Language Understanding $\lambda(\mu)$ Laboratory

Weighted Finite State Transducers (WFST)

(Invited Lecture@University of Taxila)

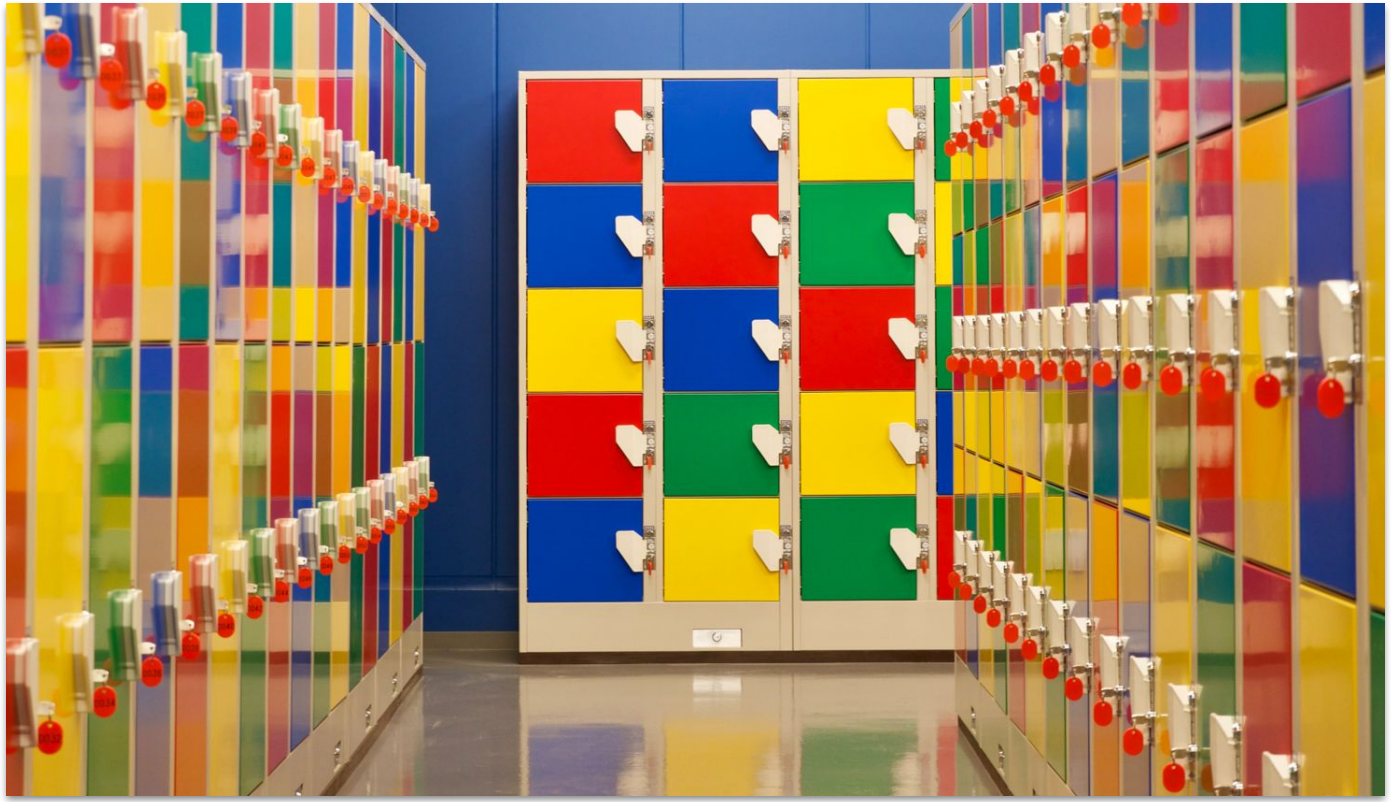
Ye Kyaw Thu
Lab Leader, LU Lab., Myanmar

30 July 2025 (SUN)

Contents

Who am I?

State Machines (Coin Locker)



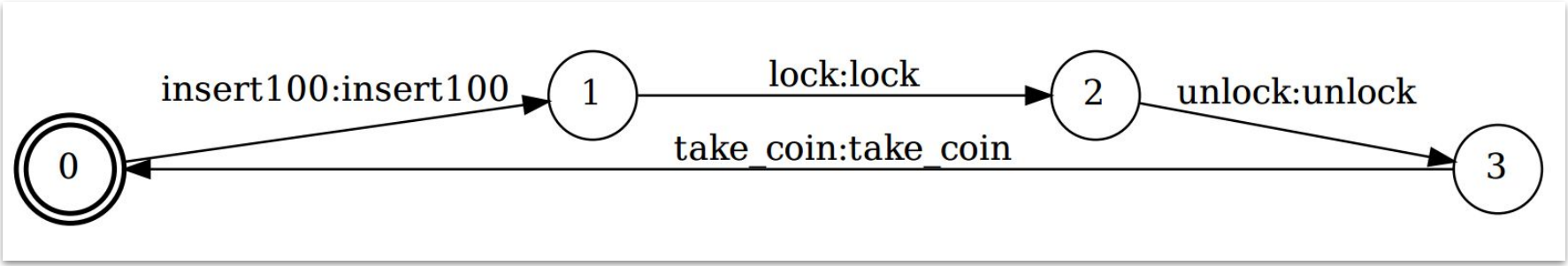
https://commons.wikimedia.org/

State Machines (Coin Locker)

https://en.qrtranslator.com/

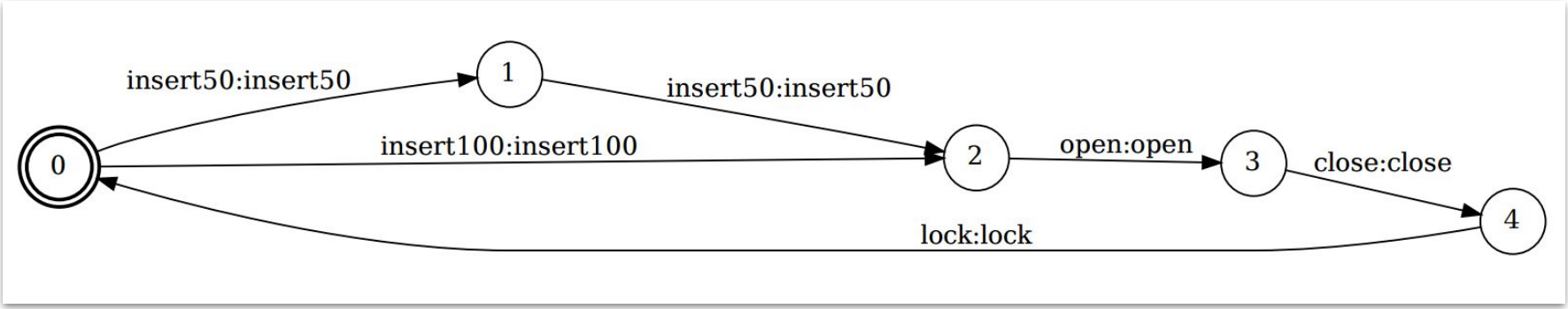
- ပစ္စည်းအပ်တဲ့အခါ
  1. တခါးကိုဆွဲဖွင့်ပြီး အပ်မယ့် ပစ္စည်းကိုထည့်ပါ
  2. အကြွေစေ့ ယန်း၁၀၀ ကိုထည့်ပါ
  3. တံခါးကို ပိတ် သော့ခတ်ပြီး၊ သော့ကို နှုတ်ယူပါ
- အပ်ထားတဲ့ ပစ္စည်းကို ပြန်ယူတဲ့အခါ
  1. သော့ဖွင့်ပြီး၊ တံခါးကိုဖွင့်ပါ
  2. ယန်း၁၀၀ အကြွေစေ့ကို ပြန်ယူပါ
  3. အပ်ထားတဲ့ ပစ္စည်းကို ပြန်ယူပြီး၊ တံခါးကို ပြန်ပိတ်ထားခဲ့ပါ

State Machines (Coin Locker)



- ဂျပန်နိုင်ငံက အကြွေစေ့ ခဏထည့်ပြီး သုံးရတဲ့ coin locker ရဲ့ အဓိကျတဲ့ အလုပ်လုပ်ပုံ အဆင့်ဆင့်ကိုပဲ finite state machine အနေနဲ့ စဉ်းစားမယ် ဆိုရင် အထက်ပါအတိုင်း graph ပုံရလိမ့်မယ်
- Programmer ကောင်းတစ်ယောက် ဖြစ်ချင်ရင် process flow ကို finite state machine အနေနဲ့ ဆွဲတတ်ရတယ် (သို့) flowchart ပေါ့

State Machines (Coin Locker)



- တကယ်လို့ ပိုက်ဆံပေးပြီးမှ သုံးရမယ် ဆိုရင် တံခါးက ပိတ်နေတဲ့ state က စမယ်
- အကြွေ ၁၀၀ ပြည့်အောင်ထည့်မှ locker တံခါးက ပွင့်မယ်
- အကြွေ ၅၀ စေ့ကိုလည်း လက်ခံတယ် (၁၀၀ ပြည့်အောင် နှစ်ခါထည့်ရမယ်)
- ကိုယ့်ရဲ့ အိတ်ဘာညာ ကိုထည့်ပြီးမှ lock လုပ်တဲ့ အထိ simulation လုပ်ကြည့်ရင် အထက်ပါ လိုမျိုး state machine ဖြစ်လိမ့်မယ်

State Machines (Temperature Controller)

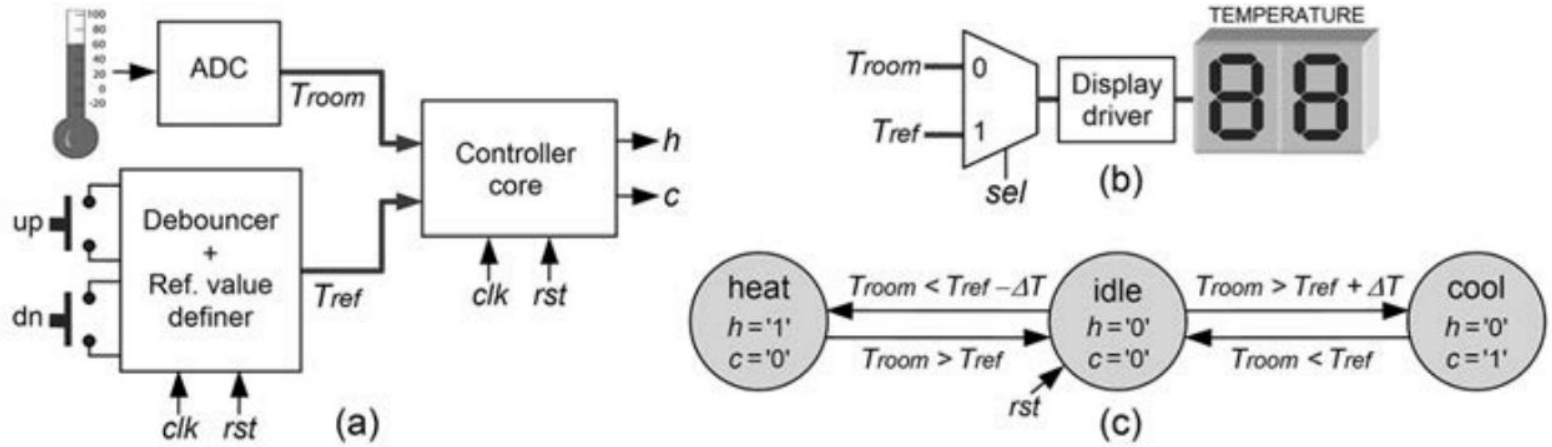


Figure 5.8

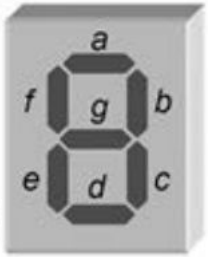
Temperature controller. (a) Overall circuit diagram. (b) Display driver. (c) State machine for the controller core block.

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

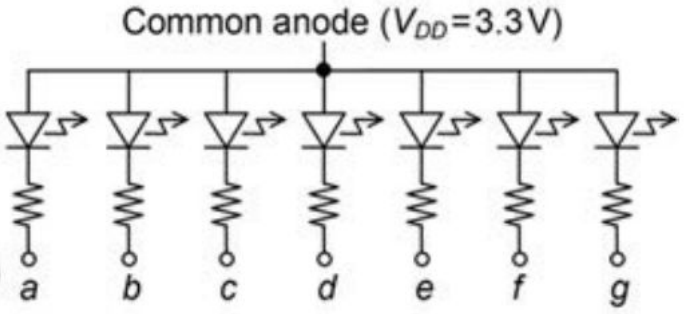
State Machines (7 Segment Display)

(a)

(b)



(c)



(d)

char	*abcdefg*	char	*abcdefg*	char	*abcdefg*	char	*abcdefg*
0	"0000001"	4	"1001100"	8	"0000000"	C	"0110001"
1	"1001111"	5	"0100100"	9	"0000100"	d	"1000010"
2	"0010010"	6	"0100000"	A	"0001000"	E	"0110000"
3	"0000110"	7	"0001111"	b	"1100000"	F	"0111000"

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 တွေကို 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

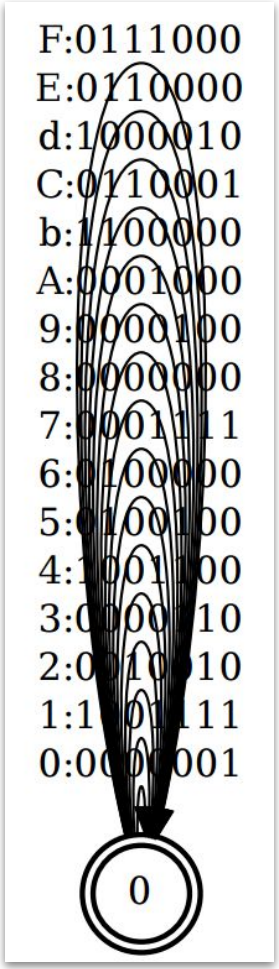


Fig.Mapping between 7 segments and characters

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**	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
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	$\Sigma^* \cup \{\infty\}$	$\wedge$	.	∞	ϵ

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:

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

- Σ: Input alphabet, Δ: Output alphabet
- $\delta : Q \times \Sigma \to Q$: State transition function
- $\sigma : Q \times \Sigma \to \Delta^*$: 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 $\mathbb{K} = (K, \oplus, \otimes, \overline{0}, \overline{1})$ is a 8-tuple:

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

where:

- $I \subseteq Q$: Initial states with weights $\lambda : I \to K$
- $F \subseteq Q$: Final states with weights $\rho : F \to K$
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times K \times Q$: 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 → Output "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 ε-transitions, unique paths).
- **Non-determinism** enables compact modeling of ambiguity but increases computational cost.

FSA Construction (Myanmar Rooster)

- ကြက်ဖတွန်သံကို FSA ဆောက်ပြီး၊ testing လုပ်ကြည့်ကြရအောင်။
- ဝီကီမီဒီယာကနေ ယူလာတဲ့ပုံ။ မန္တလေးက ကြက်ဖတဲ့…

https://commons.wikimedia.org/

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 အွတ်
4

FSA Construction (Myanmar Rooster)

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

```
<epsilon>	0
အောက်	1
အီး	2
အွတ်	3
```

FSA Construction (Myanmar Rooster)

- **correct_input.txt** ဖိုင်ထဲမှာက အောက်ပါအတိုင်း ကြက်တွန်သံကို မှန်မှန်ကန်ကန် ရိုက်ပေးထားတယ်။

အောက်
အီး
အီး
အွတ်

FSA Construction (Myanmar Rooster)

- **wrong_input.txt** ဖိုင်ထဲမှာက အောက်ပါအတိုင်း ကြက်တွန်သံကို တမင်မှားရိုက်ထားတယ်။

အောက်
အွတ်
အောက်
အွတ်

FSA Construction (Myanmar Rooster)

- compile_and_test.sh ကိုတော့ ဆာဗာထဲဝင်ပြီး 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)

```
(openfst) ye@lst-hpc3090:~/talk/wfst/r
=== Rooster FSA ===
0       1       အောက်    1
1       2       အီး      2
2       3       အီး      2
3       4       အုတ်     3
4

=== Processing correct_input.txt ===
Input FSA:
0
0 1 အောက်
1 2 အီး
2 3 အီး
3 4 အုတ်
4
```

- Run လိုက်ရင် မြင်ရမယ့် output တွေပါ

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      အုတ်     3
23  4
24  Composition result:
25  0      1      အောက်    1
26  1      2      အီး      2
27  2      3      အီး      2
28  3      4      အုတ်     3
29  4
30  Result: ACCEPTED
```

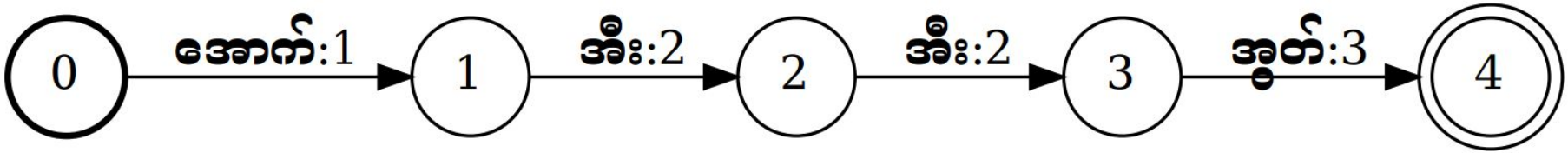
- Correct_input.txt အတွက်က Accept ဖြစ်လိမ့်မယ်
- ဆိုလိုတာက FSA ရဲ့ အဆုံး terminal node အထိ ရောက်သွားတယ်

FSA Construction (Myanmar Rooster)

```
=== Processing wrong_input.txt ===
Input FSA:
0
0 1 အောက်
1 2 အွတ်
2 3 အောက်
3 4 အွတ်
4
Compiled input FSA:
0       1       အောက်    1
0
1       2       အွတ်     3
2       3       အောက်    1
3       4       အွတ်     3
4
Composition result:
Result: REJECTED
```

- wrong_input.txt အတွက်တော့ Reject ပါ
- ဆိုလိုတာက pass မဖြစ်ဘူး

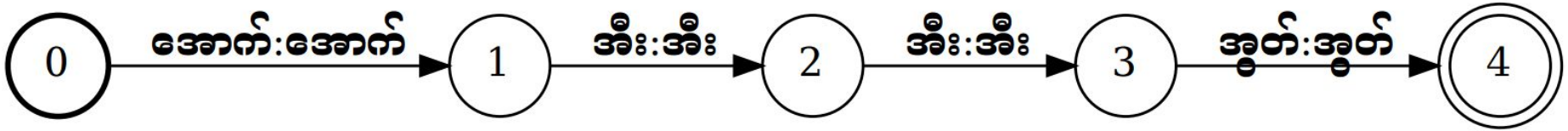
FSA Construction (Myanmar Rooster)



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

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

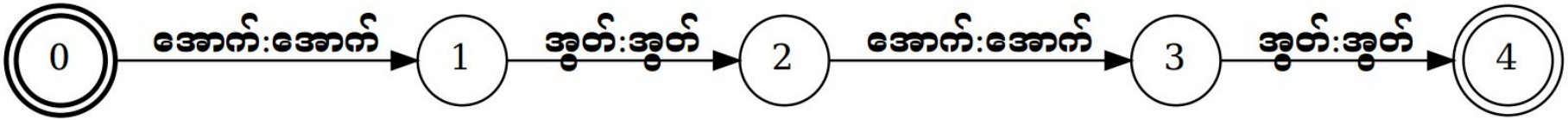
FSA Construction (Myanmar Rooster)



- --osymbols option ပါ ပေးရင် အထက်ပါအတိုင်း output ပြောင်းသွားလိမ့်မယ်။ FSA မို့လို့ output မရှိလို့ 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 သို့မဟုတ် ပုံဖိုင် ထုတ်ကြည့်ရအောင်

```
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ fstdraw --portrait --isymbols=syllable.syms --osymbols=syllable.syms wrong.b
in > 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 ထုတ်ကြည့်တဲ့အခါမှာ မထုတ်ပေးနိုင်တာကို အောက်ပါအတိုင်း တွေ့ရပါလိမ့်မယ်

```
(openfst) ye@lst-hpc3090:~/talk/wfst/rooster_fsa$ fstdraw --portrait --isymbols=syllable.syms --osymbols=syllable.syms wrong.r
esult.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)

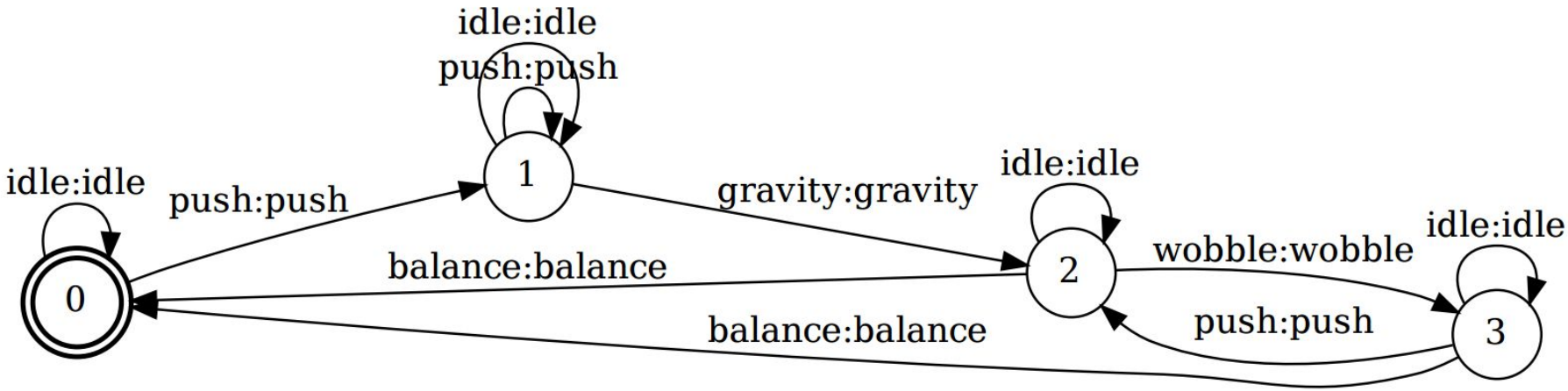
```
(openfst) ye@lst-hpc3090:~/talk/wfst/rolling_3$ cat pyit_syms.txt
<eps> 0
push 1
gravity 2
wobble 3
balance 4
idle 5

(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"
```

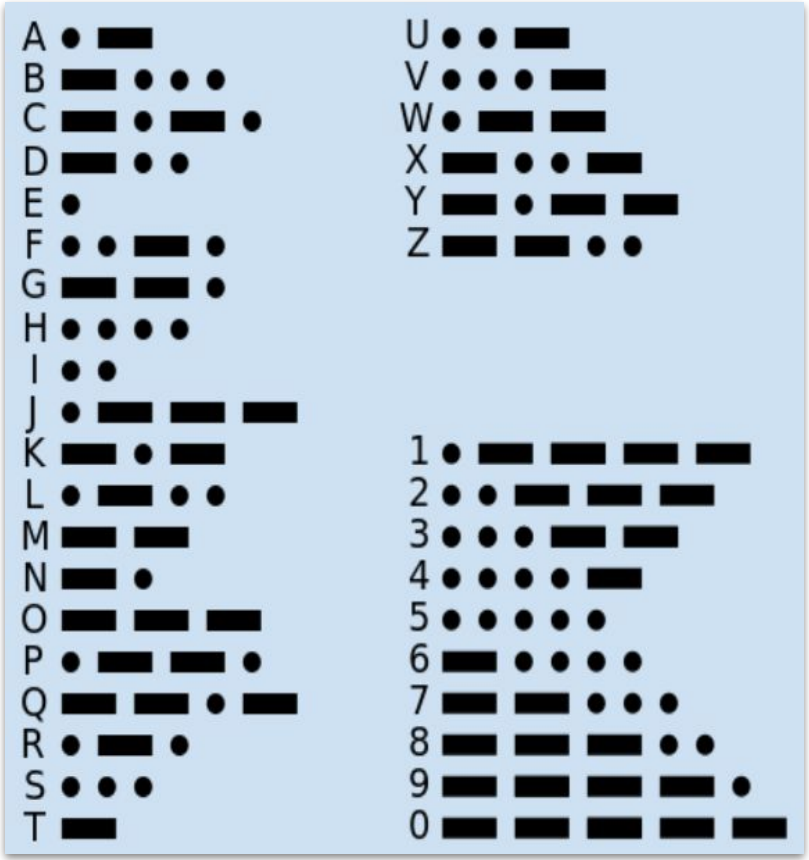
Fig. simulate.sh

FSA Construction (Pyit Taing Htaung Simulation)



- ပစ်တိုင်းထောင် အရုပ်ကို simulation အကြမ်း လုပ်ကြည့်တာပါ
- ဒီမှာ ပြောနေအတဲ့ idle ဆိုတာက လက်ရှိ ရောက်နေတဲ့ state မှာပဲ ပုံမှန်ဆက်သွားနေတာကို ဆိုလိုတာပါ။ ရှုပဇေဒ အနေနဲ့ ကြည့်ရင်တော့ ပြောစရာတွေ ရှိပါလိမ့်မယ်…

FST Construction (Morse encoder/decoder)



- Rules/heuristic process တွေအတွက် WFST က တအားကောင်းတယ်
- ဥပမာ အနေနဲ့ အင်္ဂလိပ်စာကနေ မောစ်ကုဒ် (i.e. encoding)၊ မောစ်ကုဒ် ကနေ အင်္ဂလိပ်စာ (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      O
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
```

- အင်္ဂလိပ်စာအတွက် symbol ဖိုင်

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
```

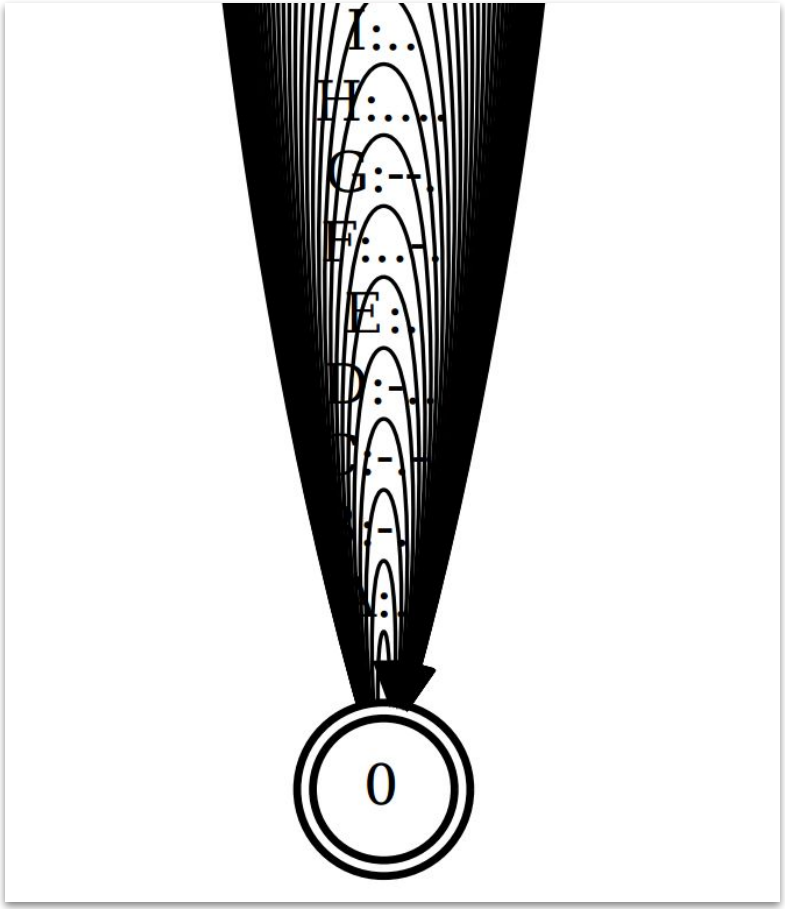
- မောစ်ကုဒ်အတွက် symbol ဖိုင်

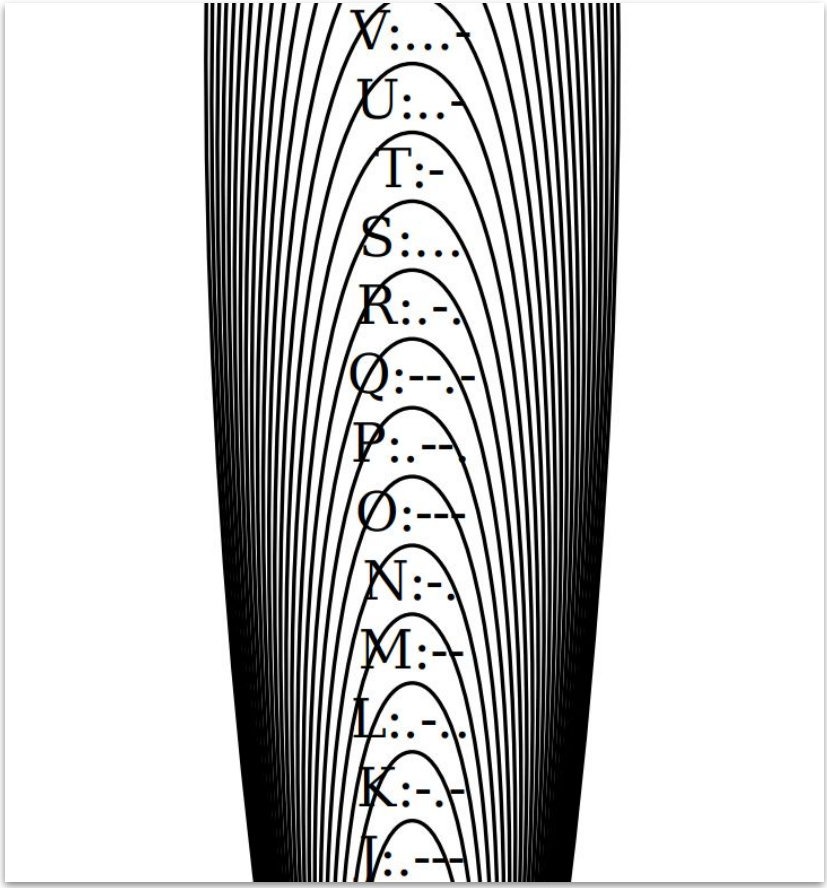
FST Construction (Morse encoder/decoder)

```
(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
```

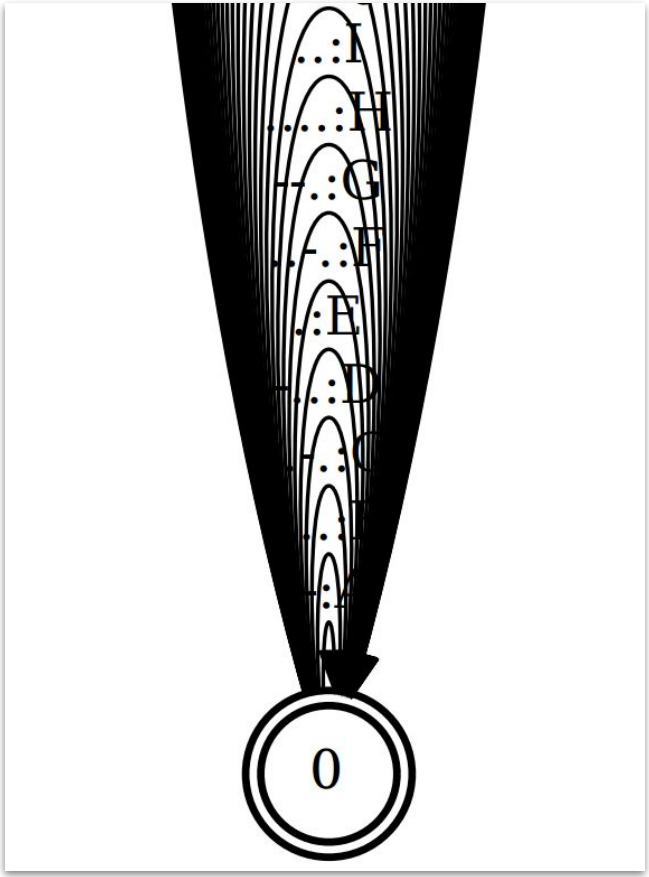
- မောစ်ကုဒ်အတွက် symbol ဖိုင်

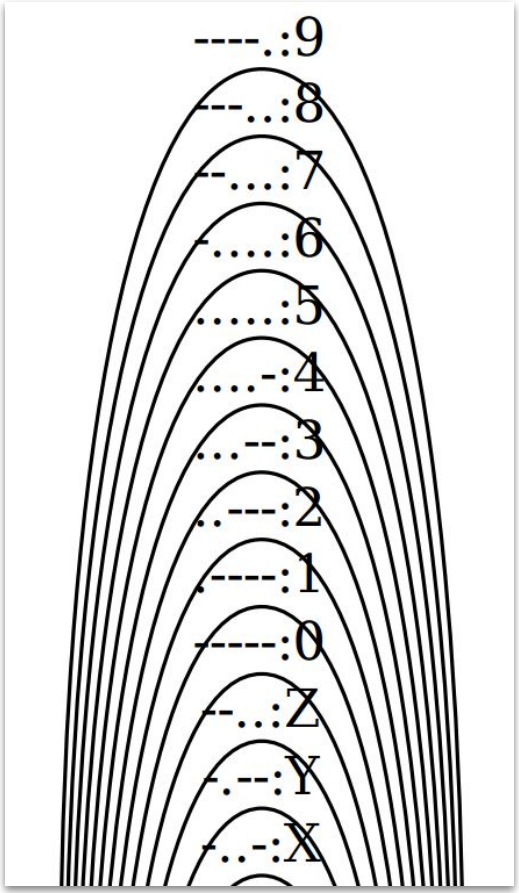
FST Construction (en2morse)





FST Construction (morse2en)





- Lexicon ကို နှစ်ခုဆောက်ထားဖို့ လိုအပ်
- 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 ဆိုတဲ့ command ကိုသုံးပြီး en2morse ကနေ morse2en အဖြစ် ပြောင်းလိုက်တာပါ

OpenFST Operations

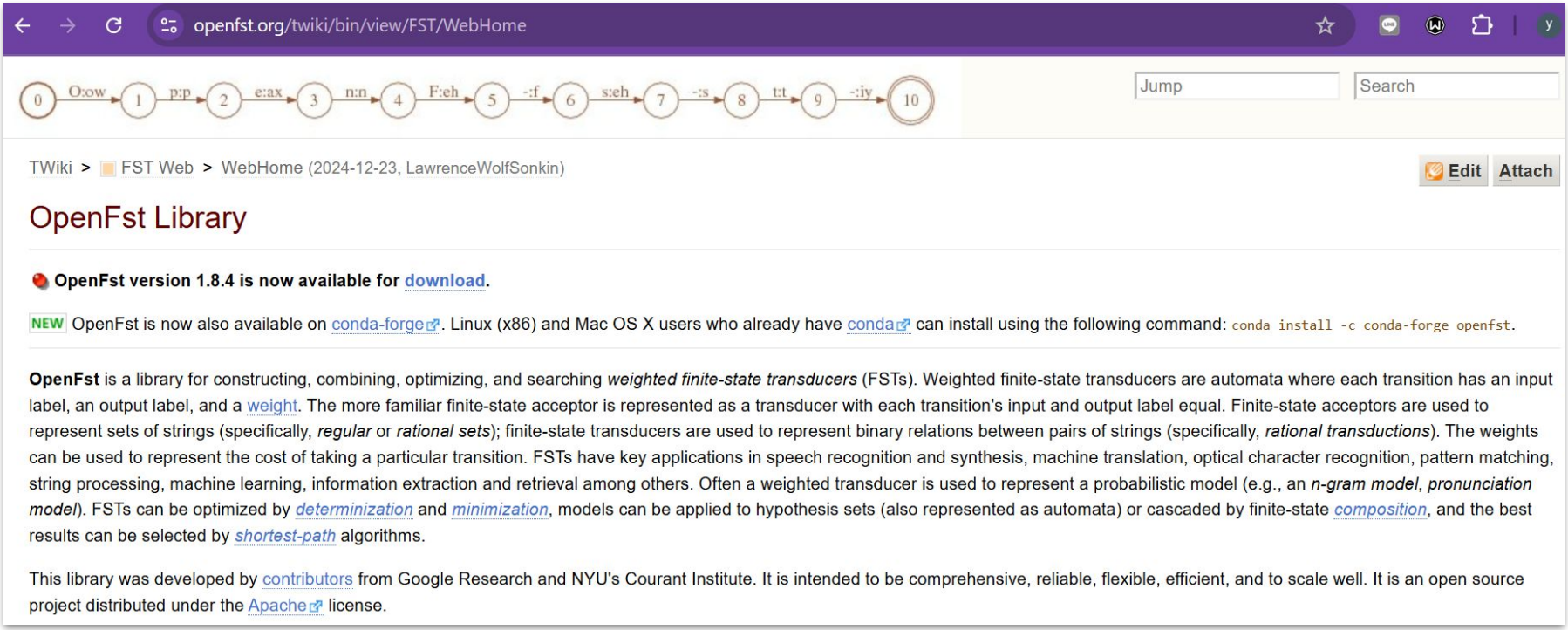


Fig. Homepage of OpenFST

OpenFST Operations

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             fstrandgen            fstspecial
fstcompose            fstencode             fstloglinearapply     fstrelabel            fstsymbols
fstcompress           fstepsnormalize       fstmap                fstreplace            fstsynchronize
fstconcat             fstequal              fstminimize           fstreverse            fsttopsort
fstconnect            fstequivalent         fstopgm               fstreweight           fstunion
fstconvert            fstinfo               fstprint              fstrim
```

- Installation လုပ်လို့ အဆင်ပြေရင် အောက်ပါလိုမျိုး command တွေကို သုံးလို့ ရပါလိမ့်မယ်
- 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

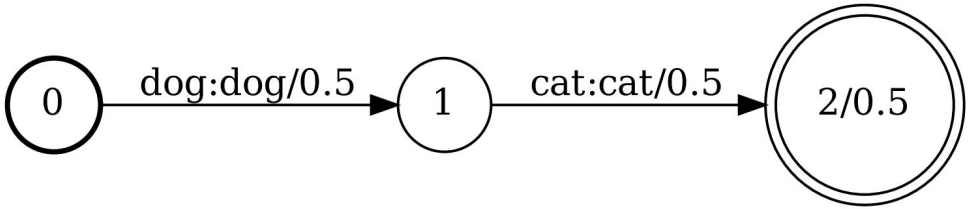
```
0 1 bird 0.7
1 0.7
```

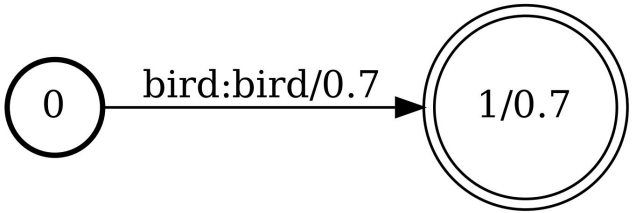
- syms.txt

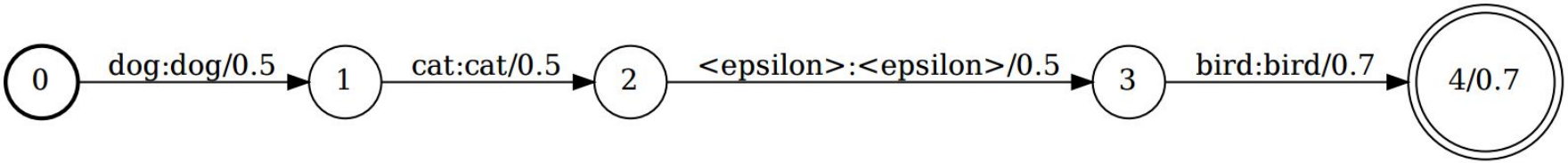
```
<epsilon> 0
dog 1
cat 2
bird 3
```

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

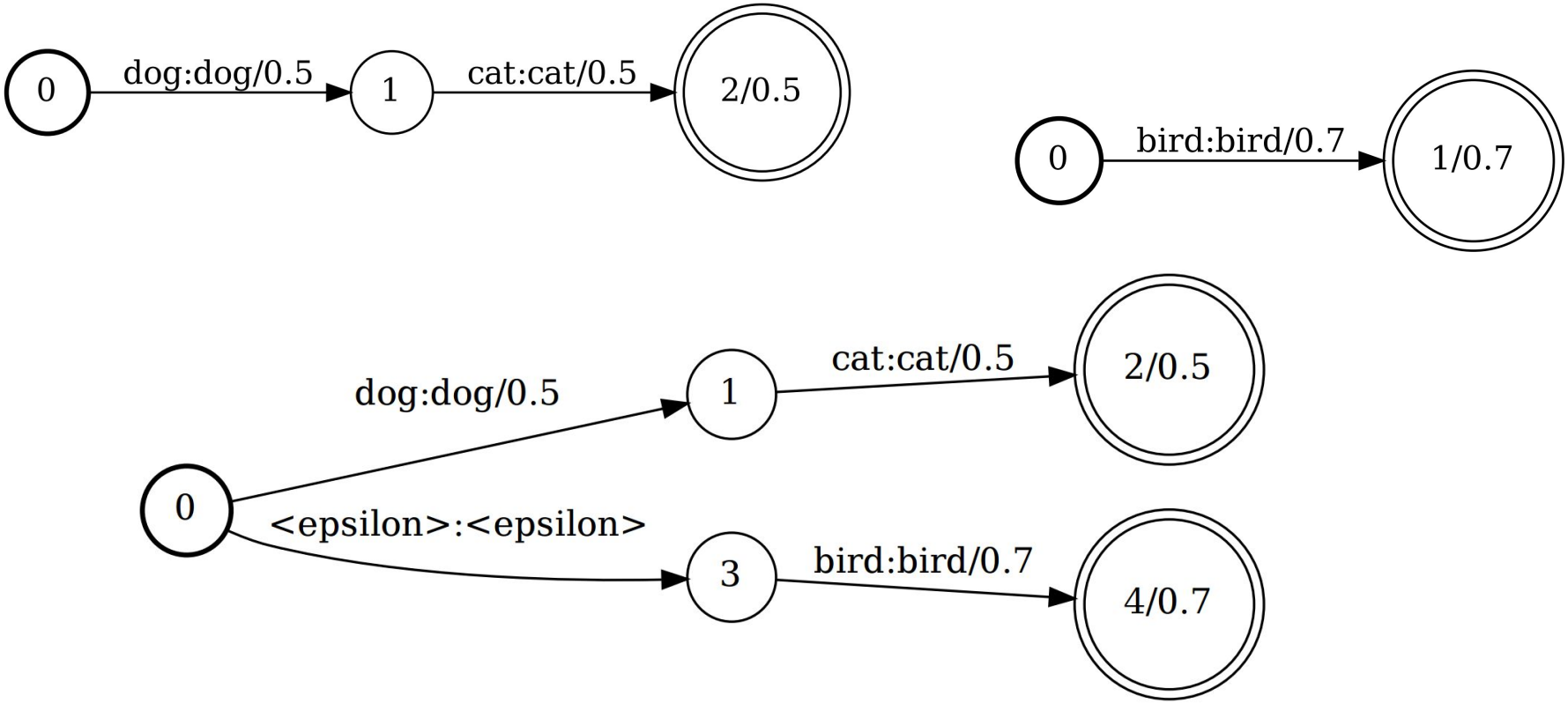
OpenFST Operations (Concat = Product)







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

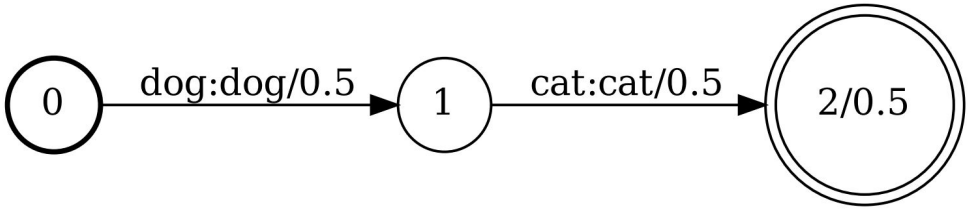
For an FST A, its closure A^* satisfies:

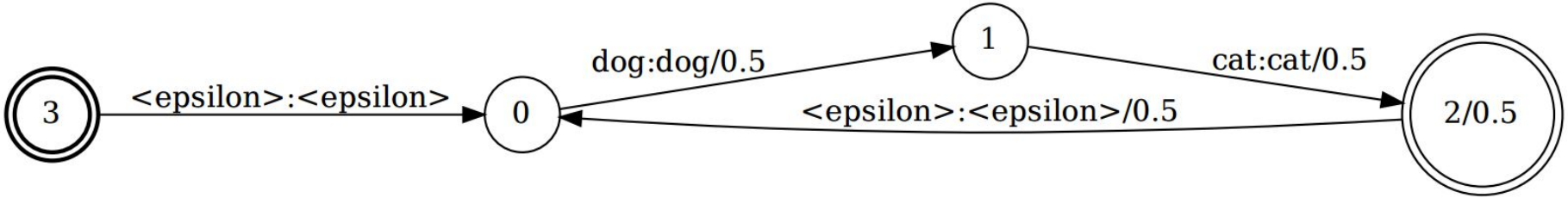
$$A^* = \{\epsilon\} \cup A \cup AA \cup AAA \cup \cdots$$

where:

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

OpenFST Operations (Closure)





OpenFST Operations (Closure)

- Now accept:
- ε (empty string)
- "dog cat"
- "dog cat dog cat"
- "dog cat dog cat dog cat"
- etc.

Spoken Dialect Translation

IEEE Access
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

Transfer and Triangulation Pivot Translation Approaches for Burmese Dialects

THAZIN MYINT OO[1], THITIPONG TANPRASERT[1], YE KYAW THU[2], AND THEPCHAI SUPNITHI[2]

[1]Vincent Mary School of Science and Technology, Assumption University of Thailand, Bangkok 10240, Thailand
[2]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

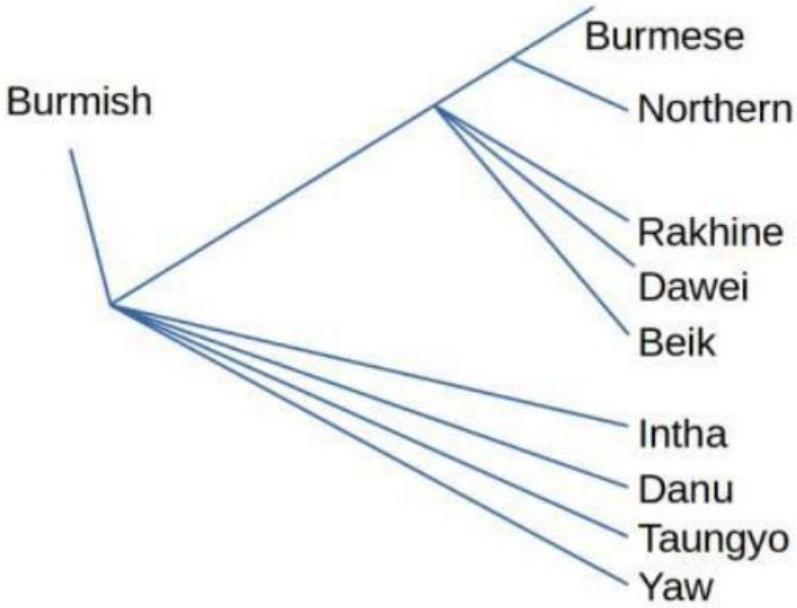
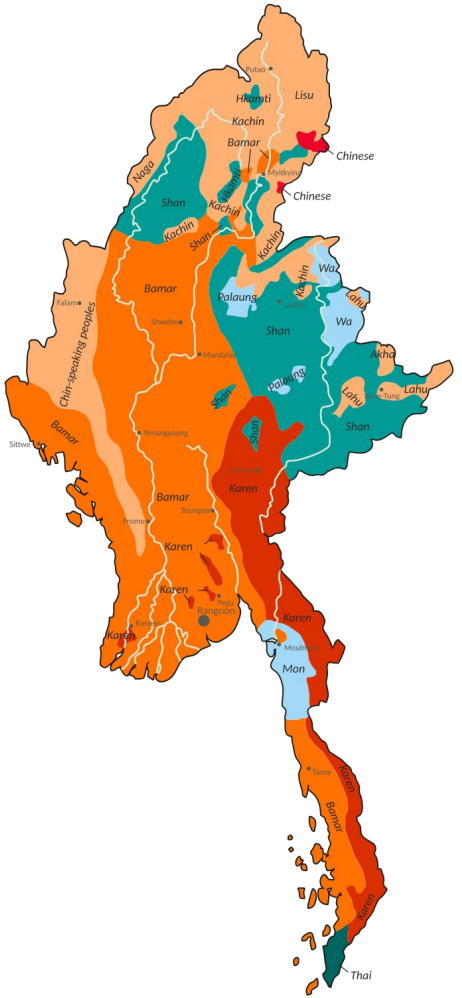


FIGURE 1. Classification of burmese (adapted from Bradley 1997: 38-39 [20]).



Spoken Dialect Translation

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

dw: သယ်ဝယ်သား က လှ ဖြင်း ဟှယ် ။

bm: ဒီကောင်မလေး က လှ လွန်း တယ် ။

(“The girl is so beautiful” in English)

dw: လတ်ဖတ်ရယ် က ရှိ ဖြင်း ဟှယ် ။

bm: လက်ဖက်ရည် ချို လွန်း တယ် ။

(“The tea is so sweet” in English)

dw: ကောန်သား ကျောန်း မှန်းမှန် သွား ဟှယ် ။

bm: ကောင်လေး ကျောင်း မှန်မှန် တက် တယ် ။

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

Spoken Dialect Translation

bk: မင်း ငါ့ ကို ကြေးပြား ပေး ဝို့ မေ့ နေရယ် လား ။

bm: မင်း ငါ့ ကို ပိုက်ဆံ ပေး ဖို့ မေ့ နေပြီလား ။

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

bk: ငါ မောလင်း နိုင်ငံခြား သော မယ်။

bm: ကျွန်တော် မနက်ဖြန် နိုင်ငံခြား သွား မယ် ။

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

bk: ကျွန်တော် ဒယ် ဝို လာ ရဇာ ပျော် ရယ် ။

bm: ကျွန်တော် ဒီ လာ ရတာ 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: ဒယော တစ် ထည် ဇာလောက်လေး ။

bm: လုံချည် တစ် ထည် ဘယ်လောက်လဲ ။

(“How much for a longyi?” in English)

rk: ကလေချေ တိ ဘောလုံး ကျောက် နီရေ ။

bm: ကောင်လေး တွေ ဘောလုံး ကန် နေတယ် ။

(“Boys are playing football” in English)

rk: ဇာ ပြော နီချင့် ယင်းသူရို့ ။

bm: သူတို့ ဘာ ပြော နေတာလဲ ။

(“What are they talking about” in English)

rk: အဘောင်သျင် စျီး က သပုံ ဝယ် လာတယ် ။

bm: အဘွား စျေး က 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 express 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

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

Spoken Dialect Translation

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

$$P(r_t \mid r_{t-1}) = (1-\alpha)\, P_{ML}(r_t \mid r_{t-1}) + \alpha P_{ML}(r_t)$$

$$\begin{aligned} P(r_t = \text{ခြိန်} \mid r_{t-1} = \text{ဇာ}) &= (1-\alpha)\, P_{ML}(r_t = \text{ခြိန်} \mid r_{t-1} = \text{ဇာ}) \\ &\quad + \alpha \mathrm{P_{ML}}(r_t = \text{ခြိန်}) \\ &= 0.9\frac{1}{3} + 0.1\frac{1}{12} \\ &= 0.308\overline{3} \end{aligned} \tag{1}$$

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 "ဏာ" (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: ဏာ ချိန် လေး

Rakhine Sentence 2: ဏာ ပြော လေး

Rakhine Sentence 3: ဏာ လုပ် လေး

Spoken Dialect Translation

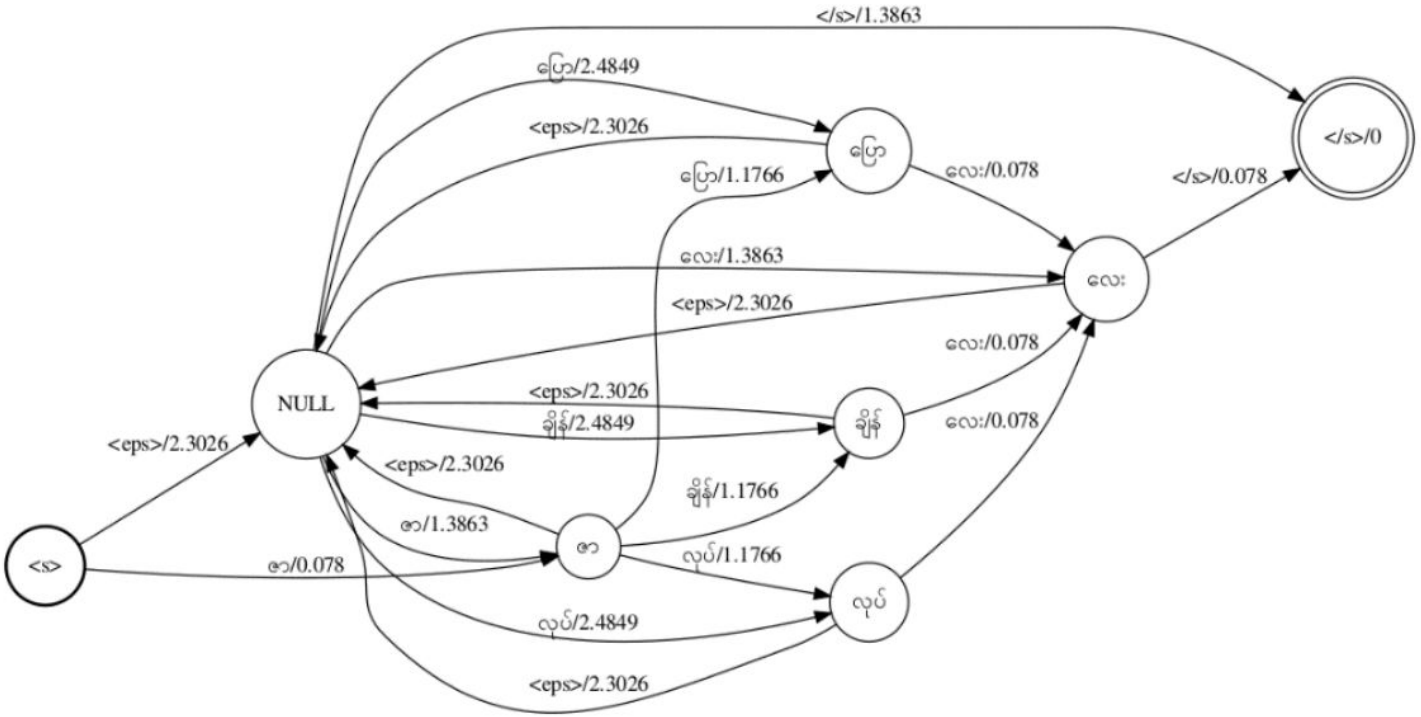


FIGURE 2. An example of WFSA representing a 2-gram language model of three Rakhine language sentences.

Spoken Dialect Translation

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

$$P\left(r = \text{ချိန်} \mid b = \text{အချိန်}\right) = 1$$
$$P\left(r = \text{သလဲ} \mid b = \text{လေး}\right) = 0.66\overline{6}$$
$$P\left(r = \text{ဘယ်} \mid b = \text{ဇာ}\right) = 0.33\overline{3}$$

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

FIGURE 3. An example of word-to-word transducer where $P(R \mid B) = \prod_{t=1}^{|B|} P(r_t \mid b_t)$.

Spoken Dialect Translation

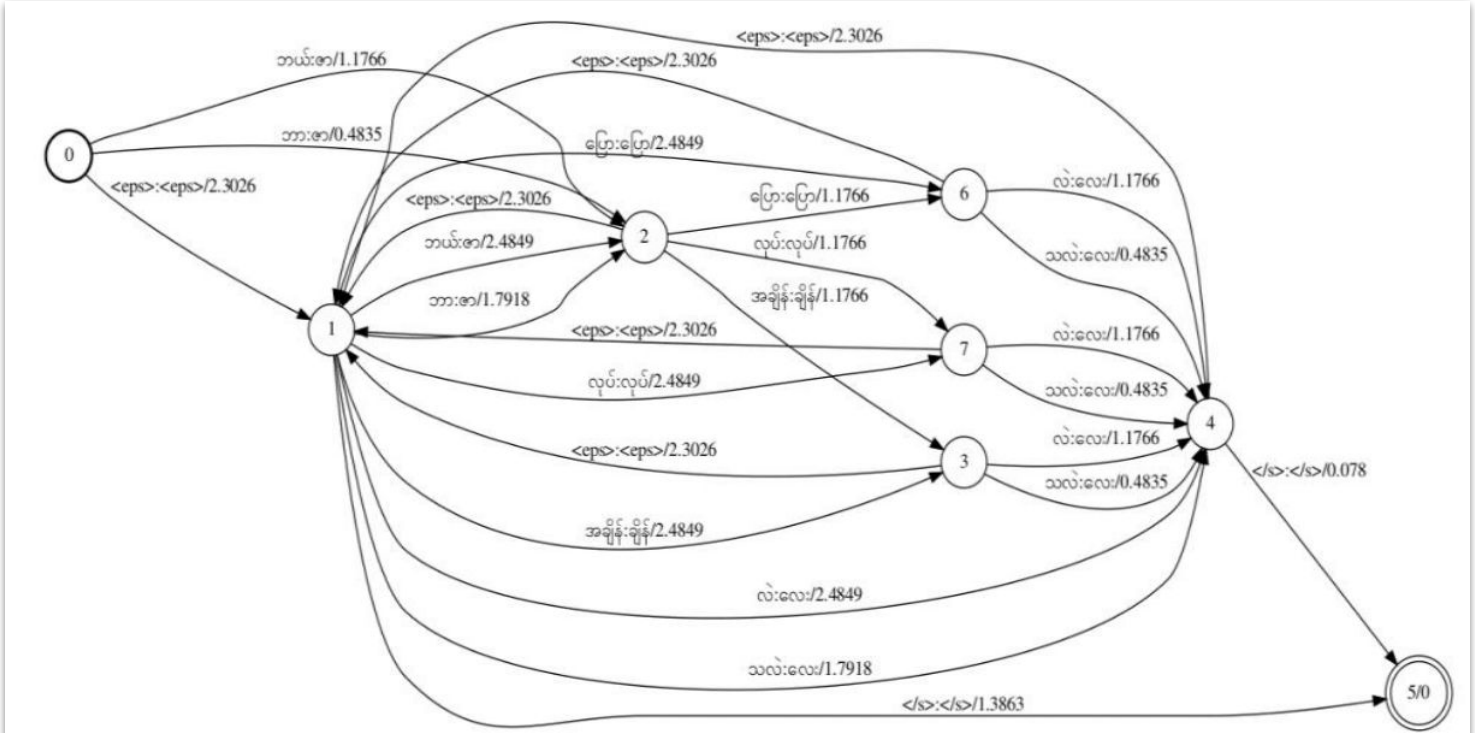
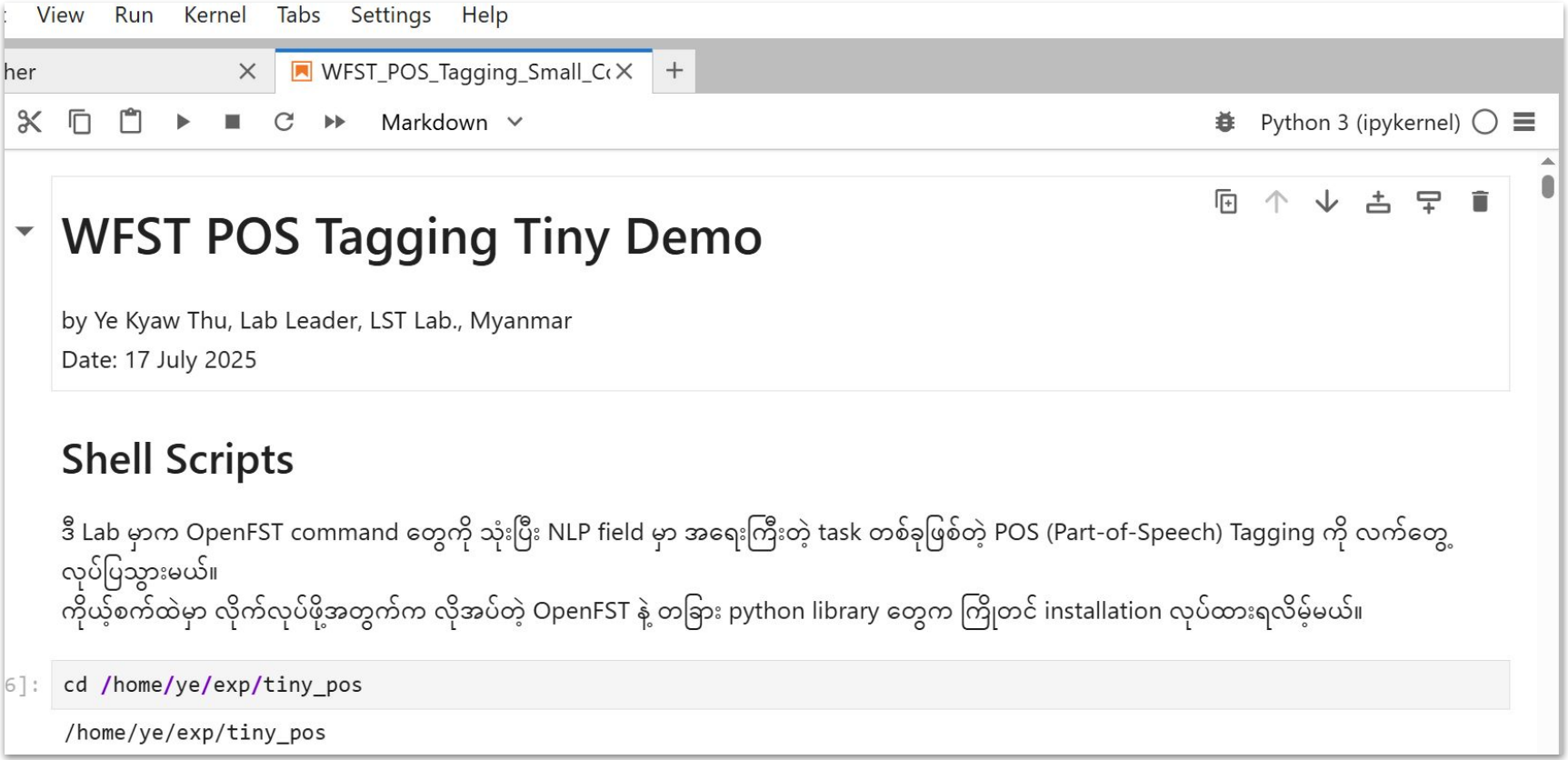


FIGURE 4. An example of a WFST representing the negative log probability of R given B ($TP(R \mid B)$), created by composing Figure 3 (WFST of $P(R|B)$) and Figure 2 (WFSA of $P(R)$).

WFST POS Tagging Demo



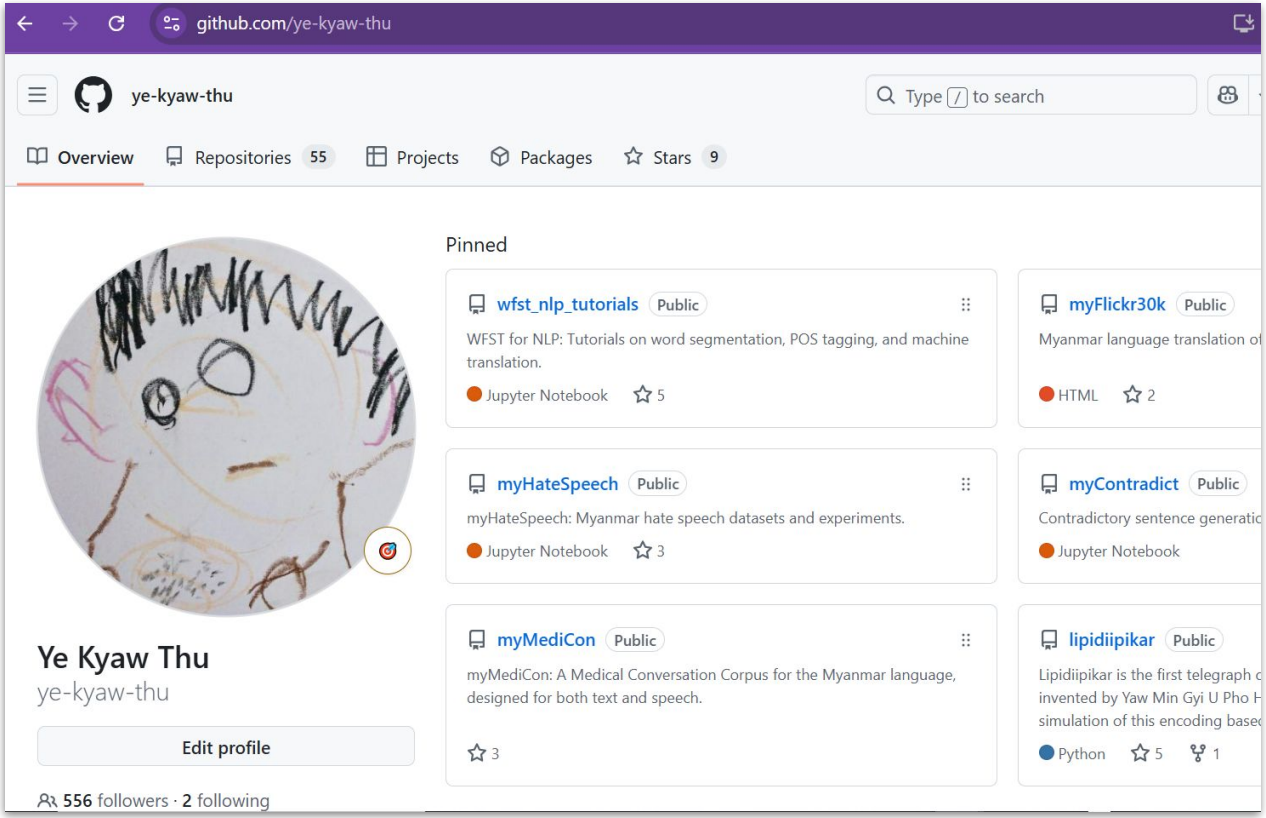
Best Practices

- **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

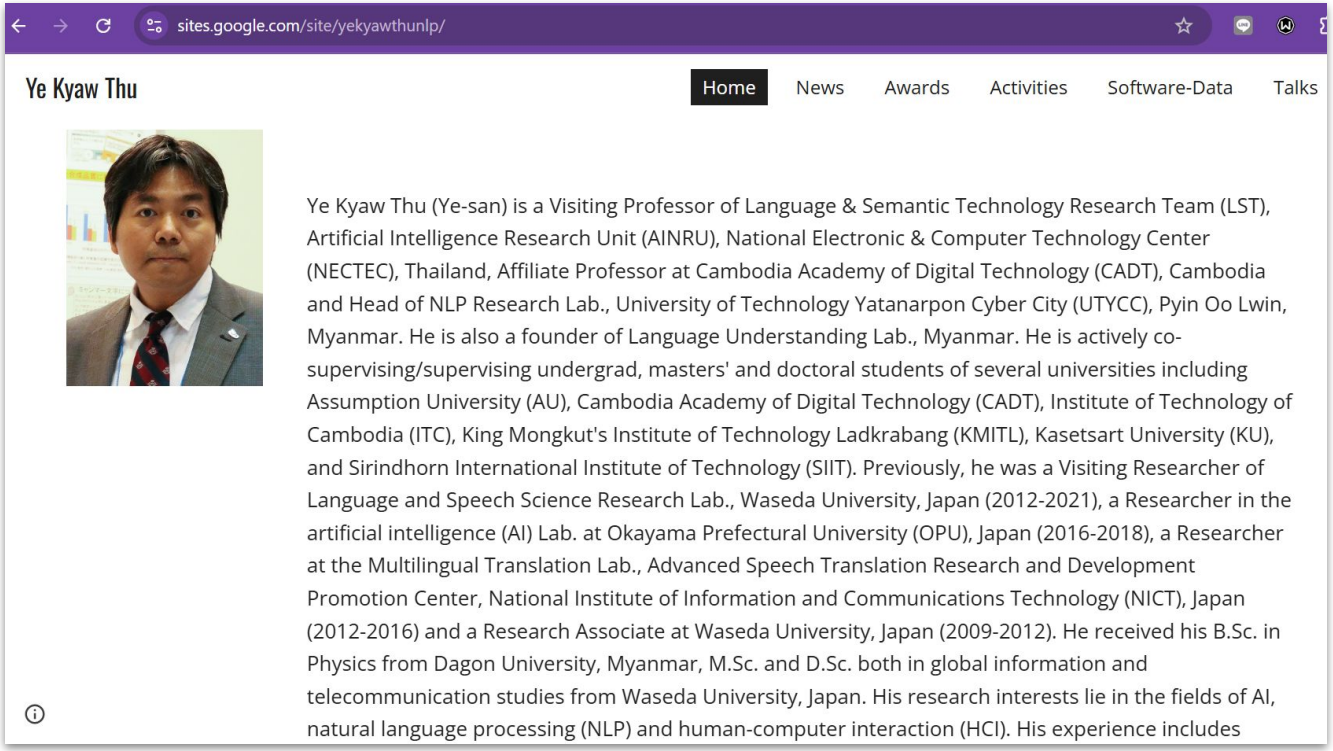
- FSA, FST, WFST ကို မိတ်ဆက်ပေးခဲ့တယ်
- အခြေခံ သီအိုရီရော၊ လက်တွေ့ပိုင်းတွေရော
- OpenFST framework ကိုသုံးပြခဲ့တယ်
- WFST ကိုသုံးပြီး spoken dialect language pair တွေအကြား (ဥပမာ ဘိတ်-ထားဝယ်၊ ဗမာ-ရခိုင်၊ ဗမာ-ပအိုဝ်) လုပ်လို့ ရကြောင်း သင်ကြားပေးခဲ့
- POS tagging ကိုလည်း WFST ကိုအခြေခံပြီး လုပ်လို့ရကြောင်း လက်တွေ့ လုပ်ပြခဲ့
- သုံးလို့ရတဲ့ နေရာတွေက အများကြီးပါ (e.g. Automatic Speech Recognition, Text to Speech, Machine Translation, Parsing etc.)

Conclusion



- လုပ်ခဲ့တဲ့အလုပ်တော်တော်များများကို github မှာ

Conclusion



- 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
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

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=eyJTdGF0ZW1lbnQiOiBbeyJSZXNvdXJjZSI6Imh0dHBzOi8vY2RuLndpbGRhcHJpY290LmNvbS80MTk0MjQvcmVzb3VyY2VzL0RvY3VtZW50cy9PdXRyZWFjaC9EaXN0aW5ndWlzaGVkJTIwbGVjdHVyZXMvTWljaGFlbFJpbGV5MjAxNFNlcHRTbGlkZXMucGRmP3ZlcnNpb249MTY5ODc1NDM0MTAwMCIsIkNvbmRpdGlvbiI6eyJEYXRlTGVzc1RoYW4iOnsiQVdTOkVwb2NoVGltZSI6MTc1NDAxMzAyNn0sIklwQWRkcmVzcyI6eyJBV1M6U291cmNlSXAiOiIwLjAuMC4wLzAifX19XX0_&Signature=pFw4WUWPINVCG3dAFssVnzr3V6YZzjUM3NebTyPIUStph5OGCLq0rsPuB2F5JB40Oitsqe39N6EjUnLjamDr3mDrN~zOfptDBSiEsd1Q3GH3GyBx5yY8yAw69qf3KFtlspyLeWdBD7oekhyxDXXN18jnYwvpxYlYCoiaPxjMCUIEoX27wpwtnm7AlbdJX1GD5QYUpUQMgmw0rljMzeKklkAbyZ5abwv7euWBY9~6mL~iJDgEXSQPBN396i7kpe~vK9aJxXPOP36ow80a2zZKivmrc2WPbmkNTf-ZgZFlAzZzdKMjZizBWP5Hvcpa5v6YdzRwRmTrmgVvi9sWUSSSEA__&Key-Pair-Id=K27MGQSHTHAGGF
11. Experiments with Weighted Finite State Transducer (Repot): 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