khazanah informatika
Jurnal Ilmu Komputer dan Informatika

Role of Finite State Automata in Transliterating Latin Script into Javanese Script

Suprihatin[1]*, Imam Riadi[2], Furizal[3], Izzan Julda D.E Purwadi Putra[4]
*suprihatin@uad.ac.id
[1,2]Department of Information System
Universitas Ahmad Dahlan
Yogyakarta
[3,4]Master Program of Informatics
Universitas Ahmad Dahlan
Yogyakarta

Abstract- Writing a Javanese script is considered complicated writing for people who would learn it. The process of transliterating Latin into Javanese script cannot be done directly because each alphabet is only sometimes represented by one Javanese script. Javanese script is not depicted by one or more Latin letters, so if transliteration of Latin letters to Javanese letters is required, a parsing process is required. The rows of Javanese letters form a ligature with specific rules, so parsing is also needed to correctly arrange the rows of Javanese letters. This study aims to design a program to facilitate the transliteration of Latin script to Javanese script. Finite State Automata (FSA) is used to describe writing rules. This study is limited to lowercase letters only. Capital letters and number symbols are not discussed in this study. The results of the study are in the form of a program design that can transliterate Latin writing into Javanese. Experiments were carried out on as many as 4 structures of vowel-consonant variations. All syllabic structures that include CV, CPV, CVC, and CPVC have been tried. The transliteration results show conformity with a 100% accuracy rate by the rules of writing Javanese script. This research shows that the application of FSA can handle the transliteration of Latin letters into Javanese.

Keywords: FSA, Javanese Transliteration, Javanese Script, Parser

Article info: *submitted January 1, 2020, revised February 2, 2020, accepted March 15, 2020*

1. Introduction

Javanese script as a cultural symbol that needs to be preserved [1]–[4]. In writing Javanese script is not simple [5], [6], because each alphabet is not always represented with one Javanese script [7], [8]. Javanese characters are not represented by one or more Latin letters, so if transliteration of Latin letters to Javanese letters is needed, a parsing process is needed [9]. Parsing is also needed to form Javanese ligature correctly. This study aims to design a program to make it easier to transliterate Latin script to Javanese script. Finite State Automata (FSA) is used to describe writing rules [10]– [12]. There are 20 Javanese characters called carakan [8], [13], [14], which can sound even if they are not given swara/vowel support [15]. Clothing consists of 3 types, namely: vowels (a, i, u, e, é, o) [16], special dead letters (r, ng, h), incoming / punchy letters (r, y, l, w). Characters and sands will form ligatures. Figure 1 contains carakan character and couple character.

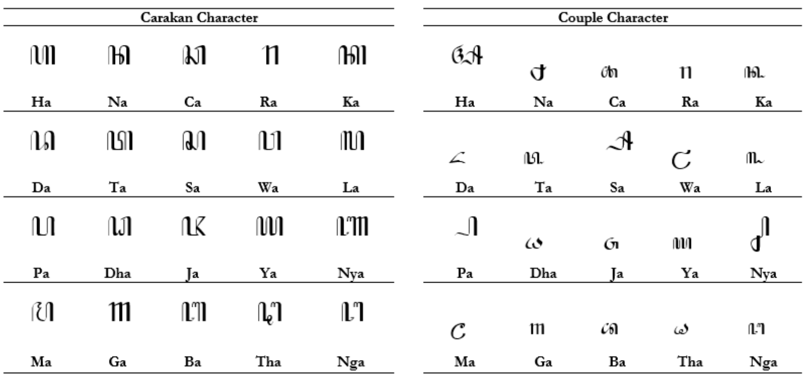

Figure 1. Carakan and Couple Characters

Couple characters are used if in front of them are consonants that are not special consonants letters. For example, the word: **abdi** of the letter b is a consonant is not special consonant so the letter d is the couple character. Figure 2 here is a table of one of the characters if it is sheathed.

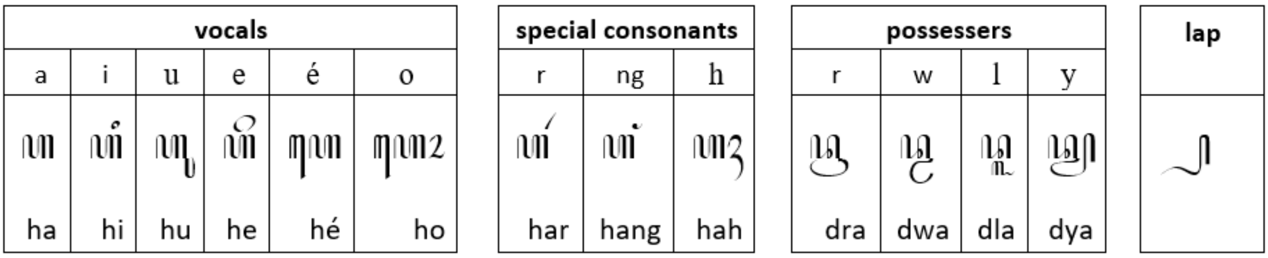

Figure 2. Sandangan script

Lap is used to close words that end in a consonant, especially at the end of a sentence.

Finite State Automata (FSA) can be used to transliterate Javanese script into Latin [17], [18]. This study discusses how the role of FSA helps in transliteration of Javanese script, namely: input in the form of Javanese characters while output in the form of syllables that can be read by the general public.

There are several previous studies that serve as a reference for transliteration of characters. Mahastama researched on transliteration from Javanese Latin into Javanese script utilizing a list of consonant-vowels and a list of symbols simultaneously using FSA to reduce the creation of complex Finite State Diagrams, the accuracy results obtained in the average word transliteration of 96.44% [19]. This study has a drawback of error in the model input which has not been fixed; Hence, it could not be applied in the high-complexity model. Sanjani et al. conducted research on Latin Balinese script to Balinese script with Noto Serif Balinese font using finite state machine for transliteration, this paper was able to convert text with accuration rate of 97.67% [20]; Unfortunately, due to the new proposed, its variant is still limited in certain words. Research conducted by Rachman et al. with the same method, namely FSA using objects from 3 types of Madurese language, namely Enja'-iyyeh, engghi-enten, and enggi-bunten obtained an accuracy of 85% which was still below average accuracy because of lack of corpus audio syllables of Madurese [21]. Using the same method, Wolf-Sonkin et al. applied it to South Asian regional languages with a result of 54% relative error rates by using transliteration transducers with output readings from input bands. This study requires additional integration with other modules that can represent other deploying methods [22]. In addition, research was conducted by Pratama et al. entitled "Design and Build an Application for Transliteration of Latin Script into Sasak Script using Rule-Based Algorithm using the android program". This study used a rule-based algorithm with Sasak script objects with an accuracy rate of 85.39% from 1650 test data. This research have not conducted in reverse translation from aksara Sasak into aksara Latin in widely-range data [23]. Karmani et al. conducted research using transliteration tools from Arabic script to Latin script or vice versa. TACA-TA is one of the tools developed. This study compared the developed tool with the existing transliteration machine, EiKtub. The results are more accurate TACA-TA with an accuracy rate of ±82% in words and characters. Since its low accuracy in longer words, it needs more evaluation with various transliteration tools [24]. Rajapaksha et al. applied the transliteration of Sri Lankan Sinhalese and Tamil scripts into English. The study presents a hybrid approach using machine learning and statistical machine translation for transliteration and obtained the highest accuracy of 93.7 in Sinhalese to English transliteration. This research should be developed with different location names, organizational names, and designations for the sake of improving quality [25]. Research conducted by Slamet et al. entitled "Latin to Sundaese Script Conversion using Finite State Automata Algorithm" used respondents to determine the level of accuracy of the tools developed. 30 respondents using the Likert technique as data processing, the results got a precision level of 79.8%. However, its precision level was far lower than the former research that has been mentioned; Therefore, it is better to expand with other methods in order to achieve high efficiency and effectiveness [26]. Birawidya et al. used a finite state machine to assign UNICODE to each character of Balinese script text by checking the characters in each state. The usability scale of the system given to 20 respondents received a score of 68 and was considered to have qualified as a transliteration of Balinese script text into Latin [27].

Starting from those points, this research would conduct the evaluation model in order to gain high accuracy and efficiency with different attempts. Thus, this study uses FSA to marshal and parse sentences into syllables, regarding to the table of these syllables transliterated into the Javanese script. Formally, Finite State Automata (FSA) is defined as 5 tuples [28]–[30] (Q, ∑, δ, q0, F) [31]–[34], in which Q is a finite set of states [35], ∑ is a finite set of input symbols (Alphabet) [36], [37], q0 in Q is the initial state [38], [39], F ⊆ Q is the set of finish states [37], [40], [41], and δ is a transition function that maps Q x ∑ to Q [41]–[44]. An FSA can be described as a directional graph whose points represent its states [45], [46]. If a state q transitions to state p in input a, then a line labeled A connects state q to state p in that graph.

2. Methods

The stages of the research are depicted in Figure 2. The input is in the form of sentences and the output is in the form of transition results. There are three parser processes, each assisted by an FSA. Javanese doesn't have syllables that start with vowels, for that the input needs to be normalized by adding the letter h to syllables that start with vowels.

The parser 1 process normalizes input by adding the letter h to syllables beginning with vowels and removing excess blanks. The parser 2 process uses parser 1 output as input. Parser 2 functions to convert normal input into a series of Javanese consonants (h, n, c, r, k, d, t. s, w, l, p, dh, j, y, ny, m, g, b, th, ng), a row of Javanese vowels (a, i, u, e, é, o) and blank.

The output of parser 2 is processed by parser 3 and the transliteration table generates the transliteration. The study begins by normalizing the lowercase input string, adding the letter h if a word begins with a vowel, or two or more consecutive vowels [47], [48]. The formation of normal strings using FSA and the parser aims to conform to Javanese phonemes. The results of string normalization will be processed to know consonants, vowels or spaces (blanks). This recognition of consonants, vowels and spaces also requires FSA and parsers. The next step is for the FSA and its parser to recognize the structure of syllables and their transliterations. The process step if depicted looks like Figure 3.

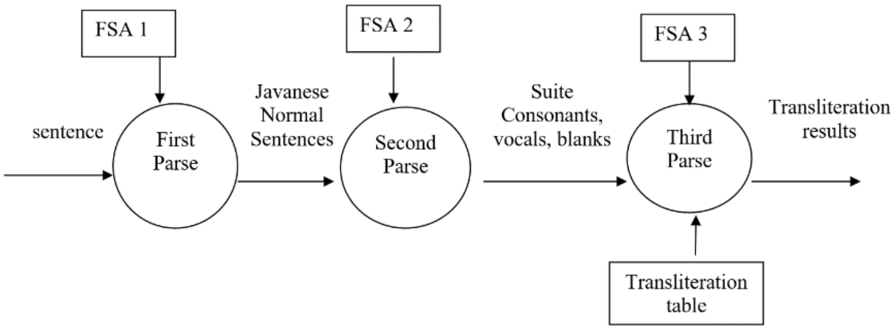

Figure 3. Transliteration Process Diagram

3. Result and Discussion

Javanese writing does not recognize words that begin with vowels (such as: anda, orang), or words that contain 2 or more consecutive vowels (such as: aan, taat). If it happens, you will add the letter h in front of the vowel, (like anda → handa, orang → horang) and like (aan → hahan, taat → tahat). Figure 4 here is the FSA to normalize the input string.

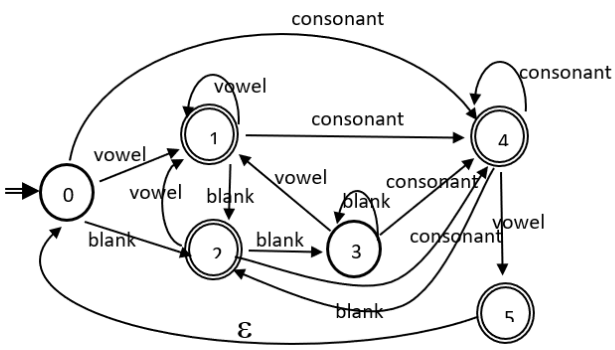

Figure 4. FSA1 Normal Javanese Sentence

Formally, the FSA diagram 1 in Figure 4 is written as follows:

FSA = (Q, ∑, δ, q0, F).

Q = {0,1,2,3,4,5}

C = {‘h’,’n’,’c’,’r’,’k’,’d’,’t’,’s’,’w’,’p’,’j’,’y’,’m’,’g’,’b’}

V = {’a’,’i’,’u’,’e’,’é’,’o’}

∑ = C ∪ V ∪ {blank}

q0 = 0

Table 1. FSA 1 Transition Function

∑	C																V						blank	ε
q	‘h'	‘n'	‘c'	‘r'	‘k'	‘d'	‘t'	‘s'	‘w'	‘l'	‘p'	‘j'	‘y'	‘m'	‘g'	‘b'	‘a'	‘i'	‘u'	‘e'	‘é'	‘o’		
0	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	1	1	1	1	1	1	2	
1	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	1	1	1	1	1	1	2	
2	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	1	1	1	1	1	1	3	
3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	1	1	1	1	1	1	3	
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	2	
5																								0

F = {1, 2, 4, 5}

Figure 4 is also useful for removing excessive blanks so that between two words there is only one blank. Phonemes in Javanese there are 6 vowels (a, i, u, e, é, o) and 20 consonants namely h, n, c, r, k, d, s, w, l, p, dh, j, y, ny, m, g, b, th, ng. FSA for recognizing phonemes and spaces, as in Figure 5.

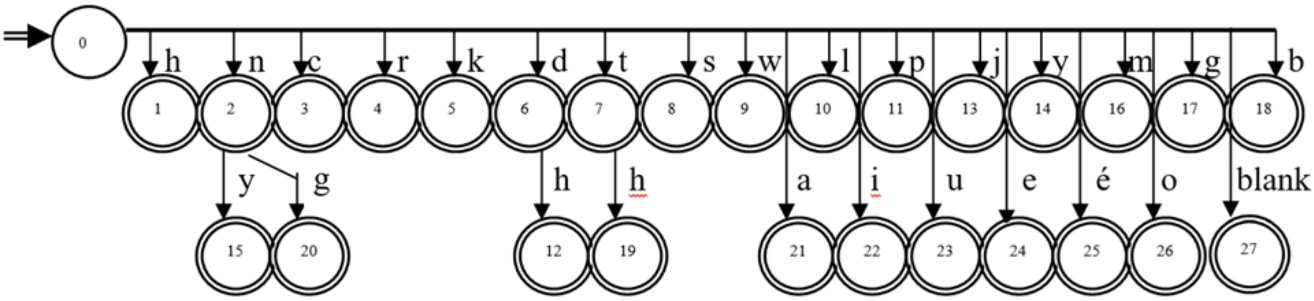

Figure 5. FSA2 Introduction to Javanese Phonemes and Blanks

Formally, the FSA diagram 2 in Figure 5 is written as follows:
FSA = (Q, ∑, δ, q0, F).
Q = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27}
∑ = {'h','n','c','r','k','d','t','s','w','p','j','y','m','g','b', 'a','i','u','e','é','o'}
q0 = 0

Table 2. FSA 2 Transition Function

∑ q	'h'	'n'	'c'	'r'	'k'	'd'	't'	's'	'w'	'l'	'p'	'j'	'y'	'm'	'g'	'b'	'a'	'i'	'u'	'e'	'é'	'o'	blank
0	1	2	3	4	5	6	7	8	9	10	11	13	14	16	17	18	21	22	23	24	25	26	27
2													15		20								
6	12																						
7	19																						

F = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27}

The two circles indicate the final status of phoneme recognition. Figure 5 is FSA 2 consisting of 27 finish states which show the structure of Javanese phonemes. Final states 1 through 20 indicate consonant phonemes, statuses 21 through 26 indicate vowels, and 27 blanks.

The syllabic structure pattern consists of 4 namely: CV (example: pari, tari), CVC (example: muntah, tindak), CCV (example: dwi), CCVC (example: kram, tyas). The second of the 2 consecutive consonants is called the possesser consonant consisting of: r, w, l, y. To simplify further, the syllable structure is CV, CVC, CPV, CPVC. The FSA image looks like Figure 6.

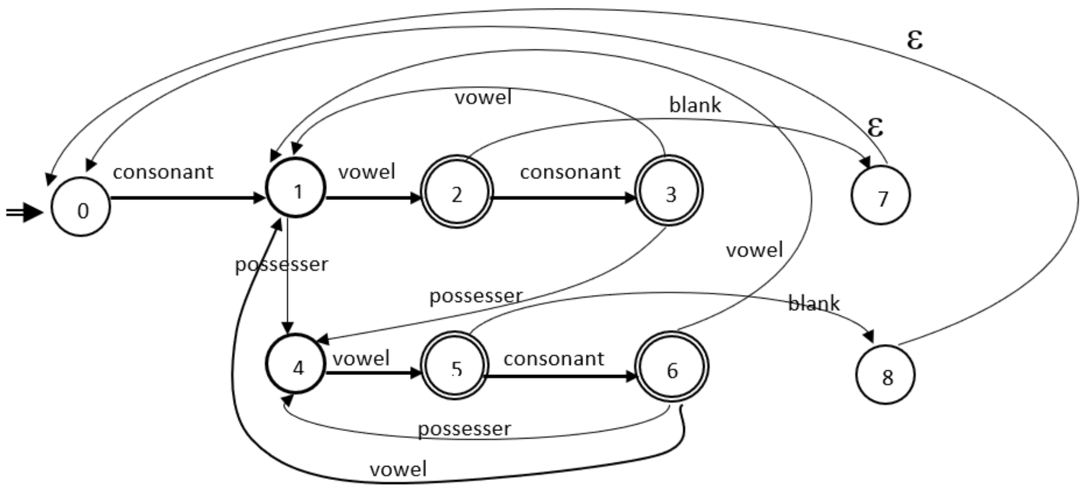

Figure 6. FSA3 Javanese Syllabic Structure

Formally, the FSA diagram 3 in Figure 6 is written as follows:
FSA = (Q, ∑, δ, q0, F).
Q = {0,1,2,3,4,5,6,7,8}
C = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}
V = {21,22,23,24,25,26}
P = {4, 8, 14}
∑ = C ∪ V ∪ P ∪ {blank}
q0 = 0

Table 3. FSA 3 Transition Function

∑	C																				V						blank	ε
q	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1								
1				4					4	4				4							2	2	2	2	2	2		
2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3							7	
3				4					4	4				4							1	1	1	1	1	1		
4																					5	5	5	5	5	5		
5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6							8	
6				4					4	4				4							1	1	1	1	1	1		
7																												0
8																												0

F = {2, 3, 5, 6}

The final status consists of four statuses with respect to four kinds of Javanese syllabic structures. Status 2 pertains to the CV pattern, status 3: CVC, status 5: CPV, and status 6: CPVC. The three FSA images in Figures 4-6 will form three algorithms, while the algorithms are as follows:

1. Normal Algorithm of Javanese Sentences

The Javanese Normal Sentence algorithm is useful for converting general sentences into standard/normal sentences that can be transliterated into Javanese script. Figure 4 FSA1 can be created Algorithm 1 as follows:

```
1    function Parser1(s: String): string;
2    var i, N,q: integer;
3      lw, helper: string;
4    function d (state: integer; k: char): integer;
5    const
6       vowel= ['a','e','i','o','u','é'];
7       blank = ' ';
8    var q: integer;
9    begin
10    q ← 0;
11    case state of
12    0: if k in vowel then q ← 1 else
13       if k = blank then q ← 2 else q ← 4;
14    1: if k in vowel then q ← 1 else
15       if k = blank then q ← 2 else q ← 4;
16    2: if k in vowel then q ← 1 else
17       if k = blank then q ← 3 else q ← 4;
18    3: if k in vowel then q ←1 else
19       if k = blank then q ← 3 else q ← 4;
20    4: if k in vowel then q ← 5 else
21       if k = blank then q ← 2 else q ← 4;
22    end;
23    result ← q;
24   end;
25   begin
26     helper ← ''; lw ← LowerCase(s);
27     N ← length(s);
28     q ← 0;
29      for i: = 1 to N do
30        begin
31          q ← d(qlw[i]);
32          case q of
33           1: helper ← helper + 'h' + lw[i];
34           2: helper ← helper + lw[i];
35           4: helper ← helper + lw[i];
36           5: begin helper ← helper + lw[i]; q ← 0 ; end;
37          end;
38        end;
39    result ← helper;
40    end;
```

Lines 4 through 24 are FSA Pseudocode 1. Line 26 is to make the sentence lowercase. On line 33, if state 1 is input in the form of a vowel so that the letter h is added in front of the vowel. If state 2 (line 34) or state 4 (line 35) then the result is equal to the input. Meanwhile, if state 5 (line 36) then the result is equal to the input value and the state returns to state 0. Then, if the state is 3 then no action means removing the blank input.

2. Phoneme Parser Algorithm

The Phoneme Parser algorithm is useful for converting normal Javanese sentences into phoneme tokens. Figure 5 FSA2 can be created Algorithm 2 as follows:

```
procedure Parser2(KalNor: String);
var
  kal : string;
  q,i, lihat,len : integer;

function d(state:integer;k:char):integer;
var q: integer;
begin
 case state of
 0: if k = 'h' then q ← 1  else
    if k = 'n' then q ← 2  else
    if k = 'c' then q ← 3  else
    if k = 'r' then q ← 4  else
    if k = 'k' then q ← 5  else
    if k = 'd' then q ← 6  else
    if k = 't' then q ← 7  else
```

```
17        if k = 's' then q ← 8  else
18        if k = 'w' then q ← 9  else
19        if k = 'l' then q ← 10  else
20        if k = 'p' then q ← 11  else
21        if k = 'j' then q ← 13  else
22        if k = 'y' then q ← 14  else
23        if k = 'm' then q ← 16  else
24        if k = 'g' then q ← 17  else
25        if k = 'b' then q ← 18  else
26        if k = 'a' then q ← 21  else
27        if k = 'i' then q ← 22  else
28        if k = 'u' then q ← 23  else
29        if k = 'e' then q ← 24  else
30        if k = 'o' then q ← 25  else
31        if k = 'é' then q ← 26  else
32        if k = ' ' then q ← 27  else q ←err;
33     2: if k = 'y' then q ← 15 else
34        if k = 'g' then q ← 20 else q ← err;
35     6: if k = 'h' then q ← 12 else q ← err;
36     7: if k = 'h' then q ← 19 else q ← err;
37     27: if k = ' ' then q ← 27 else q ← err;
38     else q ← err;
39     end;
40     result ← q;
41    end;
42
43    function lookhead(p,i:integer):integer;
44    var look : integer ;
45    begin
46     if i > len then look ← err else
47     look ← d(p,kal[i]);
48     Lookhead ← look;
49    end;
50
51    begin
52     kal := Normaljawa(KalNor);
53     len ← length(kal);
54     q := 0; i ← 1; JumTok ← 0;
55     while ( i <= len ) do
56     begin
57      q ← d(q,kal[i]);
58      if q = err then
59      begin
60      q := 0;
61      end  else
62      begin
63      lihat ← lookhead(q,i+1);
64      if lihat= err then
65      begin
66       JumTok ← JumTok + 1;
67       Tok[JumTok] ← q;
68       q := 0;
69      end;
70     end;
71     i ← i + 1;
72     end;
73    end;
```

FSA2 is embodied in the functions on line 6 through line 41. The lookhead function on lines 43 through 49 is useful for seeing the next state with the next input. The Tok data structure is a global variable integer array that serves to store a series of state created by the second FSA. On lines 63 through 69, if the next state is error then the state is stored in the Tok variable line 67.

3. Syllable Parser Algorithm

The Syllable Parser algorithm is useful converting phoneme token strings into syllables to transliterate. Figure 6 FSA3 can be created Algorithm 3 as follows:

```
1     procedure Parser3;
2     var
3       q,i, lihat : integer;
4       save2,save5 : string;
5
6     function d2(state, k: integer): integer;
7     const
8        Con = [1..20];
9        Poss = [4,9,10,14];
10       Vow = [21..26];
11    var q: integer;
12    begin
13     case state of
14          0: if k in Con then q ← 1  else q ← err;
15          1: if k in Vow then q ← 2  else
16             if k in Poss then q ← 4  else q ← err;
17          2: if k in Con then q ← 3  else
18             if k = 27  then q ← 7  else q ← err;
19          3: if k in Vow then q ← 1  else
20             if k in Poss then q ← 4 else q ← err;
21          4: if k in Vow then q ← 5  else q ← err;
22          5: if k in Con then q ← 6  else
23             if k = 27   then q ← 8  else q ← err;
24          6: if k in Vow then q ← 1  else
25             if k in Poss then q ← 4 else q ← err;
26       else q ← err;
27      end;
28    Result ← q;
29    end;
30    function lookhead(p,i:integer):integer;
31    var look : integer ;
32    begin
33     if i > jumtok then look ← err else
34     look ← d2(p,Tok[i]);
35     lookhead ← look;
36    end;
37    begin
38     q ← 0; i ← 1; s ←'';
39     couple ← false;
40     while ( i <= jumtok ) do
41     begin
42      q ← d2(q,tok[i]);
43      case q of
44      100 : q := 0;
45      1 : begin
46           one ← tok[i];  // dapat konsonan
47           if(couple) then one ← tok[i] + 20 ;
48           couple ← false ;
49          end;
50      4 : begin
51           four ← tok[i];  // dapat panjingan
52          end;
53      2: begin
54           two ← tok[i];
55           save2 ← trans(2);
56           if i = jumtok then s ← s + save2;
57         end;
58      5: begin
59           five ← tok[i];
60           save5 ← trans(5);
61           if i = jumtok then  s ← s + save5;
62         end;
63      3: begin
64           three ← tok[i];
65           lihat ← lookhead(q,i+1);
66           case lihat of
67           1,4: begin
68               s ← s + save2;
69               q ← 1;
70               one ← tok[i];
71              end
72           else
73           begin
74            s← s + trans(3);
75            if (three in pati) then couple ← false
76            else  couple ← true ;
77            q ← 0;
78            init;
79           end;
80         end;
81         end;
82      6: begin
83           enam ← tok[i];
84           lihat ← lookhead(q,i+1);
85           case lihat of
86           1,4: begin
87               s ← s + save5;
88               q ← 1;
89               one ← tok[i];
90              end
91           else
92           begin
93             s ← s + trans(6) ;
94                     if (enam in pati) then couple ← false
95                     else  couple ← true ;
96             q ← 0;
97             init;
98           end;
99         end;
100     end;
101     7:   begin
102           s ← s + save2 ;
103           q := 0; couple ← false;
104           init;
105          end;
106     8:   begin
107           s ← s + save5 ;
108           q ← 0; couple ← false;
109           init;
110          end;
111     end;
112    i ← i + 1;
113    end;
114    tulispangku;
115   end;
```

Con is the set of consonant phonemes numbered 1 through 20. **Vow** is the set of vowel phonemes that are phoneme numbers: 21 to 25, **Poss** is the phoneme of Possesser which is numbered 4, 9, 10, and 14. FSA3 is embodied in functions **d2** line 6 through line 29. The **lookhead** function on lines 30 to 36 is useful for seeing the next state with the next input. The **Tok** data structure is an integer array global variable serving as input by the third FSA.

S is a global variable to save the transliteration results. The variable **one** means to save the input from a state to state 1. Variable **two** means saving input from a state to state 2. So are **three**, **four**, and **five**. Line 47 of the variable **couple** to indicate if the input is a pair, then list the pairs simply by adding the number

20 for transliteration needs. Line 55 variable **save2** to store the result of transliteration at state 2, line 56 if the last input then **save2** goes to global variable **s**. So do lines 60 and 61, if the last input then **save5** goes into the global variable **s**.

In lines 67 through 70 of state 3, if the next state is 2 or 4, save2 is added to the variable s. If otherwise, it means stop at state 3, then add the variable s with the transliteration of state 3 line number 74. Line 75, the pair flag is false if the consonants in starch (r, h, ng). Otherwise, give the value true. Likewise, state 6 on line 82 through line 100.

State 7, line 101 through line 105, then the state ends in state 2 to store the transliteration result of state 2. Likewise, state 8, line 196 until 111 is a transliteration of state 5. False pair flags end in state 7 or 8. The init procedure is for initialization of variables one, two, six with a value of 0. The procedure is to add the Lap script if the consonant ending is not consonant special. This stage will simulate the results of the transliteration process with several inputs, the first parser, the second parser and the third parser to get the transliteration. Table 4 is an example of the transliteration.

Table 4. Examples of Transliteration Results

No	Structure	Input	Normal Javanese	Javanese phonemes	Transliteration Results
1	CV	ma TI	Lowercase: ma ti State: 0->4->5->2->3->3->3->3->3->4->5 Normal Results: ma ti	State: 16->21->27->7->22 Phonemes: m->a-> ->t->i	State: 0->1->2->7->1->2 Result: 
2	CPV	Kroco	Lowercase: kroco State: 0->4->4->5->4->5 Normal Results: kroco	State: 5->4->25->3->25 Phonemes: k->r->o->c->o	State:0->1->4->5->6->2 Result: 
3	CVC	Konco lawas	Lowercase: konco lawas State: 0->4->5->4->4->5->2->3->3->3->3->3->3->3->3->3->3->3->3->3->4->5->4->5->4 Normal Results: konco lawas	State: 5->25->2->3->25->27->10->21->9->21->8 Phonemes: k->o->n->c->o-> ->l->a->w->a->s	State:0->1->2->3->1->2->7->1->2->3->2->3 Result: 
4	CPVC	Kripik anget	Lowercase: kripik anget State: 0->4->4->5->4->5->4->2->1->4->4->5->4 Normal Results: kripik hanget	State: 5->4->22->11->22->5->27->1->21->20->24->7 Phonemes: k->r->i->p->i->k-> ->h->a->ng->e->t	State:0->1->4->5->6->2->3->100->1->2->3->2->3 Result: 

4. Conclusion

Based on the results of the experiment, the algorithm has been able to transliterate Latin script to Javanese script. Experiments were carried out as many as 4 structures of vowel consonant variations. All syllabic structures that include CV, CPV, CVC, and CPVC have been tested. The transliteration results show conformity with a 100% accuracy rate in accordance with the rules of writing Javanese script. This research shows that the application of FSA can handle the transliteration of Latin letters into Javanese.

Reference

[1] F. K. Sari, "The Local Wisdom in Javanese Thinking Culture within Hanacaraka Philosophy," *Diksi*, vol. 28, no. 1, Mar. 2020, doi: 10.21831/diksi.v28i1.31960.

[2] P. Ardhianto, W.-H. Hsieh, S. A. Mahanaim, and C.-H. Chen, "Cross-Cultural Concepts in Cultural Product Design," in *Proceedings of the 3rd International Conference on Arts and Design Education (ICADE 2020)*, Paris, France: Atlantis Press, 2021. doi: 10.2991/assehr.k.210203.031.

[3] P. Bintoro and A. Harjoko, "Lampung Script Recognition Using Convolutional Neural Network," *IJCCS (Indonesian Journal of*

Computing and Cybernetics Systems), vol. 16, no. 1, p. 23, Jan. 2022, doi: 10.22146/ijccs.70041.

[4] C. A. Purnomo, S. A. Mahanaim, F. Riva Lo, V. Ardaniah, and P. Ardhianto, "Visual Language of Javanese Script on Shoe Design as Cultural Identity," *Gelar : Jurnal Seni Budaya*, vol. 19, no. 2, pp. 105–113, Dec. 2021, doi: 10.33153/glr.v19i2.3951.

[5] G. N. Adli Kesaulya, A. Fariza, and T. Karlita, "Javanese Script Text Image Recognition Using Convolutional Neural Networks," in *2022 International Electronics Symposium (IES)*, IEEE, Aug. 2022, pp. 534–539. doi: 10.1109/IES55876.2022.9888527.

[6] A. Susanto, I. U. W. Mulyono, C. A. Sari, E. H. Rachmawanto, and D. R. I. M. Setiadi, "Javanese Script Recognition based on Metric, Eccentricity and Local Binary Pattern," in *2021 International Seminar on Application for Technology of Information and Communication (iSemantic)*, IEEE, Sep. 2021, pp. 118–121. doi: 10.1109/iSemantic52711.2021.9573232.

[7] A. A. Nggofur and S. Dwijonagoro, "Improving Javanese Letter Reading Skill through the Iqro Script Method," *International Journal Corner of Educational Research*, vol. 1, no. 2, pp. 69–77, Oct. 2022, doi: 10.54012/ijcer.v1i2.95.

[8] A. Susanto, C. Atika Sari, I. U. W. Mulyono, and M. Doheir, "Histogram of Gradient in K-Nearest Neighbor for Javanese Alphabet Classification," *Scientific Journal of Informatics*, vol. 8, no. 2, pp. 289–296, Nov. 2021, doi: 10.15294/sji.v8i2.30788.

[9] A. R. Widiarti and R. Pulungan, "A method for solving scriptio continua in Javanese manuscript transliteration," *Heliyon*, vol. 6, no. 4, p. e03827, Apr. 2020, doi: 10.1016/j.heliyon.2020.e03827.

[10] A. Yuniar Rahman, F. Wanditya Setiawan, A. Lia Hananto, and B. Setyawan, "Modeling financial statements for small and medium businesses in Worm-Made Fertilizer Using Finite State Automata (FSA)," *J Phys Conf Ser*, vol. 1908, no. 1, p. 012025, Jun. 2021, doi: 10.1088/1742-6596/1908/1/012025.

[11] E. Fulop and N. Pataki, "Symbolic Execution with Finite State Automata," in *2019 IEEE 15th International Scientific Conference on Informatics*, IEEE, Nov. 2019, pp. 000293–000298. doi: 10.1109/Informatics47936.2019.9119287.

[12] I. Lobzhanidze, "Computational Modeling," in *Finite-State Computational Morphology*, Cham: Springer International Publishing, 2022, pp. 117–166. doi: 10.1007/978-3-030-90248-3_3.

[13] D. Lisufiana, M. Khumaedi, and T. Supriyatno, "The Developing of Mengalihaksarakan Serat Wulangreh Pupuh Gambuh Assessment Instruments for Eighth Class," in *Proceedings of the 6th International Conference on Science, Education and Technology (ISET 2020)*, 2022. doi: 10.2991/assehr.k.211125.063.

[14] I. Prihandi, I. Ranggadara, S. Dwiasnati, Y. S. Sari, and Suhendra, "Implementation of Backpropagation Method for Identified Javanese Scripts," *J Phys Conf Ser*, vol. 1477, no. 3, p. 032020, Mar. 2020, doi: 10.1088/1742-6596/1477/3/032020.

[15] N. Nabilah and F. Nikmah, "The Relationship of Aksara Jawa as Local Folklore with Moderate and Progressive Islamic Education," *Annual International Conference on Islamic Education for Students*, vol. 1, no. 1, Jun. 2022, doi: 10.18326/aicoies.v1i1.257.

[16] H. I. Sa'adah and B. Setiawan, "Simbol Bunyi Vokal Huruf Hijaiyyah Dan Huruf Carakan Jawa (Studi Analisis Linguistik Fonologi)," *Al-Fakkaar*, vol. 1, no. 1, pp. 101–122, 2020.

[17] Suprihatin, "Penerapan Finite State Automata dalam Mengalihaksarakan Tulisan Aksara Jawa ke Tulisan Huruf Latin, [Finite State Automata for Converting Javanese Letters to Latin Letters]," Master Thesis, Universitas Gadjah Mada, Yogyakarta, 2003.

[18] Y. Zhou, F. Huang, and H. Chen, "Combining probability models and web mining models: A framework for proper name transliteration," *Information Technology and Management*, vol. 9, no. 2, pp. 91–103, 2008, doi: 10.1007/s10799-007-0031-9.

[19] A. W. Mahastama, "Model Berbasis Aturan untuk Transliterasi Bahasa Jawa dengan Aksara Latin ke Aksara Jawa," *Jurnal Buana Informatika*, vol. 13, no. 02, pp. 146–154, 2022, doi: 10.24002/jbi.v13i02.6526.

[20] D. A. P. P. Sanjani, G. Indrawan, and I. G. A. Gunadi, "Pengembangan Metode Pemisahan Suku Kata Untuk Transliterasi Teks Latin Ke Bali Berbasis Finite State Machine Dengan Huruf Noto Serif Bali," *Jurnal Ilmu Komputer Indonesia (JIK)*, vol. 6, no. November, pp. 3–6, 2021, [Online]. Available: https://doi.org/10.23887/jik.v6i2.3659

[21] F. H. Rachman, Qudsiyah, and F. Solihin, "Finite State Automata Approach for Text to Speech Translation System in Indonesian-Madurese Language," *J Phys Conf Ser*, vol. 1569, no. 2, pp. 0–7, 2020, doi: 10.1088/1742-6596/1569/2/022091.

[22] L. Wolf-Sonkin, V. Schogol, B. Roark, and M. Riley, "Latin script keyboards for South Asian languages with finite-state normalization," *FSMNLP 2019 - 14th International Conference on Finite-State Methods and Natural Language Processing, Proceedings*, pp. 108–117, 2019, doi: 10.18653/v1/w19-3114.

[23] P. W. Pratama, A. Aranta, and F. Bimantoro, "Rancang Bangun Aplikasi Transliterasi Aksara Latin menjadi Aksara Sasak Menggunakan Algoritma Rule Based Berbasis Android," *Jurnal Teknologi Informasi, Komputer, dan Aplikasinya (JTIKA)*, vol. 3, no. 2, pp. 232–243, 2021.

[24] N. Karmani, H. Soussou, and A. Alimi, "Tunisian Arabic chat alphabet transliteration using probabilistic finite state transducers," *International Arab Journal of Information Technology*, vol. 16, no. 2, pp. 295–303, 2019.

[25] H. S. Priyadarshani, M. D. W. Rajapaksha, M. M. S. P. Ranasinghe, K. Sarveswaran, and G. V. Dias, "Statistical Machine Learning for Transliteration: Transliterating names between Sinhala, Tamil and English," *Proceedings of the 2019 International Conference on Asian Language Processing, IALP 2019*, pp. 244–249, 2019, doi: 10.1109/IALP48816.2019.9037651.

[26] C. Slamet, Y. A. Gerhana, D. S. Maylawati, M. A. Ramdhani, and N. Z. Silmi, "Latin to Sundanese script conversion using Finite State automata algorithm," *IOP Conf Ser Mater Sci Eng*, vol. 434, no. 1, pp. 0–10, 2018, doi: 10.1088/1757-899X/434/1/012063.

[27] C. O. Birawidya,, vol. 7, pp. 41–46, 2022.

[28] M. M. Sulaiman, R. Andrianto, and M. A. Yulianto, "Mobile Learning Application for Language and Automata Theory using Android-based," *Jurnal Online Informatika*, vol. 5, no. 2, p. 176, Dec. 2020, doi: 10.15575/join.v5i2.630.

[29] K. Kumar, "Design of vending machine through implementation of visual automata simulator and finite state machine," *International Journal of Research in Circuits, Devices and Systems*, vol. 2, no. 2, pp. 60–64, 2021.

[30] J. Mantik, H. B. Kusnawan, W. Gata, and L. Kurniawati, "Simulation Signature-Based Carving Raster Image Using Finite State Automata," *Jurnal Mantik*, vol. 6, no. 1, pp. 202–209, 2022.

[31] T. Hari Wicaksono, F. Dwiki Amrizal, H. Atun Mumtahana, and J. Setia Budi No, "Pemodelan Vending Machine dengan Metode FSA (Finite State Automata)," *DoubleClick: Journal of Computer and Information Technology*, vol. 2, no. 2, pp. 66–69, 2019, [Online]. Available: http://e-journal.unipma.ac.id/index.php/doubleclick/article/view/3901

[32] C. Fikri, W. Gata, B. Pratama, K. S. Parthama, and T. Haryanti, "Penerapan Finite State Automata Pada DesainVending Machine Alat Tulis Sekolah," *Jurnal Sistem Komputer TGD*, vol. 1, no. 6, pp. 296–302, 2022, [Online]. Available: https://ojs.trigunadharma.ac.id/index.php/jskom

[33] B. Asrun, “Konsep Finite State Automata dalam Proses Pendaftaran Ujian Skripsi di Fakultas Teknik Komputer UNCP,” *Jurnal Ilmiah Information Technology d'Computare*, vol. 10, pp. 5–9, 2020.

[34] B. Chen, K. Leahy, A. Jones, and M. Hale, “Differential privacy for symbolic systems with application to Markov Chains,” *Automatica*, vol. 152, p. 110908, Jun. 2023, doi: 10.1016/j.automatica.2023.110908.

[35] E. Šestáková, O. Guth, and J. Janoušek, “Inexact tree pattern matching with 1-degree edit distance using finite automata,” *Discrete Appl Math (1979)*, vol. 330, pp. 78–97, May 2023, doi: 10.1016/j.dam.2023.01.003.

[36] P. Samuel, S. Subbaiyan, B. Balusamy, S. Doraikannan, and A. H. Gandomi, “A Technical Survey on Intelligent Optimization Grouping Algorithms for Finite State Automata in Deep Packet Inspection,” *Archives of Computational Methods in Engineering*, vol. 28, no. 3, pp. 1371–1396, May 2021, doi: 10.1007/s11831-020-09419-z.

[37] F. James, I. Ray, and D. Medhi, “Situational Awareness for Smart Home IoT Security via Finite State Automata Based Attack Modeling,” in *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, IEEE, Dec. 2021, pp. 61–69. doi: 10.1109/TPSISA52974.2021.00007.

[38] V. F. Hakim, W. Gata, S. Rahayu, H. Setiawan, and H. B. Novitasari, “Implementation of Finite State Automata in Graphic Design Class Process Online,” *JUSIKOM PRIMA (Jurnal Sistem Informasi dan Ilmu Komputer Prima)*, vol. 6, no. 1, 2022.

[39] Z. Yang *et al.*, “FSAFlow: Lightweight and Fast Dynamic Path Tracking and Control for Privacy Protection on Android Using Hybrid Analysis with State-Reduction Strategy,” in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, May 2022, pp. 2114–2129. doi: 10.1109/SP46214.2022.9833764.

[40] N. Lediwara, H. Saragih, R. A. G. Gultom, E. Mukmin, G. Rahmad Zuwa, and R. Hadi Fajri, “Finite State Automata On The Administrative Selection System Of New Student Admission in Universitas Pertahanan Republik Indonesia,” in *2022 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, IEEE, Oct. 2022, pp. 95–98. doi: 10.1109/ICACSIS56558.2022.9923514.

[41] S. Chakraborty, R. Grossi, K. Sadakane, and S. R. Satti, “Succinct representation for (non)deterministic finite automata,” *J Comput Syst Sci*, vol. 131, pp. 1–12, 2023, doi: 10.1016/j.jcss.2022.07.002.

[42] J. E. Hopcroft, R. Motwani, and J. D. Ullman, “Introduction to automata theory, languages, and computation, 2nd edition,” *ACM SIGACT News*, vol. 32, no. 1, pp. 60–65, Mar. 2001, doi: 10.1145/568438.568455.

[43] H. Djidjev, “Automaton-based methodology for implementing optimization constraints for quantum annealing,” in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, New York, NY, USA: ACM, May 2020, pp. 118–125. doi: 10.1145/3387902.3392619.

[44] T. Andriani and Pristiwanto, “Transducer Function In Vending Machine Simulation Design,” *Instal : Jurnal Komputer*, vol. 11, no. 01, pp. 26–34, Feb. 2019, doi: 10.54209/jurnalkomputer.v11i01.5.

[45] I. Chraibi Kaadoud, L. Fahed, T. Tian, Y. Haralambous, and P. Lenca, “Automata-based Explainable Representation for a Complex System of Multivariate Times Series,” in *Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, SCITEPRESS - Science and Technology Publications, 2022, pp. 170–179. doi: 10.5220/0011363400003335.

[46] M. Erkurt, “Dynamics and Complexity of Computrons,” *Entropy*, vol. 22, no. 2, p. 150, Jan. 2020, doi: 10.3390/e22020150.

[47] A. Mallik and A. Khetarpal, “Turing Machine based Syllable Splitter,” in *2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, IEEE, Jul. 2021, pp. 87–90. doi: 10.1109/CCICT53244.2021.00028.

[48] H. Haryanto and Aripin, “A Finite State Machine Model to Determine Syllables of Indonesian Text,” in *2019 1st International Conference on Cybernetics and Intelligent System (ICORIS)*, IEEE, Aug. 2019, pp. 238–241. doi: 10.1109/ICORIS.2019.8874889.