

Finite State Machines for NLP

Ye Kyaw Thu^{1,2,3}

¹National Electronics and Computer Technology Center (NECTEC), Thailand

²Language Understanding Lab., Myanmar

³Language and Speech Science Research Lab., Waseda University, Japan

NLP Class, UTYCC, Pyin Oo Lwin, Myanmar

email: ka2pluskha2@gmail.com

December 6, 2019

Lecture Outline

- 1 Motivation
- 2 Finite State Automata (FSA)
- 3 Finite State Transducer (FST)
- 4 FST Examples
- 5 Important Operations
- 6 Limitation of FSA, FST
- 7 What Can We Do with FST

- ကျွန်တော် သီအိုရီအနေနဲ့ စိတ်ဝင်စားတယ်
- နောက်တချက်က ကျောင်းသားအများစုက **FSA** သီအိုရီကို သင်ဖူးကြပေမဲ့ လက်တွေ့မသုံးတတ်ကြဘူး
- မြန်မာစာ **NLP R&D** အတွက် finite state machine တွေကိုလည်း သုံးကြရအောင်

Finite State Automata (FSA)

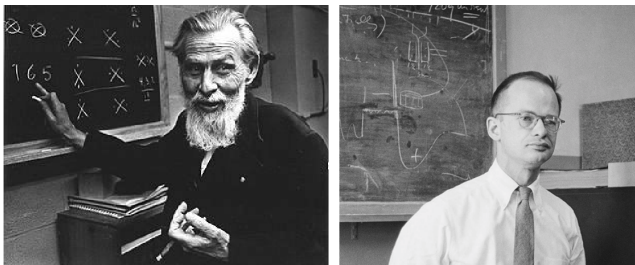


Figure: Left: Warren S. McCullough, Right: Walter Pitts

- ၁၉၄၃ မှာ neuro-psychologists တွေဖြစ်ကြတဲ့ Warren S. McCullough Walter Pitts က ပထမဦးဆုံး Finite State Automata သီအိုရီကို a model for human brain ဆိုပြီးတော့ မိတ်ဆက်ခဲ့တယ်
- Finite automata နဲ့ microprocessor တွေကိုလည်း မော်ဒယ်ဆောက်လို့ရတယ်
- Regular set of sequences တွေဖြစ်တဲ့ logic, algebra, regular expression တွေနဲ့ ဆက်စပ်တယ်

Finite State Automata (FSA)

A finite state automaton is a quintuple $(S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite input alphabet;
- $\delta : S \times \Sigma \rightarrow S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

Finite State Automata (FSA)

FSA ကို စာနဲ့ အလွယ်ရှင်းပြရရင်

- သူက **nodes** တွေ၊ မြားတွေ နဲ့ ဆွဲထားတဲ့ **graph** ပုံပါပဲ။
- ဝင်လာတဲ့ **input** ကို လက်ခံတာ၊ ပယ်တာကို လုပ်ပါတယ်။
- **input** အားလုံးကို လက်ခံနိုင်တယ်၊ ဖတ်လို့ ပြီးတယ်ဆိုရင်၊ **initial node** ကနေ **final node** အထိ အရောက်သွားနိုင်တယ်၊ တနည်းအားဖြင့်ပြောပြရရင် **automaton** ရဲ့ **final state** ကိုရောက်သွားတယ်။
- **node** တစ်ခုကနေ နောက်ထပ် **node** တစ်ခု၊ **state** တစ်ခုကနေ နောက်ထပ် **state** တစ်ခုစီကို **input value** ကိုကြည့်ပြီး သွားမယ့်လမ်းကြောင်းကို ရွေးသွားတဲ့ ပုံစံပါ။
- တချို့ **node** တွေက ϵ (**epsilon**) သို့မဟုတ် **empty string** ကို **pass** လုပ်ပေးပါလိမ့်မယ်။
- **final state** ကိုတော့ **double wall** (**double circle**) နဲ့ ကိုယ်စားပြုပုံဆွဲတယ်။

Finite State Automata (FSA)

- ဒီ lecture မှာ OpenFST ကိုသုံးပါမယ်။
- OpenFST က Google နဲ့ Courant Institute of Mathematical Sciences, New York University တို့က ပူးပေါင်းပြီးတော့ develop လုပ်ထားတဲ့ Open source tool ဖြစ်ပါတယ်။
- Finite state automata ကို အခြေခံပြီးတော့ ဖြစ်လာတဲ့ Finite state transducers တွေကို လွယ်လွယ်ကူကူ မော်ဒယ်ဆောက်ပြီးတော့ operation တွေကို run နိုင်ဖို့ build လုပ်ထားတာပါ
- Link: <http://www.openfst.org>

Finite State Automata (FSA)

- ဒီနေရာမှာ အသေးစိတ် မရှင်းနိုင်ပေမဲ့ FSA, FST တွေကို ထဲထဲဝင်ဝင် နားလည်ဖို့က ကျောင်းသားတွေအနေနဲ့က regular expression (RE) ကို သိထားသင့်ပါတယ်။
- ဥပမာ $k^*a+n?$ ဆိုတဲ့ RE ကို ပြန်စဉ်းစားကြည့်ရအောင်
- A regular expression followed by an asterisk (*) matches zero or more occurrences of the regular expression
- A regular expression followed by a plus sign (+) matches one or more occurrences of the one-character regular expression ရွေးစရာရှိရင် ပထမဆုံး matched ဖြစ်တဲ့ string ကိုပဲ ယူလိမ့်မယ်
- A regular expression followed by a question mark (?) matches zero or one occurrence of the one-character regular expression

Finite State Automata (FSA)

Expression	Expression
<code>/က*စ+ဂ?/g</code>	<code>/က*စ+ဂ?/g</code>
Text	Text
ကစဂ စဂ ဂ စက ကကက ကစစစ စဂ ကမစက	စစစဂ - ကစဂ စက - စဂ
Expression	Expression
<code>/က*စ+ဂ?/</code>	<code>/က*စ+ဂ?/</code>
Text	Text
စစစဂ - ကစဂ စက - စဂ	

Figure: testing RE m^*a+n ?

- online RE tool တစ်ခုဖြစ်တဲ့ <https://regexr.com/> ကို သုံးပြီး စမ်းကြည့်နိုင်တယ်
- RE ရေးနေကြအတိုင်း ကိုယ် စမ်းချင်တဲ့ RE ကို / (forward slash) နှစ်ခုရဲ့ကြားထဲမှာ ရေးတယ်
- g က global ကို ဆိုလိုတယ်
- g REF flag ထည့်ထားမှ စာကြောင်း တစ်ကြောင်းလုံးမှာ match ဖြစ်သမျှ RE pattern တွေကို ဆွဲထုတ်ပေးနိုင်တယ်

Finite State Automata (FSA)

symbol file ဆောက်ဖို့ လိုအပ်တယ်

Finite state acceptor မောဒယ်အတွက်ဆိုရင်တော့ --isymbols option အတွက်ပဲ
လိုအပ်လိမ့်မယ်။

filename: my.syms

ε<TAB>0

က<TAB>1

ခ<TAB>2

ဂ<TAB>3

ဃ<TAB>4

င<TAB>5

Finite State Automata (FSA)

regex file လည်း ပြင်ဆင်ဖို့ လိုအပ်တယ်
filename က ကြိုက်သလိုပေးလိုရပါတယ်။

filename: regex.fsa.txt

0<TAB>1<TAB>ε

0<TAB>2<TAB>ε

2<TAB>2<TAB>က

2<TAB>1<TAB>ε

1<TAB>3<TAB>ခ

3<TAB>3<TAB>ခ

3<TAB>4<TAB>ε

3<TAB>4<TAB>ဂ

4

Finite State Automata (FSA)

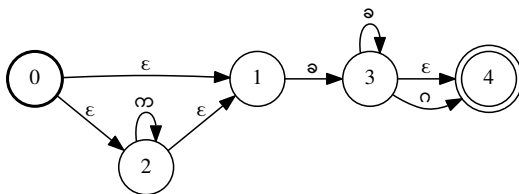


Figure: Finite state automata of $m^*a^+o?$

- `fstcompile --acceptor --isymbols=my.syms regex.fsa.txt > regex.fsa`
- `fstdraw --portrait --acceptor --isymbols=my.syms regex.fsa | dot -Tpdf > regex.pdf`

Finite State Automata (FSA)

testing လုပ်ကြည့်ဖို့အတွက် input ဖိုင်ကို ပြင်ဆင်မယ်

filename: input.fsa.txt

0<TAB>1<TAB>က

1<TAB>2<TAB>က

2<TAB>3<TAB>ခ

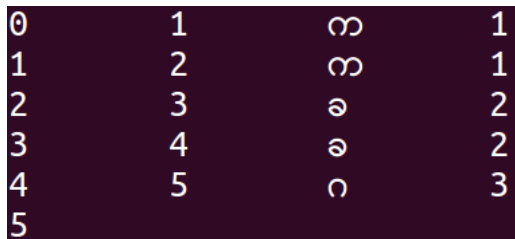
3<TAB>4<TAB>ခ

4<TAB>5<TAB>ဂ

5

Finite State Automata (FSA)

- `fstcompile --acceptor --isymbols=my.syms ./input.fsa.txt > input.fsa`
- `fstprint --isymbols=my.syms ./input.fsa`



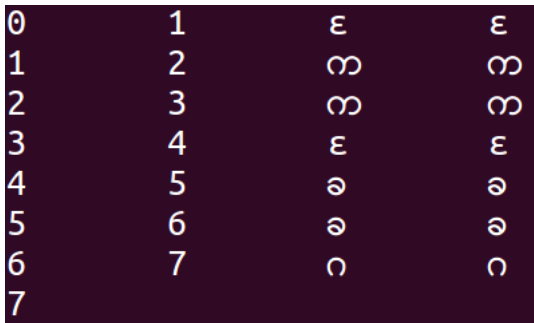
0	1	∞	1
1	2	∞	1
2	3	∅	2
3	4	∅	2
4	5	∅	3
5			

Figure: fstprint command output screen

Finite State Automata (FSA)

fsa မော်ဒယ် နှစ်ခုကို compose လုပ်ပြီးတော့ output ကို print ထုတ်ကြည့်ရအောင်

- `fstcompose ./input.fsa ./regex.fsa | fstprint --isymbols=my.syms --osymbols=my.syms`



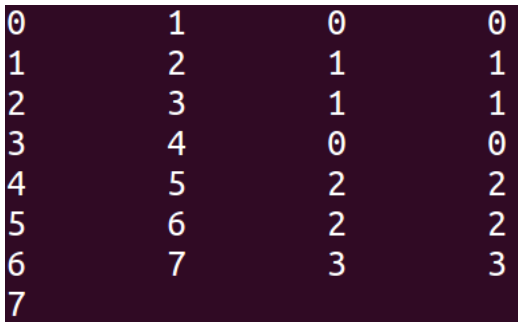
0	1	ε	ε
1	2	က	က
2	3	က	က
3	4	ε	ε
4	5	ခ	ခ
5	6	ခ	ခ
6	7	ဂ	ဂ
7			

Figure: fstprint command output screen

Finite State Automata (FSA)

symbols file ကို မပေးရင် ဂဏန်းနဲ့ပဲ ရိုက်ထုတ်ပြလိမ့်မယ်။

- `fstcompose ./input.fsa ./regex.fsa | fstprint`



0	1	0	0
1	2	1	1
2	3	1	1
3	4	0	0
4	5	2	2
5	6	2	2
6	7	3	3
7			

Figure: fstprint command output screen

Let's do above steps on your computer
1st install OpenFST

Finite State Transducer (FST)

A finite state automaton is a sextuple $(\Sigma, \Gamma, S, s_0, \delta, \omega)$, where:

- S is a finite, non-empty set of states
- Σ is the input alphabet (a finite non-empty set of symbols)
- Γ is the output alphabet (a finite, non-empty set of symbols)
- s_0 is the initial state, an element of S . In a nondeterministic finite automaton, s_0 is a set of initial states.
- δ is the state-transition function: $\delta : S \times \Sigma \rightarrow S$.
- ω is the output function.

Finite State Transducer (FST)

- FSA မှာက input (input tape လို့လည်း ခေါ်ကြ) ပဲ ရှိပါတယ်။
- FST မှာကျတော့ input, output နှစ်မျိုး ပါဝင်ပါတယ်။
- input မှာ ϵ (epsilon) သို့မဟုတ် empty string ကို လက်ခံနိုင်သလို။
- output မှာလည်း ϵ (epsilon) သို့မဟုတ် empty string ကို ထုတ်ပေးတာ မျိုး ဖြစ်နိုင်ပါတယ်။
- မြား တွေရဲ့အပေါ်မှာတော့ input:output/weight ဆိုတဲ့ ပုံစံနဲ့ ဖော်ပြကြပါတယ်။

Finite State Transducer (FST)

Let's build a random language generator with FST

The following is the Subject-Object-Verb grammar file (sentence.txt):

0	1	subj
1	2	obj
2	3	v
3		

Figure: Subject-Object-Verb grammar text file

Language generator FST ဆောက်တာကို လေ့ကျင့်ခန်း နံပါတ် ၁၂ အနေနဲ့
လုပ်ခိုင်းခဲ့ပါတယ်

<https://github.com/ye-kyaw-thu/NLP-Class/tree/master/exercise/exe-12/fsa-exe1>

Finite State Transducer (FST)

We need to prepare input and output symbol files
The following is the pos.txt (POS tag) file:

```
-<TAB>0  
subj<TAB>1  
obj<TAB>2  
v<TAB>3
```

Finite State Transducer (FST)

The following is the word.txt (i.e. output) file:

-<TAB>0

ကျွန်တော်<TAB>1

ကျွန်မ<TAB>2

ကျောင်း<TAB>3

ဈေး<TAB>4

သွားတယ်<TAB>5

ပြန်တယ်<TAB>6

Finite State Transducer (FST)

An example of random language generator with FST

We have a very simple grammar in our language (Subject-Object-Verb)

- `fstcompile --acceptor --isymbols=pos.txt sentence.txt > sentence.fsa;`
- `fstdraw --portrait --isymbols=pos.txt ./sentence.fsa | dot -Tpdf -Gsize=6,3 -Eheadport=e -Etailport=w > sentence.pdf;`

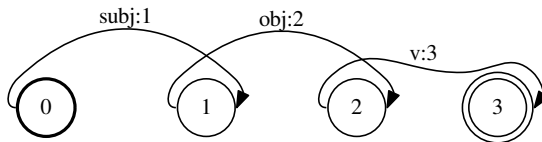


Figure: FST graph for Subject-Object-Verb grammar

Finite State Transducer (FST)

Let's say, we have a small dictionary (dict.txt)
that contained only six words

0<TAB>0<TAB>subj<TAB>ကျွန်တော်

0<TAB>0<TAB>subj<TAB>ကျွန်မ

0<TAB>0<TAB>obj<TAB>ကျောင်း

0<TAB>0<TAB>obj<TAB>ဈေး

0<TAB>0<TAB>v<TAB>သွားတယ်

0<TAB>0<TAB>v<TAB>ပြန်တယ်

0

Finite State Transducer (FST)

Let's compile the dictionary FST and make PDF file for checking

- `fstcompile --isymbols=pos.txt --osymbols=word.txt dict.txt > dict.fst`
- `fstdraw --portrait --isymbols=pos.txt --osymbols=word.txt dict.fst | dot -Tpdf -Gsize=6,3 -Eheadport=e -Etailport=w > dict.pdf`

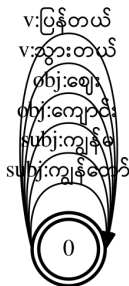


Figure: FST graph for our six words dictionary

Finite State Transducer (FST)

By composing the grammar (FSA) and the dictionary (FST), we got a small language generator FST (strings.fst)

- `fstcompose sentence.fsa dict.fst > strings.fst;`
- `fstdraw --portrait --isymbols=pos.txt --osymbols=word.txt ./strings.fst | dot -Tpdf -Gsize=6,3 -Eheadport=e -Etailport=w > strings.pdf;`

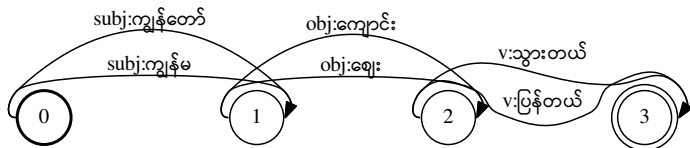


Figure: FST graph for a language generator

Finite State Transducer (FST)

Now we can generate Myanmar sentences randomly based on the Subject-Object-Verb grammar

```
1 #!/bin/bash
2
3 fstrandgen strings.fst | fstproject --project_output |
4 fstprint --acceptor --isymbols=word.txt |
5 awk 'BEGIN{printf("\n")}{printf("%s ", $3)}END{printf("\n")}'
```

Figure: FST graph for a language generator

Some example outputs are as follows:

ကျွန်မ ကျောင်း သွားတယ်
ကျွန်တော် ကျောင်း သွားတယ်
ကျွန်မ ဈေး သွားတယ်
ကျွန်တော် ဈေး ပြန်တယ်
ကျွန်မ ဈေး ပြန်တယ်

Finite State Transducer (FST)

Let's do string similarity measurement with FST

For example, we want to measure following Myanmar words:

စား (“eat” in English)

စ (“letter” in English)

ကား (“car” in English)

Finite State Transducer (FST)

- The Levenshtein distance between two strings a, b (of length $|a|$ and $|b|$ respectively) is given by $lev_{a,b}(|a|, |b|)$ where

Theorem (Levenshtein distance)

$$lev_{a,b}(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1, j) + 1 \\ lev_{a,b}(i, j-1) + 1 \\ lev_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

- where $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and equal to 1 otherwise, and $lev_{a,b}(i, j)$ is the distance between the first i characters of a and the first j characters of b . i and j are 1-based indices.
- Source: Levenshtein Distance Wiki

Finite State Transducer (FST)

Edit distance operation FST မဆောက်ခင်မှာ input ဖိုင် ကို perl ရိုက်ထုတ်ဖို့အတွက် “myanmar.syms” ဖိုင် ကို ပြင်ဆင်ကြရအောင်:

၀

၀

း

က

Finite State Transducer (FST)

Perl script for printing edit distance operations as OpenFST format:

```
17 my $delWeight = "1.00";
18 my $insertWeight = "1.00";
19 my $substituteWeight = "1.00";
20 my $noEditWeight = "0.00";
21 my $epsilon = "ε";
22 my @myChar;
23
24 # this line required for reading UTF8 Myanmar characters from STDIN
25 eof() ? exit : binmode ARGV, ':utf8';
26
27 while (my $line = <>) {
28     chomp($line); # remove \n
29     push (@myChar, $line);
30 }
31
```

Figure: some codes from mk-ed-operation.pl

Finite State Transducer (FST)

no edit, delete, insert, substitution operations:

```
32 my $oneChar;
33
34 foreach $oneChar (@myChar)
35 {
36
37   # No edit operation
38   print ("0\t0\t$oneChar\t$oneChar\t$noEditWeight\n");
39
40   # Delete operation
41   print ("0\t0\t$oneChar\t$epsilon\t$delWeight\n");
42
43   # Insert operation
44   print ("0\t0\t$epsilon\t$oneChar\t$insertWeight\n");
45
46   # Substitute operation
47   my $subChar;
48   foreach $subChar (@myChar)
49   {
50     # Substitute operation
51     if ($subChar ne $oneChar)
52     {
53       print ("0\t0\t$oneChar\t$subChar\t$substituteWeight\n");
54     }
55   }
56 }
57 print ("0\n");
```

Figure: some codes from mk-ed-operation.pl

Finite State Transducer (FST)

- Example output with “myanmar.syms”

၀	၀	၀	၀	၀.၀၀
၀	၀	၀	ε	၁.၀၀
၀	၀	ε	၀	၁.၀၀
၀	၀	၀	၀၀	၁.၀၀
၀	၀	၀	၀:	၁.၀၀
၀	၀	၀	၀၀	၁.၀၀
၀	၀	၀၀	၀၀	၀.၀၀
၀	၀	၀၀	ε	၁.၀၀
၀	၀	ε	၀၀	၁.၀၀
၀	၀	၀၀	၀	၁.၀၀
၀	၀	၀၀	၀:	၁.၀၀
၀	၀	၀၀	၀၀	၁.၀၀

Figure: some outputs when you run “mk-ed-operation.pl” perl script

Finite State Transducer (FST) for Data Cleaning

Myanmar words are formed by syllables and we can clean impossible syllable formation errors by defining the all possible combination patterns (we need a big monolingual corpus)

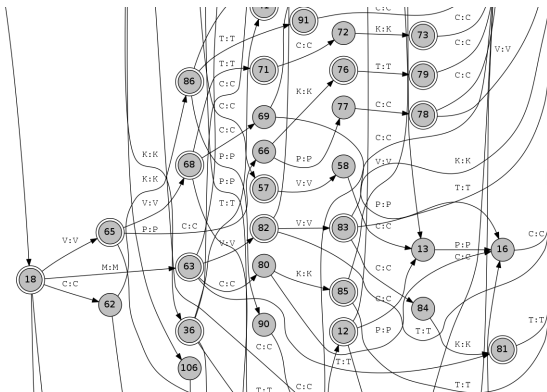


Figure: A part of Myanmar syllable cleaning FST

Important Operations

Let's recall our first regex Finite State Acceptor
Yes, it contained four ϵ (epsilon) symbols

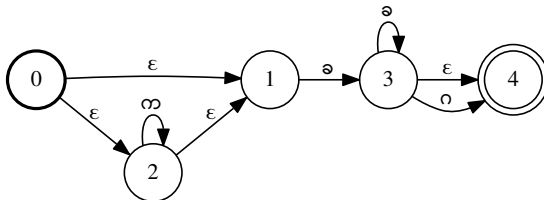


Figure: FSA graph of regex $m^*a+n?$

Important Operations

Removing epsilon operation (fstrmepsilon)

- #Remove epsilon
- `fstrmepsilon ./regex.fst > regex.rmepsilon.fst`
- `fstdraw --portrait --acceptor --isymbols=my.syms
regex.rmepsilon.fst | dot -Tpdf > regex.rmepsilon.pdf`
- `evince ./regex.rmepsilon.pdf`

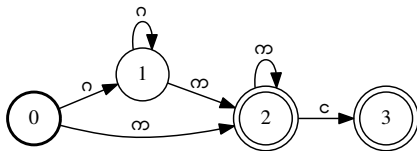


Figure: regex $a^*ba^?$ graph after applying “fstrmepsilon” operation

FST Examples

bla bla

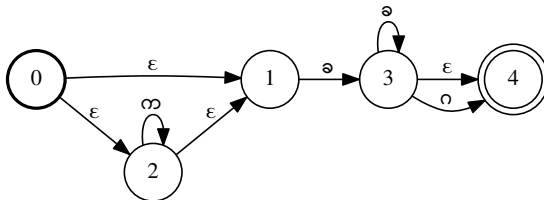


Figure: FSA graph of regex $m^*a+n?$

to be continue ... :)

Thank you!