Why Weighted Finite-State Transducers?

1. Efficiency and Generality of Classical Automata Algorithms
   - Efficient algorithms for a variety of problems (e.g. string-matching, compilers, Unix, design of controllability systems in aircrafts).
   - General algorithms: rational operations, intersection.

2. Weights
   - Handling uncertainty: text, handwritten text, speech, image, biological sequences.
   - Increased generality: finite-state transducers, multiplicity.

3. Applications
   - Text: pattern-matching, indexation, compression.
   - Speech: Large-vocabulary speech recognition, speech synthesis.
   - Image: image compression, filters.
Software Libraries

- **FSM Library**: Finite-State Machine Library – general software utilities for building, combining, optimizing, and searching weighted automata and transducers.

- **GRM Library**: Grammar Library – general software collection for constructing and modifying weighted automata and transducers representing grammars and statistical language models.
Weight Sets: Semirings

A semiring \((\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})\) is a ring that may lack negation.

- **Sum**: to compute the weight of a sequence (sum of the weights of the paths labeled with that sequence).

- **Product**: to compute the weight of a path (product of the weights of constituent transitions).

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Set</th>
<th>(\oplus)</th>
<th>(\otimes)</th>
<th>(\overline{0})</th>
<th>(\overline{1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>{0, 1}</td>
<td>\lor</td>
<td>\land</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Probability</td>
<td>(\mathbb{R}_+)</td>
<td>+</td>
<td>\times</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Log</td>
<td>(\mathbb{R} \cup {-\infty, +\infty})</td>
<td>(\oplus_{\log})</td>
<td>+</td>
<td>+(\infty)</td>
<td>0</td>
</tr>
<tr>
<td>Tropical</td>
<td>(\mathbb{R} \cup {-\infty, +\infty})</td>
<td>min</td>
<td>+</td>
<td>+(\infty)</td>
<td>0</td>
</tr>
</tbody>
</table>

with \(\oplus_{\log}\) defined by: \(x \oplus_{\log} y = -\log(e^{-x} + e^{-y})\).
Automata/Acceptors

- **Graphical Representation** (*A.ps)*:

```
red/0.5
0 --------> 1
|       |       |
| green/0.3 |       |
0 --------> 1
|       |       |
| blue/0   |       |
| yellow/0.6 |     |
1 --------> 2
```

- **Acceptor File** (*A.txt)*:

```
0 0 red .5
0 1 green .3
1 2 blue
1 2 yellow .6
2 .8
```

- **Symbols File** (*A.sym)*:

```
red 1
green 2
blue 3
yellow 4
```
Transducers

- **Graphical Representation** (T.ps):

  red:yellow/0.5

  ![Graphical representation of a transducer]

  0 → green:blue/0.3 → 1 → blue:green/0 → 2 /0.8

  yellow:red/0.6

- **Transducer File** (T.txt):

  0 0 red yellow .5
  0 1 green blue .3
  1 2 blue green
  1 2 yellow red .6
  2 .8

- **Symbols File** (T.sym):

  red 1
green 2
blue 3
yellow 4
Definitions and Notation – Paths

• Path \( \pi \)
  – Origin or previous state: \( p[\pi] \).
  – Destination or next state: \( n[\pi] \).
  – Input label: \( i[\pi] \).
  – Output label: \( o[\pi] \).

[Diagram: A circle labeled \( p[\pi] \) connected to a circle labeled \( n[\pi] \) with \( i[\pi]: o[\pi] \)]

• Sets of paths
  – \( P(R_1, R_2) \): set of all paths from \( R_1 \subseteq Q \) to \( R_2 \subseteq Q \).
  – \( P(R_1, x, R_2) \): paths in \( P(R_1, R_2) \) with input label \( x \).
  – \( P(R_1, x, y, R_2) \): paths in \( P(R_1, x, R_2) \) with output label \( y \).
Definitions and Notation – Automata and Transducers

1. General Definitions
   - Alphabets: input $\Sigma$, output $\Delta$.
   - States: $Q$, initial states $I$, final states $F$.
   - Transitions: $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q$.
   - Weight functions:
     initial weight function $\lambda : I \rightarrow \mathbb{K}$
     final weight function $\rho : F \rightarrow \mathbb{K}$.

2. Machines
   - Automaton $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ with for all $x \in \Sigma^*$:
     $$[A](x) = \bigoplus_{\pi \in P(I, x, F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$
   - Transducer $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ with for all $x \in \Sigma^*, y \in \Delta^*$:
     $$[T](x, y) = \bigoplus_{\pi \in P(I, x, y, F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$
Rational Operations – Algorithms

- Definitions

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>DEFINITION AND NOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum</td>
<td>([T_1 \oplus T_2](x, y) = [T_1](x, y) \oplus [T_2](x, y))</td>
</tr>
<tr>
<td>Product</td>
<td>([T_1 \otimes T_2](x, y) = \bigoplus_{x = x_1, y = y_1, y_2} [T_1](x_1, y_1) \otimes [T_2](x_2, y_2))</td>
</tr>
<tr>
<td>Closure</td>
<td>([T^*](x, y) = \bigoplus_{n=0}^\infty [T]^n(x, y))</td>
</tr>
</tbody>
</table>

- **Conditions on the closure operation**: condition on \(T\): e.g. weight of \(\epsilon\)-cycles = 0 (regulated transducers), or semiring condition: e.g. \(\top \oplus x = \top\) as with the tropical semiring (locally closed semirings).

- **Complexity and implementation**
  - Complexity (linear): \(O((|E_1| + |Q_1|) + (|E_2| + |Q_2|))\) or \(O(|Q| + |E|)\).
  - Lazy implementation.
Sum - Illustration

- **Program:** fsmunion A.fsm B.fsm > C.fsm
- **Graphical Representation:**

![Graphical Representation of FSMs](image-url)
Product - Illustration

- **Program:**  fsmconcat A.fsm B.fsm >C.fsm
- **Graphical Representation:**

![Diagram of A.fsa](image1)

![Diagram of B.fsa](image2)

![Diagram of C.fsa](image3)
Some Elementary Unary Operations – Algorithms

• Definitions

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>DEFINITION AND NOTATION</th>
<th>LAZY IMPLEMENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reversal</td>
<td>$[\tilde{T}](x, y) = [T](\tilde{x}, \tilde{y})$</td>
<td>No</td>
</tr>
<tr>
<td>Inversion</td>
<td>$[T^{-1}](x, y) = [T](y, x)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Projection</td>
<td>$<a href="x">A</a> = \bigoplus_{y} [T](x, y)$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

• Complexity and implementation
  – Complexity (linear): $O(|Q| + |E|)$.
  – Lazy implementation (see table).
Reversal - Illustration

- **Program**: `fsmreverse A.fsm > C.fsm`

- **Graphical Representation**:

```
0 --> 1: green/0.3, yellow/0.6

0: red/0.5

1 --> 2: blue/0

1: green/0.3

2 --> 3: green/1.2, blue/2

2: yellow/0.6

3 --> 4: green/1.2

4 --> 5: eps/0

4: green/1.2

5: eps/0.3

5 --> 2: blue/2

6: eps/0.3

2 --> 1: green/0.3

2: blue/0

3 --> 2: green/0.3

3: blue/0

4 --> 5: green/1.2

5: eps/0.3

```

A.fsa

C.fsa
Inversion – Illustration

- **Program:**  
  `fsminvert A.fsm > C.fsm`

- **Graphical Representation:**

  ![Graphical Representation](image-url)

A.fst

C.fst
Some Fundamental Binary Operations – Algorithms

- **Definitions**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Definition and Notation</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition</td>
<td>$[T_1 \circ T_2](x, y) = \bigoplus_z [T_1](x, z) \otimes [T_2](z, y)$</td>
<td>$\mathbb{K}$ commutative</td>
</tr>
<tr>
<td>Intersection</td>
<td>$<a href="x">A_1 \cap A_2</a> = <a href="x">A_1</a> \otimes <a href="x">A_2</a>$</td>
<td>$\mathbb{K}$ commutative</td>
</tr>
<tr>
<td>Difference</td>
<td>$<a href="x">A_1 - A_2</a> = <a href="x">A_1 \cap \overline{A_2}</a>$</td>
<td>$A_2$ unweighted &amp; deterministic</td>
</tr>
</tbody>
</table>

- **Complexity and implementation**
  - Complexity (quadratic): $O((|E_1| + |Q_1|)(|E_2| + |Q_2|))$.
  - Path multiplicity in presence of $\epsilon$-transitions: $\epsilon$-filter.
  - Lazy implementation.
Composition – Illustration

- **Program:**  
  ```  
  fsmcompose A.fsm B.fsm > C.fsm  
  ```

- **Graphical Representation:**

```plaintext
  0  \rightarrow_\text{a:b/0.1} 1  \rightarrow_\text{a:a/0.4} 3/0.6  \rightarrow_\text{c:a/0.3} 1  \rightarrow_\text{a:a/0.4} 2  \rightarrow_\text{b:b/0.5} 3/0.6  \rightarrow_\text{a:b/0.6} 2/0.7  
  0  \rightarrow_\text{b:c/0.3} 1  \rightarrow_\text{a:b/0.4} 2/0.7  
  (0, 0)  \rightarrow_\text{a:c/0.4} (1, 1)  \rightarrow_\text{c:b/0.7} (1, 2)  \rightarrow_\text{c:b/0.9} (1, 2)  \rightarrow_\text{a:b/1} (3, 2)/1.3  
```
Multiplicity & \( \epsilon \)-Transitions – Problem

Redundant \( \epsilon \)-paths.
Solution – Filter $F$ for Composition

Replace $T_1 \circ T_2$ by $T_1 \circ F \circ T_2$. 
Intersection - Illustration

- **Program:**  fsmintersect A.fsm B.fsm > C.fsm

- **Graphical Representation:**

```
A.fsa

0 → 1 [green/0.3, red/0.5]
  ↓[yellow/0.6] 2 / 0.8

B.fsa

0 / 0 → 1 [blue/0, red/0.2]
  ↓[blue/0.6] 2 / 0.8

C.fsa

0 → 1 [red/0.7, green/0.7]
  ↓[blue/0.6] 2 / 0.8
  ↓[yellow/1.9] 3 / 0.8
  ↓[yellow/1.3] 4 / 1.3
```
Single-Source Shortest-Distance Algorithms – Algorithm

- **Generic single-source shortest-distance algorithm**
  - Definition: for each state \( q \),
    \[
    d[q] = \bigoplus_{\pi \in P(q,F)} w[\pi]
    \]
  - Works with any queue discipline and any semiring \( k \)-closed for the graph.
  - Coincides with classical algorithms in the specific case of the tropical semiring and the specific queue disciplines: best-first (Dijkstra), FIFO (Bellman-Ford), or topological sort order (Lawler).

- **\( N \)-best strings algorithm**
  - General \( N \)-best paths algorithm augmented with the computation of the potentials.
  - On-the-fly weighted determinization.
Single-Source Shortest-Distance Algorithms – Illustration

- **Program:** `fsmbestpath [-n N] A.fsm > C.fsm`

- **Graphical Representation:**

  ![Diagram 1](image1.png)

  ![Diagram 2](image2.png)
Pruning - Illustration

- **Program:**  fsmprune -c1.0 A.fsm >C.fsm

- **Graphical Representation:**

![Diagram of a finite-state automaton with states 0, 1, 2, 3, and 4, showing transitions labeled with colors and probabilities.]

A.fsa

![Diagram of a simplified finite-state automaton with states 0, 1, and 2, showing transitions labeled with colors and probabilities.]

C.fsa