Weighted Finite-State Transducers
Important Algorithms

Fig. 12. Weight pushing algorithm. (a) Weighted automaton $A$.
(b) Equivalent weighted automaton $B$ obtained by weight pushing in the tropical semiring.
(c) Weighted automaton $C$ obtained from $A$ by weight pushing in the probability semiring.
(d) Minimal weighted automaton over the tropical semiring equivalent to $A$.

An approximate version of a generic shortest-distance algorithm can be used instead to compute $d[q]$ efficiently.

Note that if $d[q]=0$, then, since $S$ is zero-sum-free, the weight of all paths from $q$ to $F$ is 0. Let $A$ be a weighted automaton over the semiring $S$. Assume that $S$ is complete or $k$-closed and that the shortest-distances $d[q]$ are well-defined in $S-\{0\}$. Note that in both cases we can use the distributivity over the infinite sums defining shortest distances. Let $e'(\pi')$ denote the transition $e(\pi)$ after application of the weight pushing algorithm.

The following proposition is proven in [38, 43].

Proposition 7 ([38, 43]). Let $B=(A, Q, I, F, E', \lambda', \rho')$ be the result of the weight pushing algorithm applied to the weighted automaton $A$, then

1. the weight of a successful path $\pi$ is unchanged after application of weight pushing:

$$\lambda'[p[\pi']] \otimes w[\pi'] \otimes \rho'[n[\pi']] = \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]).$$

(39)

2. the weighted automaton $B$ is stochastic, i.e.

$$\bigoplus_{e' \in E'}[q]w[e'] = 1.$$ (40)

These two properties of weight pushing are illustrated by Figures 12(a)-(c): the total weight of a successful path is unchanged after pushing; at each state of the weighted automaton of Figure 12(b), the minimum weight of the outgoing transitions is 0, and at each state of the weighted automaton of Figure 12(c), the weights of outgoing transitions sum to 1.

Weight pushing can also be used to test the equivalence of two subsequential weighted automata [38, 43]. Let $A$ and $B$ be two subsequential weighted automata to which weight pushing can be applied and let $A'$ and $B'$ be the
Lecture Outline

• Preliminaries
  • Semirings
  • Transducers and Acceptors

• Important Algorithms
  • Composition
  • Determinization
  • Epsilon Removal
  • Other algorithms
Semirings (I)

- WFSTs and WFST-based operations are underpinned by algebraic objects called **semirings**

**Definition** A semiring is a system \((\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})\) such that,

1. \((\mathbb{K}, \oplus, \bar{0})\) is a commutative [monoid](https://wikipedia.org) with \(\bar{0}\) as the identity element for \(\oplus\),
2. \((\mathbb{K}, \otimes, \bar{1})\) is a monoid with \(\bar{1}\) as the identity element for \(\otimes\),
3. \(\oplus\) distributes over \(\oplus\): for all \(a, b, c \in \mathbb{K}\),
   \[(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c),\]
   \[c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b),\]
4. \(\bar{0}\) is an annihilator for \(\otimes\): \(\forall a \in \mathbb{K}, a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}\)

- This has implications for optimization, search, and combination algorithms such as determinization, shortest-path, and composition

A monoid is an algebraic structure that supports a single associative binary operation and an identity element. ([wikipedia](https://wikipedia.org))

*Speech Recognition with Weighted Finite-State Transducers*, Mohri et. al
Semirings (II)

- A variety of semirings exist, but there are two that are of particular interest for NLP and ASR applications
  - The log semiring
    - Isomorphic to the real semiring, high numerical stability
  - The tropical semiring
    - Convenient for shortest-path algorithms because global path weights are guaranteed to be monotonic non-decreasing, viterbi approximation is fast

<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>( \lor )</td>
<td>( \land )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Probability</td>
<td>( \mathbb{R}^+ \cup {+\infty} )</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} )</td>
<td>( \min )</td>
<td>( + )</td>
<td>( +\infty )</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. Semiring examples. \( \oplus_{\log} \) is defined by: \( x \oplus_{\log} y = -\log(e^{-x} + e^{-y}) \).

Semiring frameworks and algorithms for shortest-distance problems
Mohri, 2002, for more details
Semirings (example)

- Tropical Semiring example
  - Simple, but unintuitive!

<table>
<thead>
<tr>
<th>Operation Definitions</th>
<th>Trivial Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a \otimes b = a + b)</td>
<td>(5 \oplus 3 = 3)</td>
</tr>
<tr>
<td>(a \oplus b = \min(a,b))</td>
<td>(4 \oplus 2 \oplus 7 = 2)</td>
</tr>
<tr>
<td>(\overline{1} = 0)</td>
<td>((3 \otimes 4) \oplus 6 = 6)</td>
</tr>
<tr>
<td>(\overline{0} = +\infty)</td>
<td>(1 \otimes 5 = 6)</td>
</tr>
<tr>
<td></td>
<td>((1 \otimes 5) \oplus (2 \otimes 4) = 6)</td>
</tr>
<tr>
<td></td>
<td>(3 \oplus \overline{1} \Leftrightarrow 3 \oplus 0 \Leftrightarrow 0)</td>
</tr>
<tr>
<td></td>
<td>(3 \otimes \overline{1} \Leftrightarrow 3 \otimes 0 \Leftrightarrow 3)</td>
</tr>
</tbody>
</table>
Transducers and Acceptors

Definition A WFST is defined as an 8-tuple, \( T = (\Sigma, \triangle, Q, I, F, E, \lambda, \rho) \). Here \( \Sigma \) represents the finite alphabet of input symbols, \( \triangle \) represents the finite output alphabet, \( Q \) represents the finite set of states, \( I \subseteq Q \) the set of initial states, \( F \subseteq Q \) the set of final states, \( E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\triangle \cup \{\epsilon\}) \times K \times Q \) a finite set of state-to-state transitions, \( \lambda : I \rightarrow K \) the initial weight function, and \( \rho : F \rightarrow K \) the final weight function mapping \( F \) to \( K \).

- A WFSA is simply a WFST where the output labels have been omitted
- Similarly, FSAs and FSTs lack weights on the arcs or states

Speech Recognition with Weighted Finite-State Transducers, Mohri et. al
Basic examples

- Finite-State Acceptor (FSA)

- Weighted Finite-State Acceptor (WFSA)

- Finite-State Transducer (FST)

- Weighted Finite-State Transducer (WFST)
WFST Practice

- OpenFST (www.openfst.org)
- $ emacs test.fst.txt
- $ emacs test.syms
- $ fstcompile --isymbols=test.syms --osymbols=test.syms test.fst.txt | fstdraw --isymbols=test.syms --osymbols=test.syms --portrait=true | dot -Tpdf > test.pdf

- $ emacs test2.fst.txt
- $ emacs test2.syms
- $ fstcompile --acceptor=true --isymbols=test2.syms test2.fst.txt | fstdraw --portrait=true --acceptor=true --isymbols=test2.syms | dot -Tpdf > test2.pdf
Important algorithms

- There exists a wide variety of algorithms that operate on Weighted Finite-State Transducers
  - composition, determinization, minimization, epsilon-removal, epsilon-normalization, synchronization, weight-pushing, reversal, projection, shortest-path, connection, closure, concatenation, pruning, re-weighting, union, etc.

- Arguably the most important operations are composition, determinization, epsilon-removal, weight-pushing, and minimization
Composition (I)

- Fundamental operation for combining related transducers,

\[(T_1 \circ T_2)(x, y) = \bigoplus_{z \in \Delta^*} T_1(x, z) \otimes T_2(z, y)\]

- Used to iteratively combine multiple related knowledge sources to produce a single, integrated result

**Fig. 6.** Weighted transducers (a) \(T_1\) and (b) \(T_2\) over the probability semiring. (c) Illustration of composition of \(T_1\) and \(T_2\), \(T_1 \circ T_2\). Some states might be constructed during the execution of the algorithm that are not co-accessible, e.g., \((3, 2)\). Such states and the related transitions can be removed by a trimming (or connection) algorithm in linear-time.
### Algorithm 1 Weighted-Composition($T_1$, $T_2$)

1: $Q \leftarrow I_1 \times I_2$
2: $\mathcal{K} \leftarrow I_1 \times I_2$
3: while $\mathcal{K} \neq \emptyset$ do
4:   $q = (q_1, q_2) \leftarrow \text{Head}(\mathcal{K})$
5:   Dequeue($\mathcal{K}$)
6:   if $q \in I_1 \times I_2$ then
7:     $I \leftarrow I \cup \{q\}$
8:     $\lambda(q) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$
9:   if $q \in F_1 \times F_2$ then
10:      $F \leftarrow F \cup \{q\}$
11:      $p(q) \leftarrow p_1(q_1) \otimes p_2(q_2)$
12:     for each $(e_1, e_2) \in E[q_1] \times E[q_2]$ such that $o[e_1] = i[e_2]$ do
13:        if $(q') = (n[e_1], n[e_2] \notin Q)$ then
14:           $Q \leftarrow Q \cup \{q'\}$
15:           Enqueue($\mathcal{K}, q'$)
16:     $E \leftarrow E \cup \{(q, i[e_1], o[e_2], w[e_1] \otimes w[e_2], q')\}$
17: return $T$

<table>
<thead>
<tr>
<th>Algorithm complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
</tr>
<tr>
<td><strong>Space</strong></td>
</tr>
</tbody>
</table>

$V_i$=#states, $D_i$=max out-degree, $M_i$=max multiplicity of $i^{th}$ WFST. www.OpenFST.org
Composition (II)

Algorithm 1 Weighted-Composition($T_1$, $T_2$)

1: $Q \leftarrow I_1 \times I_2$ \hspace{1cm} \text{Initialize the queue and new WFST}
2: $\mathcal{K} \leftarrow I_1 \times I_2$
3: while $\mathcal{K} \neq \emptyset$ do
4: \hspace{1cm} $q = (q_1, q_2) \leftarrow \text{Head}(\mathcal{K})$
5: \hspace{1cm} $\text{Dequeue}(\mathcal{K})$
6: \hspace{1cm} if $q \in I_1 \times I_2$ then
7: \hspace{1.5cm} $I \leftarrow I \cup \{q\}$
8: \hspace{1.5cm} $\lambda(q) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$
9: \hspace{1cm} if $q \in F_1 \times F_2$ then
10: \hspace{1.5cm} $F \leftarrow F \cup \{q\}$
11: \hspace{1.5cm} $p(q) \leftarrow p_1(q_1) \otimes p_2(q_2)$
12: \hspace{1cm} for each $(e_1, e_2) \in E[q_1] \times E[q_2]$ such that $o[e_1] = i[e_2]$ do
13: \hspace{1.5cm} if $(q') = (n[e_1], n[e_2] \notin Q)$ then
14: \hspace{2cm} $Q \leftarrow Q \cup \{q'\}$
15: \hspace{2cm} $\text{Enqueue}(\mathcal{K}, q')$
16: \hspace{1.5cm} $E \leftarrow E \cup \{(q, i[e_1], o[e_2], w[e_1] \otimes w[e_2], q')\}$
17: return $T$

Algorithm complexity

| Time | $O(V_1 \cdot V_2 \cdot D_1 \cdot (\log D_2 + M_2))$ |
| Space | $O(V_1 \cdot V_2 \cdot V_1 \cdot M_2)$ |

$V_i = \#\text{states}, D_i = \max \text{out-degree}, M_i = \max \text{multiplicity of } i^{th} \text{ WFST}$. www.OpenFST.org
Composition (example)

C $(0_a, 0_b)$

- $a:b/0.1 \rightarrow (1_a)$
- $b:b/0.1 \rightarrow (1_b)$
- $a:b/0.01 \rightarrow (1_a, 1_b)$

Queue $(K)$

- $(0_a, 0_b)$

Node transitions:
- $a:b/0.1 \rightarrow (1)$
- $b:b/0.3 \rightarrow (1)$
- $a:b/0.5 \rightarrow (1)$
- $b:b/0.4 \rightarrow (1)$
- $a:a/0.6 \rightarrow (1)$
- $a:b/0.1 \rightarrow (2)$
- $b:a/0.2 \rightarrow (2)$
- $a:b/0.3 \rightarrow (2)$
- $b:b/0.1 \rightarrow (2)$
- $a:b/0.4 \rightarrow (2)$
- $b:a/0.5 \rightarrow (2)$
- $a:b/0.24 \rightarrow (3, 3)$
- $a:b/0.18 \rightarrow (3, 3)$
- $a:b/0.08 \rightarrow (3, 1)$
- $b:a/0.06 \rightarrow (3, 1)$
- $a:a/0.04 \rightarrow (3, 1)$
- $a:a/0.02 \rightarrow (3, 1)$
Composition (example)

\[ C \ (0_a, 0_b) \]

\[ a:b/0.1 \rightarrow (1_a) \]
\[ b:b/0.1 \rightarrow (1_b) \]
\[ a:b/0.01 \rightarrow (1_a, 1_b) \]

\[ \text{Queue (K)} \]
\[ (0_a, 0_b) \]

\[ \text{Queue (K+1)} \]
\[ (1_a, 1_b) \]
Composition (example)

\[ \begin{align*}
  c & \quad (1_a, 1_b) \\
  a : b & \rightarrow (0_a) \quad b : a & \rightarrow (1_b) \\
  b : b & \rightarrow (2_a) \quad a : b & \rightarrow (2_b) \\
  b : b & \rightarrow (3_a) \quad a : b & \rightarrow (3_b) \\
  a : a & \rightarrow (0_a, 1_b) \\
  b : a & \rightarrow (2_a, 1_b) \\
  b : a & \rightarrow (3_a, 1_b)
\end{align*} \]
Composition (example)

**C \((0_a, 1_b)\)**

<table>
<thead>
<tr>
<th>Transition</th>
<th>Action</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a:b/0.1)</td>
<td>((1_a))</td>
<td></td>
</tr>
<tr>
<td>(a:b/0.3)</td>
<td>((2_a))</td>
<td></td>
</tr>
<tr>
<td>(a:b/0.4)</td>
<td>((3_a))</td>
<td></td>
</tr>
<tr>
<td>(a:a/0.02)</td>
<td>((1_a, 1_b))</td>
<td></td>
</tr>
</tbody>
</table>

**Queue \((K)\)**

- \((0_a, 1_b)\)
- \((2_a, 1_b)\)
- \((3_a, 1_b)\)

**Queue \((K+1)\)**

- \((2_a, 1_b)\)
- \((3_a, 1_b)\)
Composition (example)

\[ C \ (2_a, 1_b) \]

\[
\begin{array}{c|c}
\text{a:b/0.5} & (3_a) \\
\text{b:a/0.2} & (1_b) \\
\text{a:b/0.3} & (2_b) \\
\text{a:b/0.4} & (3_b) \\
\text{a:a/0.1} & (3_a, 1_b) \\
\end{array}
\]

Queue \((K)\)

\[(2_a, 1_b)\]
\[(3_a, 1_b)\]

Queue \((K+1)\)

\[(3_a, 1_b)\]

\[ c \]

\[
\begin{array}{c}
(0, 0) \\
(1, 1) \\
(2, 1) \\
(0, 1) \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{a:b/0.01} & (1, 1) \\
\text{a:a/0.02} & (1, 1) \\
\text{b:a/0.04} & (0, 1) \\
\text{b:a/0.06} & (2, 1) \\
\text{a:a/0.08} & (2, 1) \\
\text{a:a/0.1} & (3, 1) \\
\text{a:b/0.24} & (3, 1) \\
\text{a:b/0.18} & (3, 2) \\
\text{a:b/0.42} & (3, 3) \\
\end{array}
\]
Composition (example)

**C (3_a,1_b)**

<table>
<thead>
<tr>
<th>Transition</th>
<th>Symbol</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a:a/0.6</td>
<td>3_a</td>
<td></td>
</tr>
<tr>
<td>b:a/0.2</td>
<td>1_b</td>
<td></td>
</tr>
<tr>
<td>a:b/0.3</td>
<td>2_b</td>
<td></td>
</tr>
<tr>
<td>a:b/0.4</td>
<td>3_b</td>
<td></td>
</tr>
<tr>
<td>a:b/0.18</td>
<td>3_a,2_b</td>
<td></td>
</tr>
<tr>
<td>a:b/0.24</td>
<td>3_a,3_b</td>
<td></td>
</tr>
</tbody>
</table>

**Queue (K)**

- (3_a,1_b)

**Queue (K+1)**

- (3_a,2_b)
- (3_a,3_b)
Composition (example)

C \( (3_a, 2_b) \)

\[
\begin{align*}
\text{a} & : \text{a} \rightarrow 0.6 \rightarrow (3_a) \\
\text{b} & : \text{a} \rightarrow 0.5 \rightarrow (3_b)
\end{align*}
\]

Null

Queue \( (K) \)

\( (3_a, 2_b) \)

\( (3_a, 3_b) \)

Queue \( (K+1) \)

\( (3_a, 3_b) \)
Composition (example)

C \((3_a,3_b)\)

<table>
<thead>
<tr>
<th>a:a/0.6 (\rightarrow) ((3_a))</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td></td>
</tr>
</tbody>
</table>

Queue \((K)\)

\((3_a,3_b)\)

Queue \((K+1)\)

Null
Composition (example)

(C) composition result

<table>
<thead>
<tr>
<th>(0,0) → a:b/0.01 → (1,1)</th>
<th>(0,1) → a:a/0.02 → (1,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,1) → a:a/0.04 → (0,1)</td>
<td>(0,1) → b:a/0.08 → (3,1)</td>
</tr>
<tr>
<td>(2,1) → b:a/0.06 → (3,1)</td>
<td>(3,1) → a:a/0.1 → (3,1)</td>
</tr>
<tr>
<td>(3,1) → a:b/0.24 → (3,3)</td>
<td>(3,3) → a:b/0.42 → (3,3)</td>
</tr>
</tbody>
</table>

(c)
Composition (III)

- \(\varepsilon\)-transitions (epsilon) cause problems
  - Wildcard (*) that matches zero or more occurrences of any label in the input/output alphabet
  - Create redundant paths during simple composition
  - May produce incorrect results for non-idempotent semirings as path-weights may be counted more than once

Solution is to utilize an intermediate filter transducer to eliminate the redundant paths
Fig. 7. Redundant $\epsilon$-paths in composition. All transition and final weights are equal to 1. (a) A straightforward generalization of the $\epsilon$-free case would generate all the paths from $(1,1)$ to $(3,2)$ when composing $T_1$ and $T_2$ and produce an incorrect result in non-idempotent semirings. (b) Filter transducer $F$ [46]. The shorthand $x$ is used to represent an element of $\Sigma$. 
**Composition (IV)**

Fig. 7. Redundant \(\epsilon\)-paths in composition. All transition and final weights are equal to 1. (a) A straightforward generalization of the \(\epsilon\)-free case would generate all the paths from (1, 1) to (3, 2) when composing \(T_1\) and \(T_2\) and produce an incorrect results in non-idempotent semirings. (b) Filter transducer \(F\) [46]. The shorthand \(x\) is used to represent an element of \(\Sigma\).
Composition (example)
Composition with epsilons – generic

\[
\begin{align*}
C \ (0_a, 0_b) \\
\text{a:a} &\rightarrow (1_a) \\
\text{a:d} &\rightarrow (1_b) \\
\text{a:d} &\rightarrow (1_a, 1_b)
\end{align*}
\]

Queue (K)

\[(0_a, 0_b)\]

Queue (K+1)

\[(1_a, 1_b)\]
Composition (example)
Composition with epsilons – generic

\[ C \ (0_a, 0_b) \]
\[ a:a \to (1_a) \quad a:d \to (1_b) \]
\[ a:d \to (1_a, 1_b) \]

Queue (\(K\))
\[ (0_a, 0_b) \]

Queue (\(K+1\))
\[ (1_a, 1_b) \]
Composition (example)
Composition with epsilons – generic

\[ C \ (1_a, 1_b) \]

\[
\begin{array}{c|c|c}
\text{b: } & \varepsilon & (2_a) \\
\hline
\varepsilon & (1_a) & \varepsilon : e \rightarrow (2_b) \\
\hline
b: & \varepsilon & (2_a) \\
\hline
& \varepsilon : e \rightarrow (1_a, 1_b) \\
\hline
& \varepsilon : e \rightarrow (2_a, 1_b) \\
\hline
& \varepsilon : e \rightarrow (2_a, 2_b)
\end{array}
\]

Queue \((K)\)

\( (1_a, 1_b) \)

Queue \((K+1)\)

\( (1_a, 2_b) \)

\( (2_a, 1_b) \)

\( (2_a, 2_b) \)
Composition (example)
Composition with epsilons – generic

Epsilon labels can either generate a wildcard match, or act as a null transition. This causes the corresponding WFST to either stay in the current state or skip to the next.
Composition (example)
Composition with epsilons – generic

\[ C \ (1_a, 2_b) \]
- \[ b : \varepsilon \rightarrow (1_a) \]
- \[ d : a \rightarrow (2_b) \]
- \[ b : \varepsilon \rightarrow (2_a, 2_b) \]

Queue (\( K \))
- \( (1_a, 2_b) \)
- \( (2_a, 1_b) \)
- \( (2_a, 2_b) \)

Queue (\( K+1 \))
- \( (2_a, 1_b) \)
- \( (2_a, 2_b) \)
Composition (example)
Composition with epsilons – generic

Composition with epsilons – generic

Queue (K)
(2ₐ,1ₛ)
(2ₐ,2₄)

Queue (K+1)
(2ₐ,2₄)
(3ₐ,1ₙ)

C (2ₐ, 1ₛ)
ε → (2ₐ)
ε:e → (2₄)
c:e → (3ₐ)
ε → (1ₛ)
e:e → (2ₐ, 2₄)
c:e → (3ₐ, 1ₛ)

Composition (example)
Composition (example)
Composition with epsilons – generic

Queue \((K)\)
- \((2_a, 2_b)\)
- \((3_a, 1_b)\)

Queue \((K+1)\)
- \((3_a, 1_b)\)
- \((3_a, 2_b)\)

Composition (example)
Composition with epsilons – generic
Composition (example)
Composition with epsilons – generic

Queue \( (K) \)
- (3_a,1_b)
- (3_a,2_b)

Queue \( (K+1) \)
- (3_a,2_b)

Composition (example)
Composition with epsilons – generic

Graph with transitions and states.
Composition (example)
Composition with epsilons – generic

$$C (3a, 2b)$$

- $$d: d \Rightarrow (4a)$$
- $$d: a \Rightarrow (3b)$$
- $$d: a \Rightarrow (4a, 3b)$$

Queue ($K$)
$$\begin{align*}
(3a, 2b)
\end{align*}$$

Queue ($K+1$)
$$\begin{align*}
(4a, 3b)
\end{align*}$$

Composition with epsilons – generic
Composition (example)
Composition with epsilons - generic
Composition (example)
Composition with epsilons – generic

- Naive application of the algorithm generates many superfluous transitions as well as states
  - Necessary to filter the composition operation in order to prevent the unnecessary generation of these elements
- The final result should only include unique, valid paths
Determinization (I)

- Eliminates ambiguity in the input paths
  - Improves efficiency of downstream operations such as shortest-path
  - Each state has at most one outgoing transition containing any particular input label

- Determinization of weighted automata. (a) Weighted automaton over the tropical semiring $A$. (a') Equivalent weighted automaton a' obtained by determinization of a. (Mohri, 2002)
Determinization (II)

Weighted-Determinization($A$)

1. $i' \leftarrow \{(i, \lambda(i)) : i \in I\}$
2. $\lambda'(i') \leftarrow \top$
3. $Q \leftarrow \{i'\}$
4. while $Q \neq \emptyset$ do
5.   $p' \leftarrow \text{Head}(Q)$
6.   $\text{Dequeue}(Q)$
7.   for each $x \in i[E[Q[p']]])$ do
8.     $w' \leftarrow \bigoplus\{v \otimes w : (p, v) \in p', (p, x, w, q) \in E\}$
9.     $q' \leftarrow \{(q, \bigoplus\{w'^{-1} \otimes (v \otimes w) : (p, v) \in p', (p, x, w, q) \in E\}) : q = n[e], i[e] = x, e \in E[Q[p']]\}$
10.    $E' \leftarrow E' \cup \{(p', x, w', q')\}$
11.   if $q' \not\in Q'$ then
12.     $Q' \leftarrow Q' \cup \{q'\}$
13.     if $Q[q'] \cap F \neq \emptyset$ then
14.       $F' \leftarrow F' \cup \{q'\}$
15.       $\rho'(q') \leftarrow \bigoplus\{v \otimes \rho(q) : (q, v) \in q', q \in F\}$
16.     $\text{Enqueue}(Q, q')$
17. return $T'$
Determinization (II)

**Weighted-Determinization**($A$)

```
1  i' ← {(i, λ(i)) : i ∈ I}
2  λ'(i') ← Π
3  Q ← {i'}
4  while Q ≠ ∅ do
5    p' ← HEAD(Q)
6      DEQUEUE(Q)
7      for each x ∈ i[E[Q[p']]] do
8        w' ← Ω{v ⊗ w : (p, v) ∈ p', (p, x, w, q) ∈ E}
9        q' ← {(q, Ω{w'^{-1} ⊗ (v ⊗ w) : (p, v) ∈ p', (p, x, w, q) ∈ E}) :
                      q = n[e], i[e] = x, e ∈ E[Q[p']]}:
10       E' ← E' ∪ {(p', x, w', q')}
11      if q' ∉ Q' then
12        Q' ← Q' ∪ {q'}
13        if Q[q'] ∩ F ≠ ∅ then
14          F' ← F' ∪ {q'}
15          ρ'(q') ← Ω{v ⊗ ρ(q) : (q, v) ∈ q', q ∈ F'}
16      ENQUEUE(Q, q')
17  return T'
```
Determinization (II)

**Weighted-Determinization** ($A$)

1. $i' \leftarrow \{(i, \lambda(i)) : i \in I\}$
2. $\lambda'(i') \leftarrow \emptyset$
3. $Q \leftarrow \{i'\}$
4. while $Q \neq \emptyset$ do
5.   $p' \leftarrow \text{DEQUEUE}(Q)$
6.   for each $x \in i'[E[Q][p']]$ do
7.     $w' \leftarrow \bigoplus \{v \otimes w : (p, v) \in p', (p, x, w, q) \in E\}$
8.     $q' \leftarrow \{(q, \bigoplus \{w'^{-1} \otimes (v \otimes w) : (p, v) \in p', (p, x, w, q) \in E\}) : q = n[e], i[e] = x, e \in E[Q[p']]\}$
9.     $E' \leftarrow E' \cup \{(p', x, w', q')\}$
10. if $q' \notin Q'$ then
11.    $Q' \leftarrow Q' \cup \{q'\}$
12.    if $Q[q'] \cap F \neq \emptyset$ then
13.       $F' \leftarrow F' \cup \{q'\}$
14.       $\rho'(q') \leftarrow \bigoplus \{v \otimes \rho(q) : (q, v) \in q', q \in F\}$
15. return $T'$

**Algorithm complexity**

<table>
<thead>
<tr>
<th>Time</th>
<th>exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>exponential</td>
</tr>
</tbody>
</table>

Not the same!!

Compute new arc weight

Compute new states and residual weights

Compute new final weight, if necessary
Determinization (example)

Recall the properties of the tropical semiring

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tropical</td>
<td>$\mathbb{R}_+ \cup {+\infty}$</td>
<td>min</td>
<td>+</td>
<td>$+\infty$</td>
<td>0</td>
</tr>
</tbody>
</table>

Queue $(\mathcal{K})$

$(0,1)$
Determinization (example)

Recall the properties of the tropical semiring

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tropical</td>
<td>$\mathbb{R}_+ \cup {+\infty}$</td>
<td>$\text{min}$</td>
<td>$+$</td>
<td>$+\infty$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

Table 1. Semiring examples.

$\oplus \log$ is defined by:

$x \oplus \log y = -\log(e^{-x} + e^{-y})$. 
Determinization (example)

Recall the properties of the tropical semiring

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>⊕</th>
<th>⊗</th>
<th>₀</th>
<th>₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tropical</td>
<td>( \mathbb{R}_+ \cup {+\infty} )</td>
<td>( \text{min} )</td>
<td>+</td>
<td>+∞</td>
<td>0</td>
</tr>
</tbody>
</table>

\( w' = \oplus\{(\overline{1} \otimes 1),(\overline{1} \otimes 2)\} \)

\( w' = \min\{(0 + 1),(0 + 2)\} \)

\( w' = 1 \)

\( q' = \left\{ (1,\min\{-1 + (0 + 1)\}), (2,\min\{-1 + (0 + 2)\}) \right\} \)

\( q' = \left\{ (1,0),(2,1) \right\} \)

\( \{(0,0), a, 1, ((1,0),(2,1))\} \)

\{(0,0), a, 1, ((1,0),(2,1))\}
Determinization (example)

Recall the properties of the tropical semiring

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tropical</td>
<td>$\mathbb{R}_+ \cup {+\infty}$</td>
<td>min</td>
<td>+</td>
<td>+∞</td>
<td>0</td>
</tr>
</tbody>
</table>

$w' = \oplus \{ (1 \otimes 1), (\bar{1} \otimes 2) \}$

$w' = \min \{0 + 1, 0 + 2\}$

$w' = 1$

$q' = \left\{ (1, \oplus \{1^{-1} \otimes (1 \otimes 1)\}), (2, \oplus \{1^{-1} \otimes (\bar{1} \otimes 2)\}) \right\}$

$q' = \left\{ (1, \min \{-1 + (0 + 1)\}), (2, \min \{-1 + (0 + 2)\}) \right\}$

$q' = \{ (1,0),(2,1) \}$

Queue $((0,0),(1,0),(2,1))$
Determinization (example)

**Table 1. Semiring 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>Tropical</td>
<td>(\mathbb{R}_+ \cup {+\infty})</td>
<td>(\min)</td>
<td>+</td>
<td>+(\infty)</td>
<td>0</td>
</tr>
</tbody>
</table>

Recall the properties of the tropical semiring for the Queue ((1,0),(2,1)).

Given the initial state \(q = ((1,0),(2,1))\) and input \(a\) with \(a/(0,0)\) and \(a'/(1,0),(2,1)\), the next states are calculated as follows:

- \(w' = \oplus \{(0 \otimes 3),(1 \otimes 3)\}\)
- \(w' = \min \{0 + 3, 1 + 3\}\)
- \(w' = 3\)

For the queue \(q'\):

- \(q' = \left\{ (1, \min \{-3 + (0 + 3)\}), (2, \min \{-3 + (1 + 3)\}) \right\}\)
- \(q' = \left\{ (1,0),(2,1) \right\}\)

The set for the queue is \(\{(1,0),(2,1), b, 3, (1,0),(2,1)\}\).
Determinization (example)

Recall the properties of the tropical semiring

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tropical</td>
<td>( \mathbb{R}_+ \cup { +\infty } )</td>
<td>( \min )</td>
<td>+</td>
<td>+</td>
<td>0</td>
</tr>
</tbody>
</table>

Queue \((K)\)

\(((1,0),(2,1))\)

\((3,0)\)

\(w' = \oplus \{0 \otimes 5\}\)

\(w' = \min \{0 + 5\}\)

\(w' = 5\)

\(q' = \{3, \oplus \{5^{-1} \otimes (0 \otimes 5)\}\}\)

\(q' = \{3, \min \{-5 + (0 + 5)\}\}\)

\(q' = \{(3,0)\}\)

\{ \(((1,0),(2,1)), c, 5, (3,0) \}\}
Determinization (example)

Recall the properties of the tropical semiring

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tropical</td>
<td>( \mathbb{R}_+ \cup {+\infty} )</td>
<td>( \min )</td>
<td>( + )</td>
<td>( +\infty )</td>
<td>0</td>
</tr>
</tbody>
</table>

Where:
- \( w' = \oplus\{1 \otimes 6\} \)
- \( w' = \min\{1 + 6\} \)
- \( w' = 7 \)
- \( q' = \{3, \ominus\{7^{-1} \otimes (1 \otimes 6)\}\} \)
- \( q' = \{3, \min\{-7 + (1 + 6)\}\} \)
- \( q' = \{(3, 0)\} \)

\( \{ (1,0),(2,1), d, 7, (3,0) \} \)
Determinization (example)

Recall the properties of the tropical *semiring*

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>⊕</th>
<th>⊗</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tropical</td>
<td>$\mathbb{R}_+ \cup {+\infty}$</td>
<td>min</td>
<td>+</td>
<td>+∞</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 1. Semiring examples.**

$\oplus \log$ is defined by:

$$x \oplus \log y = -\log(e^{-x} + e^{-y}).$$

**Queue (K)**

(3,0)

**Null**

(3,0)
Determinization (example)

**a’ determinization result**

<table>
<thead>
<tr>
<th>Start State</th>
<th>Transition</th>
<th>Result State</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td>a/1</td>
<td>(1,0), (2,1)</td>
</tr>
<tr>
<td>(1,0), (2,1)</td>
<td>b/3</td>
<td>(1,0), (2,1)</td>
</tr>
<tr>
<td>(1,0), (2,1)</td>
<td>c/5</td>
<td>(2^{a}, 1^{b})</td>
</tr>
<tr>
<td>(1,0), (2,1)</td>
<td>d/7</td>
<td>(3^{a}, 1^{b})</td>
</tr>
</tbody>
</table>
Determinization (III)

- Some issues with weighted determinization
  - May not terminate for some inputs!
  - May not finish for some inputs, even if they are valid!
  - May blow up in memory

The Twins property can be utilized to test for determinizability

\[ w(P(q,y,q)) = w(P(q',y,q')) \]
Determinization (III)

- Some issues with weighted determinization
  - May not terminate for some inputs!
  - May not finish for some inputs, even if they are valid!
  - May blow up in memory

- The *Twins property* can be utilized to test for determinizability

\[ w(P(q, y, q)) = w(P(q', y, q')) \]

"Weighted Automata Algorithms", Mehryar Mohri, 2002 for details
Generic Epsilon-Removal (I)

- Removes $\varepsilon$-transitions from the input transducer and produces a new, epsilon-free transducer equivalent to the original
- Preferable to remove $\varepsilon$-transitions prior to performing downstream processing or optimization
  - Introduce delays
  - Cause problems or complications
Generic Epsilon-Removal (II)

$$
\text{Algorithm complexity*}
\begin{array}{|c|c|}
\hline
\text{Time} & O(V^2 \log V + V \cdot E) \\
\text{Space} & O(E) \\
\hline
\end{array}
\]

V = #states, E = #arcs.
*Tropical semiring www.OpenFST.org

**Epsilon-Removal**\((T)\)

1. **for** each \(p \in Q\) **do**
2. \quad \text{Compute-Closure}(C(p))
3. \quad \(E' \leftarrow \overline{E}_\epsilon \leftarrow \{(p, a, b, w, q) \in E : (a, b) \neq (\epsilon, \epsilon)\}\)
4. \quad \(F' \leftarrow F\)
5. \quad \(\rho' \leftarrow \rho\)
6. **for** each \(p \in Q\) **do**
7. \quad **for** each \((q, w') \in C'[p]\) **do**
8. \quad \quad \(E'[p] \leftarrow E'[p] \cup \{(p, a, b, w' \otimes w, r) : (q, a, b, w, r) \in \overline{E}_\epsilon\}\)
9. \quad **if** \(q \in F\) **then**
10. \quad \quad **if** \(p \not\in F\) **then**
11. \quad \quad \quad \(F' \leftarrow F \cup \{p\}\)
12. \quad \quad \quad \(\rho'[p] \leftarrow \overline{0}\)
13. \quad \quad \(\rho'[p] \leftarrow \rho'[p] \oplus (w' \otimes \rho(q))\)
14. \quad **return** \(T' = (\Sigma, \Delta, Q, I, F', E', \lambda, \rho')\)

- \(\mathcal{E}\)-closure computation of state \(p\):
  \[C(p) = \{(q, w) : P(p, \epsilon, q) \neq \emptyset, w = d_\epsilon[p, q]\}\]
Generic Epsilon-Removal (II)

**Algorithm complexity**

<table>
<thead>
<tr>
<th>Time</th>
<th>$O(V^2 \log V + V \cdot E)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>$O(V \cdot E)$</td>
</tr>
</tbody>
</table>

$V = \# \text{states}, \ E = \# \text{arcs}$.

*Tropical semiring [www.OpenFST.org]

**Epsilon-Removal($T$)**

1. for each $p \in Q$ do
2. \quad \text{Compute-Closure}(C(p)) \quad \text{\{Compute the e-closure for each state, $q$\}}
3. \quad $E' \leftarrow \overline{E}_\epsilon \leftarrow \{(p, a, b, w, q) \in E : (a, b) \neq (\epsilon, \epsilon)\}$
4. \quad $F' \leftarrow F$ \quad \text{\{Initialize the new WFST with all non-epsilon arcs and states\}}
5. \quad $\rho' \leftarrow \rho$
6. for each $p \in Q$ do
7. \quad for each $(q, w') \in C[p]$ do
8. \quad \quad $E'[p] \leftarrow E'[p] \cup \{(p, a, b, w' \otimes w, r) : (q, a, b, w, r) \in \overline{E}_\epsilon\}$
9. \quad if $q \in F$ then
10. \quad \quad if $p \notin F$ then
11. \quad \quad \quad $F' \leftarrow F \cup \{p\}$
12. \quad \quad \quad $\rho'[p] \leftarrow 0$
13. \quad \quad \quad $\rho'[p] \leftarrow \rho'[p] \oplus (w' \otimes \rho(q))$
14. return $T' = (\Sigma, \Delta, Q, I, F', E', \lambda, \rho')$

- $\epsilon$-closure computation of state $p$:
  \[ C(p) = \{(q, w) : P(p, \epsilon, q) \neq \emptyset, w = d_\epsilon[p, q]\} \]
  \text{\{Epsilon-closure of $p$ is the set of states reachable from $p$ via epsilon transitions\}}
Generic Epsilon-Removal (II)

**Algorithm complexity**

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(O(V^2\log V + V \cdot E))</td>
<td>(O(V \cdot E))</td>
</tr>
</tbody>
</table>

**Epsilon-Removal** \((T)\)

1. for each \(p \in Q\) do
2. \(\text{Compute-Closure}(C(p))\) \{Compute the \(e\)-closure for each state, \(q\)\}
3. \(E' \leftarrow \overline{E_e} \leftarrow \{(p, a, b, w, q) \in E : (a, b) \neq (e, e)\}\)
4. \(F' \leftarrow F\) \{Initialize the new WFST with all non-epsilon arcs and states\}
5. \(\rho' \leftarrow \rho\)
6. for each \(p \in Q\) do
7. \(\text{for each} \ (q, w') \in C'[p]\) do
8. \(E'[p] \leftarrow E'[p] \cup \{(p, a, b, w' \otimes w, r) : (q, a, b, w, r) \in \overline{E_e}\}\)
9. if \(q \in F\) then
10. if \(p \not\in F\) then
11. \(F' \leftarrow F \cup \{p\}\)
12. \(\rho'[p] \leftarrow 0\)
13. \(\rho'[p] \leftarrow \rho'[p] \oplus (w' \otimes \rho(q))\)
14. return \(T' = (\Sigma, \Delta, Q, I, F', E', \lambda, \rho')\)

- \(e\)-closure computation of state \(p\):
  \(C(p) = \{(q, w) : P(p, e, q) \neq \emptyset, w = d_e[p, q] \}\)

- Epsilon-closure of \(p\) is the set of states reachable from \(p\) via epsilon transitions
Generic Epsilon-Removal (example)
Copy all states, and non-epsilon arcs from the input transducer
For each state in the original input transducer, compute the epsilon-closure – the states that are accessible via epsilon-arcs.

Epsilon closure of state 1 is depicted in yellow. Arc (1, b, b, 3, 4) is not included as it is not an epsilon arc.
Generate new, non-epsilon arcs based on the epsilon closure. Similarly, generate new final states where necessary.
Generic Epsilon-Removal (example)
Generic Epsilon-Removal (example)
Generic Epsilon-Removal (example)
Generic Epsilon-Removal (example)
Generic Epsilon-Removal (example)
States 2 and 3 are not co-accessible. These may be deleted along with associated transitions.
Generic Epsilon-Removal (example)
Generic Epsilon-Removal (III)

- Epsilon removal may be employed to remove arbitrary symbols other than $\varepsilon$,
  - Relabel the $\varepsilon$ to an alternative, map other desired symbols to epsilon, perform generic epsilon removal, and restore the original $\varepsilon$ labels
- Choice of semiring affects the time complexity of the algorithm.
  - Tropical semiring is faster because it takes advantage of the \textit{viterbi approximation} for the shortest path computation
  - Other semirings exhibit \textbf{exponential} time complexity
  - Un-weighted epsilon-removal exhibits time complexity $O(V^2+VE)$
Other algorithms

- **Minimization**
  - Given an input WFST, produces a ‘minimal’ version which is guaranteed to have the smallest possible number of states while preserving the input language and weight/path properties of the original.

- **Projection**
  - Turns a transducer into an acceptor by projecting either just the input language or just the output language.

- **Weight-pushing**
  - Pushes arc weights forward or backward, accumulating and/or distributing them where appropriate according to the semiring.
Limitations

- Weighted Finite-State Transducers provide a powerful and flexible framework for performing many operations however they do have some limitations
  - Expressive power
    WFSTs are equivalent in terms of representational power to regular languages thus they suffer from the same limitations. In particular there are certain grammars which cannot be represented by WFSTs, e.g.,
    \[ B^N A^N \]
    where \( N \) is not specified at compile time. Although this may be approximated for various values of \( N \) it cannot be explicitly represented by a WFST or a regular expression
  - Computing resources
    Large static graphs such as those often employed for LVCSR tasks may require large amounts of memory, space, and time to construct, compute, and store.