OVERVIEW

1. Description of the Finite-State Machine (FSM) Toolkit

2. Applications and Examples

3. Rules of Thumb
Part I. Finite-State Machine (FSM) Toolkit

The FSM tools construct, combine, minimize, and search weighted finite-state machines (FSMs).

User-Program Level: Programs that read from and write to files or pipelines. \(fsm(I)\):.

Library Level: Header archives or C++ functions can also be obtained. \(fsm(2)\):.

Definition Level: Specification of labels, costs, and of kinds of FSM representations. \(fsm(out\, \text{fsm, init}, \text{fsm})\):

FSMdump("out\, \text{fsm, init}, \text{fsm})

FSM init = FSMload("init\, \text{fsm})

FSM in2 = FSMload("in2\, \text{fsm})

FSM in3 = FSMload("in3\, \text{fsm})

The FSM tools construct, combine, minimize, and search weighted finite-state machines (FSMs).
FSM FILE TYPES

- **Binary Format**
  - Compiled representation used by all FSM
  - Symbols, Files
  - Transducer Files
  - Acceptor Files

- **Textual Format**
  Used for manual inputting and viewing of

utilites.
Graphical Representation (A. P's):

<table>
<thead>
<tr>
<th>FIN CST</th>
<th>PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.8</td>
<td>2</td>
</tr>
<tr>
<td>6.6</td>
<td>1</td>
</tr>
<tr>
<td>3.3</td>
<td>1</td>
</tr>
<tr>
<td>2.0</td>
<td>0</td>
</tr>
<tr>
<td>1.5</td>
<td>0</td>
</tr>
</tbody>
</table>

Textual Representation (A. text):

Acceptor Piles
Gr aphical R epresentation (T.ps):

<table>
<thead>
<tr>
<th>FIN</th>
<th>CST</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4.8</td>
</tr>
<tr>
<td>4</td>
<td>6.2</td>
</tr>
<tr>
<td>6</td>
<td>1.4</td>
</tr>
<tr>
<td>1</td>
<td>2.2</td>
</tr>
<tr>
<td>0</td>
<td>3.3</td>
</tr>
<tr>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Textual Representation (T.txt):

Transducer File
Graphical Representation (A. sps):

Symbols File (A. sym):

Acceptor File (A. stxt):
Graphical Representation (1 sps):

Symbols File (1 sym):

Transmission File (1 txt):
Symbols

Compiling, Printing, and Drawing FSMs with
Graphical Representation:

Program: ftsa B.fts ftsa A.fts

(Eqation: \( B + A = C \)  \( B \land A = C \))

Union (Sum)
Graphical Representation:

Program: fsmconcat A.fsa B.fsa > C.fsa

Equation: C = AB

Concatenation (Product)
(Concatenative) Closure

Equation: \( C = B_* = B^0 + B^1 + B^2 \ldots \)

Program: 
```plaintext
fsmclosure B.fsa > C.fsa
```

Graphical Representation:
Graphical Representation: •

Program: freverse A.fsa > C.fsa •

Equation: \( C = A^r \) •

Reversal
Inversion

Program: fsminvert A.fst > C.fst

Graphical Representation:

Equation: $C = A^{-1}$
A.fsa

\[ I \xrightarrow{\text{yellow}/0.6} 0 \]

\[ I \xrightarrow{\text{green}/0.3} 0 \]

\[ I \xleftarrow{\text{red}/0.5} 0 \]

\[ 0 \xrightarrow{\text{red}/0.5} I \]

Graphical Representation:

Program: if project -1 T test >A.fsa

Equation: \( A = L^T \)
Graphical Representation:

Program: FSM\rightarrow \text{ATSa} \rightarrow C\cdot \text{FTA}

Epsilon Removal
Graphical Representation:

Program: fswm intersection A. fsa B. fsa > C. fsa

Equation: $C = A \cup B$

Intersection (Hademard Product)
Graphical Representation:

Program: fswdifforencet A, FSA B, FSA C, FSA

Equation: C = A - B

Difference
Composition
Graphical Representation:

Program: From nont FA to FA

Connection (Trimming)
Graphical Representation:

Program: \texttt{DFA \rightarrow \text{Determinization}}

\texttt{DFA}

\texttt{AFA}

\texttt{BFA}

\texttt{CFA}

\texttt{DFA}
Graphical Representation:

Program: FSMs > Minimization
Graphical Representation:  
Program: tspbestpath [-n N] A.fsa > C.fsa 
Best Path(s)
Graphical Representation:

Random Paths(s)
Graphical Representation:

Pruning
<table>
<thead>
<tr>
<th>Format</th>
<th>Flag</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSM_Packet</td>
<td>p</td>
<td>p.s</td>
</tr>
<tr>
<td>FSM_Constitute</td>
<td>c0</td>
<td>c0.s</td>
</tr>
<tr>
<td>FSM_Constitute</td>
<td>c1</td>
<td>c1.s</td>
</tr>
<tr>
<td>FSM_Constitute</td>
<td>c2</td>
<td>c2.s</td>
</tr>
<tr>
<td>FSM_Constitute</td>
<td>0</td>
<td>0.s</td>
</tr>
<tr>
<td>FSM_Input</td>
<td>1</td>
<td>1.s</td>
</tr>
<tr>
<td>FSM_ISO</td>
<td>1</td>
<td>1.s</td>
</tr>
<tr>
<td>FSM_ISO</td>
<td>q</td>
<td>q.s</td>
</tr>
<tr>
<td>FSM_ISO</td>
<td>q</td>
<td>q.s</td>
</tr>
<tr>
<td>FSM_ISO</td>
<td>r</td>
<td>r.s</td>
</tr>
</tbody>
</table>

Program:

FSM Format Conversion
Graphical Representation:

Program: Fm transform A. Fsa > C. Fsa

Topological Sort
Graphical Representation:

- # of states: 5
- Class: fsm
- File: A.fsa

Program:

FSM Information
FSM Information (numeric)
<table>
<thead>
<tr>
<th>Final cost</th>
<th>arc cost</th>
<th>multiplicity</th>
<th>label</th>
<th>out-deg</th>
<th>in-deg</th>
<th>quantities</th>
<th>FSM Information (distributions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Program: FSM Information (distributions)</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0.8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>0.5</td>
<td>0.3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.25</td>
<td>0.5</td>
<td>0.75</td>
<td></td>
</tr>
</tbody>
</table>
det
e
non-
costs
costless
no

sorted w\text{rt} \epsilon
top sorted

sorted

acyclic w\text{rt} \epsilon
acyclic w\text{rt} start state
acyclic
connected
arc cost sorted

FSM Information (Properties)
6. Sentence Generation (with pronunciations)
5. Context-dependency in ASR
4. Language normalization
3. String alignment (matching with errors)
2. String matching
1. Keyword detection/recognition

Part 2 - Applications and Examples
FSTs in Speech and Language Processing

M. Mohri, P. Pereira, M. Riley

\begin{verbatim}
else return 0;
else return 1;
if (strcmp(token, "string") == 0) return 1;
else return 1;
if (strcmp(token, "short") == 0) return 1;
else return 1;
if (strcmp(token, "else") == 0) return 1;
if (strcmp(token, "continue") == 0) return 1;
if (strcmp(token, "char") == 0) return 1;
else return 0;

Brute-force search:
\begin{verbatim}
{char, const, continue, if, int, else, short, signed, sizeof}
\end{verbatim}

C Identities:

Keyword Detection
search(token, keywords, NKES, sizeof(char *) keycmp);

\{ (void **) char * (char * x, const void * const void *)

int keycmp(const void * const void * const void * (\x27\x27"

"sizeof"

sizeof"

"short"

"int"

"if"

"else"

"continue"

"const"

"char"

\} = [NKES] keywords char

#define NKES 9

Keyword Detection - Tabular Search
Keyword Detection - Automata Search
Keyword Detection  –  Deterministic Search
Search

Keyword Detection – Minimal Deterministic
Keyword Recognition – Automata Search
Keyword Recognition - Deterministic Search
Search
Keyword Recognition – Minimal Deterministic
Graphical Representation:

\[ A \times \mathcal{L} = \mathcal{W} \]

String matching
String Alignment

Bases form Phone Word
\[ d(a^i, b^j) = \min \{ ds(a^i, b^j), d_d(a^i, b^j) + w(a^i, e), d_i(a^i, b^j) + w(e, b^j) \} \]

**Minimum string edit distance (prefixes):**

<table>
<thead>
<tr>
<th>BASEFORM</th>
<th>PHONE</th>
<th>WEIGHTS</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_i</td>
<td>d</td>
<td>e</td>
<td>1</td>
</tr>
<tr>
<td>b_j</td>
<td>pr</td>
<td>w(a_i, b_j)</td>
<td>insertion</td>
</tr>
</tbody>
</table>

Symbol edit weights (task-specific):

- **ae**
- **eh**
- **pr**

**Weighted Edit Distance**
Graphical Representation:

Compose A × Tsa · Tst · Fsa | Tstbestpath × C · Fst

Program:

\[
C = argmin \left( \alpha \circ T \circ B \right)
\]

String Alignment by Automata
<table>
<thead>
<tr>
<th>Context-dependent</th>
<th>0</th>
<th>dx</th>
<th>$\Lambda/\Lambda'$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>ax</td>
<td>r ETH</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>em</td>
<td>ETH m</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>ped pr</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expansion</th>
<th>$w(q',q)$</th>
<th>$q_i$</th>
<th>$q_i'$</th>
</tr>
</thead>
</table>

Generalized Weighted Edit Distance
Identity Transducer (I):

Phrase Fragment Transducer (L):

<table>
<thead>
<tr>
<th>Neither nor theoretical computer science</th>
<th>Neither nor computer science</th>
</tr>
</thead>
<tbody>
<tr>
<td>to indications</td>
<td>to indications</td>
</tr>
</tbody>
</table>

English Phrase

M. Mohri Transducer

Language Normalization
\[(\mathcal{L}) I = N \]

then:

\[ \exists \mathcal{A} \mathcal{L} - \exists \mathcal{A} = I \]

and

\[(\mathcal{L})^\dagger = \mathcal{A} \]

Let

Method 3:

\[ (\exists \mathcal{L}) \exists \mathcal{A} = N \]

Method 2:

\[ \exists \mathcal{L} \exists \mathcal{A} = N \]

Method 1:

with \( L = \text{fragments} \text{transducer and} \exists \text{identity transducer} = \mathcal{L} = \text{Normalization transducer} (N) \)
Tripicmonic context transducer for two symbols $x$ and $y$.

- CD units. Example: $ae/p q/aeb/p$.
- Context-dependent phone models: Maps from CI units to

Context-Dependency in ASR Modeling
N-Gram Language Model Examples
4. Use minimization for space

(b) epsilon transitions

(a) ordinary non-determinism

3. Reduce non-determinism for speed

(d) do not depend on the distribution of costs along a path

labels along a path

(c) do not depend on the registration between input and output

transitions

(b) do not depend on the presence or absence of epsilon

(a) encode information on transitions not states

2. Do not interpret equivalent FSMs differently:

I. Think in terms of operations on sets and relations

Rules of Thumb