

# Weighted Finite State Transducers (WFST)

(Invited Lecture@University of Taxila)

Ye Kyaw Thu  
Lab Leader, LU Lab., Myanmar

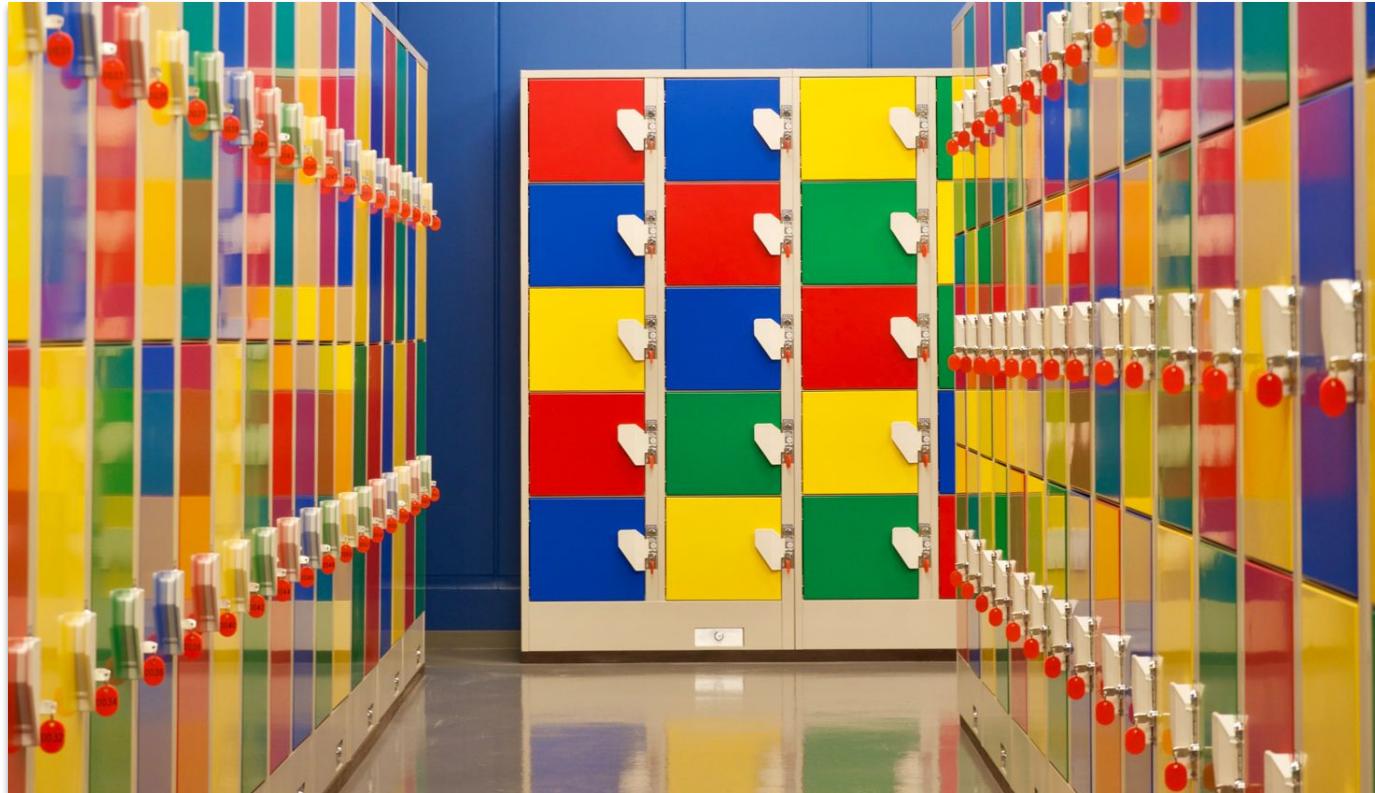
30 July 2025 (SUN)

# Contents

1. Who am I?
2. State Machines
3. Formalization
4. FSA Construction
5. FST Construction
6. Openfst Operations
7. WFST Spoken Dialect Translation
8. WFST POS Tagging (demo)
9. Conclusion

# Who am I?

# State Machines (Coin Locker)



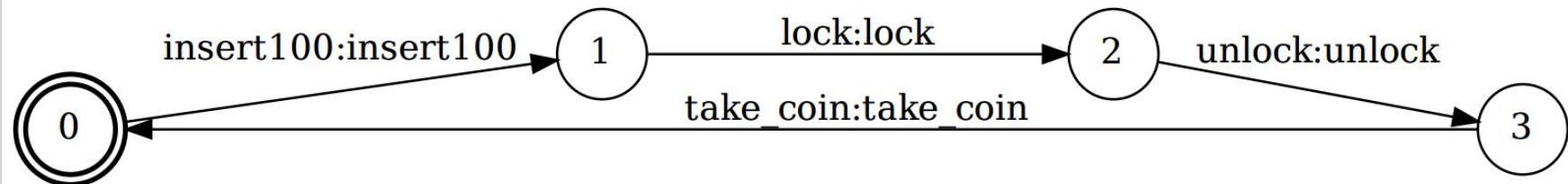
<https://commons.wikimedia.org/>

# State Machines (Coin Locker)



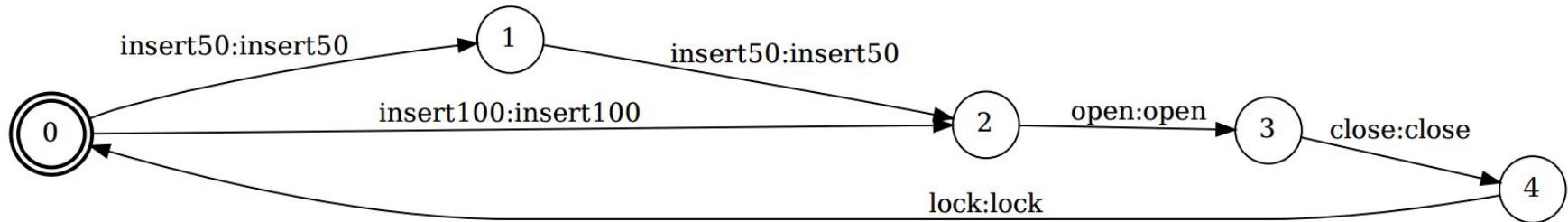
- ပစ္စည်းအပ်တဲ့အခါ
  - တခါးကိုဆွဲဖွင့်ပြီး အပ်မယ့်ပစ္စည်းကုတည်ပါ
  - အကြောင်း ယန်း၁၀၀ ကိုထည့်ပါ
  - တံခါးကို ပိတ် သော့ခတ်ပြီး၊ သော့ကို နှုတ်ယူပါ
- အပ်ထားတဲ့ ပစ္စည်းကို ပြန်ယတဲ့အခါ
  - သော့ဖွင့်ပြီး၊ တံခါးကုဖွင့်ပါ
  - ယန်း၁၀၀ အကြောင်းကို ပြန်ယူပါ
  - အပ်ထားတဲ့ ပစ္စည်းကို ပြန်ယူပြီး၊ တံခါးကို ပြန်ပိတ်ထားခဲ့ပါ

# State Machines (Coin Locker)



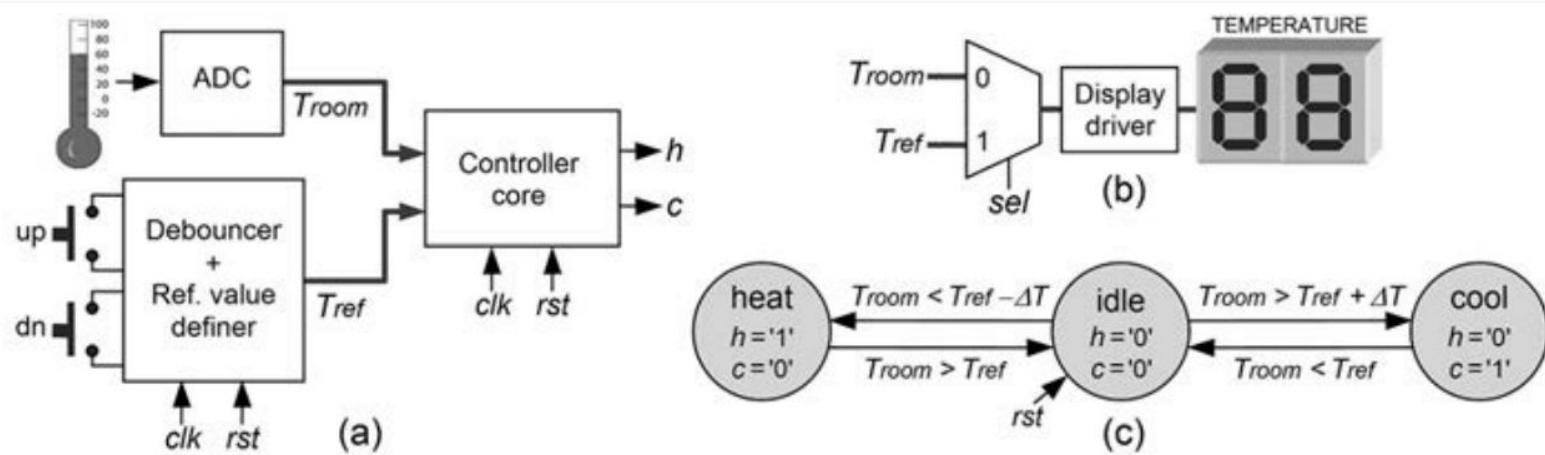
- ဂျပန်နိုင်ကဗ္ဗော်စွေးခေါ်ထဲတဲ့ သုံးရတဲ့ coin locker ရဲ့ အဓိကတဲ့ အလုပ်လုပ်ပုံ အဆင့်ဆင်ကိုပဲ finite state machine အနေနဲ့ စဉ်းစားမယ် ဆိုရင် အတက်ပါအတိုင်း graph ပုံရလိမ့်မယ်
- Programmer ကောင်းတစ်ယောက် ဖြစ်ချင်ရင် process flow ကို finite state machine အနေနဲ့ ဆွဲတတ်ရတယ် (သုံး) flowchart ပေါ့

# State Machines (Coin Locker)



- တကယ်လို့ ပိုက်ဆံပေးပြီးမှ သုံးရမယ် ဆိုရင် တံခါးက ပိုတ်နေတဲ့ state က စမယ်
- အကြောင်း ၁၀၀ ပြည့်အောင်ထည့်မှ locker တံခါးက ပွုင့်မယ်
- အကြောင်း ၅၀ စွဲကုံလည်း လက်ခံတယ် (၁၀၀ ပြည့်အောင် နှစ်ခါတည့်ရမယ်)
- ကိုယ့်ရဲ့ အိတ်ဘာညာ ကိုထည့်ပြီးမှ lock လုပ်တဲ့ အထိ simulation လုပ်ကြည့်ရင် အထက်ပါ လုံမျိုး state machine ဖြစ်လိမ့်မယ်

# State Machines (Temperature Controller)



**Figure 5.8**

Temperature controller. (a) Overall circuit diagram. (b) Display driver. (c) State machine for the controller core block.

# State Machines (7 Segment Display)

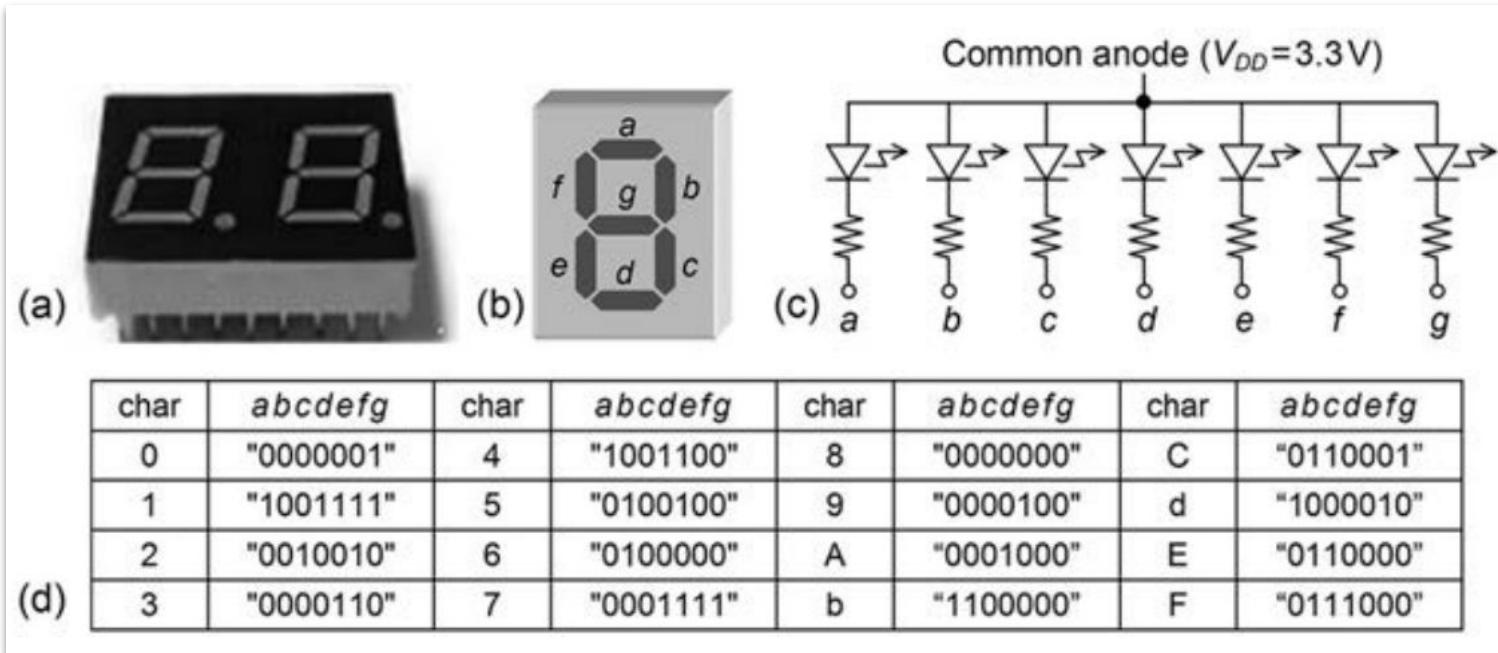


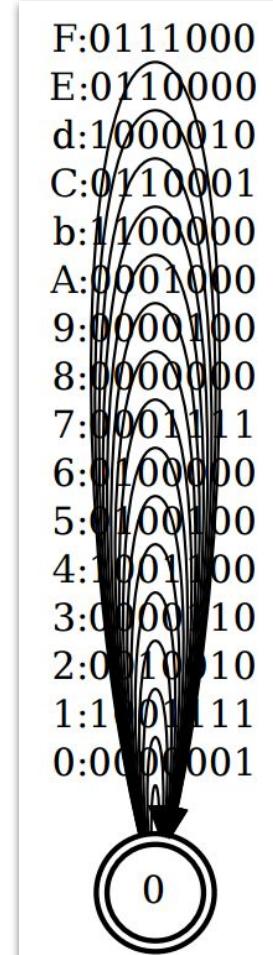
Fig. A 7-segment display works by controlling individual segments (a-g)

Reference: Finite State Machines in Hardware (Volnei A. Pedroni, the MIT Press, 2013)

# State Machines (7 Segment Display)

- 7 segment display (i.e. a, b, c, d, e, f, g)
- ဘယ် segment တွက် ON (1) or OFF (0) လုပ်မလဲ ဆိုတာကို သတ်မှတ်ရတယ်
- Truth table (သို့) lookup table ဆောက်ရတယ်
- ဒါကိုလည်း finite state machine အနေနဲ့ မြင်လို့ ရတယ်
- Mapping ကိုသိမ်းထားတဲ့ dictionary အနေနဲ့လည်း စဉ်းစားလို့ ရတယ်

Fig. Mapping between 7 segments and characters



# Formalization (Semiring)

- A **semiring**  $(K, \oplus, \otimes, 0, 1)$  is a ring that may lack negation.
  - Sum ( $\oplus$ ): Computes the weight of a sequence (sum of the weights of the paths labeled with that sequence)
  - Product ( $\otimes$ ): Computes the weight of a path (product of the weights of constituent transitions)
- Where:
  - $\oplus_{\log}$  is defined by:  $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$
  - $\wedge$  is the longest common prefix operator
  - The string semiring is a left semiring

# Formalization (Semiring)

- $\wedge$  is the Longest Common Prefix (LCP) Operator
- String Semiring Context:
  - $\oplus = \wedge$  (LCP): Given two strings, output their longest shared prefix.
  - Example: "hello"  $\wedge$  "heap" = "he".
  - $\otimes = .$  (concatenation): "he"  $\otimes$  "llo" = "hello"
- Why LCP?
  - In text processing, LCP helps merge paths with shared prefixes (e.g., in morphology FSTs for word stems).

# Formalization (Semiring)

Table 1: Semiring Examples

Semiring	Set	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+$	+	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	+	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	+	$+\infty$	0
String	$\Sigma^* \cup \{\infty\}$	$\wedge$	.	$\infty$	$\epsilon$

## Formalization (Semiring)

**Probability Semiring**  $(\mathbb{R}_+, +, \times, 0, 1)$ :

$$[T](ab, r) = (1 \times 2 \times 2) + (3 \times 2 \times 2) = 16$$

**Tropical Semiring**  $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ :

$$[T](ab, r) = \min(1 + 2 + 2, 3 + 2 + 2) = 5$$

# Formalization (Semiring)

- Why Different Semirings?
  - Tropical: Optimize for cost (e.g., speech recognition lattices).
  - Probability: Model stochastic processes (e.g., language models).
  - String: Build transducers for text (e.g., "c a t" → "cat").
- "What happens if we use the tropical semiring in a spell-checker FST?"
  - Finds the lowest-cost (most probable) correction.
- "Why can't we use the boolean semiring for weighted tasks?"
  - It discards weight information (only accept/reject).

# Formalization (Finite State Automaton (FSA))

A **deterministic** FSA is a 5-tuple:

$$A = (\Sigma, Q, q_0, F, \delta)$$

where:

- $\Sigma$ : Finite input alphabet
- $Q$ : Finite set of states
- $q_0 \in Q$ : Initial state
- $F \subseteq Q$ : Set of final states
- $\delta : Q \times \Sigma \rightarrow Q$ : Transition function

**Acceptance:** A string  $w = w_1 \dots w_n$  is accepted iff:

$$\delta^*(q_0, w) \in F \quad \text{where} \quad \delta^*(q, \epsilon) = q \quad \text{and} \quad \delta^*(q, aw') = \delta^*(\delta(q, a), w')$$

# Formalization (Finite State Transducer (FST))

An FST is a 7-tuple:

$$T = (\Sigma, \Delta, Q, q_0, F, \delta, \sigma)$$

where:

- $\Sigma$ : Input alphabet,  $\Delta$ : Output alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ : State transition function
- $\sigma : Q \times \Sigma \rightarrow \Delta^*$ : Output function

**Relation:** For  $w = w_1 \dots w_n$ , output  $y = y_1 \dots y_n$  where:

$$y_i = \sigma(q_{i-1}, w_i) \quad \text{and} \quad q_i = \delta(q_{i-1}, w_i)$$

# Formalization (Weighted Finite State Transducer (WFST))

A WFST over semiring  $\mathbb{K} = (K, \oplus, \otimes, \bar{0}, \bar{1})$  is a 8-tuple:

$$T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$$

where:

- $I \subseteq Q$ : Initial states with weights  $\lambda : I \rightarrow K$
- $F \subseteq Q$ : Final states with weights  $\rho : F \rightarrow K$
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times K \times Q$ : Weighted transitions

# Formalization (Weighted Finite State Transducer (WFST))

**Path Weight:** For path  $\pi = e_1 \dots e_n$ :

$$w[\pi] = \bigotimes_{i=1}^n w[e_i]$$

**String Pair Weight:** For  $(x, y) \in \Sigma^* \times \Delta^*$ :

$$[T](x, y) = \bigoplus_{\pi \in P(I, x, y, F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

# Formalization (FSA, FST, WFST)

Table 2: Comparison of Automata Types

Type	Output	Weights	Semiring
FSA	Accept/Reject	None	Boolean
FST	Strings	None	N/A
WFST	Strings	$K$	Any (Tropical, Probability, ...)

- **FSA:** Regular expression matching.
- **FST:** "cat" \$\to\$ "cats" (pluralization rules).
- **WFST:** Speech recognition lattices.

# Formalization (FSA, FST, WFST)

- **Non-deterministic FST**
  - **Input:** "read"
  - Path 1: "read" → "reed" (verb)
  - Path 2: "read" → "red" (past tense)
  - Same input, multiple outputs.
- **Non-deterministic WFST (Tropical Semiring)**
  - **Input:** "cat"
  - Path 1: "cat" → "cats" (cost = 1.2)
  - Path 2: "cat" → "cat" (cost = 0.5)
  - Best path:  $\min(1.2, 0.5) = 0.5 \rightarrow \text{Output } \text{"cat"}$ .

# Formalization (FSA, FST, WFST)

- **FSTs/WFSTs** are typically non-deterministic in practice (ambiguity in input/output mappings).
- **Deterministic** variants exist but are restrictive (no  $\epsilon$ -transitions, unique paths).
- **Non-determinism** enables compact modeling of ambiguity but increases computational cost.

# FSA Construction (Myanmar Rooster)



- ကြက်ဖတ္တန်သံကို FSA ဆောက်ပြီး  
testing လုပ်ကြည့်ကြရအောင်။
- ဝိက္ခမိဒီယာကနဲ့ ယူလာတဲ့ပုံ။  
မန္တလေးက ကြက်ဖတဲ့...

# FSA Construction (Myanmar Rooster)

- အောက်ပါအတိုင်း ဖိုင်လေးဖိုင် ပြင်ဆင်ခဲ့တယ်။

```
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ tree .
.
└── compile_and_test.sh
└── correct_input.txt
└── rooster.txt
└── syllable_syms
└── wrong_input.txt

1 directory, 5 files
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$
```

# FSA Construction (Myanmar Rooster)

- **rooster.txt** ဖိုင်ထဲမှာ အောက်ပါအတိုင်း FSA ကို utf-8, plain text ဖိုင်အနေနဲ့ ဆောက်ထားတယ်။

0 1 အောက်

1 2 အီး

2 3 အီး

3 4 အွဲတ်

4

# FSA Construction (Myanmar Rooster)

- Input symbol ဖိုင် လိုအပ်တယ်
- syllable.syms ဖိုင်ထဲမှာက အောက်ပါအတိုင်း
- TAB သို့မဟုတ် Space နဲ့ခြားပါ
- ထိပ်ဆုံးလိုင်းမှာ Epsilon ကုသတ်မှတ်ပါ (<eps>, ε သုံးလည်းရတယ်)

<epsilon> 0

အောက် 1

အံံး 2

အွဲတ် 3

# FSA Construction (Myanmar Rooster)

- **correct\_input.txt** ဖိုင်ထဲမှာက အောက်ပါအတိုင်း ကြက်တွန်သံကို  
မှန်မှန်ကန်ကန် ရှိက်ပေးထားတယ်။

အောက်

အီး

အီး

အုတ်

# FSA Construction (Myanmar Rooster)

- **wrong\_input.txt** ဖိုင်ထဲမှာက အောက်ပါအတိုင်း ကြက်တွန်သံကို  
တမ်းရှိခိုက်ထားတယ်။

အောက်

အွေတ်

အောက်

အွေတ်

# FSA Construction (Myanmar Rooster)

- `compile_and_test.sh` ကိုတော့ ဆာဗာထဲင်ပြီး run ပြပါမယ်။

```
#!/bin/bash

# Clean up
rm -f *.bin *.pdf *.dot *.verdict.txt *.shortest.bin

# 1. Compile rooster FSA
fstcompile --isymbols=syllable_syms --acceptor rooster.txt rooster.bin
echo "==== Rooster FSA ==="
fstprint --isymbols=syllable_syms rooster.bin

process_input() {
    local input_file=$1
    local prefix=$2
    echo -e "\n==== Processing $input_file ==="

    # Create input FSA with PROPER STATE PROGRESSION
    echo "0" > ${prefix}.txt
    state=0
    while read -r syllable; do
        next_state=$((state+1))
        echo "$state $next_state $syllable" >> ${prefix}.txt
        state=$next_state
    done < $input_file
    echo "$state" >> ${prefix}.txt # Mark final state
    echo "Input FSA:"
    cat ${prefix}.txt
```

# FSA Construction (Myanmar Rooster)

```
# Compile input FSA
fstcompile --isymbols=syllable_syms --acceptor ${prefix}.txt ${prefix}.bin
echo "Compiled input FSA:"
fstprint --isymbols=syllable_syms ${prefix}.bin

# Compose
fstcompose ${prefix}.bin rooster.bin ${prefix}.result.bin
echo "Composition result:"
fstprint --isymbols=syllable_syms ${prefix}.result.bin

# Check acceptance
if fstprint --isymbols=syllable_syms ${prefix}.result.bin | grep -q "$state"; then
    echo "ACCEPTED" > ${prefix}.verdict.txt
    echo "Result: ACCEPTED"
else
    echo "REJECTED" > ${prefix}.verdict.txt
    echo "Result: REJECTED"
fi
}

# Process
process_input correct_input.txt correct
process_input wrong_input.txt wrong
```

# FSA Construction (Myanmar Rooster)

```
1 (openfst) · ye@lst-hpc3090:~/talk/wfst/r
2 === Rooster FSA ===
3 0 ..... 1 ..... အောက် 1
4 1 ..... 2 ..... အီး 2
5 2 ..... 3 ..... အီး 2
6 3 ..... 4 ..... အုတ် 3
7 4
8
9 === Processing correct_input.txt ===
10 Input FSA:
11 0
12 0 · 1 · အောက်
13 1 · 2 · အီး
14 2 · 3 · အီး
15 3 · 4 · အုတ်
16 4
```

- Run လိုက်ရင်  
မြင်ရေ့မယ့် output  
ထွေပါ

# FSA Construction (Myanmar Rooster)

```
17 Compiled input FSA:  
18 0 ..... 1 ..... အောက် 1  
19 0  
20 1 ..... 2 ..... အီး 2  
21 2 ..... 3 ..... အီး 2  
22 3 ..... 4 ..... အူတ် 3  
23 4  
24 Composition result:  
25 0 ..... 1 ..... အောက် 1  
26 1 ..... 2 ..... အီး 2  
27 2 ..... 3 ..... အီး 2  
28 3 ..... 4 ..... အူတ် 3  
29 4  
30 Result: ACCEPTED
```

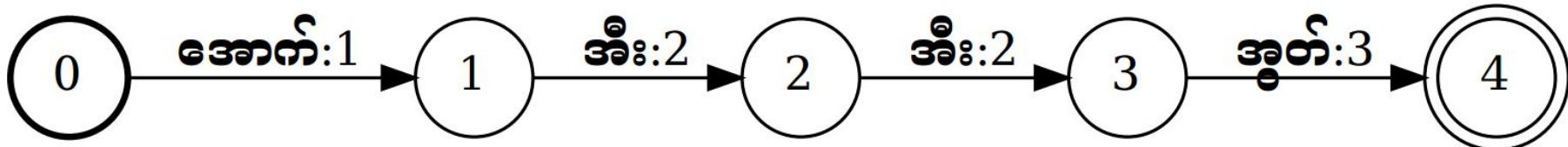
- Correct\_input.txt  
အတွက်က Accept  
ဖြစ်လိမ့်မယ်
- ဆုလိတာက FSA ရဲ့  
အဆုံး terminal node  
အထိ ရောက်သွားတယ်

# FSA Construction (Myanmar Rooster)

```
32     === Processing wrong_input.txt ===  
33     Input FSA:  
34     0  
35     0 . 1 . အောက်  
36     1 . 2 . အုတ်  
37     2 . 3 . အောက်  
38     3 . 4 . အုတ်  
39     4  
40     Compiled input FSA:  
41     0 . . . . . 1 . . . . . အောက် . . . . . 1  
42     0  
43     1 . . . . . 2 . . . . . အုတ် . . . . . 3  
44     2 . . . . . 3 . . . . . အောက် . . . . . 1  
45     3 . . . . . 4 . . . . . အုတ် . . . . . 3  
46     4  
47     Composition result:  
48     Result: REJECTED
```

- wrong\_input.txt  
အတွက်တော့ **Reject** ပါ
- ဆိုလိုတာက pass  
မဖြစ်ဘူး

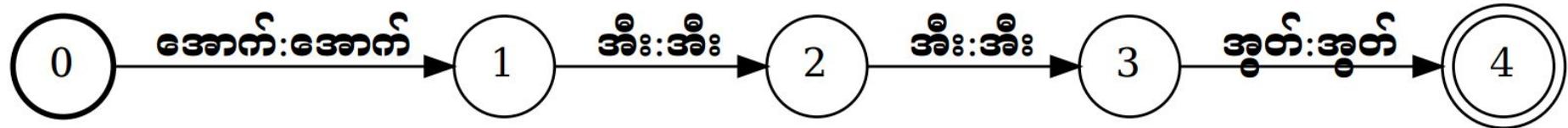
# FSA Construction (Myanmar Rooster)



- rooster.bin (i.e FSA) ကို visualization လုပ်ကြည့်လို့ ရတယ်။
- အခင်ဆုံး dot ဖိုင်ထုတ်ပြုးမှ pdf သို့မဟုတ် png ပြောင်း

```
$ fstdraw --portrait --isymbols=syllable_syms rooster.bin > rooster.dot
$ dot -Tpdf ./rooster.dot -o rooster.pdf
```

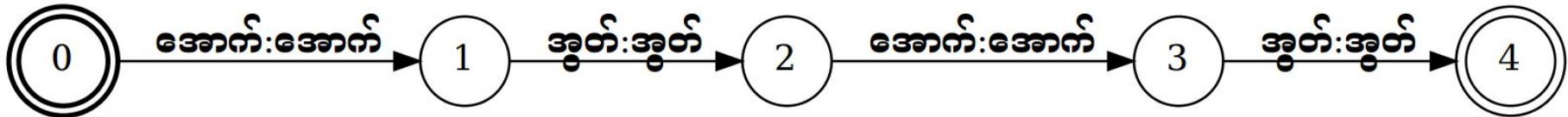
# FSA Construction (Myanmar Rooster)



- --osymbols option ပါ ပေးရင် အထက်ပါအတိုင်း output ပြောင်းသွားလိမ့်မယ်။ FSA မှိုလို output မရှိလို syllable.syms ကိုပဲ သုံးလို့ရတယ်။

```
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ fstdraw --portrait --isymbols=syllable.syms --osymbols=syllable.syms rooster.bin > rooster2.dot
```

# FSA Construction (Myanmar Rooster)



- Wrong\_input.bin ကိုလည်း pdf ဖိုင် သိမဟုတ် ပုံဖိုင်  
ထုတ်ကြည့်ရအောင်

```
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ fstdraw --portrait --isymbols=syllable_syms --osymbols=syllable_syms wrong.bin > wrong_input.dot  
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ dot -Tpdf ./wrong_input.dot -o wrong_input.pdf
```

# FSA Construction (Myanmar Rooster)

- ကြက်တွန်သံက မှားနေတဲ့ wrong.bin နဲ့ rooster.bin နဲ့ compose လုပ်ထားတဲ့ binary ဖိုင်ကို pdf ထဲတဲ့ ကြည့်တဲ့အခါမှ မထဲတဲ့ပေးနိုင်တာကို အောက်ပါအတွင်း တွေ့ရပါလိမ့်မယ်

```
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ fstdraw --portrait --isymbols=syllable_syms --osymbols=syllable_syms wrong_result.bin > wrong_result.dot
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ dot -Tpdf ./wrong_result.dot -o wrong_result.pdf
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ cat wrong_result.dot
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ ll -lh ./wrong_result.pdf
ls: cannot access './wrong_result.pdf': No such file or directory
```

# FSM Construction (Pyit Taing Htaung Simulation)



# FSM Construction (Pyit Taing Htaung Simulation)

```
(openfst) ye@lst-hpc3090:~/talk/wfst/rolling_3$ cat pyit_syms.txt
<eps> 0
push 1
gravity 2
wobble 3
balance 4
idle 5
```

```
(openfst) ye@lst-hpc3090:~/talk/wfst/rolling_3$ 
(openfst) ye@lst-hpc3090:~/talk/wfst/rolling_3$ cat input.txt
0 1 push push
1 2 gravity gravity
2 3 wobble wobble
3 4 balance balance
4
(openfst) ye@lst-hpc3090:~/talk/wfst/rolling_3$ ■
```

# FST Construction (Pyit Taing Htaung Simulation)

```
#!/bin/bash
# Usage: ./simulate.sh push gravity wobble balance ...

if [ $# -eq 0 ]; then
    echo "Usage: $0 input1 input2 ..."
    exit 1
fi

rm -f input.txt result.txt input.fst result.fst

echo "0 1 $1 $1" > input.txt
i=1
shift

for sym in "$@"; do
    echo "$i $((i+1)) $sym $sym" >> input.txt
    i=$((i+1))
done

echo "$i" >> input.txt

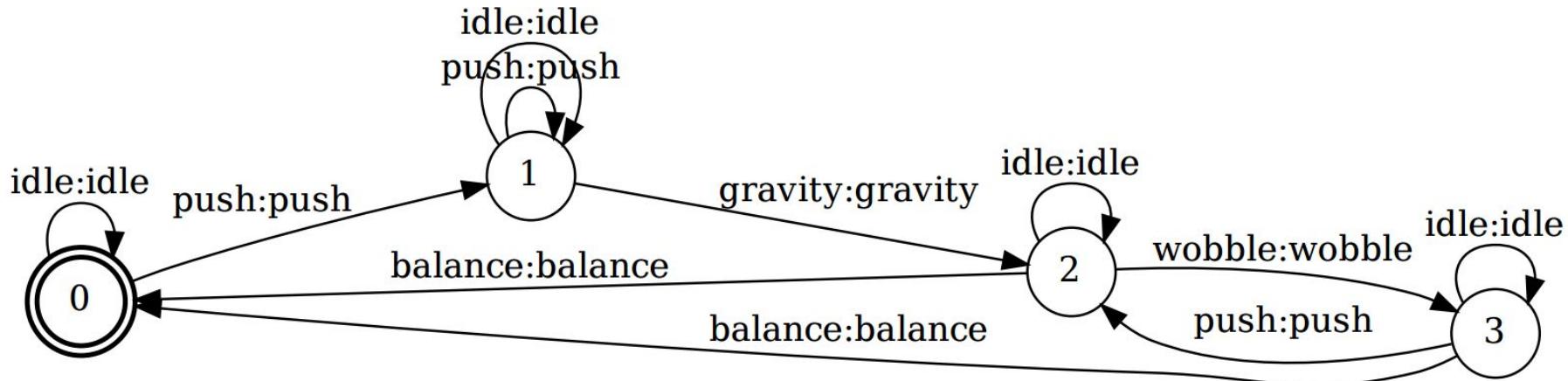
fstcompile --isymbols=pyit_syms.txt --osymbols=pyit_syms.txt input.txt > input.fst
fstcompose input.fst pyit_fsm.fst > result.fst
fstprint --isymbols=pyit_syms.txt --osymbols=pyit_syms.txt result.fst > result.txt

echo "Simulation result:"
cat result.txt

fstdraw --portrait --isymbols=pyit_syms.txt --osymbols=pyit_syms.txt result.fst | dot -Tpdf > result.pdf
echo "Result graph saved to result.pdf"
```

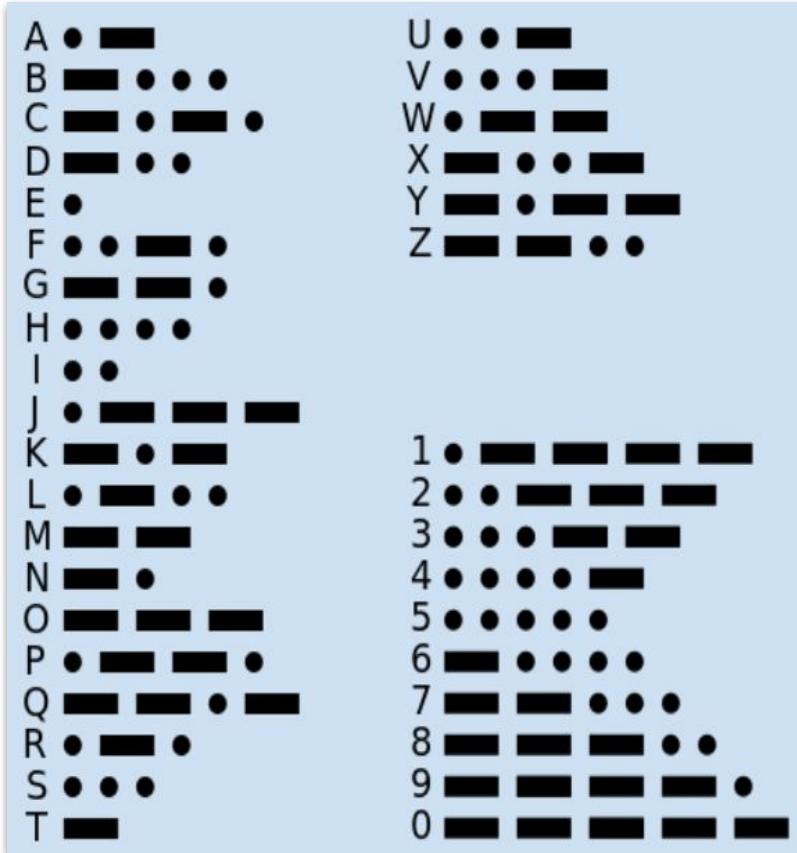
Fig. simulate.sh

# FSA Construction (Pyit Taing Htaung Simulation)



- ပစ်တိုင်းထောင် အရှပ်ကို simulation အကြမ်း လုပ်ကြည့်တာပါ
- ဒီမှာ ပြောနေအတဲ့ idle ဆုတေဂါ လက်ရှု ရောက်နေတဲ့ state မှာပဲ ပုံမှန်ဆက်သွားနေတောက် ဆုလုံတာပါ။ ရှုပေးခွဲ အနေနဲ့ ကည့်ရင်တွေ့ပြောစရာတွေ ရှိပါလိမ့်မယ်...

# FST Construction (Morse encoder/decoder)



- Rules/heuristic process  
တွေအတွက် WFST က  
တအားကောင်းတယ်
- ဥပမာ အနေနဲ့ အဂ်လိုပ်စာကနေ  
မောစ်ကုံး (i.e. encoding)၊  
မောစ့်ကုံး ကနေ အဂ်လိုပ်စာ  
(i.e. decoding) လုပ်လို့ ရတယ်

# FST Construction (Morse encoder/decoder)

```
(openfst) ye@lst-hpc3090:~/talk/wfst/morse2$ ./tst_run1.sh  
==== English to Morse: 'SOS' ====  
0      1      19      ...  
1      2      15      ---  
2      3      19      ...  
3  
==== Morse to English: '... --- ...' ====  
0      1      19      S  
1      2      15      0  
2      3      19      S  
3  
(openfst) ye@lst-hpc3090:~/talk/wfst/morse2$ -
```

# FST Construction (Morse encoder/decoder)

```
(openfst) ye@lst-hpc3090:~/talk/wfst/morse2$ head -n 15 ./symbols/input.syms
<eps> 0
A 1
B 2
C 3
D 4
E 5
F 6
G 7
H 8
I 9
J 10
K 11
L 12
M 13
N 14
```

- အက်လိပ်စာအတွက် symbol ဖို့

# FST Construction (Morse encoder/decoder)

```
(openfst) ye@lst-hpc3090:~/talk/wfst/morse2$ head -n 15 ./symbols/output.syms
<eps> 0
. - 1
- . . . 2
- - . . 3
- . . . 4
. . 5
.. - . . 6
-- . . 7
... . . . 8
.. . . 9
. - - - . 10
- . - . . 11
. - . . . 12
-- . . . 13
- . . . . 14
```

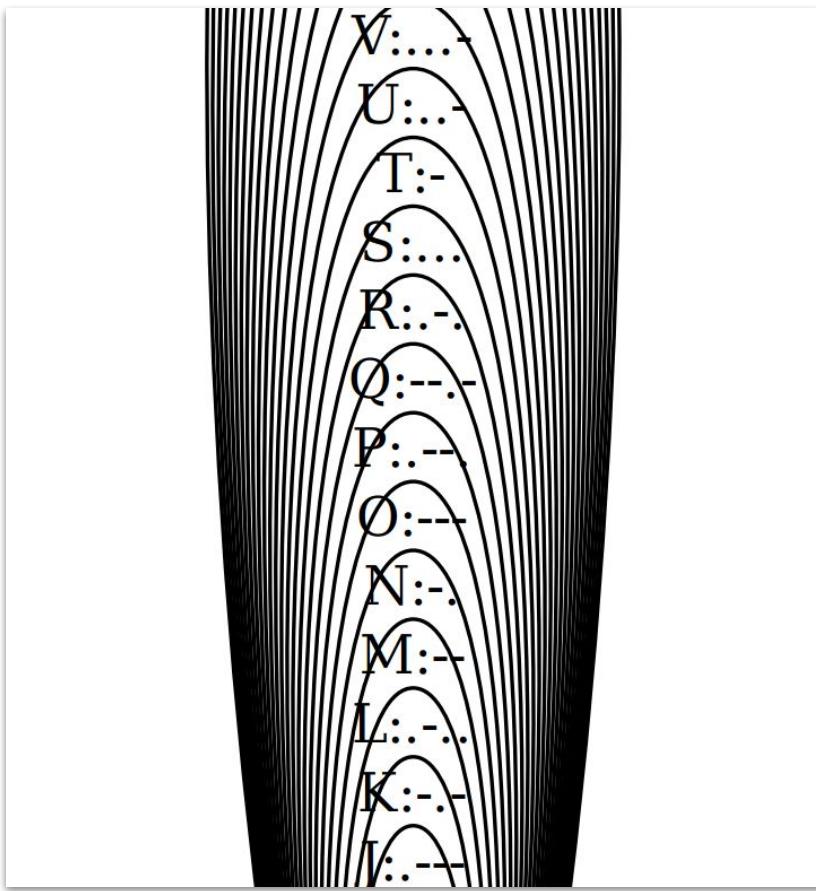
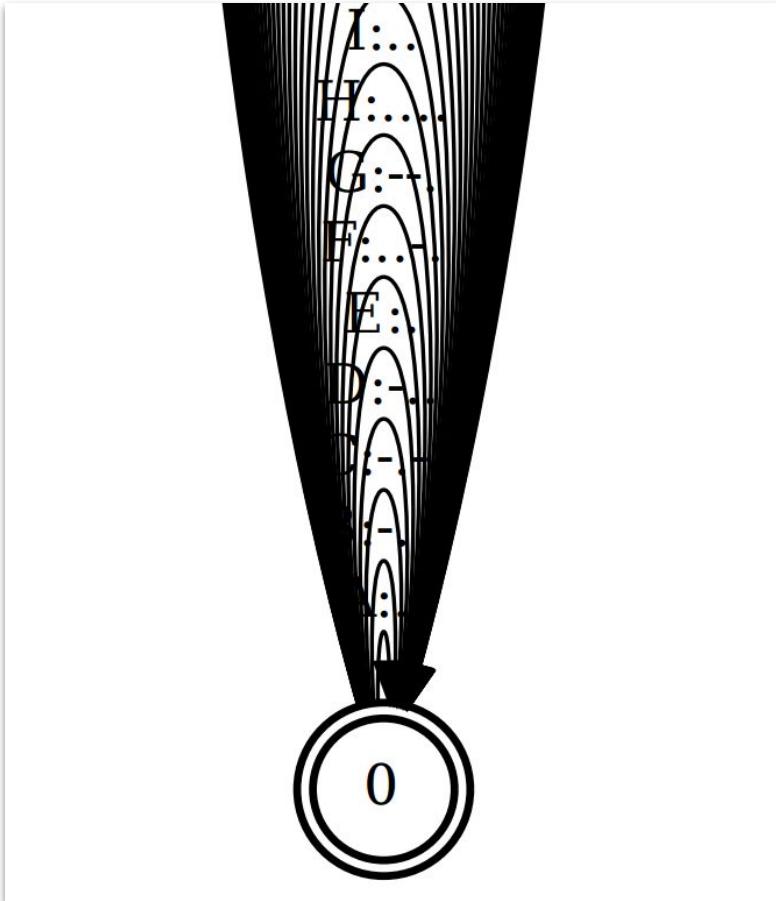
- မောစွဲကုဒ်အတွက် symbol ဖိုင်

# FST Construction (Morse encoder/decoder)

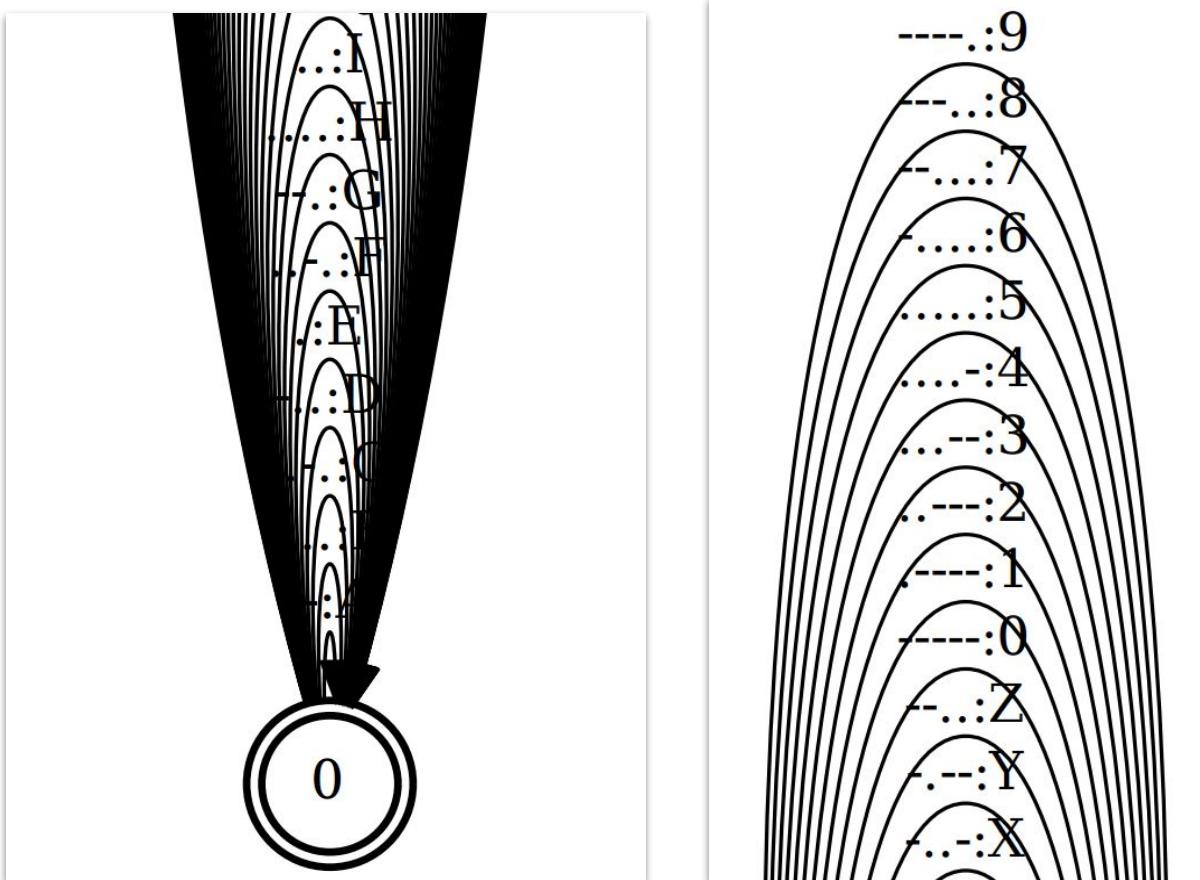
```
(openfst) ye@lst-hpc3090:~/talk/wfst/morse2$ tail -n 15 ./symbols/output.syms  
...- 22  
. -- 23  
-..- 24  
-.- 25  
--.. 26  
----- 27  
.---- 28  
..--- 29  
...-- 30  
....- 31  
..... 32  
-..... 33  
--... 34  
--.-.. 35  
--.--. 36
```

- මෙයුත්ගැනීමට පෙන්වනු ලබන symbol සංශෝධනය

# FST Construction (en2morse)



# FST Construction (morse2en)



- Lexicon ကို  
နစ်ခေါ်ဆောက်ထာ  
းဖွဲ့လုပ်အပ်
- en2morse
- morse2en

# FST Construction (Morse encoder/decoder)

```
(openfst) ye@lst-hpc3090:~/talk/wfst/morse2$ cat build_fsts.sh
#!/bin/bash
mkdir -p bin

# Build en2morse FST
fstcompile --isymbols=symbols/input.syms --osymbols=symbols/output.syms \
    fst/lexicon.txt | fstarcsort > bin/en2morse.fst

# Build morse2en FST (by inverting)
fstinvert bin/en2morse.fst > bin/morse2en.fst
```

- အထက်ပါ bash shell script က ပထမဆုံး en2morse FST ကိုဆောက်
- ပြီးတော့ fstinvert ဆိတ္တဲ့ command ကိုသုံးပြီး en2morse ကနေ morse2en အဖြစ် ပြောင်းလိုက်တယ်

# OpenFST Operations

The screenshot shows the homepage of the OpenFST library. At the top, there is a navigation bar with icons for back, forward, search, and user profile. The URL is openfst.org/twiki/bin/view/FST/WebHome. Below the navigation bar, there is a diagram of a finite-state transducer (FST) with 11 states labeled 0 through 10. Transitions are labeled with pairs of input and output symbols: (O:ow), (P:p), (e:ax), (n:n), (F:eh), (-f), (s:eh), (-s), (t:t), and (-:iv). State 10 is highlighted with a double circle. To the right of the diagram are search and jump fields. Below the diagram, the page title is "TWiki > FST Web > WebHome (2024-12-23, LawrenceWolfSonkin)". On the right side, there are "Edit" and "Attach" buttons. The main content area has a red header "OpenFst Library". A red info icon points to a message about a new version: "OpenFst version 1.8.4 is now available for [download](#)". A green "NEW" icon points to a note about conda-forge availability: "OpenFst is now also available on [conda-forge](#). Linux (x86) and Mac OS X users who already have [conda](#) can install using the following command: `conda install -c conda-forge openfst`". The main text describes OpenFst as a library for constructing, combining, optimizing, and searching weighted finite-state transducers (FSTs). It explains the difference between acceptors and transducers, and lists various applications and optimization techniques. The text concludes by mentioning contributors from Google Research and NYU's Courant Institute, and the Apache license under which it is distributed.

**OpenFst Library**

**OpenFst version 1.8.4 is now available for [download](#).**

**NEW** OpenFst is now also available on [conda-forge](#). Linux (x86) and Mac OS X users who already have [conda](#) can install using the following command: `conda install -c conda-forge openfst`.

**OpenFst** is a library for constructing, combining, optimizing, and searching *weighted finite-state transducers* (FSTs). Weighted finite-state transducers are automata where each transition has an input label, an output label, and a [weight](#). The more familiar finite-state acceptor is represented as a transducer with each transition's input and output label equal. Finite-state acceptors are used to represent sets of strings (specifically, *regular* or *rational sets*); finite-state transducers are used to represent binary relations between pairs of strings (specifically, *rational transductions*). The weights can be used to represent the cost of taking a particular transition. FSTs have key applications in speech recognition and synthesis, machine translation, optical character recognition, pattern matching, string processing, machine learning, information extraction and retrieval among others. Often a weighted transducer is used to represent a probabilistic model (e.g., an *n-gram model*, *pronunciation model*). FSTs can be optimized by [determinization](#) and [minimization](#), models can be applied to hypothesis sets (also represented as automata) or cascaded by finite-state [composition](#), and the best results can be selected by [shortest-path](#) algorithms.

This library was developed by [contributors](#) from Google Research and NYU's Courant Institute. It is intended to be comprehensive, reliable, flexible, efficient, and to scale well. It is an open source project distributed under the [Apache](#) license.

Fig. Homepage of OpenFST

# OpenFST Operations

The screenshot shows a web page from the ACL Anthology. At the top left is the ACL Anthology logo (a red square with a white 'A'). To its right are links for News, FAQ, Corrections, Submissions, and Github. On the far right is a search bar with a magnifying glass icon. The main title of the paper is "OpenFst: An Open-Source, Weighted Finite-State Transducer Library and its Applications to Speech and Language". Below it are the authors' names: Michael Riley, Cyril Allauzen, Martin Jansche. A horizontal line separates this from the metadata. The metadata includes: Anthology ID: N09-4005; Volume: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Tutorial Abstracts; Month: May; Year: 2009; Address: Boulder, Colorado; Editors: Ciprian Chelba, Paul Kantor, Brian Roark; Venue: NAACL; SIG: –; Publisher: Association for Computational Linguistics; Note: –; Pages: 9–10; Language: –; URL: <https://aclanthology.org/N09-4005/>. To the right of the metadata are four buttons: a blue 'PDF' button, a grey 'Cite' button, a grey 'Search' button, and a yellow 'Fix data' button.

ACL Anthology News FAQ Corrections Submissions Github

Search...

## OpenFst: An Open-Source, Weighted Finite-State Transducer Library and its Applications to Speech and Language

Michael Riley, Cyril Allauzen, Martin Jansche

**Anthology ID:** N09-4005

**Volume:** Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Tutorial Abstracts

**Month:** May

**Year:** 2009

**Address:** Boulder, Colorado

**Editors:** Ciprian Chelba, Paul Kantor, Brian Roark

**Venue:** NAACL

**SIG:** –

**Publisher:** Association for Computational Linguistics

**Note:** –

**Pages:** 9–10

**Language:** –

**URL:** <https://aclanthology.org/N09-4005/>

PDF Cite Search Fix data

# OpenFST Operations

```
(openfst) ye@lst-hpc3090:~/talk/wfst$ fst
fstab-decode      fstdeterminize   fstintersect    fstproject     fstrmepsilon
fstarcsort        fstdifference    fstinvert       fstprune       fstshortestdistance
fstclosure        fstdisambiguate  fstisomorphic   fstpush        fstshortestpath
fstcompile        fstdraw          fstlinear       fstrelabel     fstspecial
fstcompose        fstencode         fstloglinearapply fstreplace    fstsymbols
fstcompress       fstepsnormalize  fstmap          fstreverse    fst synchronize
fstconcat         fstequal         fstminimize    fstreweight   fsttopsort
fstconnect        fstequivalent   fststopgm     fsttrim       fstunion
fstconvert        fstinfo          fstprint       fstrim
```

- Installation လပ်လိုအဆင်ပြေရင် အောက်ပါလိုမျိုး command တွေကို  
သုံးလို့ ရပါလိမ့်မယု
- Version မတူရင် တချို့ command line parameter တွေ တူမှာ  
မဟုတ်ဘူး

# OpenFST Operations (Rational Operations)

- A.txt

0 1 dog 0.5  
1 2 cat 0.5  
2 0.5

- B.txt

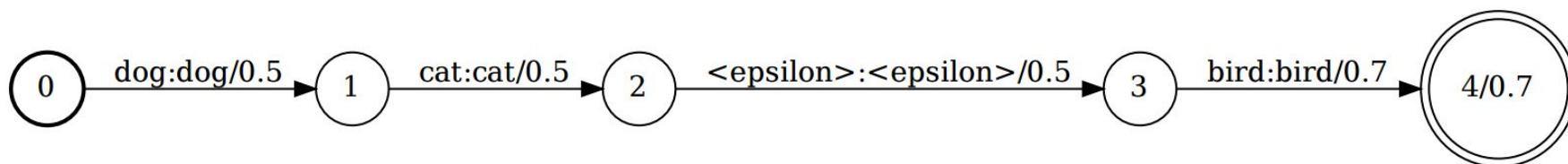
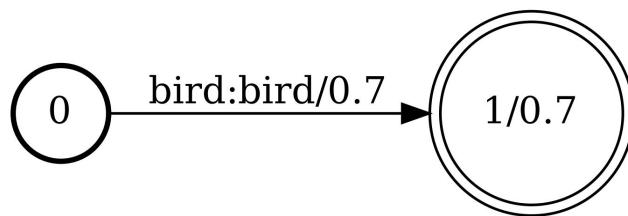
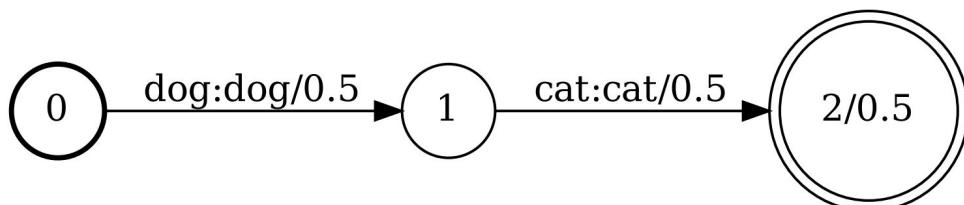
0 1 bird 0.7  
1 0.7

- syms.txt

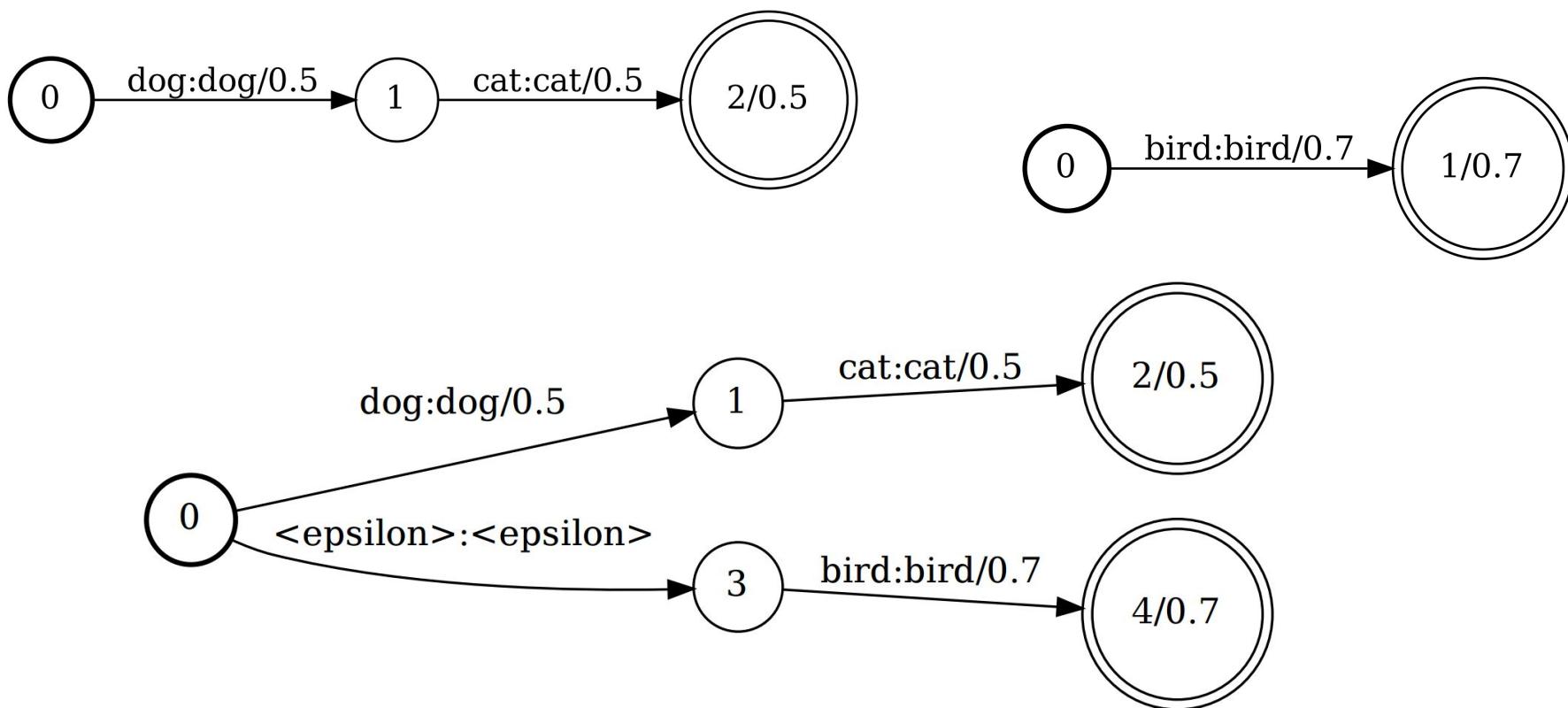
<epsilon> 0  
dog 1  
cat 2  
bird 3

- A.txt, B.txt ရယ်၊ syms.txt ဖိုင် သုံးဖန်ကို create လုပ်မယ်
- Union, concat, closure ဆိုတဲ့ command သုံးခုကို မိတ်ဆက်မယ်

# OpenFST Operations (Concat = Product)



# OpenFST Operations (Union =Sum)



# OpenFST Operations (Closure)

- The closure operation (also called Kleene closure) in OpenFST creates a finite-state acceptor/transducer that accepts zero or more repetitions of the original FST's language. It's denoted as  $A^*$  and is fundamental in regular expressions.

# OpenFST Operations (Closure)

## Mathematical Definition

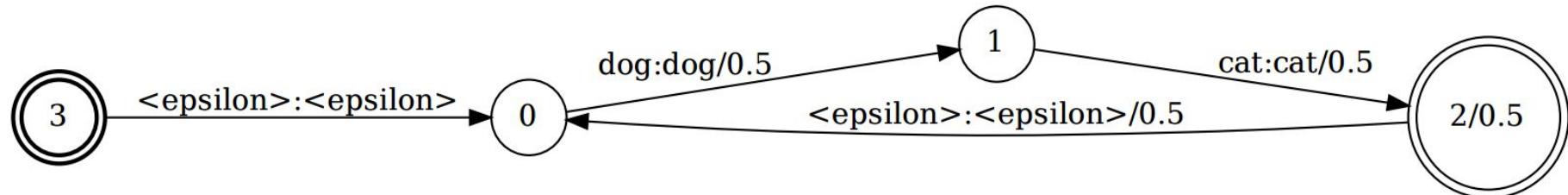
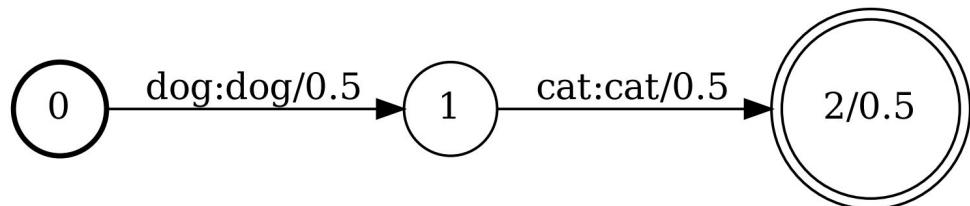
For an FST  $A$ , its closure  $A^*$  satisfies:

$$A^* = \{\epsilon\} \cup A \cup AA \cup AAA \cup \dots$$

where:

- $\epsilon$ : Empty string (zero repetitions)
- $A$ : Original language
- $AA$ : All two-sequence concatenations

# OpenFST Operations (Closure)



# OpenFST Operations (Closure)

- Now accept:
- $\epsilon$  (empty string)
- "dog cat"
- "dog cat dog cat"
- "dog cat dog cat dog cat"
- etc.

# Spoken Dialect Translation



Multidisciplinary | Rapid Review | Open Access Journal

Received 15 October 2022, accepted 27 December 2022, date of publication 13 January 2023, date of current version 20 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3236804

 APPLIED RESEARCH

## Transfer and Triangulation Pivot Translation Approaches for Burmese Dialects

**THAZIN MYINT OO<sup>ID1</sup>, THITIPONG TANPRASERT<sup>ID1</sup>,  
YE KYAW THU<sup>ID2</sup>, AND THEPCHAI SUPNITHI<sup>2</sup>**

<sup>1</sup>Vincent Mary School of Science and Technology, Assumption University of Thailand, Bangkok 10240, Thailand

<sup>2</sup>Language and Semantic Technology Research Team (LST), Artificial Intelligence Research Group (AINRG), National Electronics and Computer Technology Center (NECTEC), Khlong Nueng, Pathumtani 12120, Thailand

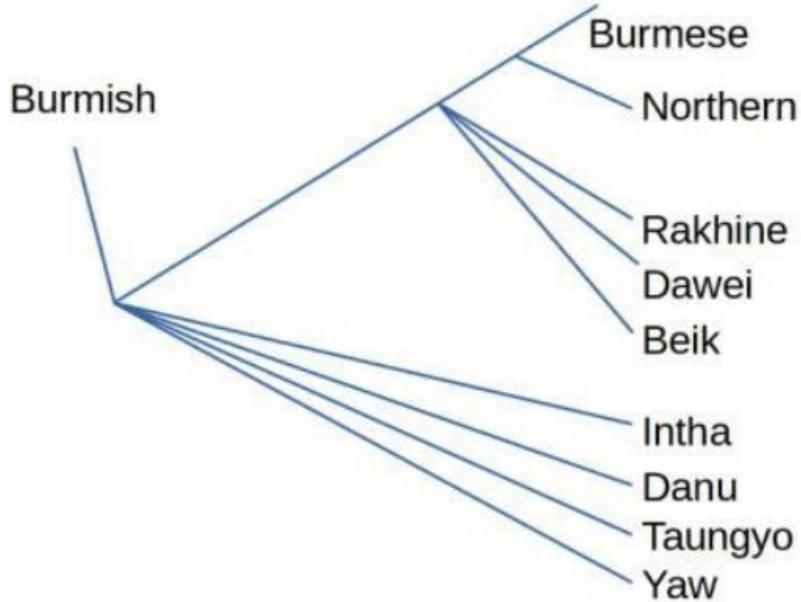
Corresponding author: Ye Kyaw Thu (yekyaw.thu@nectec.or.th)

# Spoken Dialect Translation

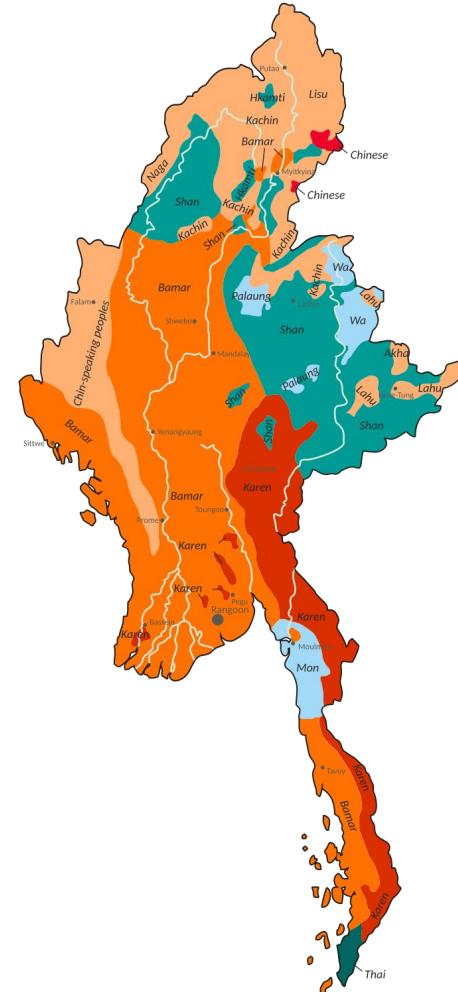
**ABSTRACT** Parallel corpora for the languages of Myanmar (Beik, Burmese, Rakhine) are extremely scarce but a necessary requirement for machine translation R&D. Previous studies have proved that pivoting leads to better translation quality if the bridge language is closely related to the source and target language pair. The baseline study is conducted based on the three major approaches of machine translation; Weighted Finite State Transducer (WFST), Phrase-Based Statistical Machine Translation (PBSMT) and Deep Recurrent Neural Network (Deep-RNN). Based on the baseline results, this paper mainly investigated the pivot language technique for PBSMT with Burmese dialects. We employed two different pivot translation methods: transfer (sentence level) and triangulation (phrase level). We present the experimental results on Dawei-Beik, Beik-Dawei translations and Beik-Rakhine, Rakhine-Beik translation via Burmese. Both the transfer and triangulation approaches outperformed the baseline (direct translation), specifically for the Rakhine-Beik language pair. Moreover, the results of the average BiLingual Evaluation Understudy (BLEU), Character  $n$ -gram F-score ( $chrF$ ), and Word Error Rate (WER) scores of the 10-fold cross-validation experiments proved that the triangulation pivot has significantly better acceleration than the transfer pivot. We plan to release the parallel corpora of Burmese dialects and present several avenues for further research.

**INDEX TERMS** Burmese dialects, pivot translation, transfer, triangulation, machine translation.

# Spoken Dialect Translation



**FIGURE 1.** Classification of burmese (adapted from Bradley 1997: 38-39 [20]).



# Spoken Dialect Translation

The following are some examples of parallel sentences in Burmese (bm) and Dawei (dw):

dw: သယ်ဝယ်သား က လူမြင်း ဟုယ် ။

bm: ဒီကောင်မလေး က လူလွန်း တယ် ။

(“The girl is so beautiful” in English)

dw: လတ်ဖတ်ရယ် က ရှိမြင်း ဟုယ် ။

bm: လက်ဖက်ရည် ခိုလွန်း တယ် ။

(“The tea is so sweet” in English)

dw: ကော်သား ကော် မှန်းမှန် သွား ဟုယ် ။

bm: ကောင်လေး ကောင်း မှန်မှန် တက် တယ် ။

(“The boy goes to school regularly” in English)

# Spoken Dialect Translation

bk: မင်း ငါ ကို ကြေးပြား ပေး ပို့ မေ့ နေရယ် လား ။

bm: မင်း ငါ ကို ပိုက်ဆံ ပေး ပို့ မေ့ နေပြီလား ။

(“Did you forget to pay me?” in English)

bk: ငါ မောင်င်း နိုင်ငံခြား သော မယ်။

bm: ကျွန်တော် မနက်ဖြန် နိုင်ငံခြား သွား မယ် ။

(“I will go to a foreign country tomorrow.” in English)

bk: ကျွန်တော် အယ် ပို့ လာ ရအေ ပျော် ရယ် ။

bm: ကျွန်တော် ဒ္ဓါ လာ ရတေ ပျော် တယ် ။

(“I am happy to be here.” in English)

# Spoken Dialect Translation

(SOV). Examples of parallel sentences in Myanmar (bm) and Rakhine (rk) are as follows:

rk: အယော တစ် ထည် ရာလောက်လေး။

bm: လုံချိုင် တစ် ထည် ဘယ်လောက်လဲ။

("How much for a longyi?" in English)

rk: ကလေချွေ တိ ဘေးလုံး ကျောက် နှီရေး။

bm: ကောင်လေး တွေ ဘေးလုံး ကန် နေတယ်။

("Boys are playing football" in English)

rk: အ ပြော နိချင့် ယင်းသူရှို့။

bm: သူတို့ ဘာ ပြော နေတာလဲ။

("What are they talking about" in English)

rk: အဘောင်သူ၏ စီးက သပုံ ဝယ် လာတယ်။

bm: အဘား စွေးက ဆပ်ပြာ ဝယ် လာတယ်။

("The grandmother buys soap from the market" in English)

- မြန်မာစာကို pivot အဖြစ်ထားပြီး experiment လုပ်ခဲ့တယ်
- WFST-based machine translation ကိုလည်း လုပ်ခဲ့တယ်

# Spoken Dialect Translation

## 2) WFST BASED TRANSLATION MODEL

The translation process that takes in a string and translates it into another string (i.e. source to target) can be expressed with WFST. WFSTs are basically similar to WFSAs but use two labels on every edge. It specifies a mapping between two sets of strings. To give a very simple translation model even simpler than IBM Model 1: it calculates  $P(R|B)$  for translation from Burmese ( $B$ ) to Rakhine ( $R$ ) by taking one  $b_t$  at a time and independently calculates the translation probability of the corresponding word  $r_t$

$$P(R|B) = \prod_{t=1}^{|B|} P(r_t|b_t) \quad (2)$$

## Spoken Dialect Translation

We assume the interpolation coefficient is  $\alpha = 0.1$  and if the bigram count is non zero, such as  $P(\text{ချိန်} | \text{ကောင်})$ , the simple smoothing unigram and bigram probabilities becomes:

$$\begin{aligned} P(r_t | r_{t-1}) &= (1 - \alpha) P_{ML}(r_t | r_{t-1}) + \alpha P_{ML}(r_t) \\ P(r_t = \text{ချိန်} | r_{t-1} = \text{ကောင်}) &= (1 - \alpha) P_{ML}(r_t = \text{ချိန်} | r_{t-1} = \text{ကောင်}) \\ &\quad + \alpha P_{ML}(r_t = \text{ချိန်}) \\ &= 0.9 \frac{1}{3} + 0.1 \frac{1}{12} \\ &= 0.308\bar{3} \end{aligned} \tag{1}$$

# Spoken Dialect Translation

## 1) WEIGHTED FINITE STATE AUTOMATA (WFSA) BASED LANGUAGE MODEL

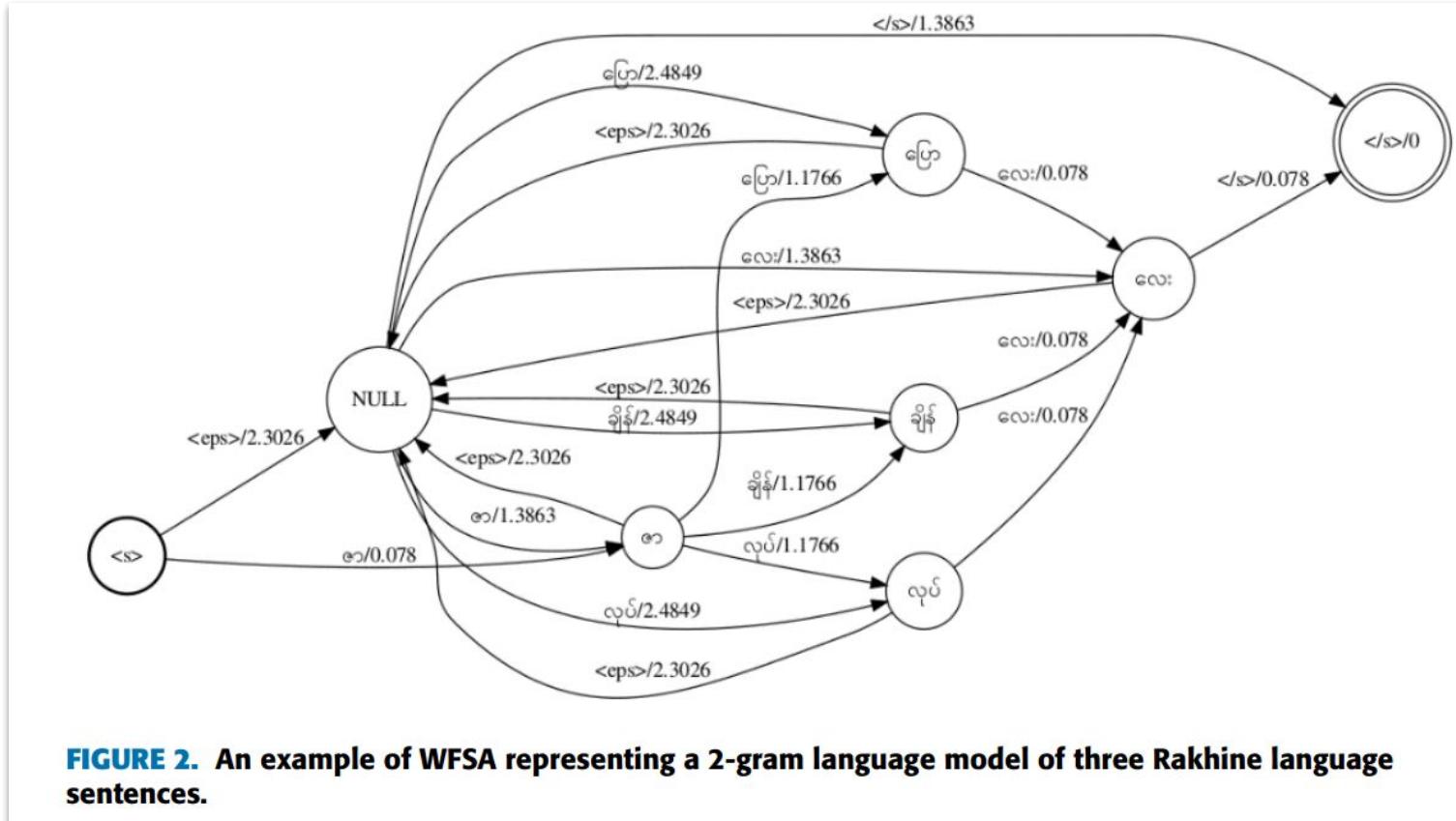
Weighted Finite State Automaton (WFSA) are able to work like a language model (i.e. the smoothed *n-gram* languages model) by expressing sets of strings with corresponding scores over them. For example we have a 2-gram language model interpolated [40] over the set of Rakhine words “၏” (what), “ချိန်” (time), “လေး” (question particle word in Rakhine), “ပြော” (say), “လုပ်” (do) calculated from the following three lines of the Rakhine corpus:

Rakhine Sentence 1: ၏ ချိန် လေး

Rakhine Sentence 2: ၏ ပြော လေး

Rakhine Sentence 3: ၏ လုပ် လေး

# Spoken Dialect Translation



# Spoken Dialect Translation

Here, we can learn translation probabilities for each word, for example:

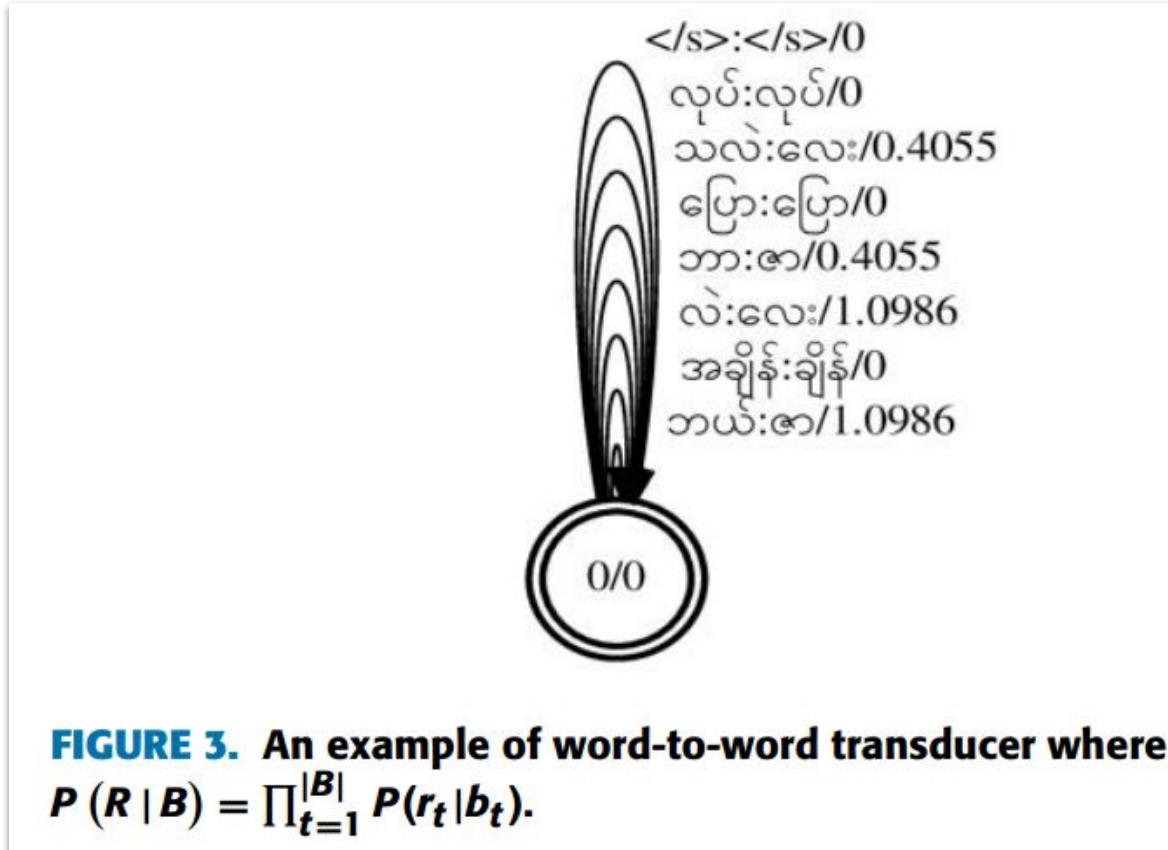
$$P(r = \text{ချိန်} | b = \text{အချိန်}) = 1$$

$$P(r = \text{သလဲ} | b = \text{ငလူ}) = 0.66\bar{6}$$

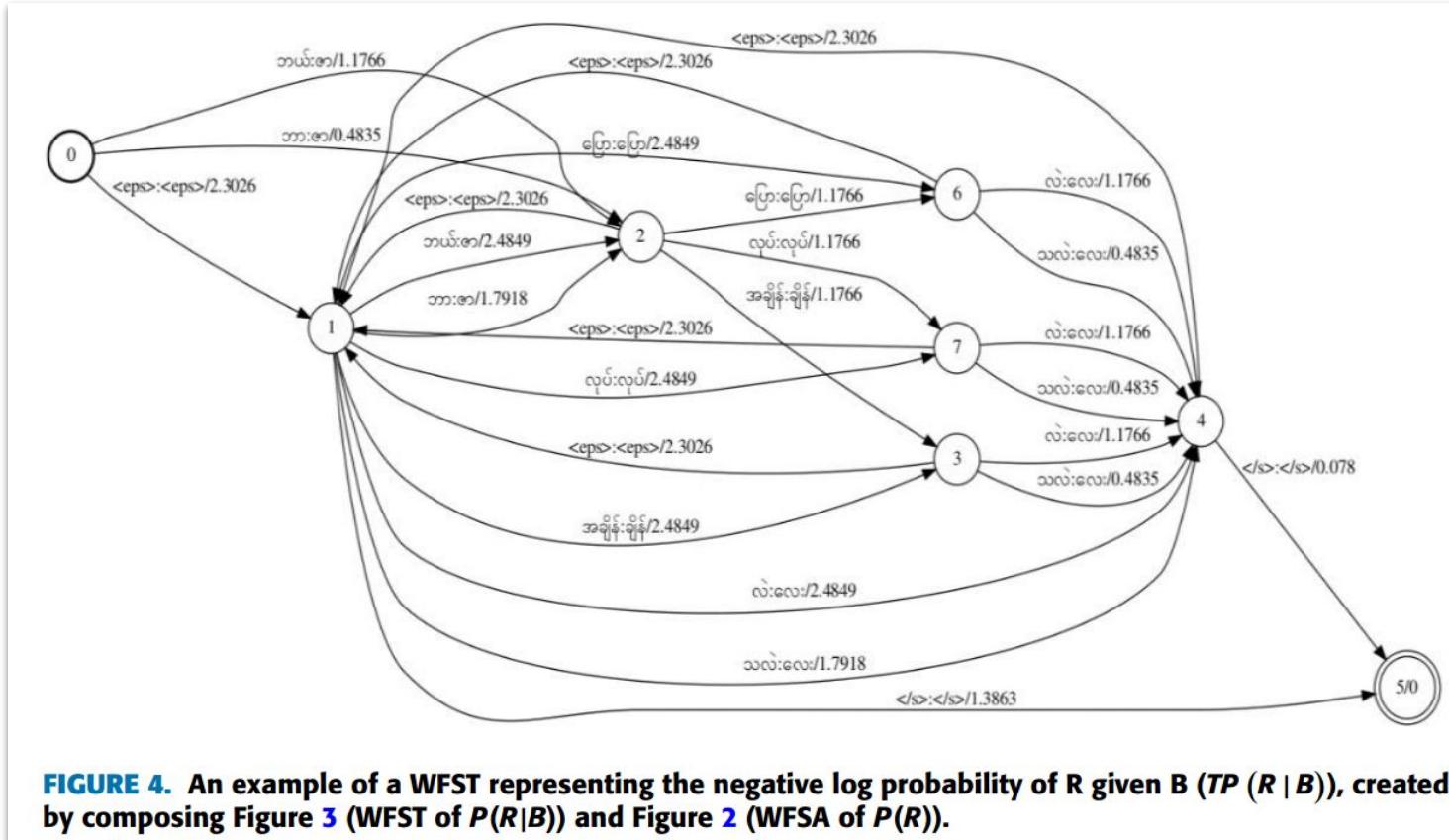
$$P(r = \text{ဘယ်} | b = \text{ဘု}) = 0.33\bar{3}$$

Because translation probability is independent of the others, we do not need to keep the “state” of the translation model and thus we can just employ one input state and one output state per edge. Fig. 3 shows an example of a WFST representing the translation model of three lines Burmese-Rakhine parallel corpus.

# Spoken Dialect Translation



# Spoken Dialect Translation



# WFST POS Tagging Demo

The screenshot shows a Jupyter Notebook interface. The top navigation bar includes 'View', 'Run', 'Kernel', 'Tabs', 'Settings', and 'Help'. A tab labeled 'WFST\_POS\_Tagging\_Small\_Crx' is active. Below the tabs is a toolbar with icons for file operations like open, save, and run, along with a 'Markdown' dropdown and a kernel status indicator for 'Python 3 (ipykernel)'. The main content area features a section header 'WFST POS Tagging Tiny Demo' with a dropdown arrow, followed by author information ('by Ye Kyaw Thu, Lab Leader, LST Lab., Myanmar') and a date ('Date: 17 July 2025'). The bottom part of the interface shows a code cell with the command 'cd /home/ye/exp/tiny\_pos' and its output '/home/ye/exp/tiny\_pos'.

## WFST POS Tagging Tiny Demo

by Ye Kyaw Thu, Lab Leader, LST Lab., Myanmar  
Date: 17 July 2025

## Shell Scripts

ဒါ Lab မှာက OpenFST command တွေကို သုံးပြီး NLP field မှာ အရေးကြီးတဲ့ task တစ်ခုဖြစ်တဲ့ POS (Part-of-Speech) Tagging ကို လက်တွေ လုပ်ပြသွားမယ်။  
ကိုယ့်စက်ထဲမှာ လိုက်လုပ်ဖို့အတွက်က လိုအပ်တဲ့ OpenFST နဲ့ တော်း python library တွေက ကြိုတင် installation လုပ်ထားရလိမ့်မယ်။

```
6]: cd /home/ye/exp/tiny_pos
/home/ye/exp/tiny_pos
```

# Best Practices

- **For FSAs (Acceptors):**
  - Always use `--acceptor` flag
  - Weights go at the end of transition lines
  - Final state weight is separate line
- **For FSTs (Transducers):**
  - Never use `--acceptor`
  - Weights come after output symbols
  - Format: `src dest in out [weight]`
- **Symbol Tables:**
  - FSAs only need `--isymbols`
  - FSTs need both `--isymbols` and `--osymbols`

# Conclusion

- FSA, FST, WFST ကို မိတ်ဆက်ပေးခဲ့တယ်
- အခြေခံ သီအိုရီရော၊ လက်တွေပိုင်းတွေရော
- OpenFST framework ကိုသုံးပြုခဲ့တယ်
- WFST ကိုသုံးပြီး spoken dialect language pair တွေအကြား (ဥပမာ ဘိတ်-ထားဝယ်၊ ဗမာ-ရခိုင်၊ ဗမာ-ပအိုဝ္ဗ) လုပ်လို့ ရကြောင်းသင်ကြားပေးခဲ့
- POS tagging ကိုလည်း WFST ကိုအခြေခံပြီး လုပ်လို့ရကြောင်းလက်တွေ လုပ်ပြုခဲ့
- သုံးလို့ရတဲ့ နေရာတွေက အများကြီးပါ (e.g. Automatic Speech Recognition, Text to Speech, Machine Translation, Parsing etc.)

# Conclusion

The screenshot shows the GitHub profile page for the user `ye-kyaw-thu`. The profile picture is a circular drawing of a face with various colored lines and shapes. The user's name is **Ye Kyaw Thu** and their GitHub handle is `ye-kyaw-thu`. Below the profile picture is a button labeled "Edit profile". The GitHub URL `github.com/ye-kyaw-thu` is visible at the top. The profile has 55 repositories, 9 stars, and 556 followers. A search bar is present at the top right. The main content area is titled "Pinned" and contains four repository cards:

- wfst\_nlp\_tutorials** (Public) - WFST for NLP: Tutorials on word segmentation, POS tagging, and machine translation. (Jupyter Notebook, 5 stars)
- myFlickr30k** (Public) - Myanmar language translation of Flickr30k. (HTML, 2 stars)
- myHateSpeech** (Public) - myHateSpeech: Myanmar hate speech datasets and experiments. (Jupyter Notebook, 3 stars)
- myContradict** (Public) - Contradictory sentence generation. (Jupyter Notebook)
- myMediCon** (Public) - myMediCon: A Medical Conversation Corpus for the Myanmar language, designed for both text and speech. (3 stars)
- lipidiipikar** (Public) - Lipidiipikar is the first telegraph code invented by Yaw Min Gyi U Pho H. This card is partially cut off.

- လုပ်ခဲ့တဲ့အလုပ်တော်များများကို github မှာ  
ကင်ပေးကြားပါကျွုံ

# Conclusion

[sites.google.com/site/yekyawthunlp/](https://sites.google.com/site/yekyawthunlp/)

## Ye Kyaw Thu



Ye Kyaw Thu (Ye-san) is a Visiting Professor of Language & Semantic Technology Research Team (LST), Artificial Intelligence Research Unit (AINRU), National Electronic & Computer Technology Center (NECTEC), Thailand, Affiliate Professor at Cambodia Academy of Digital Technology (CADT), Cambodia and Head of NLP Research Lab., University of Technology Yatanarpon Cyber City (UTYCC), Pyin Oo Lwin, Myanmar. He is also a founder of Language Understanding Lab., Myanmar. He is actively co-supervising/supervising undergrad, masters' and doctoral students of several universities including Assumption University (AU), Cambodia Academy of Digital Technology (CADT), Institute of Technology of Cambodia (ITC), King Mongkut's Institute of Technology Ladkrabang (KMITL), Kasetsart University (KU), and Sirindhorn International Institute of Technology (SIIT). Previously, he was a Visiting Researcher of Language and Speech Science Research Lab., Waseda University, Japan (2012-2021), a Researcher in the artificial intelligence (AI) Lab. at Okayama Prefectural University (OPU), Japan (2016-2018), a Researcher at the Multilingual Translation Lab., Advanced Speech Translation Research and Development Promotion Center, National Institute of Information and Communications Technology (NICT), Japan (2012-2016) and a Research Associate at Waseda University, Japan (2009-2012). He received his B.Sc. in Physics from Dagon University, Myanmar, M.Sc. and D.Sc. both in global information and telecommunication studies from Waseda University, Japan. His research interests lie in the fields of AI, natural language processing (NLP) and human-computer interaction (HCI). His experience includes

- Homepage မှာလည်း ရေးထားတဲ့ စာတမ်း တော်တော်များများ တင်ပေးထား

Thank you!  
Q&A?

(ykt.nlp.ai@gmail.com)

# References

1. DAG: [https://en.wikipedia.org/wiki/Directed\\_acyclic\\_graph](https://en.wikipedia.org/wiki/Directed_acyclic_graph)
2. Finite State Machines in Hardware (Volnei A. Pedroni, the MIT Press, 2013):  
<https://direct.mit.edu/books/monograph/4016/Finite-State-Machines-in-HardwareTheory-and-Design>
3. OpenFST Library: <https://www.openfst.org/twiki/bin/view/FST/WebHome>
4. Weighted Finite-State Transducer Algorithms An Overview:  
<https://cs.nyu.edu/~mohri/pub/fla.pdf>
5. Finite-state text processing tutorials:  
<https://wellformedness.com/courses/fstp/>
6. Rewrite Rules Demo:  
[https://colab.research.google.com/drive/1pE1Q3rR\\_EOnqBUM-xMMjrPqPpjQjiPcx](https://colab.research.google.com/drive/1pE1Q3rR_EOnqBUM-xMMjrPqPpjQjiPcx)

# References

7. Helsinki Finite-State Technology (library and application suite): <https://github.com/hfst/hfst>
8. HFST Project Page: <https://hfst.github.io/>
9. Mehryar Mohri, Fernando Pereira, Michael Riley, Weighted finite-state transducers in speech recognition, Computer Speech & Language, Volume 16, Issue 1, 2002, Pages 69-88, ISSN 0885-2308, <https://doi.org/10.1006/csla.2001.0184>.
10. Weighted Finite State Transducers in Speech Recognition Slide:  
[https://cdn.wildapricot.com/419424/resources/Documents/Outreach/Distinguished%20lectures/MichaelRiley2014SeptSlides.pdf?version=1698754341000&Policy=eyJTdGF0ZW1lbnQiOiBbeyJSZXNvdXJjZSI6Imh0dHBzOi8vY2RuLndpbGRhcHJpY290LmNvbS80MTk0MjQvcvVzb3VyY2VzL0RvY3VtZW50cy9PdXRyZWFrjaC9EaN0aW5ndWIzaGVkJTlwbGVjdHVyZXMuTWIjaGFlbFJpbGV5MjAxNFNIcHRTbGlkZXMuGRmP3ZlcnNpb249MTY5ODc1NDM0MTAwMCIsIkNvbmRpdlvbi16eyJEYXRITGVzc1RoYW4iOnsiQVdTOKVwb2NoVGltZSI6MTc1NDAxMzAyNn0sIkIwQWRkcmVzcyl6eyJBV1M6U291cmNISXAiOilwLjAuMC4wLzAifX19XX0\\_&Signature=pFw4WUWPINVCG3dAFssVnzs3V6YZzjUM3NebTyPIUStph5OGCLq0rsPuB2F5JB40Oitsqe39N6EjUnLjamDr3mDrN~zOfptDBSiEsd1Q3GH3GyBx5yY8yAw69qf3KFtlspyLeWdBD7oekhyxDXXN18jnYwpxYIYCoiaPxjMCUIEoX27wpwtm7AlbdJX1GD5QYUpUQMgmw0rljMzeKklkAbyZ5abwv7euWBY9~6mL~iJDgEXSQPBNN396i7kpe~vK9aJxXPOP36ow80a2zZKivmrc2WPbmkNTf-ZgZFIAzZzdKMjZizBWP5Hvcpa5v6YdzRwRmTrmgVvi9sWUSSSEA\\_&Key-Pair-Id=K27MGQSHTHAGGE](https://cdn.wildapricot.com/419424/resources/Documents/Outreach/Distinguished%20lectures/MichaelRiley2014SeptSlides.pdf?version=1698754341000&Policy=eyJTdGF0ZW1lbnQiOiBbeyJSZXNvdXJjZSI6Imh0dHBzOi8vY2RuLndpbGRhcHJpY290LmNvbS80MTk0MjQvcvVzb3VyY2VzL0RvY3VtZW50cy9PdXRyZWFrjaC9EaN0aW5ndWIzaGVkJTlwbGVjdHVyZXMuTWIjaGFlbFJpbGV5MjAxNFNIcHRTbGlkZXMuGRmP3ZlcnNpb249MTY5ODc1NDM0MTAwMCIsIkNvbmRpdlvbi16eyJEYXRITGVzc1RoYW4iOnsiQVdTOKVwb2NoVGltZSI6MTc1NDAxMzAyNn0sIkIwQWRkcmVzcyl6eyJBV1M6U291cmNISXAiOilwLjAuMC4wLzAifX19XX0_&Signature=pFw4WUWPINVCG3dAFssVnzs3V6YZzjUM3NebTyPIUStph5OGCLq0rsPuB2F5JB40Oitsqe39N6EjUnLjamDr3mDrN~zOfptDBSiEsd1Q3GH3GyBx5yY8yAw69qf3KFtlspyLeWdBD7oekhyxDXXN18jnYwpxYIYCoiaPxjMCUIEoX27wpwtm7AlbdJX1GD5QYUpUQMgmw0rljMzeKklkAbyZ5abwv7euWBY9~6mL~iJDgEXSQPBNN396i7kpe~vK9aJxXPOP36ow80a2zZKivmrc2WPbmkNTf-ZgZFIAzZzdKMjZizBWP5Hvcpa5v6YdzRwRmTrmgVvi9sWUSSSEA_&Key-Pair-Id=K27MGQSHTHAGGE)
11. Experiments with Weighted Finite State Transducer (Report):  
[https://homes.esat.kuleuven.be/~spch/flavor/reports/fst\\_report.pdf](https://homes.esat.kuleuven.be/~spch/flavor/reports/fst_report.pdf)

# References

12. <https://www.openfst.org/twiki/pub/FST/FstSltTutorial/part1.pdf>
13. KaldiFST: <https://k2-fsa.github.io/kaldifst/referencens/index.html>
14. Pronunciation Lab with OpenFST:  
[https://hollingk.github.io/tutorials/JHU\\_tutorial.html](https://hollingk.github.io/tutorials/JHU_tutorial.html)