
Total strings in common string sample file	355165
Unique strings in common string sample file	26451
Common string percentage	355165 ÷ 629012 = 56%

Table 7: Common String Sample File

alpha defines the proximity range when merging nodes. Larger *alpha* brings smaller set of common strings and vice versa. The following is part of common string sample file sorted by occurrence frequency:

```
Frequency       Common String
----------------------------
64931           - //empty string
20463           3
10961           2
 8149           1
 6727           23
 6446           13
 5933           33
 5106           32
 4581           323
 4501           0
…… // skip most of data for brevity

    1           00011323
    1           000103223
    1           000100332322323223
    1           0000133
    1           000013
```

The above data shows the frequency of common strings in 49 in-world sample files. There are 64931 empty strings in all 49 in-world sample files. Usually, shorter strings have higher frequency. Some strings, such as 00011323 only appear once and can be accepted by 49 automatons. The frequency of common strings is a key factor during merge process.

Then we build a Common Pattern Automaton from Common String sample file (total string = 355165) with 0.8 alpha setting. The following table shows the common string counts in 49 in-world sample files by filtering with Common Pattern Automaton.

49 in-world automaton built at *alpha=0.8,* common pattern automaton built at *alpha=0.8*	
Total strings in 49 sample files	629012
Total strings accepted by Common Pattern Automaton	508281
Common pattern percentage	508281 ÷ 629012 = 81%

Table 8: common pattern automaton analysis

From above table, we can tell Common Pattern Automaton captures "patterns" of common string and therefore accepts extra strings besides common string sample files. It is just like from common string 12 and 1212, Common Pattern Automaton forms a pattern to accepts 121212 and 12121212, etc.

7.4.3 Common Pattern Verification Test

In order to verify accuracy of our Common Pattern Automaton, we defined 3 test cases. Test results are included in Appendix C.

Verification Test Case 1:

Purpose: to verify that our finding common pattern method obeys the closure rules of Finite Automata. That is, Common Pattern Automaton is able to tell all common strings from 49 in-world sample files.

Range: in-world sample files, in-world automaton

Test Scenario: input 49 in-world "pure" sample files, i.e. rejected strings from Common Pattern Automaton, into 49 in-world automatons to find Common String.

Expected Result: 0 Common String found

Actual Result: as expected.

Verification Test Case 2:

Purpose: To verify if Common Pattern Automaton only accepts common strings. That is, if there are any strings accepted by Common Pattern Automaton but rejected by one or more of 49 in-world automatons.

Range: in-world sample files, in-world automaton

Test Scenario: feed strings accepted by Common Pattern Automaton, i.e. common strings, into each in-world automaton to check if acceptance percentage is 100%.

Ideal Result: every automaton accepts 100% common strings from Common Pattern Automaton.

Expected Result: close to Ideal Result.

Actual Result: average 96.69%, as expected, it is close to Ideal result

Actual result doesn't meet Ideal result because Common Pattern Automaton expands acceptance capability from merge process. Increase α may improve test result, but on the other hand, Common Pattern Automaton may have less capability to cover complete common strings. In next Test Case, we are going to test Common Pattern Automaton's "complete" attribute, that is, if it can cover all common strings among in-world automatons.

Hence, this 3.8 error range can be corrected during acceptance test. If an automaton does not accept a common string, then don't count it as common string in acceptance test.

Verification Test Case 3:

Purpose: to verify that Common Pattern Automaton's completeness. That is, Common Pattern Automaton can tell all common strings of 49 in-world automatons. Out-world sample files are used for this test purpose.

Range: 25 out-world sample files, 49 in-world automatons

Test Scenario: feed strings rejected by Common Pattern Automaton, into 49 in-world automatons to find common string.

Ideal Result: 0 common strings found.

Expected Result: close to Ideal Result.

Actual Result: less than 0.5%, as expected, it is close to Ideal result.

The Test Result doesn't match Ideal Result because the procedure of finding common pattern is based on a finite set of strings from in-world sample files. Using as many as possible sample files as input to find Common Pattern Automaton, instead 49 in-world sample files only, will improve the test Result. Or

using even small α setting to expand Common Pattern Automaton's acceptance ability, however it will result in worse test result in Test Case 2.

7.5 Results Comparison Before and After Filtering Common String

7.5.1 Acceptance Percentage Calculation

Since common strings are the strings that can be accepted by any automaton, filtering common strings means getting less accepted strings. Therefore, the acceptance percentage will be lowered down. The following equation shows the acceptance percentage before filtering common strings:

$$\textit{Acceptance Percentage} = \frac{AcceptedStrings}{TotalStrings}$$

In our application, filtering common strings is to remove *Common Strings* from *Accepted Strings.* The following shows the equation after filtering common strings:

$$\textit{Acceptance Percentage} = \frac{AcceptedStrings - CommonStrings}{TotalStrings}$$

7.5.2 1st set of testing result

We used 49 in-world automatons and their out-world sample files which didn't contribute to automaton built. We completely tested 45 authors' out-world sample files and the complete results are put in Appendix D. Note, we couldn't find out-world sample files for author 13,15,30 and 49.

Here two sample files are shown: One is from the author of 23rd in-world authors and another is from the author of 9th in-world authors.

We use ratio to indicate the improvement and higher ratios are expected after removing common strings.

$$\textit{Ratio} = \frac{AgainstSameAuthorAutomatonAcceptace\%}{AgainstDifferentAuthorAutomatonAcceptace\%}$$

Since we expect higher acceptance percentage for matching an author's sample file with his own automaton than others' automatons. Higher ratio indicates better discerning test result.

We feed a sample file into 2 automatons; one represents its author's writing characteristics, the other represents a different author's writing characteristics. Note, the sample file didn't contribute to its author's automaton built; otherwise, acceptance percentage will be 100% for same author's test.

From the view of simple math, give two numbers, *A* and *B*, *A* is greater than *B,* and *C* is less than *B,* then we can say *(A-C)/(B-C)* is greater than *A/B.* For example, A = 95, B = 85, C=70, 95/85 = 1.1176, (95-70)/(85-70) = 1.6667, 1.6667 is greater than 1.1176

Therefore, once we take away common strings from sample file, same author's acceptance percentage (*A)* and different author's percentage (*B)* drop a same figure, i.e. drop the percentage of common strings(*C)* in the input sample file, we will get the result of *(A-C)/(B-C)* > *A/B.* However, if *A* is NOT greater than *B*, we couldn't get a better ratio, that is, better discerning ability test results.

We have done our best in testing, but results are not sharp. In section of conclusion, we have speculated some reasons for future students to explore.

First, we list some positive results, i.e., same author's acceptance percentage (*A)* is greater than different author's percentage (*B).* These results were tested with automatons of 5th, 6th, 42nd and 44th author.

The following table use sample file Lost Illusions, which is from the 23rd author. Note: Lost Illusions was not used to build its author's automaton.

Sample File: Lost Illusions by 23rd author

- The sample file "Lost Illusions" Include *77.35%* common strings

Automaton	Before remove cmn string, accepts	Ratio (before remove cmn str)	After remove 77.35% cmn string, accepts	Ratio (after remove cmn str)
SameAuthor (23rd Author)	96.61%	96.61÷ 85.41 = 1.131	19.26%	19.26 ÷ 8.06 = 2.39
Different Author (5th Author)	85.41%		8.06%	
SameAuthor (23rd Author)	96.61%	1.107	19.26%	1.949
Different Author (6th Author)	87.24%		9.88%	

SameAuthor (23rd Author)	96.61%	1.131	19.26%	2.8
Different Author (42nd Author)	84.23%		6.88%	
SameAuthor (23rd Author)	96.61%	1.116	19.26%	2.09
Different Author (44th Author)	86.58%		9.23%	

Table 9: 1st set result of discerning ability test (1)

The following table use sample file Her Prairie Knight that is from the 9th author. Note: Her Prairie Knight was not used to build its author's automaton.

Sample File: Her Prairie Knight by 9th author • The sample file "Lost Illusions" Include *85.73%* common strings				
Automaton	Before remove cmn string, accepts		After remove 85.73% cmn string, accepts	
SameAuthor (23rd Author)	97.36%	1.055	11.63%	1.77
Different Author (5th Author)	92.31%		6.57%	
SameAuthor (23rd Author)	97.36%	1.057	11.63%	1.82
Different Author (6th Author)	92.13%		6.39%	
SameAuthor (23rd Author)	97.36%	1.128	11.63%	1.84
Different Author (42nd Author)	86.34%		6.32%	
SameAuthor (23rd Author)	97.36%	1.093	11.63%	1.669
Different Author (44th Author)	89.06%		6.97%	
SameAuthor (23rd Author)	97.36%	1.001	11.63%	1.01
Different Author (15th Author)	97.22%		11.48%	
SameAuthor (23rd Author)	97.36%	1.002	11.63%	1.016
Different Author (43rd Author)	97.18%		11.45%	

Table 10: 1st set result of discerning ability test (2)

From above results, we can see ratio gets higher after removing common strings from input sample file, that is, better results. Note sample files has a large amount of common strings because alpha setting 0.8 is used to find and build common pattern automaton.

Then, some negative results are list in the following table.

The following table uses sample file Her Prairie Knight that is from the 9th author. Note: Her Prairie Knight was not used to build its author's automaton.

Sample File: Her Prairie Knight by 9th author • The sample file "Lost Illusions" Include *85.73%* common strings				
Automaton	Before remove cmn string, accepts		After remove 85.73% cmn string, accepts	
SameAuthor (23rd Author)	97.36%	0.996	11.63%	0.97
Different Author (20th Author)	97.72%		11.99%	
SameAuthor (23rd Author)	97.36%	0.998	11.63%	0.985
Different Author (45th Author)	97.54%		11.81%	

Table 11: 1st set result of discerning ability test (3)

From above result, we see ratio drops. The root cause is the incorrect results from Authorship Learning System. Incorrect results won't be able to be improved by removing common strings. In a word, common pattern automaton shows its power only when automaton gives accurate acceptance percentage. Note, results being highlighted in pink are incorrect results.

For complete set of test data, please refer to Appendix D.

In the section of Conclusion, we will discuss the possible reason of inaccurate and even incorrect acceptance percentage.

7.5.3 2nd set of testing result

We tested another set of files because, first, alpha 1.5 gives more accurate result according to table 4; and then, famous writings could be more laudable.

We pick eight classic novels from eight different authors to build 8 automatons and one common pattern automaton for these 8 authors:

1) 100 years of solitude by *Gabriel José de la Concordia García Márquez,* translated by *Gregory Rabassa*
2) Gone with the wind (part 3 and 4) by *Margaret Munnerlyn Mitchell*
3) Harry Potter (book 5) by *Joanne "Jo" Rowling*
4) Jane Eyre by *Charlotte Brontë*
5) Pride and Prejudice by *Jane Austen*
6) The Thorn Bird by *Colleen McCullough*
7) Uncle Tom's Cabin by *Harriet Beecher Stowe*
8) War and Peace (book6, 7,8) by *Leo Nikolayevich Tolstoy*

And 2 sample files are used to test: one is Harry Potter (Book 3) by *Joanne "Jo" Rowling*, another is Anna Karenina by *Leo Nikolayevich Tolstoy.*

We list results in ratio from high to low in a table.

The following table use sample file Harry Potter (Book 3) that is written by *Joanne "Jo" Rowling*. Note: Harry Potter (Book 3) was not used to build its author's automaton.

Sample File: Harry Potter (Book 3) by *Joanne "Jo" Rowling*. • The sample file "Lost Illusions" Include *58.61%* common strings				
Automaton	Before remove cmn string, accepts		After remove 58.61% cmn string, accepts	
Same Author (built from Harry Potter book 5)	70.95%	1.139	12.34%	3.344
Different Author (built from 100 years of solitude)	62.29%		3.69%	
Same Author (built from Harry Potter book 5)	70.95%	1.061	12.34%	1.499
Different Author(built from Jane Eyre)	66.84%		8.23%	
Same Author (built from Harry Potter book 5)	70.95%	1.042	12.34%	1.302

Different Author(built from Uncle Sam's Cabin)	68.09%		9.48%	
Same Author (built from Harry Potter book 5)	70.95%	1.029	12.34%	1.195
Different Author (built from War and Peace book 6,7,8)	68.94%		10.33%	
Same Author (built from Harry Potter book 5)	70.95%	1.006	12.34%	1.037
Different Author(built from The Thorn Bird)	70.5%		11.9%	
Same Author (built from Harry Potter book 5)	70.95%	1.004	12.34%	1.024
Different Author (built from Pride and Prejudice)	70.66%		12.05%	
Same Author (built from Harry Potter book 5)	70.95%	1	12.34%	1
Different Author (built from Pride and Prejudice)	70.95%		12.34%	

Table 12: 2nd set result of discerning ability test (1)

The following table use sample file Anna Karenina that is written by *Leo Nikolayevich Tolstoy*. Note: Anna Karenina was not used to build its author's automaton.

Sample File: Anna Karenina by *Leo Nikolayevich Tolstoy*. • The sample file "Lost Illusions" Include *38.72%* common strings				
Automaton	Before remove cmn string, accepts		After remove 38.72% cmn string, accepts	
Same Author (built from War and Peace book 6,7,8)	50.07%	1.172	11.35%	2.823

Different Author (built from 100 years of solitude)	42.74%		4.02%	
Same Author (built from War and Peace book 6,7,8)	50.07%	1.048	11.35%	1.257
Different Author (built from Jane Eyre)	47.76%		9.03%	
Same Author (built from War and Peace book 6,7,8)	50.07%	1.023	11.35%	1.112
Different Author (built from Uncle Sam's Cabin)	48.93%		10.21%	
Same Author (built from War and Peace book 6,7,8)	50.07%	0.974	11.35%	0.894
Different Author (built from The Thorn Bird)	51.42%		12.69%	
Same Author (built from War and Peace book 6,7,8)	50.07%	0.971	11.35%	0.882
Different Author (built from Harry Potter book 5)	51.59%		12.87%	
Same Author (built from War and Peace book 6,7,8)	50.07%	0.971	11.35%	0.882
Different Author (built from Pride and Prejudice)	51.59%		12.87%	
Same Author (built from War and Peace book 6,7,8)	50.07%	**0.967**	11.35%	**0.868**
Different Author (built from Gone with the Wind, part 3,4)	51.8%		13.08%	

Table 13: 2nd set result of discerning ability test (2)

Above results emphasis the conclusion we drew from the 1st set of test result, common pattern automaton boosts ratio, i.e, gives better discerning ability test result, when author's automaton is built well and could grasp author's writing characteristics well.

Common pattern automaton's test is successful when automaton is successful. Therefore, future works should focus on how to construct more accurate automatons. In our project, we followed ALERGIA algorithm strictly in merge process, and made small modifications in function words categorization. As a result, we could see improvements comparing to previous implementation [5], please refer to data in Table 4.

Function words are fundamental construction of sample strings and hence automaton built. Classifying 320 function words into only 5 categories might not accurate enough to capture author's characteristics.

8. Conclusion

In this project, we worked on two major tasks: one is ALERGIA merge logic implementation; the other is Common Pattern Automaton construction.

Prious merge implementation in [5] has two weak points, one is no updating of nodes information after each merge, and another is that merges only happen on separate branches, i.e., without merges between branches. In this project, we corrected these two weak points and hence followed merge logic in [2].

We verified our implementation with three experiments: first, compared merge steps to the example in [2], the verification was successful and our implementation gave exact same three merge steps and final automaton as [2]; Second, verified 100% self acceptance at different alpha settings (0.1 ~ 2.0); Third, compared discerning ability test results to previous implementation in [5], and our implementation gave shaper results.

With our through testing, we believe our merge implementation is successful.

We have done our best in testing, but some discerning ability test results are not sharp. We have speculated some reasons for future students to explore.

First, function words categorization might be too coarse and not accurate enough. Writing characteristics might be lost when classifying 320 function words into only 5 categories. Moreover, some function words have dual meanings and putting these words into correct categories is critical, result in 6.2 indicates so.

Then, using a sentence as a unit is good at capturing sentence pattern, but we might lose larger patterns, such as paragraph pattern.

Last, though this is not our original goal, we speculate that an author has ability to learn and possibilities to be learned by others; it is a great challenge to discern subtle difference between two authors with similar writing style.

Besides merge implementation, we constructed common pattern automaton and verified it with three test cases. Some discerning ability test results are improved after removing common strings from sample files, but some are not. Test results indicate common pattern automaton is successful only when automaton construction is successful.

References

[1] Tsau Young Lin, Shangxuan Zhang: *An Automata Based Authorship Identification System*. PAKDD Workshops (2008) 134-142

[2] Rafael C. Carrasco, José Oncina: *Learning Stochastic Regular Grammars by Means of a State Merging Method.* International Colloquium on Grammatical Inference – ICGI (1994) 139-152

[3] Klammer , Thomas, Muriel R. Schulz and Angela Della Volpe: *Analyzing English Grammar (6th ed)*.Longma (2009)

[4] I. S. P. Nation: *Learning vocabulary in another language.*(2001) 430-431

[5] Shangxuan Zhang: *An automata based authorship identification system 2008*

[6] J.E.Hopcroft, R.Motwani and J.D.Ullman: *Introduction to Automata Theory, Language, and Computation.* Addison Wesley (2001).

[7] P.Baliga and T.Y.Lin: *Kolmogorov Complexity Based Automata Modeling for Intrusion Detection.* Proceeding of the 2005 IEEE International Conference on Granular Computing, " July 25-27, Beijing, China (2005) 387-392

[8] T. Y. Lin: *Patterns in Numerical Data: Practical Approximations to Kolmogorov Complexity.* RSFDGrC 1999: 509-513

[9] M.Young-Lai and F.Tompa: *Stochastic Grammatical Inference of Text Database Structure.* Machine Learning (2000) 111-137.

[10] Akram, Hasan Ibne, Colin de la Higuera, Huang Xiao, and Claudia Eckert: *Grammatical Inference Algorithms in MATLAB*. *Proceedings of the 10th International Colloquium on Grammatical Inference*, Valencia, Spain, Springer-Verlag (2010)

[11] Martin L. Puterman: *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons (1994)

[12] Mosteller, F., and D.L. Wallace: *Inference and Disputed Authorship: The Federalist. Reading*. Reading, Mass: Addison Wesley (1964)

Appendix A: E-books used in project

In-world files:

We collected e-books from a free e-book web site: http://www.ebooktakeaway.com/author. We downloaded 48 authors' works and together with books "starwar[Oliver Simon]" to build 49 automatons and then obtained an automaton representing their common strings.

In order to achieve best result at reasonable process speed, we combined an author's several books into a single text file whose size is around 1250KB. Theoretically, more writings are used in automaton building process, more accurate automaton are obtained. In practice, the process speed is also a concern. Hence, some writings need to be saved for writing testing of same authors' works.

The following 49 authors' works are used in our common pattern experiment:

```
Note: this doc shows File name and indexes (order is as in bash
command "ls -1" output)

1) Algernon Blackwood (1869-1951)
2) Ambrose Bierce (1842-1914)
3) Amelia Edith Huddleston Barr (1831-1919)
4) Andy Adams(1859-1935)
5) Annie Wood Besant (1847-1933)
6) Aristotle(384BC-322BC)
7) Arthur Christopher Benson (1862-1925)
8) Arthur Scott Bailey (1877-1949)
9) B. M. Bower Bertha Muzzy Sinclair (1871-1940)
10)   Charles Brockden Brown (1771-1810)
11)   Charlotte Bronte (1816-1855)
12)   Charlotte M. (Charlotte Monica) Brame (1836-1884)
13)   E. A. Wallis Budge (1857-1934)
14)   Edgar Rice Burroughs (1875-1950)
15)   Elizabeth von Arnim(1866-1941)
16)   Frances Hodgson Burnett (1849-1924)
17)   George Henry Borrow (1803-1881)
18)   Gertrude Franklin Horn Atherton(1857-1948)
19)   Grant Allen(1848-1899)
20)   Hans Christian Andersen(1805-1875)
21)   Harold Bindloss (1866-1945)
22)   Hilaire Belloc (1870-1953)
23)   Honore de Balzac (1799-1850)
```

```
24)   Horatio Alger(1832-1899)
25)   Irene Temple Bailey (c1885-1953)
26)   Irving Bacheller (1859-1950)
27)   Isabella L. (Isabella Lucy) Bird (1831-1904)
28)   J. M. James Matthew) Barrie (1860-1937)
29)   Jacob Abbott(1803-1879)
30)   James Baldwin (1841-1925)
31)   Jane Austen(1775-1817)
32)   John Buchan (1875-1940)
33)   John Burroughs (1837-1921)
34)   John Kendrick Bangs (1862-1922)
35)   Kate Langley Bosher (1865-1932)
36)   Louisa May Alcott(1832-1888)
37)   M. E. (Mary Elizabeth) Braddon (1835-1915)
38)   Max Beerbohm (1872-1956)
39)   Max Brand Frederick Schiller Faust (1892-1944)
40)   Ralph Henry Barbour (1870-1944)
41)   Robert Barr (1850-1912)
42)   Robert Browning (1812-1889)
43)   Robert Hugh Benson (1871-1914)
44)   Samuel Butler (1835-1902)
45)   Thomas Bailey Aldrich (1836-1907)
46)   Thornton W. (Thornton Waldo) Burgess (1874-1965)
47)   Timothy Shay Arthur (1809-1885)
48)   Victor Appleton (pseud)
49)   starwar[Oliver Simon]
```

out-world files:

We collect 25 files that are not from the in-world 49 authors to verify Common Pattern Automaton's ability to accept "complete" common strings.

```
Note: this doc shows File name and indexes (order is as in bash
command "ls -1" output)

1) MobyDickorthewhale[HermanMelville]
2) MrMidshipmanEasy[FrederickMarryat]
3) NewtonForster[FrederickMarryat]
4) Omoo[HermanMelville]
5) OntheSpanishMain[JohnMasefield]
6) ParadiseLost[JohnMilton]
7) ParadiseRegained[JohnMilton]
8) PoeticalWorks[JohnMilton]
9) PoorJack[FrederickMarryat]
10) RedburnHisFirstVoyage[HermanMelville]
11) Snarleyyow[FrederickMarryat]
12) Stickeen[JohnMuir]
13) THECASE-BOOKOFSHERLOCKHOLMES
14) THEMEMOIRSOFSHERLOCKHOLMES
15) THERETURNOFSHERLOCKHOLMES
```

16) TheAdventuresofSherlockHolmesbySirArthurConanDoyle
17) TheChildrenoftheNewForest[FrederickMarryat]
18) TheCourageoftheCommonplace[MaryRaymondShipmanAndrews]
19) TheFlamingoFeather[KirkMunroe]
20) TheLittleSavage[FrederickMarryat]
21) TheMission[FrederickMarryat]
22) TheOldFrontLine[JohnMasefield]
23) ThePachaofManyTales[FrederickMarryat]
24) ThePiazzaTales[HermanMelville]
25) WhiteJacketortheWorldonaMan-of-War[HermanMelville]

Appendix B: Function Words used in project

Function Words are also known as Stop Words; in our program we use the name "StopWords". StopWords are defined in a constant class in our implementation. The details are as following:

```
// constant class to store STOPWORDS
final class StopWords{
	final static int numCat = 5; // how many category/transition symbols, now
0~4
	//adverb
	final static String[] theArray00 =
{"absolutely","again","ago","almost","alone","already","also","always","anywher
e","away","back","barely","carefully","downtown","else","even","ever","everywhe
re", "far","fast", "frequently","hard","hardly","hence","here","hither","home",
"how","however","immediately","lately","later","mostly","near","nearby","nearly
","never","not","now",
"nowhere","occasionally","often","only","out","pretty","quickly",
"quite","rarely","rather","really","recently","seldom","slowly","sometimes","so
mewhere","soon","still","then","thence","there","therefore","thither","thus","t
oday","together","tomorrow","tonight","too","underneath","usually","very","well
","when","whence","where","whither","why","yes","yesterday","yet"};
	final static List<String> theList00 = Arrays.asList(theArray00);
	final static Set<String> category00 = new HashSet<String>(theList00);
	//auxiliary verb
	final static String[] theArray01 =
{"'d","'ll","'s","am","ain't","are","aren't","be","been","being","can","can't",
"could","couldn't","did","didn't","do","does","doesn't","doing","done","don't",
"get","gets","getting","got","had","hadn't","has","hasn't","have","haven't","ha
ving","he'd","he'll","he's",
"i'd","i'll","i'm","is","i've","isn't","it's","may","mayn't","might","must","mu
stn't","ought","oughtn't","'re","shall","shan't","she'd","she'll","she's","shou
ld","shouldn't","that's","they'd","they'll","they're","was","wasn't","we'd","we
'll","were","we're","weren't","we've","will","won't","would","wouldn't","you'd"
,"you'll","you're","you've"};
	final static List<String> theList01 = Arrays.asList(theArray01);
	final static Set<String> category01 = new HashSet<String>(theList01);
	//preposition and conjunction
	final static String[] theArray02 =
{"aboard","about","above","across","after","against","along","alongside","altho
ugh","amid", "amidst","among","amongst","and","around","as","aside",
"astride","at","before","behind","below","beneath","beside","besides","between"
,"beyond","by","concerning","despite","down","during","except","excluding",
"following","for","from","given","if","in","including", "inside","into","like",
"minus","near",
"next","nor","of","off","on","onto","or","out","outside","over","past","per",
"regarding","round","since","so","than","though","through","till","to","toward"
,"towards","under","underneath","unless","unlike","until","up","upon",
"versus","via","whereas","while","with","within","without"};
	final static List<String> theList02 = Arrays.asList(theArray02);
	final static Set<String> category02 = new HashSet<String>(theList02);
	//pronoun
	final static String[] theArray03 =
{"a","all","an","and","another","any","anybody","anyone","anything","because","
```

```
both","but","each","either","enough","every","everybody","everyone","everything
","few","fewer","he","her","hers","herself","him","himself","his","i","it","its
","itself","less","little","many","me","mine","more","most","much","my","myself
","neither","no","nobody","none","nor","nothing","or","other","others","our","o
urs","ourselves","provided","several","she","so","some","somebody","someone","s
omething","such","that","the","their","theirs","them","themselves","these","the
y","this","those","us","we","what","whatever","whenever","whether","which","whi
chever","while","who","whoever","whom","whose","yet","you","your","yours","your
self","yourselves"};
	final static List<String> theList03 = Arrays.asList(theArray03);
	final static Set<String> category03 = new HashSet<String>(theList03);
	//number
	final static String[] theArray04 =
{"billion","billionth","eight","eighteen","eighteenth","eighth","eightieth","ei
ghty","eleven","eleventh","fifteen","fifteenth","fifth","fiftieth","fifty","fir
st","five","fortieth","forty","four","fourteen","fourteenth","fourth","hundred"
,"hundredth","last","million","millionth","next","nine","nineteen","nineteenth"
,"ninetieth",
"ninety","ninth","once","one","second","seven","seventeen","seventeenth","seven
th","seventieth","seventy","six","sixteen","sixteenth","sixth","sixtieth","sixt
y","ten","tenth","third","thirteen","thirteenth","thirtieth","thirty","thousand
","thousandth","three","thrice","twelfth","twelve","twentieth","twenty","twice"
,"two","zero"};
	final static List<String> theList04 = Arrays.asList(theArray04);
	final static Set<String> category04 = new HashSet<String>(theList04);

		static int getStopWords(String word)
		{
			if(category00.contains(word))
				return 0;
			if(category01.contains(word))
				return 1;
			if(category02.contains(word))
				return 2;
			if(category03.contains(word))
				return 3;
			if(category04.contains(word))
				return 4;
			return -1; // not a stop word
		}
	}
```

Appendix C: Test Result of Common Pattern Verification

This appendix is a complete set of test data for section 7.4.3.

Test Case 1: Common Pattern Automaton Verification Test

Test Sample Files: 49 in-world sample files, which are used to build Common Pattern Automaton.

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: to check if our finding common pattern method obeys the closure rules of Finite Automata

Test Scenario: feed rejected strings from Common Pattern Automaton, into 49 automatons in alphabet ascending order to see if there are any common strings found. For example, common pattern automaton rejects 2255 strings for sample1, feed 2255 into 1st automaton, accepts 2255 strings, then feed 2255 into 2nd automaton, accepts 1805, then feed 1805 into 3rd automaton, and so on, until 49th automaton. This is the procedure to find common strings among 49 automatons.

Expected Result: No common Strings found

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts.*
2) *Each column starts from the number of strings rejected by Common Pattern Automaton;*
3) *Feed the output from Common Pattern Automaton, i.e., rejected strings from Common Pattern Automaton, into 1st author's automaton*
4) *Feed the output from 1st author's automaton, i.e., accepted strings from 1st author's automaton, into next author's automaton, i.e., 2nd author's automaton.*
5) *Keep on feeding output, accepted strings, into next author's automaton until last author in the list, i.e, 49th author's automaton.*
6) *We care about how many strings are eventually accepted by 49th author's automaton. Expected result is 0.*

	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6	Sample 7	Sample 8	Sample 9	Sample 10	Sample 11	Sample 12	Sample 13	Sample 14	Sample 15	Sample 16
Cmn ptn automaton rejects	2255	2481	2540	3955	2576	2631	2788	2227	2797	2574	2185	2230	2105	2252	2999	2115
1 accepts	2255	1896	2106	2734	1857	1605	2110	1931	2255	2219	1647	1887	1613	1761	2133	1640
2 accepts	1805	1896	1841	2149	1466	1158	1726	1745	1928	1979	1368	1682	1354	1492	1693	1399
3 accepts	1583	1611	1841	1813	1248	950	1512	1619	1741	1833	1214	1554	1191	1324	1470	1247
4 accepts	1432	1450	1667	1813	1094	800	1349	1504	1576	1710	1112	1434	1056	1207	1293	1123
5 accepts	1097	1094	1310	1355	1094	604	1047	1226	1253	1388	864	1121	830	944	1001	912
6 accepts	862	882	1035	1143	905	604	875	965	1023	1160	695	928	728	769	781	754
7 accepts	794	808	959	1037	808	545	875	921	960	1106	639	886	678	711	730	703
8 accepts	645	647	782	797	574	417	696	921	788	879	522	741	535	574	592	563
9 accepts	626	606	756	746	535	393	666	877	788	852	497	719	513	551	564	541
10accepts	599	575	714	700	491	362	624	839	743	852	467	691	482	517	535	510
11accepts	572	543	685	669	463	334	603	799	713	825	467	661	460	501	508	476
12accepts	553	517	662	638	442	316	575	766	691	799	442	661	444	483	491	454
13accepts	503	469	610	562	404	277	515	713	633	728	396	609	444	449	457	405
14accepts	484	438	574	521	367	251	487	676	599	691	378	576	422	449	434	386
15accepts	459	417	559	501	345	232	460	651	577	669	363	545	401	428	434	367
16accepts	434	394	531	475	318	213	431	618	547	637	352	517	385	405	406	367
17accepts	406	374	506	451	302	196	400	584	521	599	332	498	347	386	389	353
18accepts	386	352	488	422	279	179	373	558	491	575	314	475	327	360	367	340
19accepts	374	343	477	401	268	167	366	545	476	555	308	465	317	347	353	331
20accepts	364	335	464	390	259	158	357	536	463	538	299	447	305	337	339	322
21accepts	346	311	450	369	242	148	338	521	437	505	285	429	287	312	325	308
22accepts	317	285	419	340	220	138	316	483	399	462	265	395	266	287	303	287
23accepts	305	273	397	329	214	131	306	466	384	444	257	378	253	279	294	273
24accepts	299	266	387	320	206	124	292	454	372	433	248	369	248	271	285	265
25accepts	291	252	374	310	194	117	277	441	358	428	241	366	237	259	273	257
26accepts	283	242	361	296	187	112	266	431	342	411	231	355	227	252	264	251
27accepts	245	216	303	265	169	106	236	361	298	372	200	305	199	221	223	220
28accepts	236	208	295	255	161	100	228	352	286	357	192	298	191	212	214	215
29accepts	200	180	251	217	145	85	191	290	236	292	160	271	162	173	178	181
30accepts	193	162	229	207	133	80	182	282	218	277	151	261	156	167	170	170
31accepts	180	148	217	194	119	73	175	271	200	263	138	241	144	155	161	158
32accepts	180	147	215	192	119	72	172	269	196	261	138	238	144	154	161	155
33accepts	175	136	206	185	115	68	165	262	189	251	133	226	138	147	153	151
34accepts	163	122	194	169	108	65	152	241	177	241	127	207	133	139	139	143
35accepts	158	120	191	164	102	63	149	238	166	240	123	203	130	136	134	141
36accepts	144	105	170	149	89	56	135	209	149	219	107	181	113	124	125	126
37accepts	133	94	153	133	80	52	124	192	131	203	97	164	106	112	115	117
38accepts	128	92	150	129	78	52	120	188	127	199	95	160	104	111	112	114
39accepts	126	91	150	124	76	50	119	185	124	195	93	158	102	109	112	112
40accepts	122	86	146	112	69	50	113	178	118	184	89	152	98	103	108	105
41accepts	105	78	126	101	62	48	95	152	105	160	81	133	84	88	95	90
42accepts	79	58	98	78	51	34	68	128	77	121	57	92	67	69	69	65
43accepts	64	46	90	68	43	27	57	114	66	108	50	86	55	56	57	62
44accepts	8	11	17	20	14	5	18	26	9	19	7	15	12	12	5	12
45accepts	7	7	12	16	14	5	14	20	6	17	4	14	10	10	3	7
46accepts	6	4	8	15	11	5	10	18	5	16	2	11	7	6	3	6
47accepts	4	2	5	8	8	4	9	5	4	13	1	7	4	4	0	3
48accepts	2	0	1	3	2	2	3	0	0	0	0	0	1	0	0	0
49accepts	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 14: Test Data of Common Pattern Automaton Verification Test Case 01 (1)

Test Case 1: Common Pattern Automaton Verification Test

Test Sample Files: 49 in-world sample files, which are used to build Common Pattern Automaton.

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: to check if our finding common pattern method obeys the closure rules of Finite Automata

Test Scenario: feed rejected strings from Common Pattern Automaton, into 49 automatons in alphabet ascending order to see if there are any common strings found. For example, common pattern automaton rejects 2583 strings for sample17, feed 2583 into 1st automaton, accepts 1680 strings, then feed 1680 into 2nd automaton, accepts 1282, then feed 1282 into 3rd automaton, and so on, until 49th automaton. This is the procedure to find common strings among 49 automatons.

Expected Result: No common Strings found

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts.*
2) *Each column starts from the number of strings rejected by Common Pattern Automaton;*
3) *Feed the output from Common Pattern Automaton, i.e., rejected strings from Common Pattern Automaton, into 1st author's automaton*
4) *Feed the output from 1st author's automaton, i.e., accepted strings from 1st author's automaton, into next author's automaton, i.e., 2nd author's automaton.*
5) *Keep on feeding output, accepted strings, into next author's automaton until last author in the list, i.e, 49th author's automaton.*
6) *We care how many strings are eventually accepted by 49th author's automaton. Expected result is 0.*

	Sample 17	Sample 18	Sample 19	Sample 20	Sample 21	Sample 22	Sample 23	Sample 24	Sample 25	Sample 26	Sample 27	Sample 28	Sample 29	Sample 30	Sample 31	Sample 32
Cmn ptn automaton rejects	2583	2313	2723	2596	2315	2691	2351	2643	1938	2130	3076	2614	2273	2731	2444	2517
1 accepts	1680	1722	2049	1997	1895	1803	1725	2132	1665	1805	2049	1961	1608	2217	1685	2071
2 accepts	1282	1417	1682	1668	1676	1369	1394	1879	1475	1600	1588	1611	1293	1931	1293	1794
3 accepts	1101	1224	1470	1457	1536	1151	1228	1731	1350	1503	1350	1436	1121	1755	1108	1662
4 accepts	960	1110	1328	1311	1414	1008	1124	1613	1255	1402	1201	1291	964	1612	976	1537
5 accepts	712	871	1037	1026	1128	772	870	1305	991	1128	898	964	706	1244	729	1232
6 accepts	570	739	841	833	909	643	701	1069	804	938	736	768	594	1063	582	1009
7 accepts	520	683	759	779	847	590	657	1006	760	869	684	727	542	981	535	944
8 accepts	412	536	620	620	698	462	527	871	632	726	534	582	425	797	430	784
9 accepts	394	504	588	599	680	432	508	843	613	695	505	555	397	765	413	755
10accepts	371	474	548	565	653	407	475	817	595	663	475	520	368	721	387	716
11accepts	350	450	523	538	627	382	453	789	569	639	446	495	347	690	363	684
12accepts	328	432	499	517	604	364	430	767	556	612	418	476	339	664	353	657
13accepts	294	396	447	462	558	330	389	717	515	577	376	423	301	601	318	625
14accepts	281	368	425	439	520	317	365	688	496	551	351	387	272	560	302	593
15accepts	271	353	406	417	494	302	346	665	477	536	330	366	261	533	285	569
16accepts	257	334	381	396	465	288	329	628	461	515	306	345	239	506	258	533
17accepts	257	325	359	369	448	265	309	593	430	490	289	328	232	469	236	501
18accepts	246	325	345	341	422	246	299	569	415	479	270	304	222	444	217	483
19accepts	235	312	345	329	414	236	286	553	407	468	264	293	213	423	213	469
20accepts	223	295	328	329	404	223	281	539	399	455	256	286	210	414	209	446
21accepts	218	279	309	308	404	209	268	519	388	428	241	277	196	395	204	425
22accepts	195	255	281	286	369	209	249	498	357	397	221	253	174	358	185	396
23accepts	183	246	260	271	352	193	249	473	332	382	206	242	165	347	181	376
24accepts	178	235	252	266	341	189	239	473	321	370	194	227	160	332	176	367
25accepts	172	231	245	257	333	176	232	462	321	365	190	220	160	318	168	348
26accepts	161	225	240	249	324	171	225	454	318	365	185	219	152	313	164	336
27accepts	140	199	208	226	279	152	205	359	264	319	185	189	130	281	141	287
28accepts	138	191	200	220	270	151	199	348	258	314	176	189	125	274	139	280
29accepts	111	167	168	192	239	123	157	288	222	263	149	163	125	224	120	241
30accepts	103	162	158	178	228	111	151	279	214	249	140	154	119	224	115	230
31accepts	96	151	148	166	213	105	140	263	196	241	130	144	114	207	115	220
32accepts	95	149	147	166	210	104	139	263	194	241	129	144	114	204	114	220
33accepts	93	143	142	162	201	101	134	254	187	232	123	140	111	199	111	214
34accepts	88	133	130	148	190	93	125	243	176	211	114	125	101	179	102	204
35accepts	86	130	127	144	188	92	121	237	173	205	106	119	96	177	97	199
36accepts	73	116	110	126	162	86	111	220	147	177	95	109	85	161	92	176
37accepts	65	104	98	118	147	80	109	203	139	166	91	97	81	140	77	158
38accepts	62	99	94	115	143	75	107	198	138	164	88	97	78	137	76	154
39accepts	62	97	94	114	140	73	103	194	136	158	86	94	76	132	75	148
40accepts	60	94	92	111	137	68	100	188	133	155	80	88	73	127	67	144
41accepts	52	81	79	100	123	55	92	164	112	138	69	79	65	110	60	131
42accepts	35	60	61	72	85	39	73	124	88	105	53	56	49	82	43	96
43accepts	28	52	49	62	81	34	65	106	81	100	40	48	46	67	38	83
44accepts	6	16	8	13	22	14	19	18	14	17	19	10	16	15	6	16
45accepts	5	15	7	9	13	11	17	18	11	16	16	7	14	13	5	13
46accepts	5	13	6	7	11	10	12	13	8	13	14	6	10	9	4	12
47accepts	4	10	4	5	7	7	7	8	3	7	10	3	4	5	3	6
48accepts	0	3	2	0	3	1	0	2	0	0	2	0	1	2	0	2
49accepts	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 15: Test Data of Common Pattern Automaton Verification Test Case 01 (2)

Test Case 1: Common Pattern Automaton Verification Test

Test Sample Files: 49 in-world sample files, which are used to build Common Pattern Automaton.

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: to check if our finding common pattern method obeys the closure rules of Finite Automata

Test Scenario: feed rejected strings from Common Pattern Automaton, into 49 automatons in alphabet ascending order to see if there are any common strings found. For example, common pattern automaton rejects 2322 strings for sample33, feed 2322 into 1st automaton, accepts 1782 strings, then feed 1782 into 2nd automaton, accepts 1473, then feed 1473 into 3rd automaton, and so on, until 49th automaton. This is the procedure to find common strings among 49 automatons.

Expected Result: No common Strings found

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts.*
2) *Each column starts from the number of strings rejected by Common Pattern Automaton;*
3) *Feed the output from Common Pattern Automaton, i.e., rejected strings from Common Pattern Automaton, into 1st author's automaton*
4) *Feed the output from 1st author's automaton, i.e., accepted strings from 1st author's automaton, into next author's automaton, i.e., 2nd author's automaton.*
5) *Keep on feeding output, accepted strings, into next author's automaton until last author in the list, i.e, 49th author's automaton.*
6) *We care how many strings are eventually accepted by 49th author's automaton. Expected result is 0.*

	Sample 33	Sample 34	Sample 35	Sample 36	Sample 37	Sample 38	Sample 39	Sample 40	Sample 41	Sample 42	Sample 43	Sample 44	Sample 45	Sample 46	Sample 47	Sample 48	Sample 49
Cmn ptn automaton rejects	2322	2383	2647	2555	2160	2521	2339	2546	2455	1921	2630	2337	2312	2087	2287	1987	2594
1 accepts	1782	1733	2074	1991	1662	2050	1979	1909	1859	1397	1793	1484	1813	1741	1915	1639	2278
2 accepts	1473	1408	1780	1638	1383	1801	1733	1566	1563	1153	1375	1070	1545	1526	1666	1436	2071
3 accepts	1314	1232	1601	1452	1227	1665	1583	1385	1379	1008	1187	872	1368	1400	1542	1315	1950
4 accepts	1198	1101	1435	1280	1104	1529	1475	1289	1251	892	1054	743	1265	1266	1398	1204	1811
5 accepts	953	850	1115	983	845	1236	1180	991	984	672	776	552	985	1019	1142	961	1470
6 accepts	784	703	894	810	689	983	998	774	812	545	636	457	790	825	945	808	1195
7 accepts	730	653	838	749	646	931	942	709	753	508	585	432	724	778	892	765	1127
8 accepts	567	526	696	581	510	766	796	597	619	396	476	329	591	656	748	625	946
9 accepts	545	499	671	545	494	748	766	569	596	377	452	306	570	615	720	595	918
10accepts	512	478	649	513	468	715	724	536	574	357	425	278	537	588	693	569	890
11accepts	493	455	616	490	439	677	690	520	551	338	411	263	526	551	655	543	861
12accepts	467	437	589	464	418	654	655	508	521	323	396	248	500	523	631	525	836
13accepts	425	398	546	424	381	612	605	460	481	292	354	232	452	480	580	481	780
14accepts	393	380	522	386	360	585	572	431	448	274	330	223	425	456	545	456	746
15accepts	376	364	508	371	340	567	552	418	424	263	314	218	416	435	521	442	725
16accepts	351	345	485	345	325	539	528	404	397	253	284	198	395	418	497	416	705
17accepts	331	320	457	321	318	509	494	387	371	244	268	178	379	395	465	392	677
18accepts	312	303	432	298	301	478	467	369	346	230	255	161	363	379	447	372	642
19accepts	305	293	416	287	289	468	450	363	341	224	250	152	358	367	437	360	627
20accepts	293	284	407	277	280	451	444	352	332	222	242	147	342	362	427	355	612
21accepts	275	268	391	257	266	434	428	341	315	207	226	135	325	346	411	340	588
22accepts	245	255	362	225	237	409	401	321	293	190	215	117	301	314	372	322	539
23accepts	235	244	341	216	223	388	377	297	275	187	207	111	290	298	361	306	523
24accepts	227	238	336	207	218	377	366	285	262	183	199	108	280	292	347	294	515
25accepts	220	229	328	198	209	367	356	275	255	175	191	98	265	283	333	288	502
26accepts	216	222	324	189	202	353	350	267	246	172	189	95	254	276	327	286	493
27accepts	193	192	271	166	180	307	304	228	219	158	156	89	219	233	298	237	437
28accepts	184	184	264	161	173	296	297	225	210	155	151	86	211	231	284	233	427
29accepts	148	144	223	140	150	244	253	188	174	128	126	64	173	188	235	200	343
30accepts	141	138	208	133	147	240	240	175	163	128	120	60	162	181	222	189	321
31accepts	132	131	192	129	139	225	222	165	157	122	114	55	149	177	210	177	299
32accepts	132	129	187	128	136	224	219	165	157	121	112	54	147	177	208	177	298
33accepts	132	125	181	120	131	216	214	162	148	120	107	54	139	173	202	173	288
34accepts	123	125	168	113	112	197	200	151	131	117	98	53	127	166	186	158	272
35accepts	120	119	168	110	105	191	194	149	129	114	95	51	126	161	183	156	264
36accepts	105	110	152	110	94	170	168	126	116	97	82	46	102	142	156	144	233
37accepts	94	100	140	99	94	156	158	121	98	85	75	39	91	136	142	135	217
38accepts	93	98	135	95	91	156	152	120	96	81	73	35	89	130	139	133	211
39accepts	87	98	133	94	89	153	152	118	92	76	70	34	88	126	134	131	210
40accepts	82	94	129	83	87	146	152	118	90	73	68	32	88	122	128	127	198
41accepts	73	87	117	76	77	130	140	105	90	65	61	27	79	105	105	116	170
42accepts	54	76	95	51	52	98	115	77	67	65	42	15	60	83	85	87	133
43accepts	43	66	91	41	51	82	110	76	57	52	42	4	50	76	74	75	116
44accepts	8	18	11	11	14	13	21	11	10	12	10	4	12	12	18	14	18
45accepts	6	13	8	10	14	9	17	8	9	10	8	3	12	10	14	11	16
46accepts	6	11	7	8	11	8	13	8	7	8	5	3	8	10	12	8	13
47accepts	3	4	2	5	8	6	10	4	5	4	4	2	7	4	12	2	11
48accepts	0	0	1	2	1	0	1	1	1	0	1	0	0	1	1	2	0
49accepts	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 16: Test Data of Common Pattern Automaton Verification Test Case 01 (3)

Test Case 2: Common Pattern Automaton Verification Test

Test Files: 49 In-world sample files

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: Common Pattern Automaton's accuracy, i.e. if a string is accepted by common pattern automaton, then it is a common string of 49 in-world automatons.

Test Scenario: feed accepted strings from common pattern automaton, into 49 individual automatons to see if any automatons reject strings.

Ideal Result:: no automatons reject common strings from common pattern automaton

Expected Result: close to Ideal Result. Averagely automaton accepts 96.69% common strings from common pattern automaton. In a word, common pattern automaton gives 3.31% non-common strings incorrectly.

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts and acceptance percentage.*
2) *Each column starts from the total strings in a sample file;*
3) *Feed the total strings into Common Pattern Automaton, record the number of strings accepted by Common Pattern Automaton.*
4) *Feed the output from Common Pattern Automaton, i.e., accepted strings from Common Pattern Automaton, into 1st author's automaton, then calculate acceptance percentage. For example: sample 1 has 9425 strings accepted by Common Pattern Automaton, feed 9425 strings into 1st author's automaton, accepts 9425 strings. Then acceptance percentage is 9425/9425 = 100%*
5) *Same scenario as 4), feed the accepted strings from Common Pattern Automaton into 2nd author's automaton and then calculate acceptance percentage.*
6) *For each automaton, we perform the same scenario, i.e,. Feed the accepted strings from Common Pattern Automaton into author's automaton and then calculate acceptance percentage.*
7) *At last, add up all acceptance percentage and divided by "49X49", to get an average number, which is 96.69%.*

	Sample 1		Sample 2		Sample 3		Sample 4		Sample 5		Sample 6	
Total strings	13166		10502		14416		12100		10535		9285	
Cmn ptn automaton accepts	9425	Automaton accepts	7366	Automaton accepts	10889	Automaton accepts	7608	Automaton accepts	6340	Automaton accepts	4509	Automaton accepts
1 accepts	9425	100%	7141	96.95%	10707	98.33%	7383	97.04%	6041	95.28%	4149	92.02%
2 accepts	9236	97.99%	7366	100%	10671	98%	7383	97.04%	6037	95.22%	4116	91.28%
3 accepts	9282	98.48%	7169	97.33%	10889	100%	7447	97.88%	6082	95.93%	4193	92.99%
4 accepts	9212	97.74%	7082	96.14%	10630	97.62%	7608	100%	6019	94.94%	4093	90.77%
5 accepts	8917	94.61%	6687	90.78%	10283	94.43%	6900	90.69%	6340	100%	3786	83.97%
6 accepts	8892	94.34%	6857	93.09%	10306	94.65%	7118	93.56%	5827	91.91%	4509	100%
7 accepts	9226	97.89%	7138	96.9%	10663	97.92%	7378	96.98%	6090	96.06%	4183	92.77%
8 accepts	9099	96.54%	6878	93.37%	10454	96.01%	7123	93.63%	5788	91.29%	3890	86.27%
9 accepts	9311	98.79%	7197	97.71%	10756	98.78%	7464	98.11%	6139	96.83%	4207	93.3%
10accepts	9300	98.67%	7207	97.84%	10754	98.76%	7445	97.86%	6098	96.18%	4189	92.9%
11accepts	9274	98.4%	7129	96.78%	10644	97.75%	7410	97.4%	6068	95.71%	4175	92.59%
12accepts	9300	98.67%	7189	97.6%	10719	98.44%	7446	97.87%	6113	96.42%	4206	93.28%
13accepts	9144	97.02%	6958	94.46%	10540	96.79%	7219	94.89%	5933	93.58%	4050	89.82%
14accepts	9216	97.78%	7077	96.08%	10639	97.7%	7322	96.24%	6009	94.78%	4112	91.2%
15accepts	9262	98.27%	7127	96.76%	10687	98.14%	7383	97.04%	6079	95.88%	4159	92.24%
16accepts	9234	97.97%	7106	96.47%	10666	97.95%	7365	96.81%	6051	95.44%	4154	92.13%
17accepts	9257	98.22%	7152	97.09%	10686	98.14%	7412	97.42%	6082	95.93%	4194	93.01%
18accepts	9239	98.03%	7101	96.4%	10670	97.99%	7364	96.79%	6047	95.38%	4129	91.57%
19accepts	9286	98.53%	7143	96.97%	10677	98.05%	7355	96.67%	6086	95.99%	4201	93.17%
20accepts	9272	98.38%	7155	97.14%	10710	98.36%	7390	97.13%	6106	96.31%	4213	93.44%
21accepts	9233	97.96%	7097	96.35%	10638	97.69%	7358	96.71%	5961	94.02%	4067	90.2%
22accepts	9142	97%	7008	95.14%	10562	97%	7271	95.57%	6010	94.79%	4145	91.93%
23accepts	9286	98.53%	7164	97.26%	10704	98.3%	7426	97.61%	6100	96.21%	4185	92.81%
24accepts	9249	98.13%	7050	95.71%	10675	98.03%	7377	96.96%	5980	94.32%	4025	89.27%
25accepts	9266	98.31%	7130	96.8%	10709	98.35%	7402	97.29%	6025	95.03%	4104	91.02%
26accepts	9269	98.34%	7140	96.93%	10689	98.16%	7401	97.28%	6026	95.05%	4144	91.91%
27accepts	9197	97.58%	7106	96.47%	10609	97.43%	7360	96.74%	6061	95.6%	4175	92.59%
28accepts	9296	98.63%	7153	97.11%	10712	98.37%	7434	97.71%	6121	96.55%	4197	93.08%
29accepts	9055	96.07%	6945	94.28%	10417	95.67%	7166	94.19%	5894	92.97%	4042	89.64%
30accepts	9207	97.69%	7042	95.6%	10624	97.57%	7288	95.79%	5975	94.24%	4024	89.24%
31accepts	9187	97.47%	7096	96.33%	10590	97.25%	7297	95.91%	6007	94.75%	4117	91.31%
32accepts	9299	98.66%	7190	97.61%	10757	98.79%	7477	98.28%	6115	96.45%	4150	92.04%
33accepts	9266	98.31%	7137	96.89%	10646	97.77%	7368	96.85%	6065	95.66%	4148	91.99%
34accepts	9175	97.35%	7017	95.26%	10582	97.18%	7279	95.68%	5943	93.74%	4081	90.51%
35accepts	9278	98.44%	7108	96.5%	10715	98.4%	7361	96.75%	6069	95.73%	4160	92.26%
36accepts	9269	98.34%	7155	97.14%	10668	97.97%	7382	97.03%	6083	95.95%	4200	93.15%
37accepts	9214	97.76%	7078	96.09%	10611	97.45%	7310	96.08%	5972	94.2%	4093	90.77%
38accepts	9282	98.48%	7194	97.66%	10735	98.59%	7421	97.54%	6126	96.62%	4196	93.06%
39accepts	9263	98.28%	7126	96.74%	10728	98.52%	7394	97.19%	6067	95.69%	4163	92.33%
40accepts	9233	97.96%	7021	95.32%	10636	97.68%	7292	95.85%	5886	92.84%	3982	88.31%
41accepts	9258	98.23%	7184	97.53%	10700	98.26%	7417	97.49%	6079	95.88%	4195	93.04%
42accepts	8847	93.87%	6599	89.59%	10216	93.82%	6852	90.06%	5579	88%	3671	81.41%
43accepts	9281	98.47%	7105	96.46%	10712	98.37%	7383	97.04%	6052	95.46%	4186	92.84%
44accepts	8945	94.91%	6806	92.4%	10269	94.31%	7001	92.02%	5798	91.45%	3963	87.89%
45accepts	9290	98.57%	7175	97.41%	10735	98.59%	7443	97.83%	6123	96.58%	4191	92.95%
46accepts	9172	97.32%	7018	95.28%	10614	97.47%	7275	95.62%	5909	93.2%	3990	88.49%
47accepts	9272	98.38%	7110	96.52%	10701	98.27%	7387	97.1%	6030	95.11%	4107	91.08%
48accepts	9244	98.08%	7108	96.5%	10694	98.21%	7371	96.88%	6023	95%	4135	91.71%
49accepts	9302	98.69%	7168	97.31%	10743	98.66%	7401	97.28%	6099	96.2%	4193	92.99%

Table 17: Test Data of Common Pattern Automaton Verification Test Case 02 (1)

Test Case 2: Common Pattern Automaton Verification Test

Test Files: 49 In-world sample files

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: Common Pattern Automaton's accuracy, i.e. if a string is accepted by common pattern automaton, then it is a common string of 49 in-world automatons.

Test Scenario: feed accepted strings from common pattern automaton, into 49 individual automatons to see if any automatons reject strings.

Ideal Result:: no automatons reject common strings from common pattern automaton

Expected Result: close to Ideal Result. Averagely automaton accepts 96.69% common strings from common pattern automaton. In a word, common pattern automaton gives 3.31% non-common strings incorrectly.

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts and acceptance percentage.*
2) *Each column starts from the total strings in a sample file;*
3) *Feed the total strings into Common Pattern Automaton, record the number of strings accepted by Common Pattern Automaton.*
4) *Feed the output from Common Pattern Automaton, i.e., accepted strings from Common Pattern Automaton, into 1st author's automaton, then calculate acceptance percentage. For example: sample 7 has 7622 strings accepted by Common Pattern Automaton, feed 7622 strings into 1st author's automaton, accepts 7262 strings. Then acceptance percentage is 7262/7622 = 95.28%*
5) *Same scenario as 4), feed the accepted strings from Common Pattern Automaton into 2nd author's automaton and then calculate acceptance percentage.*
6) *For each automaton, we perform the same scenario, i.e,. Feed the accepted strings from Common Pattern Automaton into author's automaton and then calculate acceptance percentage.*
7) *At last, add up all acceptance percentage and divided by "49X49", to get an average number, which is 96.69%.*

	Sample 7		Sample 8		Sample 9		Sample 10		Sample 11		Sample 12	
Total strings	10938		21013		14655		15597		11102		13120	
Cmn ptn automaton accepts	7622		15140		10928		12407		7585		10379	
1 accepts	7262	95.28%	15001	99.08%	10714	98.04%	12171	98.1%	7375	97.23%	10197	98.25%
2 accepts	7243	95.03%	14990	99.01%	10686	97.79%	12133	97.79%	7339	96.76%	10160	97.89%
3 accepts	7300	95.78%	15038	99.33%	10750	98.37%	12222	98.51%	7395	97.5%	10220	98.47%
4 accepts	7179	94.19%	14915	98.51%	10635	97.32%	12098	97.51%	7297	96.2%	10134	97.64%
5 accepts	6783	88.99%	14504	95.8%	10231	93.62%	11564	93.21%	6998	92.26%	9734	93.79%
6 accepts	6950	91.18%	14397	95.09%	10242	93.72%	11637	93.79%	7063	93.12%	9715	93.6%
7 accepts	7622	100%	14952	98.76%	10680	97.73%	12116	97.65%	7330	96.64%	10172	98.01%
8 accepts	6949	91.17%	15140	100%	10504	96.12%	11841	95.44%	7137	94.09%	9971	96.07%
9 accepts	7365	96.63%	15006	99.11%	10928	100%	12262	98.83%	7449	98.21%	10241	98.67%
10accepts	7363	96.6%	15034	99.3%	10755	98.42%	12407	100%	7417	97.79%	10237	98.63%
11accepts	7285	95.58%	14909	98.47%	10696	97.88%	12148	97.91%	7585	100%	10162	97.91%
12accepts	7289	95.63%	15016	99.18%	10737	98.25%	12201	98.34%	7366	97.11%	10379	100%
13accepts	6995	91.77%	14839	98.01%	10497	96.06%	11986	96.61%	7191	94.81%	9993	96.28%
14accepts	7180	94.2%	14899	98.41%	10647	97.43%	12094	97.48%	7317	96.47%	10100	97.31%
15accepts	7237	94.95%	14961	98.82%	10684	97.77%	12191	98.26%	7357	96.99%	10178	98.06%
16accepts	7274	95.43%	14958	98.8%	10701	97.92%	12157	97.99%	7355	96.97%	10166	97.95%
17accepts	7291	95.66%	14930	98.61%	10674	97.68%	12144	97.88%	7391	97.44%	10158	97.87%
18accepts	7230	94.86%	14957	98.79%	10681	97.74%	12173	98.11%	7329	96.62%	10160	97.89%
19accepts	7273	95.42%	14985	98.98%	10713	98.03%	12144	97.88%	7391	97.44%	10156	97.85%
20accepts	7300	95.78%	14950	98.75%	10740	98.28%	12204	98.36%	7362	97.06%	10204	98.31%
21accepts	7151	93.82%	14968	98.86%	10649	97.45%	12115	97.65%	7320	96.51%	10141	97.71%
22accepts	7142	93.7%	14793	97.71%	10531	96.37%	11968	96.46%	7249	95.57%	10019	96.53%
23accepts	7310	95.91%	14978	98.93%	10720	98.1%	12221	98.5%	7367	97.13%	10171	98%
24accepts	7171	94.08%	15003	99.1%	10645	97.41%	12150	97.93%	7332	96.66%	10160	97.89%
25accepts	7279	95.5%	15024	99.23%	10704	97.95%	12217	98.47%	7345	96.84%	10186	98.14%
26accepts	7243	95.03%	14993	99.03%	10694	97.86%	12179	98.16%	7354	96.95%	10195	98.23%
27accepts	7219	94.71%	14811	97.83%	10635	97.32%	12093	97.47%	7339	96.76%	10105	97.36%
28accepts	7349	96.42%	14997	99.06%	10718	98.08%	12202	98.35%	7390	97.43%	10180	98.08%
29accepts	7004	91.89%	14636	96.67%	10457	95.69%	11827	95.33%	7178	94.63%	9875	95.14%
30accepts	7116	93.36%	14927	98.59%	10606	97.05%	12065	97.24%	7306	96.32%	10126	97.56%
31accepts	7195	94.4%	14882	98.3%	10632	97.29%	12115	97.65%	7312	96.4%	10120	97.5%
32accepts	7333	96.21%	15074	99.56%	10788	98.72%	12281	98.98%	7426	97.9%	10265	98.9%
33accepts	7279	95.5%	14945	98.71%	10678	97.71%	12144	97.88%	7333	96.68%	10129	97.59%
34accepts	7107	93.24%	14922	98.56%	10575	96.77%	12013	96.82%	7280	95.98%	10037	96.7%
35accepts	7251	95.13%	14982	98.96%	10725	98.14%	12168	98.07%	7379	97.28%	10166	97.95%
36accepts	7291	95.66%	14948	98.73%	10715	98.05%	12167	98.07%	7379	97.28%	10153	97.82%
37accepts	7153	93.85%	14898	98.4%	10611	97.1%	12101	97.53%	7279	95.97%	10094	97.25%
38accepts	7330	96.17%	15010	99.14%	10763	98.49%	12215	98.45%	7394	97.48%	10243	98.69%
39accepts	7291	95.66%	15003	99.1%	10741	98.29%	12173	98.11%	7390	97.43%	10230	98.56%
40accepts	7090	93.02%	14973	98.9%	10614	97.13%	12052	97.14%	7261	95.73%	10110	97.41%
41accepts	7293	95.68%	14961	98.82%	10728	98.17%	12178	98.15%	7374	97.22%	10183	98.11%
42accepts	6578	86.3%	14513	95.86%	10161	92.98%	11532	92.95%	6908	91.07%	9621	92.7%
43accepts	7274	95.43%	14970	98.88%	10704	97.95%	12141	97.86%	7361	97.05%	10157	97.86%
44accepts	6893	90.44%	14444	95.4%	10297	94.23%	11729	94.54%	7048	92.92%	9714	93.59%
45accepts	7338	96.27%	15000	99.08%	10733	98.22%	12216	98.46%	7411	97.71%	10211	98.38%
46accepts	7076	92.84%	14940	98.68%	10622	97.2%	12061	97.21%	7273	95.89%	10108	97.39%
47accepts	7243	95.03%	14994	99.04%	10719	98.09%	12148	97.91%	7363	97.07%	10211	98.38%
48accepts	7239	94.98%	14970	98.88%	10698	97.9%	12150	97.93%	7351	96.91%	10209	98.36%
49accepts	7306	95.85%	15034	99.3%	10755	98.42%	12198	98.32%	7379	97.28%	10221	98.48%

Table 18: Test Data of Common Pattern Automaton Verification Test Case 02 (2)

Test Case 2: Common Pattern Automaton Verification Test

Test Files: 49 In-world sample files

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: Common Pattern Automaton's accuracy, i.e. if a string is accepted by common pattern automaton, then it is a common string of 49 in-world automatons.

Test Scenario: feed accepted strings from common pattern automaton, into 49 individual automatons to see if any automatons reject strings.

Ideal Result:: no automatons reject common strings from common pattern automaton

Expected Result: close to Ideal Result. Averagely automaton accepts 96.69% common strings from common pattern automaton. In a word, common pattern automaton gives 3.31% non-common strings incorrectly.

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts and acceptance percentage.*
2) *Each column starts from the total strings in a sample file;*
3) *Feed the total strings into Common Pattern Automaton, record the number of strings accepted by Common Pattern Automaton.*
4) *Feed the output from Common Pattern Automaton, i.e., accepted strings from Common Pattern Automaton, into 1st author's automaton, then calculate acceptance percentage. For example: sample 13 has 6947 strings accepted by Common Pattern Automaton, feed 6947 strings into 1st author's automaton, accepts 6738 strings. Then acceptance percentage is 6738/6947 = 96.99%*
5) *Same scenario as 4), feed the accepted strings from Common Pattern Automaton into 2nd author's automaton and then calculate acceptance percentage.*
6) *For each automaton, we perform the same scenario, i.e,. Feed the accepted strings from Common Pattern Automaton into author's automaton and then calculate acceptance percentage.*
7) *At last, add up all acceptance percentage and divided by "49X49", to get an average number, which is 96.69%.*

	Sample 13		Sample 14		Sample 15		Sample 16		Sample 17		Sample 18	
Total strings	12278		10863		14603		10840		10240		9998	
Cmn ptn automaton accepts	6947		8047		9830		7819		6680		6894	
1 accepts	6738	96.99%	7858	97.65%	9532	96.97%	7614	97.38%	6438	96.38%	6662	96.63%
2 accepts	6715	96.66%	7822	97.2%	9505	96.69%	7582	96.97%	6404	95.87%	6657	96.56%
3 accepts	6767	97.41%	7884	97.97%	9571	97.37%	7654	97.89%	6451	96.57%	6696	97.13%
4 accepts	6717	96.69%	7826	97.25%	9484	96.48%	7556	96.64%	6375	95.43%	6642	96.34%
5 accepts	6409	92.26%	7514	93.38%	9121	92.79%	7226	92.42%	6084	91.08%	6301	91.4%
6 accepts	6528	93.97%	7521	93.46%	9153	93.11%	7279	93.09%	6227	93.22%	6427	93.23%
7 accepts	6735	96.95%	7853	97.59%	9517	96.82%	7586	97.02%	6426	96.2%	6695	97.11%
8 accepts	6480	93.28%	7649	95.05%	9300	94.61%	7368	94.23%	6254	93.62%	6409	92.96%
9 accepts	6783	97.64%	7909	98.29%	9607	97.73%	7663	98%	6501	97.32%	6753	97.95%
10accepts	6820	98.17%	7908	98.27%	9595	97.61%	7640	97.71%	6481	97.02%	6723	97.52%
11accepts	6752	97.19%	7869	97.79%	9552	97.17%	7557	96.65%	6451	96.57%	6670	96.75%
12accepts	6775	97.52%	7883	97.96%	9604	97.7%	7629	97.57%	6431	96.27%	6716	97.42%
13accepts	6947	100%	7742	96.21%	9378	95.4%	7478	95.64%	6274	93.92%	6524	94.63%
14accepts	6705	96.52%	8047	100%	9466	96.3%	7564	96.74%	6393	95.7%	6659	96.59%
15accepts	6740	97.02%	7884	97.97%	9830	100%	7626	97.53%	6424	96.17%	6695	97.11%
16accepts	6732	96.91%	7838	97.4%	9536	97.01%	7819	100%	6417	96.06%	6664	96.66%
17accepts	6769	97.44%	7874	97.85%	9535	97%	7609	97.31%	6680	100%	6693	97.08%
18accepts	6749	97.15%	7853	97.59%	9508	96.72%	7579	96.93%	6384	95.57%	6894	100%
19accepts	6752	97.19%	7878	97.9%	9577	97.43%	7615	97.39%	6455	96.63%	6695	97.11%
20accepts	6741	97.03%	7890	98.05%	9559	97.24%	7614	97.38%	6449	96.54%	6719	97.46%
21accepts	6669	96%	7809	97.04%	9492	96.56%	7545	96.5%	6355	95.13%	6603	95.78%
22accepts	6683	96.2%	7763	96.47%	9443	96.06%	7502	95.95%	6349	95.04%	6599	95.72%
23accepts	6769	97.44%	7879	97.91%	9573	97.39%	7623	97.49%	6434	96.32%	6721	97.49%
24accepts	6675	96.08%	7817	97.14%	9477	96.41%	7547	96.52%	6377	95.46%	6620	96.03%
25accepts	6701	96.46%	7864	97.73%	9560	97.25%	7601	97.21%	6442	96.44%	6655	96.53%
26accepts	6751	97.18%	7856	97.63%	9538	97.03%	7558	96.66%	6408	95.93%	6683	96.94%
27accepts	6739	97.01%	7844	97.48%	9501	96.65%	7566	96.76%	6415	96.03%	6653	96.5%
28accepts	6791	97.75%	7890	98.05%	9579	97.45%	7653	97.88%	6453	96.6%	6712	97.36%
29accepts	6609	95.13%	7654	95.12%	9270	94.3%	7391	94.53%	6221	93.13%	6472	93.88%
30accepts	6670	96.01%	7766	96.51%	9446	96.09%	7533	96.34%	6333	94.81%	6559	95.14%
31accepts	6677	96.11%	7807	97.02%	9476	96.4%	7551	96.57%	6416	96.05%	6619	96.01%
32accepts	6790	97.74%	7918	98.4%	9612	97.78%	7658	97.94%	6472	96.89%	6743	97.81%
33accepts	6730	96.88%	7837	97.39%	9536	97.01%	7579	96.93%	6431	96.27%	6673	96.79%
34accepts	6678	96.13%	7752	96.33%	9432	95.95%	7492	95.82%	6332	94.79%	6575	95.37%
35accepts	6738	96.99%	7864	97.73%	9548	97.13%	7593	97.11%	6418	96.08%	6671	96.77%
36accepts	6776	97.54%	7854	97.6%	9554	97.19%	7609	97.31%	6430	96.26%	6698	97.16%
37accepts	6700	96.44%	7829	97.29%	9485	96.49%	7554	96.61%	6366	95.3%	6618	96%
38accepts	6788	97.71%	7923	98.46%	9597	97.63%	7622	97.48%	6476	96.95%	6705	97.26%
39accepts	6707	96.55%	7881	97.94%	9553	97.18%	7617	97.42%	6433	96.3%	6654	96.52%
40accepts	6621	95.31%	7766	96.51%	9438	96.01%	7503	95.96%	6318	94.58%	6530	94.72%
41accepts	6761	97.32%	7884	97.97%	9523	96.88%	7634	97.63%	6469	96.84%	6701	97.2%
42accepts	6357	91.51%	7418	92.18%	9017	91.73%	7151	91.46%	5960	89.22%	6182	89.67%
43accepts	6722	96.76%	7846	97.5%	9547	97.12%	7588	97.05%	6459	96.69%	6652	96.49%
44accepts	6521	93.87%	7576	94.15%	9148	93.06%	7312	93.52%	6152	92.1%	6384	92.6%
45accepts	6815	98.1%	7889	98.04%	9573	97.39%	7642	97.74%	6452	96.59%	6720	97.48%
46accepts	6603	95.05%	7788	96.78%	9411	95.74%	7500	95.92%	6329	94.75%	6541	94.88%
47accepts	6719	96.72%	7844	97.48%	9522	96.87%	7594	97.12%	6404	95.87%	6638	96.29%
48accepts	6715	96.66%	7865	97.74%	9529	96.94%	7571	96.83%	6408	95.93%	6642	96.34%
49accepts	6790	97.74%	7885	97.99%	9567	97.32%	7646	97.79%	6457	96.66%	6717	97.43%

Table 19: Test Data of Common Pattern Automaton Verification Test Case 02 (3)

Test Case 2: Common Pattern Automaton Verification Test

Test Files: 49 In-world sample files

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: Common Pattern Automaton's accuracy, i.e. if a string is accepted by common pattern automaton, then it is a common string of 49 in-world automatons.

Test Scenario: feed accepted strings from common pattern automaton, into 49 individual automatons to see if any automatons reject strings.

Ideal Result:: no automatons reject common strings from common pattern automaton

Expected Result: close to Ideal Result. Averagely automaton accepts 96.69% common strings from common pattern automaton. In a word, common pattern automaton gives 3.31% non-common strings incorrectly.

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts and acceptance percentage.*
2) *Each column starts from the total strings in a sample file;*
3) *Feed the total strings into Common Pattern Automaton, record the number of strings accepted by Common Pattern Automaton.*
4) *Feed the output from Common Pattern Automaton, i.e., accepted strings from Common Pattern Automaton, into 1st author's automaton, then calculate acceptance percentage. For example: sample 19 has 7992 strings accepted by Common Pattern Automaton, feed 7992 strings into 1st author's automaton, accepts 7761 strings. Then acceptance percentage is 7761/7992 = 97.11%*
5) *Same scenario as 4), feed the accepted strings from Common Pattern Automaton into 2nd author's automaton and then calculate acceptance percentage.*
6) *For each automaton, we perform the same scenario, i.e,. Feed the accepted strings from Common Pattern Automaton into author's automaton and then calculate acceptance percentage.*
7) *At last, add up all acceptance percentage and divided by "49X49", to get an average number, which is 96.69%.*

	Sample 19		Sample 20		Sample 21		Sample 22		Sample 23		Sample 24	
Total strings	11851		12960		13974		8655		10862		20928	
Cmn ptn automaton accepts	7992		9393		10616		5460		7496		15933	
1 accepts	7761	97.11%	9157	97.49%	10437	98.31%	5196	95.16%	7317	97.61%	15809	99.22%
2 accepts	7753	97.01%	9113	97.02%	10386	97.83%	5185	94.96%	7291	97.27%	15754	98.88%
3 accepts	7807	97.69%	9163	97.55%	10474	98.66%	5237	95.92%	7322	97.68%	15841	99.42%
4 accepts	7741	96.86%	9101	96.89%	10361	97.6%	5179	94.85%	7260	96.85%	15736	98.76%
5 accepts	7337	91.8%	8700	92.62%	9989	94.09%	4831	88.48%	6979	93.1%	15327	96.2%
6 accepts	7455	93.28%	8767	93.34%	9961	93.83%	4988	91.36%	7006	93.46%	15120	94.9%
7 accepts	7753	97.01%	9153	97.44%	10384	97.81%	5219	95.59%	7317	97.61%	15737	98.77%
8 accepts	7504	93.89%	8892	94.67%	10197	96.05%	4903	89.8%	7114	94.9%	15639	98.15%
9 accepts	7805	97.66%	9230	98.26%	10492	98.83%	5257	96.28%	7353	98.09%	15850	99.48%
10accepts	7770	97.22%	9184	97.77%	10455	98.48%	5259	96.32%	7361	98.2%	15841	99.42%
11accepts	7774	97.27%	9124	97.14%	10385	97.82%	5206	95.35%	7295	97.32%	15758	98.9%
12accepts	7775	97.28%	9169	97.62%	10447	98.41%	5230	95.79%	7352	98.08%	15821	99.3%
13accepts	7609	95.21%	8948	95.26%	10257	96.62%	5066	92.78%	7191	95.93%	15651	98.23%
14accepts	7693	96.26%	9112	97.01%	10392	97.89%	5187	95%	7259	96.84%	15731	98.73%
15accepts	7785	97.41%	9152	97.43%	10412	98.08%	5201	95.26%	7301	97.4%	15789	99.1%
16accepts	7733	96.76%	9154	97.46%	10402	97.98%	5189	95.04%	7297	97.35%	15765	98.95%
17accepts	7744	96.9%	9145	97.36%	10399	97.96%	5223	95.66%	7300	97.39%	15776	99.01%
18accepts	7733	96.76%	9105	96.93%	10380	97.78%	5175	94.78%	7295	97.32%	15779	99.03%
19accepts	7992	100%	9143	97.34%	10410	98.06%	5207	95.37%	7321	97.67%	15775	99.01%
20accepts	7799	97.59%	9393	100%	10453	98.46%	5233	95.84%	7346	98%	15815	99.26%
21accepts	7679	96.08%	9094	96.82%	10616	100%	5113	93.64%	7274	97.04%	15755	98.88%
22accepts	7647	95.68%	9028	96.11%	10281	96.84%	5460	100%	7191	95.93%	15587	97.83%
23accepts	7779	97.33%	9161	97.53%	10442	98.36%	5248	96.12%	7496	100%	15790	99.1%
24accepts	7719	96.58%	9089	96.76%	10376	97.74%	5128	93.92%	7263	96.89%	15933	100%
25accepts	7754	97.02%	9148	97.39%	10459	98.52%	5184	94.95%	7317	97.61%	15834	99.38%
26accepts	7742	96.87%	9148	97.39%	10420	98.15%	5183	94.93%	7306	97.47%	15816	99.27%
27accepts	7689	96.21%	9090	96.77%	10311	97.13%	5210	95.42%	7283	97.16%	15633	98.12%
28accepts	7784	97.4%	9167	97.59%	10448	98.42%	5271	96.54%	7335	97.85%	15814	99.25%
29accepts	7545	94.41%	8880	94.54%	10116	95.29%	5085	93.13%	7091	94.6%	15404	96.68%
30accepts	7677	96.06%	9114	97.03%	10356	97.55%	5108	93.55%	7245	96.65%	15739	98.78%
31accepts	7695	96.28%	9080	96.67%	10340	97.4%	5138	94.1%	7232	96.48%	15733	98.74%
32accepts	7807	97.69%	9211	98.06%	10515	99.05%	5259	96.32%	7345	97.99%	15872	99.62%
33accepts	7718	96.57%	9125	97.15%	10402	97.98%	5214	95.49%	7304	97.44%	15776	99.01%
34accepts	7656	95.8%	9028	96.11%	10334	97.34%	5103	93.46%	7206	96.13%	15713	98.62%
35accepts	7777	97.31%	9142	97.33%	10425	98.2%	5206	95.35%	7295	97.32%	15819	99.28%
36accepts	7752	97%	9155	97.47%	10414	98.1%	5239	95.95%	7335	97.85%	15748	98.84%
37accepts	7667	95.93%	9071	96.57%	10328	97.29%	5181	94.89%	7256	96.8%	15679	98.41%
38accepts	7800	97.6%	9198	97.92%	10477	98.69%	5246	96.08%	7341	97.93%	15839	99.41%
39accepts	7772	97.25%	9166	97.58%	10470	98.62%	5210	95.42%	7329	97.77%	15824	99.32%
40accepts	7637	95.56%	9029	96.12%	10352	97.51%	5043	92.36%	7215	96.25%	15774	99%
41accepts	7785	97.41%	9172	97.65%	10442	98.36%	5214	95.49%	7312	97.55%	15801	99.17%
42accepts	7274	91.02%	8635	91.93%	9946	93.69%	4713	86.32%	6871	91.66%	15349	96.33%
43accepts	7740	96.85%	9133	97.23%	10419	98.14%	5236	95.9%	7290	97.25%	15796	99.14%
44accepts	7434	93.02%	8785	93.53%	10000	94.2%	4942	90.51%	7022	93.68%	15387	96.57%
45accepts	7770	97.22%	9181	97.74%	10468	98.61%	5246	96.08%	7331	97.8%	15820	99.29%
46accepts	7628	95.45%	9038	96.22%	10338	97.38%	5065	92.77%	7226	96.4%	15750	98.85%
47accepts	7737	96.81%	9136	97.26%	10418	98.13%	5161	94.52%	7309	97.51%	15815	99.26%
48accepts	7723	96.63%	9135	97.25%	10406	98.02%	5174	94.76%	7296	97.33%	15771	98.98%
49accepts	7783	97.38%	9172	97.65%	10473	98.65%	5207	95.37%	7323	97.69%	15840	99.42%

Table 20: Test Data of Common Pattern Automaton Verification Test Case 02 (4)

Test Case 2: Common Pattern Automaton Verification Test

Test Files: 49 In-world sample files

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: Common Pattern Automaton's accuracy, i.e. if a string is accepted by common pattern automaton, then it is a common string of 49 in-world automatons.

Test Scenario: feed accepted strings from common pattern automaton, into 49 individual automatons to see if any automatons reject strings.

Ideal Result:: no automatons reject common strings from common pattern automaton

Expected Result: close to Ideal Result. Averagely automaton accepts 96.69% common strings from common pattern automaton. In a word, common pattern automaton gives 3.31% non-common strings incorrectly.

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts and acceptance percentage.*
2) *Each column starts from the total strings in a sample file;*
3) *Feed the total strings into Common Pattern Automaton, record the number of strings accepted by Common Pattern Automaton.*
4) *Feed the output from Common Pattern Automaton, i.e., accepted strings from Common Pattern Automaton, into 1st author's automaton, then calculate acceptance percentage. For example: sample 25 has 11484 strings accepted by Common Pattern Automaton, feed 11484 strings into 1st author's automaton, accepts 11332 strings. Then acceptance percentage is 11332/11484 = 98.68%*
5) *Same scenario as 4), feed the accepted strings from Common Pattern Automaton into 2nd author's automaton and then calculate acceptance percentage.*
6) *For each automaton, we perform the same scenario, i.e,. Feed the accepted strings from Common Pattern Automaton into author's automaton and then calculate acceptance percentage.*
7) *At last, add up all acceptance percentage and divided by "49X49", to get an average number, which is 96.69%.*

	Sample 25		Sample 26		Sample 27		Sample 28		Sample 29		Sample 30	
Total strings	14561		14159		9086		12000		9214		14218	
Cmn ptn automaton accepts	11484		10971		5471		8412		5909		10297	
1 accepts	11332	98.68%	10825	98.67%	5211	95.25%	8146	96.84%	5613	94.99%	10074	97.83%
2 accepts	11290	98.31%	10778	98.24%	5187	94.81%	8154	96.93%	5600	94.77%	10045	97.55%
3 accepts	11337	98.72%	10852	98.92%	5241	95.8%	8204	97.53%	5616	95.04%	10103	98.12%
4 accepts	11259	98.04%	10791	98.36%	5182	94.72%	8132	96.67%	5563	94.14%	10009	97.2%
5 accepts	10897	94.89%	10415	94.93%	4830	88.28%	7764	92.3%	5239	88.66%	9564	92.88%
6 accepts	10871	94.66%	10435	95.11%	5002	91.43%	7842	93.22%	5424	91.79%	9636	93.58%
7 accepts	11296	98.36%	10795	98.4%	5241	95.8%	8138	96.74%	5628	95.24%	10063	97.73%
8 accepts	11154	97.13%	10638	96.96%	4928	90.07%	7913	94.07%	5316	89.96%	9776	94.94%
9 accepts	11353	98.86%	10871	99.09%	5272	96.36%	8240	97.96%	5661	95.8%	10158	98.65%
10accepts	11377	99.07%	10876	99.13%	5251	95.98%	8195	97.42%	5683	96.18%	10103	98.12%
11accepts	11291	98.32%	10784	98.3%	5217	95.36%	8159	96.99%	5601	94.79%	10077	97.86%
12accepts	11358	98.9%	10830	98.71%	5254	96.03%	8227	97.8%	5647	95.57%	10092	98.01%
13accepts	11186	97.41%	10707	97.59%	5044	92.2%	7995	95.04%	5488	92.88%	9886	96.01%
14accepts	11262	98.07%	10770	98.17%	5167	94.44%	8127	96.61%	5575	94.35%	10001	97.13%
15accepts	11307	98.46%	10839	98.8%	5193	94.92%	8140	96.77%	5624	95.18%	10069	97.79%
16accepts	11313	98.51%	10814	98.57%	5215	95.32%	8166	97.08%	5594	94.67%	10033	97.44%
17accepts	11292	98.33%	10814	98.57%	5242	95.81%	8146	96.84%	5629	95.26%	10078	97.87%
18accepts	11332	98.68%	10814	98.57%	5174	94.57%	8159	96.99%	5589	94.58%	10051	97.61%
19accepts	11310	98.48%	10830	98.71%	5218	95.38%	8197	97.44%	5639	95.43%	10082	97.91%
20accepts	11312	98.5%	10836	98.77%	5232	95.63%	8237	97.92%	5650	95.62%	10096	98.05%
21accepts	11262	98.07%	10786	98.31%	5096	93.15%	8136	96.72%	5544	93.82%	10001	97.13%
22accepts	11199	97.52%	10692	97.46%	5129	93.75%	8021	95.35%	5556	94.03%	9937	96.5%
23accepts	11309	98.48%	10843	98.83%	5241	95.8%	8201	97.49%	5645	95.53%	10103	98.12%
24accepts	11300	98.4%	10808	98.51%	5115	93.49%	8107	96.37%	5493	92.96%	10028	97.39%
25accepts	11484	100%	10867	99.05%	5191	94.88%	8178	97.22%	5572	94.3%	10075	97.84%
26accepts	11326	98.62%	10971	100%	5201	95.06%	8198	97.46%	5620	95.11%	10079	97.88%
27accepts	11209	97.61%	10736	97.86%	5471	100%	8139	96.75%	5676	96.06%	9987	96.99%
28accepts	11348	98.82%	10848	98.88%	5249	95.94%	8412	100%	5656	95.72%	10109	98.17%
29accepts	10993	95.72%	10558	96.24%	5083	92.91%	7961	94.64%	5909	100%	9809	95.26%
30accepts	11258	98.03%	10770	98.17%	5098	93.18%	8068	95.91%	5490	92.91%	10297	100%
31accepts	11267	98.11%	10724	97.75%	5150	94.13%	8130	96.65%	5594	94.67%	9979	96.91%
32accepts	11385	99.14%	10890	99.26%	5259	96.13%	8224	97.77%	5662	95.82%	10163	98.7%
33accepts	11285	98.27%	10794	98.39%	5214	95.3%	8156	96.96%	5614	95.01%	10035	97.46%
34accepts	11217	97.68%	10741	97.9%	5104	93.29%	8083	96.09%	5496	93.01%	9943	96.56%
35accepts	11301	98.41%	10828	98.7%	5191	94.88%	8154	96.93%	5624	95.18%	10064	97.74%
36accepts	11314	98.52%	10797	98.41%	5236	95.7%	8185	97.3%	5662	95.82%	10079	97.88%
37accepts	11265	98.09%	10748	97.97%	5156	94.24%	8058	95.79%	5561	94.11%	9995	97.07%
38accepts	11352	98.85%	10864	99.02%	5268	96.29%	8194	97.41%	5672	95.99%	10118	98.26%
39accepts	11337	98.72%	10852	98.92%	5199	95.03%	8190	97.36%	5605	94.86%	10095	98.04%
40accepts	11253	97.99%	10793	98.38%	5046	92.23%	8045	95.64%	5432	91.93%	9968	96.8%
41accepts	11317	98.55%	10836	98.77%	5250	95.96%	8161	97.02%	5650	95.62%	10094	98.03%
42accepts	10858	94.55%	10429	95.06%	4731	86.47%	7664	91.11%	5103	86.36%	9479	92.06%
43accepts	11323	98.6%	10833	98.74%	5219	95.39%	8177	97.21%	5633	95.33%	10086	97.95%
44accepts	10918	95.07%	10440	95.16%	4954	90.55%	7790	92.61%	5382	91.08%	9650	93.72%
45accepts	11361	98.93%	10865	99.03%	5264	96.22%	8219	97.71%	5686	96.23%	10106	98.15%
46accepts	11230	97.79%	10761	98.09%	5041	92.14%	8039	95.57%	5439	92.05%	9947	96.6%
47accepts	11313	98.51%	10825	98.67%	5175	94.59%	8175	97.18%	5550	93.92%	10066	97.76%
48accepts	11318	98.55%	10816	98.59%	5173	94.55%	8156	96.96%	5562	94.13%	10019	97.3%
49accepts	11336	98.71%	10863	99.02%	5254	96.03%	8196	97.43%	5639	95.43%	10105	98.14%

Table 21: Test Data of Common Pattern Automaton Verification Test Case 02 (5)

Test Case 2: Common Pattern Automaton Verification Test

Test Files: 49 In-world sample files

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: Common rPattern Automaton's accuracy, i.e. if a string is accepted by common pattern automaton, then it is a common string of 49 in-world automatons.

Test Scenario: feed accepted strings from common pattern automaton, into 49 individual automatons to see if any automatons reject strings.

Ideal Result:: no automatons reject common strings from common pattern automaton

Expected Result: close to Ideal Result. Averagely automaton accepts 96.69% common strings from common pattern automaton. In a word, common pattern automaton gives 3.31% non-common strings incorrectly.

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts and acceptance percentage.*
2) *Each column starts from the total strings in a sample file;*
3) *Feed the total strings into Common Pattern Automaton, record the number of strings accepted by Common Pattern Automaton.*
4) *Feed the output from Common Pattern Automaton, i.e., accepted strings from Common Pattern Automaton, into 1st author's automaton, then calculate acceptance percentage. For example: sample 31 has 6368 strings accepted by Common Pattern Automaton, feed 6368 strings into 1st author's automaton, accepts 6115 strings. Then acceptance percentage is 6115/6368 = 96.03%*
5) *Same scenario as 4), feed the accepted strings from Common Pattern Automaton into 2nd author's automaton and then calculate acceptance percentage.*
6) *For each automaton, we perform the same scenario, i.e,. Feed the accepted strings from Common Pattern Automaton into author's automaton and then calculate acceptance percentage.*
7) *At last, add up all acceptance percentage and divided by "49X49", to get an average number, which is 96.69%.*

	Sample 31		Sample 32		Sample 33		Sample 34		Sample 35		Sample 36	
Total strings	9493		15569		9876		12871		15938		10838	
Cmn ptn automaton accepts	6368		11295		6606		8736		11792		7505	
1 accepts	6115	96.03%	11129	98.53%	6389	96.72%	8559	97.97%	11522	97.71%	7263	96.78%
2 accepts	6059	95.15%	11109	98.35%	6400	96.88%	8538	97.73%	11487	97.41%	7210	96.07%
3 accepts	6114	96.01%	11155	98.76%	6441	97.5%	8558	97.96%	11578	98.19%	7293	97.18%
4 accepts	6070	95.32%	11073	98.03%	6357	96.23%	8521	97.54%	11451	97.11%	7182	95.7%
5 accepts	5764	90.52%	10693	94.67%	6018	91.1%	8225	94.15%	11046	93.67%	6792	90.5%
6 accepts	5850	91.87%	10672	94.48%	6150	93.1%	8208	93.96%	11075	93.92%	6961	92.75%
7 accepts	6102	95.82%	11096	98.24%	6405	96.96%	8520	97.53%	11511	97.62%	7259	96.72%
8 accepts	5898	92.62%	10933	96.8%	6153	93.14%	8379	95.91%	11308	95.9%	6974	92.92%
9 accepts	6180	97.05%	11201	99.17%	6482	98.12%	8608	98.53%	11585	98.24%	7323	97.57%
10accepts	6165	96.81%	11196	99.12%	6450	97.64%	8591	98.34%	11619	98.53%	7284	97.06%
11accepts	6097	95.74%	11092	98.2%	6404	96.94%	8521	97.54%	11519	97.68%	7249	96.59%
12accepts	6128	96.23%	11162	98.82%	6427	97.29%	8589	98.32%	11582	98.22%	7283	97.04%
13accepts	5930	93.12%	10936	96.82%	6272	94.94%	8425	96.44%	11360	96.34%	7080	94.34%
14accepts	6038	94.82%	11080	98.1%	6347	96.08%	8508	97.39%	11467	97.24%	7223	96.24%
15accepts	6118	96.07%	11092	98.2%	6388	96.7%	8546	97.83%	11536	97.83%	7263	96.78%
16accepts	6127	96.22%	11094	98.22%	6379	96.56%	8531	97.65%	11531	97.79%	7204	95.99%
17accepts	6102	95.82%	11122	98.47%	6402	96.91%	8541	97.77%	11522	97.71%	7258	96.71%
18accepts	6058	95.13%	11103	98.3%	6372	96.46%	8525	97.58%	11525	97.74%	7211	96.08%
19accepts	6125	96.18%	11102	98.29%	6430	97.34%	8558	97.96%	11533	97.8%	7257	96.7%
20accepts	6122	96.14%	11139	98.62%	6415	97.11%	8558	97.96%	11565	98.07%	7291	97.15%
21accepts	6051	95.02%	11093	98.21%	6302	95.4%	8487	97.15%	11490	97.44%	7167	95.5%
22accepts	6041	94.86%	10994	97.34%	6338	95.94%	8462	96.86%	11385	96.55%	7170	95.54%
23accepts	6125	96.18%	11127	98.51%	6422	97.21%	8567	98.07%	11548	97.93%	7255	96.67%
24accepts	6040	94.85%	11116	98.42%	6364	96.34%	8482	97.09%	11504	97.56%	7155	95.34%
25accepts	6082	95.51%	11159	98.8%	6385	96.65%	8539	97.74%	11557	98.01%	7230	96.34%
26accepts	6093	95.68%	11127	98.51%	6384	96.64%	8543	97.79%	11547	97.92%	7243	96.51%
27accepts	6085	95.56%	11019	97.56%	6370	96.43%	8517	97.49%	11464	97.22%	7227	96.3%
28accepts	6127	96.22%	11167	98.87%	6433	97.38%	8577	98.18%	11563	98.06%	7278	96.98%
29accepts	5912	92.84%	10854	96.1%	6200	93.85%	8379	95.91%	11242	95.34%	7045	93.87%
30accepts	6027	94.65%	11070	98.01%	6309	95.5%	8474	97%	11439	97.01%	7137	95.1%
31accepts	6368	100%	11061	97.93%	6331	95.84%	8488	97.16%	11448	97.08%	7208	96.04%
32accepts	6142	96.45%	11295	100%	6467	97.9%	8603	98.48%	11613	98.48%	7303	97.31%
33accepts	6096	95.73%	11088	98.17%	6606	100%	8530	97.64%	11519	97.68%	7214	96.12%
34accepts	5981	93.92%	11014	97.51%	6290	95.22%	8736	100%	11444	97.05%	7143	95.18%
35accepts	6122	96.14%	11132	98.56%	6410	97.03%	8561	98%	11792	100%	7242	96.5%
36accepts	6105	95.87%	11077	98.07%	6413	97.08%	8539	97.74%	11542	97.88%	7505	100%
37accepts	6051	95.02%	11043	97.77%	6323	95.72%	8491	97.2%	11450	97.1%	7187	95.76%
38accepts	6151	96.59%	11178	98.96%	6428	97.31%	8597	98.41%	11575	98.16%	7304	97.32%
39accepts	6117	96.06%	11167	98.87%	6424	97.24%	8561	98%	11543	97.89%	7248	96.58%
40accepts	5978	93.88%	11057	97.89%	6271	94.93%	8481	97.08%	11471	97.28%	7138	95.11%
41accepts	6121	96.12%	11141	98.64%	6401	96.9%	8573	98.13%	11548	97.93%	7272	96.9%
42accepts	5649	88.71%	10643	94.23%	5941	89.93%	8147	93.26%	11042	93.64%	6693	89.18%
43accepts	6112	95.98%	11116	98.42%	6412	97.06%	8524	97.57%	11516	97.66%	7257	96.7%
44accepts	5831	91.57%	10711	94.83%	6113	92.54%	8275	94.72%	11131	94.39%	6938	92.45%
45accepts	6125	96.18%	11138	98.61%	6441	97.5%	8581	98.23%	11566	98.08%	7281	97.02%
46accepts	6002	94.25%	11062	97.94%	6301	95.38%	8479	97.06%	11420	96.85%	7124	94.92%
47accepts	6079	95.46%	11136	98.59%	6362	96.31%	8548	97.85%	11518	97.68%	7214	96.12%
48accepts	6100	95.79%	11098	98.26%	6367	96.38%	8543	97.79%	11520	97.69%	7234	96.39%
49accepts	6122	96.14%	11142	98.65%	6411	97.05%	8584	98.26%	11553	97.97%	7300	97.27%

Table 22: Test Data of Common Pattern Automaton Verification Test Case 02 (6)

Test Case 2: Common Pattern Automaton Verification Test

Test Files: 49 In-world sample files

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: Common rPattern Automaton's accuracy, i.e. if a string is accepted by common pattern automaton, then it is a common string of 49 in-world automatons.

Test Scenario: feed accepted strings from common pattern automaton, into 49 individual automatons to see if any automatons reject strings.

Ideal Result:: no automatons reject common strings from common pattern automaton

Expected Result: close to Ideal Result. Averagely automaton accepts 96.69% common strings from common pattern automaton. In a word, common pattern automaton gives 3.31% non-common strings incorrectly.

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts and acceptance percentage.*
2) *Each column starts from the total strings in a sample file;*
3) *Feed the total strings into Common Pattern Automaton, record the number of strings accepted by Common Pattern Automaton.*
4) *Feed the output from Common Pattern Automaton, i.e., accepted strings from Common Pattern Automaton, into 1st author's automaton, then calculate acceptance percentage. For example: sample 37 has 6859 strings accepted by Common Pattern Automaton, feed 6859 strings into 1st author's automaton, accepts 6664 strings. Then acceptance percentage is 6664/6859 = 97.16%*
5) *Same scenario as 4), feed the accepted strings from Common Pattern Automaton into 2nd author's automaton and then calculate acceptance percentage.*
6) *For each automaton, we perform the same scenario, i.e,. Feed the accepted strings from Common Pattern Automaton into author's automaton and then calculate acceptance percentage.*
7) *At last, add up all acceptance percentage and divided by "49X49", to get an average number, which is 96.69%.*

	Sample 37		Sample 38		Sample 39		Sample 40		Sample 41		Sample 42	
Total strings	9849		14746		15246		15779		11355		12444	
Cmn ptn automaton accepts	6859		10887		11587		11103		8263		7170	
1 accepts	6664	97.16%	10679	98.09%	11437	98.71%	10985	98.94%	8078	97.76%	6988	97.46%
2 accepts	6640	96.81%	10636	97.69%	11410	98.47%	10958	98.69%	8031	97.19%	6971	97.22%
3 accepts	6698	97.65%	10714	98.41%	11449	98.81%	10998	99.05%	8104	98.08%	7018	97.88%
4 accepts	6632	96.69%	10605	97.41%	11379	98.2%	10908	98.24%	8033	97.22%	6946	96.88%
5 accepts	6300	91.85%	10215	93.83%	11022	95.12%	10654	95.96%	7670	92.82%	6738	93.97%
6 accepts	6384	93.07%	10216	93.84%	11011	95.03%	10499	94.56%	7743	93.71%	6780	94.56%
7 accepts	6677	97.35%	10634	97.68%	11384	98.25%	10914	98.3%	8060	97.54%	6971	97.22%
8 accepts	6441	93.91%	10459	96.07%	11273	97.29%	10842	97.65%	7857	95.09%	6771	94.44%
9 accepts	6722	98%	10739	98.64%	11463	98.93%	11018	99.23%	8132	98.41%	7021	97.92%
10accepts	6711	97.84%	10742	98.67%	11459	98.9%	11017	99.23%	8112	98.17%	7007	97.73%
11accepts	6653	97%	10659	97.91%	11391	98.31%	10948	98.6%	8040	97.3%	6996	97.57%
12accepts	6701	97.7%	10709	98.37%	11468	98.97%	10973	98.83%	8083	97.82%	6993	97.53%
13accepts	6540	95.35%	10487	96.33%	11298	97.51%	10845	97.68%	7922	95.87%	6850	95.54%
14accepts	6590	96.08%	10606	97.42%	11391	98.31%	10934	98.48%	8022	97.08%	6966	97.15%
15accepts	6677	97.35%	10649	97.81%	11450	98.82%	10986	98.95%	8063	97.58%	6982	97.38%
16accepts	6618	96.49%	10650	97.82%	11394	98.33%	10966	98.77%	8052	97.45%	6965	97.14%
17accepts	6646	96.89%	10654	97.86%	11410	98.47%	10942	98.55%	8015	97%	7001	97.64%
18accepts	6641	96.82%	10627	97.61%	11380	98.21%	10972	98.82%	8025	97.12%	6980	97.35%
19accepts	6663	97.14%	10683	98.13%	11430	98.65%	10976	98.86%	8048	97.4%	7015	97.84%
20accepts	6667	97.2%	10694	98.23%	11419	98.55%	10991	98.99%	8086	97.86%	7012	97.8%
21accepts	6620	96.52%	10614	97.49%	11407	98.45%	10951	98.63%	8010	96.94%	6923	96.56%
22accepts	6567	95.74%	10529	96.71%	11325	97.74%	10853	97.75%	7937	96.05%	6908	96.35%
23accepts	6661	97.11%	10675	98.05%	11439	98.72%	10968	98.78%	8072	97.69%	6980	97.35%
24accepts	6616	96.46%	10630	97.64%	11423	98.58%	10965	98.76%	8013	96.97%	6962	97.1%
25accepts	6652	96.98%	10680	98.1%	11422	98.58%	11005	99.12%	8065	97.6%	6982	97.38%
26accepts	6658	97.07%	10657	97.89%	11426	98.61%	10996	99.04%	8046	97.37%	7004	97.68%
27accepts	6589	96.06%	10569	97.08%	11327	97.76%	10886	98.05%	7993	96.73%	6976	97.29%
28accepts	6668	97.22%	10701	98.29%	11441	98.74%	10994	99.02%	8085	97.85%	7011	97.78%
29accepts	6478	94.45%	10396	95.49%	11148	96.21%	10726	96.6%	7836	94.83%	6808	94.95%
30accepts	6557	95.6%	10581	97.19%	11383	98.24%	10943	98.56%	7983	96.61%	6941	96.81%
31accepts	6601	96.24%	10604	97.4%	11387	98.27%	10913	98.29%	7982	96.6%	6962	97.1%
32accepts	6710	97.83%	10738	98.63%	11493	99.19%	11023	99.28%	8134	98.44%	7027	98.01%
33accepts	6672	97.27%	10656	97.88%	11411	98.48%	10945	98.58%	8029	97.17%	6976	97.29%
34accepts	6547	95.45%	10548	96.89%	11338	97.85%	10926	98.41%	7915	95.79%	6929	96.64%
35accepts	6652	96.98%	10684	98.14%	11422	98.58%	10966	98.77%	8049	97.41%	6991	97.5%
36accepts	6658	97.07%	10661	97.92%	11393	98.33%	10945	98.58%	8037	97.26%	6994	97.55%
37accepts	6859	100%	10569	97.08%	11361	98.05%	10905	98.22%	7987	96.66%	6942	96.82%
38accepts	6708	97.8%	10887	100%	11447	98.79%	10995	99.03%	8093	97.94%	7043	98.23%
39accepts	6687	97.49%	10666	97.97%	11587	100%	10998	99.05%	8072	97.69%	6997	97.59%
40accepts	6568	95.76%	10566	97.05%	11342	97.89%	11103	100%	7973	96.49%	6892	96.12%
41accepts	6661	97.11%	10663	97.94%	11415	98.52%	10978	98.87%	8263	100%	6991	97.5%
42accepts	6201	90.41%	10146	93.19%	10929	94.32%	10623	95.68%	7587	91.82%	7170	100%
43accepts	6673	97.29%	10661	97.92%	11432	98.66%	10986	98.95%	8050	97.42%	7001	97.64%
44accepts	6344	92.49%	10261	94.25%	11019	95.1%	10625	95.69%	7715	93.37%	6753	94.18%
45accepts	6707	97.78%	10687	98.16%	11450	98.82%	11019	99.24%	8087	97.87%	7032	98.08%
46accepts	6551	95.51%	10582	97.2%	11343	97.89%	10924	98.39%	7930	95.97%	6877	95.91%
47accepts	6652	96.98%	10660	97.91%	11421	98.57%	10986	98.95%	8061	97.56%	6963	97.11%
48accepts	6655	97.03%	10648	97.8%	11394	98.33%	10968	98.78%	8037	97.26%	6979	97.34%
49accepts	6859	100%	10887	100%	11587	100%	11103	100%	8263	100%	7170	100%

Table 23: Test Data of Common Pattern Automaton Verification Test Case 02 (7)

Test Case 2: Common Pattern Automaton Verification Test

Test Files: 49 In-world sample files

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order

Test Purpose: Common rPattern Automaton's accuracy, i.e. if a string is accepted by common pattern automaton, then it is a common string of 49 in-world automatons.

Test Scenario: feed accepted strings from common pattern automaton, into 49 individual automatons to see if any automatons reject strings.

Ideal Result:: no automatons reject common strings from common pattern automaton

Expected Result: close to Ideal Result. Averagely automaton accepts 96.69% common strings from common pattern automaton. In a word, common pattern automaton gives 3.31% non-common strings incorrectly.

Actual Result: as expected

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts and acceptance percentage.*
2) *Each column starts from the total strings in a sample file;*
3) *Feed the total strings into Common Pattern Automaton, record the number of strings accepted by Common Pattern Automaton.*
4) *Feed the output from Common Pattern Automaton, i.e., accepted strings from Common Pattern Automaton, into 1st author's automaton, then calculate acceptance percentage. For example: sample 43 has 6750 strings accepted by Common Pattern Automaton, feed 6750 strings into 1st author's automaton, accepts 6538 strings. Then acceptance percentage is 6538/6750 = 96.86%*
5) *Same scenario as 4), feed the accepted strings from Common Pattern Automaton into 2nd author's automaton and then calculate acceptance percentage.*
6) *For each automaton, we perform the same scenario, i.e,. Feed the accepted strings from Common Pattern Automaton into author's automaton and then calculate acceptance percentage.*
7) *At last, add up all acceptance percentage and divided by "49X49", to get an average number, which is 96.69%.*

	Sample 43		Sample 44		Sample 45		Sample 46		Sample 47		Sample 48		Sample 49	
Total strings	10306		7786		11856		13994		15746		15088		22513	
Cmn ptn automaton accepts	6750		4306		8474		10217		11215		11070		17249	
1 accepts	6538	96.86%	4000	92.89%	8323	98.22%	10043	98.3%	11067	98.68%	10907	98.53%	17137	99.35%
2 accepts	6509	96.43%	3976	92.34%	8293	97.86%	10042	98.29%	11037	98.41%	10893	98.4%	17071	98.97%
3 accepts	6587	97.59%	4034	93.68%	8315	98.12%	10056	98.42%	11059	98.61%	10937	98.8%	17154	99.45%
4 accepts	6487	96.1%	3968	92.15%	8241	97.25%	9968	97.56%	10984	97.94%	10835	97.88%	17023	98.69%
5 accepts	6224	92.21%	3695	85.81%	7964	93.98%	9638	94.33%	10624	94.73%	10523	95.06%	16623	96.37%
6 accepts	6283	93.08%	3842	89.22%	8004	94.45%	9622	94.18%	10591	94.44%	10448	94.38%	16371	94.91%
7 accepts	6532	96.77%	4051	94.08%	8294	97.88%	10015	98.02%	10972	97.83%	10876	98.25%	17044	98.81%
8 accepts	6336	93.87%	3773	87.62%	8138	96.03%	9842	96.33%	10858	96.82%	10753	97.14%	16942	98.22%
9 accepts	6604	97.84%	4077	94.68%	8365	98.71%	10097	98.83%	11093	98.91%	10963	99.03%	17178	99.59%
10accepts	6572	97.36%	4058	94.24%	8381	98.9%	10072	98.58%	11081	98.81%	10931	98.74%	17174	99.57%
11accepts	6548	97.01%	4064	94.38%	8307	98.03%	10023	98.1%	11029	98.34%	10874	98.23%	17078	99.01%
12accepts	6552	97.07%	4078	94.71%	8330	98.3%	10053	98.39%	11037	98.41%	10946	98.88%	17115	99.22%
13accepts	6419	95.1%	3902	90.62%	8180	96.53%	9863	96.54%	10901	97.2%	10763	97.23%	16931	98.16%
14accepts	6492	96.18%	3986	92.57%	8267	97.56%	9968	97.56%	10984	97.94%	10877	98.26%	17057	98.89%
15accepts	6526	96.68%	4031	93.61%	8300	97.95%	10022	98.09%	11034	98.39%	10890	98.37%	17096	99.11%
16accepts	6519	96.58%	4020	93.36%	8287	97.79%	10016	98.03%	10990	97.99%	10862	98.12%	17082	99.03%
17accepts	6562	97.21%	4036	93.73%	8302	97.97%	10008	97.95%	11021	98.27%	10885	98.33%	17088	99.07%
18accepts	6495	96.22%	3975	92.31%	8294	97.88%	10021	98.08%	11031	98.36%	10891	98.38%	17103	99.15%
19accepts	6549	97.02%	4027	93.52%	8316	98.14%	10028	98.15%	11045	98.48%	10889	98.36%	17085	99.05%
20accepts	6570	97.33%	4070	94.52%	8345	98.48%	10035	98.22%	11090	98.89%	10924	98.68%	17131	99.32%
21accepts	6493	96.19%	3906	90.71%	8256	97.43%	10006	97.93%	10987	97.97%	10874	98.23%	17078	99.01%
22accepts	6477	95.96%	3966	92.1%	8187	96.61%	9838	96.29%	10910	97.28%	10779	97.37%	16914	98.06%
23accepts	6550	97.04%	4022	93.4%	8322	98.21%	10035	98.22%	11051	98.54%	10904	98.5%	17116	99.23%
24accepts	6453	95.6%	3917	90.97%	8278	97.69%	9999	97.87%	11002	98.1%	10867	98.17%	17101	99.14%
25accepts	6520	96.59%	3967	92.13%	8326	98.25%	10063	98.49%	11045	98.48%	10931	98.74%	17149	99.42%
26accepts	6532	96.77%	3995	92.78%	8309	98.05%	10036	98.23%	11049	98.52%	10881	98.29%	17111	99.2%
27accepts	6506	96.39%	4065	94.4%	8242	97.26%	9946	97.35%	10957	97.7%	10813	97.68%	16963	98.34%
28accepts	6553	97.08%	4042	93.87%	8333	98.34%	10086	98.72%	11059	98.61%	10945	98.87%	17099	99.13%
29accepts	6366	94.31%	3891	90.36%	8070	95.23%	9733	95.26%	10744	95.8%	10626	95.99%	16715	96.9%
30accepts	6451	95.57%	3927	91.2%	8249	97.34%	10015	98.02%	10955	97.68%	10857	98.08%	17062	98.92%
31accepts	6498	96.27%	3998	92.85%	8250	97.36%	9950	97.39%	10975	97.86%	10873	98.22%	17034	98.75%
32accepts	6555	97.11%	4042	93.87%	8378	98.87%	10108	98.93%	11099	98.97%	10980	99.19%	17200	99.72%
33accepts	6545	96.96%	4041	93.85%	8317	98.15%	10028	98.15%	11019	98.25%	10886	98.34%	17088	99.07%
34accepts	6418	95.08%	3932	91.31%	8213	96.92%	9916	97.05%	10945	97.59%	10820	97.74%	17015	98.64%
35accepts	6555	97.11%	4025	93.47%	8317	98.15%	10035	98.22%	11050	98.53%	10916	98.61%	17102	99.15%
36accepts	6572	97.36%	4065	94.4%	8303	97.98%	10036	98.23%	11023	98.29%	10900	98.46%	17034	98.75%
37accepts	6500	96.3%	3963	92.03%	8249	97.34%	9957	97.46%	10995	98.04%	10856	98.07%	17039	98.78%
38accepts	6557	97.14%	4043	93.89%	8342	98.44%	10068	98.54%	11063	98.64%	10955	98.96%	17166	99.52%
39accepts	6554	97.1%	4027	93.52%	8313	98.1%	10084	98.7%	11048	98.51%	10918	98.63%	17133	99.33%
40accepts	6422	95.14%	3885	90.22%	8237	97.2%	10000	97.88%	10981	97.91%	10856	98.07%	17095	99.11%
41accepts	6537	96.84%	4038	93.78%	8334	98.35%	10018	98.05%	11055	98.57%	10917	98.62%	17109	99.19%
42accepts	6118	90.64%	3622	84.12%	7874	92.92%	9553	93.5%	10562	94.18%	10504	94.89%	16653	96.54%
43accepts	6750	100%	4013	93.2%	8293	97.86%	10035	98.22%	11024	98.3%	10902	98.48%	17108	99.18%
44accepts	6260	92.74%	4306	100%	7976	94.12%	9625	94.21%	10726	95.64%	10591	95.67%	16643	96.49%
45accepts	6578	97.45%	4047	93.99%	8474	100%	10056	98.42%	11068	98.69%	10925	98.69%	17134	99.33%
46accepts	6403	94.86%	3845	89.29%	8233	97.16%	10217	100%	10965	97.77%	10848	97.99%	17088	99.07%
47accepts	6539	96.87%	3992	92.71%	8292	97.85%	10050	98.37%	11215	100%	10921	98.65%	17105	99.17%
48accepts	6487	96.1%	3989	92.64%	8294	97.88%	10031	98.18%	11032	98.37%	11070	100%	17103	99.15%
49accepts	6750	100%	4306	100%	8474	100%	10217	100%	11215	100%	11070	100%	17249	100%

Table 24: Test Data of Common Pattern Automaton Verification Test Case 02 (8)

Test Case 3: Common Pattern Automaton Verification Test

Test Sample Files: 25 Out-world sample files, which are not from 49 in-world authors.

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order.

Test Purpose: Common pattern automaton's completeness, i.e, if Common pattern automaton covers all common strings among 49 in-world automatons.

Test Scenario: feed rejected strings from common pattern automaton, into 49 in-world automaton list to check if there is any common string being found. For example, common pattern automaton rejects 2537 strings of sample 1, feed 2537 strings into 1st automaton, accepts 1711, and then feed 1711 into 2nd automaton, accepts 1318, and so on, until 49th automaton, ideal result at 49th automaton is 0 string accepted.

Expected Result: No common string found

Actual Result: small amount found, about 0.5% common strings are found

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts; Last row data are common string percentage.*
2) *Each column starts from the number of strings rejected by Common Pattern Automaton;*
3) *Feed the output from Common Pattern Automaton, i.e., rejected strings from Common Pattern Automaton, into 1st author's automaton*
4) *Feed the output from 1st author's automaton, i.e., accepted strings from 1st author's automaton, into next author's automaton, i.e., 2nd author's automaton.*
5) *Keep on feeding output, accepted strings, into next author's automaton until last author in the list, i.e, 49th author's automaton.*
6) *We care how many strings are eventually accepted by 49th author's automaton. Ideal result should be 0.*
7) *Last row is the percentage of common strings found from the set of rejected strings from Common Pattern Automaton. For example: Sample 1 has 2537 strings rejected by Common Pattern Automaton, and at last 14 strings of 2537 are accepted by all 49 in-world automatons, i.e., 14 common strings for 49 in-world automatons. Therefore, the percentage is 14/2537 = 0.5%.*

	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6	Sample 7	Sample 8	Sample 9	Sample 10	Sample 11	Sample 12	Sample 13
Cmn ptn automaton rejects	2537	1835	1659	1339	1561	685	162	1249	1927	1478	1690	111	923
1 accepts	1711	1244	1179	1010	1152	411	117	808	1355	1008	1201	80	776
2 accepts	1318	957	916	822	909	295	95	595	1066	765	945	65	697
3 accepts	1130	811	791	731	810	244	86	501	910	666	819	57	643
4 accepts	981	710	693	647	743	207	72	427	806	576	714	46	599
5 accepts	747	541	532	498	580	153	61	318	607	417	527	37	483
6 accepts	603	445	434	422	502	118	46	245	480	337	423	31	393
7 accepts	549	401	401	382	460	107	44	225	438	306	396	30	378
8 accepts	441	317	322	307	374	86	32	182	350	236	309	23	307
9 accepts	418	306	309	288	358	84	31	176	329	221	294	20	303
10accepts	390	290	290	271	330	78	28	166	316	206	279	18	296
11accepts	376	272	280	264	309	75	26	159	291	197	267	17	283
12accepts	357	257	266	252	296	71	26	147	279	185	255	17	272
13accepts	306	230	235	219	268	57	24	123	252	172	236	17	246
14accepts	286	213	219	210	257	52	23	115	240	158	221	16	235
15accepts	274	199	212	205	243	50	22	109	225	147	209	16	225
16accepts	259	184	203	192	236	45	19	99	212	136	197	15	222
17accepts	245	168	194	185	225	44	18	95	201	129	187	15	211
18accepts	223	157	183	178	212	42	17	89	183	124	178	15	198
19accepts	217	153	178	174	205	38	17	84	179	121	173	14	194
20accepts	204	148	170	166	202	35	15	80	175	116	166	12	188
21accepts	199	139	163	151	188	30	15	75	171	113	161	11	183
22accepts	181	120	154	140	171	24	13	64	158	104	151	11	171
23accepts	173	114	146	137	165	22	13	62	147	98	143	10	162
24accepts	166	107	140	134	162	20	13	60	142	96	136	9	160
25accepts	160	101	134	130	156	20	11	59	140	94	131	9	156
26accepts	157	100	130	128	151	20	10	57	135	91	125	9	150
27accepts	140	88	114	119	143	18	9	52	126	79	109	8	121
28accepts	137	86	111	117	137	18	9	52	126	78	106	8	118
29accepts	118	74	93	97	114	15	8	46	103	63	92	8	105
30accepts	113	71	86	89	110	15	7	44	99	57	87	8	101
31accepts	100	65	81	84	102	15	7	44	95	53	83	8	93
32accepts	100	65	80	84	101	15	7	43	92	52	83	8	93
33accepts	93	64	78	79	99	15	7	42	89	51	78	8	90
34accepts	88	60	75	74	92	14	6	38	83	50	75	8	84
35accepts	86	60	73	72	90	12	6	36	81	49	72	8	84
36accepts	76	53	68	69	81	11	5	33	71	47	59	6	78
37accepts	73	47	62	62	75	9	5	29	67	42	55	6	70
38accepts	70	47	60	59	75	9	5	29	67	38	54	6	67
39accepts	67	46	59	54	74	9	5	28	65	36	52	6	67
40accepts	64	44	54	50	68	9	5	27	61	35	51	5	66
41accepts	58	39	51	50	61	8	4	24	51	32	46	5	62
42accepts	49	32	39	38	50	7	3	20	41	28	39	5	48
43accepts	46	27	36	29	44	6	2	17	37	25	35	5	43
44accepts	22	12	17	17	22	5	0	10	11	12	14	1	12
45accepts	17	12	16	16	21	5	0	9	11	11	12	1	12
46accepts	16	12	13	16	20	5	0	9	10	10	11	1	12
47accepts	15	12	9	16	17	5	0	9	9	9	7	0	11
48accepts	15	10	8	16	15	5	0	8	9	8	6	0	6
49accepts	14	9	7	15	15	5	0	8	9	8	5	0	5
Cmn str %	0.5	0.5	0.4	1	0.9	0.7	0	0.6	0.5	0.5	0.3	0	0.5

Table 25: Test Data of Common Pattern Automaton Verification Test Case 03 (1)

Test Case 3: Common Pattern Automaton Verification Test

Test Sample Files: 25 Out-world sample files, which are not from 49 in-world authors.

Test Automatons: 49 in-world automatons, automaton list is in alphabet ascending order.

Test Purpose: Common pattern automaton's completeness, i.e, if Common pattern automaton covers all common strings among 49 in-world automatons.

Test Scenario: feed rejected strings from common pattern automaton, into 49 in-world automaton list to check if there is any common string being found. For example, common pattern automaton rejects 1188 strings of sample 14, feed 1188 strings into 1st automaton, accepts 902, and then feed 902 into 2nd automaton, accepts 764, and so on, until 49th automaton, ideal result at 49th automaton is 0 string accepted.

Expected Result: No common string found

Actual Result: small amount found, about 0.5% common strings are found

How to read the table:

1) *Rows are sample files; Columns are automatons; Data are string counts; Last row data are common string percentage.*
2) *Each column starts from the number of strings rejected by Common Pattern Automaton;*
3) *Feed the output from Common Pattern Automaton, i.e., rejected strings from Common Pattern Automaton, into 1st author's automaton*
4) *Feed the output from 1st author's automaton, i.e., accepted strings from 1st author's automaton, into next author's automaton, i.e., 2nd author's automaton.*
5) *Keep on feeding output, accepted strings, into next author's automaton until last author in the list, i.e, 49th author's automaton.*
6) *We care how many strings are eventually accepted by 49th author's automaton. Ideal result should be 0.*
7) *Last row is the percentage of common strings found from the set of rejected strings from Common Pattern Automaton. For example: Sample 14 has 1188 strings rejected by Common Pattern Automaton, and at last 12 strings of 1188 are accepted by all 49 in-world automatons, i.e. 12 common strings for 49 in-world automatons. Therefore, the percentage is 12/1188 =1%.*

	Sample 14	sample 15	sample 16	sample 17	sample 18	sample 19	sample 20	sample 21	Sample 22	sample 23	sample 24	sample 25
Cmn ptn automaton rejects	1188	855	1320	1521	164	595	1281	1405	318	1778	919	1707
1 accepts	902	696	1018	1108	121	454	892	965	238	1268	654	1166
2 accepts	764	582	851	900	103	367	719	734	196	1001	526	948
3 accepts	686	519	756	777	89	311	623	617	164	889	462	832
4 accepts	629	476	704	700	81	274	545	531	150	787	411	739
5 accepts	488	378	551	531	63	209	402	380	116	616	315	562
6 accepts	389	301	449	429	51	180	336	327	102	522	259	478
7 accepts	361	274	422	394	48	167	308	301	96	482	234	446
8 accepts	294	218	364	312	46	130	247	227	76	381	183	343
9 accepts	285	211	351	302	43	119	231	217	73	359	176	320
10accepts	272	205	344	281	42	109	220	201	69	342	165	304
11accepts	265	192	327	264	40	100	209	183	65	320	156	289
12accepts	250	185	311	251	37	95	200	174	64	309	146	279
13accepts	233	163	280	232	36	81	175	152	52	278	135	251
14accepts	219	152	262	221	35	76	167	144	49	258	127	226
15accepts	213	150	255	215	35	71	158	136	49	241	122	212
16accepts	205	146	249	199	33	63	147	124	45	223	119	193
17accepts	196	142	244	188	28	61	143	114	45	210	113	186
18accepts	190	139	236	175	25	57	131	112	43	194	107	170
19accepts	186	136	231	170	24	54	123	109	40	186	98	165
20accepts	179	132	227	163	24	54	115	102	40	177	90	159
21accepts	174	125	218	149	22	50	111	93	36	164	85	149
22accepts	160	119	210	139	19	46	105	85	33	149	75	136
23accepts	155	112	206	131	19	44	102	78	32	140	73	131
24accepts	153	109	201	125	18	43	98	76	31	133	70	124
25accepts	148	109	196	120	16	39	96	76	31	129	68	119
26accepts	142	102	190	114	16	37	87	71	31	126	67	115
27accepts	119	91	150	104	16	34	84	65	27	111	58	95
28accepts	114	90	145	97	15	32	79	63	27	109	56	95
29accepts	95	82	125	85	12	28	61	61	22	95	50	90
30accepts	91	79	120	78	10	26	61	57	22	89	47	87
31accepts	88	73	113	71	10	25	60	54	21	80	45	73
32accepts	88	73	113	71	10	25	58	54	21	79	45	73
33accepts	85	72	106	68	9	24	56	51	20	77	43	71
34accepts	81	63	97	61	9	22	51	46	19	74	42	68
35accepts	80	61	91	59	9	20	49	44	18	72	42	64
36accepts	68	53	79	56	7	16	46	41	16	65	39	55
37accepts	64	49	72	54	6	16	39	35	15	61	35	53
38accepts	60	47	71	51	6	15	37	34	15	59	35	53
39accepts	59	47	70	50	6	15	35	34	15	58	35	52
40accepts	56	45	69	47	6	14	32	33	13	56	34	51
41accepts	51	40	60	44	6	14	31	30	12	49	32	47
42accepts	41	33	48	38	4	13	23	24	11	38	26	38
43accepts	39	32	47	32	2	12	20	24	11	36	23	34
44accepts	17	15	12	15	1	6	10	12	5	20	7	16
45accepts	17	13	11	14	0	6	10	12	4	18	6	15
46accepts	16	12	11	14	0	6	7	11	4	16	6	14
47accepts	16	12	10	14	0	5	6	10	3	14	5	12
48accepts	13	9	10	11	0	4	5	9	1	12	5	11
49accepts	12	9	10	10	0	3	5	9	1	10	3	11
Cmn str %	1	1	0.7	0.6	0	0.5	0.4	0.6	0.3	0.56	0.32	0.64

Table 26: Test Data of Common Pattern Automaton Verification Test Case 03 (2)

Appendix D: Test Result of Common Pattern Discerning Ability Test

This appendix is a complete set of Discerning Ability test result by contrasting before removing common strings from sample files and after removing common strings from sample files. The detail description about the test, please refer to section 7.5.2.

We tested 45 sample files, which are from 49 in-world authors but are not used to build 49 in-world automatons, against 49 in-world automatons. And then compare acceptance percentage by its author's (same author's) automaton with acceptance percentage by another author's (different author's) automaton in a pair. So for each sample file, there are 48 pairs of ratio comparisons.

The sample file name is formed as AuthorIndex_filename[AuthorName].pattern.

So 1_TheExtraDay[AlgernonBlackwood].pattern means "The Extra Day" by author "Algernon Blackwood", who is the 1st author in the 49 in-world automatons list.

Note, the sample file was not used to build its author's automaton, or we will see 100% acceptance.

"Automaton" represents automaton index of 49 in-world automatons.

"%_before" means percentage of strings in the sample file was accepted by the "index" automaton before removing common strings.

"%_after" means percentage of strings in the sample file was accepted by the "index" automaton after removing common strings.

"ratio_before" means "same author's automaton acceptance %" is divided by "different author's automaton acceptance %" before removing common strings.

"ratio_after" means "same author's automaton acceptance %" is divided by "different author's automaton acceptance %" after removing common strings.

For example: sample file "the extra day" is from 1st author. 95.38% of strings are accepted by 1st author's automaton before removing common strings; 14.47% of strings are accepted by 1st author's automaton after removing common strings. 94.76% of strings are accepted by 2nd author's automaton before removing common strings; 13.85 of strings are accepted by 2nd author's automaton. "ratio_before" = 95.38/ 94.76 = 1.007; "ratio_after" = 14.47/13.85 = 1.045.

And we can see the ratio is improved since 1.045 is greater than 1.007. Therefore, the ratio becomes sharper to discern different author after removing common strings from sample file.

Positive Results: when Automatons gave "good" results

The following ratios are improved after removing common strings from sample files. These results indicate Common Pattern Automaton is successful when Automatons give successful results.

Test Sample File is from 1st author but not participated in building its author's automaton: 1_TheExtraDay[AlgernonBlackwood].pattern
Before removing common strings, 1st author's automaton accepts 95.38 %
After removing common strings, 1st author's automaton accepts 14.47 %

Automaton	%_before	%_after	ratio_before	ratio_after
2	94.76	13.85	1.007	1.045
3	95.27	14.37	1.001	1.007
5	89.77	8.87	1.062	1.631
6	89.44	8.53	1.066	1.696
7	93.62	12.71	1.019	1.138
8	92.02	11.12	1.037	1.301
10	94.02	13.11	1.014	1.104
11	95.25	14.34	1.001	1.009
12	94.6	13.7	1.008	1.056
13	92.87	11.96	1.027	1.21
14	94.36	13.45	1.011	1.076
15	95.03	14.12	1.004	1.025
16	94.21	13.31	1.012	1.087
17	94.99	14.09	1.004	1.027
18	94.78	13.88	1.006	1.043
20	94.95	14.05	1.005	1.03
21	93.88	12.97	1.016	1.116
22	93.89	12.99	1.016	1.114
24	95.31	14.41	1.001	1.004
25	93.93	13.02	1.015	1.111
26	94.68	13.78	1.007	1.05
27	95.21	14.31	1.002	1.011
29	91.39	10.49	1.044	1.379
30	93.49	12.58	1.02	1.15
31	94.25	13.35	1.012	1.084
33	95.0	14.1	1.004	1.026
34	93.76	12.86	1.017	1.125
35	94.69	13.79	1.007	1.049
36	94.19	13.28	1.013	1.09
37	93.81	12.91	1.017	1.121
38	95.02	14.11	1.004	1.026
39	94.99	14.09	1.004	1.027
42	88.97	8.07	1.072	1.793
44	89.31	8.4	1.068	1.723
46	93.63	12.73	1.019	1.137
47	94.82	13.92	1.006	1.04
48	94.87	13.97	1.005	1.036

Test Sample File is from 2nd author but not participated in building its author's automaton: 2_ShapesofClay[AmbroseBierce].pattern
Before removing common strings, 2nd author's automaton accepts 93.86 %
After removing common strings, 2nd author's automaton accepts 12.93 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	93.59	12.65	1.003	1.022
5	88.61	7.67	1.059	1.686
6	89.53	8.59	1.048	1.505
7	93.18	12.24	1.007	1.056
8	89.67	8.73	1.047	1.481
13	92.33	11.39	1.017	1.135
14	93.49	12.55	1.004	1.03
16	93.69	12.76	1.002	1.013
18	93.52	12.59	1.004	1.027
21	92.91	11.97	1.01	1.08
22	93.18	12.24	1.007	1.056
24	93.62	12.69	1.003	1.019
25	93.04	12.11	1.009	1.068
26	93.83	12.89	1	1.003
29	90.76	9.82	1.034	1.317
30	92.84	11.9	1.011	1.087
31	92.8	11.87	1.011	1.089
34	93.01	12.07	1.009	1.071
35	93.45	12.52	1.004	1.033
36	93.79	12.86	1.001	1.005
37	92.19	11.26	1.018	1.148
38	93.62	12.69	1.003	1.019
40	93.62	12.69	1.003	1.019
42	87.35	6.41	1.075	2.017
44	89.43	8.49	1.05	1.523
46	91.37	10.44	1.027	1.239
47	93.72	12.79	1.001	1.011
48	93.52	12.59	1.004	1.027

Test Sample File is from 3rd author but not participated in building its author's automaton: 3_RemembertheAlamo[AmeliaEdithHuddlestonBarr].pattern
Before removing common strings, 3rd author's automaton accepts 94.83 %
After removing common strings, 3rd author's automaton accepts 15.56 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	94.56	15.29	1.003	1.018
2	94.36	15.09	1.005	1.031
5	87.89	8.62	1.079	1.805
6	88.83	9.55	1.068	1.629
7	92.81	13.54	1.022	1.149
8	90.41	11.14	1.049	1.397
10	94.5	15.22	1.003	1.022
11	94.16	14.89	1.007	1.045
12	93.68	14.41	1.012	1.08
13	92.16	12.89	1.029	1.207
14	93.8	14.52	1.011	1.072
15	93.53	14.26	1.014	1.091
16	93.25	13.97	1.017	1.114
17	94.5	15.22	1.003	1.022
18	94.4	15.12	1.005	1.029
20	94.16	14.89	1.007	1.045
21	92.78	13.51	1.022	1.152
22	92.9	13.62	1.021	1.142
24	94.66	15.39	1.002	1.011
25	93.48	14.21	1.014	1.095
26	93.95	14.67	1.009	1.061
27	93.65	14.37	1.013	1.083
29	90.68	11.41	1.046	1.364
30	93.11	13.84	1.018	1.124
31	92.65	13.37	1.024	1.164
33	94.5	15.22	1.003	1.022
34	93.33	14.06	1.016	1.107
35	93.7	14.42	1.012	1.079
36	93.71	14.44	1.012	1.078

37	92.75	13.47	1.022	1.155
39	94.08	14.81	1.008	1.051
42	87.31	8.04	1.086	1.935
43	94.38	15.11	1.005	1.03
44	88.79	9.52	1.068	1.634
46	92.88	13.61	1.021	1.143
47	93.63	14.36	1.013	1.084
48	94.45	15.17	1.004	1.026

Test Sample File is from 4th author but not participated in building its author's automaton: 4_TheLogofaCowboy[AndyAdams].pattern
Before removing common strings, 4th author's automaton accepts 87.88 %
After removing common strings, 4th author's automaton accepts 34.13 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	80.01	26.26	1.098	1.3
2	77.84	24.09	1.129	1.417
3	79.86	26.11	1.1	1.307
5	68.87	15.12	1.276	2.257
6	75.06	21.31	1.171	1.602
7	74.81	21.06	1.175	1.621
8	70.26	16.52	1.251	2.066
9	81.63	27.89	1.077	1.224
10	77.14	23.39	1.139	1.459
11	80.43	26.69	1.093	1.279
12	77.29	23.54	1.137	1.45
13	75.51	21.76	1.164	1.568
14	78.96	25.21	1.113	1.354
15	78.04	24.29	1.126	1.405
16	77.14	23.39	1.139	1.459
17	81.78	28.04	1.075	1.217
18	79.36	25.61	1.107	1.333
19	81.86	28.11	1.074	1.214
20	79.96	26.21	1.099	1.302
21	75.76	22.01	1.16	1.551
22	79.81	26.06	1.101	1.31
23	85.58	31.83	1.027	1.072
24	82.33	28.59	1.067	1.194
25	75.24	21.49	1.168	1.588
26	78.31	24.56	1.122	1.39
27	84.93	31.18	1.035	1.095
28	82.23	28.49	1.069	1.198
29	72.81	19.07	1.207	1.79
30	75.29	21.54	1.167	1.584
31	77.21	23.46	1.138	1.455
32	82.63	28.89	1.064	1.181
33	79.24	25.49	1.109	1.339
34	78.51	24.76	1.119	1.378
35	77.74	23.99	1.13	1.423
36	78.56	24.81	1.119	1.376
37	77.34	23.59	1.136	1.447
38	80.28	26.54	1.095	1.286
39	77.96	24.21	1.127	1.41
40	83.11	29.36	1.057	1.162
41	82.31	28.56	1.068	1.195
42	66.79	13.04	1.316	2.617
43	81.26	27.51	1.081	1.241
44	70.84	17.09	1.241	1.997
45	84.11	30.36	1.045	1.124
46	72.69	18.94	1.209	1.802
47	78.74	24.99	1.116	1.366
48	80.08	26.34	1.097	1.296
49	81.21	27.46	1.082	1.243

Test Sample File is from 5th author but not participated in building its author's automaton: 5_TheCaseforIndia[AnnieWoodBesant].pattern
Before removing common string, 5th author's automaton accepts 83.07 %
After removing common string, 5th author's automaton accepts 7.0 %

Automaton	%_before	%_after	ratio_before	ratio_after
8	82.13	6.06	1.011	1.155
42	80.88	4.81	1.027	1.455
44	82.24	6.17	1.01	1.135

Test Sample File is from 6th author but not participated in building its author's automaton: 6_PoliticsATreatiseonGovernment[Aristotle].pattern
Before removing common string, 6th author's automaton accepts 59.87 %
After removing common string, 6th author's automaton accepts 13.69 %

Automaton	%_before	%_after	ratio_before	ratio_after
2	59.57	13.39	1.005	1.022
5	49.28	3.1	1.215	4.416
8	52.62	6.45	1.138	2.122
13	58.22	12.04	1.028	1.137
18	59.27	13.09	1.01	1.046
21	57.22	11.04	1.046	1.24
25	56.57	10.39	1.058	1.318
29	56.87	10.69	1.053	1.281
30	57.27	11.09	1.045	1.234
37	57.57	11.39	1.04	1.202
40	59.82	13.64	1.001	1.004
42	47.48	1.3	1.261	10.531
44	55.32	9.15	1.082	1.496
46	55.22	9.05	1.084	1.513

Test Sample File is from 7th author but not participated in building its author's automaton: 7_FatherPayne[ArthurChristopherBenson].pattern
Before removing common string, 7th author's automaton accepts 90.68 %
After removing common string, 7th author's automaton accepts 16.81 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	82.4	8.52	1.1	1.973
6	86.04	12.16	1.054	1.382
8	85.66	11.78	1.059	1.427
12	90.41	16.53	1.003	1.017
13	87.54	13.67	1.036	1.23
14	89.86	15.98	1.009	1.052
16	89.72	15.84	1.011	1.061
18	89.89	16.01	1.009	1.05
21	89.03	15.15	1.019	1.11
22	89.23	15.35	1.016	1.095
25	90.12	16.24	1.006	1.035
29	87.39	13.51	1.038	1.244
30	88.66	14.78	1.023	1.137
31	89.38	15.5	1.015	1.085
33	90.24	16.36	1.005	1.028
34	89.29	15.41	1.016	1.091
35	90.35	16.47	1.004	1.021
37	89.31	15.43	1.015	1.089
42	80.42	6.54	1.128	2.57
44	83.18	9.3	1.09	1.808
46	87.82	13.94	1.033	1.206
47	90.39	16.52	1.003	1.018
48	90.52	16.64	1.002	1.01

Test Sample File is from 8th author but not participated in building its author's automaton: 8_TheTaleofCuffyBear[ArthurScottBailey].pattern
Before removing common string, 8th author's automaton accepts 92.01 %
After removing common string, 8th author's automaton accepts 8.79 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	89.93	6.71	1.023	1.31
6	89.06	5.84	1.033	1.505
42	89.4	6.17	1.029	1.425
44	89.73	6.51	1.025	1.35

Test Sample File is from 9th author but not participated in building its author's automaton: 9_HerPrairieKnight[BMBower].pattern
Before removing common string, 9th author's automaton accepts 97.36 %
After removing common string, 9th author's automaton accepts 11.63 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	97.07	11.34	1.003	1.026
2	96.42	10.69	1.01	1.088
3	96.68	10.94	1.007	1.063
4	96.75	11.01	1.006	1.056
5	92.31	6.57	1.055	1.77
6	92.13	6.39	1.057	1.82
7	96.39	10.65	1.01	1.092
8	94.08	8.34	1.035	1.394
10	96.82	11.09	1.006	1.049
11	96.64	10.91	1.007	1.066
12	97.0	11.27	1.004	1.032
13	94.98	9.25	1.025	1.257
14	96.53	10.8	1.009	1.077
15	97.22	11.48	1.001	1.013
16	96.68	10.94	1.007	1.063
17	96.82	11.09	1.006	1.049
18	96.61	10.87	1.008	1.07
19	97.22	11.48	1.001	1.013
21	96.28	10.55	1.011	1.102
22	95.7	9.97	1.017	1.166
23	97.07	11.34	1.003	1.026
24	97.07	11.34	1.003	1.026
25	96.06	10.33	1.014	1.126
26	97.07	11.34	1.003	1.026
27	96.39	10.65	1.01	1.092
28	97.0	11.27	1.004	1.032
29	93.75	8.02	1.039	1.45
30	96.03	10.29	1.014	1.13
31	96.28	10.55	1.011	1.102
32	97.33	11.59	1	1.003
33	96.28	10.55	1.011	1.102
34	95.88	10.15	1.015	1.146
35	96.21	10.47	1.012	1.111
36	96.68	10.94	1.007	1.063
37	96.5	10.76	1.009	1.081
38	96.79	11.05	1.006	1.052
39	96.71	10.98	1.007	1.059
40	96.61	10.87	1.008	1.07
41	97.07	11.34	1.003	1.026
42	92.05	6.32	1.058	1.84
43	97.18	11.45	1.002	1.016
44	92.7	6.97	1.05	1.669
46	95.67	9.93	1.018	1.171
47	96.5	10.76	1.009	1.081
48	96.64	10.91	1.007	1.066
49	97.15	11.41	1.002	1.019

Test Sample File is from 10th author but not participated in building its author's automaton: 10_WielandortheTransformationanAmericanTale[CharlesBrockdenBrown].pattern
Before removing common string, 10th author's automaton accepts 95.66 %
After removing common string, 10th author's automaton accepts 15.06 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	95.04	14.43	1.007	1.044
2	94.59	13.99	1.011	1.076
3	95.42	14.82	1.003	1.016
5	88.58	7.97	1.08	1.89
6	90.35	9.74	1.059	1.546
7	93.76	13.15	1.02	1.145
8	89.85	9.24	1.065	1.63
11	95.0	14.39	1.007	1.047
12	95.59	14.98	1.001	1.005
13	92.86	12.25	1.03	1.229
14	94.94	14.34	1.008	1.05
15	94.76	14.15	1.009	1.064
16	94.37	13.76	1.014	1.094
17	95.37	14.76	1.003	1.02
18	94.8	14.19	1.009	1.061
19	95.48	14.87	1.002	1.013
20	95.44	14.83	1.002	1.016
21	93.76	13.15	1.02	1.145
22	93.78	13.17	1.02	1.144
24	95.28	14.67	1.004	1.027
25	94.94	14.34	1.008	1.05
26	94.65	14.04	1.011	1.073
27	94.56	13.95	1.012	1.08
29	90.48	9.87	1.057	1.526
30	93.62	13.01	1.022	1.158
31	94.58	13.97	1.011	1.078
33	94.83	14.23	1.009	1.058
34	93.86	13.25	1.019	1.137
35	95.0	14.39	1.007	1.047
36	94.65	14.04	1.011	1.073
37	94.28	13.67	1.015	1.102
39	95.33	14.72	1.003	1.023
40	95.13	14.52	1.006	1.037
42	87.44	6.83	1.094	2.205
43	95.33	14.72	1.003	1.023
44	89.5	8.89	1.069	1.694
46	93.51	12.9	1.023	1.167
47	94.87	14.26	1.008	1.056
48	95.3	14.69	1.004	1.025

Test Sample File is from 11th author but not participated in building its author's automaton: 11_JaneEyre[CharlotteBronte].pattern
Before removing common string, 11th author's automaton accepts 90.72 %
After removing common string, 11th author's automaton accepts 14.84 %

Automaton	%_before	%_after	ratio_before	ratio_after
2	90.55	14.67	1.002	1.012
5	82.74	6.86	1.096	2.163
6	85.95	10.08	1.055	1.472
7	88.78	12.9	1.022	1.15
8	85.3	9.43	1.064	1.574
10	89.65	13.78	1.012	1.077
12	89.98	14.1	1.008	1.052
13	87.67	11.79	1.035	1.259
14	90.19	14.31	1.006	1.037
15	90.31	14.44	1.005	1.028
16	89.72	13.84	1.011	1.072
18	89.9	14.02	1.009	1.058
21	88.87	12.99	1.021	1.142

22	90.31	14.44	1.005	1.028
24	90.37	14.49	1.004	1.024
25	88.46	12.59	1.026	1.179
26	90.11	14.24	1.007	1.042
29	86.53	10.65	1.048	1.393
30	88.46	12.59	1.026	1.179
31	89.4	13.53	1.015	1.097
33	90.11	14.24	1.007	1.042
34	89.33	13.45	1.016	1.103
35	89.96	14.09	1.008	1.053
36	90.12	14.25	1.007	1.041
37	89.07	13.2	1.019	1.124
39	90.47	14.59	1.003	1.017
42	81.4	5.53	1.114	2.684
44	84.97	9.09	1.068	1.633
46	87.32	11.45	1.039	1.296
47	89.76	13.89	1.011	1.068
48	90.28	14.41	1.005	1.03

Test Sample File is from 12th author but not participated in building its author's automaton: 12_Coralie[CharlotteMCharlotteMonicaBrame].pattern
Before removing common string, 12th author's automaton accepts 93.58 %
After removing common string, 12th author's automaton accepts 13.01 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	88.32	7.75	1.06	1.679
6	89.44	8.87	1.046	1.467
7	93.18	12.61	1.004	1.032
8	91.04	10.47	1.028	1.243
13	91.76	11.19	1.02	1.163
14	93.14	12.57	1.005	1.035
16	93.23	12.66	1.004	1.028
21	93.32	12.75	1.003	1.02
22	92.69	12.12	1.01	1.073
29	90.86	10.29	1.03	1.264
30	93.49	12.92	1.001	1.007
31	93.45	12.88	1.001	1.01
37	92.42	11.85	1.013	1.098
42	88.15	7.58	1.062	1.716
44	88.59	8.02	1.056	1.622
46	93.14	12.57	1.005	1.035

Test Sample File is from 14th author but not participated in building its author's automaton: 14_TheMucker[EdgarRiceBurroughs].pattern
Before removing common string, 14th author's automaton accepts 91.19 %
After removing common string, 14th author's automaton accepts 13.78 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	85.43	8.03	1.067	1.716
6	87.03	9.62	1.048	1.432
7	90.2	12.79	1.011	1.077
8	88.15	10.74	1.034	1.283
12	91.1	13.69	1.001	1.007
13	89.42	12.02	1.02	1.146
16	91.0	13.59	1.002	1.014
18	91.05	13.64	1.002	1.01
21	90.69	13.28	1.006	1.038
25	89.9	12.5	1.014	1.102
29	87.79	10.38	1.039	1.328
30	89.82	12.42	1.015	1.11
31	90.6	13.19	1.007	1.045
34	90.71	13.31	1.005	1.035
36	91.01	13.6	1.002	1.013
37	90.78	13.37	1.005	1.031

42	83.9	6.49	1.087	2.123
44	86.11	8.71	1.059	1.582
46	89.31	11.9	1.021	1.158

Test Sample File is from 16th author but not participated in building its author's automaton: 16_TTembarom[FrancesHodgsonBurnett].pattern
Before removing common string, 16th author's automaton accepts 92.95 %
After removing common string, 16th author's automaton accepts 12.59 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	87.74	7.38	1.059	1.706
6	89.12	8.76	1.043	1.437
7	92.4	12.05	1.006	1.045
8	90.05	9.7	1.032	1.298
13	91.9	11.55	1.011	1.09
21	92.69	12.33	1.003	1.021
22	92.7	12.35	1.003	1.019
29	90.45	10.09	1.028	1.248
30	92.26	11.9	1.007	1.058
31	92.62	12.26	1.004	1.027
37	92.58	12.22	1.004	1.03
42	87.28	6.93	1.065	1.817
44	88.16	7.81	1.054	1.612
46	91.96	11.6	1.011	1.085

Test Sample File is from 17th author but not participated in building its author's automaton: 17_TheRomanyRye[GeorgeHenryBorrow].pattern
Before removing common string, 17th author's automaton accepts 84.24 %
After removing common string, 17th author's automaton accepts 16.67 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	83.01	15.44	1.015	1.08
2	81.78	14.21	1.03	1.173
3	83.51	15.95	1.009	1.045
5	74.53	6.96	1.13	2.395
6	78.86	11.29	1.068	1.477
7	80.81	13.25	1.042	1.258
8	76.14	8.57	1.106	1.945
10	82.39	14.83	1.022	1.124
11	82.69	15.13	1.019	1.102
12	81.87	14.31	1.029	1.165
13	80.77	13.2	1.043	1.263
14	82.17	14.61	1.025	1.141
15	83.31	15.74	1.011	1.059
16	82.25	14.68	1.024	1.136
18	81.87	14.31	1.029	1.165
20	83.23	15.66	1.012	1.064
21	80.33	12.76	1.049	1.306
22	82.87	15.3	1.017	1.09
24	82.03	14.46	1.027	1.153
25	80.83	13.26	1.042	1.257
26	82.96	15.4	1.015	1.082
29	78.29	10.72	1.076	1.555
30	80.14	12.57	1.051	1.326
31	81.87	14.31	1.029	1.165
32	83.09	15.52	1.014	1.074
33	83.07	15.51	1.014	1.075
34	82.05	14.48	1.027	1.151
35	81.98	14.42	1.028	1.156
36	83.12	15.55	1.013	1.072
37	81.24	13.67	1.037	1.219
38	83.21	15.65	1.012	1.065
39	81.95	14.38	1.028	1.159
40	82.84	15.27	1.017	1.092

42	71.07	3.51	1.185	4.749
43	83.93	16.36	1.004	1.019
44	76.87	9.3	1.096	1.792
46	78.18	10.61	1.078	1.571
47	81.78	14.21	1.03	1.173
48	82.16	14.59	1.025	1.143

Test Sample File is from 18th author but not participated in building its author's automaton:
18_TheSisters-In-Law[GertrudeFranklinHornAtherton].pattern
Before removing common string, 18th author's automaton accepts 92.66 %
After removing common string, 18th author's automaton accepts 12.12 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	87.63	7.08	1.057	1.712
6	88.08	7.53	1.052	1.61
7	91.81	11.26	1.009	1.076
8	89.39	8.85	1.037	1.369
10	92.54	12.0	1.001	1.01
13	91.3	10.76	1.015	1.126
16	92.1	11.56	1.006	1.048
21	92.15	11.6	1.006	1.045
22	92.63	12.08	1	1.003
25	91.87	11.32	1.009	1.071
29	90.06	9.51	1.029	1.274
30	91.84	11.29	1.009	1.074
34	92.23	11.69	1.005	1.037
36	92.37	11.83	1.003	1.025
37	92.39	11.85	1.003	1.023
42	86.7	6.15	1.069	1.971
44	88.71	8.16	1.045	1.485
46	91.31	10.77	1.015	1.125

Test Sample File is from 19th author but not participated in building its author's automaton: 19_HildaWadeaWomanwithTenacityofPurpose[GrantAllen].pattern
Before removing common string, 19th author's automaton accepts 95.64 %
After removing common string, 19th author's automaton accepts 13.95 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	94.77	13.09	1.009	1.066
2	94.76	13.08	1.009	1.067
3	95.03	13.34	1.006	1.046
5	89.68	8.0	1.066	1.744
6	89.71	8.02	1.066	1.739
7	92.85	11.17	1.03	1.249
8	91.82	10.13	1.042	1.377
9	95.53	13.85	1.001	1.007
10	94.26	12.57	1.015	1.11
11	94.76	13.08	1.009	1.067
12	94.07	12.39	1.017	1.126
13	93.02	11.34	1.028	1.23
14	94.1	12.41	1.016	1.124
15	94.43	12.75	1.013	1.094
16	93.93	12.24	1.018	1.14
17	94.97	13.29	1.007	1.05
18	94.43	12.75	1.013	1.094
20	94.73	13.05	1.01	1.069
21	93.93	12.24	1.018	1.14
22	93.63	11.95	1.021	1.167
24	95.37	13.69	1.003	1.019
25	93.63	11.95	1.021	1.167
26	94.42	12.73	1.013	1.096
27	95.2	13.51	1.005	1.033
28	95.36	13.67	1.003	1.02

29	91.55	9.87	1.045	1.413
30	93.77	12.08	1.02	1.155
31	93.66	11.98	1.021	1.164
32	95.62	13.94	1	1.001
33	94.67	12.98	1.01	1.075
34	94.38	12.69	1.013	1.099
35	94.22	12.53	1.015	1.113
36	94.54	12.85	1.012	1.086
37	93.7	12.02	1.021	1.161
38	94.46	12.77	1.012	1.092
39	94.87	13.18	1.008	1.058
40	95.57	13.89	1.001	1.004
41	95.31	13.62	1.003	1.024
42	89.31	7.63	1.071	1.828
43	95.48	13.79	1.002	1.012
44	89.84	8.16	1.065	1.71
46	92.9	11.22	1.029	1.243
47	94.51	12.82	1.012	1.088
48	94.34	12.65	1.014	1.103

Test Sample File is from 20th author but not participated in building its author's automaton: 20_PicturesofSweden[HansChristianAndersen].pattern
Before removing common string, 20th author's automaton accepts 90.23 %
After removing common string, 20th author's automaton accepts 19.49 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	89.4	18.65	1.009	1.045
2	88.65	17.91	1.018	1.088
3	88.93	18.19	1.015	1.071
5	80.0	9.26	1.128	2.105
6	85.72	14.98	1.053	1.301
7	87.16	16.42	1.035	1.187
8	82.19	11.44	1.098	1.704
9	90.09	19.35	1.002	1.007
10	87.53	16.79	1.031	1.161
11	88.6	17.86	1.018	1.091
12	88.37	17.63	1.021	1.106
13	85.72	14.98	1.053	1.301
14	87.3	16.56	1.034	1.177
15	88.93	18.19	1.015	1.071
16	88.0	17.26	1.025	1.129
17	89.63	18.88	1.007	1.032
18	88.93	18.19	1.015	1.071
21	87.16	16.42	1.035	1.187
22	88.09	17.35	1.024	1.123
24	88.14	17.4	1.024	1.12
25	85.91	15.16	1.05	1.286
26	88.56	17.81	1.019	1.094
29	85.4	14.65	1.057	1.33
30	86.23	15.49	1.046	1.258
31	88.14	17.4	1.024	1.12
32	89.53	18.79	1.008	1.037
33	88.88	18.14	1.015	1.074
34	87.02	16.28	1.037	1.197
35	89.26	18.51	1.011	1.053
36	89.26	18.51	1.011	1.053
37	88.74	18.0	1.017	1.083
38	89.26	18.51	1.011	1.053
39	89.4	18.65	1.009	1.045
40	88.6	17.86	1.018	1.091
41	89.63	18.88	1.007	1.032
42	79.44	8.7	1.136	2.24
43	89.12	18.37	1.012	1.061
44	83.07	12.33	1.086	1.581
46	85.12	14.37	1.06	1.356

47	88.33	17.58	1.022	1.109
48	87.77	17.02	1.028	1.145
49	89.86	19.12	1.004	1.019

Test Sample File is from 21st author but not participated in building its author's automaton: 21_RanchingforSylvia[HaroldBindloss].pattern
Before removing common string, 21st author's automaton accepts 93.41 %
After removing common string, 21st author's automaton accepts 13.23 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	87.96	7.79	1.062	1.698
6	88.47	8.3	1.056	1.594
7	92.58	12.41	1.009	1.066
8	90.26	10.09	1.035	1.311
13	92.21	12.04	1.013	1.099
29	90.52	10.35	1.032	1.278
30	92.71	12.54	1.008	1.055
31	92.99	12.82	1.005	1.032
34	92.87	12.69	1.006	1.043
35	93.08	12.91	1.004	1.025
37	92.68	12.51	1.008	1.058
42	86.54	6.37	1.079	2.077
44	88.27	8.1	1.058	1.633
46	92.4	12.22	1.011	1.083

Test Sample File is from 22nd author but not participated in building its author's automaton: 22_ThePathtoRome[HilaireBelloc].pattern
Before removing common string, 22nd author's automaton accepts 84.54 %
After removing common string, 22nd author's automaton accepts 19.27 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	83.31	18.04	1.015	1.068
2	83.36	18.08	1.014	1.066
3	84.52	19.24	1	1.002
5	75.37	10.1	1.122	1.908
6	79.43	14.16	1.064	1.361
7	81.38	16.11	1.039	1.196
8	77.25	11.98	1.094	1.609
10	82.61	17.34	1.023	1.111
11	83.89	18.62	1.008	1.035
12	82.54	17.27	1.024	1.116
13	81.24	15.97	1.041	1.207
14	83.45	18.18	1.013	1.06
15	83.45	18.18	1.013	1.06
16	82.82	17.55	1.021	1.098
18	83.59	18.31	1.011	1.052
20	84.26	18.99	1.003	1.015
21	81.24	15.97	1.041	1.207
24	84.26	18.99	1.003	1.015
25	79.92	14.65	1.058	1.315
26	82.89	17.62	1.02	1.094
29	79.6	14.32	1.062	1.346
30	80.5	15.23	1.05	1.265
31	82.82	17.55	1.021	1.098
33	84.29	19.01	1.003	1.014
34	82.38	17.11	1.026	1.126
35	83.08	17.8	1.018	1.083
36	82.92	17.64	1.02	1.092
37	81.31	16.04	1.04	1.201
38	83.59	18.31	1.011	1.052
39	83.68	18.41	1.01	1.047
40	83.94	18.66	1.007	1.033
42	72.73	7.45	1.162	2.587
44	77.6	12.33	1.089	1.563

46	79.18	13.9	1.068	1.386
47	82.92	17.64	1.02	1.092
48	83.91	18.64	1.008	1.034

Test Sample File is from 23rd author but not participated in building its author's automaton: 23_LostIllusions[HonordeBalzac].pattern
Before removing common string, 23rd author's automaton accepts 96.61 %
After removing common string, 23rd author's automaton accepts 19.26 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	91.7	14.35	1.054	1.342
2	91.02	13.67	1.061	1.409
3	92.33	14.97	1.046	1.287
4	93.95	16.59	1.028	1.161
5	85.41	8.06	1.131	2.39
6	87.24	9.88	1.107	1.949
7	90.27	12.91	1.07	1.492
8	86.67	9.31	1.115	2.069
9	92.41	15.05	1.045	1.28
10	91.19	13.83	1.059	1.393
11	91.81	14.45	1.052	1.333
12	90.93	13.58	1.062	1.418
13	89.65	12.29	1.078	1.567
14	91.07	13.71	1.061	1.405
15	91.56	14.2	1.055	1.356
16	90.59	13.23	1.066	1.456
17	92.46	15.11	1.045	1.275
18	91.91	14.56	1.051	1.323
19	92.28	14.92	1.047	1.291
20	91.96	14.6	1.051	1.319
21	90.3	12.95	1.07	1.487
22	91.23	13.88	1.059	1.388
24	92.46	15.1	1.045	1.275
25	90.1	12.74	1.072	1.512
26	92.0	14.65	1.05	1.315
27	92.91	15.55	1.04	1.239
28	92.92	15.57	1.04	1.237
29	87.84	10.48	1.1	1.838
30	89.68	12.33	1.077	1.562
31	90.87	13.52	1.063	1.425
32	92.91	15.55	1.04	1.239
33	91.64	14.28	1.054	1.349
34	90.57	13.21	1.067	1.458
35	91.12	13.76	1.06	1.4
36	91.29	13.94	1.058	1.382
37	90.74	13.39	1.065	1.438
38	91.75	14.4	1.053	1.338
39	91.35	14.0	1.058	1.376
40	92.29	14.94	1.047	1.289
41	92.75	15.4	1.042	1.251
42	84.23	6.88	1.147	2.799
43	92.13	14.77	1.049	1.304
44	86.58	9.23	1.116	2.087
45	93.38	16.03	1.035	1.201
46	88.83	11.48	1.088	1.678
47	91.09	13.73	1.061	1.403
48	91.22	13.86	1.059	1.39
49	92.43	15.07	1.045	1.278

Test Sample File is from 24th author but not participated in building its author's automaton: 24_PaulPrescottsCharge[HoratioAlger].pattern
Before removing common string, 24th author's automaton accepts 96.16 %
After removing common string, 24th author's automaton accepts 12.39 %

Automaton	%_before	%_after	ratio_before	ratio_after

1	95.81	12.04	1.004	1.029
2	95.57	11.8	1.006	1.05
3	96.11	12.34	1.001	1.004
5	90.26	6.49	1.065	1.909
6	90.15	6.38	1.067	1.942
7	94.03	10.25	1.023	1.209
8	92.88	9.11	1.035	1.36
9	96.15	12.37	1	1.002
10	95.52	11.75	1.007	1.054
11	95.09	11.32	1.011	1.095
12	94.54	10.77	1.017	1.15
13	94.36	10.58	1.019	1.171
14	94.78	11.01	1.015	1.125
15	95.19	11.41	1.01	1.086
16	94.8	11.03	1.014	1.123
17	95.85	12.08	1.003	1.026
18	95.85	12.08	1.003	1.026
19	95.63	11.86	1.006	1.045
20	96.09	12.32	1.001	1.006
21	95.02	11.25	1.012	1.101
22	93.75	9.98	1.026	1.241
25	94.87	11.1	1.014	1.116
27	95.69	11.91	1.005	1.04
28	96.07	12.3	1.001	1.007
29	92.05	8.28	1.045	1.496
30	94.39	10.62	1.019	1.167
31	94.74	10.97	1.015	1.129
33	95.83	12.06	1.003	1.027
34	94.98	11.21	1.012	1.105
35	95.33	11.56	1.009	1.072
36	94.71	10.93	1.015	1.134
37	94.39	10.62	1.019	1.167
38	95.65	11.88	1.005	1.043
39	95.67	11.89	1.005	1.042
42	90.13	6.36	1.067	1.948
44	90.47	6.69	1.063	1.852
46	94.28	10.51	1.02	1.179
47	95.22	11.45	1.01	1.082
48	95.41	11.64	1.008	1.064

Test Sample File is from 25th author but not participated in building its author's automaton: 25_TheGayCockade[IreneTempleBailey].pattern
Before removing common string, 25th author's automaton accepts 95.47 %
After removing common string, 25th author's automaton accepts 11.72 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	90.13	6.38	1.059	1.837
6	90.31	6.56	1.057	1.787
7	94.42	10.67	1.011	1.098
8	92.65	8.9	1.03	1.317
11	95.15	11.4	1.003	1.028
13	94.1	10.35	1.015	1.132
14	95.04	11.29	1.005	1.038
16	94.89	11.14	1.006	1.052
18	95.4	11.65	1.001	1.006
21	95.01	11.26	1.005	1.041
22	94.31	10.56	1.012	1.11
27	95.45	11.71	1	1.001
29	92.99	9.25	1.027	1.267
30	95.15	11.4	1.003	1.028
31	94.8	11.06	1.007	1.06
34	95.12	11.37	1.004	1.031
36	95.16	11.42	1.003	1.026
37	94.71	10.96	1.008	1.069

42	89.57	5.82	1.066	2.014
44	90.81	7.06	1.051	1.66
46	94.86	11.11	1.006	1.055

Test Sample File is from 26th author but not participated in building its author's automaton: 26_EbenHoldenataleofthenorthcountry[IrvingBacheller].pattern
Before removing common string, 26th author's automaton accepts 95.14 %
After removing common string, 26th author's automaton accepts 13.34 %

Automaton	%_before	%_after	ratio_before	ratio_after
2	95.01	13.21	1.001	1.01
5	89.88	8.08	1.059	1.651
6	90.7	8.9	1.049	1.499
7	94.12	12.32	1.011	1.083
8	91.65	9.85	1.038	1.354
10	94.79	12.99	1.004	1.027
13	93.08	11.28	1.022	1.183
14	94.33	12.53	1.009	1.065
15	94.65	12.84	1.005	1.039
16	94.79	12.99	1.004	1.027
21	94.17	12.37	1.01	1.078
22	94.52	12.72	1.007	1.049
25	94.31	12.51	1.009	1.066
29	91.67	9.87	1.038	1.352
30	93.6	11.8	1.016	1.131
31	93.9	12.1	1.013	1.102
34	94.27	12.46	1.009	1.071
35	95.07	13.27	1.001	1.005
36	94.68	12.88	1.005	1.036
37	93.78	11.97	1.015	1.114
39	94.85	13.05	1.003	1.022
42	88.93	7.13	1.07	1.871
44	90.16	8.36	1.055	1.596
46	93.65	11.85	1.016	1.126
47	95.12	13.32	1	1.002

Test Sample File is from 27th author but not participated in building its author's automaton: 27_TheGoldenChersoneseandtheWayThither[IsabellaLIsabellaLucyBird].pattern
Before removing common string, 27th author's automaton accepts 85.56 %
After removing common string, 27th author's automaton accepts 24.51 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	81.71	20.65	1.047	1.187
2	81.07	20.02	1.055	1.224
3	83.36	22.31	1.026	1.099
5	72.07	11.01	1.187	2.226
6	77.82	16.77	1.099	1.462
7	79.83	18.77	1.072	1.306
8	74.68	13.63	1.146	1.798
9	82.75	21.7	1.034	1.129
10	80.77	19.71	1.059	1.244
11	82.08	21.02	1.042	1.166
12	80.31	19.25	1.065	1.273
13	78.11	17.05	1.095	1.438
14	81.03	19.97	1.056	1.227
15	81.68	20.63	1.048	1.188
16	79.02	17.97	1.083	1.364
17	83.91	22.85	1.02	1.073
18	81.84	20.78	1.045	1.179
19	83.95	22.9	1.019	1.07
20	81.16	20.1	1.054	1.219
21	78.48	17.42	1.09	1.407
22	82.64	21.59	1.035	1.135
23	85.52	24.47	1	1.002

24	82.66	21.61	1.035	1.134
25	79.2	18.14	1.08	1.351
26	82.4	21.35	1.038	1.148
28	83.89	22.83	1.02	1.074
29	76.41	15.35	1.12	1.597
30	77.63	16.57	1.102	1.479
31	80.42	19.36	1.064	1.266
32	83.62	22.57	1.023	1.086
33	81.9	20.85	1.045	1.176
34	81.44	20.39	1.051	1.202
35	81.09	20.04	1.055	1.223
36	80.44	19.39	1.064	1.264
37	79.87	18.82	1.071	1.302
38	82.36	21.3	1.039	1.151
39	80.53	19.47	1.062	1.259
40	82.53	21.48	1.037	1.141
41	83.99	22.94	1.019	1.068
42	70.85	9.79	1.208	2.504
43	83.54	22.48	1.024	1.09
44	75.58	14.52	1.132	1.688
45	85.52	24.47	1	1.002
46	76.34	15.29	1.121	1.603
47	80.51	19.45	1.063	1.26
48	80.88	19.82	1.058	1.237
49	83.58	22.53	1.024	1.088

Test Sample File is from 28th author but not participated in building its author's automaton: 28_TommyandGrizel[JamesMatthewBarrie].pattern
Before removing common string, 28th author's automaton accepts 92.98 %
After removing common string, 28th author's automaton accepts 14.77 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	92.29	14.09	1.007	1.048
2	91.65	13.45	1.015	1.098
3	92.62	14.41	1.004	1.025
5	85.08	6.87	1.093	2.15
6	86.85	8.64	1.071	1.709
7	90.87	12.67	1.023	1.166
8	88.04	9.84	1.056	1.501
10	92.06	13.85	1.01	1.066
11	92.11	13.9	1.009	1.063
12	92.19	13.98	1.009	1.057
13	89.86	11.65	1.035	1.268
14	91.66	13.46	1.014	1.097
15	92.39	14.18	1.006	1.042
16	91.01	12.81	1.022	1.153
17	92.4	14.19	1.006	1.041
18	91.57	13.36	1.015	1.106
20	92.49	14.28	1.005	1.034
21	90.88	12.68	1.023	1.165
22	90.51	12.31	1.027	1.2
24	91.9	13.69	1.012	1.079
25	91.14	12.93	1.02	1.142
26	91.83	13.62	1.013	1.084
27	92.91	14.7	1.001	1.005
29	88.28	10.07	1.053	1.467
30	90.83	12.62	1.024	1.17
31	91.47	13.26	1.017	1.114
33	92.08	13.88	1.01	1.064
34	90.99	12.78	1.022	1.156
35	91.63	13.42	1.015	1.101
36	91.98	13.77	1.011	1.073
37	90.06	11.85	1.032	1.246
38	92.63	14.42	1.004	1.024
39	92.46	14.25	1.006	1.036

40	92.71	14.51	1.003	1.018
41	92.81	14.6	1.002	1.012
42	83.54	5.33	1.113	2.771
44	85.9	7.69	1.082	1.921
46	89.62	11.41	1.037	1.294
47	92.01	13.81	1.011	1.07
48	92.49	14.28	1.005	1.034

Test Sample File is from 29th author but not participated in building its author's automaton: 29_MaryErskine[JacobAbbott].pattern
Before removing common string, 29th author's automaton accepts 86.88 %
After removing common string, 29th author's automaton accepts 13.41 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	82.34	8.87	1.055	1.512
6	85.6	12.13	1.015	1.106
8	84.36	10.89	1.03	1.231
42	79.29	5.82	1.096	2.304
44	83.79	10.31	1.037	1.301
46	86.72	13.24	1.002	1.013

Test Sample File is from 31st author but not participated in building its author's automaton: 31_SenseandSensibility[JaneAusten].pattern
Before removing common string, 31st author's automaton accepts 87.07 %
After removing common string, 31st author's automaton accepts 16.72 %

Automaton	%_before	%_after	ratio_before	ratio_after
2	87.05	16.7	1	1.001
5	78.71	8.37	1.106	1.998
6	82.62	12.27	1.054	1.363
7	86.09	15.74	1.011	1.062
8	81.7	11.36	1.066	1.472
10	86.44	16.09	1.007	1.039
12	86.72	16.37	1.004	1.021
13	83.94	13.6	1.037	1.229
14	86.3	15.96	1.009	1.048
16	86.73	16.39	1.004	1.02
18	85.69	15.34	1.016	1.09
21	84.81	14.46	1.027	1.156
22	86.97	16.62	1.001	1.006
24	86.8	16.45	1.003	1.016
25	85.39	15.04	1.02	1.112
26	86.6	16.25	1.005	1.029
29	83.0	12.65	1.049	1.322
30	84.38	14.03	1.032	1.192
34	86.15	15.81	1.011	1.058
35	86.65	16.3	1.005	1.026
37	85.92	15.57	1.013	1.074
39	86.53	16.19	1.006	1.033
40	86.93	16.59	1.002	1.008
42	77.39	7.04	1.125	2.375
44	80.87	10.53	1.077	1.588
46	83.28	12.93	1.046	1.293
47	86.97	16.62	1.001	1.006
48	86.53	16.19	1.006	1.033

Test Sample File is from 32nd author but not participated in building its author's automaton: 32_Greenmantle[JohnBuchan].pattern
Before removing common string, 32nd author's automaton accepts 95.91 %
After removing common string, 32nd author's automaton accepts 16.88 %

Automaton	%_before	%_after	ratio_before	ratio_after

1	94.59	15.55	1.014	1.086
2	94.56	15.53	1.014	1.087
3	94.53	15.5	1.015	1.089
4	95.54	16.51	1.004	1.022
5	88.16	9.13	1.088	1.849
6	88.79	9.76	1.08	1.73
7	92.64	13.61	1.035	1.24
8	90.0	10.97	1.066	1.539
9	95.33	16.3	1.006	1.036
10	93.84	14.81	1.022	1.14
11	93.34	14.31	1.028	1.18
12	93.93	14.9	1.021	1.133
13	91.69	12.66	1.046	1.333
14	93.66	14.63	1.024	1.154
15	93.7	14.67	1.024	1.151
16	93.09	14.06	1.03	1.201
17	94.32	15.28	1.017	1.105
18	93.89	14.86	1.022	1.136
19	94.69	15.65	1.013	1.079
20	94.46	15.43	1.015	1.094
21	93.0	13.97	1.031	1.208
22	93.44	14.41	1.026	1.171
23	95.43	16.4	1.005	1.029
24	95.03	16.0	1.009	1.055
25	93.3	14.27	1.028	1.183
26	94.42	15.38	1.016	1.098
27	94.02	14.98	1.02	1.127
28	95.07	16.04	1.009	1.052
29	90.19	11.16	1.063	1.513
30	92.54	13.51	1.036	1.249
31	93.07	14.04	1.031	1.202
33	94.32	15.28	1.017	1.105
34	93.64	14.61	1.024	1.155
35	93.27	14.24	1.028	1.185
36	93.36	14.33	1.027	1.178
37	92.52	13.48	1.037	1.252
38	94.43	15.4	1.016	1.096
39	94.12	15.08	1.019	1.119
40	95.76	16.73	1.002	1.009
41	95.23	16.2	1.007	1.042
42	86.9	7.87	1.104	2.145
43	94.62	15.58	1.014	1.083
44	88.07	9.04	1.089	1.867
46	92.4	13.37	1.038	1.263
47	93.79	14.76	1.023	1.144
48	94.0	14.97	1.02	1.128
49	95.69	16.65	1.002	1.014

Test Sample File is from 33rd author but not participated in building its author's automaton: 33_TheBreathofLife[JohnBurroughs].pattern
Before removing common string, 33rd author's automaton accepts 88.0 %
After removing common string, 33rd author's automaton accepts 18.3 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	87.49	17.79	1.006	1.029
2	86.43	16.74	1.018	1.093
5	78.62	8.93	1.119	2.049
6	83.2	13.5	1.058	1.356
7	86.94	17.25	1.012	1.061
8	79.55	9.86	1.106	1.856
10	87.23	17.54	1.009	1.043
11	87.62	17.92	1.004	1.021
12	86.43	16.74	1.018	1.093
13	84.8	15.1	1.038	1.212
14	85.89	16.19	1.025	1.13

15	86.34	16.64	1.019	1.1
16	86.88	17.18	1.013	1.065
18	87.33	17.63	1.008	1.038
20	87.87	18.18	1.001	1.007
21	85.41	15.71	1.03	1.165
22	86.72	17.02	1.015	1.075
24	87.01	17.31	1.011	1.057
25	85.73	16.03	1.026	1.142
26	87.68	17.98	1.004	1.018
29	83.17	13.47	1.058	1.359
30	83.71	14.02	1.051	1.305
31	85.95	16.26	1.024	1.125
34	85.34	15.65	1.031	1.169
35	86.78	17.09	1.014	1.071
36	86.72	17.02	1.015	1.075
37	85.31	15.62	1.032	1.172
40	86.82	17.12	1.014	1.069
42	75.14	5.44	1.171	3.364
44	80.93	11.23	1.087	1.63
46	83.36	13.66	1.056	1.34
47	86.94	17.25	1.012	1.061
48	87.46	17.76	1.006	1.03

Test Sample File is from 34th author but not participated in building its author's automaton: 34_MrBonaparteofCorsica[JohnKendrickBangs].pattern
Before removing common string, 34th author's automaton accepts 90.2 %
After removing common string, 34th author's automaton accepts 15.14 %

Automaton	%_before	%_after	ratio_before	ratio_after
2	90.0	14.95	1.002	1.013
5	83.25	8.19	1.083	1.849
6	86.01	10.95	1.049	1.383
7	89.96	14.9	1.003	1.016
8	86.39	11.33	1.044	1.336
10	89.24	14.18	1.011	1.068
11	90.1	15.04	1.001	1.007
13	89.05	13.99	1.013	1.082
15	90.15	15.09	1.001	1.003
21	90.0	14.95	1.002	1.013
22	89.86	14.8	1.004	1.023
25	88.91	13.85	1.015	1.093
29	87.72	12.66	1.028	1.196
30	89.2	14.14	1.011	1.071
31	89.53	14.47	1.007	1.046
36	89.62	14.56	1.006	1.04
37	88.96	13.9	1.014	1.089
42	82.87	7.81	1.088	1.939
44	84.82	9.76	1.063	1.551
46	87.67	12.61	1.029	1.201

Test Sample File is from author but not participated in building its author's automaton: 35_TheManinLonelyLand[KateLangleyBosher].pattern
Before removing common string, author's automaton accepts 94.41 %
After removing common string, author's automaton accepts 13.38 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	94.13	13.1	1.003	1.021
5	87.99	6.96	1.073	1.922
6	88.78	7.75	1.063	1.726
7	92.52	11.49	1.02	1.164
8	90.46	9.43	1.044	1.419
9	94.31	13.28	1.001	1.008
10	93.52	12.49	1.01	1.071
11	93.83	12.8	1.006	1.045

12	93.48	12.45	1.01	1.075
13	91.8	10.77	1.028	1.242
14	93.21	12.18	1.013	1.099
15	93.93	12.9	1.005	1.037
16	93.52	12.49	1.01	1.071
18	93.38	12.35	1.011	1.083
20	93.93	12.9	1.005	1.037
21	93.17	12.14	1.013	1.102
22	93.34	12.32	1.011	1.086
24	93.31	12.28	1.012	1.09
25	93.31	12.28	1.012	1.09
26	93.52	12.49	1.01	1.071
29	91.22	10.19	1.035	1.313
30	92.42	11.39	1.022	1.175
31	93.1	12.08	1.014	1.108
32	94.17	13.14	1.003	1.018
33	93.96	12.93	1.005	1.035
34	93.24	12.21	1.013	1.096
36	93.65	12.62	1.008	1.06
37	92.21	11.18	1.024	1.197
38	93.79	12.76	1.007	1.049
39	93.45	12.42	1.01	1.077
40	93.65	12.62	1.008	1.06
42	87.44	6.42	1.08	2.084
43	93.86	12.83	1.006	1.043
44	88.64	7.62	1.065	1.756
46	92.14	11.11	1.025	1.204
47	93.62	12.59	1.008	1.063
48	93.79	12.76	1.007	1.049

Test Sample File is from 36th author but not participated in building its author's automaton: 36_LittleWomen[LouisaMayAlcott].pattern
Before removing common string, 36th author's automaton accepts 90.9 %
After removing common string, 36th author's automaton accepts 16.43 %

Automaton	%_before	%_after	ratio_before	ratio_after
2	90.52	16.05	1.004	1.024
5	83.49	9.02	1.089	1.822
6	86.18	11.71	1.055	1.403
7	89.6	15.13	1.015	1.086
8	85.44	10.97	1.064	1.498
10	90.26	15.79	1.007	1.041
12	90.4	15.93	1.006	1.031
13	87.62	13.16	1.037	1.248
14	90.24	15.77	1.007	1.042
15	90.74	16.27	1.002	1.01
16	89.85	15.38	1.012	1.068
18	90.38	15.91	1.006	1.033
21	89.62	15.15	1.014	1.084
22	90.3	15.84	1.007	1.037
24	90.75	16.29	1.002	1.009
25	88.95	14.49	1.022	1.134
26	90.5	16.03	1.004	1.025
29	86.83	12.36	1.047	1.329
30	88.65	14.18	1.025	1.159
31	89.93	15.46	1.011	1.063
33	90.67	16.2	1.003	1.014
34	89.88	15.41	1.011	1.066
35	90.68	16.21	1.002	1.014
37	89.68	15.21	1.014	1.08
39	90.75	16.28	1.002	1.009
42	81.75	7.29	1.112	2.254
44	84.74	10.27	1.073	1.6
46	88.17	13.7	1.031	1.199
47	90.7	16.23	1.002	1.012

48	90.57	16.1	1.004	1.02

Test Sample File is from 37th author but not participated in building its author's automaton: 37_PhantomFortuneaNovel[MEMaryElizabethBraddon].pattern
Before removing common string, 37th author's automaton accepts 90.42 %
After removing common string, 37th author's automaton accepts 13.76 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	84.78	8.12	1.067	1.695
6	86.89	10.23	1.041	1.345
7	89.88	13.22	1.006	1.041
8	86.44	9.78	1.046	1.407
13	89.4	12.74	1.011	1.08
21	90.31	13.65	1.001	1.008
29	87.56	10.9	1.033	1.262
31	90.4	13.73	1	1.002
42	83.42	6.76	1.084	2.036
44	86.18	9.52	1.049	1.445
46	88.74	12.08	1.019	1.139

Test Sample File is from 38th author but not participated in building its author's automaton: 38_TheWorksofMaxBeerbohm[MaxBeerbohm].pattern
Before removing common string, 38th author's automaton accepts 92.96 %
After removing common string, 38th author's automaton accepts 14.03 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	85.97	7.04	1.081	1.993
6	89.47	10.53	1.039	1.332
7	91.46	12.52	1.016	1.121
8	87.39	8.45	1.064	1.66
11	92.21	13.27	1.008	1.057
12	92.17	13.23	1.009	1.06
13	91.02	12.08	1.021	1.161
14	91.68	12.74	1.014	1.101
15	92.21	13.27	1.008	1.057
16	91.55	12.61	1.015	1.113
18	92.61	13.67	1.004	1.026
20	92.92	13.98	1	1.004
21	90.62	11.68	1.026	1.201
25	91.77	12.83	1.013	1.094
26	92.48	13.54	1.005	1.036
29	89.73	10.8	1.036	1.299
30	91.37	12.43	1.017	1.129
31	91.11	12.17	1.02	1.153
34	91.81	12.88	1.013	1.089
35	91.95	13.01	1.011	1.078
36	92.79	13.85	1.002	1.013
37	92.26	13.32	1.008	1.053
39	92.26	13.32	1.008	1.053
42	85.71	6.77	1.085	2.072
44	88.05	9.12	1.056	1.538
46	90.71	11.77	1.025	1.192
47	91.64	12.7	1.014	1.105
48	92.61	13.67	1.004	1.026

Test Sample File is from 39th author but not participated in building its author's automaton: 39_GunmansReckoning[MaxBrand].pattern
Before removing common string, 39th author's automaton accepts 95.2 %
After removing common string, 39th author's automaton accepts 12.19 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	89.75	6.74	1.061	1.809

6	90.5	7.5	1.052	1.625
7	93.84	10.83	1.014	1.126
8	92.25	9.25	1.032	1.318
10	94.84	11.84	1.004	1.03
12	94.87	11.87	1.003	1.027
13	93.27	10.27	1.021	1.187
14	94.4	11.39	1.008	1.07
15	94.59	11.59	1.006	1.052
16	94.34	11.34	1.009	1.075
18	95.1	12.09	1.001	1.008
21	94.47	11.47	1.008	1.063
22	94.01	11.01	1.013	1.107
25	94.52	11.51	1.007	1.059
26	94.81	11.81	1.004	1.032
29	91.36	8.36	1.042	1.458
30	94.28	11.28	1.01	1.081
31	94.19	11.19	1.011	1.089
34	94.61	11.6	1.006	1.051
35	94.56	11.56	1.007	1.054
36	94.5	11.5	1.007	1.06
37	94.21	11.2	1.011	1.088
42	89.41	6.4	1.065	1.905
44	89.91	6.9	1.059	1.767
46	94.37	11.36	1.009	1.073
48	95.17	12.16	1	1.002

Test Sample File is from 40th author but not participated in building its author's automaton: 40_TheAdventureClubAfloat[RalphHenryBarbour].pattern
Before removing common string, 40th author's automaton accepts 94.76 %
After removing common string, 40th author's automaton accepts 17.2 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	92.91	15.35	1.02	1.121
2	92.66	15.09	1.023	1.14
3	92.93	15.37	1.02	1.119
5	87.53	9.96	1.083	1.727
6	88.14	10.58	1.075	1.626
7	90.44	12.88	1.048	1.335
8	88.78	11.22	1.067	1.533
9	93.66	16.09	1.012	1.069
10	91.83	14.26	1.032	1.206
11	92.1	14.54	1.029	1.183
12	92.06	14.5	1.029	1.186
13	90.44	12.88	1.048	1.335
14	92.12	14.56	1.029	1.181
15	93.08	15.52	1.018	1.108
16	91.7	14.13	1.033	1.217
17	93.17	15.6	1.017	1.103
18	93.06	15.5	1.018	1.11
19	93.66	16.09	1.012	1.069
20	93.02	15.45	1.019	1.113
21	91.89	14.33	1.031	1.2
22	92.15	14.58	1.028	1.18
23	94.51	16.94	1.003	1.015
24	94.04	16.48	1.008	1.044
25	92.04	14.47	1.03	1.189
26	92.83	15.26	1.021	1.127
27	93.55	15.99	1.013	1.076
28	93.91	16.35	1.009	1.052
29	88.51	10.94	1.071	1.572
30	90.49	12.92	1.047	1.331
31	91.85	14.28	1.032	1.204
32	94.3	16.73	1.005	1.028
33	92.81	15.24	1.021	1.129
34	92.59	15.03	1.023	1.144

35	92.44	14.88	1.025	1.156
36	92.29	14.73	1.027	1.168
37	91.61	14.05	1.034	1.224
38	92.55	14.99	1.024	1.147
39	92.7	15.13	1.022	1.137
41	93.83	16.26	1.01	1.058
42	85.91	8.34	1.103	2.062
43	93.27	15.71	1.016	1.095
44	87.1	9.54	1.088	1.803
45	94.66	17.09	1.001	1.006
46	90.08	12.52	1.052	1.374
47	92.81	15.24	1.021	1.129
48	92.27	14.71	1.027	1.169
49	93.85	16.28	1.01	1.057

Test Sample File is from 41st author but not participated in building its author's automaton: 41_TheSwordMaker[RobertBarr].pattern
Before removing common string, 41st author's automaton accepts 91.78 %
After removing common string, 41st author's automaton accepts 19.17 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	90.24	17.62	1.017	1.088
2	90.38	17.76	1.015	1.079
3	90.5	17.89	1.014	1.072
5	82.64	10.03	1.111	1.911
6	84.91	12.29	1.081	1.56
7	87.97	15.36	1.043	1.248
8	84.22	11.61	1.09	1.651
9	91.45	18.84	1.004	1.018
10	89.17	16.56	1.029	1.158
11	89.97	17.36	1.02	1.104
12	89.74	17.12	1.023	1.12
13	87.66	15.04	1.047	1.275
14	89.7	17.09	1.023	1.122
15	89.52	16.9	1.025	1.134
16	88.83	16.22	1.033	1.182
17	90.72	18.11	1.012	1.059
18	89.89	17.28	1.021	1.109
19	91.44	18.83	1.004	1.018
20	90.33	17.72	1.016	1.082
21	87.74	15.12	1.046	1.268
22	89.81	17.2	1.022	1.115
24	91.14	18.53	1.007	1.035
25	88.38	15.76	1.038	1.216
26	90.13	17.51	1.018	1.095
28	91.02	18.4	1.008	1.042
29	85.49	12.87	1.074	1.49
30	88.2	15.59	1.041	1.23
31	89.13	16.51	1.03	1.161
32	91.47	18.86	1.003	1.016
33	90.14	17.53	1.018	1.094
34	89.27	16.65	1.028	1.151
35	89.3	16.68	1.028	1.149
36	89.55	16.93	1.025	1.132
37	89.03	16.42	1.031	1.167
38	90.27	17.65	1.017	1.086
39	90.02	17.4	1.02	1.102
40	91.5	18.89	1.003	1.015
42	81.05	8.44	1.132	2.271
43	91.06	18.45	1.008	1.039
44	83.42	10.81	1.1	1.773
46	86.74	14.12	1.058	1.358
47	89.58	16.97	1.025	1.13
48	89.0	16.39	1.031	1.17

Test Sample File is from 42nd author but not participated in building its author's automaton: 42_MenandWomen[RobertBrowning].pattern
Before removing common string, 42nd author's automaton accepts 81.59 %
After removing common string, 42nd author's automaton accepts 5.77 %

Note the sample file from 42nd author didn't get any positive result. The cause might be source of sample file is not reliable (dowonloaded from web) or inaccurate function words categorization gave wrong information about the author.

Test Sample File is from 43rd author but not participated in building its author's automaton: 43_Oddsfish[RobertHughBenson].pattern
Before removing common string, 43rd author's automaton accepts 89.1 %
After removing common string, 43rd author's automaton accepts 18.0 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	87.39	16.29	1.02	1.105
2	86.45	15.35	1.031	1.173
3	88.15	17.05	1.011	1.056
5	78.83	7.73	1.13	2.329
6	82.29	11.19	1.083	1.609
7	85.6	14.5	1.041	1.241
8	81.84	10.74	1.089	1.676
9	88.55	17.45	1.006	1.032
10	86.08	14.98	1.035	1.202
11	87.85	16.75	1.014	1.075
12	86.57	15.47	1.029	1.164
13	83.87	12.78	1.062	1.408
14	86.15	15.05	1.034	1.196
15	87.02	15.92	1.024	1.131
16	86.06	14.96	1.035	1.203
17	88.6	17.51	1.006	1.028
18	86.87	15.77	1.026	1.141
19	88.74	17.65	1.004	1.02
20	87.4	16.3	1.019	1.104
21	85.52	14.42	1.042	1.248
22	87.16	16.06	1.022	1.121
24	87.3	16.2	1.021	1.111
25	85.69	14.6	1.04	1.233
26	86.95	15.86	1.025	1.135
28	88.39	17.29	1.008	1.041
29	83.81	12.71	1.063	1.416
30	85.19	14.09	1.046	1.278
31	86.45	15.35	1.031	1.173
32	88.18	17.08	1.01	1.054
33	87.36	16.27	1.02	1.106
34	86.29	15.19	1.033	1.185
35	86.09	15.0	1.035	1.2
36	86.81	15.72	1.026	1.145
37	85.77	14.67	1.039	1.227
38	87.56	16.46	1.018	1.094
39	86.76	15.66	1.027	1.149
40	87.99	16.89	1.013	1.066
41	88.8	17.7	1.003	1.017
42	77.41	6.31	1.151	2.853
44	80.89	9.79	1.101	1.839
46	83.86	12.77	1.062	1.41
47	86.62	15.52	1.029	1.16
48	86.92	15.82	1.025	1.138
49	89.08	17.98	1	1.001

Test Sample File is from 44th author but not participated in building its author's automaton: 44_ErewhonRevisited[SamuelButler].pattern
Before removing common string, 44th author's automaton accepts 78.32 %

After removing common string, 44th author's automaton accepts 10.62 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	74.38	6.68	1.053	1.59
42	72.17	4.47	1.085	2.376

Test Sample File is from 45th author but not participated in building its author's automaton: 45_PonkapogPapers[ThomasBaileyAldrich].pattern
Before removing common string, 45th author's automaton accepts 92.2 %
After removing common string, 45th author's automaton accepts 20.46 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	90.5	18.76	1.019	1.091
2	90.44	18.7	1.019	1.094
3	91.23	19.49	1.011	1.05
5	83.86	12.12	1.099	1.688
6	85.99	14.25	1.072	1.436
7	88.06	16.32	1.047	1.254
8	84.29	12.55	1.094	1.63
9	90.62	18.88	1.017	1.084
10	90.26	18.51	1.021	1.105
11	90.26	18.51	1.021	1.105
12	89.46	17.72	1.031	1.155
13	87.52	15.77	1.053	1.297
14	89.52	17.78	1.03	1.151
15	90.8	19.06	1.015	1.073
16	89.34	17.6	1.032	1.162
17	90.13	18.39	1.023	1.113
18	88.98	17.24	1.036	1.187
19	91.35	19.61	1.009	1.043
20	90.74	19.0	1.016	1.077
21	89.34	17.6	1.032	1.162
22	89.95	18.21	1.025	1.124
24	90.74	19.0	1.016	1.077
25	88.43	16.69	1.043	1.226
26	90.8	19.06	1.015	1.073
27	91.84	20.1	1.004	1.018
28	91.78	20.04	1.005	1.021
29	85.38	13.64	1.08	1.5
30	86.6	14.86	1.065	1.377
31	89.04	17.3	1.035	1.183
33	89.52	17.78	1.03	1.151
34	88.49	16.75	1.042	1.221
35	89.22	17.48	1.033	1.17
36	89.65	17.9	1.028	1.143
37	89.71	17.97	1.028	1.139
38	90.26	18.51	1.021	1.105
39	89.77	18.03	1.027	1.135
40	90.38	18.64	1.02	1.098
41	91.66	19.91	1.006	1.028
42	81.24	9.5	1.135	2.154
43	90.93	19.18	1.014	1.067
44	84.53	12.79	1.091	1.6
46	86.97	15.23	1.06	1.343
47	88.37	16.63	1.043	1.23
48	90.07	18.33	1.024	1.116
49	91.17	19.43	1.011	1.053

Test Sample File is from 46th author but not participated in building its author's automaton: 46_BlackytheCrow[ThorntonWThorntonWaldoBurgess].pattern
Before removing common string, 46th author's automaton accepts 91.07 %
After removing common string, 46th author's automaton accepts 13.78 %

Automaton	%_before	%_after	ratio_before	ratio_after

5	85.41	8.11	1.066	1.699
6	86.98	9.69	1.047	1.422
8	87.74	10.45	1.038	1.319
13	89.78	12.49	1.014	1.103
25	90.83	13.54	1.003	1.018
29	88.79	11.5	1.026	1.198
42	84.94	7.65	1.072	1.801
44	86.4	9.11	1.054	1.513

Test Sample File is from 47th author but not participated in building its author's automaton: 47_CastAdrift[TimothyShayArthur].pattern
Before removing common string, 47th author's automaton accepts 93.82 %
After removing common string, 47th author's automaton accepts 13.63 %

Automaton	%_before	%_after	ratio_before	ratio_after
2	93.57	13.37	1.003	1.019
5	88.98	8.78	1.054	1.552
6	89.02	8.83	1.054	1.544
7	92.49	12.29	1.014	1.109
8	90.29	10.09	1.039	1.351
10	93.42	13.23	1.004	1.03
12	93.77	13.57	1.001	1.004
13	91.93	11.74	1.021	1.161
14	93.38	13.18	1.005	1.034
15	93.67	13.47	1.002	1.012
16	93.18	12.98	1.007	1.05
18	93.39	13.2	1.005	1.033
20	93.62	13.43	1.002	1.015
21	92.82	12.62	1.011	1.08
22	93.28	13.08	1.006	1.042
25	92.98	12.78	1.009	1.067
29	90.46	10.26	1.037	1.328
30	92.48	12.28	1.014	1.11
31	93.08	12.88	1.008	1.058
34	93.29	13.1	1.006	1.04
35	93.27	13.07	1.006	1.043
36	93.29	13.1	1.006	1.04
37	92.81	12.61	1.011	1.081
42	87.38	7.18	1.074	1.898
44	88.72	8.53	1.057	1.598
46	92.06	11.86	1.019	1.149
48	93.55	13.35	1.003	1.021

Test Sample File is from 48th author but not participated in building its author's automaton: 48_TomSwiftandHisSkyRacerortheQuickestFlightonRecord[VictorAppleton].pattern
Before removing common string, 48th author's automaton accepts 95.69 %
After removing common string, 48th author's automaton accepts 10.9 %

Automaton	%_before	%_after	ratio_before	ratio_after
5	91.53	6.73	1.045	1.62
6	91.65	6.85	1.044	1.591
7	94.5	9.71	1.013	1.123
8	93.63	8.84	1.022	1.233
11	95.28	10.48	1.004	1.04
12	95.42	10.63	1.003	1.025
13	94.33	9.54	1.014	1.143
14	95.54	10.75	1.002	1.014
15	95.52	10.73	1.002	1.016
16	95.42	10.63	1.003	1.025
21	94.99	10.19	1.007	1.07
22	94.77	9.98	1.01	1.092
25	94.99	10.19	1.007	1.07
29	92.95	8.16	1.029	1.336
30	95.01	10.22	1.007	1.067

31	94.77	9.98	1.01	1.092
34	95.21	10.41	1.005	1.047
35	95.16	10.36	1.006	1.052
36	95.59	10.8	1.001	1.009
37	94.7	9.9	1.01	1.101
42	91.43	6.63	1.047	1.644
44	91.79	7.0	1.042	1.557
46	94.99	10.19	1.007	1.07
47	95.62	10.82	1.001	1.007

Negative Results: when Automatons gave inaccurate or even incorrect results

The following ratios are unchanged or decrease after removing common string from sample files. This is caused by automaton's failure.

Test Sample File is from 1st author but not participated in building its author's automaton: 1_TheExtraDay[AlgernonBlackwood].pattern
Before removing common string, 1st author's automaton accepts 95.38 %
After removing common string, 1st author's automaton accepts 14.47 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	95.87	14.97	0.995	0.967
9	95.74	14.84	0.996	0.975
19	95.46	14.55	0.999	0.995
23	95.99	15.08	0.994	0.96
28	95.79	14.89	0.996	0.972
32	95.9	14.99	0.995	0.965
40	95.7	14.8	0.997	0.978
41	95.44	14.54	0.999	0.995
43	95.53	14.63	0.998	0.989
45	96.4	15.5	0.989	0.934
49	96.25	15.34	0.991	0.943

Test Sample File is from 2nd author but not participated in building its author's automaton: 2_ShapesofClay[AmbroseBierce].pattern
Before removing common string, 2nd author's automaton accepts 93.86 %
After removing common string, 2nd author's automaton accepts 12.93 %

Automaton	%_before	%_after	ratio_before	ratio_after
3	94.51	13.57	0.993	0.953
4	96.08	15.14	0.977	0.854
9	94.88	13.95	0.989	0.927
10	93.96	13.03	0.999	0.992
11	94.47	13.54	0.994	0.955
12	93.93	12.99	0.999	0.995
15	93.86	12.93	1	1
17	94.71	13.78	0.991	0.938
19	95.09	14.15	0.987	0.914
20	94.71	13.78	0.991	0.938
23	95.5	14.56	0.983	0.888
27	95.4	14.46	0.984	0.894
28	95.36	14.43	0.984	0.896
32	94.41	13.47	0.994	0.96
33	94.17	13.23	0.997	0.977
39	93.93	12.99	0.999	0.995
41	94.07	13.13	0.998	0.985
43	94.44	13.51	0.994	0.957
45	95.26	14.32	0.985	0.903
49	95.26	14.32	0.985	0.903

Test Sample File is from 3rd author but not participated in building its author's automaton: 3_RemembertheAlamo[AmeliaEdithHuddlestonBarr].pattern
Before removing common string, 3rd author's automaton accepts 94.83 %
After removing common string, 3rd author's automaton accepts 15.56 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	95.81	16.54	0.99	0.941
9	95.36	16.09	0.994	0.967
19	94.86	15.59	1	0.998
23	95.85	16.57	0.989	0.939
28	94.95	15.67	0.999	0.993
32	95.68	16.41	0.991	0.948
38	94.9	15.62	0.999	0.996
40	95.35	16.07	0.995	0.968
41	95.15	15.87	0.997	0.98
45	95.38	16.11	0.994	0.966
49	95.48	16.21	0.993	0.96

Test Sample File is from 4th author but not participated in building its author's automaton: 4_TheLogofaCowboy[AndyAdams].pattern
Before removing common string, 4th author's automaton accepts 87.88 %
After removing common string, 4th author's automaton accepts 34.13 %

Note the sample file from 4th author didn't get any negative results.

Test Sample File is from 5th author but not participated in building its author's automaton: 5_TheCaseforIndia[AnnieWoodBesant].pattern
Before removing common string, 5th author's automaton accepts 83.07 %
After removing common string, 5th author's automaton accepts 7.0 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	90.39	14.32	0.919	0.489
2	88.71	12.64	0.936	0.554
3	89.97	13.9	0.923	0.504
4	92.06	15.99	0.902	0.438
6	86.62	10.55	0.959	0.664
7	89.55	13.48	0.928	0.519
9	90.91	14.84	0.914	0.472
10	90.49	14.42	0.918	0.485
11	89.86	13.79	0.924	0.508
12	90.39	14.32	0.919	0.489
13	89.24	13.17	0.931	0.532
14	87.67	11.6	0.948	0.603
15	89.34	13.27	0.93	0.528
16	89.86	13.79	0.924	0.508
17	89.86	13.79	0.924	0.508
18	90.7	14.63	0.916	0.478
19	91.95	15.88	0.903	0.441
20	90.18	14.11	0.921	0.496
21	89.13	13.06	0.932	0.536
22	88.3	12.23	0.941	0.572
23	91.12	15.05	0.912	0.465
24	89.03	12.96	0.933	0.54
25	89.55	13.48	0.928	0.519
26	89.45	13.38	0.929	0.523
27	92.16	16.09	0.901	0.435
28	92.37	16.3	0.899	0.429
29	87.25	11.18	0.952	0.626
30	86.83	10.76	0.957	0.651
31	88.4	12.33	0.94	0.568
32	90.49	14.42	0.918	0.485
33	90.8	14.73	0.915	0.475
34	88.82	12.75	0.935	0.549
35	89.03	12.96	0.933	0.54
36	89.55	13.48	0.928	0.519

37	88.4	12.33	0.94	0.568
38	90.49	14.42	0.918	0.485
39	89.03	12.96	0.933	0.54
40	87.36	11.29	0.951	0.62
41	90.07	14.0	0.922	0.5
43	90.8	14.73	0.915	0.475
45	91.64	15.57	0.906	0.45
46	86.42	10.34	0.961	0.677
47	88.61	12.54	0.937	0.558
48	89.45	13.38	0.929	0.523
49	91.01	14.94	0.913	0.469

Test Sample File is from 6th author but not participated in building its author's automaton: 6_PoliticsATreatiseonGovernment[Aristotle].pattern
Before removing common string, 6th author's automaton accepts 59.87 %
After removing common string, 6th author's automaton accepts 13.69 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	60.42	14.24	0.991	0.961
3	61.22	15.04	0.978	0.91
4	63.17	16.99	0.948	0.806
7	60.57	14.39	0.988	0.951
9	60.17	13.99	0.995	0.979
10	62.02	15.84	0.965	0.864
11	62.02	15.84	0.965	0.864
12	61.02	14.84	0.981	0.923
14	61.42	15.24	0.975	0.898
15	62.57	16.39	0.957	0.835
16	61.42	15.24	0.975	0.898
17	65.52	19.34	0.914	0.708
19	63.77	17.59	0.939	0.778
20	61.02	14.84	0.981	0.923
22	65.17	18.99	0.919	0.721
23	64.27	18.09	0.932	0.757
24	60.87	14.69	0.984	0.932
26	61.12	14.94	0.98	0.916
27	68.07	21.89	0.88	0.625
28	62.42	16.24	0.959	0.843
31	62.57	16.39	0.957	0.835
32	60.27	14.09	0.993	0.972
33	61.47	15.29	0.974	0.895
34	61.87	15.69	0.968	0.873
35	60.37	14.19	0.992	0.965
36	60.82	14.64	0.984	0.935
38	62.17	15.99	0.963	0.856
39	60.42	14.24	0.991	0.961
41	65.37	19.19	0.916	0.713
43	63.37	17.19	0.945	0.796
45	63.92	17.74	0.937	0.772
47	60.87	14.69	0.984	0.932
48	62.02	15.84	0.965	0.864
49	67.07	20.89	0.893	0.655

Test Sample File is from 7th author but not participated in building its author's automaton: 7_FatherPayne[ArthurChristopherBenson].pattern
Before removing common string, 7th author's automaton accepts 90.68 %
After removing common string, 7th author's automaton accepts 16.81 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	90.98	17.1	0.997	0.983
2	90.75	16.87	0.999	0.996
3	91.65	17.77	0.989	0.946
4	92.05	18.17	0.985	0.925
9	92.28	18.4	0.983	0.914

10	90.98	17.1	0.997	0.983
11	91.27	17.39	0.994	0.967
15	90.81	16.93	0.999	0.993
17	91.68	17.8	0.989	0.944
19	90.88	17.01	0.998	0.988
20	91.16	17.28	0.995	0.973
23	92.42	18.54	0.981	0.907
24	90.75	16.87	0.999	0.996
26	90.68	16.81	1	1
27	91.53	17.65	0.991	0.952
28	91.93	18.05	0.986	0.931
32	92.39	18.51	0.981	0.908
36	90.72	16.84	1	0.998
38	91.18	17.3	0.995	0.972
39	91.18	17.3	0.995	0.972
40	90.79	16.91	0.999	0.994
41	91.85	17.97	0.987	0.935
43	91.54	17.67	0.991	0.951
45	92.48	18.6	0.981	0.904
49	92.66	18.78	0.979	0.895

Test Sample File is from 8th author but not participated in building its author's automaton: 8_TheTaleofCuffyBear[ArthurScottBailey].pattern
Before removing common string, 8th author's automaton accepts 92.01 %
After removing common string, 8th author's automaton accepts 8.79 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	95.57	12.35	0.963	0.712
2	94.77	11.54	0.971	0.762
3	95.1	11.88	0.968	0.74
4	96.11	12.89	0.957	0.682
7	93.15	9.93	0.988	0.885
9	95.5	12.28	0.963	0.716
10	94.7	11.48	0.972	0.766
11	95.3	12.08	0.965	0.728
12	95.17	11.95	0.967	0.736
13	93.62	10.4	0.983	0.845
14	94.36	11.14	0.975	0.789
15	94.43	11.21	0.974	0.784
16	94.03	10.81	0.979	0.813
17	94.97	11.74	0.969	0.749
18	94.56	11.34	0.973	0.775
19	95.64	12.42	0.962	0.708
20	96.11	12.89	0.957	0.682
21	95.1	11.88	0.968	0.74
22	94.03	10.81	0.979	0.813
23	96.64	13.42	0.952	0.655
24	95.17	11.95	0.967	0.736
25	94.43	11.21	0.974	0.784
26	95.1	11.88	0.968	0.74
27	94.83	11.61	0.97	0.757
28	95.17	11.95	0.967	0.736
29	93.29	10.07	0.986	0.873
30	93.62	10.4	0.983	0.845
31	94.5	11.28	0.974	0.779
32	95.97	12.75	0.959	0.689
33	95.37	12.15	0.965	0.723
34	93.83	10.6	0.981	0.829
35	94.43	11.21	0.974	0.784
36	95.1	11.88	0.968	0.74
37	93.36	10.13	0.986	0.868
38	95.64	12.42	0.962	0.708
39	94.63	11.41	0.972	0.77
40	94.9	11.68	0.97	0.753
41	95.84	12.62	0.96	0.697

43	95.17	11.95	0.967	0.736
45	96.31	13.09	0.955	0.672
46	93.42	10.2	0.985	0.862
47	95.3	12.08	0.965	0.728
48	95.03	11.81	0.968	0.744
49	95.37	12.15	0.965	0.723

Test Sample File is from 9th author but not participated in building its author's automaton: 9_HerPrairieKnight[BMBower].pattern
Before removing common string, 9th author's automaton accepts 97.36 %
After removing common string, 9th author's automaton accepts 11.63 %

Automaton	%_before	%_after	ratio_before	ratio_after
20	97.72	11.99	0.996	0.97
45	97.54	11.81	0.998	0.985

Test Sample File is from 10th author but not participated in building its author's automaton: 10_WielandortheTransformationanAmericanTale[CharlesBrockdenBrown].pattern
Before removing common string, 10th author's automaton accepts 95.66 %
After removing common string, 10th author's automaton accepts 15.06 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	95.77	15.17	0.999	0.993
9	96.16	15.55	0.995	0.968
23	96.35	15.74	0.993	0.957
28	95.94	15.33	0.997	0.982
32	96.24	15.63	0.994	0.964
38	95.89	15.28	0.998	0.986
41	95.92	15.31	0.997	0.984
45	96.59	15.98	0.99	0.942
49	96.0	15.39	0.996	0.979

Test Sample File is from 11th author but not participated in building its author's automaton: 11_JaneEyre[CharlotteBronte].pattern
Before removing common string, 11th author's automaton accepts 90.72 %
After removing common string, 11th author's automaton accepts 14.84 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	90.77	14.9	0.999	0.996
3	91.0	15.12	0.997	0.981
4	92.78	16.9	0.978	0.878
9	91.18	15.3	0.995	0.97
17	91.23	15.36	0.994	0.966
19	91.57	15.69	0.991	0.946
20	91.15	15.28	0.995	0.971
23	91.91	16.04	0.987	0.925
27	92.13	16.25	0.985	0.913
28	91.41	15.54	0.992	0.955
32	91.59	15.72	0.991	0.944
38	91.07	15.2	0.996	0.976
40	91.12	15.24	0.996	0.974
41	91.55	15.67	0.991	0.947
43	91.7	15.83	0.989	0.937
45	92.38	16.51	0.982	0.899
49	91.62	15.75	0.99	0.942

Test Sample File is from 12th author but not participated in building its author's automaton: 12_Coralie[CharlotteMCharlotteMonicaBrame].pattern
Before removing common string, 12th author's automaton accepts 93.58 %
After removing common string, 12th author's automaton accepts 13.01 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	94.47	13.9	0.991	0.936
2	93.98	13.41	0.996	0.97
3	95.1	14.53	0.984	0.895
4	96.52	15.95	0.97	0.816
9	96.39	15.82	0.971	0.822
10	94.16	13.59	0.994	0.957
11	94.03	13.46	0.995	0.967
15	93.85	13.28	0.997	0.98
17	94.07	13.5	0.995	0.964
18	94.39	13.81	0.991	0.942
19	95.23	14.66	0.983	0.887
20	94.16	13.59	0.994	0.957
23	95.54	14.97	0.979	0.869
24	94.96	14.39	0.985	0.904
25	94.39	13.81	0.991	0.942
26	95.19	14.62	0.983	0.89
27	94.61	14.04	0.989	0.927
28	94.96	14.39	0.985	0.904
32	95.54	14.97	0.979	0.869
33	94.3	13.73	0.992	0.948
34	93.72	13.15	0.999	0.989
35	94.56	13.99	0.99	0.93
36	93.81	13.24	0.998	0.983
38	94.39	13.81	0.991	0.942
39	94.65	14.08	0.989	0.924
40	95.41	14.84	0.981	0.877
41	95.77	15.2	0.977	0.856
43	94.56	13.99	0.99	0.93
45	95.77	15.2	0.977	0.856
47	93.85	13.28	0.997	0.98
48	94.3	13.73	0.992	0.948
49	96.08	15.51	0.974	0.839

Test Sample File is from 14th author but not participated in building its author's automaton: 14_TheMucker[EdgarRiceBurroughs].pattern
Before removing common string, 14th author's automaton accepts 91.19 %
After removing common string, 14th author's automaton accepts 13.78 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	91.67	14.26	0.995	0.966
2	91.82	14.41	0.993	0.956
3	92.35	14.94	0.987	0.922
4	94.31	16.9	0.967	0.815
9	92.84	15.43	0.982	0.893
10	91.43	14.03	0.997	0.982
11	91.73	14.32	0.994	0.962
15	91.72	14.31	0.994	0.963
17	92.1	14.7	0.99	0.937
19	92.57	15.16	0.985	0.909
20	92.35	14.94	0.987	0.922
22	91.41	14.0	0.998	0.984
23	93.29	15.88	0.977	0.868
24	91.9	14.49	0.992	0.951
26	91.56	14.16	0.996	0.973
27	93.44	16.04	0.976	0.859
28	92.84	15.43	0.982	0.893
32	93.07	15.66	0.98	0.88
33	91.51	14.11	0.997	0.977
35	91.38	13.98	0.998	0.986
38	91.96	14.56	0.992	0.946
39	91.94	14.53	0.992	0.948
40	92.54	15.14	0.985	0.91
41	93.19	15.78	0.979	0.873
43	92.81	15.41	0.983	0.894

45	93.48	16.08	0.976	0.857
47	91.6	14.2	0.996	0.97
48	91.73	14.32	0.994	0.962
49	92.81	15.41	0.983	0.894

Test Sample File is from 16th author but not participated in building its author's automaton: 16_TTembarom[FrancesHodgsonBurnett].pattern
Before removing common string, 16th author's automaton accepts 92.95 %
After removing common string, 16th author's automaton accepts 12.59 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	93.9	13.54	0.99	0.93
2	93.54	13.19	0.994	0.955
3	94.44	14.08	0.984	0.894
4	95.44	15.08	0.974	0.835
9	94.86	14.5	0.98	0.868
10	93.43	13.08	0.995	0.963
11	93.78	13.43	0.991	0.937
12	93.51	13.15	0.994	0.957
14	93.13	12.77	0.998	0.986
15	93.5	13.14	0.994	0.958
17	94.17	13.81	0.987	0.912
18	93.56	13.2	0.993	0.954
19	94.23	13.87	0.986	0.908
20	93.91	13.55	0.99	0.929
23	95.28	14.93	0.976	0.843
24	94.55	14.2	0.983	0.887
25	93.15	12.8	0.998	0.984
26	93.57	13.21	0.993	0.953
27	94.31	13.96	0.986	0.902
28	94.64	14.28	0.982	0.882
32	95.21	14.86	0.976	0.847
33	93.84	13.49	0.991	0.933
34	93.04	12.69	0.999	0.992
35	93.42	13.06	0.995	0.964
36	93.28	12.93	0.996	0.974
38	93.92	13.57	0.99	0.928
39	93.81	13.46	0.991	0.935
40	94.71	14.36	0.981	0.877
41	94.49	14.14	0.984	0.89
43	94.65	14.29	0.982	0.881
45	95.38	15.02	0.975	0.838
47	93.41	13.05	0.995	0.965
48	93.41	13.05	0.995	0.965
49	95.07	14.72	0.978	0.855

Test Sample File is from 17th author but not participated in building its author's automaton: 17_TheRomanyRye[GeorgeHenryBorrow].pattern
Before removing common string, 17th author's automaton accepts 84.24 %
After removing common string, 17th author's automaton accepts 16.67 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	85.58	18.02	0.984	0.925
9	84.3	16.74	0.999	0.996
19	84.4	16.83	0.998	0.99
23	85.68	18.11	0.983	0.92
27	86.37	18.81	0.975	0.886
28	84.7	17.13	0.995	0.973
41	84.67	17.1	0.995	0.975
45	85.73	18.16	0.983	0.918
49	84.92	17.35	0.992	0.961

Test Sample File is from 18th author but not participated in building its author's automaton: 18_TheSisters-In-Law[GertrudeFranklinHornAtherton].pattern
Before removing common string, 18th author's automaton accepts 92.66 %
After removing common string, 18th author's automaton accepts 12.12 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	93.37	12.82	0.992	0.945
2	93.36	12.81	0.993	0.946
3	94.1	13.55	0.985	0.894
4	94.88	14.33	0.977	0.846
9	94.08	13.53	0.985	0.896
11	93.08	12.53	0.995	0.967
12	92.66	12.12	1	1
14	92.85	12.31	0.998	0.985
15	93.05	12.5	0.996	0.97
17	93.87	13.33	0.987	0.909
19	93.74	13.2	0.988	0.918
20	93.6	13.06	0.99	0.928
23	94.47	13.93	0.981	0.87
24	93.89	13.35	0.987	0.908
26	93.24	12.69	0.994	0.955
27	93.77	13.22	0.988	0.917
28	93.93	13.38	0.986	0.906
31	92.66	12.12	1	1
32	94.28	13.73	0.983	0.883
33	93.33	12.78	0.993	0.948
35	92.95	12.4	0.997	0.977
38	93.53	12.98	0.991	0.934
39	92.87	12.32	0.998	0.984
40	94.11	13.56	0.985	0.894
41	93.66	13.11	0.989	0.924
43	93.92	13.37	0.987	0.907
45	94.59	14.04	0.98	0.863
47	92.91	12.36	0.997	0.981
48	92.75	12.2	0.999	0.993
49	94.43	13.88	0.981	0.873

Test Sample File is from 19th author but not participated in building its author's automaton: 19_HildaWadeaWomanwithTenacityofPurpose[GrantAllen].pattern
Before removing common string, 19th author's automaton accepts 95.64 %
After removing common string, 19th author's automaton accepts 13.95 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	96.17	14.48	0.994	0.963
23	95.88	14.19	0.997	0.983
45	96.33	14.64	0.993	0.953
49	95.81	14.12	0.998	0.988

Test Sample File is from 20th author but not participated in building its author's automaton: 20_PicturesofSweden[HansChristianAndersen].pattern
Before removing common string, 20th author's automaton accepts 90.23 %
After removing common string, 20th author's automaton accepts 19.49 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	91.91	21.16	0.982	0.921
19	90.37	19.63	0.998	0.993
23	90.93	20.19	0.992	0.965
27	91.3	20.56	0.988	0.948
28	90.42	19.67	0.998	0.991
45	91.77	21.02	0.983	0.927

Test Sample File is from 21st author but not participated in building its author's automaton: 21_RanchingforSylvia[HaroldBindloss].pattern
Before removing common string, 21st author's automaton accepts 93.41 %
After removing common string, 21st author's automaton accepts 13.23 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	94.03	13.86	0.993	0.955
2	93.75	13.57	0.996	0.975
3	94.61	14.44	0.987	0.916
4	95.78	15.61	0.975	0.848
9	95.17	14.99	0.982	0.883
10	93.48	13.3	0.999	0.995
11	94.2	14.03	0.992	0.943
12	93.56	13.39	0.998	0.988
14	93.41	13.23	1	1
15	93.92	13.74	0.995	0.963
16	93.55	13.37	0.999	0.99
17	94.37	14.2	0.99	0.932
18	93.97	13.8	0.994	0.959
19	94.44	14.27	0.989	0.927
20	94.19	14.01	0.992	0.944
22	93.5	13.33	0.999	0.992
23	95.49	15.32	0.978	0.864
24	95.25	15.08	0.981	0.877
25	93.62	13.45	0.998	0.984
26	94.33	14.16	0.99	0.934
27	94.33	14.16	0.99	0.934
28	94.95	14.78	0.984	0.895
32	95.74	15.56	0.976	0.85
33	94.06	13.89	0.993	0.952
36	93.48	13.3	0.999	0.995
38	94.26	14.08	0.991	0.94
39	94.14	13.97	0.992	0.947
40	95.04	14.87	0.983	0.89
41	95.0	14.82	0.983	0.893
43	95.17	14.99	0.982	0.883
45	96.03	15.86	0.973	0.834
47	94.16	13.99	0.992	0.946
48	94.24	14.07	0.991	0.94
49	95.45	15.28	0.979	0.866

Test Sample File is from 22nd author but not participated in building its author's automaton: 22_ThePathtoRome[HilaireBelloc].pattern
Before removing common string, 22nd author's automaton accepts 84.54 %
After removing common string, 22nd author's automaton accepts 19.27 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	88.16	22.89	0.959	0.842
9	85.63	20.36	0.987	0.946
17	85.58	20.31	0.988	0.949
19	85.77	20.5	0.986	0.94
23	86.7	21.43	0.975	0.899
27	86.77	21.49	0.974	0.897
28	85.45	20.17	0.989	0.955
32	85.33	20.06	0.991	0.961
41	85.65	20.38	0.987	0.946
43	85.45	20.17	0.989	0.955
45	86.33	21.05	0.979	0.915
49	85.84	20.57	0.985	0.937

Test Sample File is from 23rd author but not participated in building its author's automaton: 23_LostIllusions[HonordeBalzac].pattern
Before removing common string, 23rd author's automaton accepts 96.61 %
After removing common string, 23rd author's automaton accepts 19.26 %

Note the sample file from 23rd author didn't get any negative results.

Test Sample File is from 24th author but not participated in building its author's automaton: 24_PaulPrescottsCharge[HoratioAlger].pattern
Before removing common string, 24th author's automaton accepts 96.16 %
After removing common string, 24th author's automaton accepts 12.39 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	96.88	13.11	0.993	0.945
23	96.61	12.83	0.995	0.966
26	96.2	12.43	1	0.997
32	96.94	13.17	0.992	0.941
40	96.39	12.61	0.998	0.983
41	96.18	12.41	1	0.998
43	96.18	12.41	1	0.998
45	96.57	12.8	0.996	0.968
49	96.57	12.8	0.996	0.968

Test Sample File is from 25th author but not participated in building its author's automaton: 25_TheGayCockade[IreneTempleBailey].pattern
Before removing common string, 25th author's automaton accepts 95.47 %
After removing common string, 25th author's automaton accepts 11.72 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	95.84	12.09	0.996	0.969
2	96.02	12.27	0.994	0.955
3	96.23	12.48	0.992	0.939
4	96.35	12.6	0.991	0.93
9	96.56	12.81	0.989	0.915
10	95.51	11.76	1	0.997
12	95.58	11.83	0.999	0.991
15	95.56	11.82	0.999	0.992
17	95.56	11.82	0.999	0.992
19	96.31	12.56	0.991	0.933
20	95.69	11.94	0.998	0.982
23	96.66	12.91	0.988	0.908
24	95.98	12.23	0.995	0.958
26	95.88	12.13	0.996	0.966
28	96.02	12.27	0.994	0.955
32	96.67	12.92	0.988	0.907
33	95.73	11.98	0.997	0.978
35	95.59	11.84	0.999	0.99
38	96.53	12.78	0.989	0.917
39	95.94	12.19	0.995	0.961
40	96.53	12.78	0.989	0.917
41	96.28	12.53	0.992	0.935
43	95.9	12.15	0.996	0.965
45	97.36	13.61	0.981	0.861
47	95.47	11.72	1	1
48	95.61	11.86	0.999	0.988
49	96.81	13.06	0.986	0.897

Test Sample File is from 26th author but not participated in building its author's automaton: 26_EbenHoldenataleofthenorthcountry[IrvingBacheller].pattern
Before removing common string, 26th author's automaton accepts 95.14 %
After removing common string, 26th author's automaton accepts 13.34 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	95.49	13.68	0.996	0.975
3	95.68	13.87	0.994	0.962
4	95.88	14.08	0.992	0.947
9	96.4	14.6	0.987	0.914

11	95.2	13.4	0.999	0.996
12	95.14	13.34	1	1
17	95.28	13.48	0.999	0.99
18	95.25	13.45	0.999	0.992
19	95.83	14.03	0.993	0.951
20	95.3	13.49	0.998	0.989
23	96.07	14.27	0.99	0.935
24	95.5	13.7	0.996	0.974
27	95.31	13.51	0.998	0.987
28	96.02	14.22	0.991	0.938
32	96.37	14.57	0.987	0.916
33	95.39	13.59	0.997	0.982
38	95.57	13.76	0.996	0.969
40	96.17	14.36	0.989	0.929
41	95.72	13.92	0.994	0.958
43	95.68	13.87	0.994	0.962
45	96.31	14.51	0.988	0.919
48	95.14	13.34	1	1
49	96.39	14.59	0.987	0.914

Test Sample File is from 27th author but not participated in building its author's automaton: 27_TheGoldenChersoneseandtheWayThither[IsabellaLIsabellaLucyBird].pattern
Before removing common string, 27th author's automaton accepts 85.56 %
After removing common string, 27th author's automaton accepts 24.51 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	87.22	26.17	0.981	0.937

Test Sample File is from 28th author but not participated in building its author's automaton: 28_TommyandGrizel[JamesMatthewBarrie].pattern
Before removing common string, 28th author's automaton accepts 92.98 %
After removing common string, 28th author's automaton accepts 14.77 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	93.25	15.04	0.997	0.982
9	93.52	15.31	0.994	0.965
19	93.4	15.19	0.996	0.972
23	93.42	15.22	0.995	0.97
32	93.06	14.85	0.999	0.995
43	93.03	14.82	0.999	0.997
45	94.17	15.96	0.987	0.925
49	93.84	15.63	0.991	0.945

Test Sample File is from 29th author but not participated in building its author's automaton: 29_MaryErskine[JacobAbbott].pattern
Before removing common string, 29th author's automaton accepts 86.88 %
After removing common string, 29th author's automaton accepts 13.41 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	89.85	16.38	0.967	0.819
2	89.19	15.72	0.974	0.853
3	91.01	17.53	0.955	0.765
4	92.29	18.81	0.941	0.713
7	88.37	14.89	0.983	0.901
9	91.17	17.7	0.953	0.758
10	88.74	15.26	0.979	0.879
11	89.69	16.21	0.969	0.827
12	89.6	16.13	0.97	0.831
13	87.46	13.99	0.993	0.959
14	88.61	15.14	0.98	0.886
15	90.72	17.24	0.958	0.778
16	89.03	15.55	0.976	0.862

17	90.72	17.24	0.958	0.778
18	89.48	16.01	0.971	0.838
19	92.16	18.69	0.943	0.717
20	90.18	16.71	0.963	0.803
21	87.75	14.27	0.99	0.94
22	89.85	16.38	0.967	0.819
23	91.58	18.11	0.949	0.74
24	90.22	16.75	0.963	0.801
25	88.86	15.39	0.978	0.871
26	89.85	16.38	0.967	0.819
27	92.41	18.94	0.94	0.708
28	90.92	17.45	0.956	0.768
30	87.58	14.11	0.992	0.95
31	88.24	14.77	0.985	0.908
32	91.25	17.78	0.952	0.754
33	89.93	16.46	0.966	0.815
34	89.69	16.21	0.969	0.827
35	90.18	16.71	0.963	0.803
36	90.06	16.58	0.965	0.809
37	88.16	14.69	0.985	0.913
38	90.97	17.49	0.955	0.767
39	89.56	16.09	0.97	0.833
40	89.52	16.05	0.971	0.836
41	90.59	17.12	0.959	0.783
43	90.06	16.58	0.965	0.809
45	92.62	19.14	0.938	0.701
47	89.27	15.8	0.973	0.849
48	89.27	15.8	0.973	0.849
49	91.83	18.36	0.946	0.73

Test Sample File is from 31st author but not participated in building its author's automaton: 31_SenseandSensibility[JaneAusten].pattern
Before removing common string, 31st author's automaton accepts 87.07 %
After removing common string, 31st author's automaton accepts 16.72 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	87.43	17.08	0.996	0.979
3	87.5	17.15	0.995	0.975
4	88.86	18.51	0.98	0.903
9	88.31	17.96	0.986	0.931
11	87.32	16.97	0.997	0.985
15	87.25	16.9	0.998	0.989
17	88.51	18.16	0.984	0.921
19	88.15	17.8	0.988	0.939
20	87.83	17.48	0.991	0.957
23	88.81	18.46	0.98	0.906
27	89.34	18.99	0.975	0.88
28	89.11	18.76	0.977	0.891
32	87.91	17.57	0.99	0.952
33	87.51	17.17	0.995	0.974
36	87.35	17.0	0.997	0.984
38	87.63	17.28	0.994	0.968
41	88.59	18.25	0.983	0.916
43	88.59	18.25	0.983	0.916
45	89.13	18.78	0.977	0.89
49	89.24	18.89	0.976	0.885

Test Sample File is from 32nd author but not participated in building its author's automaton: 32_Greenmantle[JohnBuchan].pattern
Before removing common string, 32nd author's automaton accepts 95.91 %
After removing common string, 32nd author's automaton accepts 16.88 %

Automaton	%_before	%_after	ratio_before	ratio_after
45	96.1	17.07	0.998	0.989

Test Sample File is from 33rd author but not participated in building its author's automaton: 33_TheBreathofLife[JohnBurroughs].pattern
Before removing common string, 33rd author's automaton accepts 88.0 %
After removing common string, 33rd author's automaton accepts 18.3 %

Automaton	%_before	%_after	ratio_before	ratio_after
3	88.13	18.43	0.999	0.993
4	90.21	20.51	0.976	0.892
9	88.7	19.01	0.992	0.963
17	88.45	18.75	0.995	0.976
19	89.22	19.52	0.986	0.938
23	90.05	20.35	0.977	0.899
27	90.5	20.8	0.972	0.88
28	90.02	20.32	0.978	0.901
32	88.45	18.75	0.995	0.976
38	88.58	18.88	0.993	0.969
39	88.06	18.37	0.999	0.996
41	89.86	20.16	0.979	0.908
43	89.22	19.52	0.986	0.938
45	89.63	19.94	0.982	0.918
49	89.38	19.68	0.985	0.93

Test Sample File is from 34th author but not participated in building its author's automaton: 34_MrBonaparteofCorsica[JohnKendrickBangs].pattern
Before removing common string, 34th author's automaton accepts 90.2 %
After removing common string, 34th author's automaton accepts 15.14 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	91.24	16.18	0.989	0.936
3	91.34	16.28	0.988	0.93
4	93.43	18.37	0.965	0.824
9	91.72	16.66	0.983	0.909
12	90.58	15.52	0.996	0.976
14	90.91	15.85	0.992	0.955
16	90.53	15.47	0.996	0.979
17	91.91	16.85	0.981	0.899
18	90.48	15.42	0.997	0.982
19	91.39	16.33	0.987	0.927
20	91.29	16.23	0.988	0.933
23	93.48	18.42	0.965	0.822
24	91.72	16.66	0.983	0.909
26	90.58	15.52	0.996	0.976
27	92.0	16.94	0.98	0.894
28	91.19	16.14	0.989	0.938
32	93.24	18.18	0.967	0.833
33	90.67	15.61	0.995	0.97
35	90.72	15.66	0.994	0.967
38	91.67	16.61	0.984	0.911
39	90.53	15.47	0.996	0.979
40	91.62	16.56	0.985	0.914
41	92.38	17.33	0.976	0.874
43	91.39	16.33	0.987	0.927
45	92.96	17.9	0.97	0.846
47	91.15	16.09	0.99	0.941
48	90.67	15.61	0.995	0.97
49	92.1	17.04	0.979	0.888

Test Sample File is from 35th author but not participated in building its author's automaton: 35_TheManinLonelyLand[KateLangleyBosher].pattern
Before removing common string, 35th author's automaton accepts 94.41 %
After removing common string, 35th author's automaton accepts 13.38 %

Automaton	%_before	%_after	ratio_before	ratio_after
2	94.44	13.41	1	0.998
3	94.58	13.55	0.998	0.987
4	95.4	14.37	0.99	0.931
17	94.72	13.69	0.997	0.977
19	94.89	13.86	0.995	0.965
23	95.47	14.44	0.989	0.927
27	95.16	14.13	0.992	0.947
28	94.82	13.79	0.996	0.97
41	94.41	13.38	1	1
45	94.99	13.96	0.994	0.958
49	94.58	13.55	0.998	0.987

Test Sample File is from 36th author but not participated in building its author's automaton: 36_LittleWomen[LouisaMayAlcott].pattern
Before removing common string, 36th author's automaton accepts 90.9 %
After removing common string, 36th author's automaton accepts 16.43 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	91.29	16.82	0.996	0.977
3	91.75	17.28	0.991	0.951
4	92.7	18.23	0.981	0.901
9	92.67	18.2	0.981	0.903
11	91.17	16.7	0.997	0.984
17	91.67	17.21	0.992	0.955
19	91.67	17.21	0.992	0.955
20	91.52	17.06	0.993	0.963
23	92.93	18.46	0.978	0.89
27	92.62	18.15	0.981	0.905
28	92.11	17.64	0.987	0.931
32	92.72	18.26	0.98	0.9
38	91.45	16.98	0.994	0.968
40	91.77	17.3	0.991	0.95
41	92.02	17.55	0.988	0.936
43	92.05	17.58	0.988	0.935
45	93.3	18.83	0.974	0.873
49	92.02	17.55	0.988	0.936

Test Sample File is from 37th author but not participated in building its author's automaton: 37_PhantomFortuneaNovel[MEMaryElizabethBraddon].pattern
Before removing common string, 37th author's automaton accepts 90.42 %
After removing common string, 37th author's automaton accepts 13.76 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	91.81	15.15	0.985	0.908
2	91.69	15.03	0.986	0.916
3	92.26	15.6	0.98	0.882
4	93.64	16.98	0.966	0.81
9	92.92	16.26	0.973	0.846
10	91.17	14.5	0.992	0.949
11	91.86	15.2	0.984	0.905
12	91.86	15.2	0.984	0.905
14	90.87	14.21	0.995	0.968
15	91.24	14.58	0.991	0.944
16	90.43	13.77	1	0.999
17	92.25	15.59	0.98	0.883
18	91.87	15.21	0.984	0.905
19	92.35	15.69	0.979	0.877
20	92.06	15.4	0.982	0.894
22	91.11	14.45	0.992	0.952
23	93.71	17.05	0.965	0.807
24	92.49	15.83	0.978	0.869
25	90.46	13.8	1	0.997

26	91.8	15.14	0.985	0.909
27	92.26	15.6	0.98	0.882
28	92.92	16.26	0.973	0.846
30	90.48	13.82	0.999	0.996
32	93.41	16.74	0.968	0.822
33	91.83	15.17	0.985	0.907
34	90.7	14.04	0.997	0.98
35	91.16	14.49	0.992	0.95
36	91.44	14.78	0.989	0.931
38	92.06	15.4	0.982	0.894
39	91.49	14.83	0.988	0.928
40	91.96	15.3	0.983	0.899
41	92.47	15.81	0.978	0.87
43	92.89	16.23	0.973	0.848
45	93.67	17.01	0.965	0.809
47	91.21	14.55	0.991	0.946
48	91.22	14.56	0.991	0.945
49	92.61	15.95	0.976	0.863

Test Sample File is from 38th author but not participated in building its author's automaton: 38_TheWorksofMaxBeerbohm[MaxBeerbohm].pattern
Before removing common string, 38th author's automaton accepts 92.96 %
After removing common string, 38th author's automaton accepts 14.03 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	93.19	14.25	0.998	0.985
2	93.54	14.6	0.994	0.961
3	93.27	14.34	0.997	0.978
4	94.56	15.62	0.983	0.898
9	93.54	14.6	0.994	0.961
10	93.36	14.42	0.996	0.973
17	93.41	14.47	0.995	0.97
19	93.76	14.82	0.991	0.947
22	93.01	14.07	0.999	0.997
23	94.42	15.49	0.985	0.906
24	93.41	14.47	0.995	0.97
27	93.23	14.29	0.997	0.982
28	94.29	15.35	0.986	0.914
32	94.25	15.31	0.986	0.916
33	93.19	14.25	0.998	0.985
40	93.05	14.12	0.999	0.994
41	93.19	14.25	0.998	0.985
43	93.76	14.82	0.991	0.947
45	95.4	16.46	0.974	0.852
49	93.67	14.73	0.992	0.952

Test Sample File is from 39th author but not participated in building its author's automaton: 39_GunmansReckoning[MaxBrand].pattern
Before removing common string, 39th author's automaton accepts 95.2 %
After removing common string, 39th author's automaton accepts 12.19 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	95.7	12.7	0.995	0.96
2	95.29	12.28	0.999	0.993
3	96.0	12.99	0.992	0.938
4	96.46	13.45	0.987	0.906
9	95.79	12.79	0.994	0.953
11	95.35	12.34	0.998	0.988
17	95.35	12.34	0.998	0.988
19	96.06	13.05	0.991	0.934
20	95.41	12.4	0.998	0.983
23	96.25	13.25	0.989	0.92
24	95.75	12.74	0.994	0.957
27	95.51	12.51	0.997	0.974

28	95.81	12.8	0.994	0.952
32	96.25	13.25	0.989	0.92
33	95.33	12.33	0.999	0.989
38	95.36	12.36	0.998	0.986
40	96.1	13.1	0.991	0.931
41	95.7	12.7	0.995	0.96
43	95.84	12.83	0.993	0.95
45	96.74	13.74	0.984	0.887
47	95.41	12.4	0.998	0.983
49	95.94	12.94	0.992	0.942

Test Sample File is from 40th author but not participated in building its author's automaton: 40_TheAdventureClubAfloat[RalphHenryBarbour].pattern
Before removing common string, 40th author's automaton accepts 94.76 %
After removing common string, 40th author's automaton accepts 17.2 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	95.32	17.75	0.994	0.969

Test Sample File is from 41st author but not participated in building its author's automaton: 41_TheSwordMaker[RobertBarr].pattern
Before removing common string, 41st author's automaton accepts 91.78 %
After removing common string, 41st author's automaton accepts 19.17 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	93.53	20.92	0.981	0.916
23	92.72	20.11	0.99	0.953
27	92.39	19.78	0.993	0.969
45	92.72	20.11	0.99	0.953
49	91.94	19.33	0.998	0.992

Test Sample File is from 42nd author but not participated in building its author's automaton: 42_MenandWomen[RobertBrowning].pattern
Before removing common string, 42nd author's automaton accepts 81.59 %
After removing common string, 42nd author's automaton accepts 5.77 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	90.2	14.39	0.905	0.401
2	88.83	13.01	0.918	0.444
3	89.82	14.01	0.908	0.412
4	91.62	15.81	0.891	0.365
5	83.58	7.76	0.976	0.744
6	85.14	9.32	0.958	0.619
7	88.45	12.64	0.922	0.456
8	85.14	9.32	0.958	0.619
9	90.25	14.43	0.904	0.4
10	88.69	12.87	0.92	0.448
11	90.63	14.81	0.9	0.39
12	88.97	13.16	0.917	0.438
13	87.55	11.74	0.932	0.491
14	88.55	12.73	0.921	0.453
15	89.26	13.44	0.914	0.429
16	88.69	12.87	0.92	0.448
17	90.77	14.96	0.899	0.386
18	89.4	13.58	0.913	0.425
19	91.1	15.29	0.896	0.377
20	90.25	14.43	0.904	0.4
21	87.84	12.02	0.929	0.48
22	90.3	14.48	0.904	0.398
23	91.67	15.85	0.89	0.364
24	90.44	14.62	0.902	0.395
25	89.26	13.44	0.914	0.429

26	89.3	13.49	0.914	0.428
27	91.95	16.14	0.887	0.357
28	91.95	16.14	0.887	0.357
29	86.42	10.6	0.944	0.544
30	87.93	12.12	0.928	0.476
31	89.54	13.72	0.911	0.421
32	91.58	15.76	0.891	0.366
33	90.39	14.58	0.903	0.396
34	89.07	13.25	0.916	0.435
35	89.73	13.91	0.909	0.415
36	88.59	12.78	0.921	0.451
37	89.35	13.54	0.913	0.426
38	90.06	14.25	0.906	0.405
39	89.45	13.63	0.912	0.423
40	89.82	14.01	0.908	0.412
41	90.11	14.29	0.905	0.404
43	90.87	15.05	0.898	0.383
44	86.13	10.32	0.947	0.559
45	93.37	17.56	0.874	0.329
46	85.71	9.89	0.952	0.583
47	89.21	13.39	0.915	0.431
48	87.84	12.02	0.929	0.48
49	92.05	16.23	0.886	0.356

Test Sample File is from 43rd author but not participated in building its author's automaton: 43_Oddsfish[RobertHughBenson].pattern
Before removing common string, 43rd author's automaton accepts 89.1 %
After removing common string, 43rd author's automaton accepts 18.0 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	89.62	18.52	0.994	0.972
23	89.39	18.29	0.997	0.984
27	89.35	18.25	0.997	0.986
45	89.88	18.79	0.991	0.958

Test Sample File is from 44th author but not participated in building its author's automaton: 44_ErewhonRevisited[SamuelButler].pattern
Before removing common string, 44th author's automaton accepts 78.32 %
After removing common string, 44th author's automaton accepts 10.62 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	84.15	16.46	0.931	0.645
2	84.71	17.02	0.925	0.624
3	85.08	17.38	0.921	0.611
4	87.6	19.91	0.894	0.533
6	79.75	12.06	0.982	0.881
7	83.67	15.97	0.936	0.665
8	78.46	10.77	0.998	0.986
9	86.0	18.3	0.911	0.58
10	84.01	16.31	0.932	0.651
11	84.91	17.21	0.922	0.617
12	83.86	16.16	0.934	0.657
13	81.09	13.39	0.966	0.793
14	84.22	16.53	0.93	0.642
15	85.49	17.79	0.916	0.597
16	83.35	15.65	0.94	0.679
17	86.27	18.57	0.908	0.572
18	83.74	16.04	0.935	0.662
19	85.78	18.08	0.913	0.587
20	85.9	18.21	0.912	0.583
21	82.43	14.73	0.95	0.721
22	84.05	16.36	0.932	0.649
23	87.14	19.45	0.899	0.546
24	84.3	16.6	0.929	0.64

25	82.6	14.9	0.948	0.713
26	83.37	15.68	0.939	0.677
27	87.75	20.05	0.893	0.53
28	86.73	19.03	0.903	0.558
29	79.75	12.06	0.982	0.881
30	81.36	13.66	0.963	0.777
31	83.64	15.95	0.936	0.666
32	85.25	17.55	0.919	0.605
33	84.83	17.14	0.923	0.62
34	83.84	16.14	0.934	0.658
35	83.76	16.07	0.935	0.661
36	83.79	16.09	0.935	0.66
37	82.5	14.8	0.949	0.718
38	85.61	17.91	0.915	0.593
39	84.3	16.6	0.929	0.64
40	84.47	16.77	0.927	0.633
41	86.56	18.86	0.905	0.563
43	86.27	18.57	0.908	0.572
45	86.75	19.06	0.903	0.557
46	81.11	13.42	0.966	0.791
47	83.84	16.14	0.934	0.658
48	84.56	16.87	0.926	0.63
49	86.66	18.96	0.904	0.56

Test Sample File is from 45th author but not participated in building its author's automaton: 45_PonkapogPapers[ThomasBaileyAldrich].pattern
Before removing common string, 45th author's automaton accepts 92.2 %
After removing common string, 45th author's automaton accepts 20.46 %

Automaton	%_before	%_after	ratio_before	ratio_after
4	93.18	21.44	0.989	0.954
23	93.06	21.32	0.991	0.96
32	92.45	20.71	0.997	0.988

Test Sample File is from 46th author but not participated in building its author's automaton: 46_BlackytheCrow[ThorntonWThorntonWaldoBurgess].pattern
Before removing common string, 46th author's automaton accepts 91.07 %
After removing common string, 46th author's automaton accepts 13.78 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	93.64	16.35	0.973	0.843
2	92.18	14.89	0.988	0.925
3	92.7	15.41	0.982	0.894
4	95.97	18.68	0.949	0.738
7	91.48	14.19	0.996	0.971
9	93.17	15.88	0.977	0.868
10	92.18	14.89	0.988	0.925
11	92.64	15.35	0.983	0.898
12	92.94	15.65	0.98	0.881
14	91.83	14.54	0.992	0.948
15	91.89	14.59	0.991	0.944
16	91.94	14.65	0.991	0.941
17	92.12	14.83	0.989	0.929
18	91.54	14.24	0.995	0.968
19	93.29	16.0	0.976	0.861
20	92.59	15.29	0.984	0.901
21	91.24	13.95	0.998	0.988
22	91.94	14.65	0.991	0.941
23	94.51	17.22	0.964	0.8
24	94.1	16.81	0.968	0.82
26	92.53	15.24	0.984	0.904
27	93.4	16.11	0.975	0.855
28	94.8	17.51	0.961	0.787
30	91.07	13.78	1	1

31	91.89	14.59	0.991	0.944
32	93.58	16.29	0.973	0.846
33	92.47	15.18	0.985	0.908
34	91.59	14.3	0.994	0.964
35	92.29	15.0	0.987	0.919
36	92.53	15.24	0.984	0.904
37	91.65	14.36	0.994	0.96
38	93.05	15.76	0.979	0.874
39	92.59	15.29	0.984	0.901
40	93.7	16.4	0.972	0.84
41	94.34	17.05	0.965	0.808
43	93.05	15.76	0.979	0.874
45	94.92	17.63	0.959	0.782
47	92.64	15.35	0.983	0.898
48	92.76	15.47	0.982	0.891
49	94.34	17.05	0.965	0.808

Test Sample File is from 47th author but not participated in building its author's automaton: 47_CastAdrift[TimothyShayArthur].pattern
Before removing common string, 47th author's automaton accepts 93.82 %
After removing common string, 47th author's automaton accepts 13.63 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	94.04	13.84	0.998	0.985
3	94.71	14.51	0.991	0.939
4	94.78	14.59	0.99	0.934
9	94.44	14.24	0.993	0.957
11	94.21	14.01	0.996	0.973
17	94.15	13.96	0.996	0.976
19	94.76	14.56	0.99	0.936
23	95.29	15.09	0.985	0.903
24	94.24	14.04	0.996	0.971
26	94.0	13.8	0.998	0.988
27	94.63	14.43	0.991	0.945
28	94.43	14.23	0.994	0.958
32	95.2	15.0	0.986	0.909
33	94.07	13.87	0.997	0.983
38	94.1	13.9	0.997	0.981
39	93.84	13.64	1	0.999
40	94.76	14.56	0.99	0.936
41	94.76	14.56	0.99	0.936
43	94.66	14.46	0.991	0.943
45	95.13	14.93	0.986	0.913
49	94.98	14.79	0.988	0.922

Test Sample File is 48th from author but not participated in building its author's automaton: 48_TomSwiftandHisSkyRacerortheQuickestFlightonRecord[VictorAppleton].pattern
Before removing common string, 48th author's automaton accepts 95.69 %
After removing common string, 48th author's automaton accepts 10.9 %

Automaton	%_before	%_after	ratio_before	ratio_after
1	96.25	11.45	0.994	0.952
2	95.93	11.14	0.997	0.978
3	96.17	11.38	0.995	0.958
4	96.85	12.06	0.988	0.904
9	96.39	11.6	0.993	0.94
10	95.71	10.92	1	0.998
17	96.42	11.62	0.992	0.938
18	95.74	10.94	0.999	0.996
19	95.91	11.11	0.998	0.981
20	95.71	10.92	1	0.998
23	96.97	12.18	0.987	0.895
24	96.63	11.84	0.99	0.921
26	95.84	11.04	0.998	0.987

27	96.03	11.23	0.996	0.971
28	96.15	11.36	0.995	0.96
32	96.85	12.06	0.988	0.904
33	95.79	10.99	0.999	0.992
38	95.79	10.99	0.999	0.992
39	96.37	11.57	0.993	0.942
40	96.8	12.01	0.989	0.908
41	96.68	11.89	0.99	0.917
43	96.13	11.33	0.995	0.962
45	97.34	12.54	0.983	0.869
49	97.07	12.28	0.986	0.888