Weighted Finite-State Transducers in Speech Recognition

Part I. ***Mathematical Foundation and Algorithms***

Mehryar Mohri and Michael Riley (*)
AT&T Labs - Research
{mohri, riley}@research.att.com

(*) with contributions from Fernando Pereira and Mark-Jan Nederhof.

Why Weighted Finite-State Transducers?

1. **Efficiency and Generality of Classical Automata Algorithms**
   - Efficient algorithms for a variety of problems (e.g. string-matching, compilers, Unix, design of controllability systems in aircrafts).
   - General algorithms: rational operations, intersection.
2. **Weights**
   - Handling uncertainty: text, handwritten text, speech, image, biological sequences.
   - Increased generality: finite-state transducers, multiplicity.
3. **Applications**
   - Text: pattern-matching, indexation, compression.
   - Speech: Large-vocabulary speech recognition, speech synthesis.
   - Image: image compression, filters.

Software Libraries

- **FSM Library**: Finite-State Machine Library – general software utilities for building, combining, optimizing, and searching weighted automata and transducers.
  `http://www.research.att.com/sw/tools/fsm/`

- **GRM Library**: Grammar Library – general software collection for constructing and modifying weighted automata and transducers representing grammars and statistical language models.
  `http://www.research.att.com/sw/tools/grm/`

FSM Library

The FSM utilities construct, combine, minimize, and search *weighted finite-states machines (FSMs)*.

- **User Program Level:** Programs that read from and write to files or pipelines, *fsm(1)*:

  ```
  fsmintersect in1.fsm in2.fsm >out.fsm
  ```

- **C(++) Library Level:** Library archive of C(++) functions that implements the user program level, *fsm(3)*:

  ```
  Fsm in1 = FSMLoad("in1.fsm");
  Fsm in2 = FSMLoad("in2.fsm");
  Fsm out = FSMIntersect(fsm1, fsm2);
  FSMDump("out.fsm", out);
  ```

- **Definition Level:** Specification of *labels*, of *costs*, and of kinds of FSM representations.

FSM File Types

- **Textual Format:** Used for manual inputting and viewing of FSMs
  - Acceptor Files
  - Transducer Files
  - Symbols Files
- **Binary Format:** ‘Compiled’ representation used by all FSM utilities.

Compiling, Printing, and Drawing FSMs

- **Compiling**

```
fsmcompile -s tropical -iA.syms <A.txt >A.fsm
fsmcompile -s log -iA.syms -oA.syms -t <T.txt >T.fsm
```

- **Printing**

```
fsmprint -iA.syms <A.fsm >A.txt
fsmprint -iA.syms -oA.syms <T.fsm >T.txt
```

- **Drawing**

```
fsmdraw -iA.syms <A.fsm | dot -Tps >A.ps
fsmdraw -iA.syms -oA.syms <T.fsm | dot -Tps >T.ps
```

Weight Sets: Semirings

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

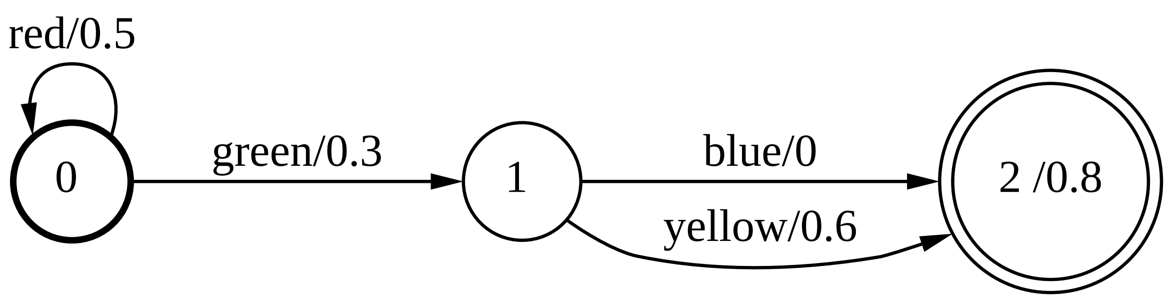
- **Sum**: to compute the weight of a sequence (sum of the weights of the paths labeled with that sequence).
- **Product:** to compute the weight of a path (product of the weights of constituent transitions).

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

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

Automata/Acceptors

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



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

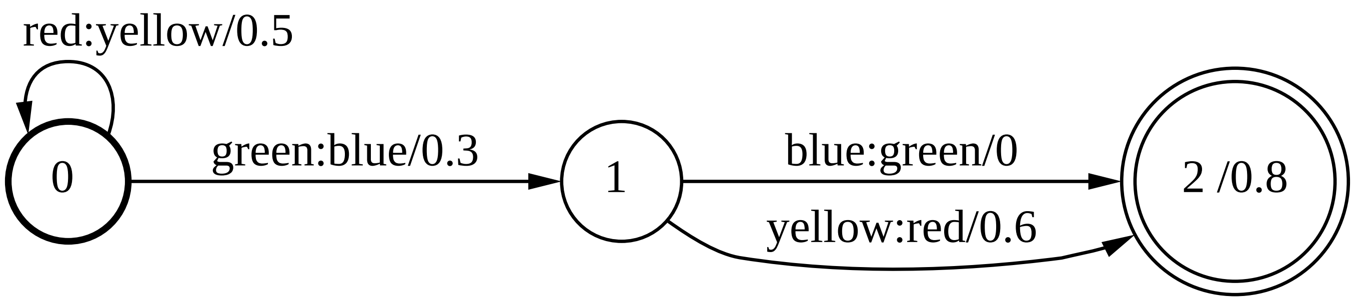
```
0   0    red      .5
0   1    green    .3
1   2    blue
1   2    yellow   .6
2   .8
```

- **Symbols File** (`A.syms`):

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

Transducers

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



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

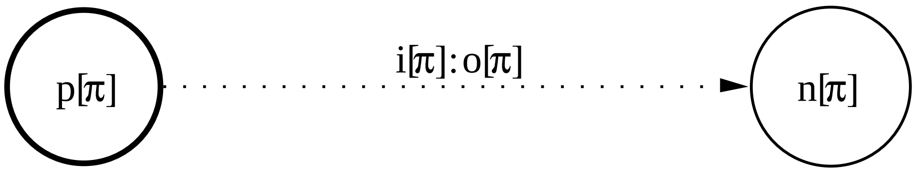
```
0   0    red      yellow   .5
0   1    green    blue     .3
1   2    blue     green
1   2    yellow   red      .6
2   .8
```

- **Symbols File** (`T.syms`):

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

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

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

Rational Operations – Algorithms

- **Definitions**

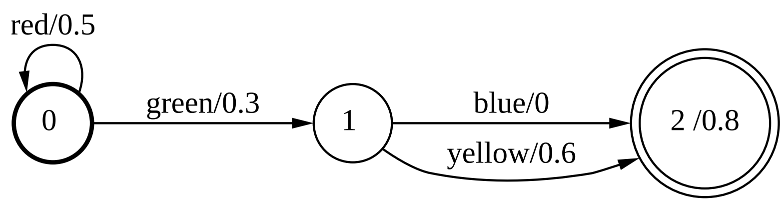
Operation	Definition and Notation
Sum	$[\![T_1 \oplus T_2]\!](x,y) = [\![T_1]\!](x,y) \oplus [\![T_2]\!](x,y)$
Product	$[\![T_1 \otimes T_2]\!](x,y) = \bigoplus_{x=x_1x_2, y=y_1y_2} [\![T_1]\!](x_1,y_1) \otimes [\![T_2]\!](x_2,y_2)$
Closure	$[\![T^*]\!](x,y) = \bigoplus_{n=0}^{\infty} [\![T]\!]^n(x,y)$

- **Conditions on the closure operation**: condition on T: e.g. weight of ϵ-cycles $= \overline{0}$ (*regulated transducers*), or semiring condition: e.g. $\overline{1} \oplus x = \overline{1}$ as with the tropical semiring (*locally closed semirings*).

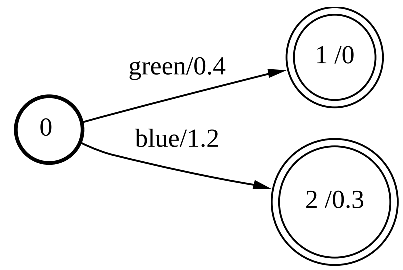
- **Complexity and implementation**
  - Complexity (linear): $O((|E_1| + |Q_1|) + (|E_2| + |Q_2|))$ or $O(|Q| + |E|)$.
  - Lazy implementation.

Sum – Illustration

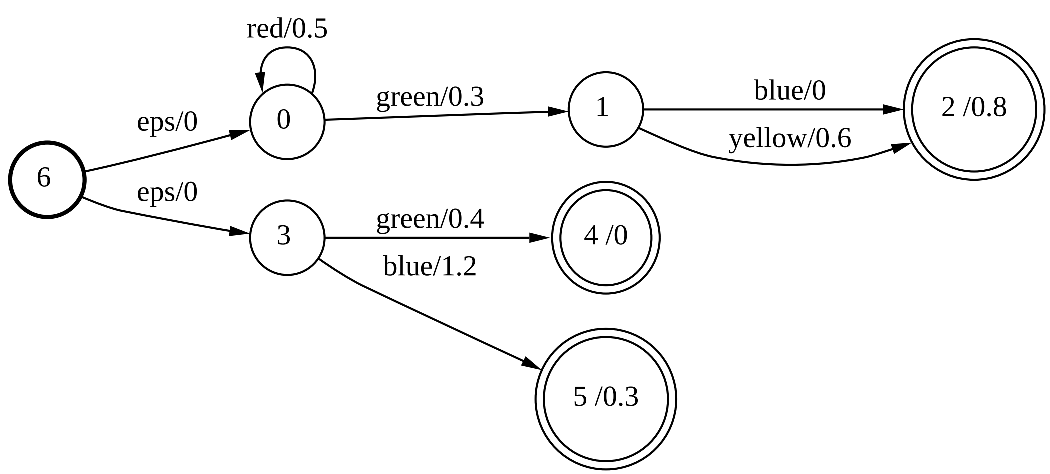
- **Program:** `fsmunion A.fsm B.fsm >C.fsm`
- **Graphical Representation:**



A.fsa



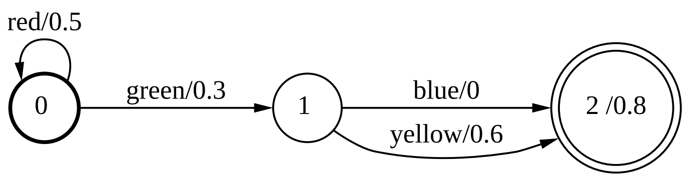
B.fsa



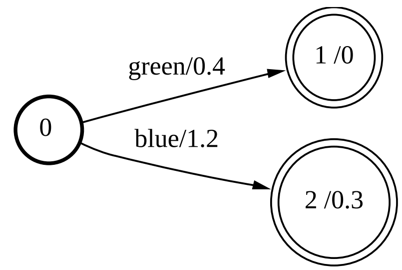
C.fsa

Product – Illustration

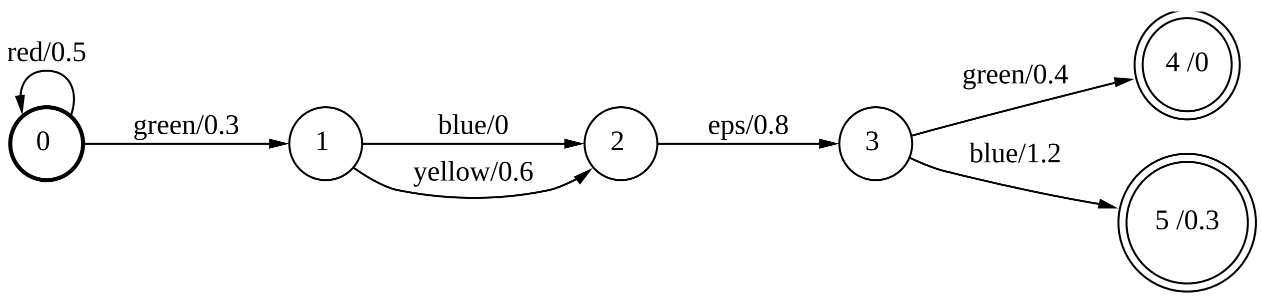
- **Program:** `fsmconcat A.fsm B.fsm >C.fsm`
- **Graphical Representation:**



A.fsa



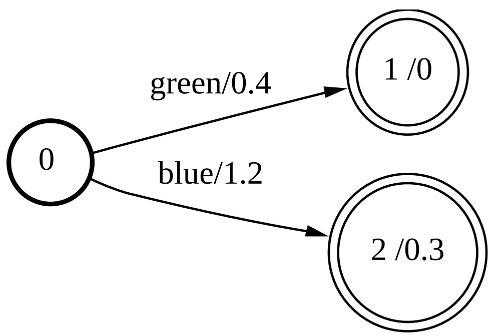
B.fsa



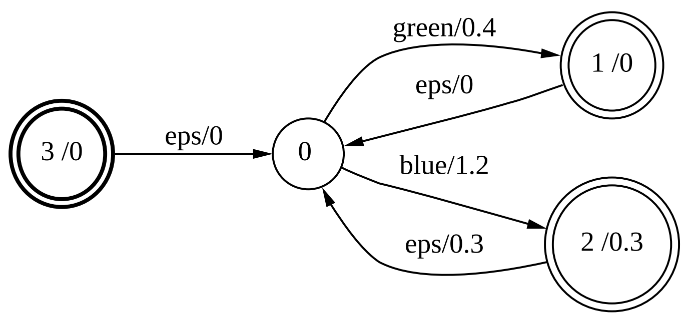
C.fsa

Closure – Illustration

- **Program:** `fsmclosure B.fsm >C.fsm`
- **Graphical Representation:**



B.fsa



C.fsa

Some Elementary Unary Operations – Algorithms

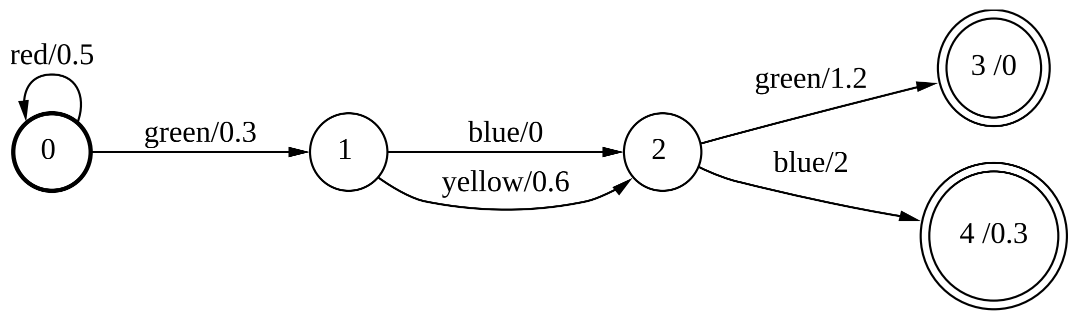
- **Definitions**

Operation	Definition and Notation	Lazy Implementation
Reversal	$[\![\widetilde{T}]\!](x, y) = [\![T]\!](\widetilde{x}, \widetilde{y})$	No
Inversion	$[\![T^{-1}]\!](x, y) = [\![T]\!](y, x)$	Yes
Projection	$[\![A]\!](x) = \bigoplus_{y} [\![T]\!](x, y)$	Yes

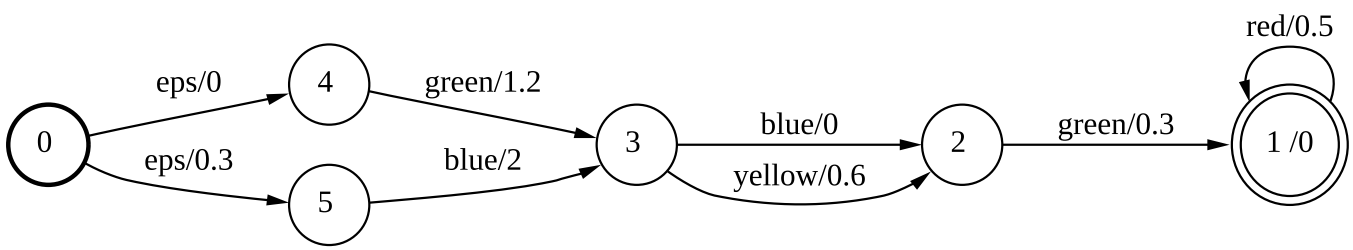
- **Complexity and implementation**
  - Complexity (linear): $O(\|Q\| + \|E\|)$.
  - Lazy implementation (see table).

Reversal – Illustration

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



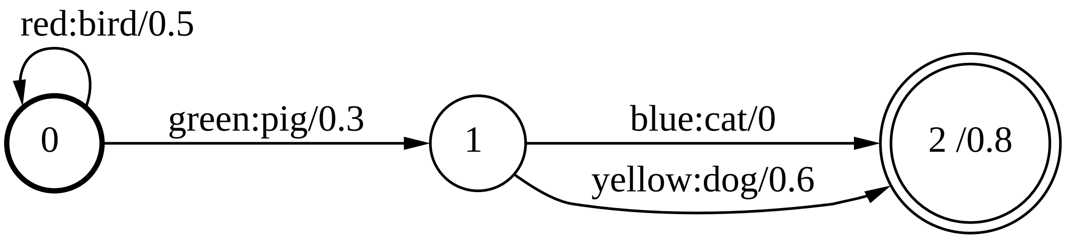
A.fsa



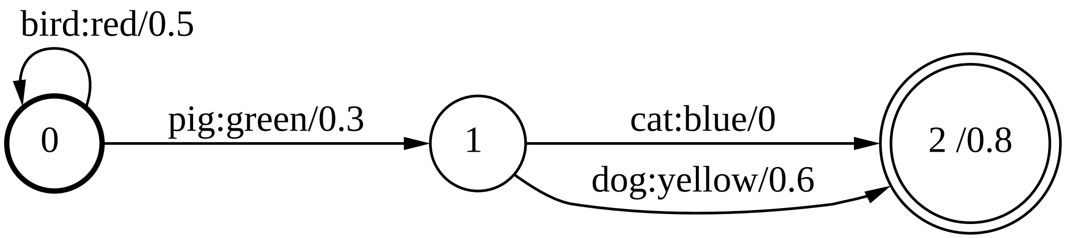
C.fsa

Inversion – Illustration

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



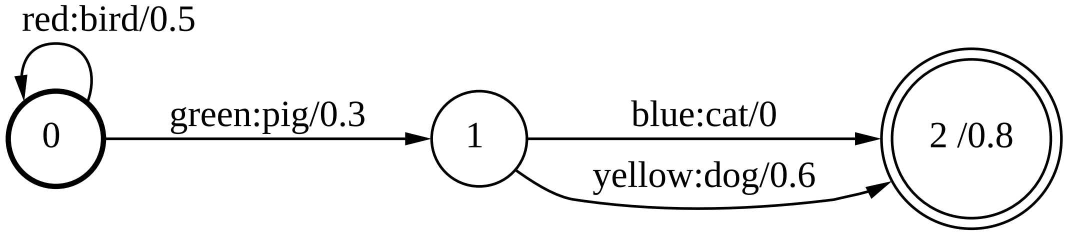
A.fst



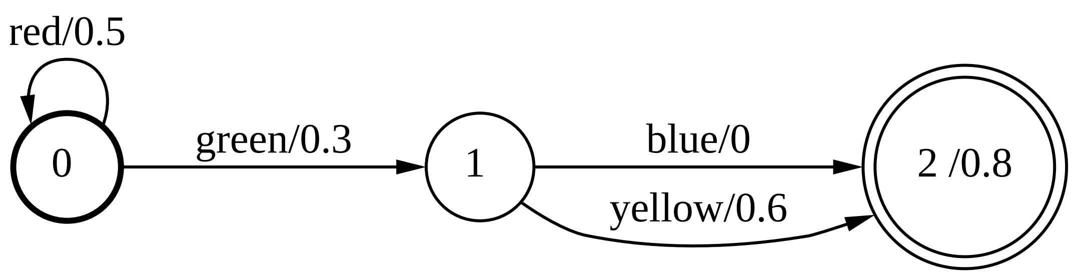
C.fst

Projection – Illustration

- **Program:** `fsmproject -1 T.fsm >A.fsm`
- **Graphical Representation:**



T.fst



A.fsa

Some Fundamental Binary Operations – Algorithms

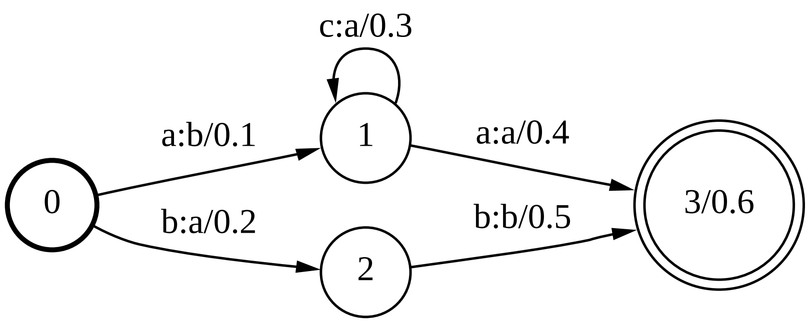
- **Definitions**

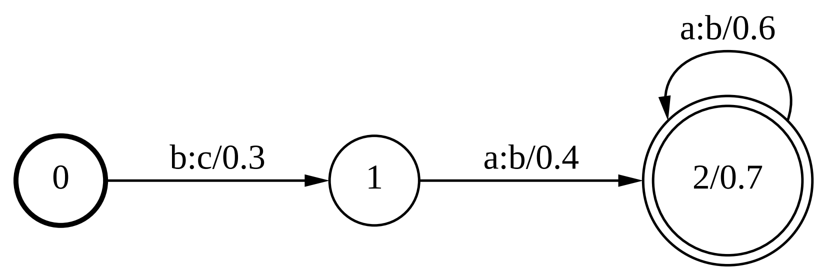
OPERATION	DEFINITION AND NOTATION	CONDITION
Composition	$[\![T_1 \circ T_2]\!](x,y) = \bigoplus_z [\![T_1]\!](x,z) \otimes [\![T_2]\!](z,y)$	$\mathbb{K}$ commutative
Intersection	$[\![A_1 \cap A_2]\!](x) = [\![A_1]\!](x) \otimes [\![A_2]\!](x)$	$\mathbb{K}$ commutative
Difference	$[\![A_1 - A_2]\!](x) = [\![A_1 \cap \overline{A_2}]\!](x)$	A_2 unweighted & deterministic

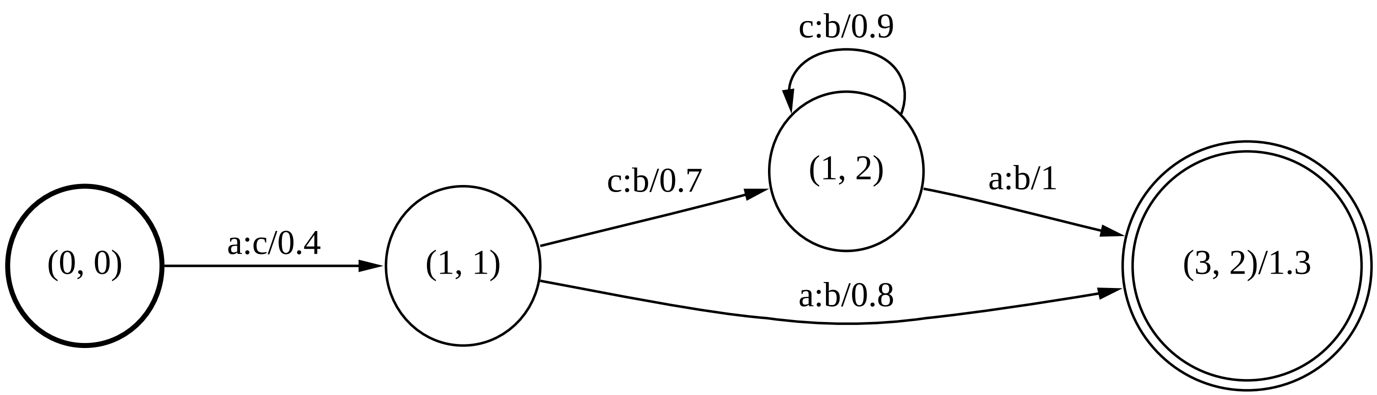
- **Complexity and implementation**
  - Complexity (quadratic): $O((|E_1| + |Q_1|)\,(|E_2| + |Q_2|))$.
  - Path multiplicity in presence of ϵ-transitions: ϵ-filter.
  - Lazy implementation.

Composition – Illustration

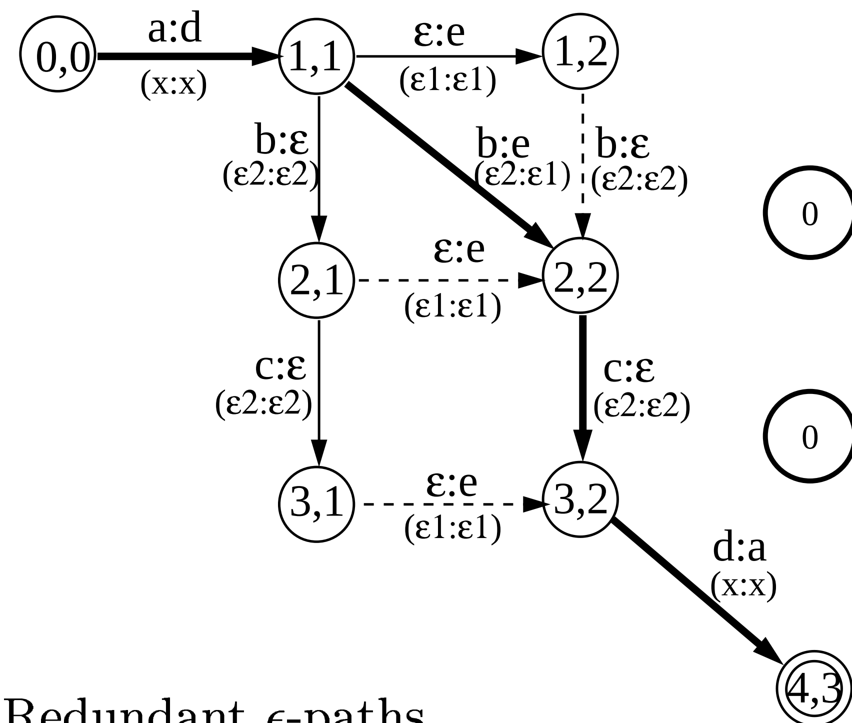
- **Program:** `fsmcompose A.fsm B.fsm >C.fsm`
- **Graphical Representation:**

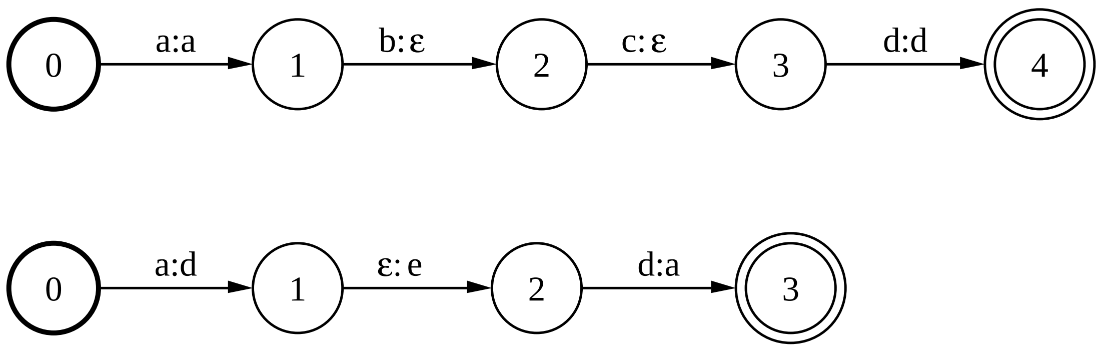






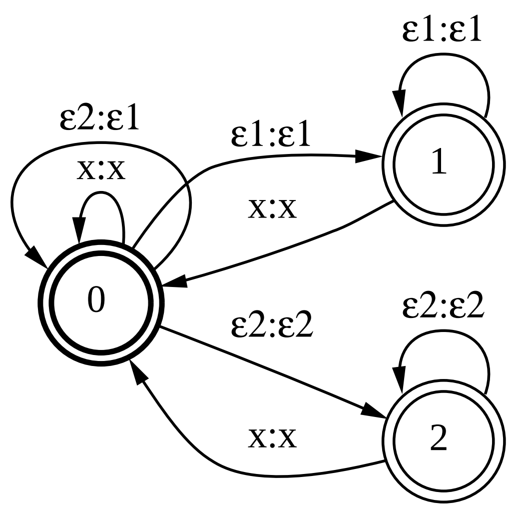
Multiplicity & ϵ-Transitions – Problem





Redundant ϵ-paths.

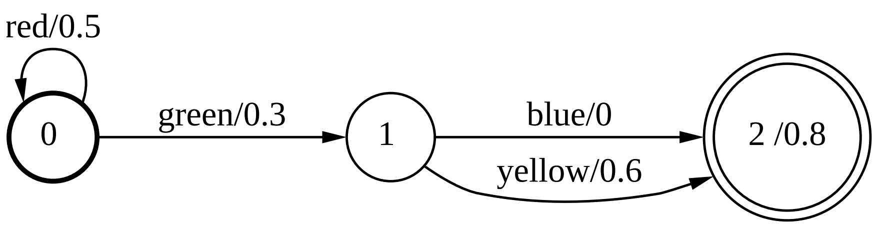
Solution – Filter F for Composition



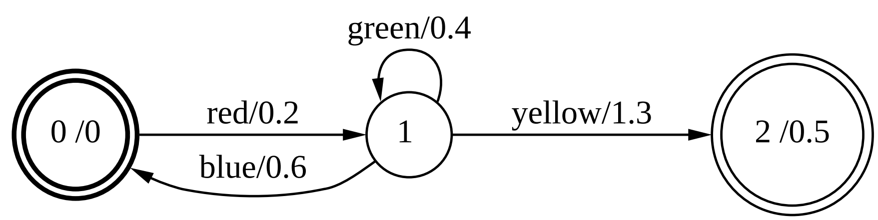
Replace $T_1 \circ T_2$ by $T_1 \circ F \circ T_2$.

Intersection – Illustration

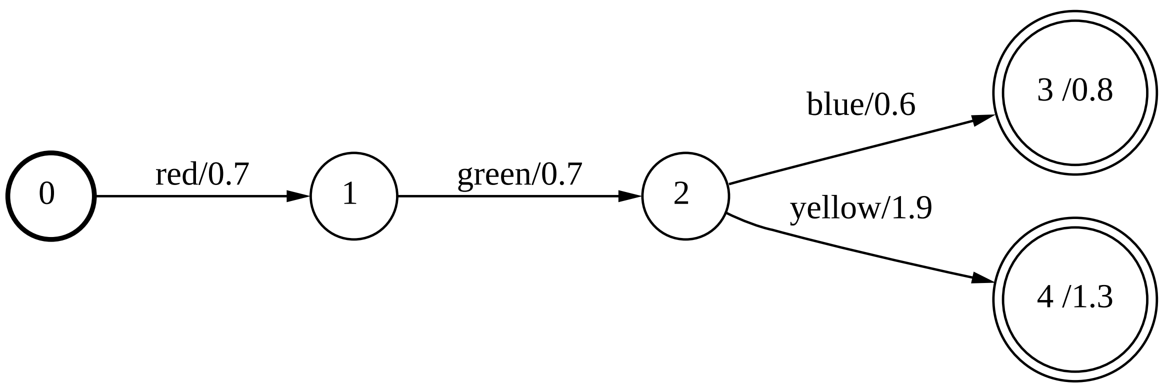
- **Program:** `fsmintersect A.fsm B.fsm >C.fsm`
- **Graphical Representation:**



A.fsa



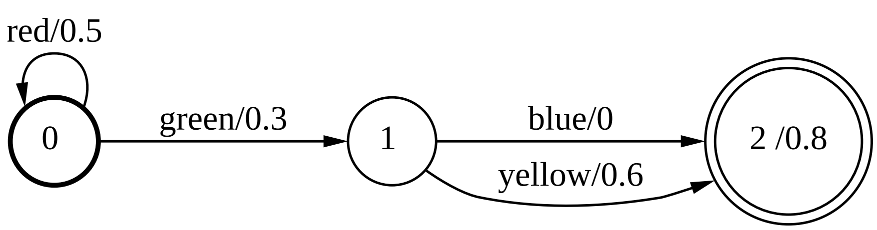
B.fsa



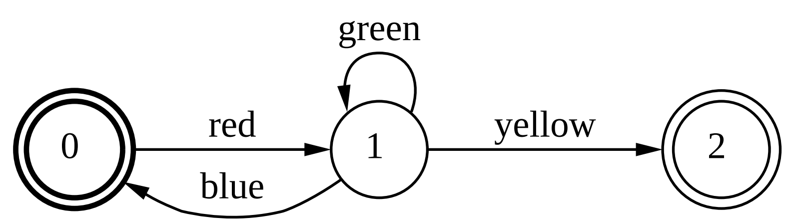
C.fsa

Difference – Illustration

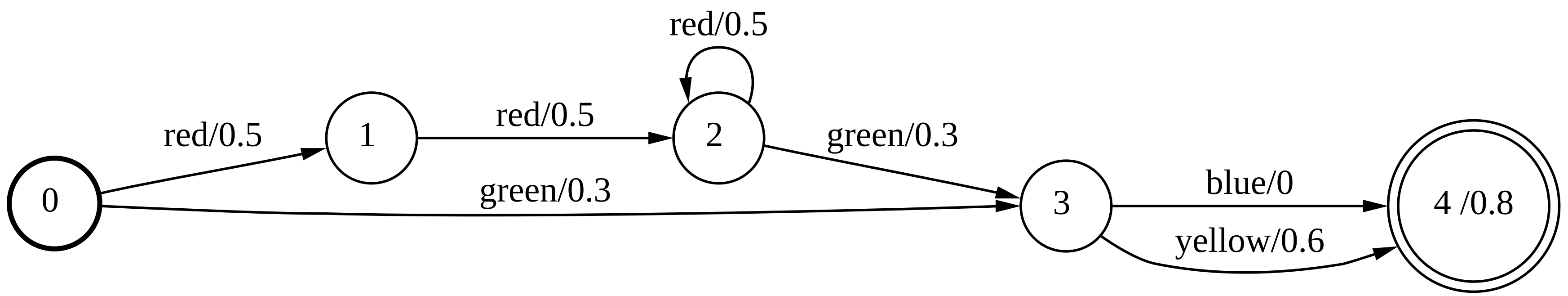
- **Program:** `fsmdifference A.fsm B.fsm >C.fsm`
- **Graphical Representation:**



A.fsa



B.fsa



C.fsa

Optimization Algorithms – Overview

- **Definitions**

Operation	Description
Connection	Removes non-accessible/non-coaccessible states
ϵ-Removal	Removes ϵ-transitions
Determinization	Creates equivalent deterministic machine
Pushing	Creates equivalent pushed/stochastic machine
Minimization	Creates equivalent minimal deterministic machine

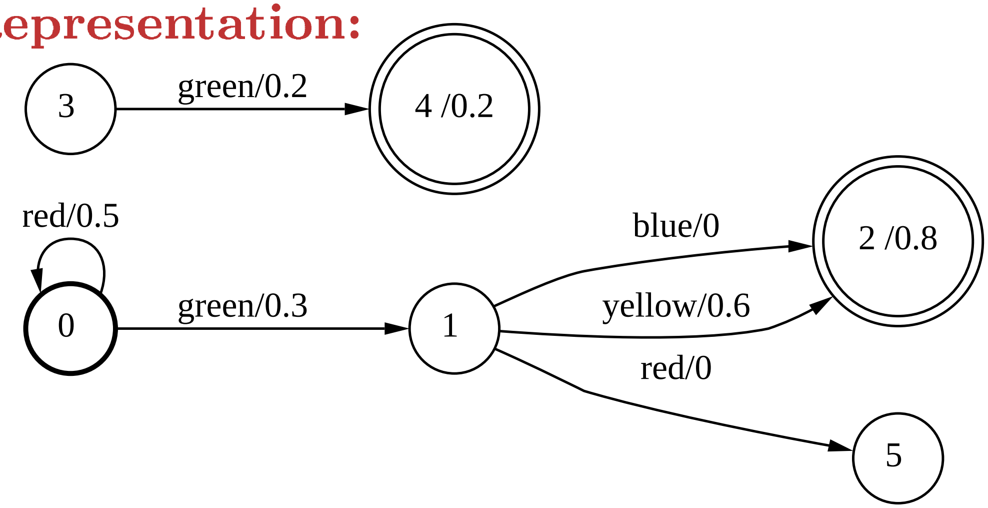
- **Conditions**: There are specific semiring conditions for the use of these algorithms. E.g. not all weighted automata or transducers can be determinized using the determinization algorithm.

Connection – Algorithm

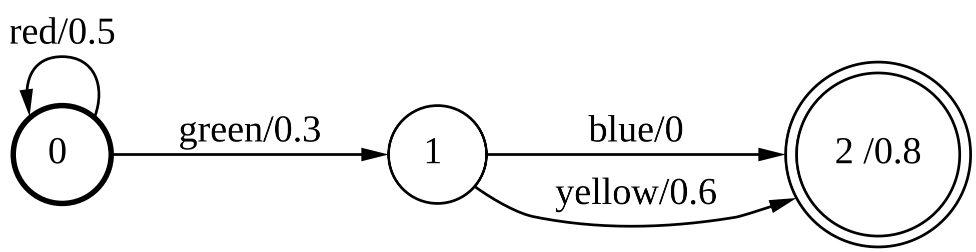
- **Definition**
  - Input: weighted transducer T_1.
  - Output: weighted transducer $T_2 \equiv T_1$ with all states connected.
- **Description**
  1. Depth-first search of T_1 from I_1.
  2. Mark accessible and coaccessible states.
  3. Keep marked states and corresponding transitions.
- **Complexity and implementation**
  - Complexity (linear): $O(|Q_1| + |E_1|)$.
  - No natural lazy implementation.

Connection – Illustration

- **Program:** `fsmconnect A.fsm >C.fsm`
- **Graphical Representation:**



A.fsa



C.fsa

ϵ-Removal – Algorithm

- **Definition**
  - Input: weighted transducer T_1 with ϵ-transitions.
  - Output: weighted transducer $T_2 \equiv T_1$ with no ϵ-transition.
- **Description** (two stages):
  1. **Computation of ϵ-closures**: for any state p, states q that can be reached from p via ϵ-paths and the total weight of the ϵ-paths from p to q.
  $$C[p] = \left\{(q, w) : q \in \epsilon[p],\ d[p, q] = w \neq \overline{0}\right\}$$
  with:
  $$d[p, q] = \bigoplus_{\pi \in P(p, \epsilon, q)} w[\pi]$$
  2. **Removal of ϵ's**: actual removal of ϵ-transitions and addition of new transitions.

$\Longrightarrow$ All-pair $\mathbb{K}$-shortest-distance problem in T_ϵ (T reduced to its ϵ-transitions).

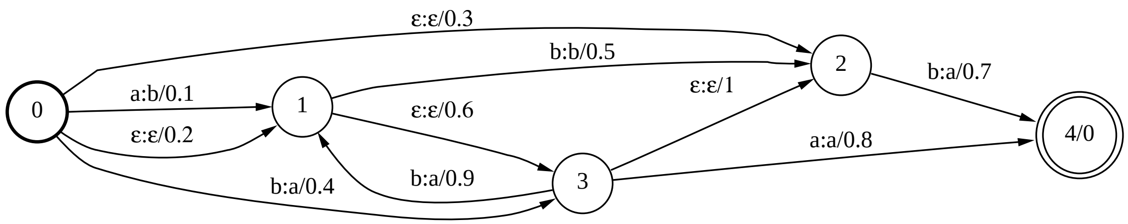
- **Complexity and implementation**
  - All-pair shortest-distance algorithm in T_ϵ.
    - k-Closed semirings (for T_ϵ) or approximation: generic sparse shortest-distance algorithm [See references].
    - Closed semirings: Floyd-Warshall or Gauss-Jordan elimination algorithm with decomposition of T_ϵ into strongly connected components [See references],

      space complexity (quadratic): $O(|Q|^2 + |E|)$.

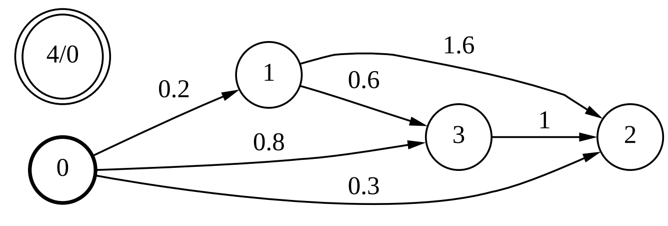
      time complexity (cubic): $O(|Q|^3(T_\oplus + T_\otimes + T_*))$.
  - Complexity:
    - Acyclic T_ϵ: $O(|Q|^2 + |Q||E|(T_\oplus + T_\otimes))$.
    - General case (tropical semiring): $O(|Q||E| + |Q|^2 \log |Q|)$.
  - Lazy implementation: integration with on-the-fly weighted determinization.

ϵ-Removal – Illustration

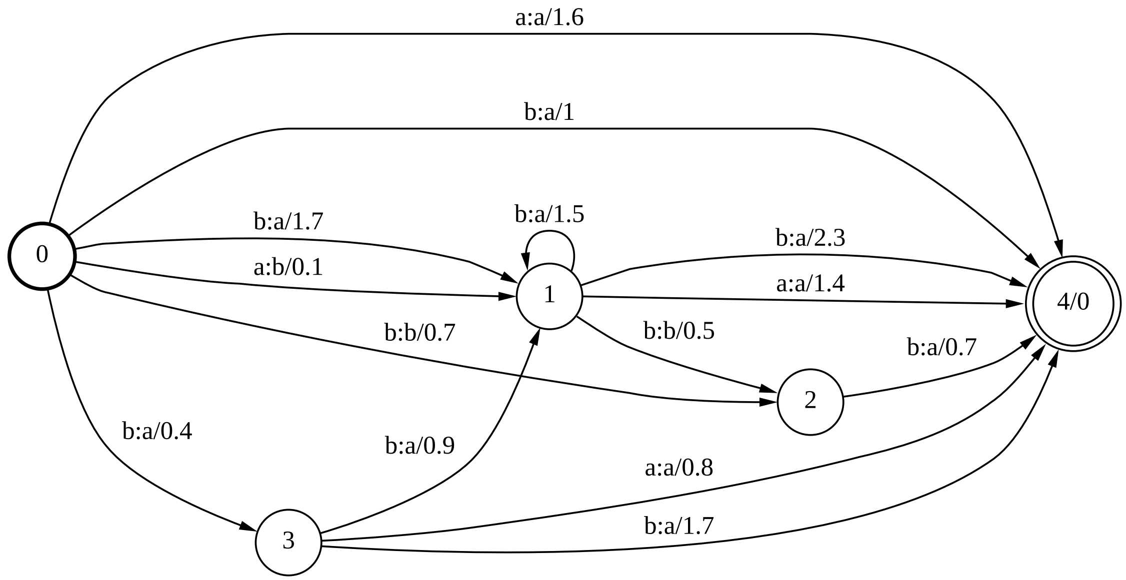
- **Program:** `fsmrmepsilon T.fsm >TP.fsm`
- **Graphical Representation:**



T.fsm



ϵ-Distances



TP.fsm

Determinization – Algorithm

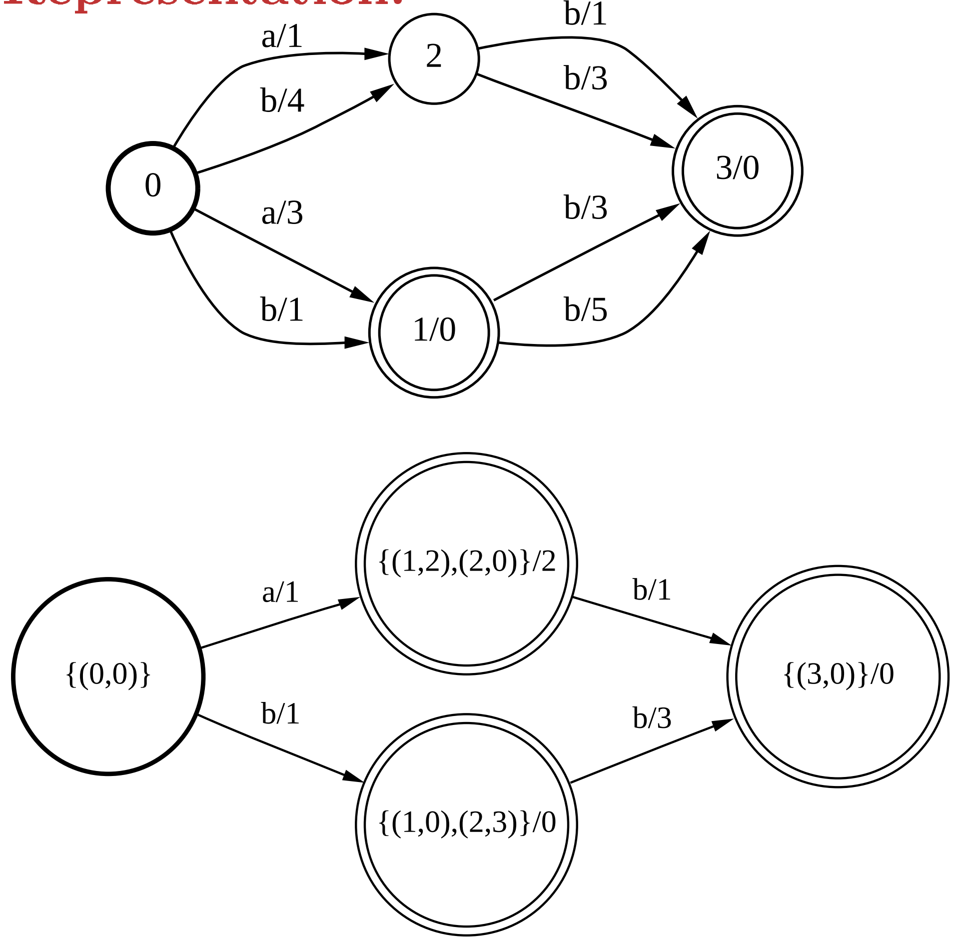
- **Definition**
  - Input: *determinizable* weighted automaton or transducer M_1.
  - Output: $M_2 \equiv M_1$ *subsequential* or *deterministic*: M_2 has a unique initial state and no two transitions leaving the same state share the same input label.
- **Description**
  1. **Generalization of subset construction**: weighted subsets $\{(q_1, w_1), \ldots, (q_n, w_n)\}$, w_i remainder weight at state q_i.
  2. **Weight of a transition in the result**: $\oplus$-sum of the original transitions pre-$\otimes$-multiplied by remainders.
- **Conditions**
  - Semiring: weakly left divisible semirings.
  - M is determinizable $\equiv$ the determinization algorithm applies to M.
  - 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 and Implementation**
  – Complexity: exponential.
  – Lazy implementation.

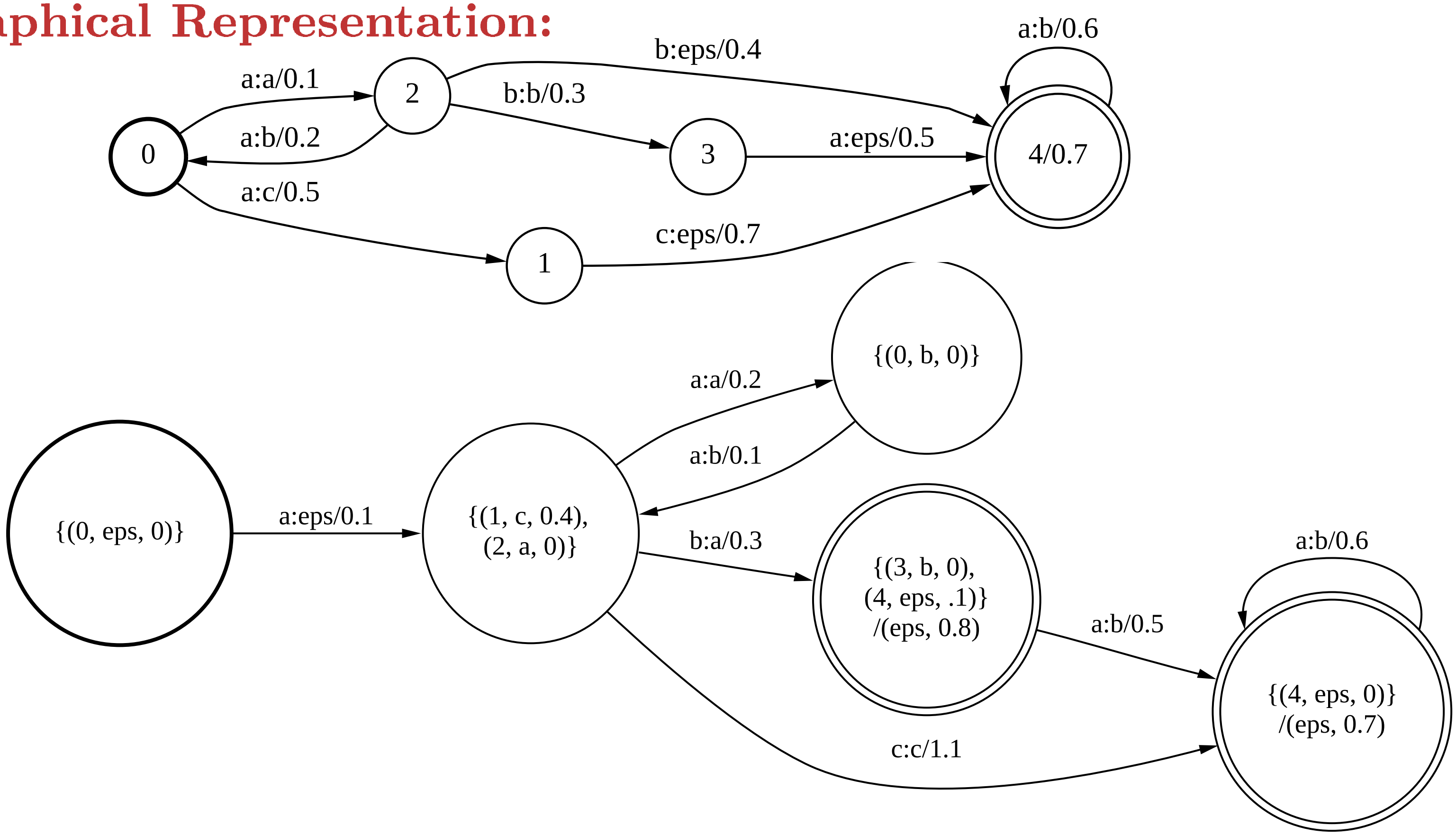
Determinization of Weighted Automata – Illustration

- **Program:** `fsmdeterminize A.fsm >D.fsm`
- **Graphical Representation:**



Determinization of Weighted Transducers – Illustration

- **Program:** `fsmdeterminize T.fsm >D.fsm`
- **Graphical Representation:**



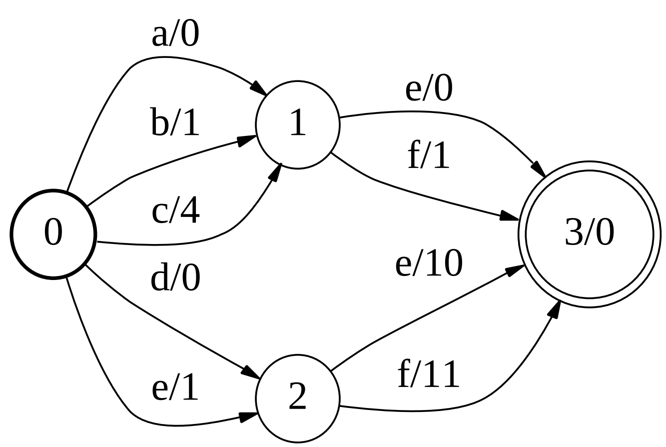
Pushing – Algorithm

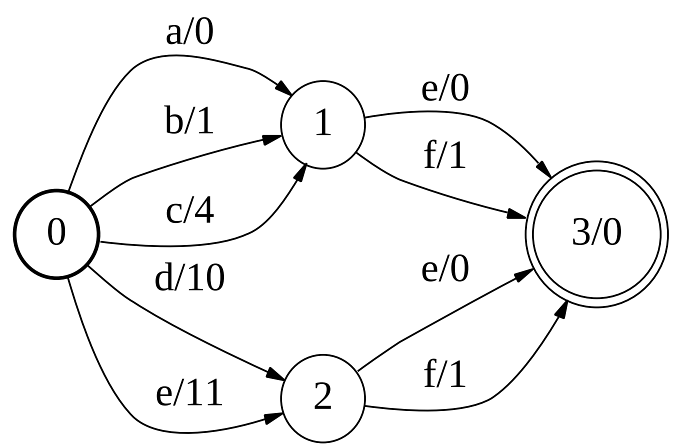
- **Definition**
  - Input: weighted automaton or transducer M_1.
  - Output: $M_2 \equiv M_1$ such that the longest common prefix of all outgoing paths $= \epsilon$ or such that the $\oplus$-sum of the weights of all outgoing transitions $= \overline{1}$ modulo the string/weight at the initial state.
- **Description** (two stages):
  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 $d[p[e]] \neq \overline{0}$,
  $$w[e] \leftarrow (d[p[e]])^{-1}(w[e] \otimes d[n[e]])$$
- **Conditions** (automata case)
  - Weakly divisible semiring.
  - Zero-sum free semiring or zero-sum free machine.

- **Complexity**
  - Automata case
    - Acyclic case (linear): $O(|Q| + |E|(T_{\oplus} + T_{\otimes}))$.
    - General case (tropical semiring): $O(|Q| \log |Q| + |E|)$.
  - Transducer case: $O((|P_{max}| + 1)\,|E|)$.

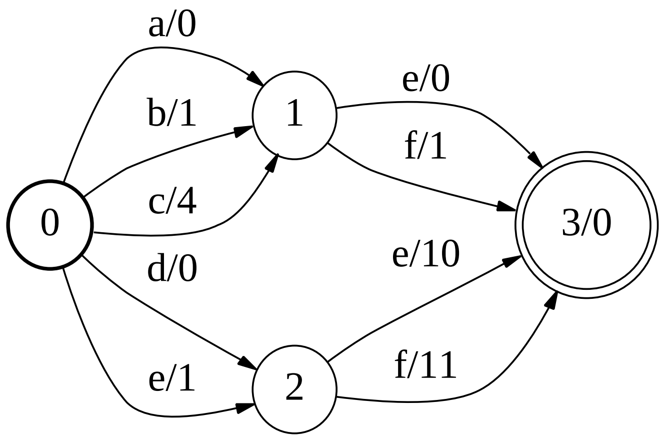
Weight Pushing – Illustration

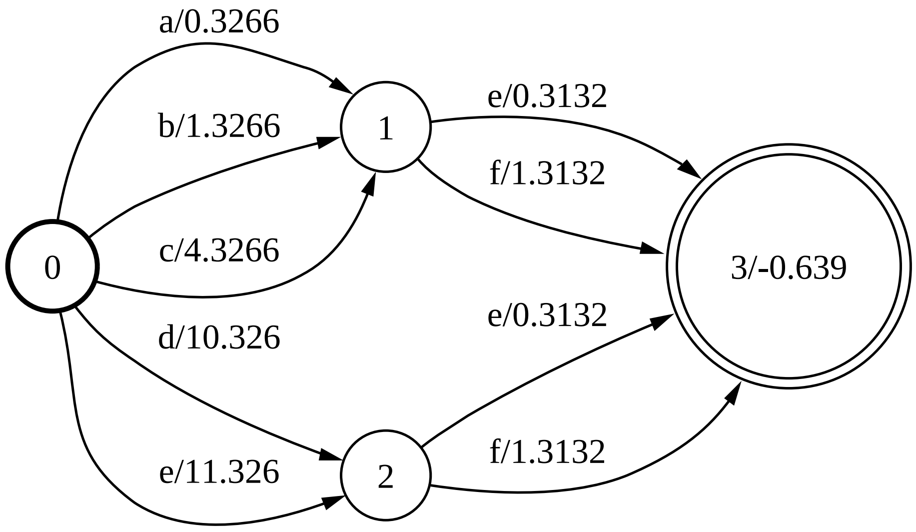
- **Program:** `fsmpush -ic A.fsm >P.fsm`
- **Graphical Representation:**
  - **Tropical semiring**





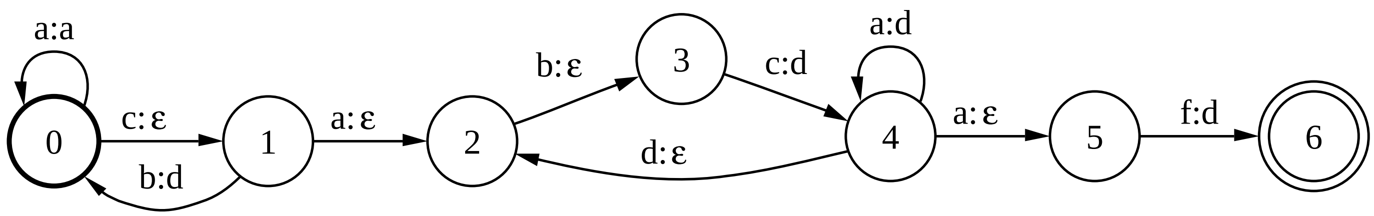
– **Log semiring**

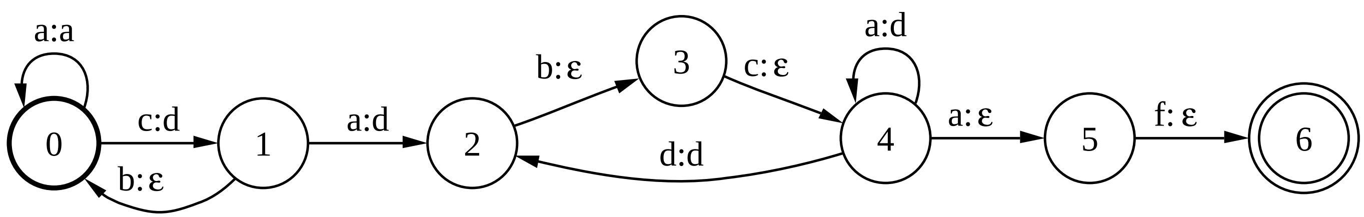




Label Pushing – Illustration

- **Program:** `fsmpush -il T.fsm >P.fsm`
- **Graphical Representation:**



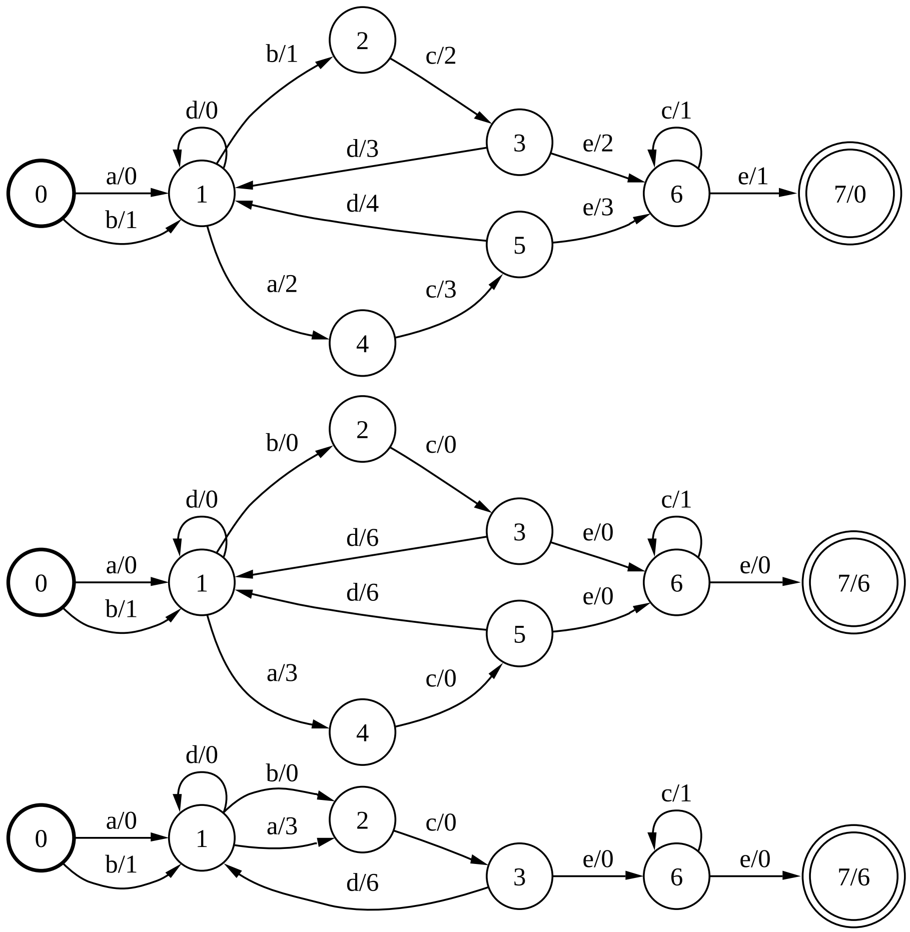


Minimization – Algorithm

- **Definition**
  - Input: deterministic weighted automaton or transducer M_1.
  - Output: deterministic $M_2 \equiv M_1$ with minimal number of states and transitions.
- **Description**: two stages
  1. **Canonical representation**: use pushing or other algorithm to standardize input automata.
  2. **Automata minimization**: encode pairs (label, weight) as labels and use classical unweighted minimization algorithm.
- **Complexity**
  - Automata case
    * Acyclic case (linear): $O(|Q| + |E|(T_\oplus + T_\otimes))$.
    * General case (tropical semiring): $O(|E| \log |Q|)$.
  - Transducer case
    * Acyclic case: $O(S + |Q| + |E|\,(|P_{max}| + 1))$.
    * General case: $O(S + |Q| + |E|\,(\log |Q| + |P_{max}|))$.

Minimization – Illustration

- **Program:** `fsmminimize D.fsm >M.fsm`
- **Graphical Representation:**

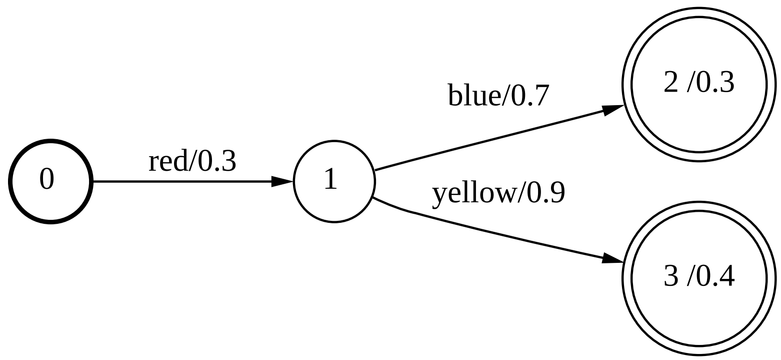


Equivalence – Algorithm

- **Definition**
  - Input: deterministic weighted automata A_1 and A_2.
  - Output: TRUE if $A_2 \equiv A_1$, FALSE otherwise.
- **Description**: two stages
  1. **Canonical representation**: use pushing or other algorithm to standardize input automata.
  2. **Test**: encode pairs (label, weight) as labels and use classical algorithm for testing the equivalence of unweighted automata.
- **Complexity**
  - First stage: $O((|E_1| + |E_2|) + (|Q_1| + |Q_2|) \log(|Q_1| + |Q_2|))$ if using pushing in the tropical semiring.
  - Second stage (quasi-linear): $O(m\,\alpha(m, n))$ where $m = |E_1| + |E_2|$ and $n = |Q_1| + |Q_2|$, and α is the *inverse of Ackermann's function*.

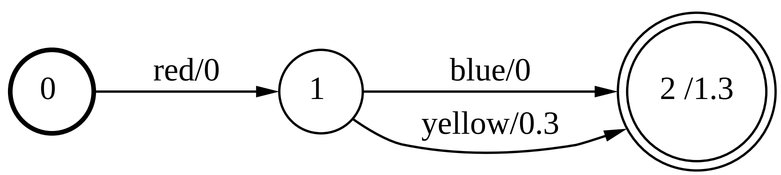
Equivalence – Illustration

- **Program:** `fsmequiv [-v] D.fsm M.fsm`
- **Graphical Representation:**



D.fsa

$$=?$$



M.fsa

Single-Source Shortest-Distance Algorithms – Algorithm

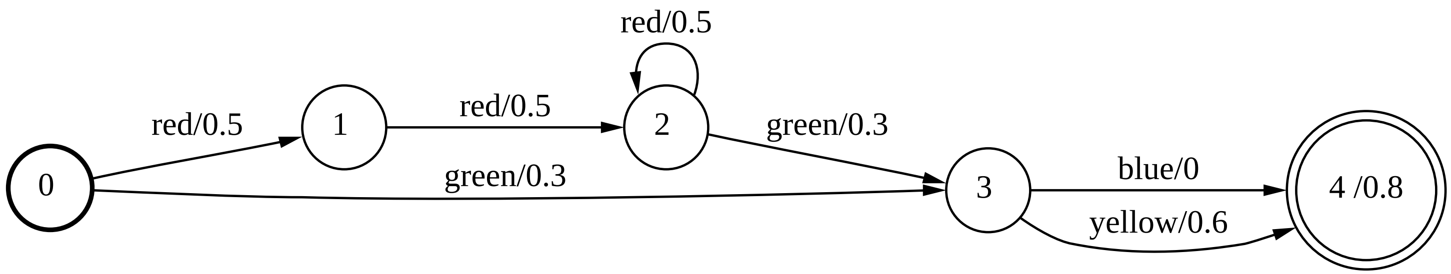
- **Generic single-source shortest-distance algorithm**
  - Definition: for each state q,

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

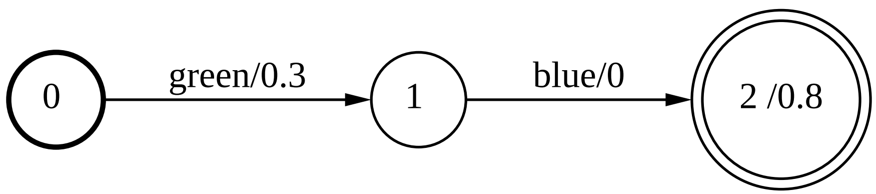
  - Works with any queue discipline and any semiring k-closed for the graph.
  - Coincides with classical algorithms in the specific case of the tropical semiring and the specific queue disciplines: best-first (Dijkstra), FIFO (Bellman-Ford), or topological sort order (Lawler).
- ***N*-best strings algorithm**
  - General N-best paths algorithm augmented with the computation of the potentials.
  - On-the-fly weighted determinization.

Single-Source Shortest-Distance Algorithms – Illustration

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



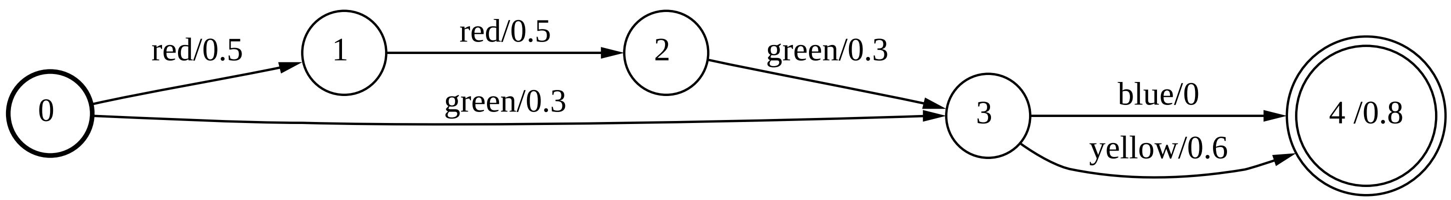
A.fsa



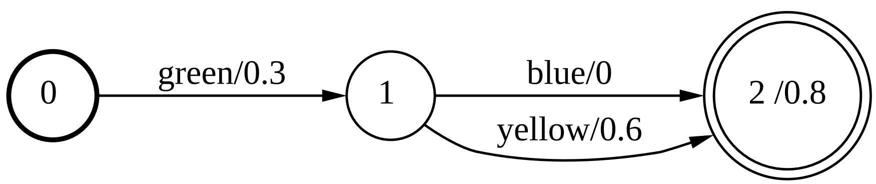
C.fsa

Pruning – Illustration

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



A.fsa



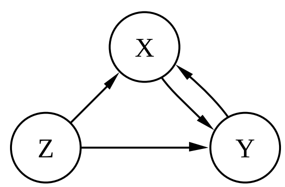
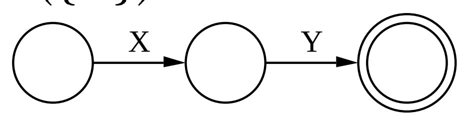
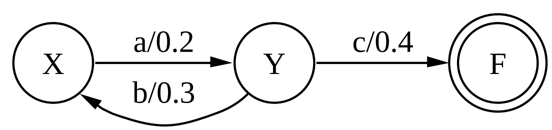
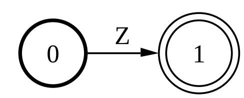
C.fsa

Compilation of Weighted CFGs – Algorithm

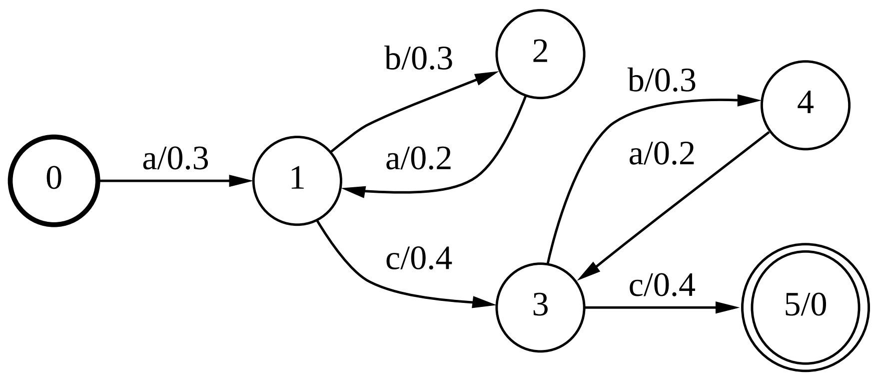
- **Definition**
  - Input: weighted context-free grammar G.
  - Output: weighted automaton A representing G.
- **Condition**: G must be *strongly regular*, e.g. rules of each set M of mutually recursive nonterminals are either all right-linear or all left-linear.
- **Description**
  1. Build the dependency graph D_G of the input grammar G.
  2. Compute the strongly connected components (SCCs) of D_G.
  3. Construct weighted automaton $K(S)$ for each SCC S and for each nonterminal $X \in S$ a weighted automaton $M(X)$ derived from $K(S)$.
  4. Create simple automaton M_G accepting exactly the set of active nonterminals A.
  5. Expand M_G on-the-fly for each input string using lazy replacement and editing.

- **Complexity and Implementation**
  - Compact intermediate representation of G by a weighted transducer T.
  - Compilation algorithm applies to T rather than G.
  - Lazy compilation algorithm, complexity (linear): $O(|T|)$.

Compilation of Weighted CFGs – Illustration

Grammar G	Dependency Graph	Weighted Automata $K(S)$	Activation Automaton M_G
$Z \; .1 \;\rightarrow\; XY$ $X \; .2 \;\rightarrow\; aY$ $Y \; .3 \;\rightarrow\; bX$ $Y \; .4 \;\rightarrow\; c$		$K(\{Z\})$:  $K(\{X, Y\})$: 	

- **Grammar G:** $Z\ .1 \rightarrow XY \quad X\ .2 \rightarrow aY \quad Y\ .3 \rightarrow bX \quad Y\ .4 \rightarrow c$
- **Program:** `grmread -i lab -w cfg.txt | grmcfcompile -i lab -s Z`
- **Graphical Representation:**



Regular Approximation of Weighted CFGs – Algorithm

- **Definition**
  - Input: arbitrary weighted context-free grammar G.
  - Output: G' strongly regular approximation of G with $L(G) \subseteq L(G')$.
- **Description**
  1. Let M be a set of mutually recursive non-terminals.
  2. For each nonterminal $A \in M$, add new nonterminal $A' \notin N$, new rule:

$$A' \;\rightarrow\; \epsilon$$

  3. Replace each rule with left-hand side $A \in M$:

$$A \;\rightarrow\; \alpha_0 \; B_1 \; \alpha_1 \; B_2 \; \alpha_2 \; \cdots \; B_m \; \alpha_m$$

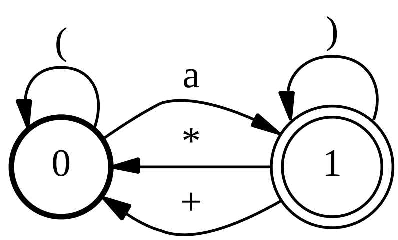
with $m \geq 0$, $B_1, \ldots, B_m \in M$, $\alpha_0 \ldots \alpha_m \in (\Sigma \cup (N - M))^*$, by:

$$\begin{array}{rcl} A & \rightarrow & \alpha_0 \; B_1 \\ B'_1 & \rightarrow & \alpha_1 \; B_2 \\ B'_2 & \rightarrow & \alpha_2 \; B_3 \\ & \ldots & \\ B'_{m-1} & \rightarrow & \alpha_{m-1} \; B_m \\ B'_m & \rightarrow & \alpha_m \; A' \end{array}$$

($A \rightarrow \alpha_0 \; A'$ when $m = 0$).

- **Complexity and Implementation**
  - At most one new non-terminal symbol for any non-terminal symbol of G.
  - Readable and modifiable result, structure of original grammar still apparent.
  - Complexity of the simple variant of the algorithm (linear): $O(|G|)$.
  - Grammar compilation algorithm directly applies to the resulting approximate grammar.

Regular Approximation of Weighted CFGs – Illustration

Grammar G	**Regular Approximation**		**Graphical Representation**
$E \rightarrow E + T$	$E' \rightarrow \epsilon$	$T \rightarrow T$	
$E \rightarrow T$	$T' \rightarrow \epsilon$	$T' \rightarrow * \; F$	
$T \rightarrow T * F$	$F' \rightarrow \epsilon$	$F' \rightarrow T'$	
$T \rightarrow F$	$E \rightarrow E$	$T \rightarrow F$	
$F \rightarrow (\; E \;)$	$E' \rightarrow + \; T$	$F' \rightarrow T'$	
$F \rightarrow a$	$T' \rightarrow E'$	$F \rightarrow (\; E$	
	$E \rightarrow T$	$E' \rightarrow) \; F'$	
	$T' \rightarrow E'$	$F \rightarrow a \; F'$	

- **Program:**

```
grmcfapproximate -i lab -o nlab cfg.fsm > ncfg.txt
grmread -i nlab ncfg.txt | grmcfcompile -i nlab -s E >M
```

Conclusion

- **Generality and Efficiency**
  - Algorithms based on a general algebraic framework (semirings).
  - Algorithms applying to machines of 500M transitions.
  - Convenient combination and optimization of different information sources (components) of a complex system.
- **Speech Processing Applications**
  - Automatic Speech Recognition (see Part II).
  - Speech Synthesis.
  - Spoken-Dialog Applications.