OpenFst: a General and Efficient Weighted Finite-State Transducer Library

Part I. *Library Design and Use*

Outline

▷ 1. Definitions
   - Semirings
   - Weighted Automata and Transducers

2. Library Overview
   - FST Construction
   - FST Component Classes
   - FST Operations

3. Library Design
   - Weight Class Design
   - FST Class Design
   - FST Operation Design

Weight Sets: Semirings

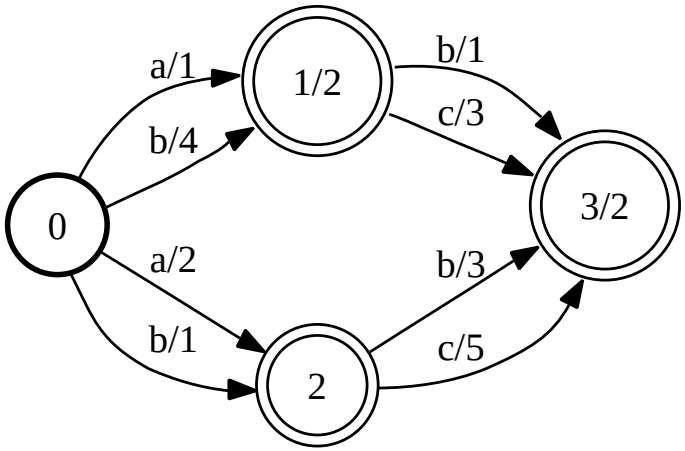
A *semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ = a ring that may lack negation.

- 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).

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	$\Sigma^* \cup \{\infty\}$	$\wedge$	$\cdot$	∞	ϵ

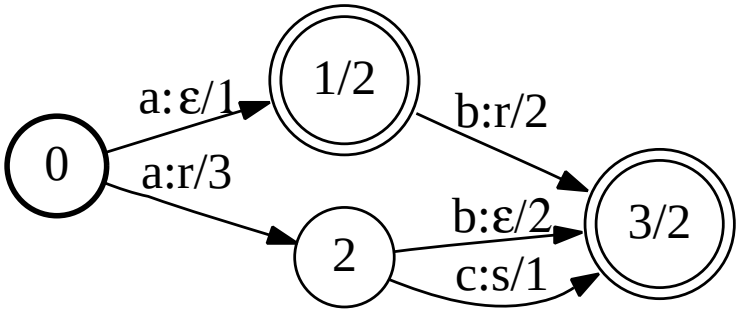
$\oplus_{\log}$ is defined by: $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$ and $\wedge$ is longest common prefix. The string semiring is a *left semiring*.

Weighted Automaton/Acceptor



Probability semiring $(\mathbb{R}_+, +, \times, 0, 1)$	Tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$
$[\![A]\!](ab) = 14$	$[\![A]\!](ab) = 4$
$(1 \times 1 \times 2 + 2 \times 3 \times 2 = 14)$	$(\min(1 + 1 + 2, 3 + 2 + 2) = 4)$

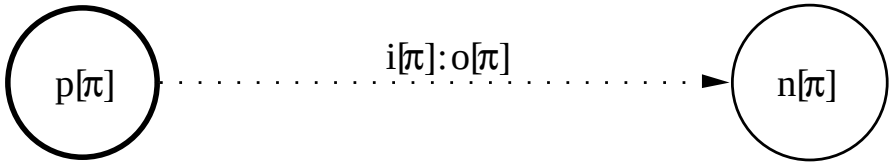
Weighted Transducer



Probability semiring $(\mathbb{R}_+, +, \times, 0, 1)$	Tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$
$[\![T]\!](ab, r) = 16$	$[\![T]\!](ab, r) = 5$
$(1 \times 2 \times 2 + 3 \times 2 \times 2 = 16)$	$(\min(1 + 2 + 2, 3 + 2 + 2) = 5)$

Definitions and Notation – Paths

- Path π
  - Origin or previous state: $p[\pi]$.
  - Destination or next state: $n[\pi]$.
  - Input label: $i[\pi]$.
  - Output label: $o[\pi]$.



- 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 Σ, output Δ.
   - 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 $\lambda : I \to \mathbb{K}$

     final weight function $\rho : F \to \mathbb{K}$.

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 $x \in \Sigma^*, y \in \Delta^*$:

$$[\![T]\!](x, y) = \bigoplus_{\pi \in P(I,x,y,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

Outline

1. Definitions
   - Semirings
   - Weighted Automata and Transducers
2. ▷ Library Overview
   - FST Construction
   - FST Component Classes
   - FST Operations
3. Library Design
   - Weight Class Design
   - FST Class Design
   - FST Operation Design

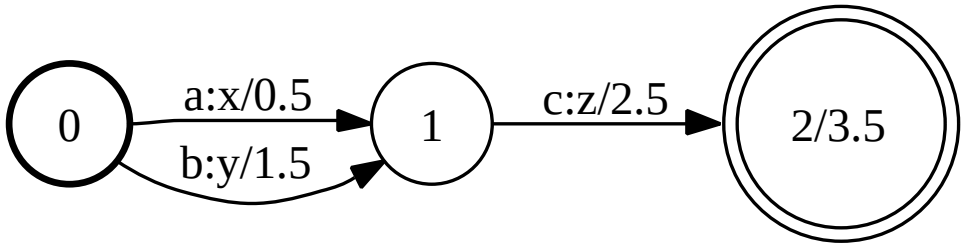
Finite-State Transducer Construction

Input Methods:

- Finite-state transducer:
  - Textual transducer file representation
  - C++ code
  - Graphical user interface
- Regular expressions
- "Context-free" rules
- "Context-dependent" rules

FST Textual File Representation

- Graphical Representation (`T.ps`)



- Transducer File (`T.txt`)

```
0   1   a   x   0.5
0   1   b   y   1.5
1   2   c   z   2.5
2   3.5
```

Input Symbols File (`T.isyms`)

```
a   1
b   2
c   3
```

Output Symbols File (`T.osyms`)

```
x   1
y   2
z   3
```

Compiling, Printing, Reading, and Writing FSTs

- Compiling

```
fstcompile -isymbols=T.isyms -osymbols=T.osyms T.txt T.fst
```

- Printing

```
fstprint -isymbols=T.isyms -osymbols=T.osyms T.fst >T.txt
```

- Drawing

```
fstdraw -isymbols=T.isyms -osymbols=T.osyms T.fst >T.dot
```

- Reading

```
Fst<Arc> *fst = Fst<Arc>::Read(``T.fst'')
```

- Writing

```
fst.Write(``T.fst'')
```

C++ FST Construction

```
// A vector FST is a general mutable FST
VectorFst<StdArc> fst;

// Add state 0 to the initially empty FST and make it the start state
fst.AddState(); // 1st state will be state 0 (returned by AddState)
fst.SetStart(0); // arg is state ID

// Add two arcs exiting state 0
// Arc constructor args:  ilabel, olabel, weight, dest state ID
fst.AddArc(0, StdArc(1, 1, 0.5, 1)); // 1st arg is src state ID
fst.AddArc(0, StdArc(2, 2, 1.5, 1));

// Add state 1 and its arc
fst.AddState();
fst.AddArc(1, StdArc(3, 3, 2.5, 2));

// Add state 2 and set its final weight
fst.AddState();
fst.SetFinal(2, 3.5); // 1st arg is state ID, 2nd arg weight
```

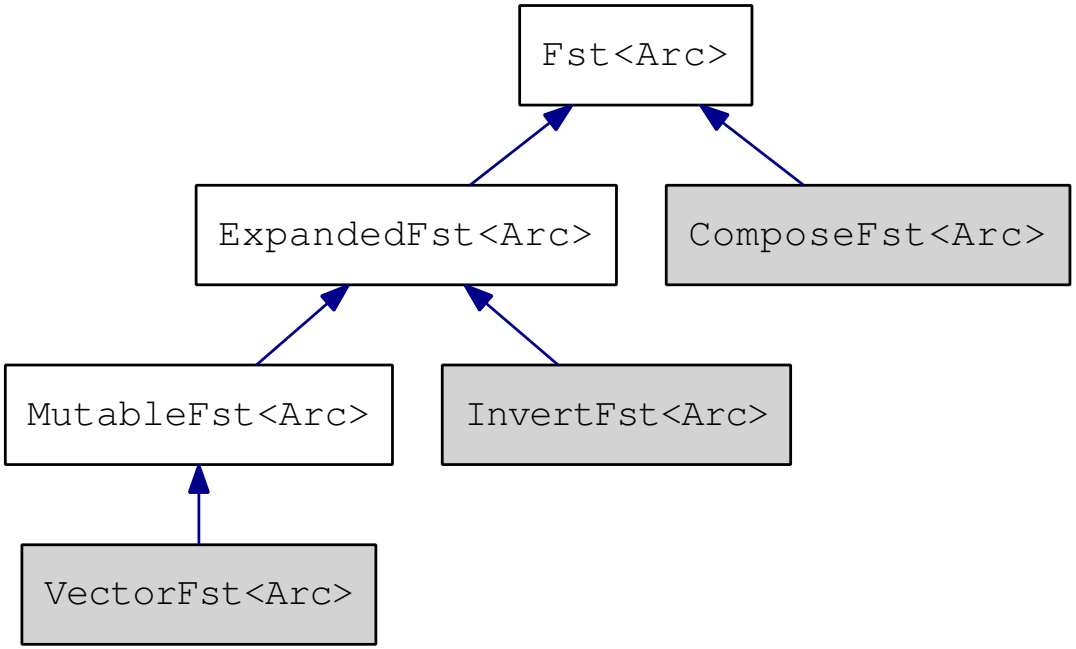
OpenFst Design: Arc (transition)

Labels and states may be any integral type; weights may be any class that forms a semiring:

```
struct StdArc {
  typedef int Label;
  typedef TropicalWeight Weight;
  typedef int StateId;

  Label ilabel;                    // Transition input label
  Label olabel;                    // Transition output label
  Weight weight;                   // Transition weight
  StateId nextstate;               // Transition destination state
};
```

OpenFst Design: Fst class hierarchy



- Virtual interfaces: `Fst<Arc>`, `ExpandedFst<Arc>`, `MutableFst<Arc>`
- Concrete classes: `VectorFst<Arc>`, `ComposeFst<Arc>`, `InvertFst<Arc>`, ...
- Typedefs for `StdArc`: `StdFst`, `StdMutableFst`, `StdVectorFst`, ...

Operation Implementation Types

- Constructive: Writes to output; $O(|Q| + |E|)$:

```
StdFst *input = StdFst::Read("input.fst");
StdVectorFst output;
Reverse(*input, &output);
```

- Destructive: Modifies input; $O(|Q| + |E|)$:

```
StdMutableFst *input = StdMutableFst::Read("input.fst");
Invert(input);
```

- Lazy (or Delayed): Creates new Fst; $O(|Q_{visit}| + |E_{visit}|)$:

```
StdFst *input = StdFst::Read("input.fst");
StdFst *output = new StdInvertFst(*input);
```

Lazy implementations are useful in applications where the whole machine may not be visited, e.g. Dijsktra (positive weights), pruned search.

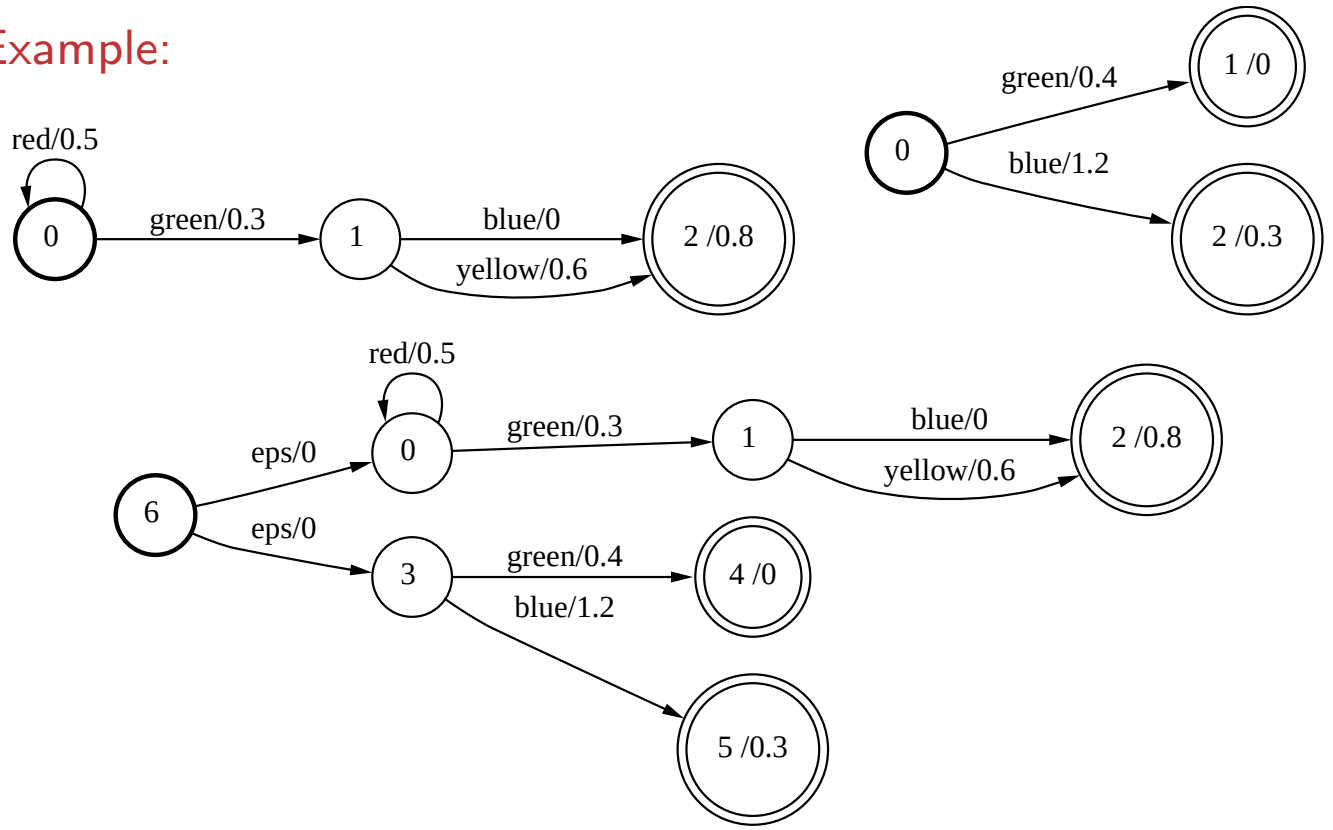
Rational Operations

Operation	Usage	Description
Union (Sum)	Union(&A, B); UnionFst<Arc>(A, B); fstunion a.fst b.fst out.fst	contains strings in A and B
Concat (Product)	Concat(&A, B); Concat(A, &B); ConcatFst<Arc>(A, B); fstconcat a.fst b.fst out.fst	contains strings in A followed by B
Closure	Closure(&A, type); ClosureFst<Arc>(A, type); fstclosure in.fst out.fst	$A^* = \{\epsilon\} \cup A \cup AA \cup \ldots$

Union (Sum)

- Definition: $[\![T_1 \oplus T_2]\!](x, y) = [\![T_1]\!](x, y) \oplus [\![T_2]\!](x, y)$
- Usage: `Union(&A, B)` `fstunion a.fst b.fst out.fst`

  `UnionFst<Arc>(A, B)`
- Example:

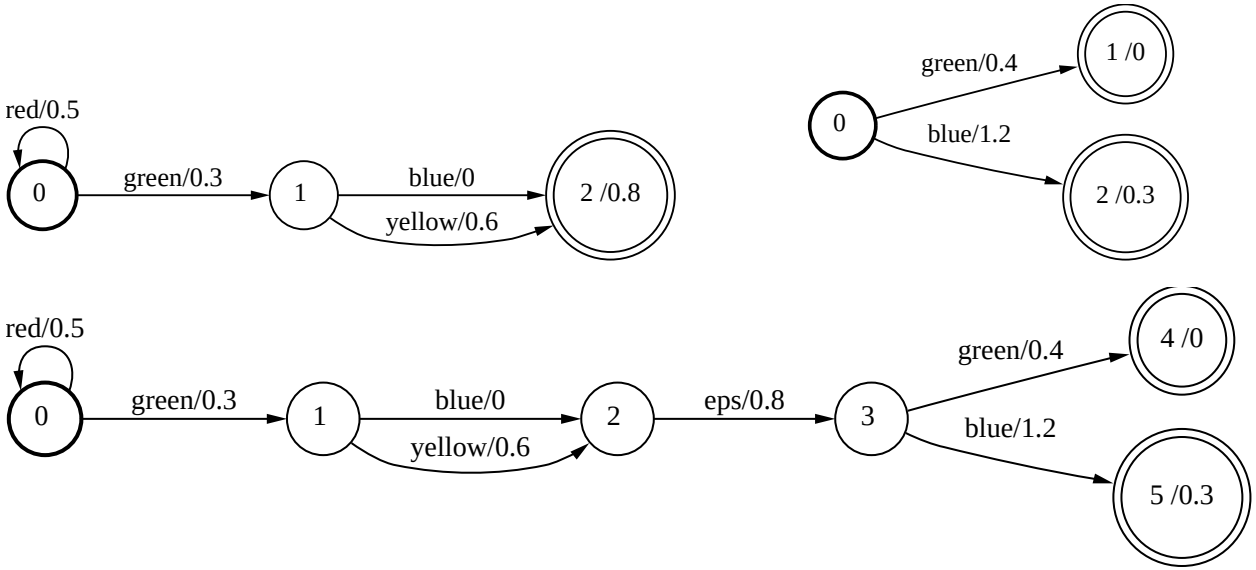


Concat (Product)

- Definition: $[\![T_1 \otimes T_2]\!](x, y) = \bigoplus_{x=x_1 x_2, y=y_1 y_2} [\![T_1]\!](x_1, y_1) \otimes [\![T_2]\!](x_2, y_2)$
- Usage:

```
Concat(&A, B)                    fstconcat a.fst b.fst out.fst
Concat(A, &B)
ConcatFst<Arc>(A, B)
```

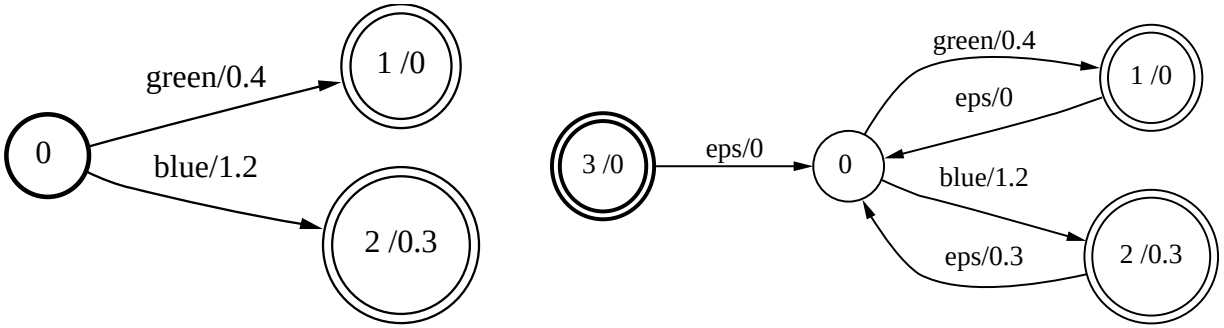
- Example:



Closure

- Definition: $[\![T^*]\!](x, y) = \bigoplus_{n=0}^{\infty} [\![T^n]\!](x, y)$
- Usage: `Closure(&A)` `fstclosure a.fst out.fst`

  `ClosureFst<Arc>(A)`
- Example:

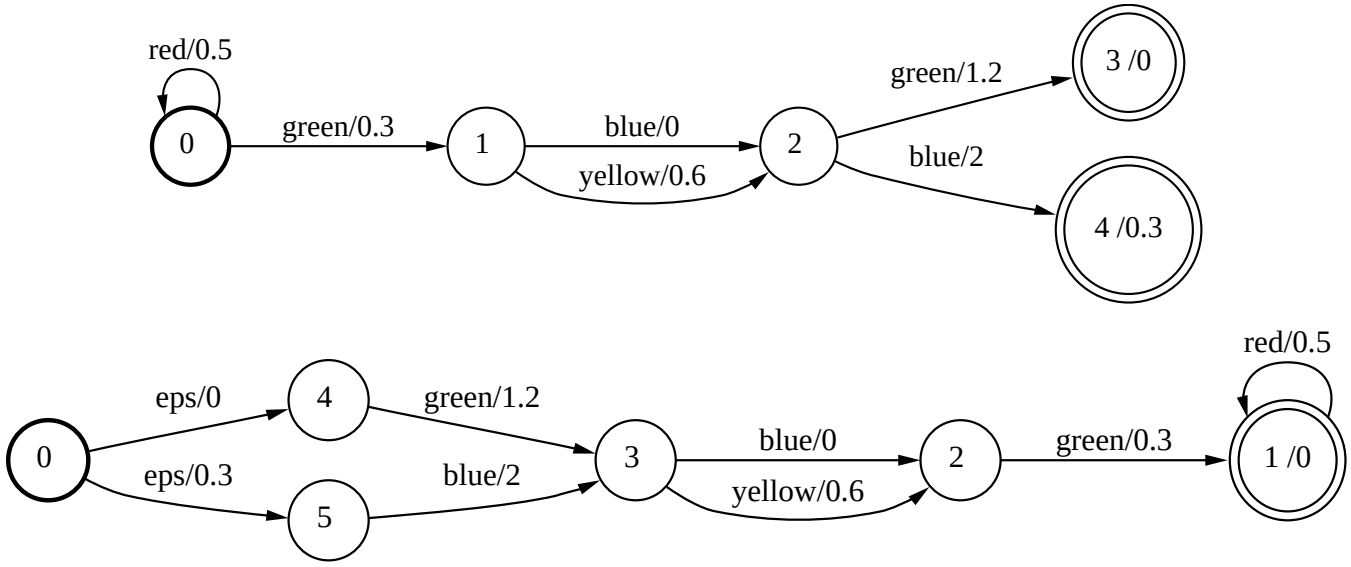


Elementary Unary Operations

Operation	Usage	Description
Reverse	Reverse(A, &B);	reversed strings of A
Invert	Invert(&A); InvertFst<Arc>(A); fstinvert in.fst out.fst	inverse binary relation; swaps input and output labels
Project	Project(&A, type); ProjectFst<Arc>(A, type); fstproject [–project_output] in.fst out.fsa	creates acceptor of just the input or output strings

Reverse

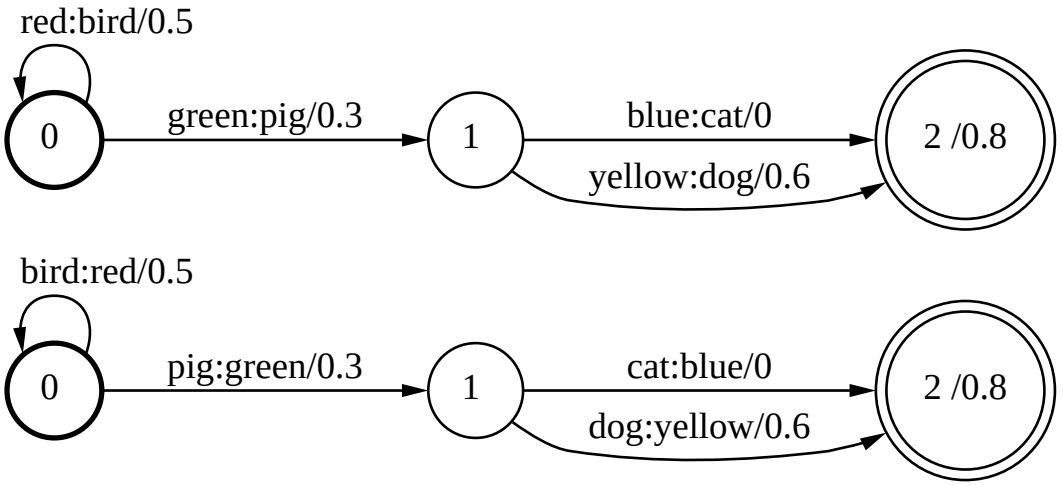
- Definition: $[\![\widetilde{T}]\!](x, y) = [\![T]\!](\widetilde{x}, \widetilde{y})$
- Usage: `Reverse(A, &B)` `fstreverse a.fst out.fst`
- Example:



Invert

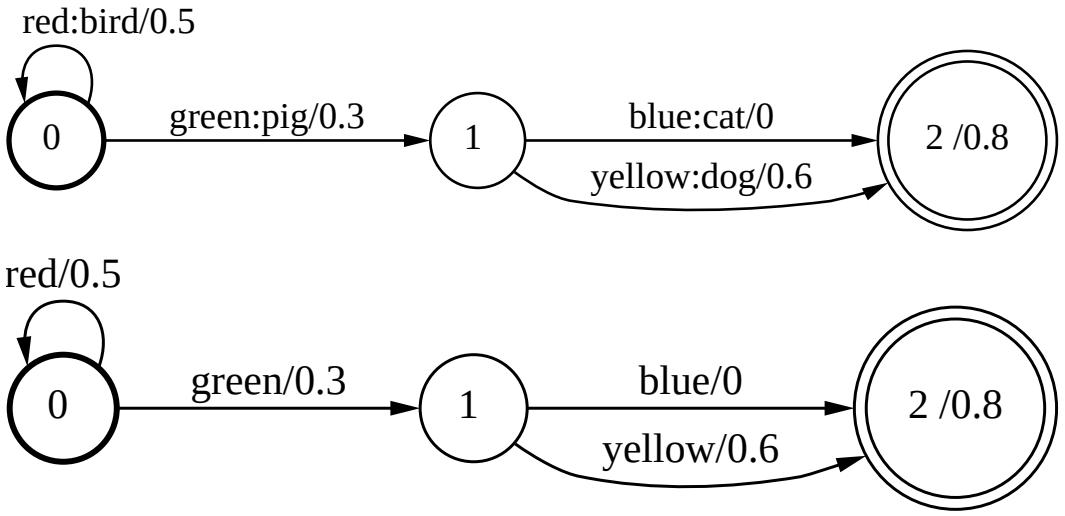
- Definition: $[\![T^{-1}]\!](x, y) = [\![T]\!](y, x)$
- Usage: `Invert(&A)` `fstinvert a.fst out.fst`

  `InvertFst<Arc>(A)`
- Example:



Project

- Definition: $[\![\Pi_1(T)]\!](x) = \bigoplus_y [\![T]\!](x, y)$
- Usage: `Project(&A, type)` `fstproject [--project_output] a.fst out.fst`
  `ProjectFst<Arc>(A, type)`
- Example:



Fundamental Binary Operations

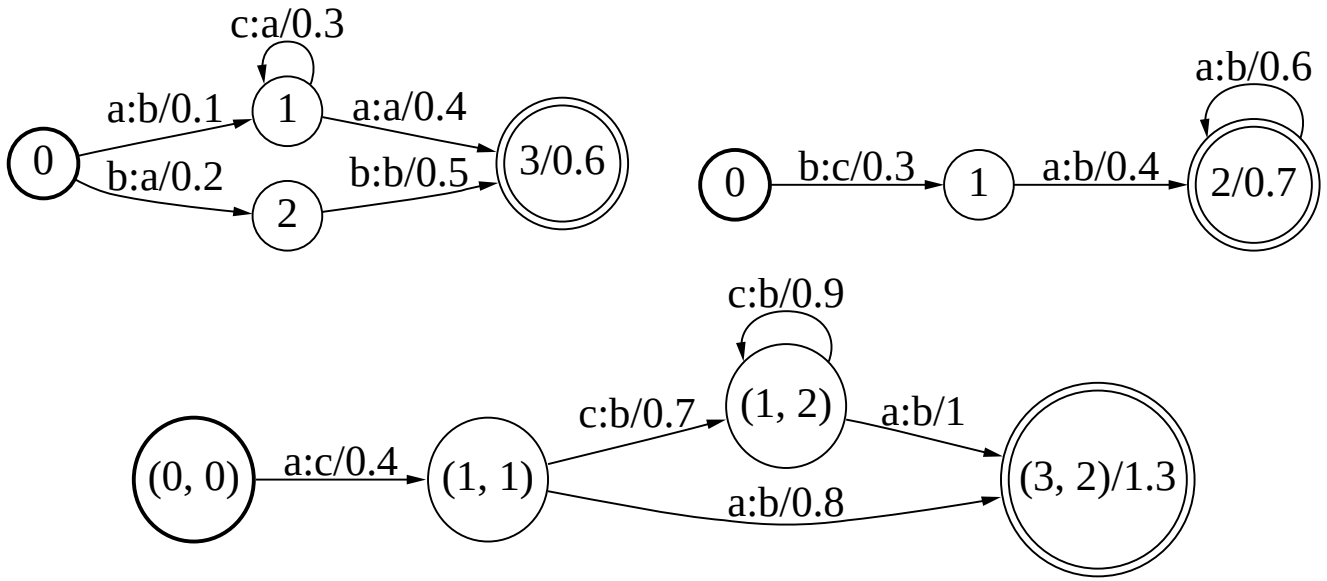
Operation	Usage	Description
Compose	Compose(A, B, &C); ComposeFst<Arc>(A, B); fstcompose a.fst b.fst out.fst	composition of binary relations
Intersect	Intersect(A, B, &C); IntersectFst<Arc>(A, B); fstintersect a.fsa b.fsa out.fsa	contains strings in both A and B
Difference	Difference(A, B, &C); DifferenceFst<Arc>(A, B); fstdifference a.fsa b.dfa out.fsa	contains strings in A but not in B; B unweighted

Compose

- Definition: $[\![T_1 \circ T_2]\!](x, y) = \bigoplus_z [\![T_1]\!](x, z) \otimes [\![T_2]\!](z, y)$
- Usage:

```
Compose(A, B, &C)          fstcompose a.fst b.fst out.fst
ComposeFst<Arc>(A, B)
```

- Example:



- Algorithm:
  States: (q_1, q_2) with q_1 in T_1 and q_2 in T_2
  Transitions: (q_1, a, b, w_1, q_1') and $(q_2, b, c, w_2, q_2') \rightsquigarrow ((q_1, q_2), a, c, w_1 \otimes w_2, (q_1', q_2'))$
- Condition: Commutative semiring

Compose – Pseudocode

COMPOSITION(T_1, T_2)

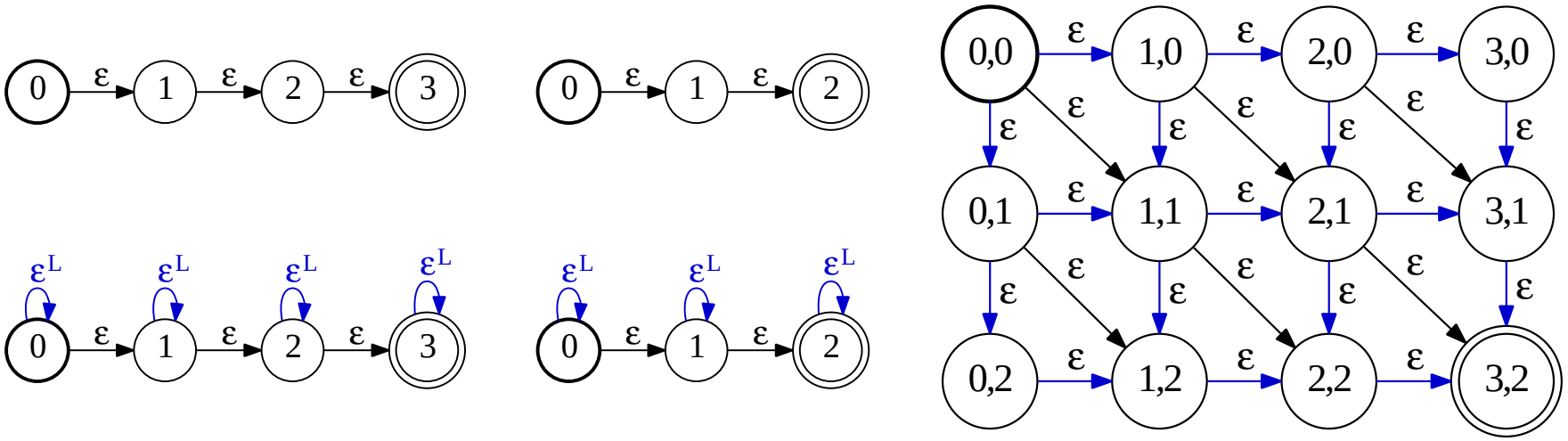
1 $S \leftarrow Q \leftarrow I_1 \times I_2$
2 **while** $S \neq \emptyset$ **do**
3 $\quad (q_1, q_2) \leftarrow$ HEAD(S)
4 $\quad$ DEQUEUE(S)
5 $\quad$ **if** $(q_1, q_2) \in I_1 \times I_2$ **then**
6 $\quad\quad I \leftarrow I_1 \times I_2$
7 $\quad\quad \lambda(q_1, q_2) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$
8 $\quad$ **if** $(q_1, q_2) \in F_1 \times F_2$ **then**
9 $\quad\quad F \leftarrow F \cup \{(q_1, q_2)\}$
10 $\quad\quad \rho(q_1, q_2) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2)$
11 $\quad$ **for** each(e_1, e_2) such that $o[e_1] = i[e_2]$ **do**
12 $\quad\quad$ **if** $(n[e_1], n[e_2]) \notin Q$ **then**
13 $\quad\quad\quad Q \leftarrow Q \cup \{(n[e_1], n[e_2])\}$
14 $\quad\quad\quad$ ENQUEUE$(S, (n[e_1], n[e_2]))$
15 $\quad\quad E \leftarrow E \cup \{((q_1, q_2), i[e_1], o[e_2], w[e_1] \otimes w[e_2], (n[e_1], n[e_2]))\}$
16 **return** $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$

Compose – Handling ϵ-transitions

- An ϵ-transition in T_1 (resp. T_2) can be matched in T_2 (resp. T_1) by an ϵ transition or by staying in the same state

▷ As if there were an ϵ self-loop at each state in T_1 and T_2
→ labeled ϵ^L

T_1 T_2 $T_1 \circ T_2$

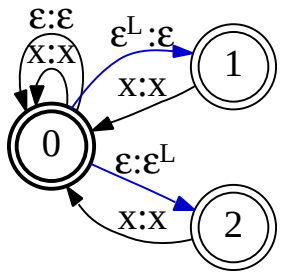


- Issue: Results in redundant ϵ-paths in $T_1 \circ T_2$

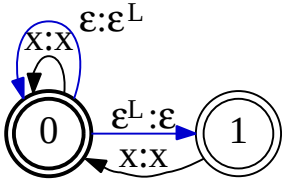
→ [Mohri, Peirera and Riley, 96] used a *filter transducer*

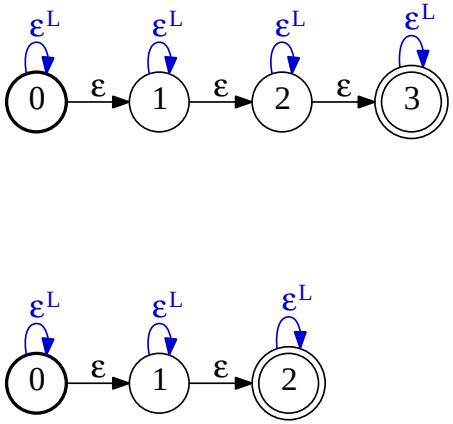
Compose – Epsilon Filters

Matching filter F_{match}

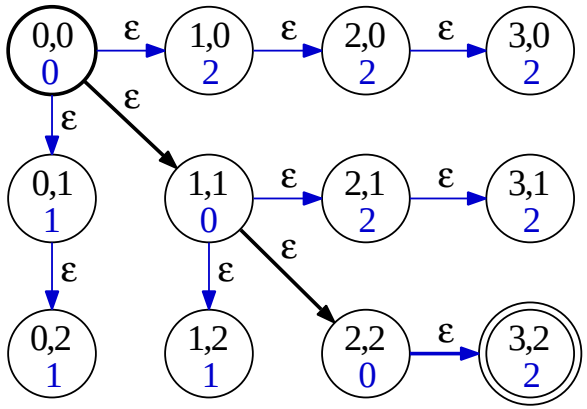


Sequencing filter F_{seq}

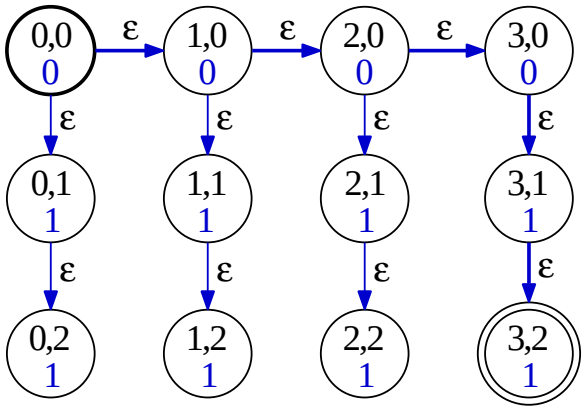




$\tilde{T}_1$ and $\tilde{T}_2$



$\tilde{T}_1 \circ F_{\text{match}} \circ \tilde{T}_2$



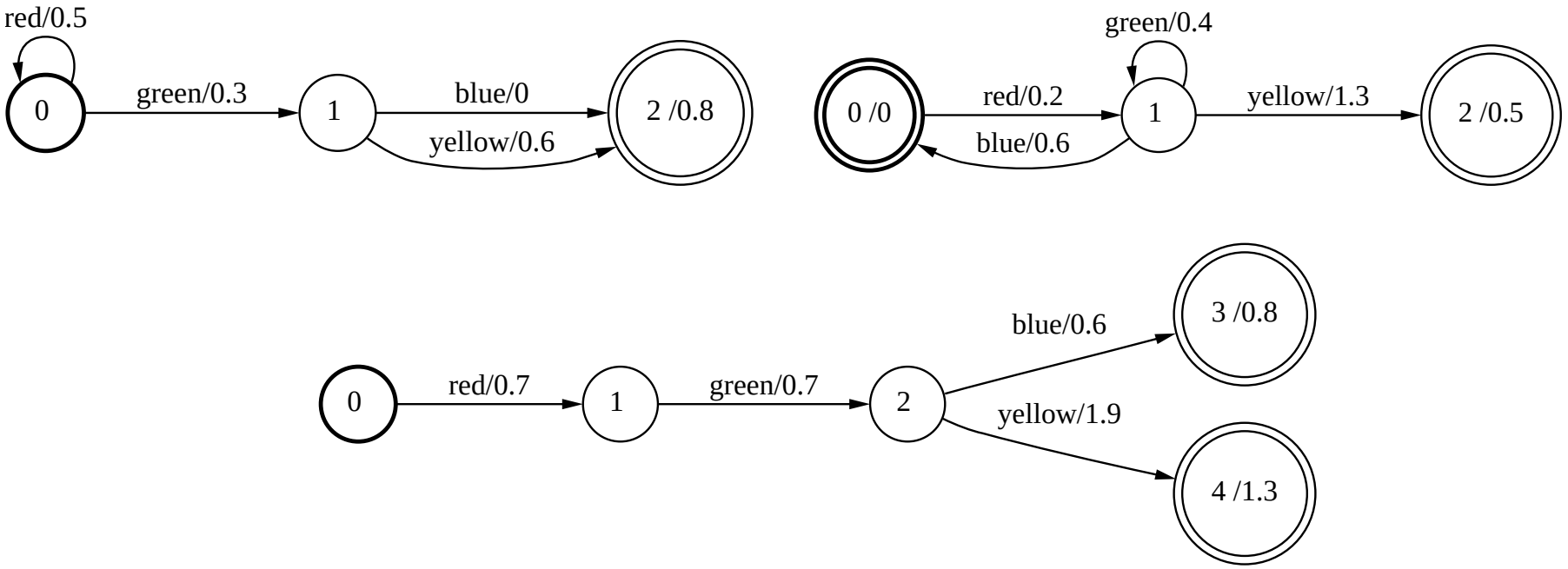
$\tilde{T}_1 \circ F_{\text{seq}} \circ \tilde{T}_2$

Remarks:

- Filters implemented as functors
- Generalized to handle more than ϵ's
  - ▷ look-ahead filters

Intersect

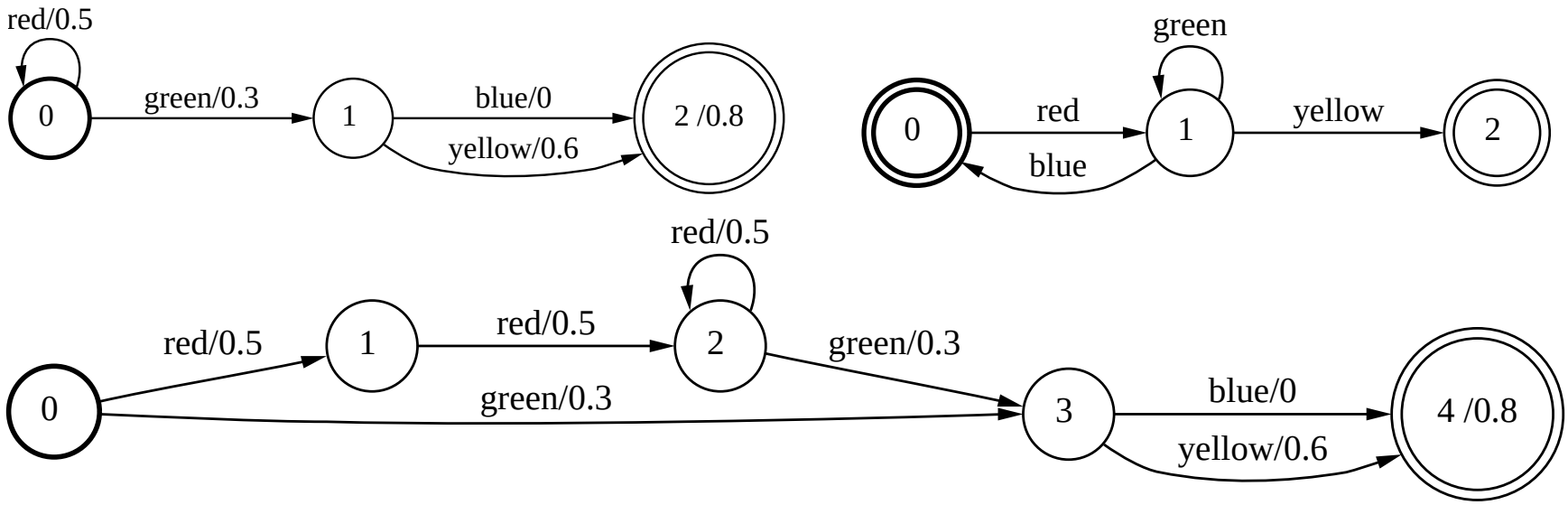
- Definition: $[\![A_1 \cap A_2]\!](x) = [\![A_1]\!](x) \otimes [\![A_2]\!](x)$
- Usage: `Intersect(A, B, &C)` `fstintersect a.fst b.fst out.fst`
  `IntersectFst<Arc>(A, B)`
- Example:



- Condition: Commutative semiring

Difference

- Definition: $[\![A_1 - A_2]\!](x) = [\![A_1 \cap \overline{A_2}]\!](x)$
- Usage: `Difference(A, B, &C)` `fstdifference a.fst b.fst out.fst`
  `DifferenceFst<Arc>(A, B)`
- Example:



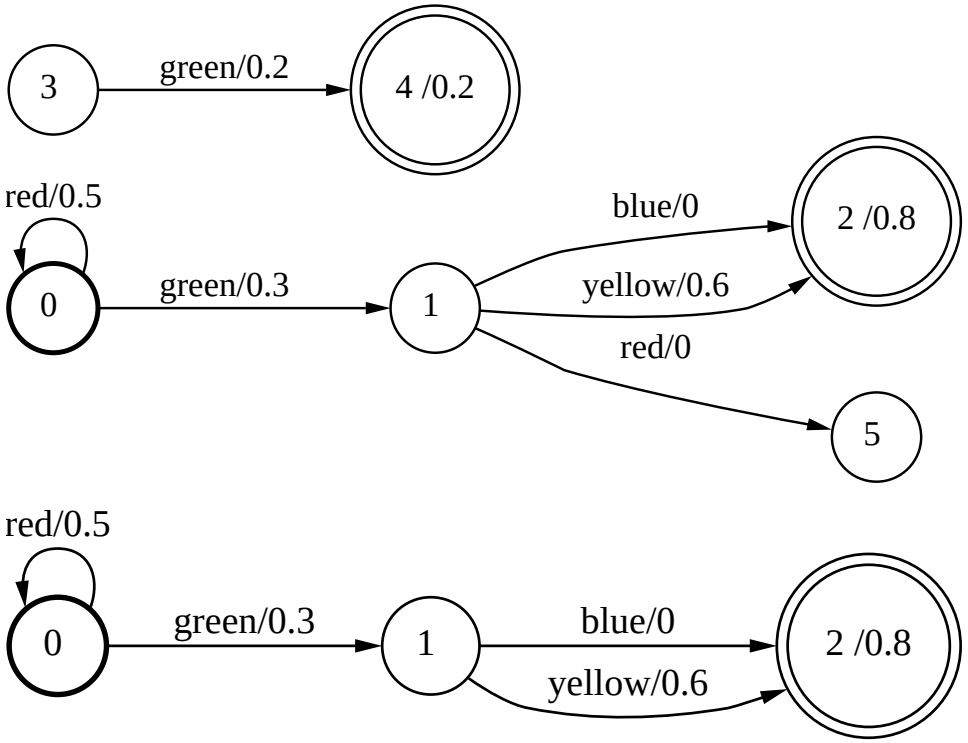
- Condition: A_2 must be deterministic and unweighted

Optimization Operations

Operation	Usage	Description
Connect	Connect(&A); fstconnect in.fst out.fst	Removes useless states and arcs
RmEpsilon	RmEpsilon(&A); RmEpsilonFst<Arc>(A); fstrmepsilon in.fst out.fst	Equiv. ϵ-free transducer
Determinize	Determinize(A, &B); DeterminizeFst<Arc>(A); fstdeterminize in.fst out.fst	Equiv. deteterministic transducr
Minimize	Mininize(&A); Mininize(&A, &B); fstminimize in.fst out1.fst [out2.fst]	Equiv. minimal det. transducer

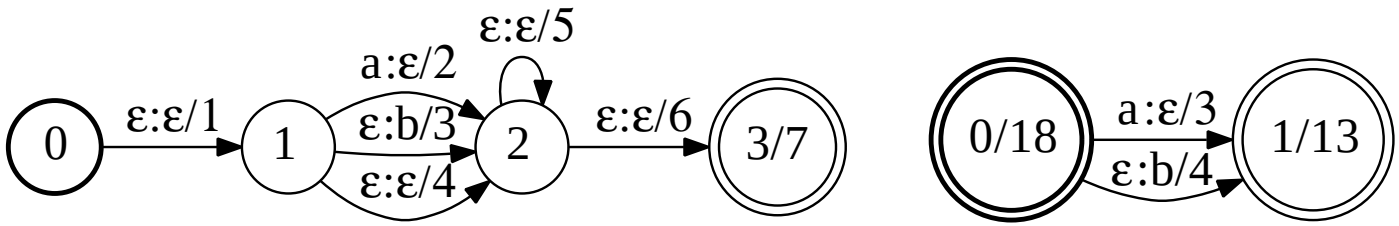
Connect

- Definition: Removes non-accessible/non-coaccessible states
- Usage: `Connect(&A)` `fstconnect a.fst out.fst`
- Example:



RmEpsilon

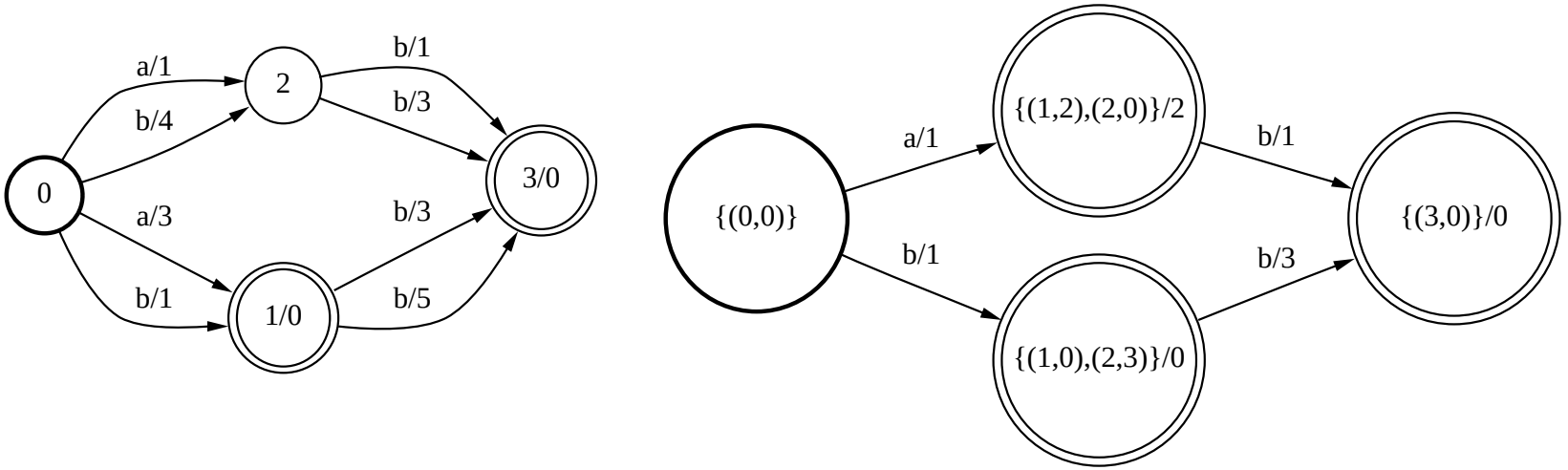
- Definition: Removes ϵ-transitions
- Usage: `RmEpsilon(&A)` `fstrmepsilon a.fst out.fst`
  `RmEpsilonFst<Arc>(A)`
- Example:



Determinize

- Definition: Creates an equivalent deterministic machine
- Usage: `Determinize(A, &B)` `fstdeterminize a.fst out.fst`

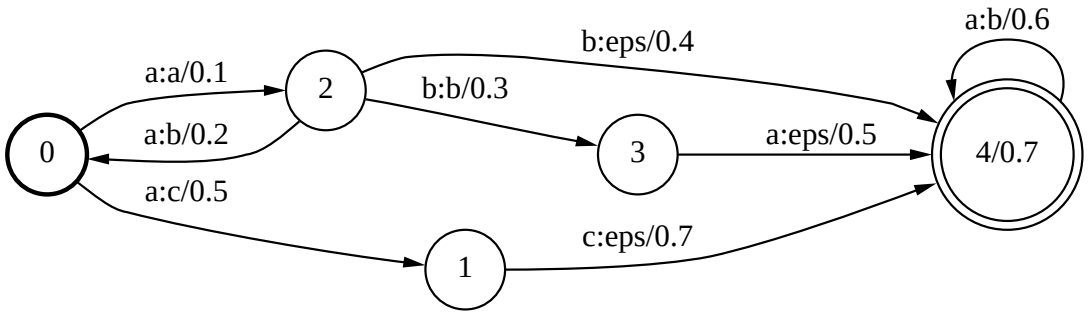
  `DeterminizeFst<Arc>(A)`
- Example (Automaton):

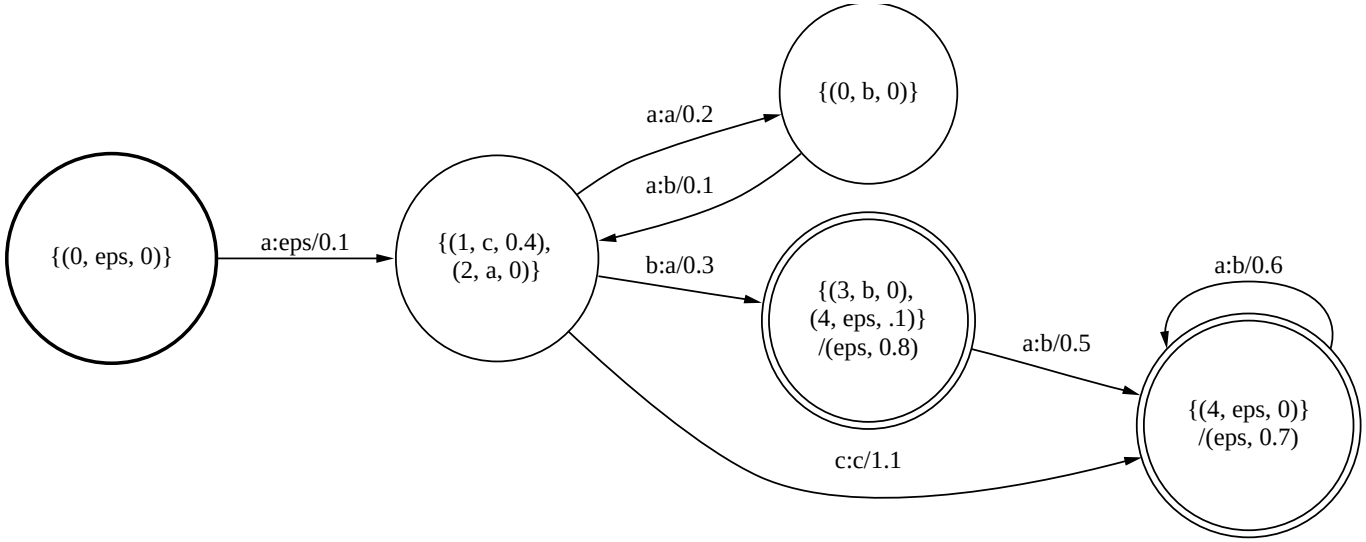


- Caveats: may not terminate, exponential complexity in the worse case

Determinize

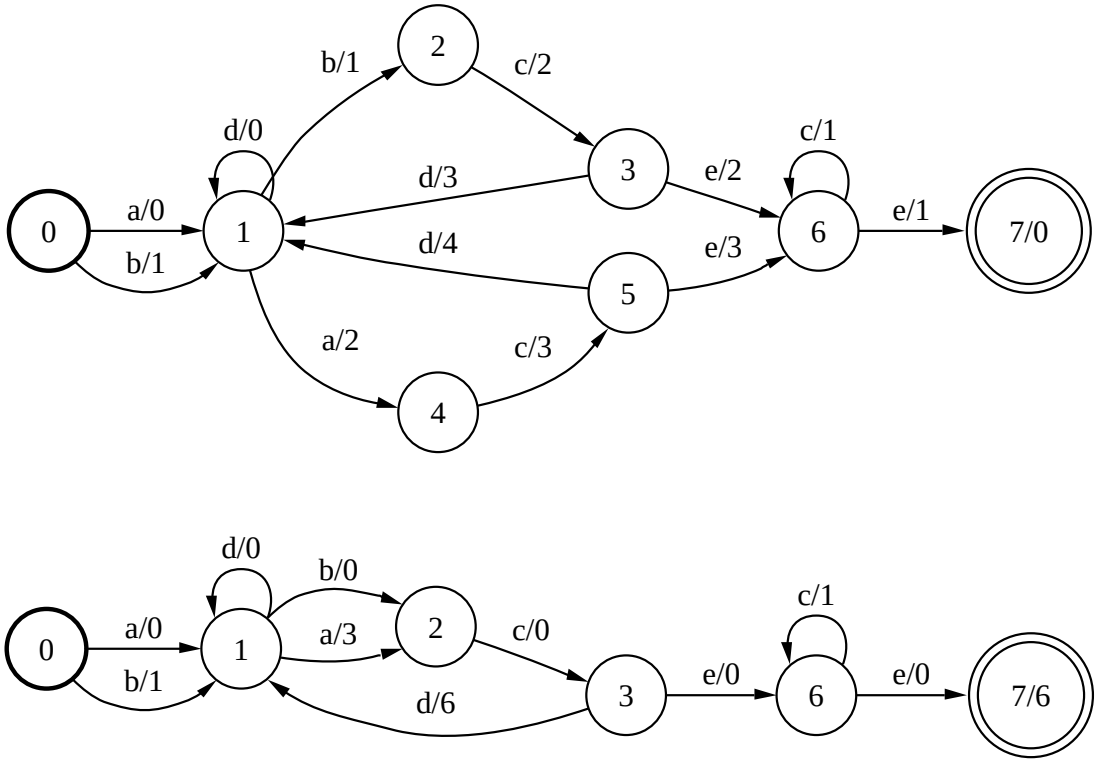
- Example (Transducer):





Minimize

- Definition: Computes a minimal equivalent deterministic machine
- Usage: `Minimize(&A)` `fstminimize a.fst out.fst`
- Example:



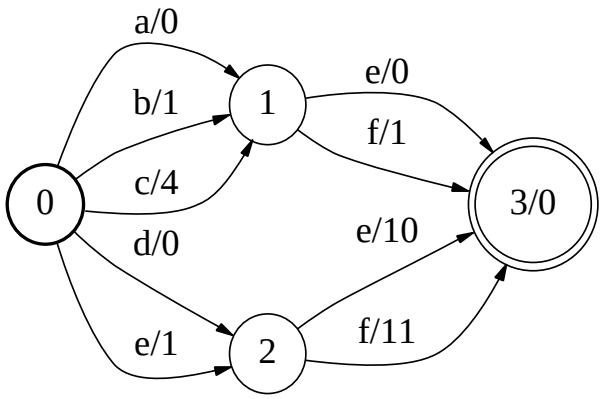
- Condition: Input must be deterministic

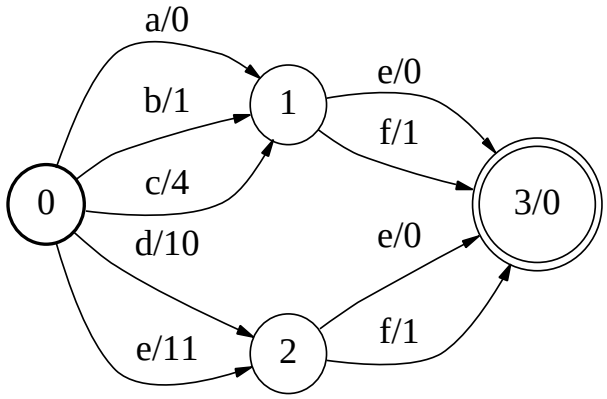
Normalization Operations

Operation	Usage	Description
TopSort	TopSort(&A); fsttopsort in.fst out.fst	Topologically sorts an acyclic FST
ArcSort	ArcSort(&A, compare; fstarcsort [–sort-type=$t] in.fst out.fst	Sorts state's arcs given an order rela
Push	Push<Arc, Type>(&A, flags); fstpush [-flags] in.fst out.fst	Creates equiv. pushed/stochastic FS
EpsNormalize	EpsNormalize(A, &B, type); fstepsnormalize [–eps_norm_output] in.fst out.fst	Places path ϵ's after non-ϵ's
Synchronize	Synchronize(A, &B); SynchronizeFst<Arc>(A); fstsynchronize in.fst out.fst	Produces monotone epsilon delay

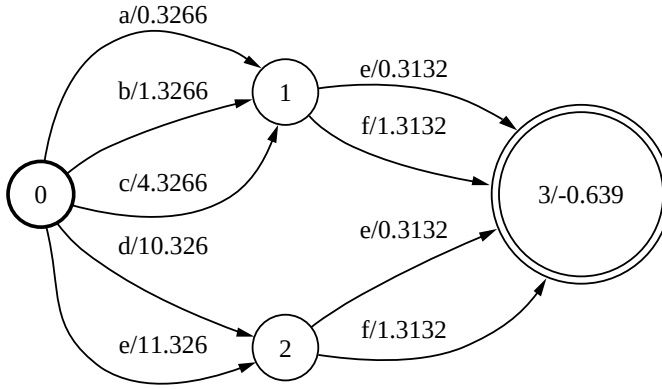
Push – Weight Pushing

- Definition: Creates an equivalent pushed/stochastic machine
- Usage: `Push<Arc, type>(&A, flags)` `fstpush [--flags] a.fst out.fst`
- Example:





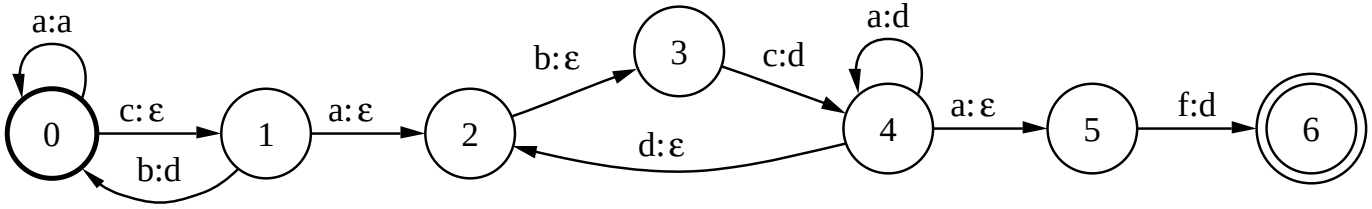
Tropical semiring

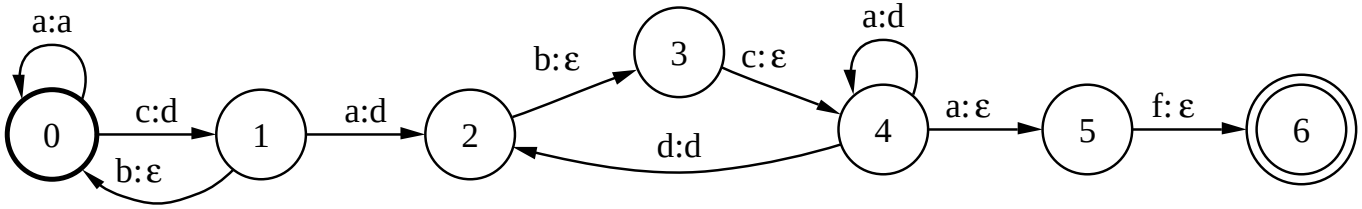


Log semiring

Push – Label Pushing

- Definition: Minimizes at each state the length of the common prefix of output labels of all outgoing paths at that state.
- Example:



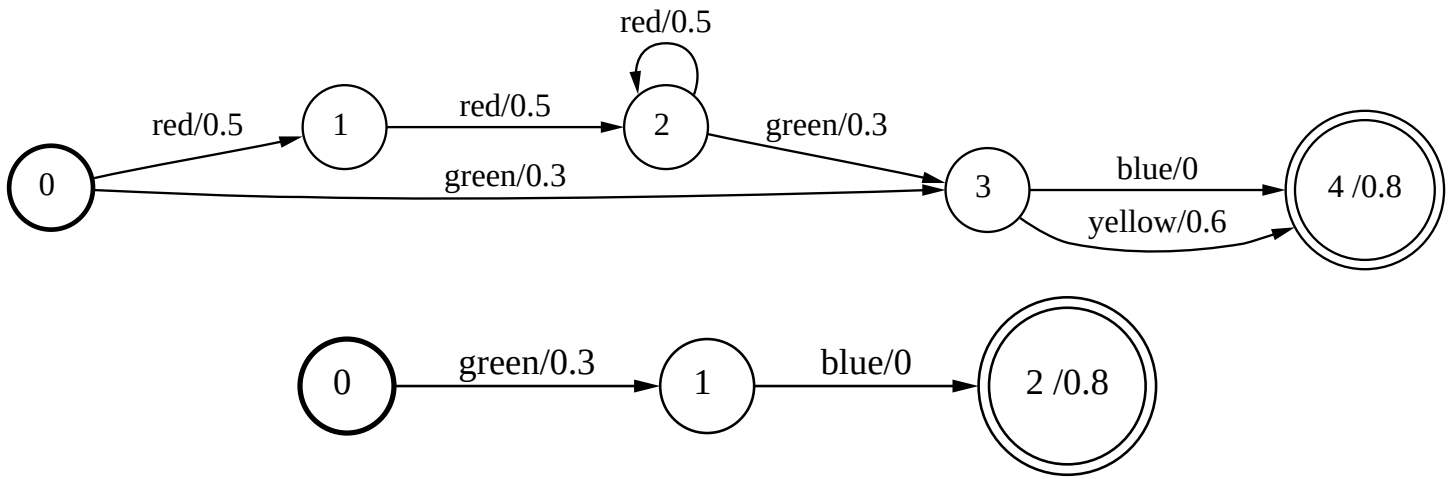


Search Operations

Operation	Usage	Description
ShortestPath	ShortestPath(A, &B, nshortest=1); fstshortestpath [–nshortest=$n] in.fst out.fst	N-shortest paths
ShortestDistance	ShortestDistance(A, &distance); ShortestDistance(A, &distance, true); fstshortestdistance [–reverse] in.fst [dist.txt]	Shortest distance from initial states Shortest distance to final states
Prune	Prune(&A, threshold); fstprune [–weight=$w] in.fst out.fst	Prunes states and arcs by path weight

ShortestPath

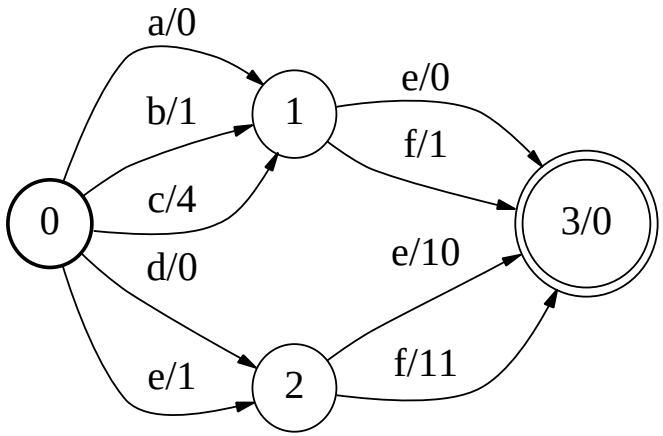
- Definition: Computes the N-shortest paths in the input machine
- Usage: `ShortestPath(A, &B, n=1)` `fstshortestpath [--nshortest=$n] a.fst out.fst`
- Example:



- Condition: Semiring needs to have the path property: $a \oplus b \in \{a, b\}$ (e.g. tropical semiring)

ShortestDistance

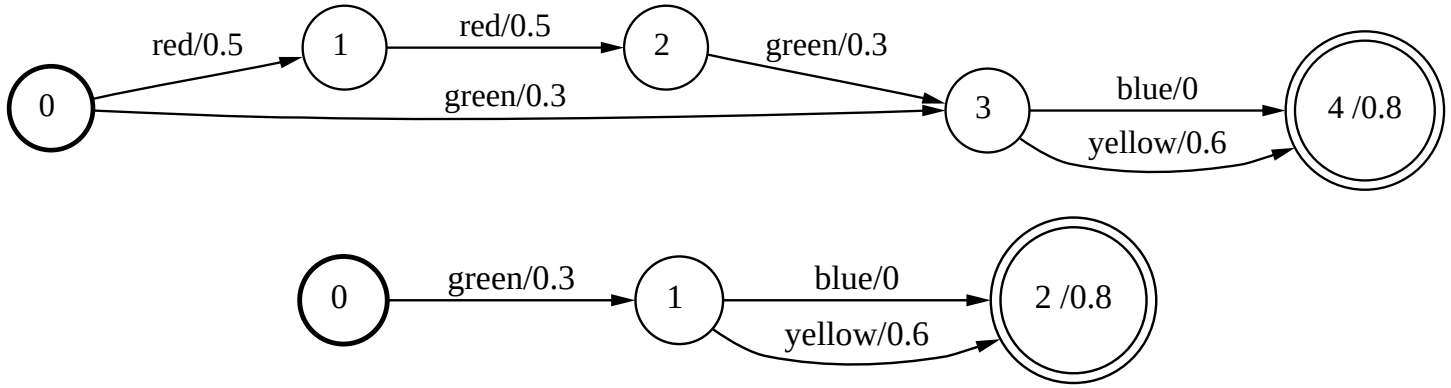
- Definition: $d[q] = \bigoplus_{\pi \in P(I,q)} \lambda(p[\pi]) \otimes w[\pi]$ ▷ from initial state(s)
  $\overline{d}[q] = \bigoplus_{\pi \in P(q,F)} w[\pi] \otimes \rho(n[\pi])$ ▷ to final states
- Usage: `ShortestDistance(A, &d, reverse)` `fstshortestdistance [--reverse] a.fst d.txt`
- Examples:



  - Tropical semiring: $d[2] = 0$ and $\overline{d}[2] = 10$
  - Log semiring: $d[2] = 0 \oplus_{\log} 1 = .368$ and $\overline{d}[2] = 10 \oplus_{\log} 11 = 9.69$
- Condition: Right semiring from initial states, left semiring to final states.

Prune

- Definition: Removes any paths which weight is more than the shortest-distance $\otimes$-multiply by a specified threshold w
- Usage: `Prune(&A, w)` `fstprune [--weight=$w] a.fst out.fst`
- Example:



- Condition: Semiring needs to be commutative and have the path property: $a \oplus b \in \{a, b\}$ (e.g. tropical semiring)

Traversal Operations

Operation	Usage	Description
Map	Map(&A, mapper); Map(A, &B, mapper); MapFst<IArc, OArc, Mapper>(A, mapper);	Transforms arcs in an FST
Visit	Visit(A, &visitor, &queue);	Visits FST using queue disc.

- Mapper: Class with method:
  - `OArc operator()(const IArc &arc));`
- Visitor: Class with methods e.g.:
  - `bool WhiteArc(StateId s, const Arc &arc));`,
  - `bool GreyArc(StateId s, const Arc &arc));`,
  - `bool BlackArc(StateId s, const Arc &arc));`
- Queue: Class with methods e.g.:
  - `bool Enqueue(StateId s);`
  - `StateId Head();`
  - `void Dequeue();`
  - `bool Empty();`

Example: FST Application - Shell-Level

```
# The FSTs must be sorted along the dimensions they will be joined.
# In fact, only one needs to be so sorted.
# This could have instead been done for "model.fst" when it was created.
$ fstarcsort --sort_type=olabel input.fst input_sorted.fst
$ fstarcsort --sort_type=ilabel model.fst model_sorted.fst

# Creates the composed FST
$ fstcompose input_sorted.fst model_sorted.fst comp.fst

# Just keeps the output label
$ fstproject --project_output comp.fst result.fst

# Do it all in a single command line
$ fstarcsort --sort_type=ilabel model.fst |
fstcompose input.fst - | fstproject --project_output result.fst
```

Example: FST Application - C++

```
// Reads in an input FST.
StdFst *input = StdFst::Read("input.fst");

// Reads in the transduction model.
StdFst *model = StdFst::Read("model.fst");

// The FSTs must be sorted along the dimensions they will be joined.
// In fact, only one needs to be so sorted.
// This could have instead been done for "model.fst" when it was created.
ArcSort(input, StdOLabelCompare());
ArcSort(model, StdILabelCompare());

// Container for composition result.
StdVectorFst result;

// Create the composed FST
Compose(*input, *model, &result);

// Just keeps the output labels
Project(&result, PROJECT_OUTPUT);
```

Outline

1. Definitions
   - Semirings
   - Weighted Automata and Transducers
2. Library Overview
   - FST Construction
   - FST Component Classes
   - FST Operations
3. ▷ Library Design
   - Weight Class Design
   - FST Class Design
   - FST Operations Design

OpenFst Design: Tropical Weight

A Weight class holds the set element and provides the semiring operations:

```
class TropicalWeight {
public:
  TropicalWeight(float f) :  value_(f) {}
  static TropicalWeight Zero() { return TropicalWeight(kPositiveInfinity); }
  static TropicalWeight One() { return TropicalWeight(0.0); }
private:
  float value_;
};

TropicalWeight Plus(TropicalWeight x, TropicalWeight y) {
   return w1.value_ < w2.value_ ?  w1 :  w2;
};
```

Similarly, e.g. `LogWeight` and `MinMaxWeight` are defined.

OpenFst Design: Product Weight

This template allows easily creating the product semiring from two (or more) semirings.

```
template <typename W1, typename W2>
class ProductWeight {
public:
  ProductWeight(W1 w1, W2 w2) :  value1_(w1), value2_(w2) {}
  static ProductWeight<W1, W2> Zero() {
    return ProductWeight(W1::Zero(), W2::Zero());
  }
  static ProductWeight<W1, W2> One() {
    return ProductWeight(W1::One(), W2::One());
  }
private:
  float value1_;
  float value2_;
};
```

```
template <typename W1, typename W2>
ProductWeight<W1, W2> Plus(ProductWeight<W1, W2> x, ProductWeight<W1, W2> y)
  return ProductWeight<W1, W2>(
    Plus(x.value1_, y.value1_),
    Plus(x.value2_, y.value2_));
  };
```

Similarly, e.g. `LexicographicWeight` is defined.

Example: Shortest-Distance with Various Semirings

- Tropical Semiring:

```
Fst<StdArc> *input = Fst<StdArc>::Read("input.fst");
vector<StdArc::Weight> distance;
ShortestDistance(*input, &distance);
```

- Log Semiring:

```
Fst<LogArc> *input = Fst::Read("input.fst");
vector<LogArc::Weight> distance;
ShortestDistance(*input, &distance);
```

- Right String Semiring:

```
typedef StringArc<TropicalWeight, STRING_RIGHT> SA;
Fst<SA> *input = Fst::Read("input.fst");
vector<SA::Weight> distance;
ShortestDistance(*input, &distance);
```

- Left String Semiring:

```
ERROR: ShortestDistance:  Weights need to be right distributive
```

Example: Expectation Semiring

Let $\mathbb{K}$ denote $(\mathbb{R} \cup \{+\infty, -\infty\}) \times (\mathbb{R} \cup \{+\infty, -\infty\})$. For pairs (x_1, y_1) and (x_2, y_2) in $\mathbb{K}$, define the following :

$$\begin{aligned}(x_1, y_1) \oplus (x_2, y_2) &= (x_1 + x_2, y_1 + y_2) \\ (x_1, y_1) \otimes (x_2, y_2) &= (x_1 x_2, x_1 y_2 + x_2 y_1)\end{aligned}$$

The system $(\mathbb{K}, \oplus, \otimes, (0, 0), (1, 0))$ defines a commutative semiring.

This semiring combined with the composition and shortest-distance algorithms has be used e.g. to compute the relative entropy between probabilistic automata [C. Cortes, M. Mohri, A. Rastogi, and M. Riley. On the Computation of the Relative Entropy of Probabilistic Automata. *International Journal of Foundations of Computer Science*, 2007.]:

$$D(A \| B) = \sum_x [\![A]\!](x) \log [\![A]\!](x) - \sum_x [\![A]\!](x) \log [\![B]\!](x).$$

This algorithm is trivially implemented in the OpenFst Library.

OpenFst Design: Fst (generic)

```
template <class Arc>
class Fst {
public:
  virtual StateId Start() const = 0;                // Initial state
  virtual Weight Final(StateId) const = 0;          // State's final weight
  static Fst<Arc> *Read(const string filename);
}
```

OpenFst Design: State Iterator

```
template <class F>
class StateIterator {
public:
  explicit StateIterator(const F &fst);
  virtual ~StateIterator();
  virtual bool Done();                    // States exhausted?
  virtual StateId Value() const;          // Current state Id
  virtual void Next();                    // Advance a state
  virtual void Reset();                   // Start over
}
```

OpenFst Design: Arc Iterator

```
template <class F>
class ArcIterator {
public:
  explicit ArcIterator(const F &fst, StateId s);
  virtual ~ArcIterator();
  virtual bool Done();                          // Arcs exhausted?
  virtual const Arc &Value() const;             // Current arc
  virtual void Next();                          // Advance an arc
  virtual void Reset();                         // Start over
  virtual void Seek(size_t a);                  // Random access
}
```

OpenFst Design: MutableFst

```
template <class Arc>
class MutableFst :  public Fst<Arc> {
public:
  void SetStart(StateId s);                  // Set initial state
  void SetFinal(StateId s, Weight w);        // Set final weight
  void AddState();                           // Add a state
  void AddArc(StateId s, const Arc &arc);    // Add an arc
}
```

OpenFst Design: Mutable Arc Iterator

```
template <class F>
class MutableArcIterator {
public:
  explicit MutableArcIterator(F *fst, StateId s);
  virtual ~MutableArcIterator();
  virtual bool Done();                          // Arcs exhausted?
  virtual const Arc &Value() const;             // Current arc
  virtual void Next();                          // Advance an arc
  virtual void Reset();                         // Start over
  virtual void Seek(size_t a);                  // Random access
  virtual void SetValue(const Arc &arc);        // Set current arc
}
```

OpenFst Design: Invert (Destructive)

```
template <class Arc> void Invert(MutableFst<Arc> *fst) {
  for (StateIterator< MutableFst<Arc> > siter(*fst);
    !siter.Done();
    siter.Next()) {
      StateId s = siter.Value();
      for (MutableArcIterator< MutableFst<Arc> > aiter(fst, s);
        !aiter.Done();
        aiter.Next()) {
          Arc arc = aiter.Value();
          Label l = arc.ilabel;
          arc.ilabel = arc.olabel;
          arc.olabel = l;
          aiter.SetValue(arc);
      }
  }
}
```

Easier to use `Map` for this case.

OpenFst Design: Invert (Lazy)

```
template <class Arc> class InvertFst :  public Fst<Arc> {
public:
  virtual StateId Start() const { return fst_->Start(); }
  ...
private:
  const Fst<Arc> *fst_;
}

template <class F> Arc ArcIterator<F>::Value() const {
  Arc arc = arcs_[i_];
  Label l = arc.ilabel;
  arc.ilabel = arc.olabel;
  arc.olabel = l;
  return arc;
}
```

Easier to use `MapFst` for this case.

Transition Representation

- We have represented a transition as:

$$e \in Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q.$$

  - Treats input and output symmetrically
  - Space-efficient single output-label per transition
  - Natural representation for composition algorithm

- Alternative representation of a transition:

$$e \in Q \times (\Sigma \cup \{\epsilon\}) \times \Delta^* \times \mathbb{K} \times Q.$$

  or equivalently,

$$e \in Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K}' \times Q, \qquad \mathbb{K}' = \Delta^* \times \mathbb{K}.$$

  - Treats string and $\mathbb{K}$ outputs uniformly
  - Natural representation for weighted transducer determinization, minimization, label pushing, and epsilon normalization.

Operations Using Alternative Transition Representation

- We can use the alternative transition representation with:

```
typedef ProductWeight<StringWeight, TropicalWeight> GallicWeight;
```

- Weighted transducer determinization becomes:

```
Fst<StdArc> *input = Fst::Read("input.fst");
// Converts into alternative transition representation
MapFst<StdArc, GallicArc> gallic(*input, ToGallicMapper);
WeightedDeterminizeFst<GallicArc> det(gallic);
// Ensures only one output label per transition (functional input)
FactorWeightFst<GallicArc> factor(det);
// Converts back from alternative transition representation
MapFst<GallicArc> result(factor, FromGallicMapper);
```

- Efficiency is not sacrificed given the lazy computation and an efficient string semiring representation.

- Weighted transducer minimization, label pushing and epsilon normalization are similarly implemented easily using the generic (acceptor) weighted minimization, weight pushing, and epsilon removal algorithms.

Operation Options

- Example Options:

```
typedef RhoMatcher< SortedMatcher<StdFst> > RM;

ComposeFstOptions<StdArc, RM> opts;

opts.matcher1 = new RM(fst1, MATCH_NONE, kNoLabel);
opts.matcher2 = new RM(fst2, MATCH_INPUT, kNoLabel);

StdComposeFst cfst(fst1, fst2, opts);
```

- Many operations optionally take similar option arguments.

Composition: Matcher Design

- Matchers can find and iterate through requested labels at FST states; principal use in composition matching.
- Matcher Form:

```
template <class F>
class Matcher {
typedef typename F::Arc Arc;

public:
  void SetState(StateId s);    // Specifies current state
  bool Find(Label label);      // Checks state for match to label
  bool Done() const;           // No more matches
  const Arc& Value() const;    // Current arc
  void Next();                 // Advance to next arc
};
```

Matchers

- Predefined Matchers:

Name	Description
SortedMatcher	Binary search on sorted input
RhoMatcher<M>	ρ symbol handling; templated on underlying matcher
SigmaMatcher<M>	σ symbol handling; templated on underlying matcher
PhiMatcher<M>	φ symbol handling; templated on underlying matcher

- The *special symbols* referenced above behave as:

	Consumes no symbol	Consumes symbol
Matches all	ϵ	σ
Matches rest	φ	ρ

Break

Next part: Applications

The idea is to have you use the library (suggested by the organizers). If you haven't do so already, please use the break to:

- Install and compile the library:

  `http://openfst.org/FstDownload`

- Download the example files:

  `http://openfst.org/FstExamples`

- Make sure everything is working by compiling some of these files into binary FSTs:

```
gunzip -c wotw.lm.gz |
fstcompile -isymbols=wotw.syms -osymbols=wotw.syms > wotw.lm.fst
```