Speech Recognition
Lecture 4: Weighted Finite-State Transducer
Software Library

Cyril Allauzen
Google, NYU Courant Institute
allauzen@cs.nyu.edu
Slide Credit: Mehryar Mohri/Eugene Weinstein
Software Libraries

- **FSM Library**: Finite-State Machine Library. General software utilities for building, combining, optimizing, and searching weighted automata and transducers (Mohri, Pereira, and Riley, 2000).


- **OpenFst Library**: Open-source Finite-state transducer Library (Allauzen et al., 2007).

  [http://openfst.org](http://openfst.org)
Software Libraries

- **GRM Library:** Grammar Library. General software collection for constructing and modifying weighted automata and transducers representing grammars and statistical language models (Allauzen, Mohri, and Roark, 2005).

  http://www2.research.att.com/~fsmtools/grm/
Software Libraries

- **OpenGRM Libraries**: open source libraries for constructing and using formal grammars in FST form, using OpenFST as underlying representation.

- **NGram Library**: create and manipulate language models encoded as weighted FSTs. (Roark et al., 2012)

- **Thrax**: compiles regular expressions and context-dependent rewrite grammars into weighted FSTs. (Tai, Skut, and Sproat, 2011)

- **http://opengrm.org**
Software Libraries

- **DCD Library**: Decoder Library. General software collection for speech recognition decoding and related functions (Mohri and Riley, 2003).

http://www2.research.att.com/~fsmtools/dcd/
The OpenFST utilities construct, combine, minimize, and search weighted finite-states transducers.

- **User Program Level**: Programs that read from and write to files or pipelines:
  \[
  \text{fstcompose [--opts] a.fst b.fst out.fst}
  \]

- **C(++) Library Level**: Library archive of C(++) functions that implements the user program level.
  \[
  \text{Fst<Arc> A, B, C;}
  \]
  \[
  \text{...}
  \]
  \[
  \text{Compose(A, B, &C);}
  \]
This Lecture

- Weighted automata and transducers
- Rational operations
- Elementary unary operations
- Fundamental binary operations
- Optimization algorithms
- Search algorithms
Weighted Automata

\[ [[A]](x) = \text{Sum of the weights of all successful paths labeled with } x \]

\[ [[A]](abb) = 0.1 \times 0.2 \times 0.3 \times 0.1 + 0.5 \times 0.3 \times 0.6 \times 0.1 \]
Weighted Transducers

\[
[[T]](x, y) = \text{Sum of the weights of all successful paths with input } x \text{ and output } y.
\]

\[
[[T]](abb, baa) = .1 \times .2 \times .3 \times .1 + .5 \times .3 \times .6 \times .1
\]
Weight Sets: Semirings

A semiring \((\mathbb{K}, \oplus, \otimes, 0, 1)\) is a grouping two operations, their identity elements, and the set of numbers they operate on.

“A ring that may lack negation”. Operations:

- **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).
## Semirings - Examples

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>{0, 1}</td>
<td>∨</td>
<td>∧</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Probability</td>
<td>(\mathbb{R}_+)</td>
<td>+</td>
<td>×</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: \(x \oplus_{\log} y = -\log(e^{-x} + e^{-y})\).
General Definitions

- **Alphabets**: input $\Sigma$, output $\Delta$.

- **States**: $Q$, initial states $I$, final states $F$.

- **Transitions**: $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times K \times Q$.

- **Weight functions**:
  - **initial**: $\lambda : I \rightarrow K$.
  - **final**: $\rho : F \rightarrow K$. 
Automata and Transducers - Definitions

- **Automaton** $A = (\Sigma, Q, I, F, E, \lambda, \rho)$
  \[ A(x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]) \]
  \[ \forall x \in \Sigma^* \]

- **Transducer** $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$
  \[ T(x, y) = \bigoplus_{\pi \in P(I,x,y,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]) \]
  \[ \forall x \in \Sigma^*, y \in \Delta^* \]
OpenFST File Types

- **Textual format**
  - automata/acceptor files,
  - transducer files,
  - symbols files.

- **Binary format**: compiled representation used by all OpenFST utilities.
Compiling and Printing

- **Compiling**
  
  - `fstcompile --arc_type=standard --isymbols=syms.txt --acceptor <A.txt >A.fst`
  
  - `fstcompile --arc_type=log --isymbols=A.syms --osymbols=A.syms <T.txt >T.fst`

- **Printing**
  
  - `fstprint --isymbols=A.syms --acceptor < A.fst >A.txt`
  
  - `fstprint --isymbols=A.syms --osymbols=A.syms < T.fst >T.txt`
Drawing

- `fstdraw --isymbols=A.syms --acceptor < A.fst | dot -Tps > A.ps`
- `fstdraw --isymbols=A.syms --osymbols=A.syms < T.fst | dot -Tps > T.ps`
Automata/Acceptors

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

- **Acceptor file** \((A.txt)\)

- **Symbols file** \((A.syms)\)
Transducers

Graphical Representation \((T.ps)\)

\[
\begin{array}{c}
\text{0} \\
\downarrow^{\text{green:blue/0.3}} \\
\text{1} \\
\downarrow^{\text{blue:green/0}} \\
\text{2 /0.8}
\end{array}
\]

- red:yellow/0.5
- green:blue/0.3
- blue:green/0
- yellow:red/0.6

Transducer file \((T.txt)\)

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>red</th>
<th>yellow</th>
<th>.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>green</td>
<td>blue</td>
<td>.3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>blue</td>
<td>green</td>
<td>.3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>yellow</td>
<td>red</td>
<td>.6</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>.8</td>
</tr>
</tbody>
</table>

Symbols file \((T.sym)\)

- red 1
- green 2
- blue 3
- yellow 4
This Lecture

- Weighted automata and transducers
- Rational operations
- Elementary unary operations
- Fundamental binary operations
- Optimization algorithms
- Search algorithms
Rational Operations

- **Sum/Union**

\[
[T_1 \oplus T_2](x, y) = [T_1](x, y) \oplus [T_2](x, y)
\]

- **Product/Concatenation**

\[
[T_1 \otimes T_2](x, y) = \bigoplus_{x=x_1 x_2, \ y=y_1 y_2} [T_1](x_1, y_1) \otimes [T_2](x_2, y_2).
\]

- **Closure**

\[
[T^*](x, y) = \bigoplus_{n=0}^{\infty} [T]^n(x, y)
\]
**Sum - Illustration**

- **Program:** `fstunion A.fst B.fst > C.fst`

- **Graphical representation:**

```
0 -------------------------- 1 -------------------------- 2
   red/0.5  green/0.3  blue/0  yellow/0.6  2 /0.8
          |                   |                   |
3 --------|-----------------|----------------|-----------
       eps/0  green/0.3  blue/0  yellow/0.6  2 /0.8
6 --------|-----------------|----------------|-----------
           eps/0
```

Courant Institute, NYU
**Program**: `fstconcat A.fst B.fst > C.fst`

**Graphical representation:**

```
0 [red/0.5] -> 1 [green/0.3] -> 2 [blue/0, 2/0.8] -> 3 [eps/0.8] <- [22]

0 [red/0.5] -> 1 [green/0.3] -> 2 [blue/0, 2/0.8] -> 3 [eps/0.8] <- [22]
```

```
0 [green/0.4] -> 1 [blue/1.2] -> 2 [1/0, 2/0.3] -> 0 [blue/1.2] <- [22]

0 [green/0.4] -> 1 [blue/1.2] -> 2 [1/0, 2/0.3] -> 0 [blue/1.2] <- [22]
```
Closure - Illustration

- **Program**: `fstclosure B.fst > C.fst`
- **Graphical representation:**

```
0 - green/0.4 -> 1 /0
   - blue/1.2 -> 2 /0.3

3 /0 - eps/0 -> 0
    - green/0.4 -> 1 /0
    - eps/0 -> 2 /0.3
```

```
0 - eps/0 -> 3 /0
    - blue/1.2 -> 1 /0
    - eps/0 -> 2 /0.3
```
This Lecture

- Weighted automata and transducers
- Rational operations
- Elementary unary operations
- Fundamental binary operations
- Optimization algorithms
- Search algorithms
Elementary Unary Operations

- **Reversal**

\[
[T](x, y) = [\tilde{T}](\tilde{x}, \tilde{y})
\]

- **Inversion**

\[
[T^{-1}](x, y) = [T](y, x)
\]

- **Projection**

\[
[A](x) = \bigoplus_y [T](x, y)
\]

- **Linear-time complexity, lazy implementation (not for reversal).**
Reversal - Illustration

- **Program**: `fstreverse A.fst > C.fst`
- **Graphical representation:**

```
red/0.5 -> 0
  
  green/0.3 -> 1
  blue/0 -> 2
  yellow/0.6 -> 2
  green/1.2 -> 3
  blue/2 -> 4

red/0.5 -> 0
  
  eps/0 -> 4
  green/1.2 -> 4
  blue/0 -> 3
  yellow/0.6 -> 3
  blue/2 -> 2
  green/0.3 -> 1
  1/0 -> 1
```

Courant Institute, NYU
Inversion - Illustration

- **Program**: `fstinvert A.fst > C.fst`

- **Graphical representation**:

```
0 -> 1: red:bird/0.5, green:pig/0.3, blue:cat/0, yellow:dog/0.6 -> 2/0.8
0 -> 1: pig:green/0.3, cat:blue/0, dog:yellow/0.6 -> 2/0.8
```
Projection - Illustration

- **Program**: fstproject --project_output=false T.fst > A.fst

- **Graphical representation**:

  ![Graphical representation](image-url)
This Lecture

- Weighted automata and transducers
- Rational operations
- Elementary unary operations
- **Fundamental binary operations**
- Optimization algorithms
- Search algorithms
Some Fundamental Binary Operations

(Pereira and Riley, 1997; Mohri et al. 1996)

- **Composition** ((\(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1}\)) commutative)

\[
[T_1 \circ T_2](x, y) = \bigoplus_z [T_1](x, z) \otimes [T_2](z, y)
\]

- **Intersection** ((\(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1}\)) commutative)

\[
[A_1 \cap A_2](x) = [A_1](x) \otimes [A_2](x)
\]

- **Difference** (\(A_2\) unweighted and deterministic)

\[
[A_1 - A_2](x) = [A_1 \cap \overline{A_2}](x)
\]
• Complexity and implementation:

  • quadratic complexity:
    \[ O((|E_1| + |Q_1|) (|E_2| + |Q_2|)) \]

  • path multiplicity in presence of \( \varepsilon \)-transitions: \( \varepsilon \)-filter;

  • lazy implementation.
Composition - Illustration

- **Program**: `fstcompose A.fst B.fst > C.fst`
- **Graphical representation**:

```
 fstcompose A.fst B.fst > C.fst
```

Graphical representation:

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```

```
Program: fstcompose A.fst B.fst > C.fst

Graphical representation:
```

```
Composition - Illustration
```
**Intersection - Illustration**

- **Program**: `fstintersect A.fst B.fst > C.fst`

- **Graphical representation**:

```plaintext
Graphical representation
```

```plaintext
red/0.5  

0 green/0.3 1 blue/0 2 /0.8

0 /0 1 red/0.2 1 green/0.4 yellow/1.3 2 /0.5

0 red/0.7 1 green/0.7 2

3 /0.8 blue/0.6

4 /1.3 yellow/1.9
```
Difference - Illustration

- **Program**: `fstdifference A.fst B.fst > C.fst`

- **Graphical representation**: 

```plaintext
1. Graph 1: 
   - Node 0: red/0.5, green/0.3
   - Node 1: blue/0, yellow/0.6
   - Node 2: red/0.8

2. Graph 2: 
   - Node 0: red
   - Node 1: blue
   - Node 2: green
   - Node 3: blue/0, yellow/0.6
   - Node 4: green/0.3
```

Cyril Allauzen - Speech Recognition
This Lecture

- Weighted automata and transducers
- Rational operations
- Elementary unary operations
- Fundamental binary operations
- Optimization algorithms
- Search algorithms
Optimization Algorithms

- **Connection**: removes non-accessible/non-coaccessible states.
- **$\varepsilon$-Removal**: removes $\varepsilon$-transitions.
- **Determinization**: creates equivalent deterministic machine.
- **Pushing**: creates equivalent pushed/stochastic machine.
- **Minimization**: creates equivalent minimal deterministic machine.
• **Conditions:** there are specific semiring conditions for the use of these algorithms, e.g., not all weighted automata or transducers can be determinized using the determinization algorithm.
Connection - Illustration

- **Program:** `fstconnect A.fst > C.fst`

- **Graphical representation:**

```
                  3	  green/0.2  
                   ↓       ↓
                   0	  red/0.5  
                        ↓
                        1	  green/0.3  
                            ↓
                            0	  red/0.5  
                                ↓
                                1	  green/0.3  
                                    ↓
                                    2 /0.8  
                                        blue/0
                                        ↓
                                        5

                  4 /0.2
                   ↓
                   1

                  2 /0.8
                   ↓
                   5
```

```
Connection - Algorithm

- **Definition:**
  - Input: weighted transducer \( T_1 \).
  - Output: equivalent weighted transducer \( T_2 \) with all states connected.

- **Description:**
  - Depth-first search of \( T_1 \) from \( I_1 \).
  - Mark accessible and coaccessible states.
  - Keep marked states and corresponding transitions.

- **Complexity:** linear \( O(|Q_1| + |E_1|) \).
ε-Removal - Algorithm

(Mohri, 2001)

Definition:

- Input: weighted transducer $T_1$.
- Output: equivalent WFST $T_2$ with no ε-transition.

Description:

- Computation of ε-closures.
- Removal of εs.

Complexity:

- Acyclic $T_ε : O(|Q|^2 + |Q||E|(T_⊕ + T_⊗))$.
- General case (tropical semiring):
  
  $O(|Q||E| + |Q|^2 \log |Q|)$
**ε-Removal - Illustration**

- **Program**: `fstrmepsilon [--connect] [--reverse] T.fst > T1.fst`
- **Graphical representation:**

![Graphical representation of ε-Removal](image)
Determinization - Algorithm

(Mohri, 1997)

Definition:
- Input: weighted automaton or transducer $T_1$
- Output: equivalent subsequential or deterministic machine $T_2$: has a unique initial state and no two transitions leaving the same state share the same input label.

Description:
- Generalization of subset construction: weighted subsets $\{(q_1, w_1), \ldots, (q_n, w_n)\}$, where $w_i$s are remainder weights.
- Computation of the weight of resulting transitions.
Determinization - Conditions

- **Semiring**: weakly left divisible semirings.
- **Definition**: $T$ is determinizable when the determinization algorithm applies to $T$.
  - All unweighted automata are determinizable.
  - All acyclic machines are determinizable.
  - Not all weighted automata or transducers are determinizable.
  - Characterization based on the *twins property*.
- **Complexity**: exponential, but lazy implementation.
Determinization of Weighted Automata - Illustration

Program: fstdeterminize A.fst > D.fst

Graphical representation:
Determinization of Weighted Transducers - Illustration

- **Program:** `fstdeterminize T.fst > D.fst`

- **Graphical representation:**

```
0 -------------------> 2 -------------------> 3 -------------------> 4
\   \                     \   \                     \   \                     \
\    \ a:a/0.1           \    \ b:b/0.3           \    \ a:eps/0.5       \
\      \ a:b/0.2         \      \                   \      \       \
\       \ a:c/0.5        \       \                   \       \       \
\         \              \         \                   \         \       \
\          \ 1           \          \ 4/0.7         \          \       \
\           \     \      \           \              \      \       \
\            \ 2/eps/0.8 \      \           \              \       \       \
\             \       \     \             \              \       \       \
\              \ 1/eps/0.8 \       \             \              \       \       \
\                \       \     \                \              \       \       \
\                 \ 1/eps/0.8 \       \                \              \       \       \
\                  \       \     \                  \              \       \       \
\                   \ 1/eps/0.8 \       \                   \              \       \       \
\                    \       \     \                    \              \       \       \
\                     \ 1/eps/0.8 \       \                     \              \       \       \
\                      \       \     \                      \              \       \       \
\                       \ 1/eps/0.8 \       \                       \              \       \       \
\                        \       \     \                        \              \       \       \
\                         \ 1/eps/0.8 \       \                         \              \       \       \
\                          \       \     \                          \              \       \       \
\                           \ 1/eps/0.8 \       \                           \              \       \       \
\                            \       \     \                            \              \       \       \
\                             \ 1/eps/0.8 \       \                             \              \       \       \
\                              \       \     \                              \              \       \       \
\                               \ 1/eps/0.8 \       \                               \              \       \       \
\                                \       \     \                                \              \       \       \
\                                 \ 1/eps/0.8 \       \                                 \              \       \       \
\                                  \       \     \                                  \              \       \       \
\                                   \ 1/eps/0.8 \       \                                   \              \       \       \
\                                    \       \     \                                    \              \       \       \
\                                     \ 1/eps/0.8 \       \                                     \              \       \       \
\                                      \       \     \                                      \              \       \       \
\                                       \ 1/eps/0.8 \       \                                       \              \       \       \
\                                        \       \     \                                        \              \       \       \
\                                         \ 1/eps/0.8 \       \                                         \              \       \       \
\                                          \       \     \                                          \              \       \       \
\                                           \ 1/eps/0.8 \       \                                           \              \       \       \
```

```
{0, eps, 0} -------------------> \{{1, c, 0.4}, (2, a, 0)\}
\   \                     \   \                     \   \                     \
\    \ a:eps/0.1           \    \ a:a/0.2           \    \ a:b/0.1       \
\      \ a:b/0.2         \      \                   \      \                   \
\       \ a:c/0.5        \       \                   \       \                   \
\         \              \         \                   \         \       \
\        \{3, b, 0\}, \/(eps, 0.8) \        \                   \        \       \
\         \              \         \                   \         \       \
\          \{4, eps, 0\} \          \                   \          \       \
\           \              \           \                   \           \       \
```

```
Pushing - Algorithm

Definition:

- **Input**: weighted automaton or transducer $T_1$
- **Output**: equivalent automaton or transducer $T_2$

such that the longest common prefix of all outgoing paths are $\varepsilon$ or such that the sum of the weights of all outgoing transitions are $\overline{1}$ modulo the string or weight at the initial state.
• **Description:**

1. Single-source shortest distance computation: for each state $q$,

$$d[q] = \bigoplus_{\pi \in P(q,F)} w[\pi].$$

2. Reweighting: for each transition $e$ such that $d[p[e]] \neq 0$,

$$w[e] \leftarrow (d[p[e]])^{-1}(w[e] \otimes d[n[e]])$$
• **Conditions** (automata case): weakly divisible semiring, zero-sum free semiring or automaton.

• **Complexity:**
  - automata case
  - acyclic case: linear $O(|Q| + |E|(T_\oplus + T_\otimes))$.
  - general case (tropical semiring):
    $$O(|Q| \log |Q| + |E|).$$
  - transducer case:
    $$O((|P_{max}| + 1) |E|).$$
Weight Pushing - Illustration

Program: fstpush --push_weights --to_final=false A.fst > P.fst

Graphical representation:

- Tropical semiring:
• **Log semiring:**

![Diagram of a log semiring with nodes labeled 0, 1, 2, 3 and transitions labeled with symbols and values.]
Label Pushing - Illustration

- **Program**: `fstpush --push_labels --to_final=false T.fst >P.fst`

- **Graphical representation**: 

```
<table>
<thead>
<tr>
<th>State</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a:a</td>
</tr>
<tr>
<td>1</td>
<td>c: ε</td>
</tr>
<tr>
<td>2</td>
<td>b: ε</td>
</tr>
<tr>
<td>3</td>
<td>a: ε</td>
</tr>
<tr>
<td>4</td>
<td>b: ε</td>
</tr>
<tr>
<td>5</td>
<td>c: d</td>
</tr>
<tr>
<td>6</td>
<td>f: d</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>State</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a:a</td>
</tr>
<tr>
<td>1</td>
<td>c: d</td>
</tr>
<tr>
<td>2</td>
<td>b: ε</td>
</tr>
<tr>
<td>3</td>
<td>a: d</td>
</tr>
<tr>
<td>4</td>
<td>b: ε</td>
</tr>
<tr>
<td>5</td>
<td>c: ε</td>
</tr>
<tr>
<td>6</td>
<td>f: ε</td>
</tr>
</tbody>
</table>
```
Minimization - Algorithm

(Mohri, 1997)

Definition:
- Input: deterministic weighted automaton or transducer $T_1$.
- Output: equivalent deterministic automaton or transducer $T_2$ with the minimal number of states and transitions.

Description:
- Canonical representation: use pushing or other algorithm to standardize input automata.
- Automata minimization: encode pairs (label, weight) as labels and use classical unweighted minimization algorithm.
• **Complexity:**

  • **Automata case**
    
    • acyclic case: linear, $O(|Q| + |E|(T_+ + T_\otimes))$.
    
    • general case (tropical semiring): $O(|E| \log |Q|)$.

  • **Transducer case**
    
    • acyclic case: $O(S + |Q| + |E|(|P_{max}| + 1))$.
    
    • general case (tropical semiring): 
      
      $$O(S + |Q| + |E| (\log |Q| + |P_{max}|)).$$
Minimization - Illustration

- **Program**: `fstminimize D.fst > M.fst`

- **Graphical representation:**

```plaintext
Program: fstminimize D.fst > M.fst
```

```plaintext
Graphical representation:
```

![Minimization Illustration](image-url)
Equivalence - Algorithm

Definition:
- Input: deterministic weighted automata $A$ and $B$.
- Output: $\text{TRUE}$ iff $A$ and $B$ equivalent.

Description (Mohri, 1997):
- Canonical representation: use pushing or other algorithm to standardize input automata.
- Automata minimization: encode pairs (label, weight) as labels and use classical algorithm for testing the equivalence of unweighted automata.

Complexity: (second stage is quasi-linear)

$$O(|E_1| + |E_2| + |Q_1| \log |Q_1| + |Q_2| \log |Q_2|).$$
Equivalence - Illustration

- **Program**: if fstequivalent A.fst B.fst; then echo true; else echo false; fi

- **Graphical representation**:

```
0  red/0.3  1  yellow/0.9
    |                 |
    |                 |  blue/0.7
    |                 |
  2 /0.3

0  red/0  1  blue/0
    |     |
    |  yellow/0.3
    |     |
  2 /1.3
```
This Lecture

- Weighted automata and transducers
- Rational operations
- Elementary unary operations
- Fundamental binary operations
- Optimization algorithms
- Search algorithms
Single-Source Shortest-Distance Algorithms - Illustration

- **Program**: `fstshortestpath [--nshortest N] A.fst > C.fst`

- **Graphical representation**:

```
0 1 2 3 4
red/0.5 red/0.5 red/0.5 green/0.3 blue/0 blue/0
0 1 2 3 4
green/0.3 green/0.3 green/0.3 yellow/0.6
0 1 2 3 4
```

38 58
Pruning - Illustration

- **Program:** `fstprune --weight 1.0 A.fst > C.fst`
- **Graphical representation:**

![Graphical representation of pruning](image-url)
Summary

- **OpenFST Library:**
  - weighted finite-state transducers (semirings);
  - elementary unary operations (e.g., reversal);
  - rational operations (sum, product, closure);
  - fundamental binary operations (e.g., composition);
  - optimization algorithms (e.g., $\varepsilon$-removal, determinization, minimization);
  - search algorithms (e.g., shortest-distance algorithms, $n$-best paths algorithms, pruning).
References


References


References
