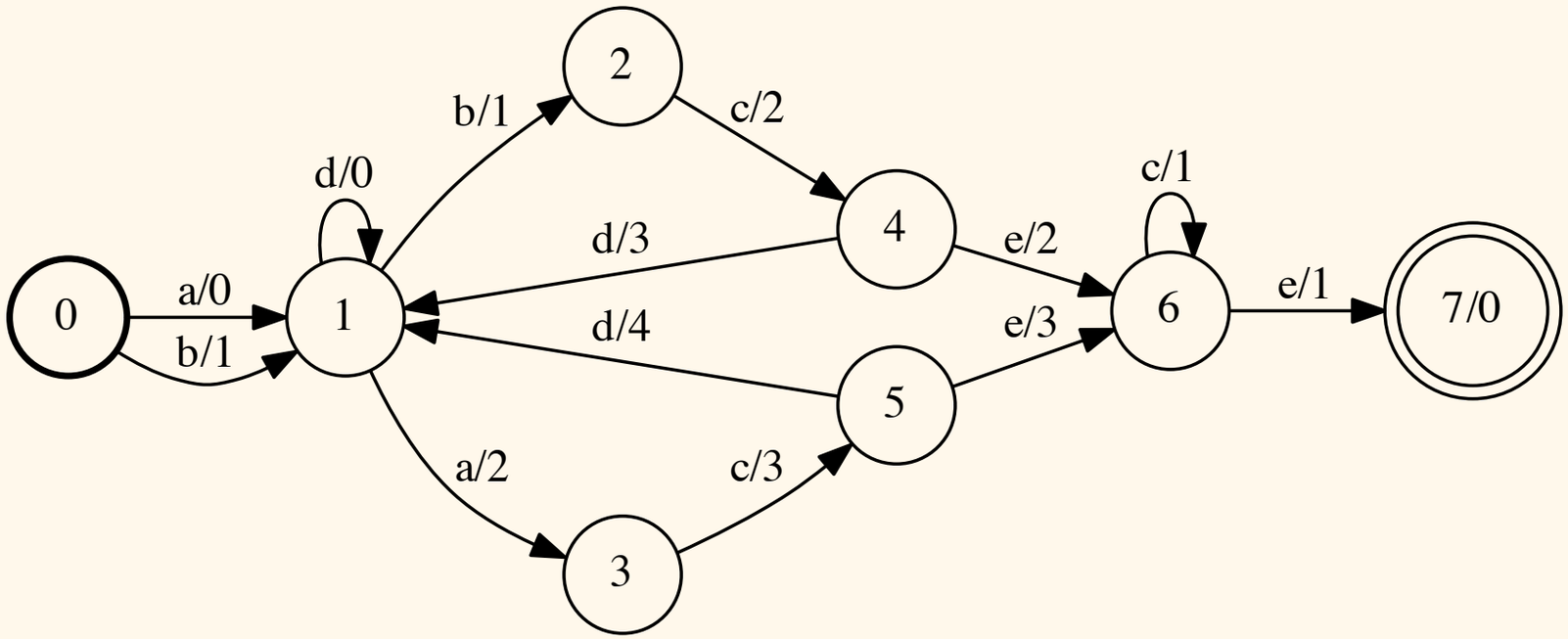
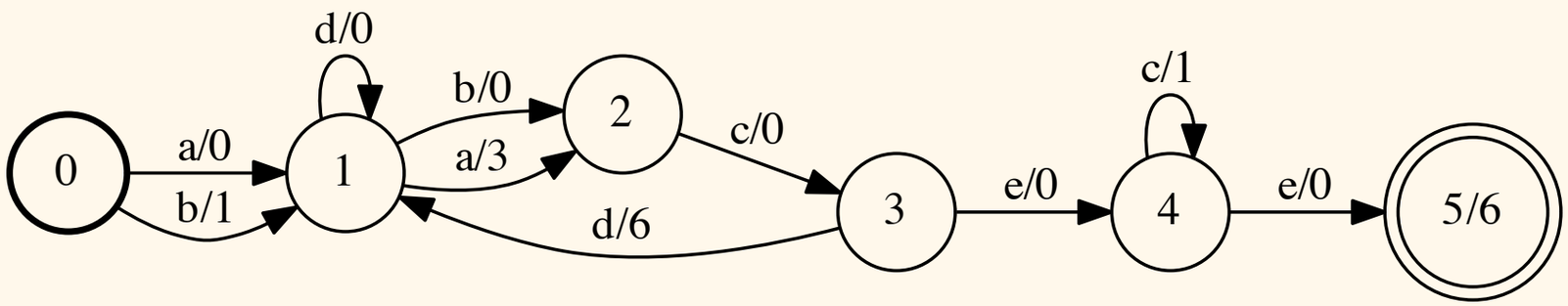
Finite State Transducers:

Their use and enjoyment





Steven Bedrick

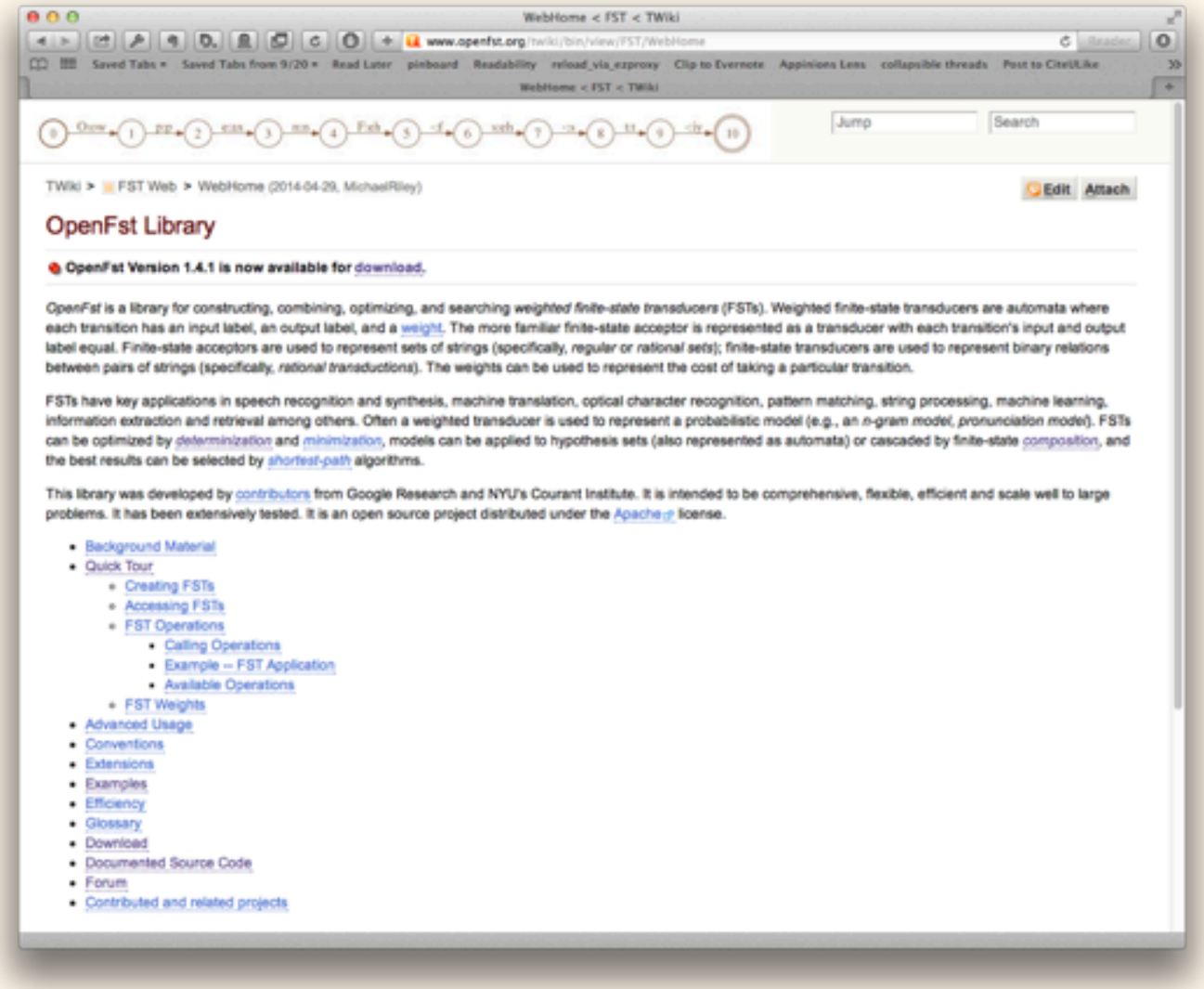
CS/EE 5/655, 10/8/14

http://www.openfst.org/twiki/pub/FST/FstHltTutorial/tutorial_part3.pdf

Plan for the day:

1. Installing OpenFST etc.
2. Revisiting Minimization
3. Binary math
4. Flight routing & Shortest Path
5. Thrax
6. Spelling correction

OpenFST is an open-source library for working with FSTs.



Written by researchers at Google, NYU, OHSU, etc. etc...

OpenFST includes a C++ API as well as a suite of command-line tools:

```
// New FST
VectorFst<StdArc> fst;

// Setup start state
fst.AddState();
fst.SetStart(0);

// Add arcs
fst.AddArc(0, StdArc('a', 'b', 1, 1));
fst.AddArc(1, StdArc('c','c', 1, 2));
fst.AddArc(1, StdArc('d','e', 0.5, 2));

fst.setFinal(2, 1); // no final weight

VectorFst<StdArc> someNewFst;
Compose(fst, someOtherFst, &someNewFst);
```

OpenFST includes a C++ API as well as a suite of command-line tools:

```
0 1 a b 1.0
1 2 c c 1.0
1 2 d e 0.5
2
```

demo.fst.txt

```
a 1
c 2
d 3
```

demo.isyms

```
b 1
c 2
e 3
```

demo.osyms

```
$ fstcompile --isymbols=demo.isyms --osymbols=demo.osyms demo.fst.txt > demo.fst
$ fstprint demo.fst
0  1  1  1  1
1  2  2  2  1
1  2  3  3  0.5
2
$ fstprint --isymbols=demo.isyms --osymbols=demo.osyms demo.fst
0  1  a  b  1
1  2  c  c  1
1  2  d  e  0.5
2
$ fstcompile --isymbols=demo.isyms --osymbols=demo.osyms \
--keep_isymbols=true --keep_osymbols=true demo.fst.txt > demo.fst
$
```

OpenFST includes a C++ API as well as a suite of command-line tools:

```
0 1 a b 1.0
1 2 c c 1.0
1 2 d e 0.5
2
```

demo.fst.txt

```
a 1
c 2
d 3
```

demo.isyms

```
b 1
c 2
e 3
```

demo.osyms

```
$ fstdraw demo.fst
digraph FST {
rankdir = LR;
size = "8.5,11";
label = "";
center = 1;
orientation = Landscape;
ranksep = "0.4";
nodesep = "0.25";
0 [label = "0", shape = circle, style = bold, fontsize = 14]
   0 -> 1 [label = "a:b/1", fontsize = 14];
1 [label = "1", shape = circle, style = solid, fontsize = 14]
   1 -> 2 [label = "c:c/1", fontsize = 14];
   1 -> 2 [label = "d:e/0.5", fontsize = 14];
2 [label = "2", shape = doublecircle, style = solid, fontsize = 14]
}
```

OpenFST includes a C++ API as well as a suite of command-line tools:

```
0 1 a b 1.0
1 2 c c 1.0
1 2 d e 0.5
2
```

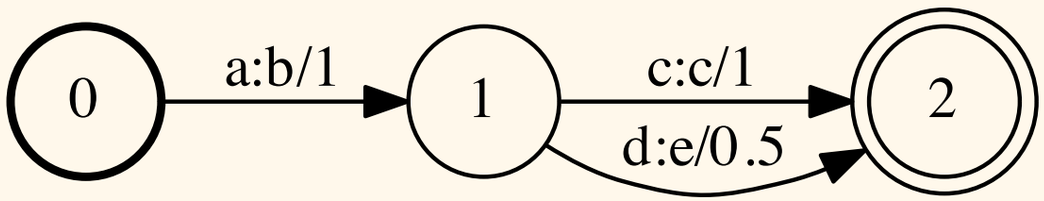
demo.fst.txt

```
a 1
c 2
d 3
```

demo.isyms

```
b 1
c 2
e 3
```

demo.osyms



OpenFST includes a C++ API as well as a suite of command-line tools:

Besides fstcompile, fstprint, and fstdraw, OpenFST provides numerous other tools:

fstinfo	Print info for FST (number of states & arcs, etc.)
fstcompose	Compose two FSTs
fstdeterminize	Determinize ndFST
fstminimize	Produce equivalent but minimized FST
fstproject	Project either input or output labels
fstinvert	Invert input and output labels
fstrmepsilon	Remove epsilon-arcs

We'll see more examples through the day...

Important additional FST concepts:

Symbol tables:

All OpenFST transducers *must* specify both input and output symbol tables...

.. tables are essentially arbitrary...

... but make sure to keep track of them: composition requires matching tables!

“FAR” files:

Like a “TAR” file, but for FSTs.

Many useful utilities: “farextract”, “farcompilestrings”, etc.

Other useful tools:

fstprintstrings:

Print (all of!) the strings represented (producible) from a given acyclic transducer.

Helpful for debugging output.

fstcompilestring.sh:

Wrapper for “farcompilestrings” that behaves in a UNIX-y fashion.

```
$ echo "hello" | ./fstcompilestring | fstprint --isymbols=ascii.syms.txt --osymbols=ascii.syms.txt
0    1    h    h
1    2    e    e
2    3    l    l
3    4    l    l
4    5    o    o
5
$
```

Both have been posted to class website!

Installing OpenFST:

Two choices: “from scratch”, or Homebrew

From scratch:

1. Download distribution
2. `./configure --with-whatever-options-you-want && make && make install`

Homebrew:

“Package Manager” for Mac OS X

Takes care of a lot of details; makes upgrading easier

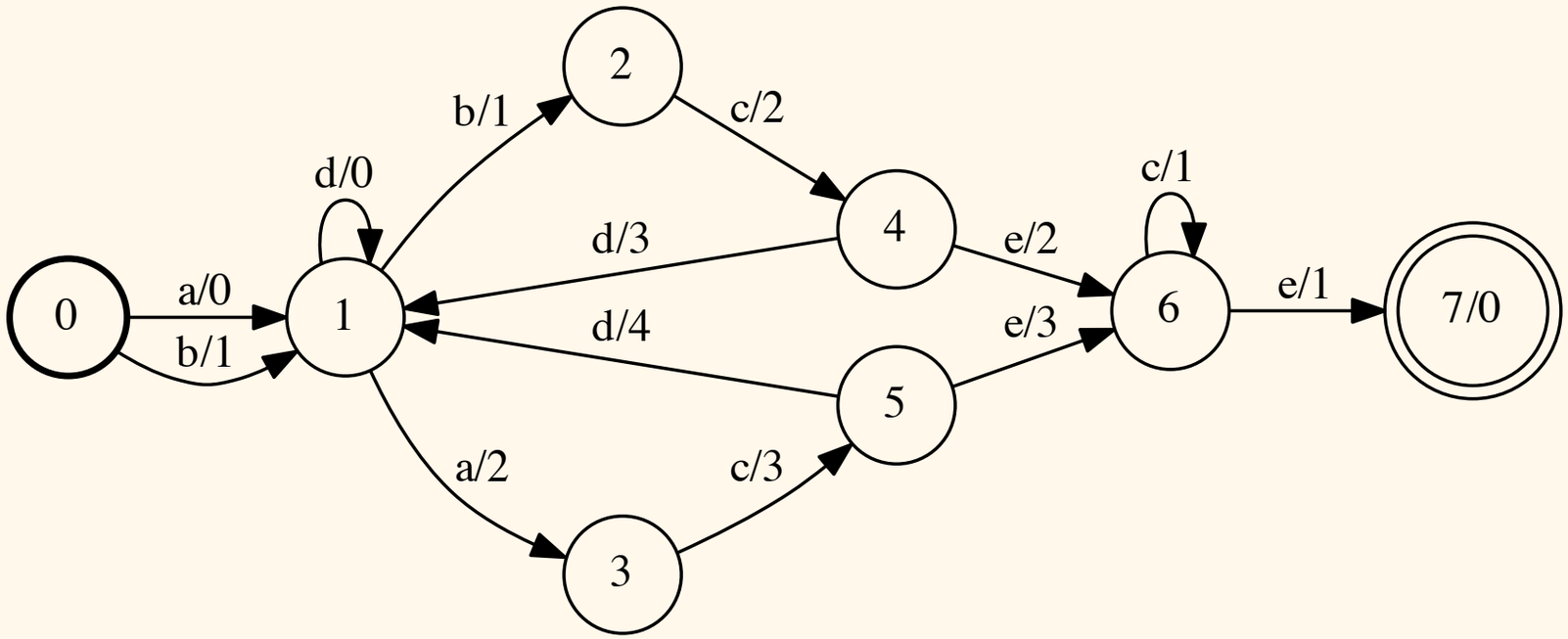
```
$ brew tap homebrew/science
$ brew install openfst
$ brew install opengrm-thrax
$ brew install opengrm-ngram
$ brew install cairo pango graphviz
```

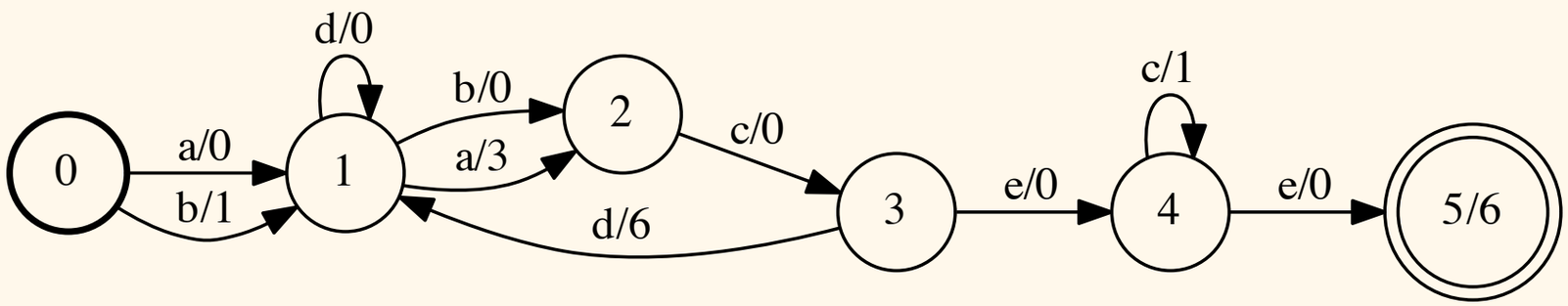
```
$ brew install opengrm-thrax
==> Installing opengrm-thrax dependency: openfst
==> Downloading http://openfst.cs.nyu.edu/twiki/pub/FST/FstDownload/openfst-1.4.1.tar.gz
Already downloaded: /Library/Caches/Homebrew/openfst-1.4.1.tar.gz
==> ./configure --prefix=/usr/local/Cellar/openfst/1.4.1 --enable-far --enable-pdt --enable-ngram-
fsts --enable-lookahead
==> make install
   /usr/local/Cellar/openfst/1.4.1: 262 files, 27M, built in 3.3 minutes
==> Installing opengrm-thrax
==> Downloading http://www.openfst.org/twiki/pub/GRM/ThraxDownload/thrax-1.1.0.tar.gz
Already downloaded: /Library/Caches/Homebrew/opengrm-thrax-1.1.0.tar.gz
==> ./configure --prefix=/usr/local/Cellar/opengrm-thrax/1.1.0
==> make install
   /usr/local/Cellar/opengrm-thrax/1.1.0: 79 files, 9.0M, built in 81 seconds
```

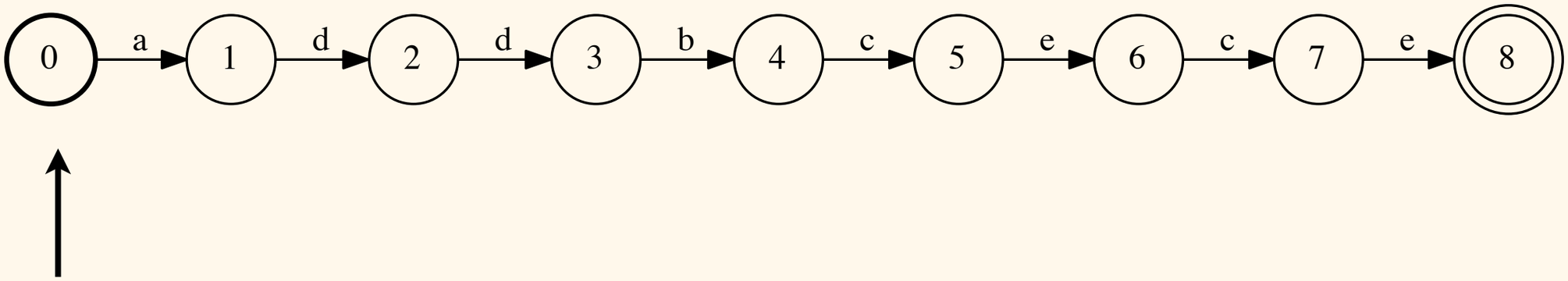
Plan for the day:

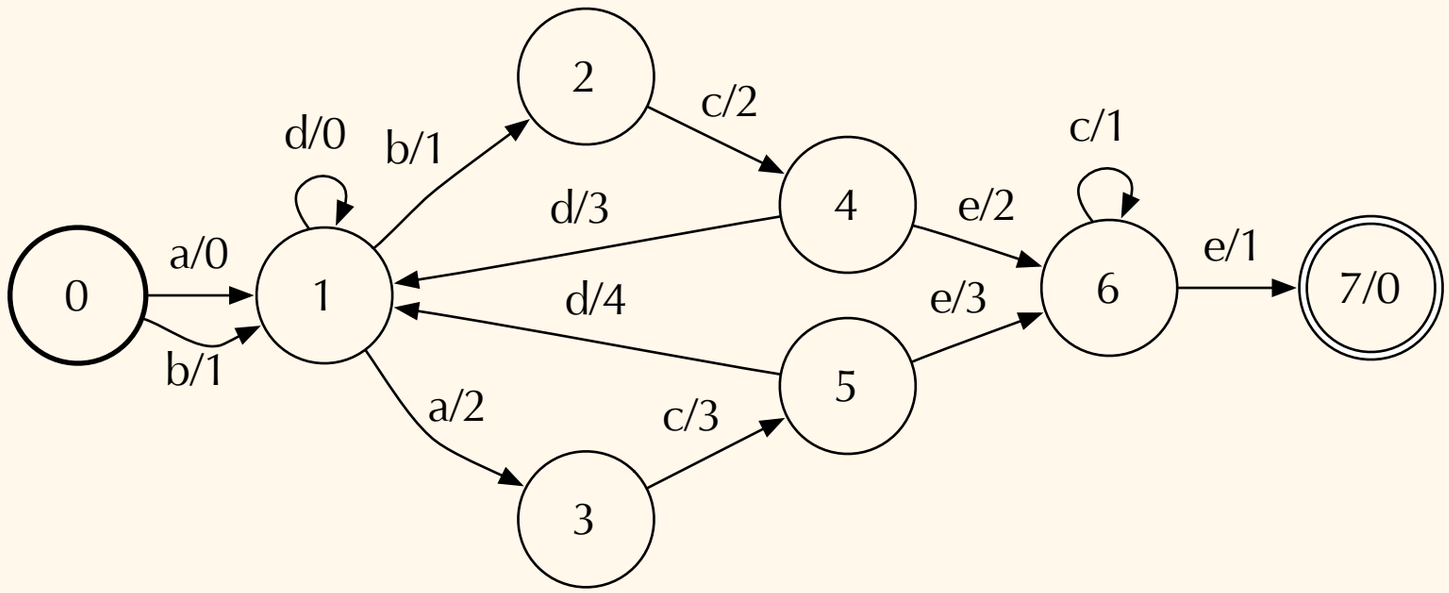
1. Installing OpenFST etc.
2. Revisiting Minimization
3. Binary math
4. Flight routing & Shortest Path
5. Thrax
6. Spelling correction

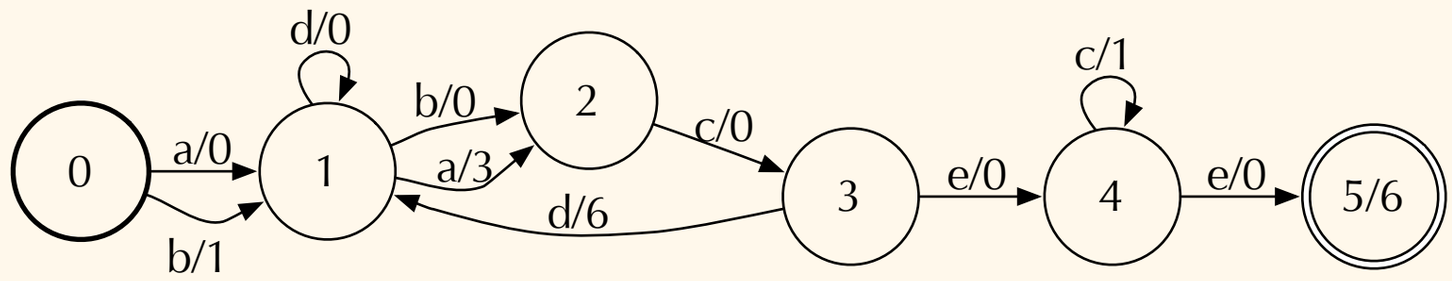
Revisiting minimization:

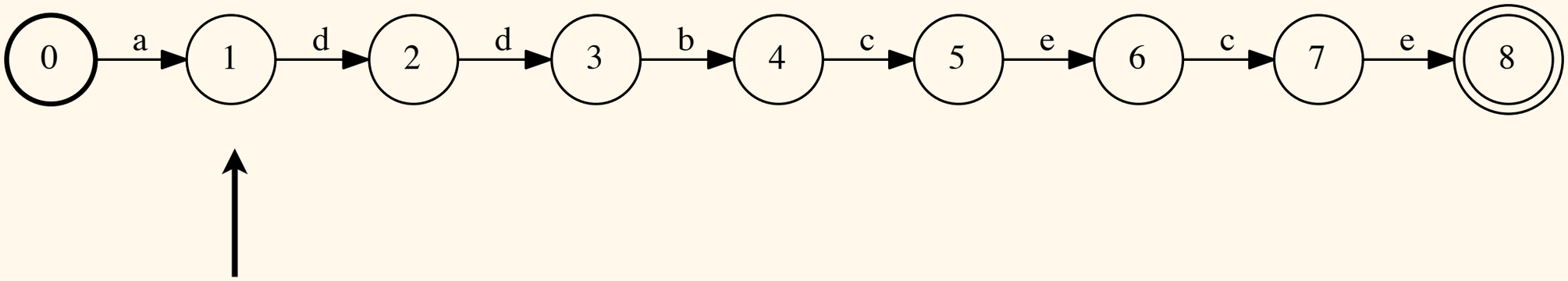


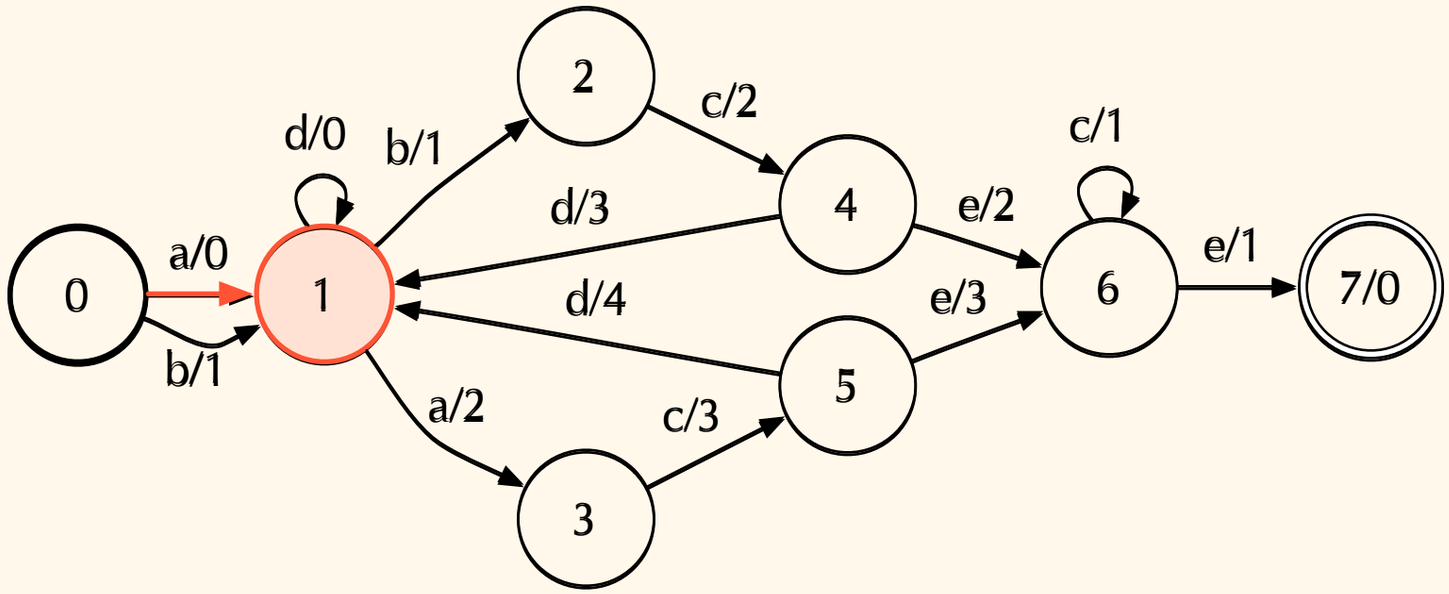



0
a
1
d
2
d
3
b
4
c
5
e
6
c
7
e
8

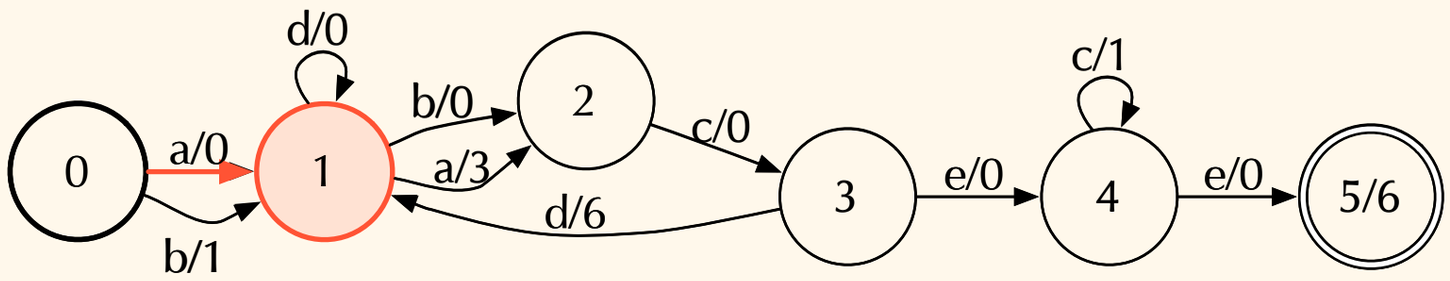

0
a/0
b/1
1
d/0
b/1
2
c/2
d/3
4
e/2
c/1
d/4
e/3
6
e/1
7/0
5
a/2
c/3
3


0
a/0
b/1
1
d/0
b/0
a/3
2
c/0
d/6
3
e/0
4
c/1
e/0
5/6

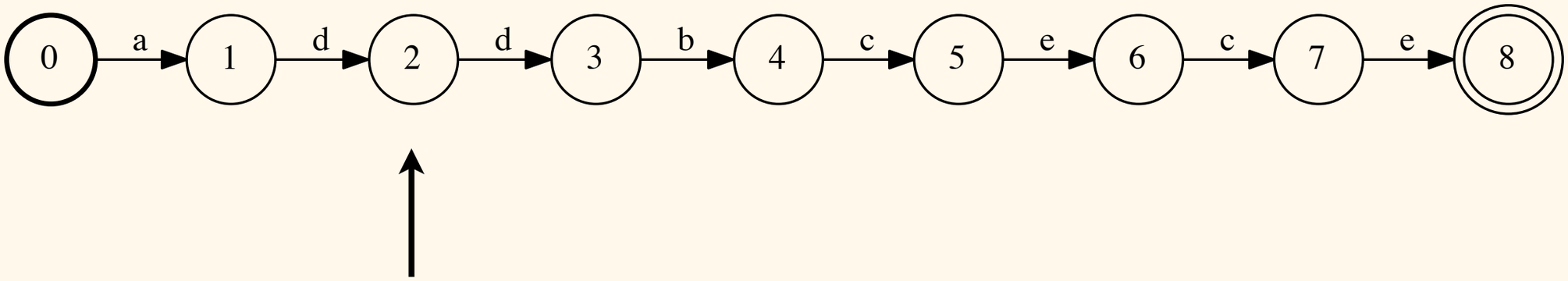


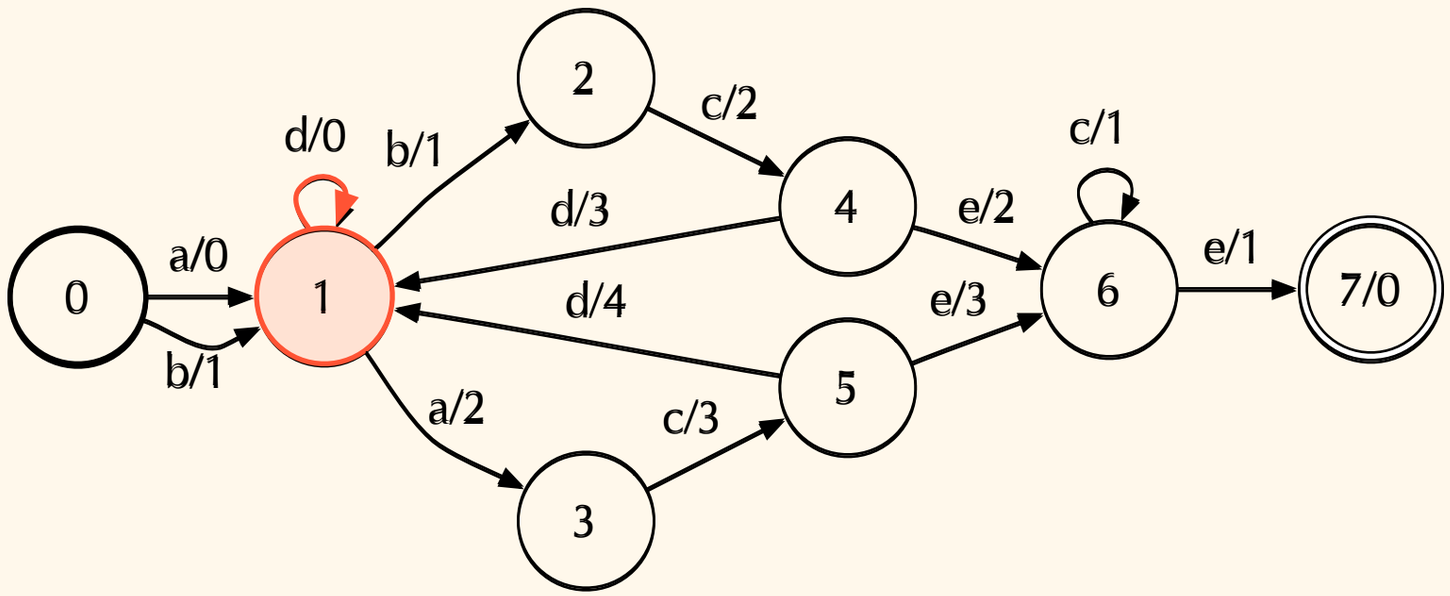


Weight: 0

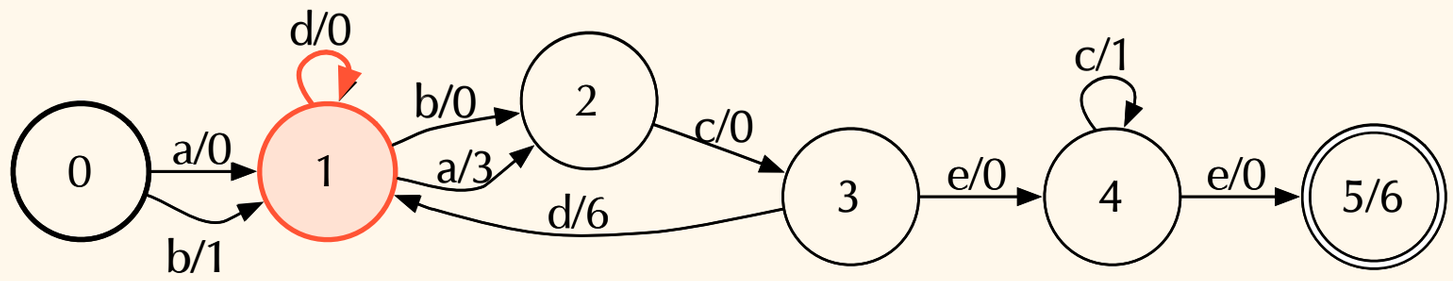


Weight: 0

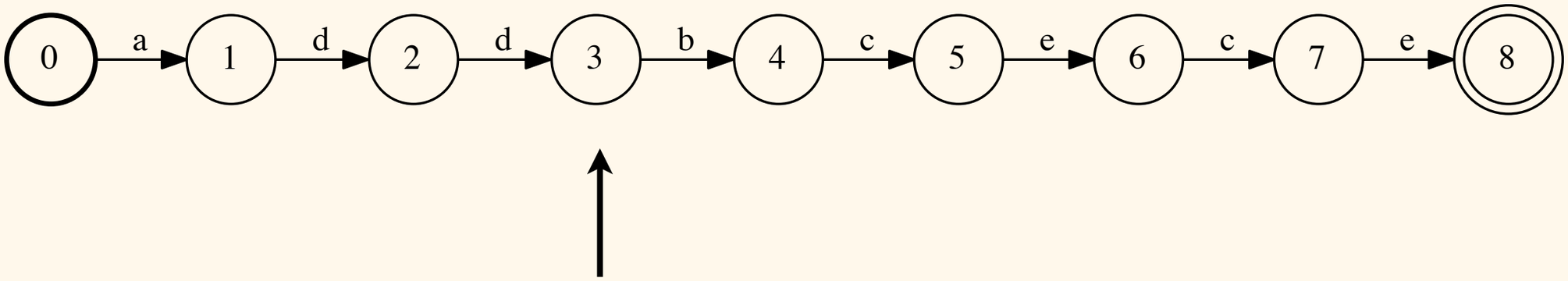


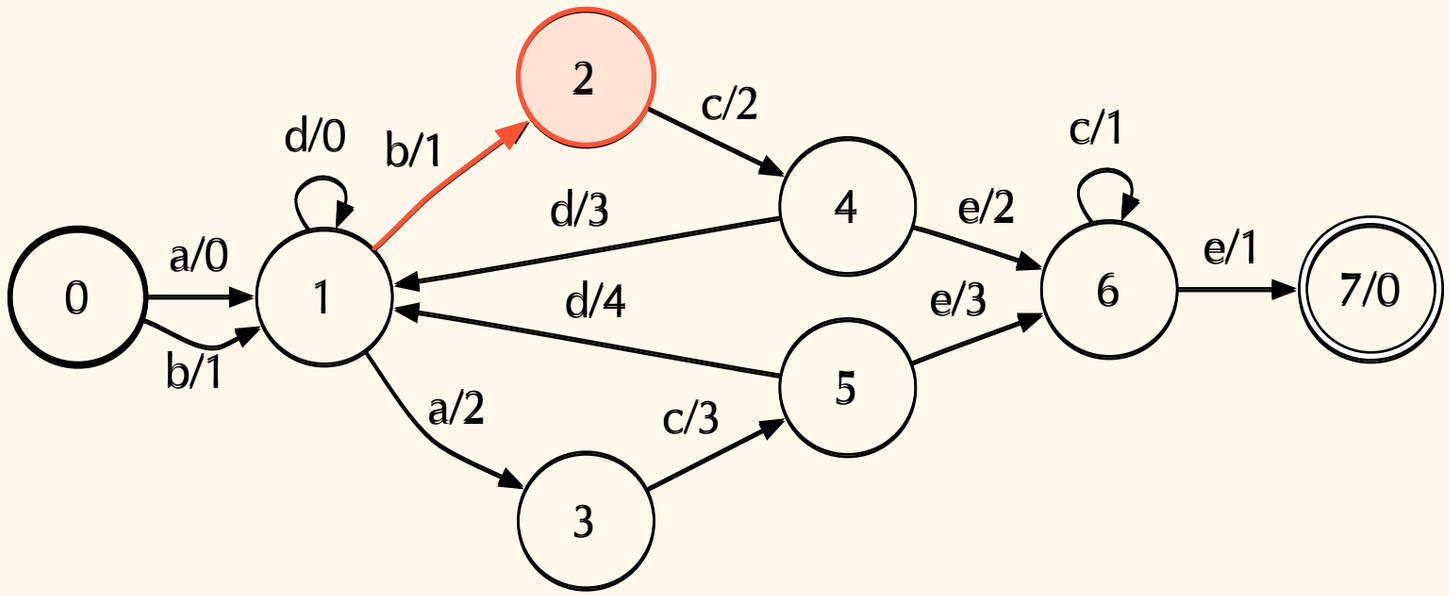


Weight: 0

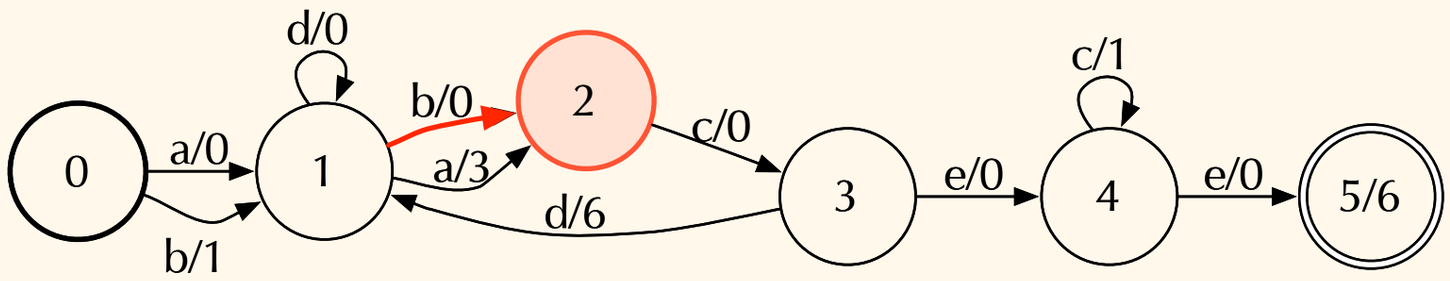


Weight: 0

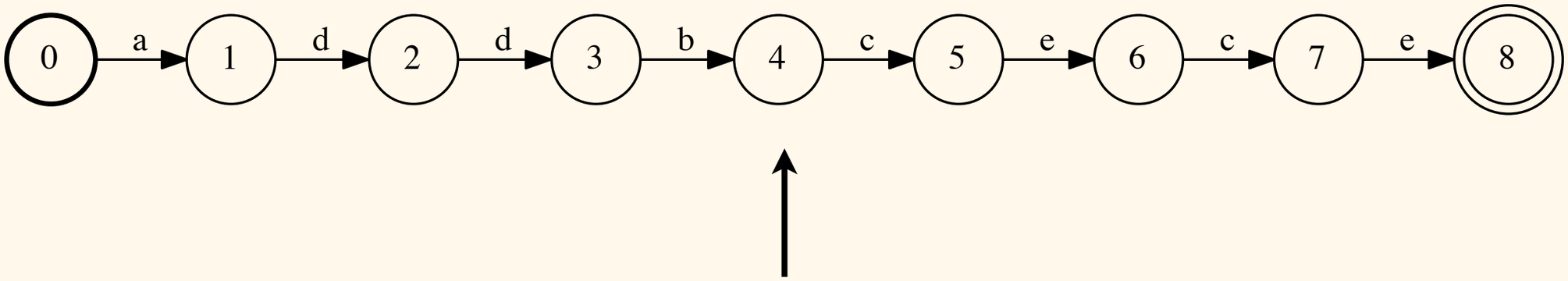


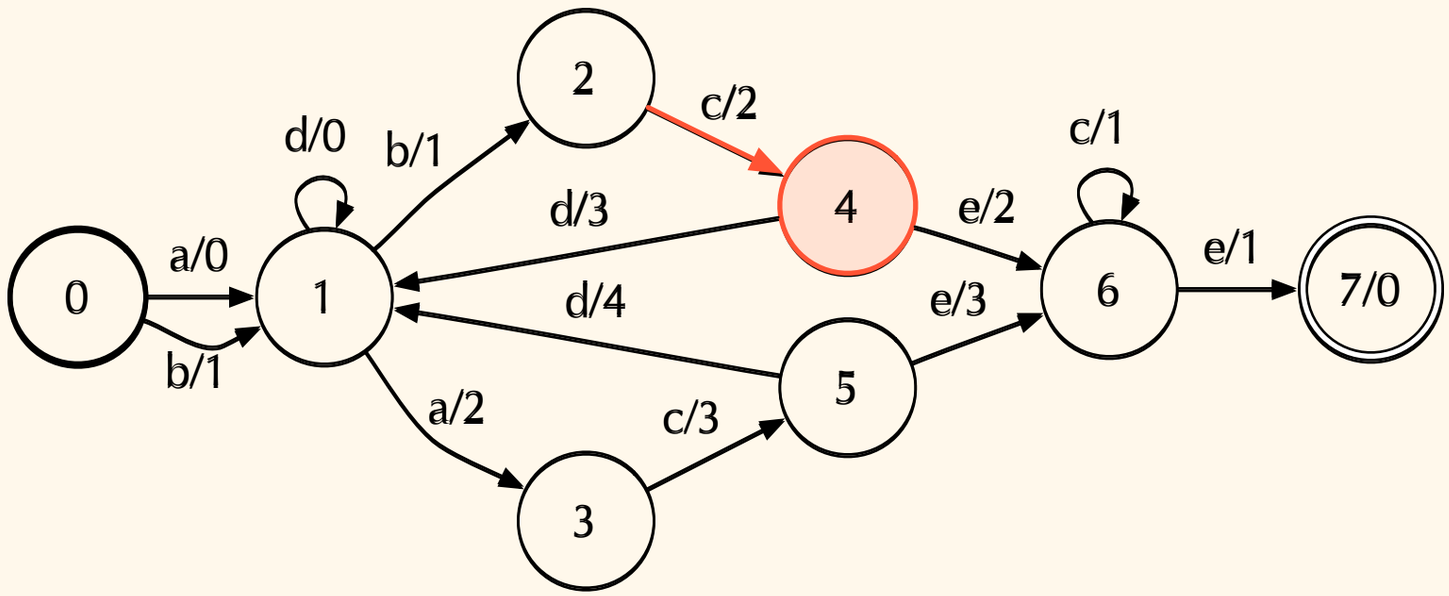


Weight: 1

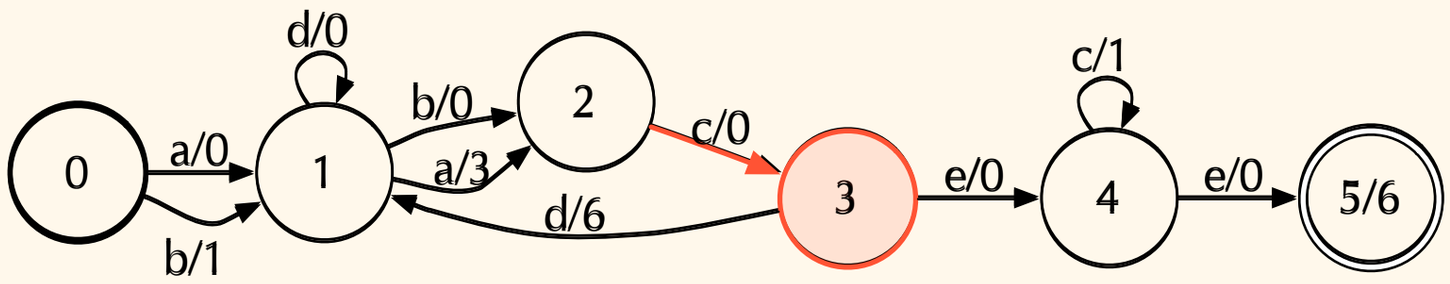


Weight: 0

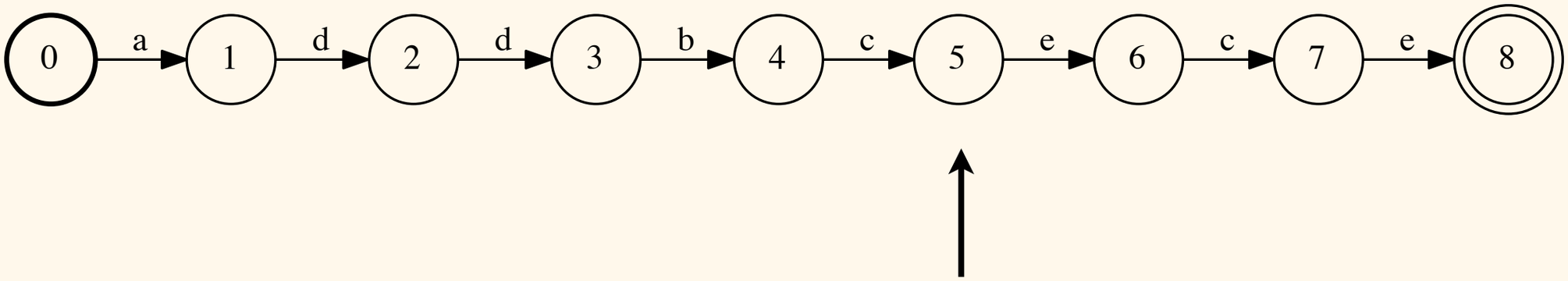


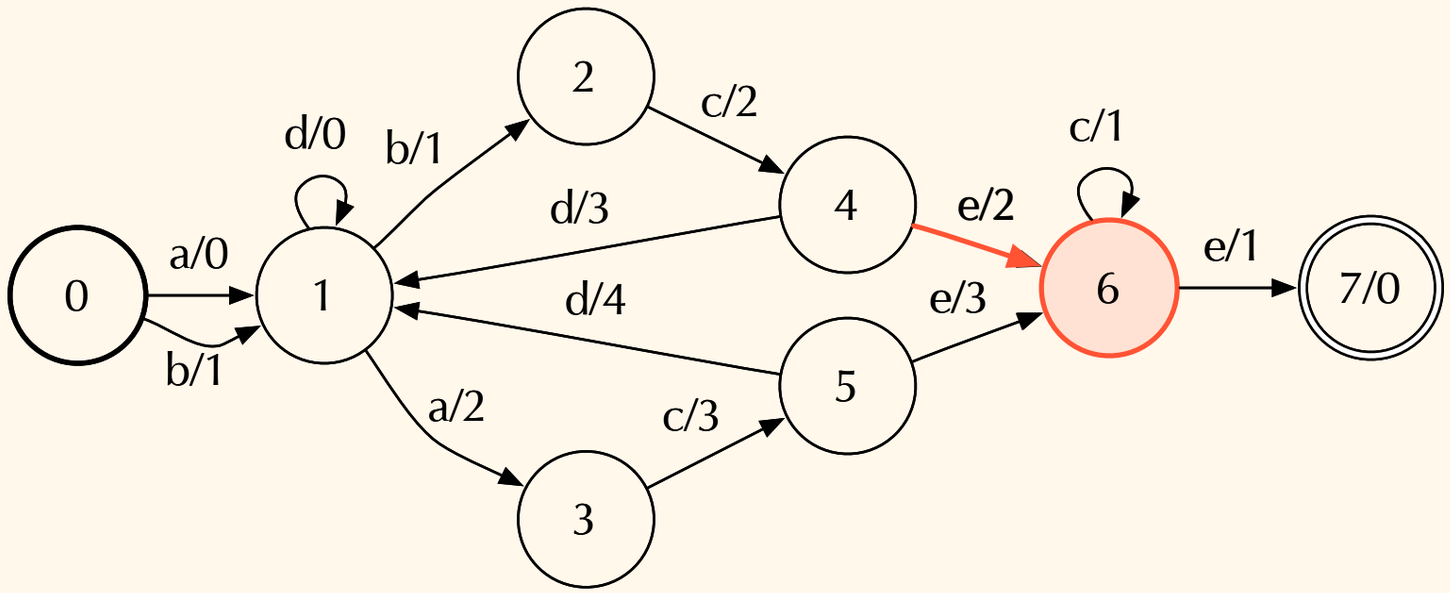


Weight: 1, 2

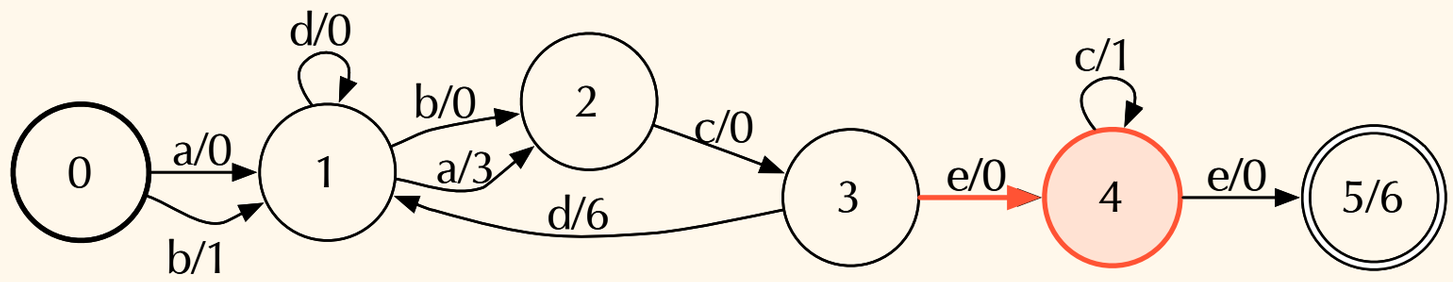


Weight: 0

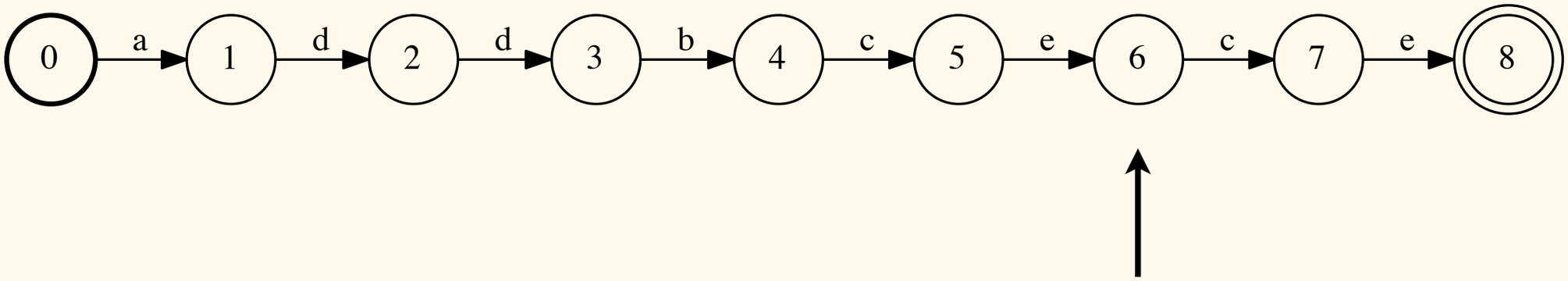


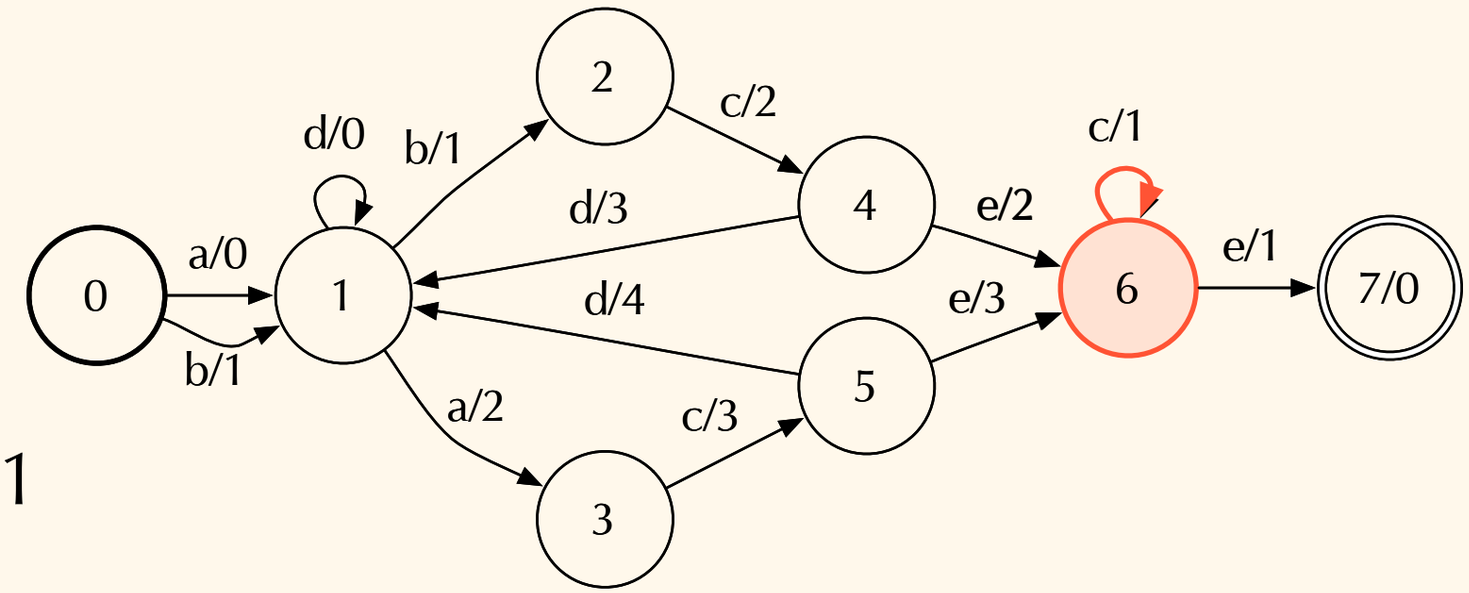


Weight: 1, 2, 2

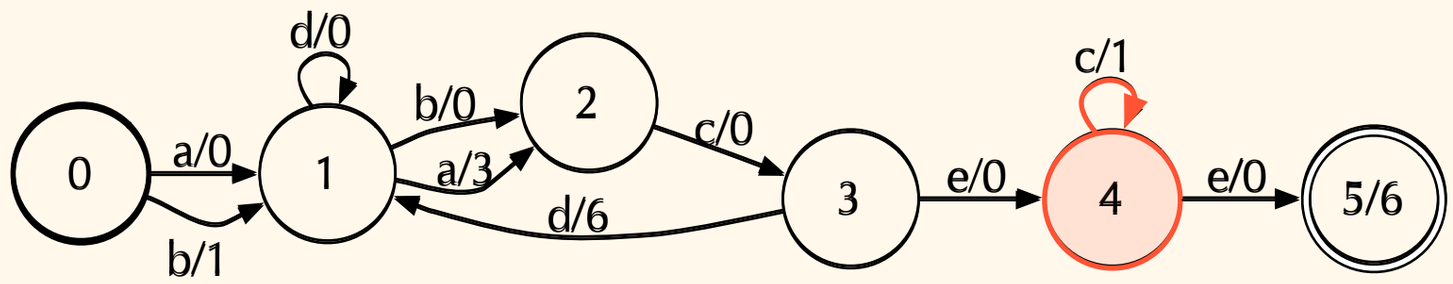


Weight: 0

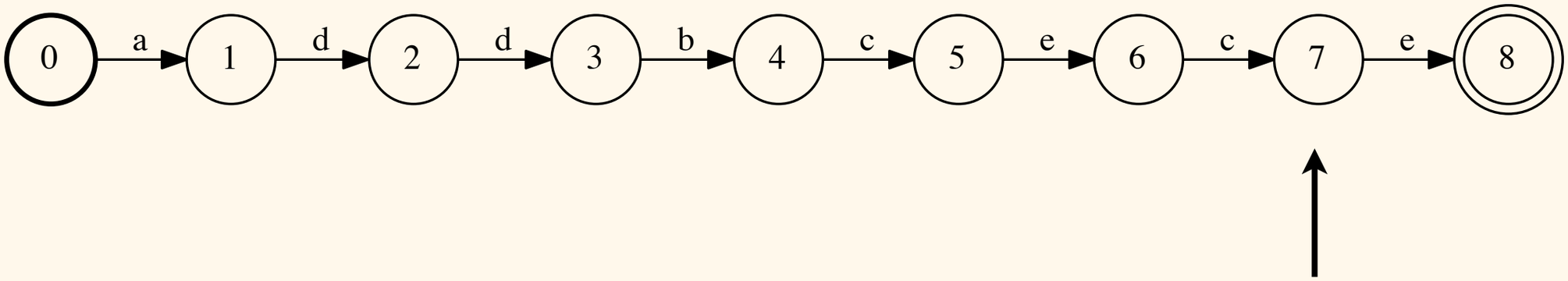


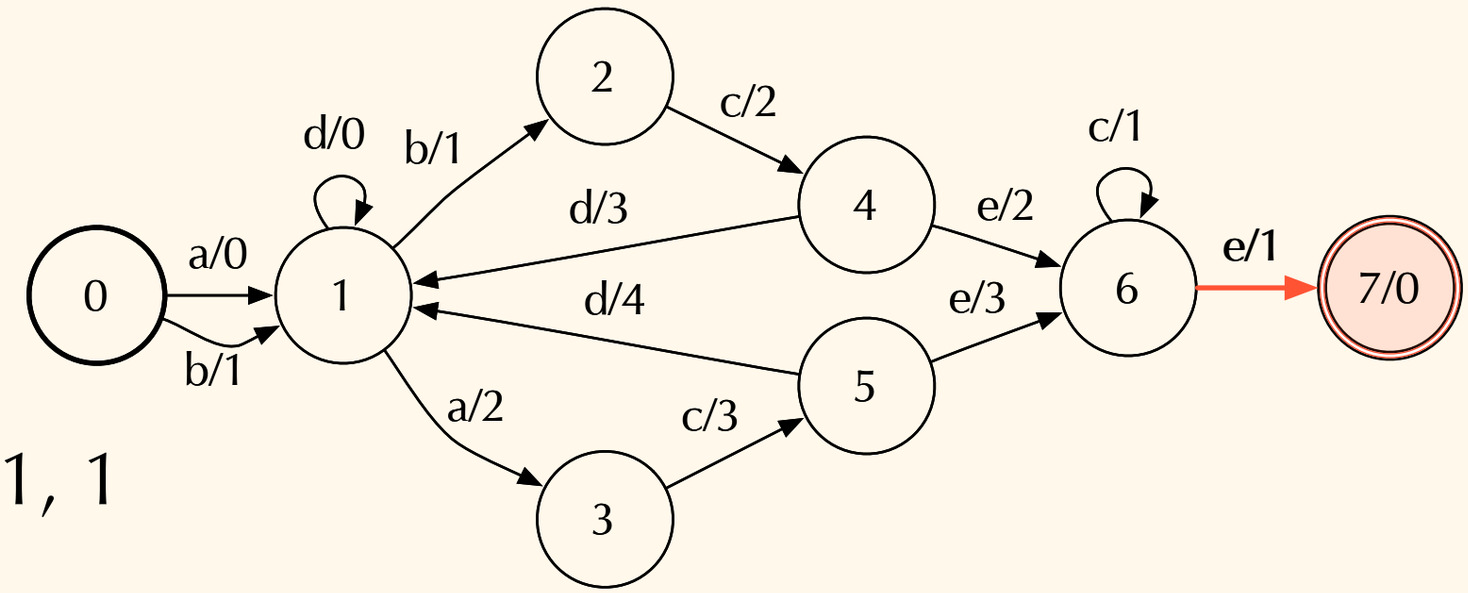


Weight: 1, 2, 2, 1

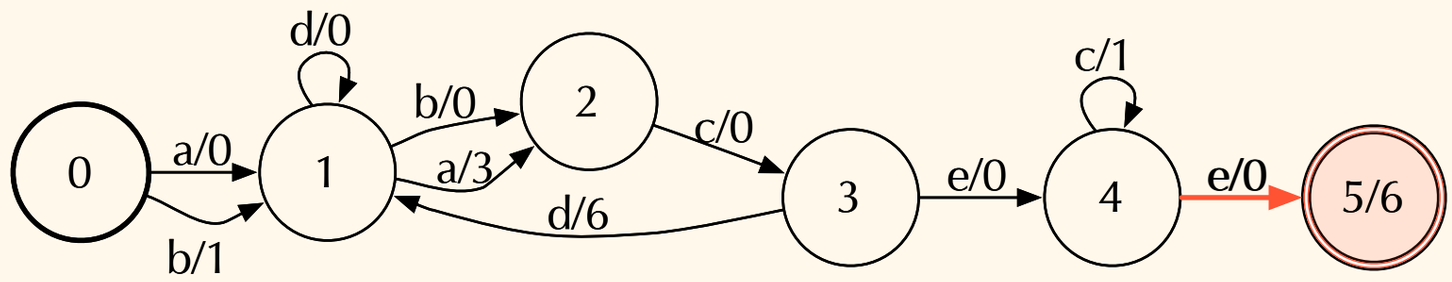


Weight: 1

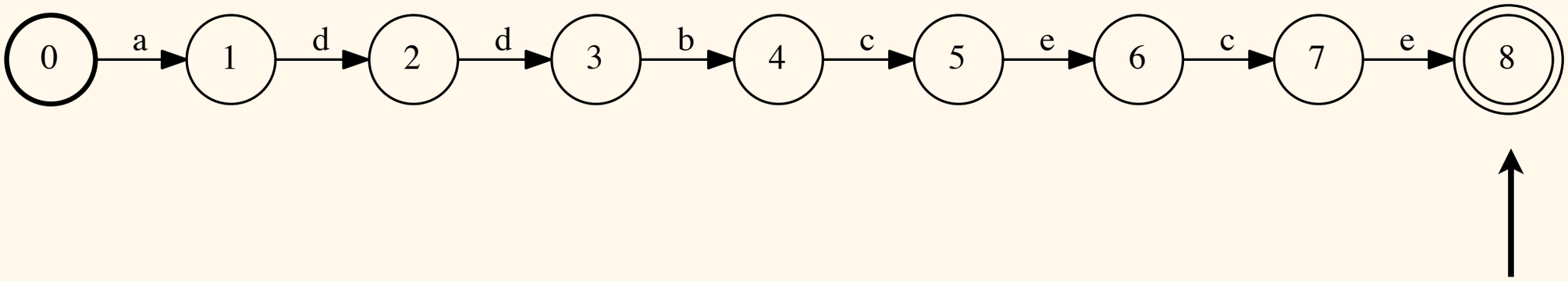


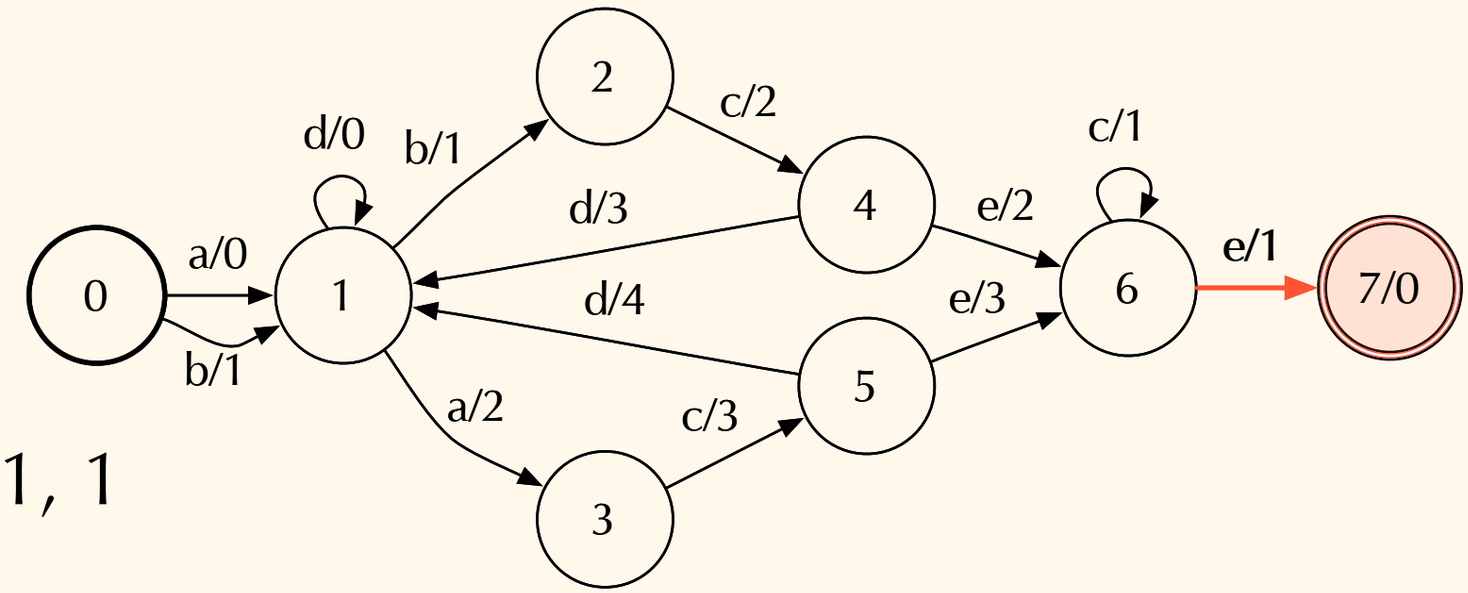


Weight: 1, 2, 2, 1, 1

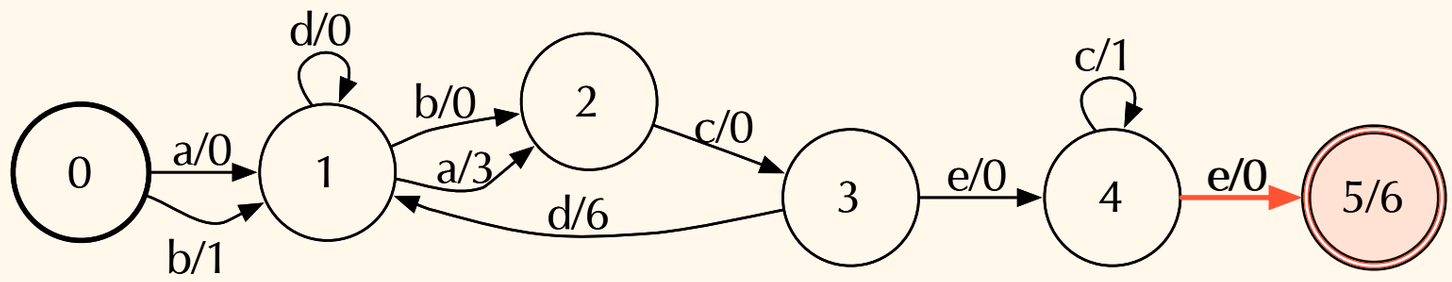


Weight: 1,

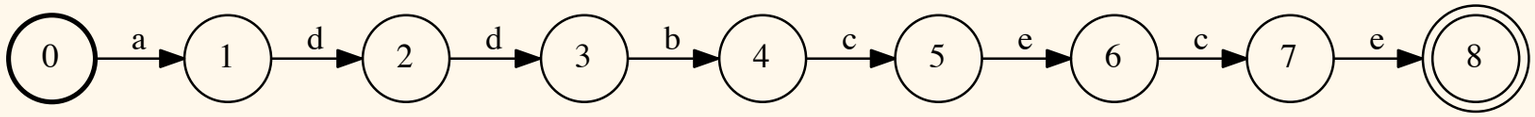


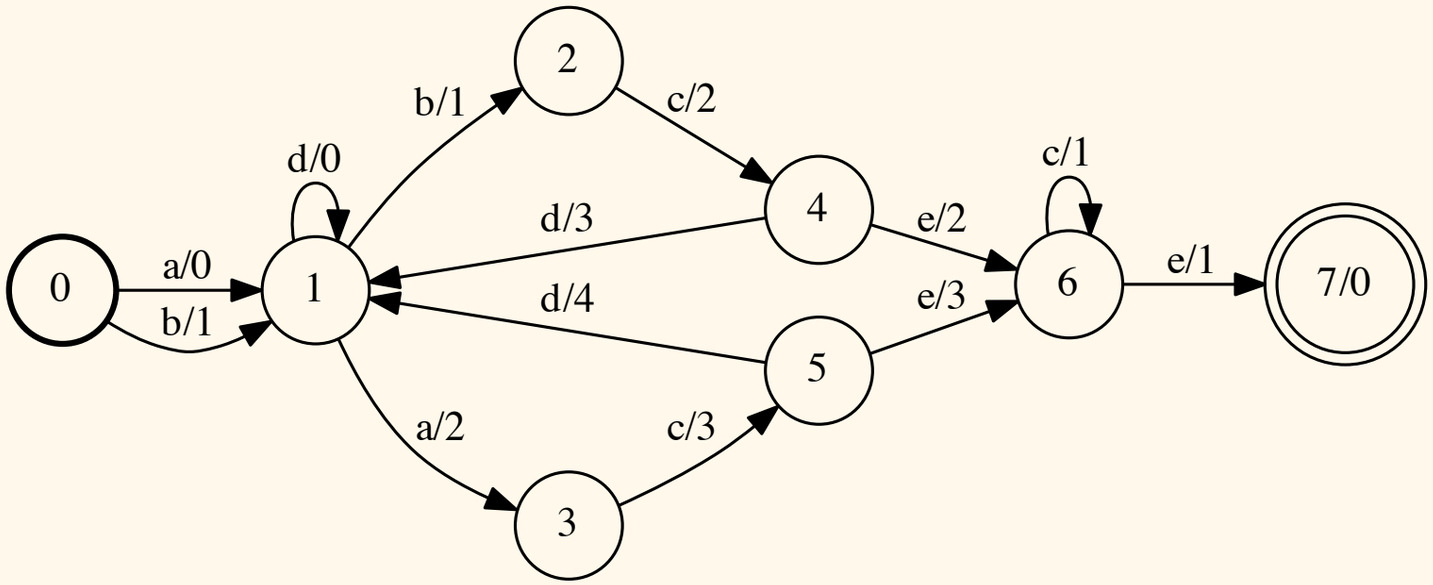


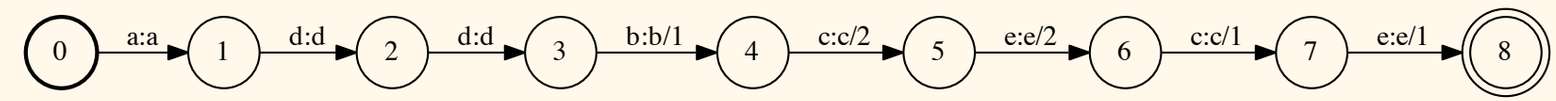
Weight: 1, 2, 2, 1, 1

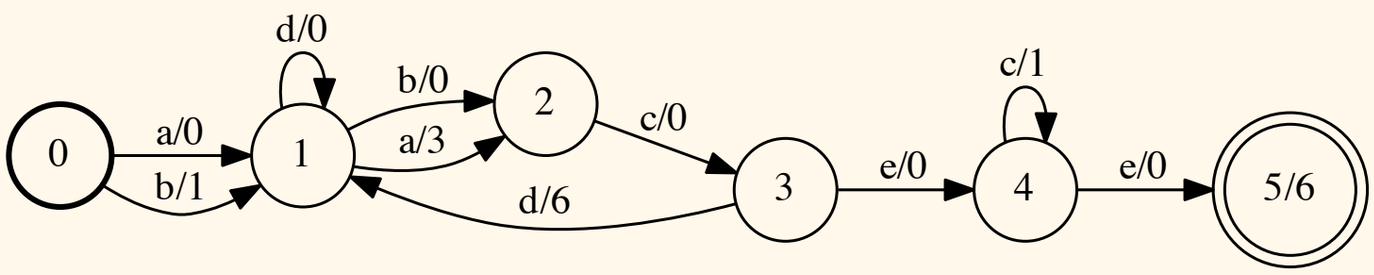


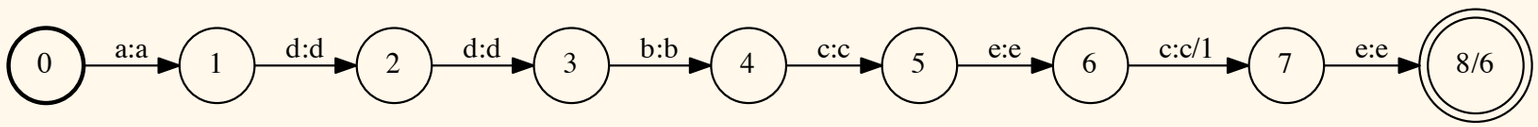
Weight: 1, 6


0
a
1
d
2
d
3
b
4
c
5
e
6
c
7
e
8

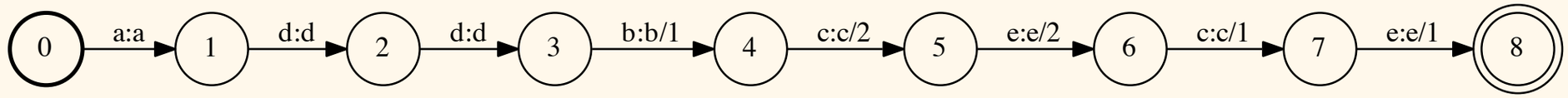

0
a/0
b/1
1
d/0
b/1
2
c/2
4
d/3
d/4
5
a/2
c/3
3
e/2
e/3
c/1
6
e/1
7/0


0
a:a
1
d:d
2
d:d
3
b:b/1
4
c:c/2
5
e:e/2
6
c:c/1
7
e:e/1
8

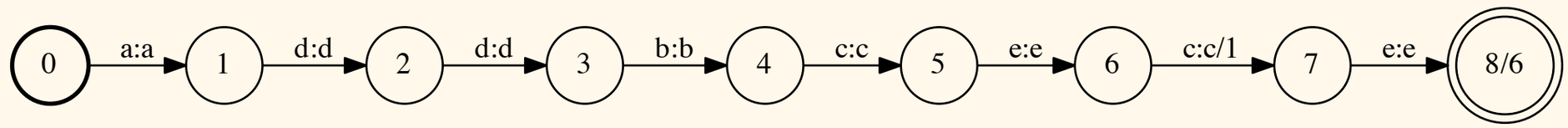

0
a/0
b/1
1
d/0
b/0
a/3
2
c/0
d/6
3
e/0
4
c/1
e/0
5/6


0
a:a
1
d:d
2
d:d
3
b:b
4
c:c
5
e:e
6
c:c/1
7
e:e
8/6

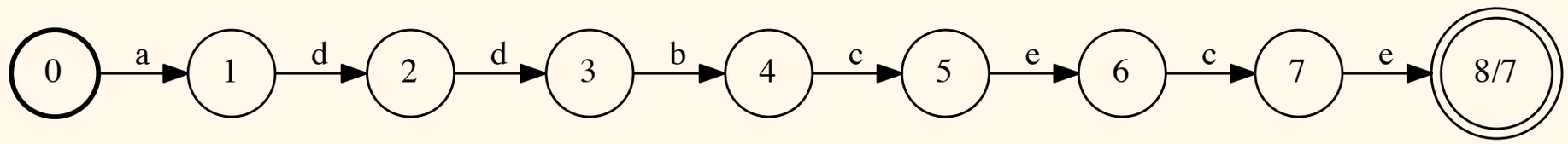
```
fstcompose input.fst first.fst
```




```
fstcompose input.fst second.fst
```




```
fstpush --push_weights=true --to_final=true | fstdraw --acceptor=true
```



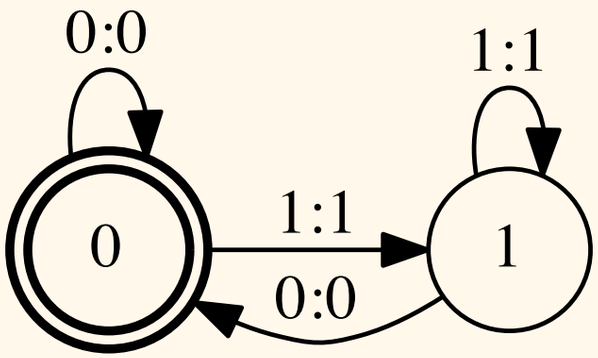

```
$ fstequivalent input_first.fst input_second.fst
$ fstequivalent --help | head
Two DFAs are equivalent iff the exit status is zero.

  Usage: fstequivalent in1.fst in2.fst
...
```

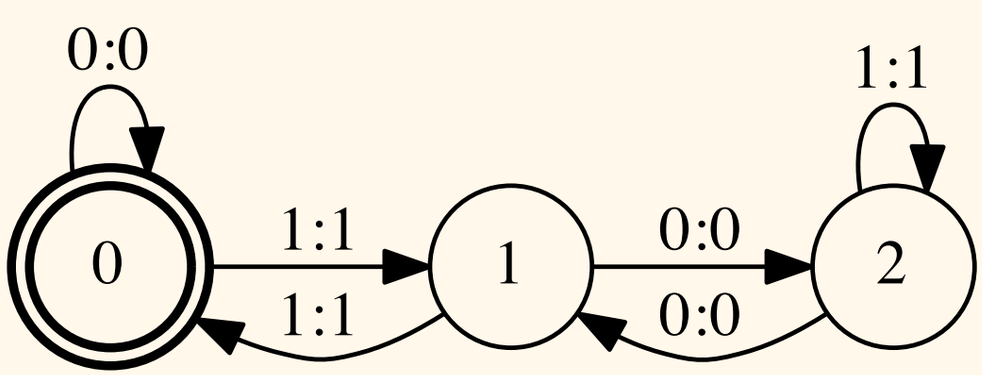
Plan for the day:

1. Installing OpenFST etc.
2. Revisiting Minimization
3. Binary math
4. Flight routing & Shortest Path
5. Thrax
6. Spelling correction

Fun FST trick: binary math



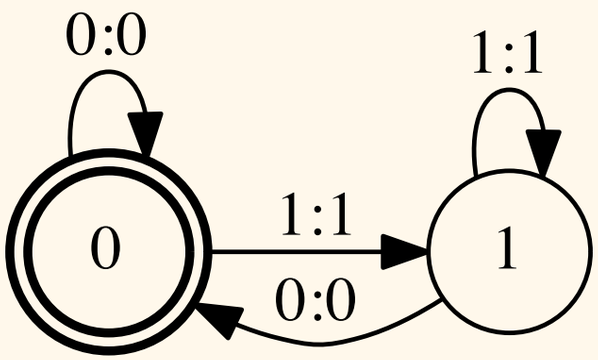
This machine represents multiples of two...

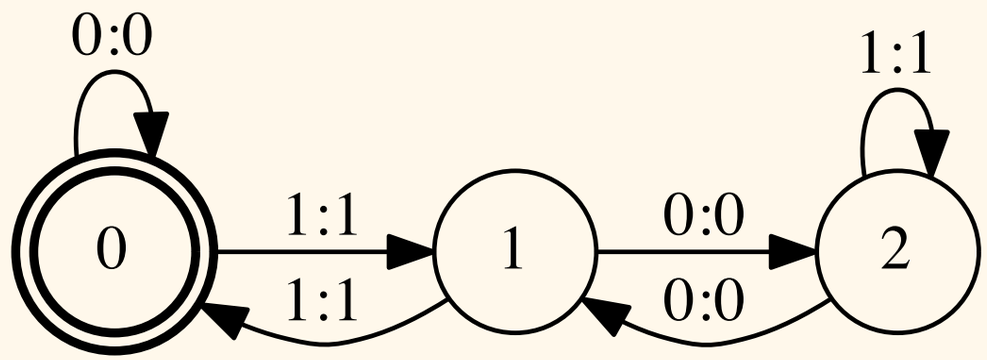


... this one represents multiples of three...

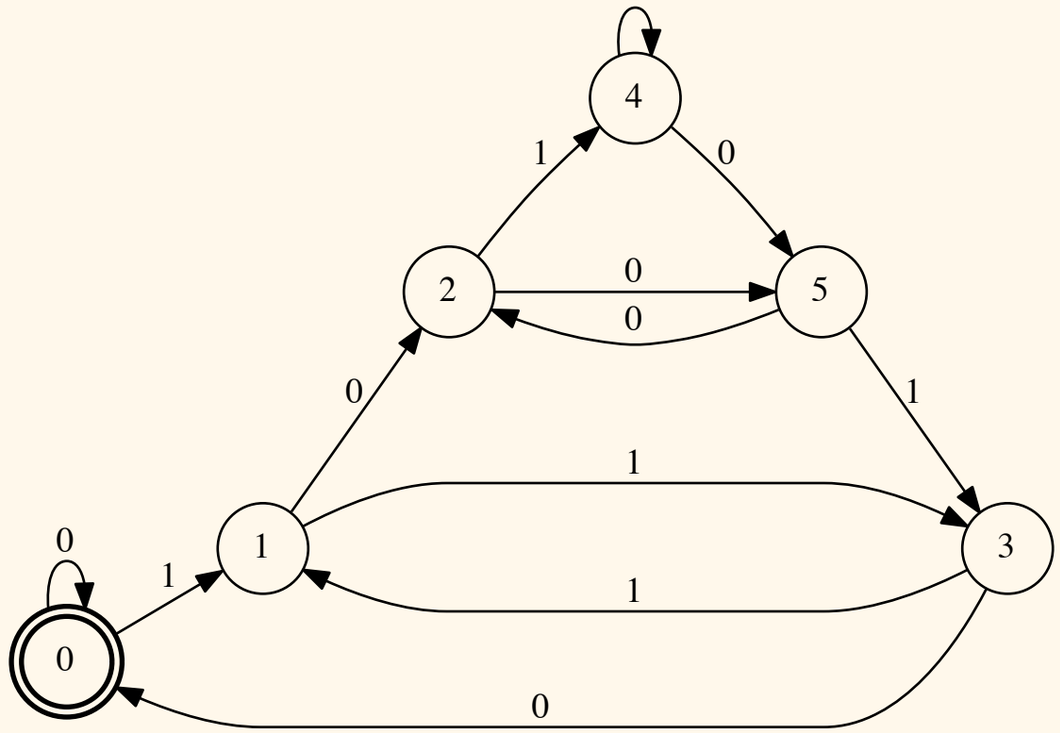
Example from Roche & Yves's "Introduction" in "Finite State Language Processing".

Fun FST trick: binary math





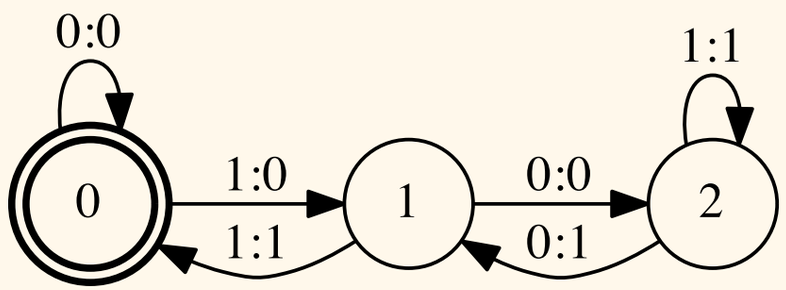
Composing them together gives us:



... a machine representing binary multiples of six!

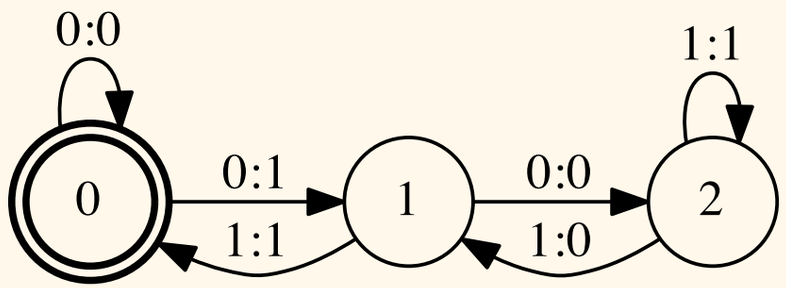
Fun FST trick: binary math

Those are acceptors (FSAs)... let's make a transducer:



This machine divides by three...

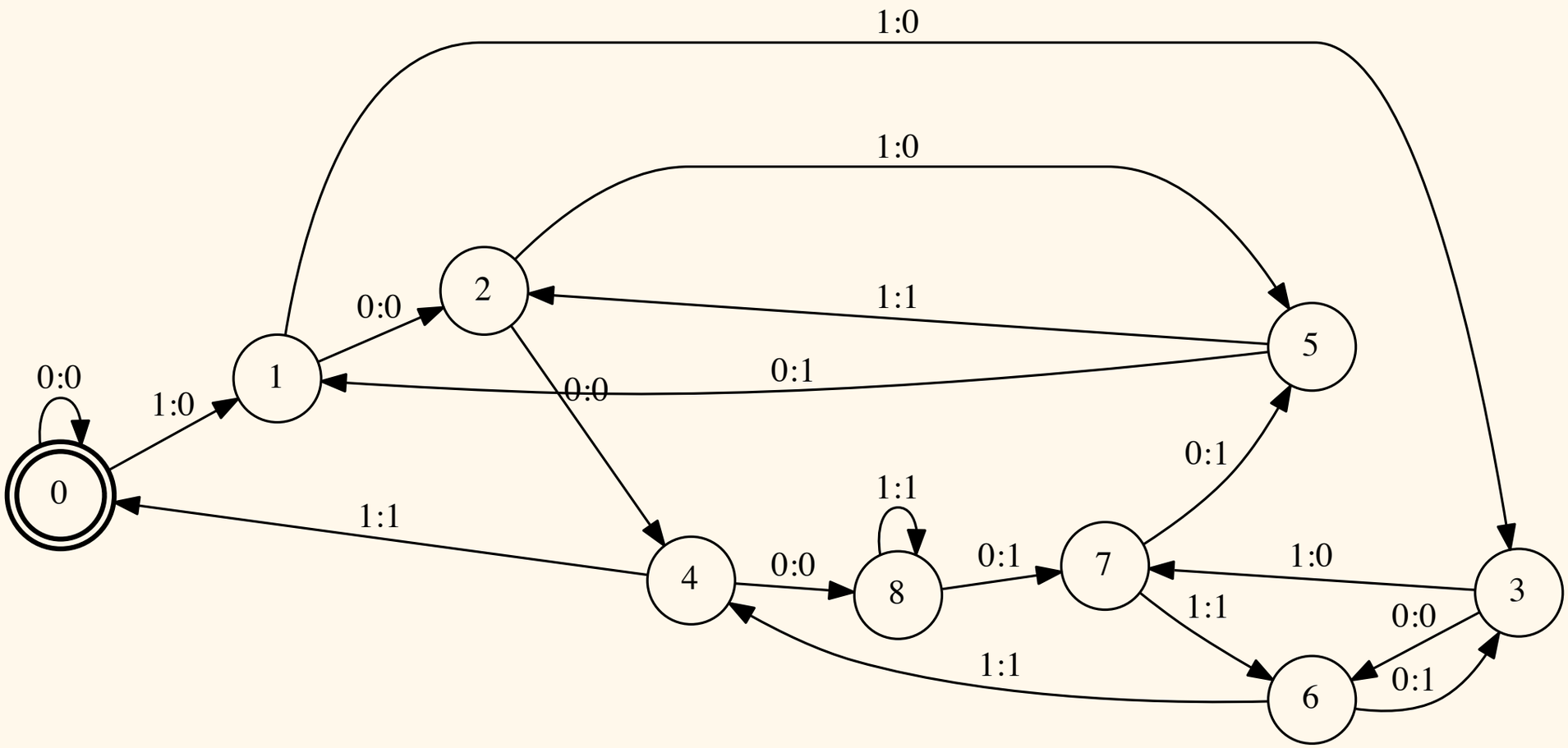
What happens if we invert this transducer?



Multiplication!

Fun FST trick: binary math

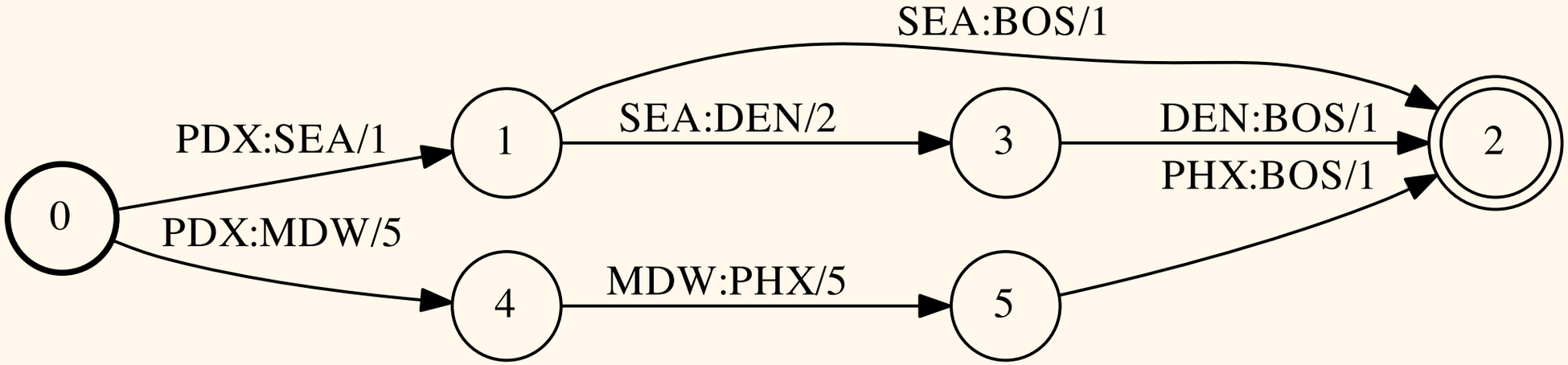
Of course, composing our division transducers together does what we expect:



div3 ∘ div3 = div9

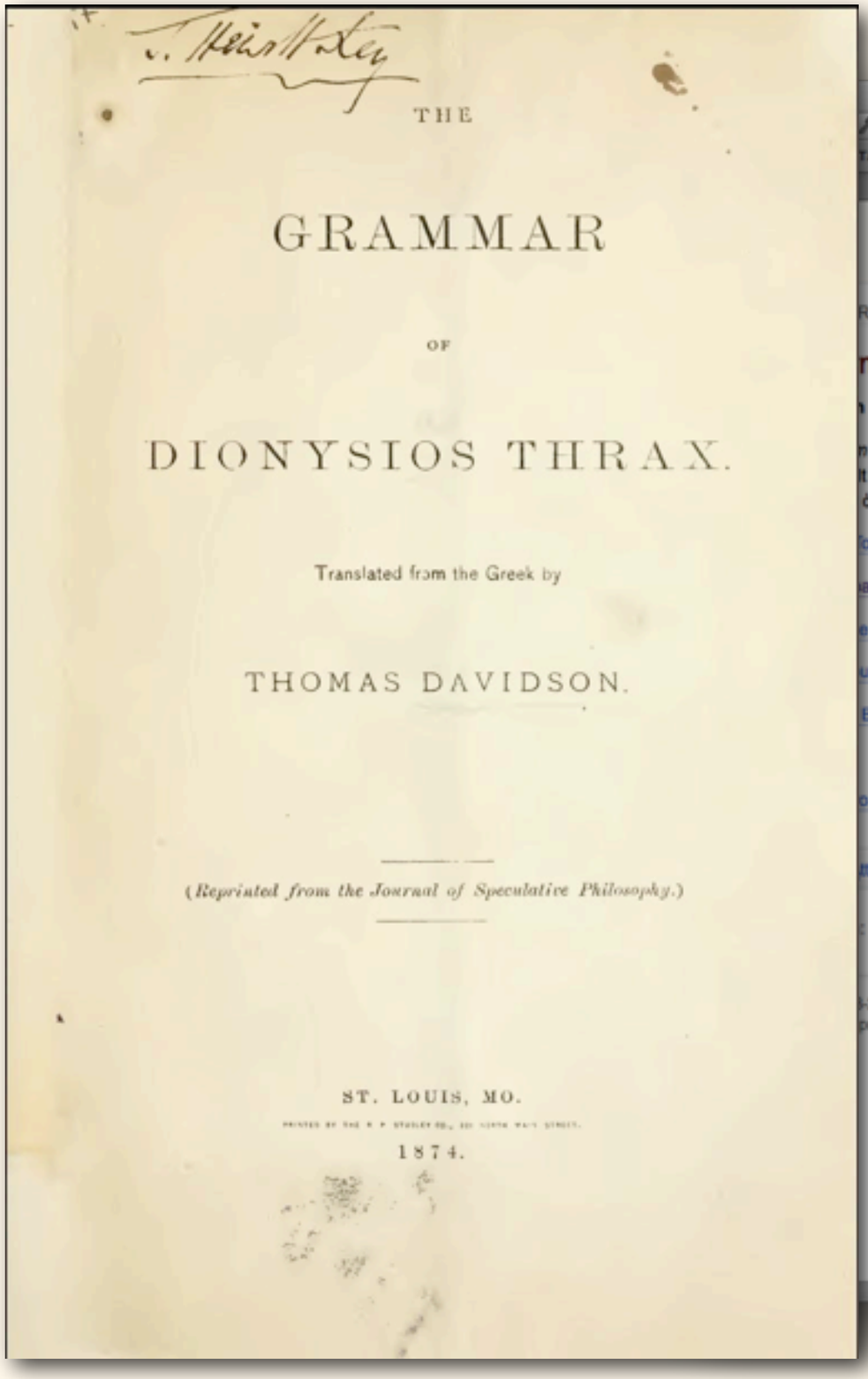
Plan for the day:

1. Installing OpenFST etc.
2. Revisiting Minimization
3. Binary math
4. Flight routing & Shortest Path
5. Thrax
6. Spelling correction


SEA:BOS/1
PDX:SEA/1
SEA:DEN/2
DEN:BOS/1
PHX:BOS/1
PDX:MDW/5
MDW:PHX/5
0
1
2
3
4
5

Plan for the day:

1. Installing OpenFST etc.
2. Revisiting Minimization
3. Binary math
4. Flight routing & Shortest Path
5. Thrax
6. Spelling correction



THE

GRAMMAR

OF

DIONYSIOS THRAX.

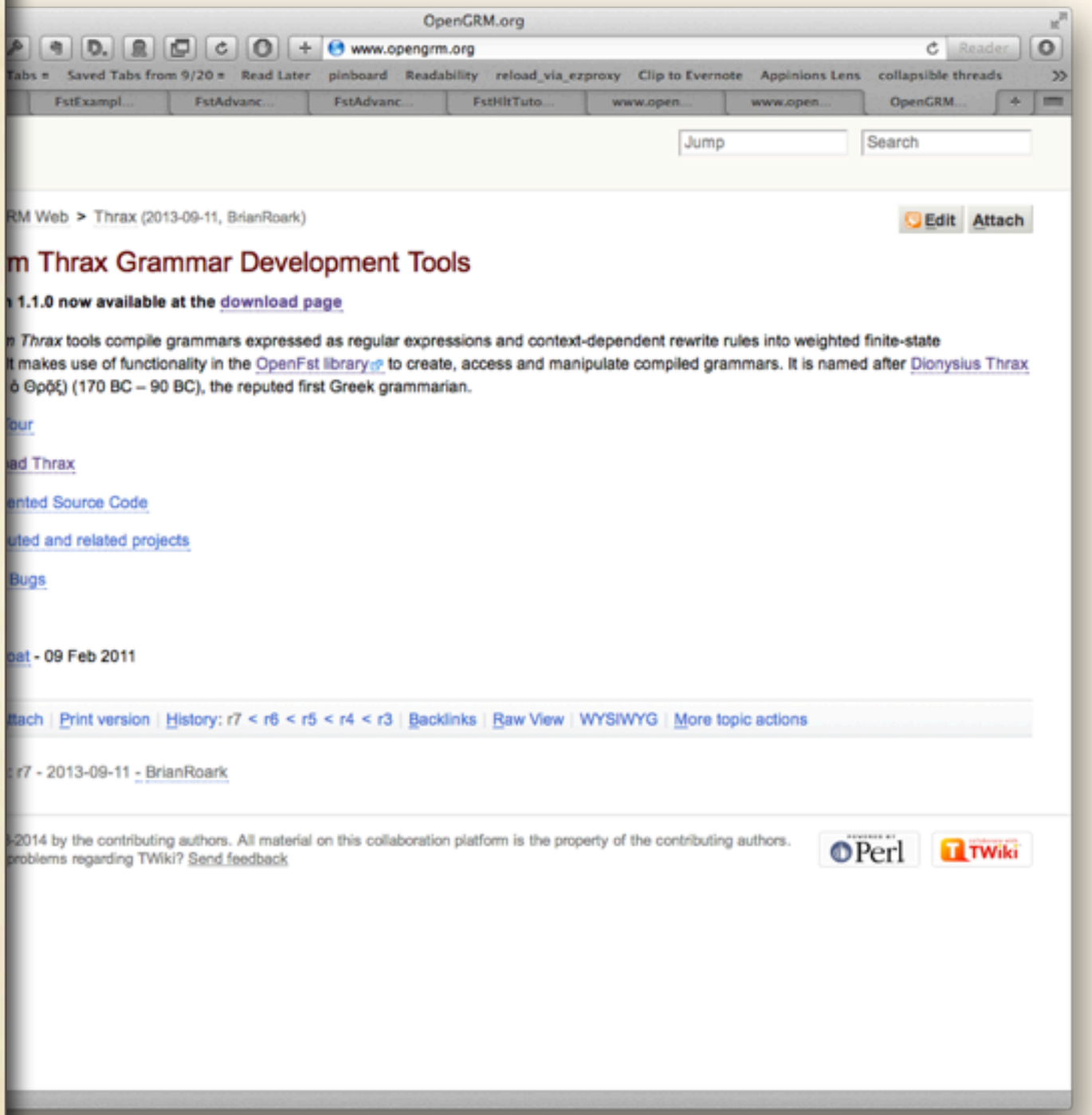
Translated from the Greek by

THOMAS DAVIDSON.

(*Reprinted from the Journal of Speculative Philosophy.*)

ST. LOUIS, MO.

1874.



Plan for the day:

1. Installing OpenFST etc.
2. Revisiting Minimization
3. Binary math
4. Flight routing & Shortest Path
5. Thrax
6. Spelling correction

To the terminal!