Overview

1. Description of the Finite-State Machine (FSM) Toolkit
2. Applications and Examples
3. Rules of Thumb

Part 1. – Finite-State Machine (FSM) Toolkit

The FSM tools 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.fsa");
Fsm in2 = FSMLoad("in2.fsa");
Fsm out = FSMIntersect(fsm1, fsm2);
FSMDump("out.fsa", out);
```

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

FSM File Types

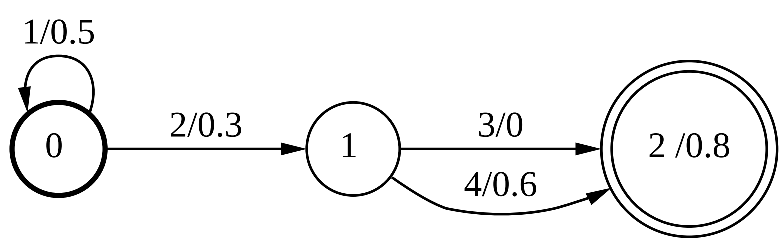
- **Textual Format:** Used for manual inputting and viewing of FSMs
  - Acceptor Files
  - Transducer Files
  - Symbols Files
- **Binary Format:** 'Compiled' representation used by all FSM utilities.

Acceptor Files

- **Textual Representation** (`A.txt`):

SRC	DST	LAB	CST
0	0	1	.5
0	1	2	.3
1	2	3	
1	2	4	.6
2	.8		
FIN	CST		

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

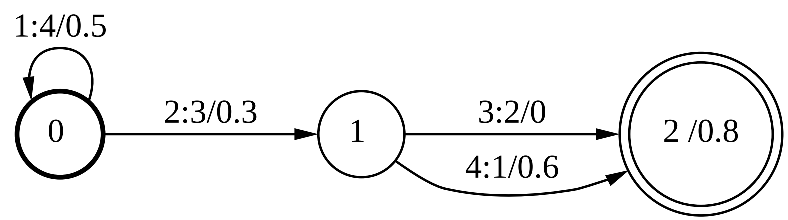


Transducer Files

- **Textual Representation** (`T.txt`):

SRC	DST	ILAB	OLAB	CST
0	0	1	4	.5
0	1	2	3	.3
1	2	3	2	
1	2	4	1	.6
2	.8			
FIN	CST			

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



Acceptor and Symbols Files

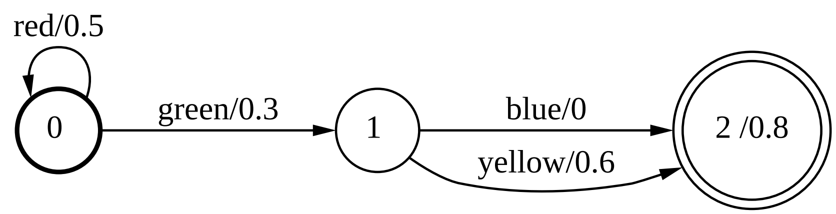
- **Acceptor File** (`A.stxt`):

```
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
```

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



Transducer and Symbols Files

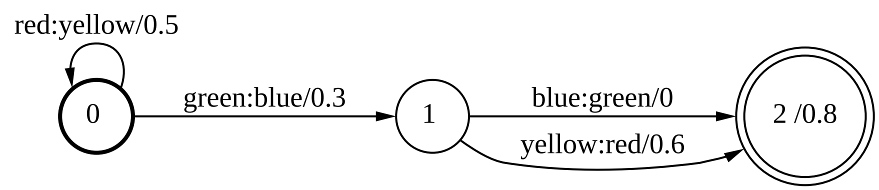
- **Transducer File** (`T.stxt`):

```
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
```

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



Compiling, Printing, and Drawing FSMs

- Compiling from textual representations

```
fsmcompile <A.txt >A.fsa
fsmcompile -t <T.txt >T.fst
```

- Printing of binary representations

```
fsmprint <A.fsa >A.txt
fsmprint <T.fst >T.txt
```

- Drawing of binary representations

```
fsmdraw <A.fsa | dot -Tps >A.ps
fsmdraw <T.fst | dot -Tps >T.ps
```

Compiling, Printing, and Drawing FSMs with Symbols

- Compiling from textual representations

```
fsmcompile -iA.syms <A.stxt >A.fsa
fsmcompile -iA.syms -oA.syms -t <T.stxt >T.fst
```

- Printing of binary representations

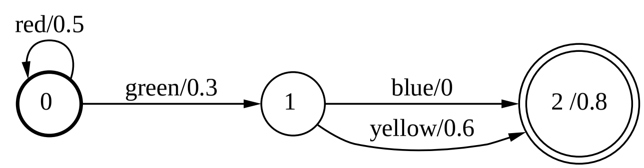
```
fsmprint -iA.syms <A.fsa >A.stxt
fsmprint -iA.syms -oA.syms <T.fst >T.stxt
```

- Drawing of binary representations

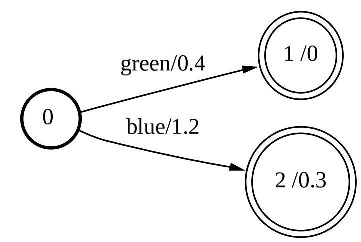
```
fsmdraw -iA.syms <A.fsa | dot -Tps >A.sps
fsmdraw -iA.syms -oA.syms <T.fst | dot -Tps >T.sps
```

Union (Sum)

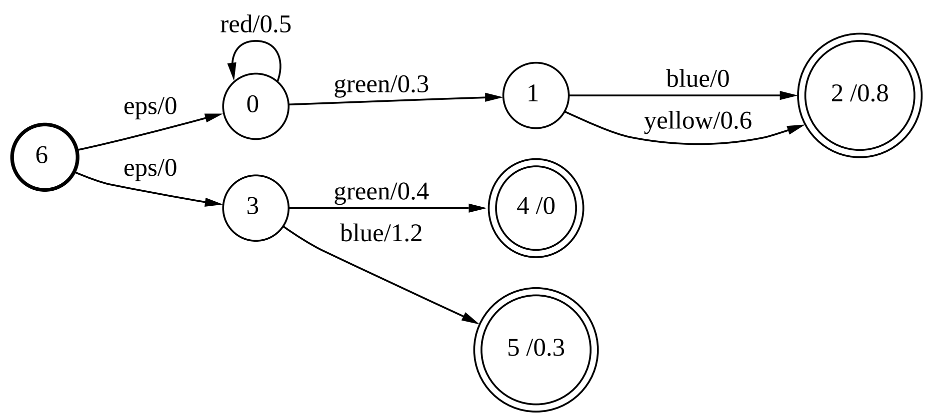
- **Equation:** $C = A \cup B \quad (C = A + B)$
- **Program:** `fsmunion A.fsa B.fsa >C.fsa`
- **Graphical Representation:**



A.fsa



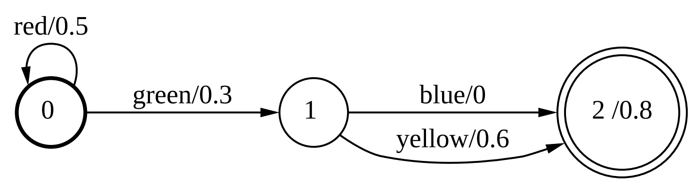
B.fsa



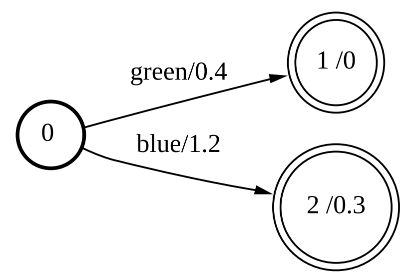
C.fsa

Concatenation (Product)

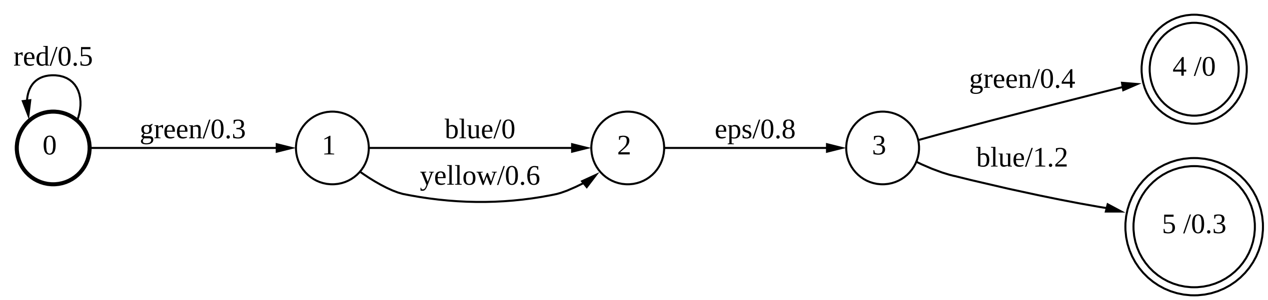
- **Equation:** $C = AB$
- **Program:** `fsmconcat A.fsa B.fsa >C.fsa`
- **Graphical Representation:**



A.fsa



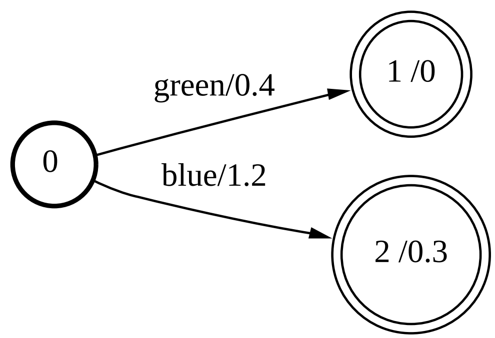
B.fsa



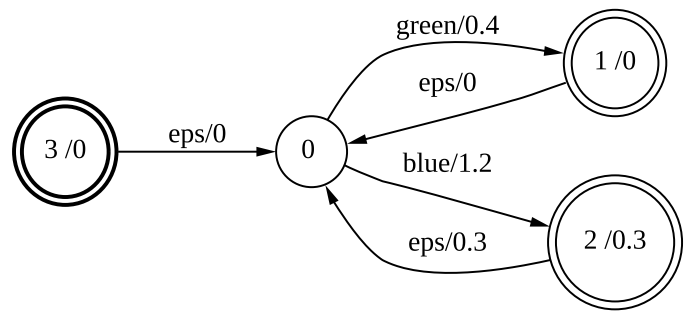
C.fsa

(Concatenative) Closure

- **Equation:** $C = B^* = B^0 + B^1 + B^2 \ldots$
- **Program:** `fsmclosure B.fsa >C.fsa`
- **Graphical Representation:**



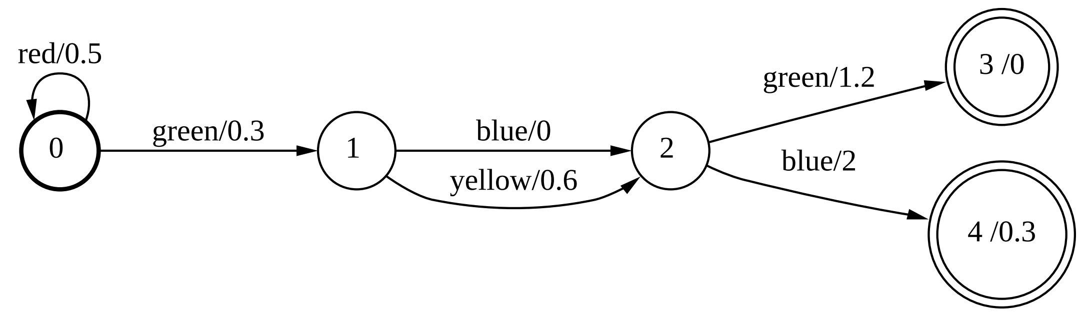
B.fsa



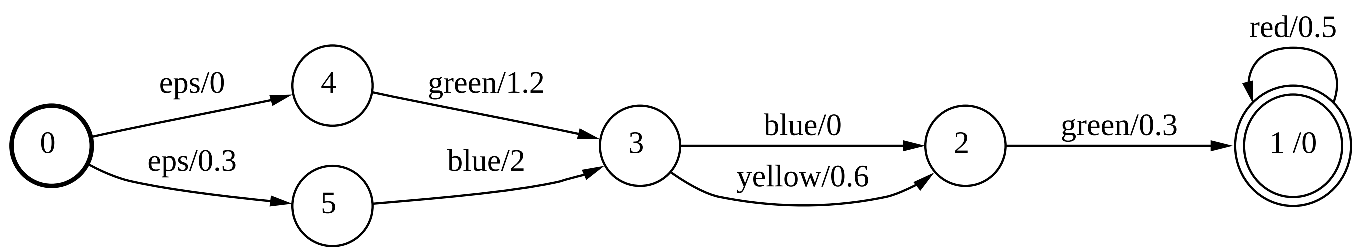
C.fsa

Reversal

- **Equation:** $C = A^r$
- **Program:** `fsmreverse A.fsa >C.fsa`
- **Graphical Representation:**



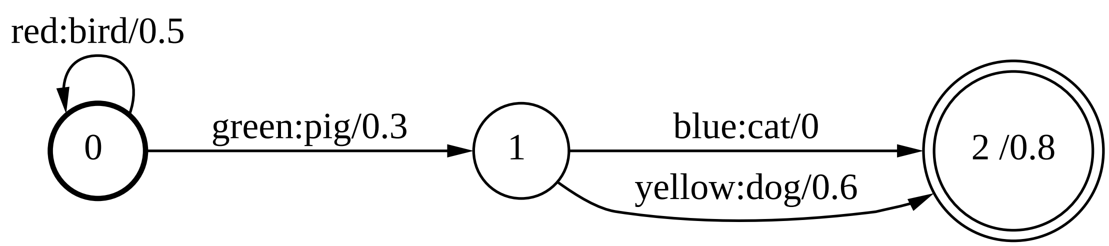
A.fsa



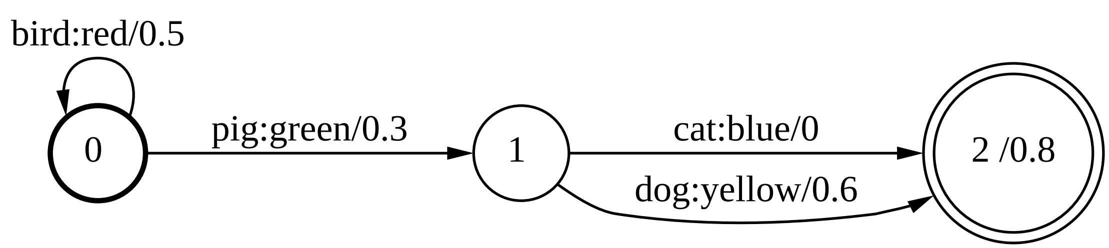
C.fsa

Inversion

- **Equation:** $C = A^{-1}$
- **Program:** `fsminvert A.fst >C.fst`
- **Graphical Representation:**



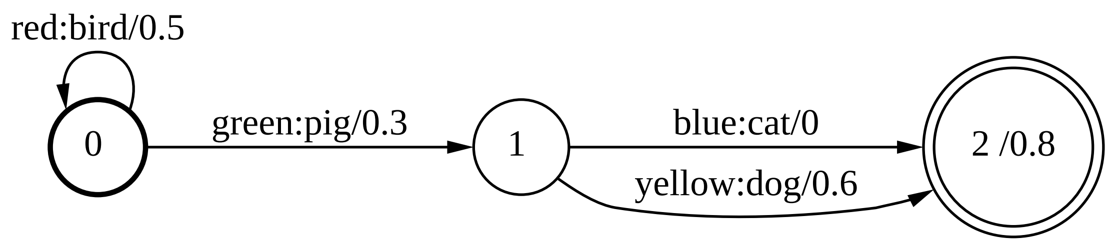
A.fst



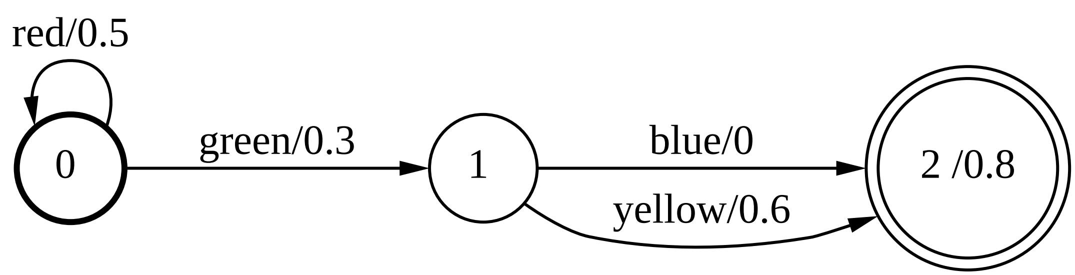
C.fst

Projection

- **Equation:** $A = \pi_1(T)$
- **Program:** `fsmproject -1 T.fst >A.fsa`
- **Graphical Representation:**



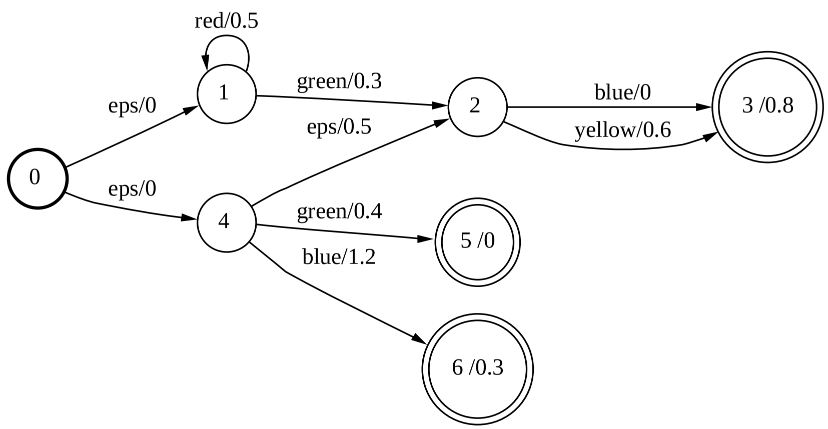
T.fst



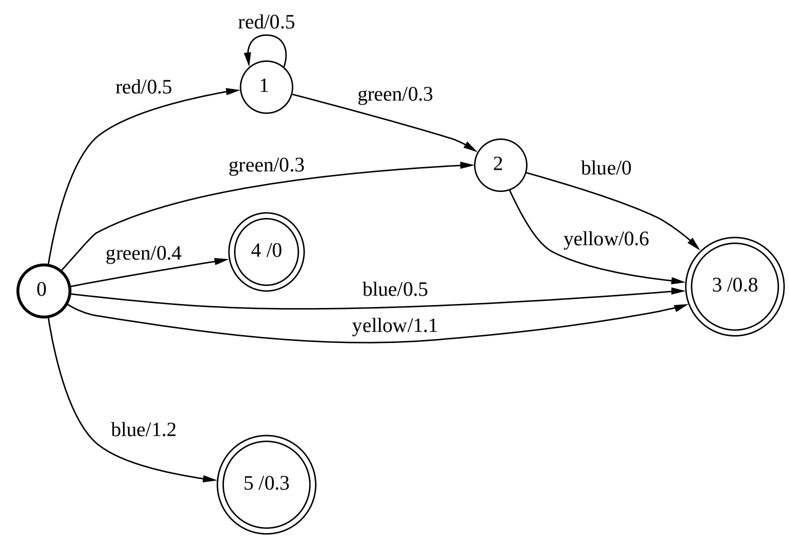
A.fsa

Epsilon Removal

- **Program:** `fsmrmepsilon A.fsa >C.fsa`
- **Graphical Representation:**



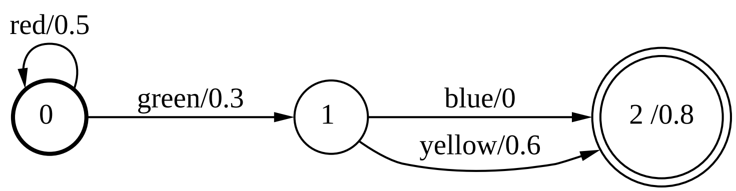
A.fsa



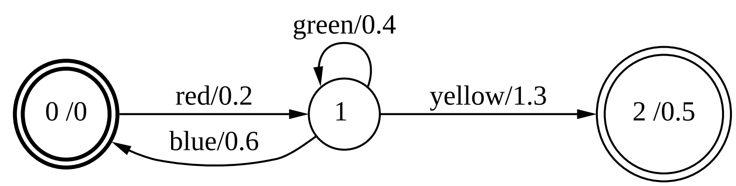
C.fsa

Intersection (Hadamard Product)

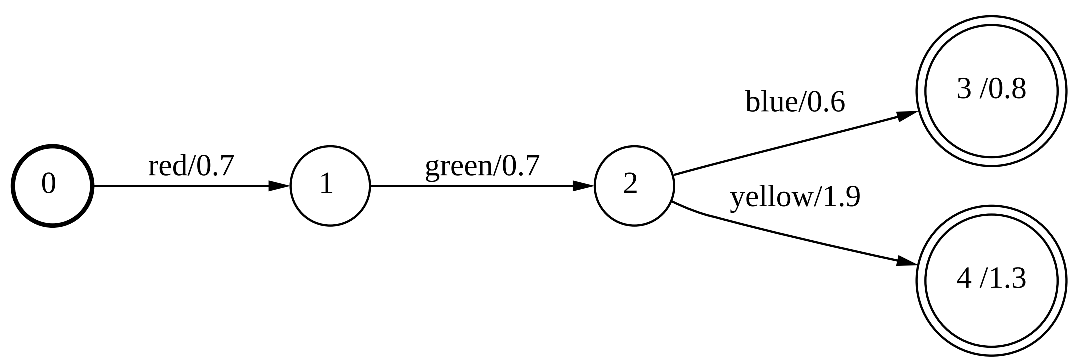
- **Equation:** $C = A \cap B$
- **Program:** `fsmintersect A.fsa B.fsa >C.fsa`
- **Graphical Representation:**



A.fsa



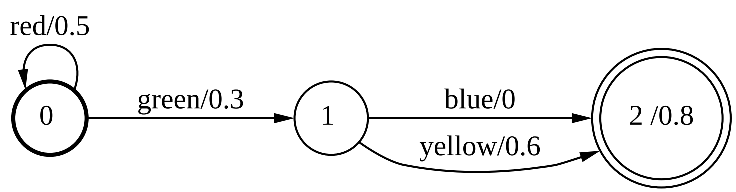
B.fsa



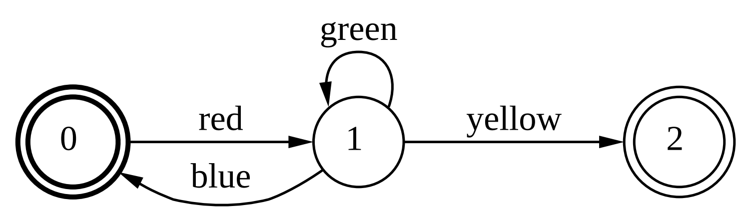
C.fsa

Difference

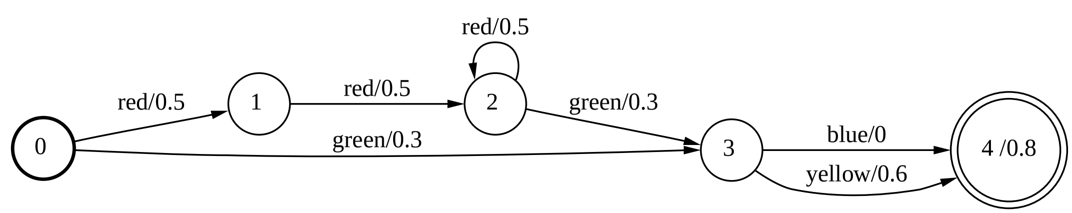
- **Equation:** $C = A - B$
- **Program:** `fsmdifference A.fsa B.fsa >C.fsa`
- **Graphical Representation:**



A.fsa



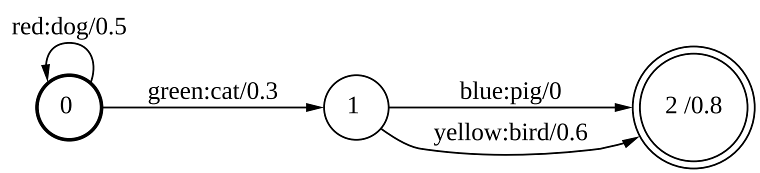
B.fsa



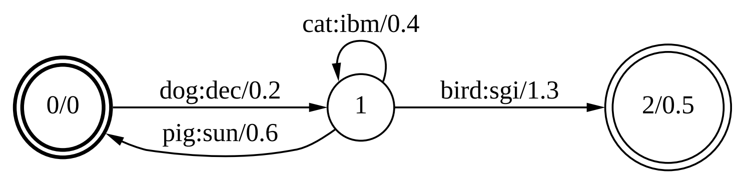
C.fsa

Composition

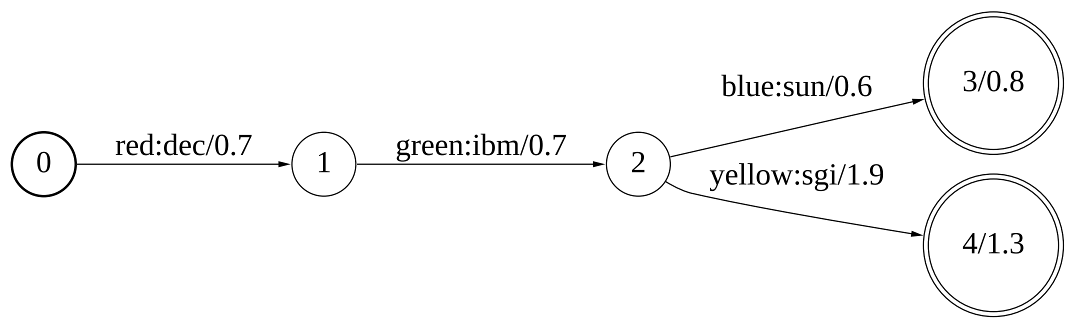
- **Equation:** $C = A \circ B$
- **Program:** `fsmcompose A.fst B.fst >C.fst`
- **Graphical Representation:**



A.fst



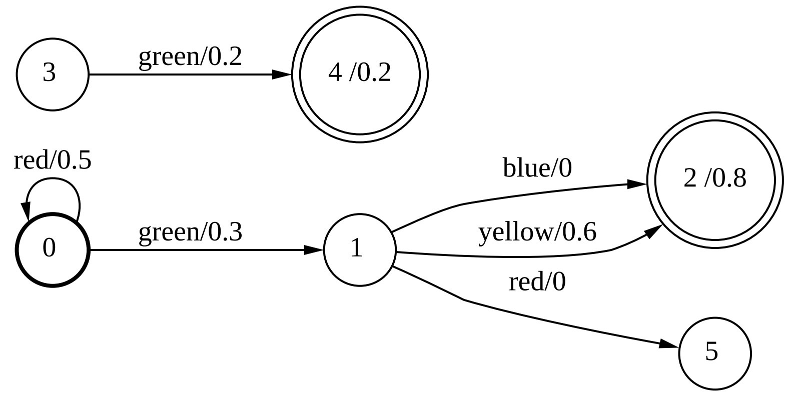
B.fst



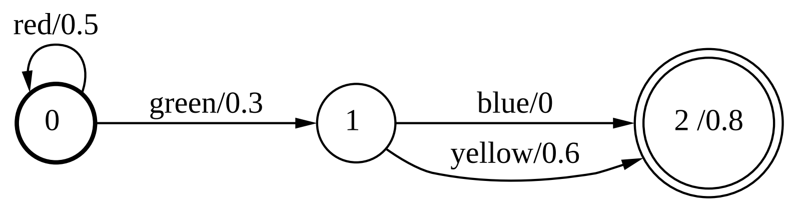
C.fst

Connection (Trimming)

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



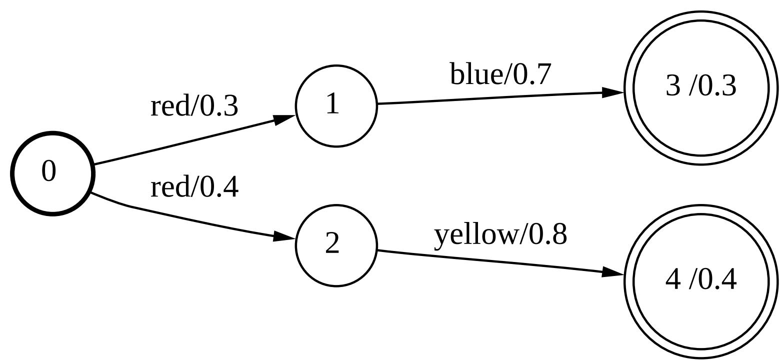
A.fsa



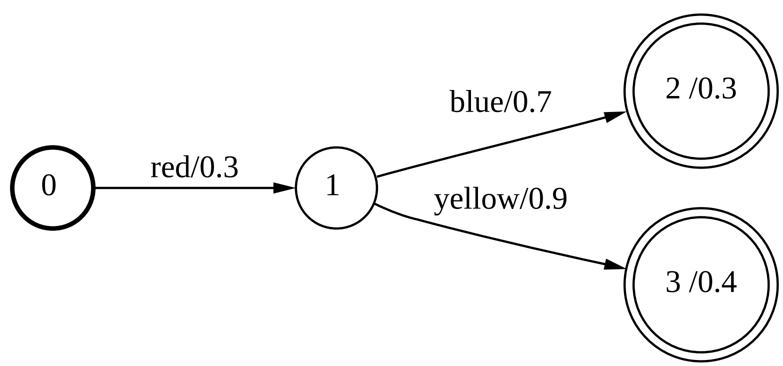
C.fsa

Determinization

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



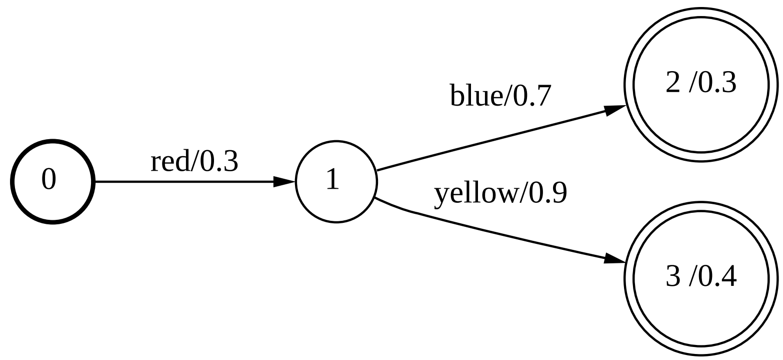
A.fsa



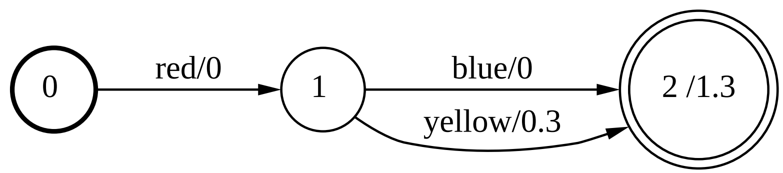
D.fsa

Minimization

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



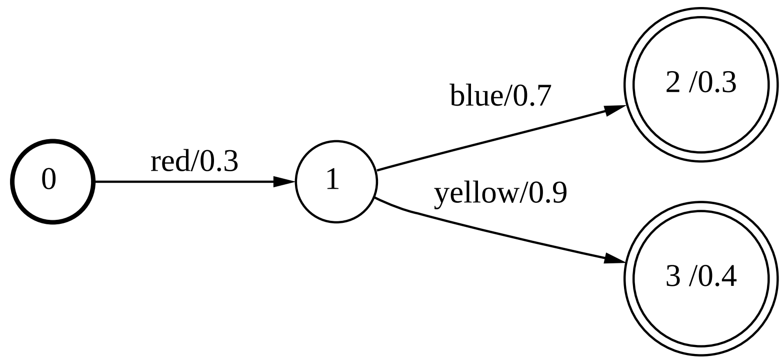
D.fsa



M.fsa

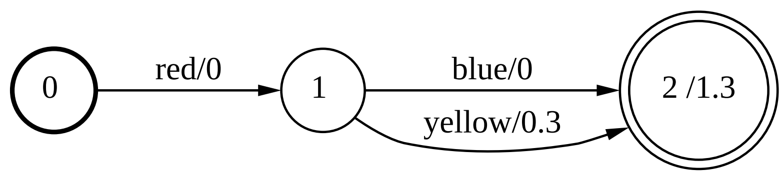
Equivalence

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



D.fsa

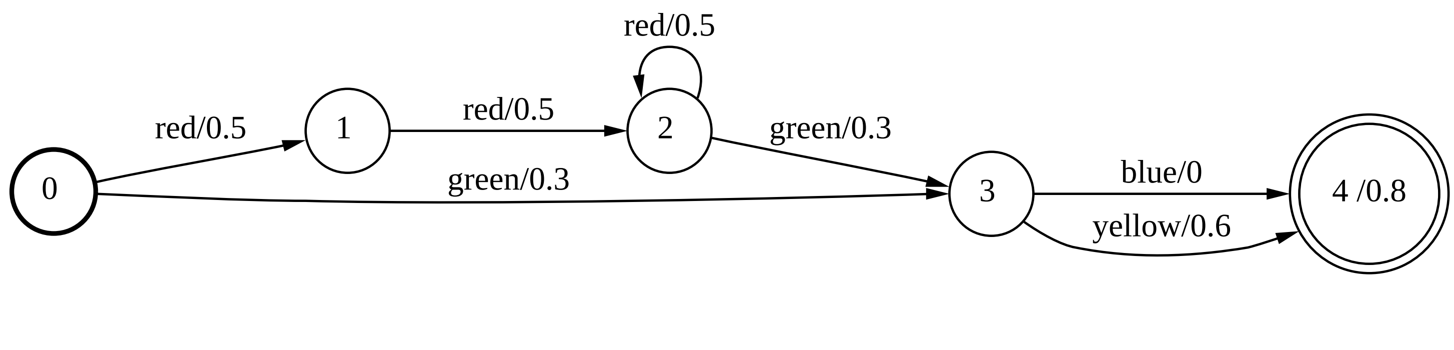
$=?$



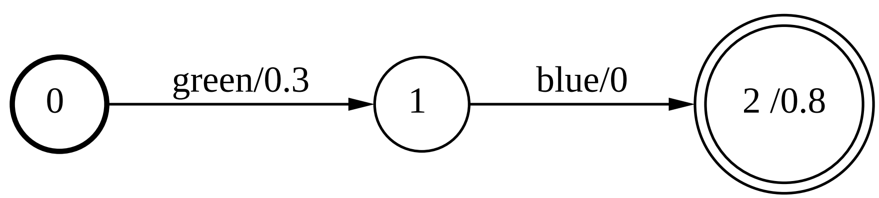
M.fsa

Best Path(s)

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



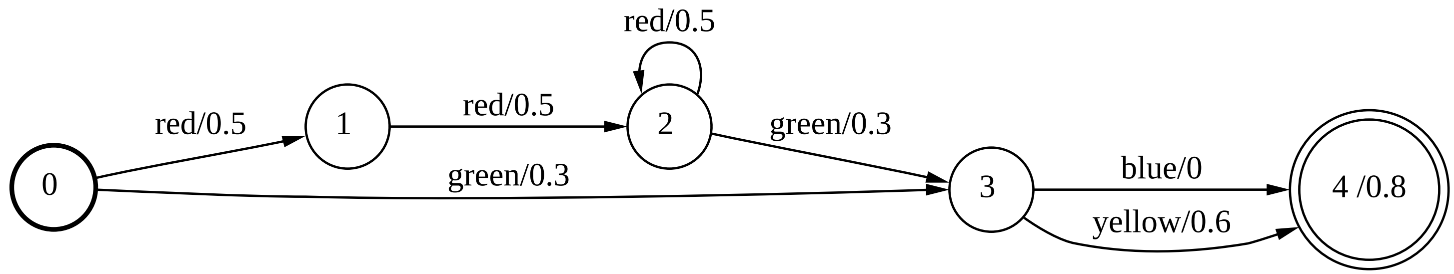
A.fsa



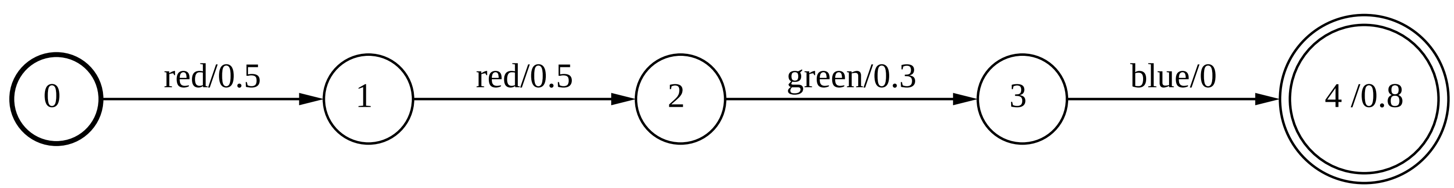
C.fsa

Random Path(s)

- **Program:** `fsmrandgen [-n N] A.fsa >C.fsa`
- **Graphical Representation:**



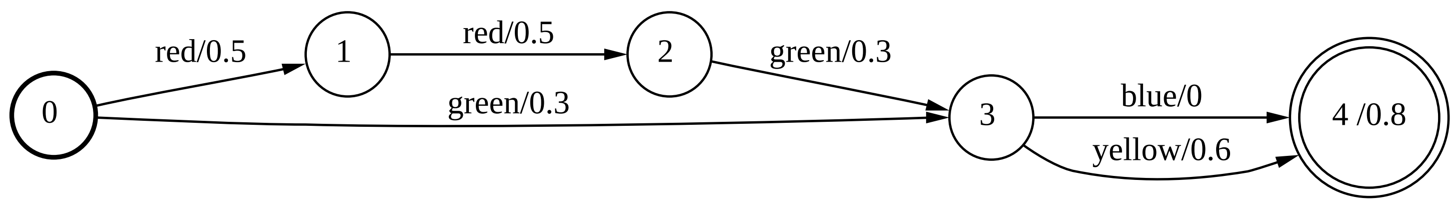
A.fsa



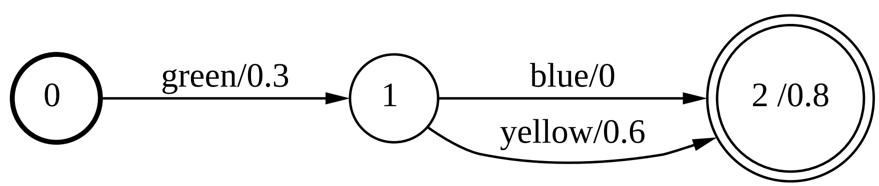
C.fsa

Pruning

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



A.fsa



C.fsa

FSM Format Conversion

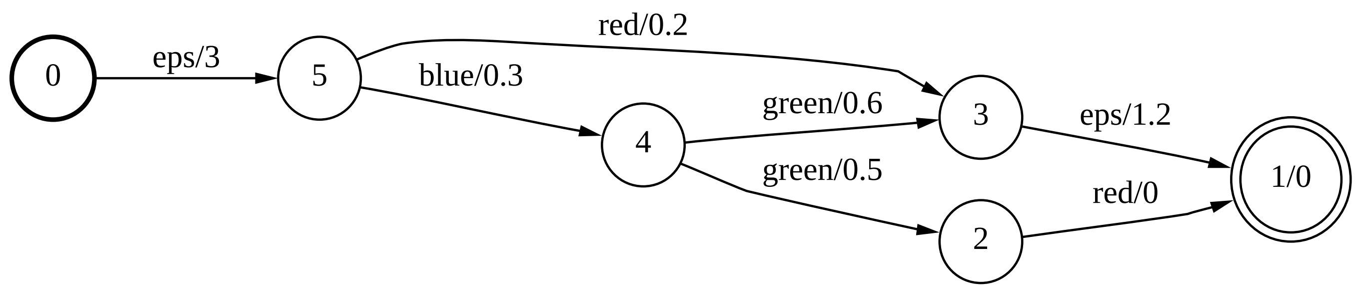
- **Program:**

  `fsmconvert -flag <A.fsa >C.fsa`

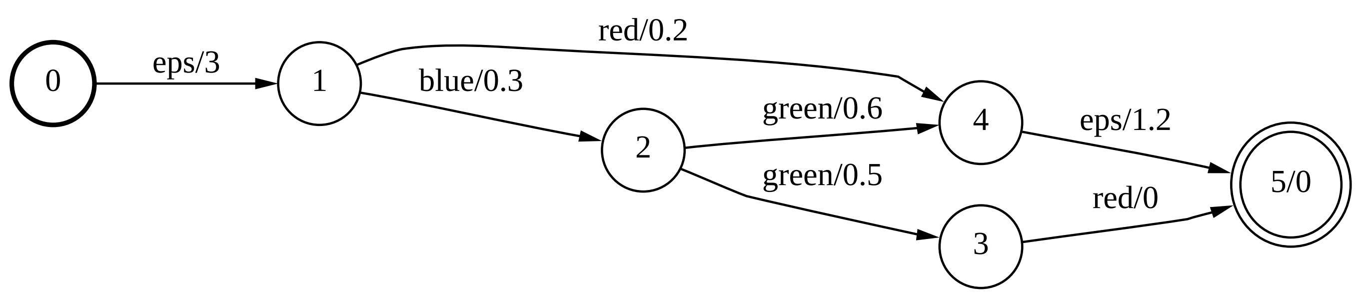
flag	format
-b	FSMBasicClass
-i	FSMIndexedClass
-I	FSMInputIndexedClass
-O	FSMOutputIndexedClass
-c	FSMConstClass
-ci	FSMConstIndexedClass
-cI	FSMConstInputIndexedClass
-cO	FSMConstOutputIndexedClass
-p	FSMPackedClass

Topological Sort

- **Program:** `fsmtopsort A.fsa >C.fsa`
- **Graphical Representation:**



A.fsa



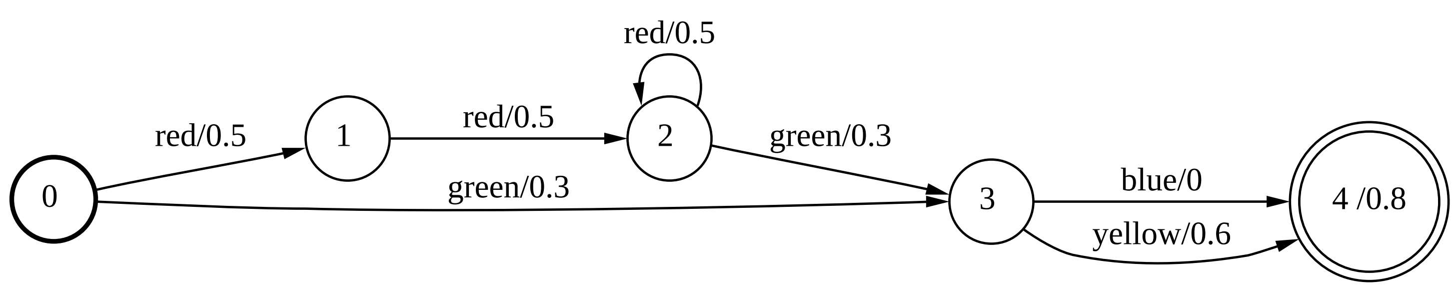
C.fsa

FSM Information

- **Program:**

```
fsminfo A.fsa
class           basic
transducer      n
# of states     5
```

- **Graphical Representation:**



A.fsa

FSM Information (numeric)

- **Program:**

```
fsminfo -n A.fsa
# of states                  5
# of arcs                    7
initial state                0
# of final states            1
# of eps                     0
# of accessible states       5
# of coaccessible states     5
# of connected states        5
# strongly conn components   5
```

FSM Information (distributions)

- **Program:**

```
fsminfo -q.25 A.fsa
quantiles            0     0.25  0.5  0.75  1
   in-deg            0     1     2    2     2
   out-deg           0     1     2    2     2
   label             1     1     2    3     4
   multiplicity      1     1     1    1     1
   arc cost          0     0.3   0.5  0.5   0.6
   final cost        0.8   0.8   0.8  0.8   0.8
```

FSM Information (properties)

- **Program:**

```
fsminfo -t A.fsa
arcs label sorted        y
arcs cost sorted         n
connected                y
acyclic                  n
acyclic wrt start state  y
acyclic wrt eps          y
top sorted               n
top sorted wrt eps       y
no eps                   y
costless                 n
costs non-negative       y
deterministic            y
```

Part 2. – Applications and Examples

1. Keyword detection/recognition
2. String matching
3. String alignment (matching with errors)
4. Language normalization
5. Context-dependency in ASR
6. Sentence generation (with pronunciations)

Keyword Detection

- **C identifiers:**

  $\{char, const, continue, if, int, else, short, signed, sizeof\}$

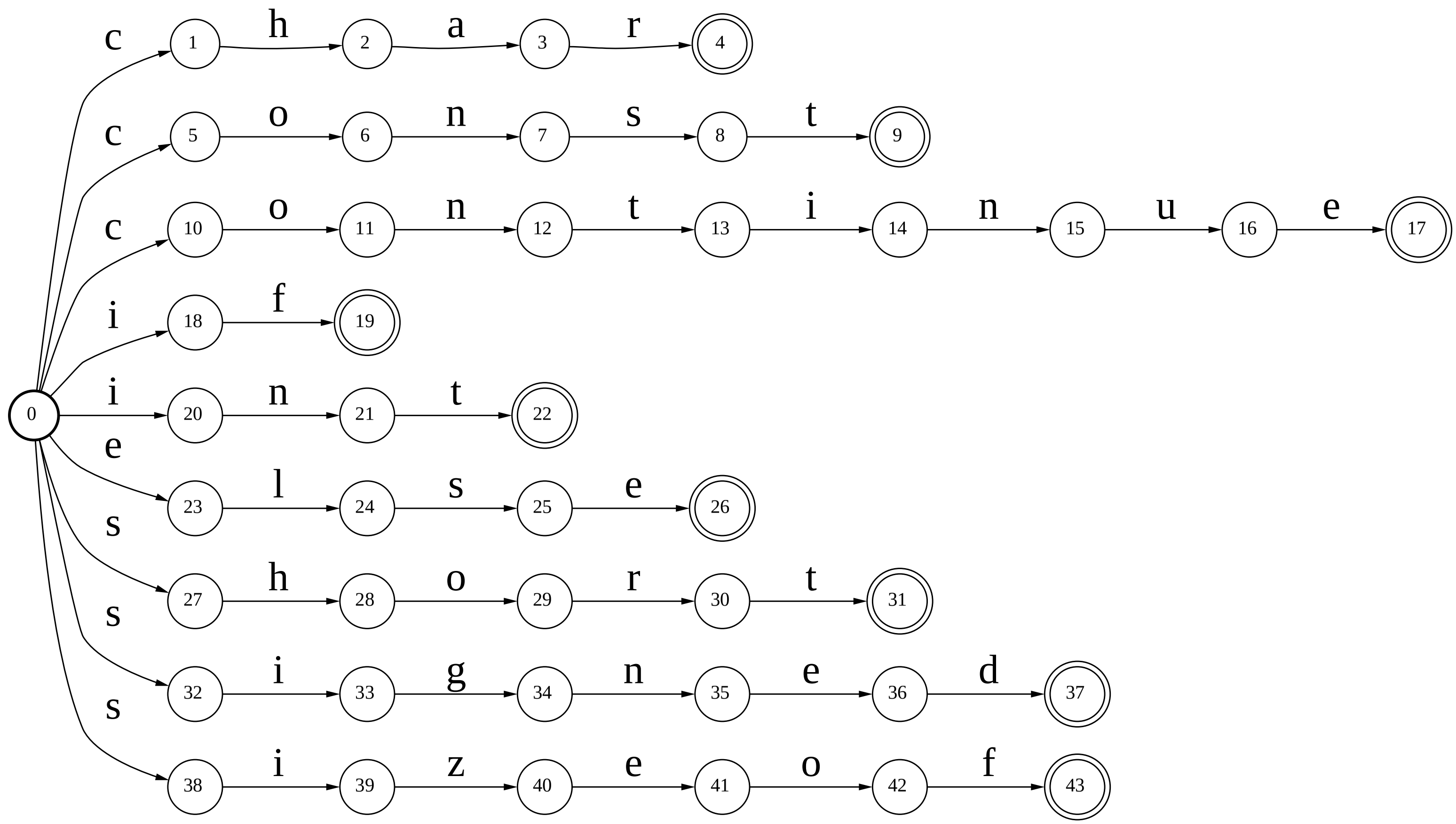
- **Brute-force search:**

```
if(strcmp(token, "char") == 0) return 1;
if(strcmp(token, "const") == 0) return 1;
if(strcmp(token, "oontinue") == 0) return 1;
if(strcmp(token, "if") == 0) return 1;
if(strcmp(token, "int") == 0) return 1;
if(strcmp(token, "else") == 0) return 1;
if(strcmp(token, "short") == 0) return 1;
if(strcmp(token, "signed") == 0) return 1;
if(strcmp(token, "sizeof") == 0) return 1;
else return 0;
```

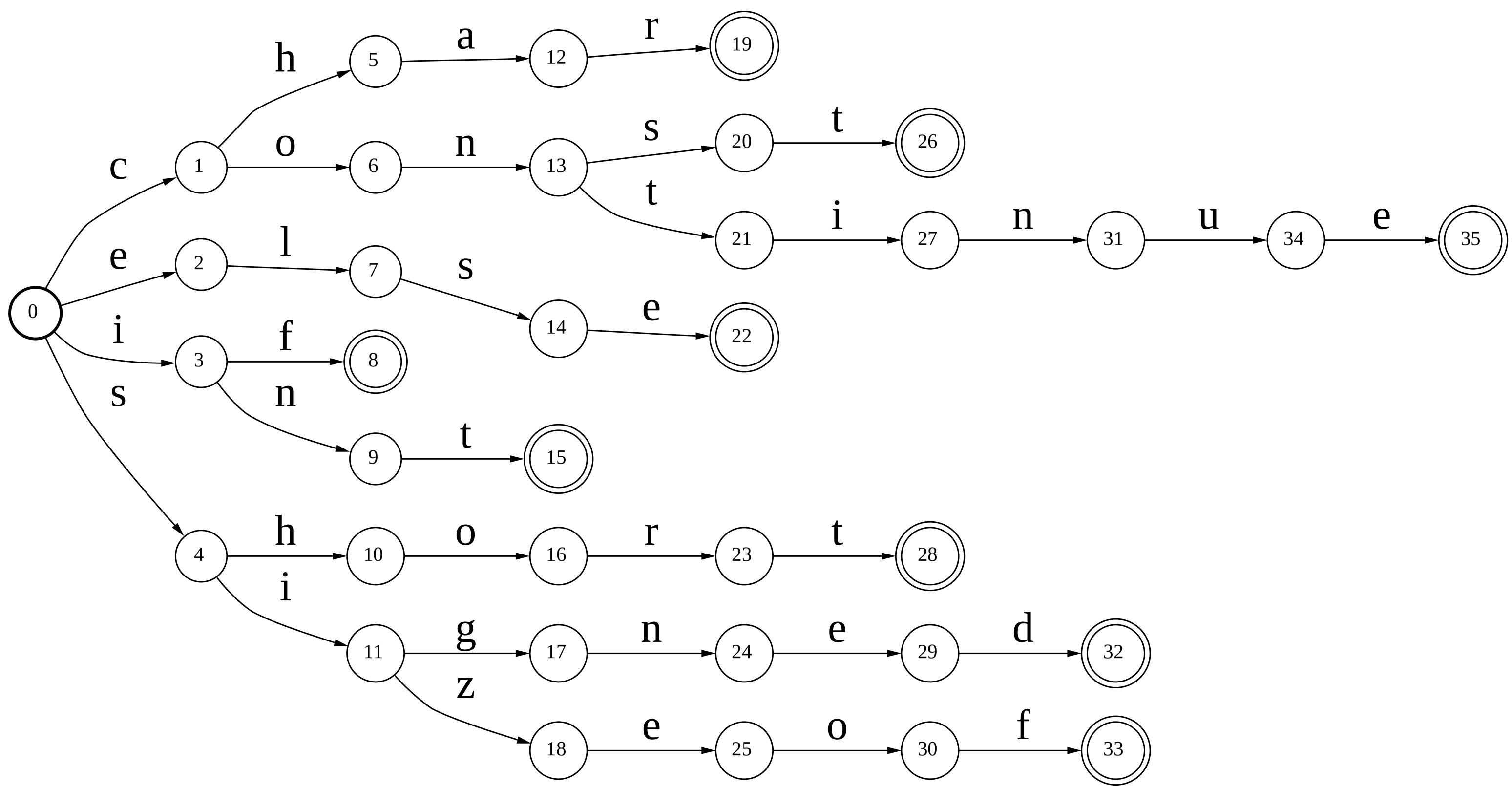
Keyword Detection – Tabular Search

```
#define NKEYS 9
char *keywrds[NKEYS] = {  "char",
                          "const",
                          "continue",
                          "else",
                          "if",
                          "int",
                          "short",
                          "signed",
                          "sizeof" };

int keycmp(const void *x, const void *y)
{ return strcmp(x, *(char **)y);}

bsearch(token, keywrds, NKEYS, sizeof(char *), keycmp);
```

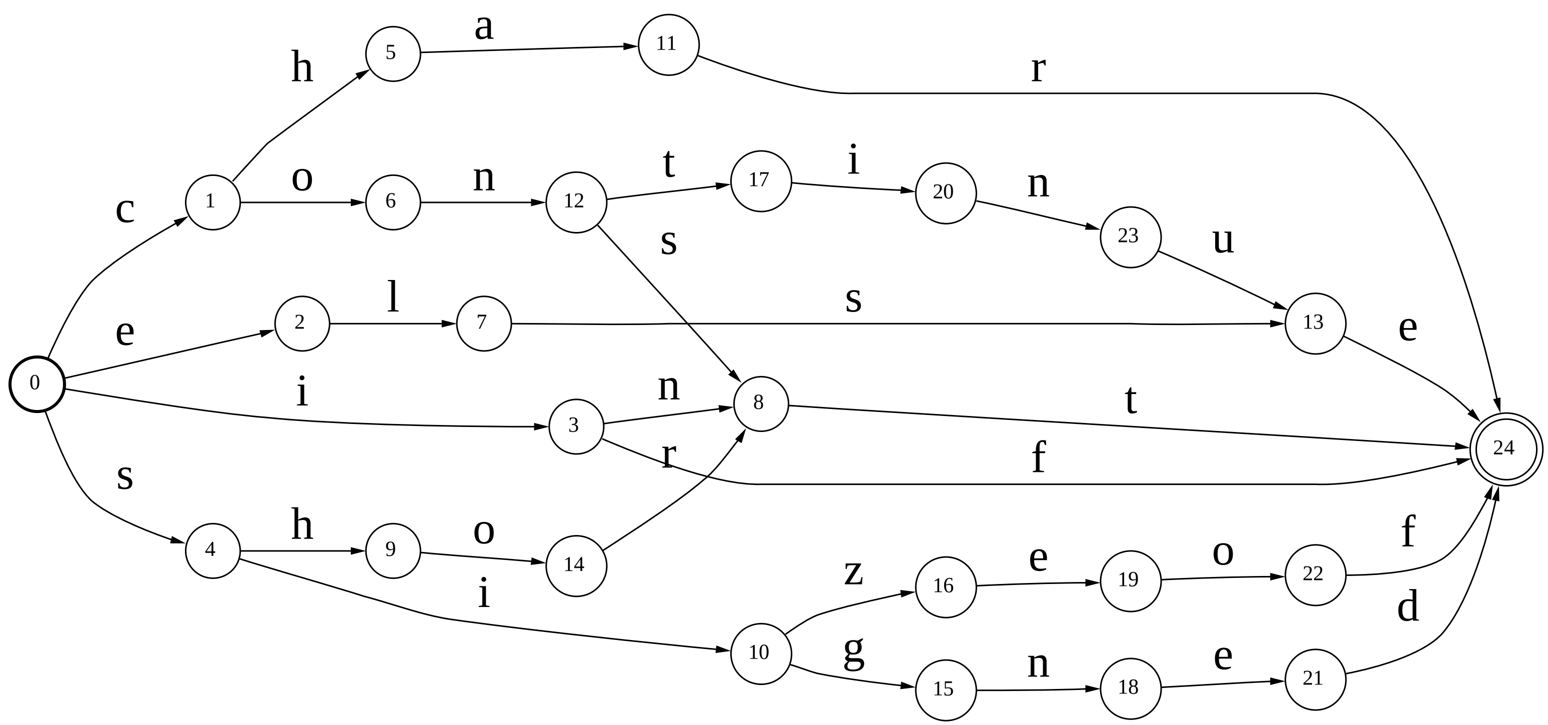
Keyword Detection – Automata Search



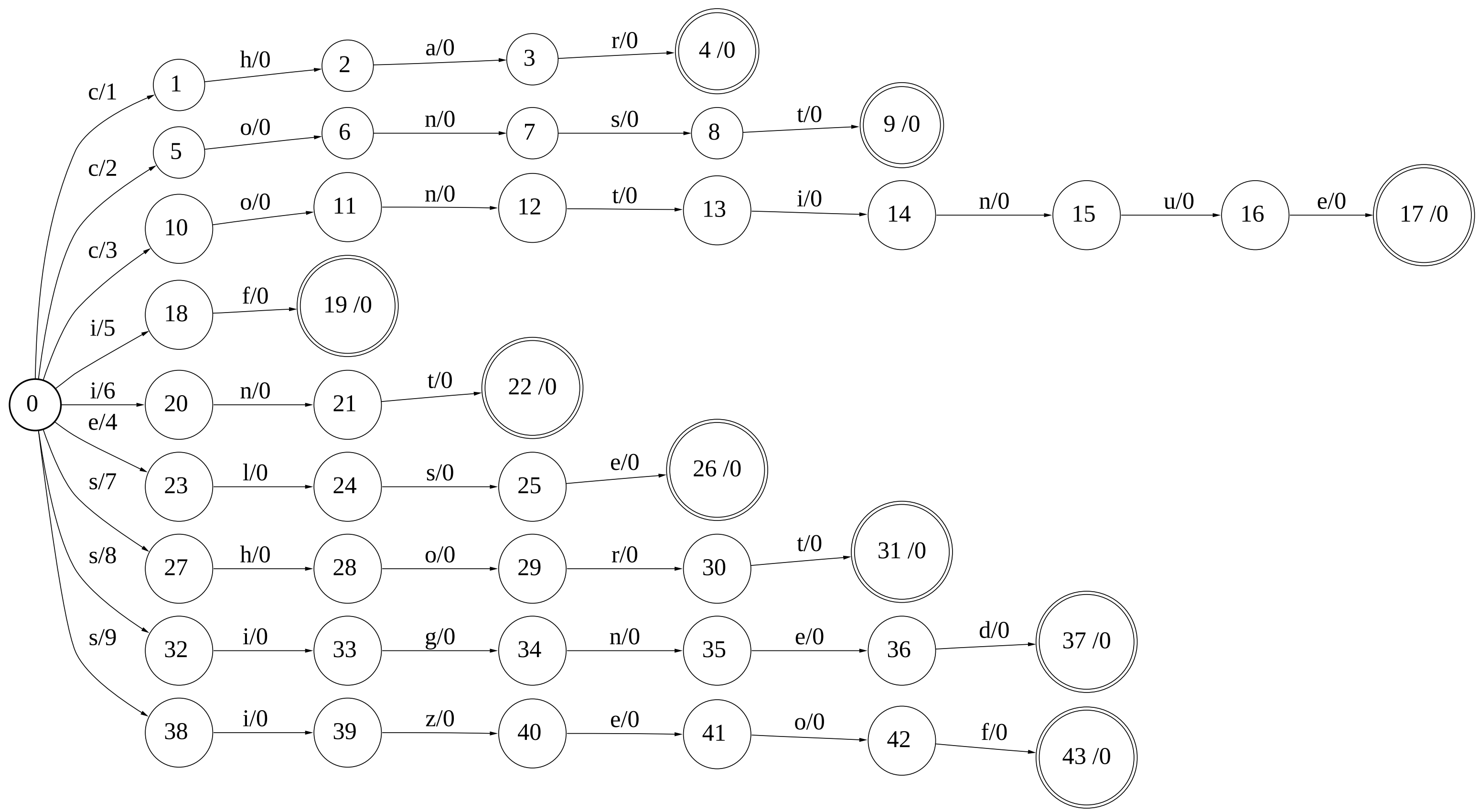
Keyword Detection – Deterministic Search



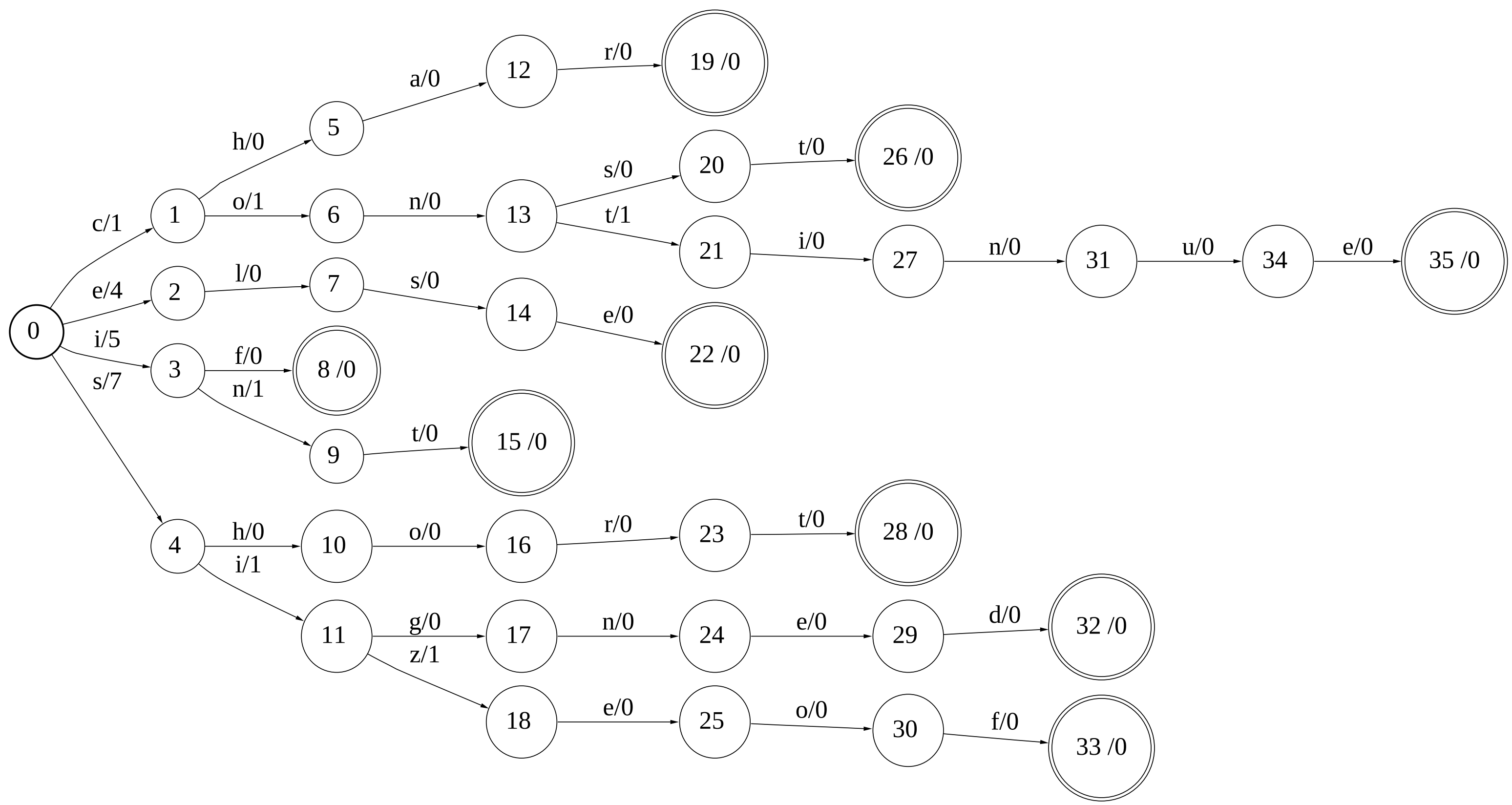
Keyword Detection – Minimal Deterministic Search



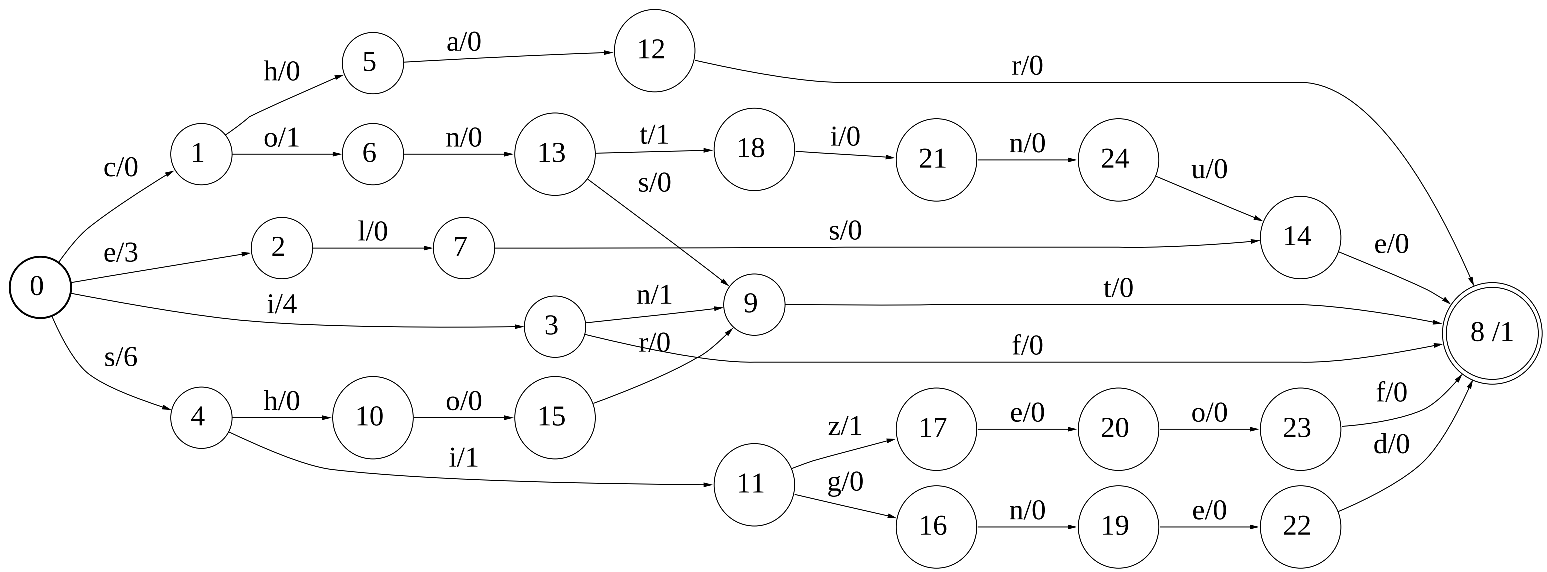
Keyword Recognition – Automata Search



Keyword Recognition – Deterministic Search

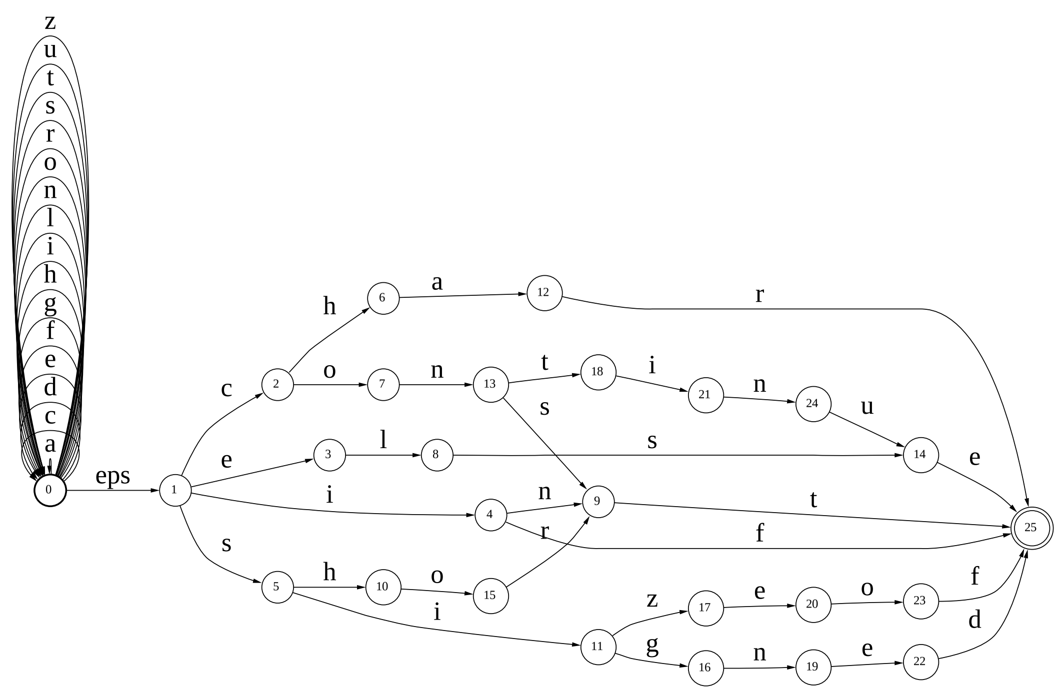


Keyword Recognition – Minimal Deterministic Search



String matching

- **Equation:** $M = \Sigma^* D$
- **Graphical Representation:**



String Alignment

BASEFORM	PHONE	WORD
p	pr	purpose
er	er	
p	pcl	
-	pr	
ax	ix	
s	s	
ae	eh	and
n	n	
d	-	
r	r	respect
ih	ix	
s	s	
p	pcl	
-	pr	
eh	eh	
k	kcl	
t	tr	

Weighted Edit Distance

- Symbol edit weights (task-specific):

BASEFORM a_i	**PHONE** b_j	**WEIGHTS** $w(a_i, b_j)$	**TYPE**
ae	eh	1	substituion
d	ϵ	3	deletion
ϵ	pr	1	insertion

- Minimum string edit distance (prefixes):

$$d(\mathbf{a}^0, \mathbf{b}^0) = 0$$

$$\begin{aligned} d_s(\mathbf{a}^i, \mathbf{b}^j) &= d(\mathbf{a}^{i-1}, \mathbf{b}^{j-1}) && + \quad w(a_i, b_j) && \text{(substitution)} \\ d_d(\mathbf{a}^i, \mathbf{b}^j) &= d(\mathbf{a}^{i-1}, \mathbf{b}^{j}) && + \quad w(a_i, \epsilon) && \text{(deletion)} \\ d_i(\mathbf{a}^i, \mathbf{b}^j) &= d(\mathbf{a}^{i}, \mathbf{b}^{j-1}) && + \quad w(\epsilon, b_j) && \text{(insertion)} \end{aligned}$$

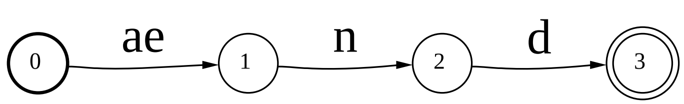
$$d(\mathbf{a}^i, \mathbf{b}^j) = min \left\{ d_s(\mathbf{a}^i, \mathbf{b}^j) + d_d(\mathbf{a}^i, \mathbf{b}^j) + d_i(\mathbf{a}^i, \mathbf{b}^j) \right\}$$

String Alignment by Automata

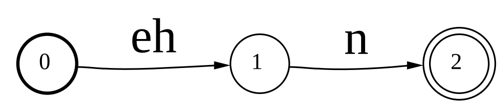
- **Equation:** $C = \text{argmin}\,(A \circ T \circ B)$
- **Program:**

```
fsmcompose A.fsa T.fst B.fsa | fsmbestpath >C.fst
```

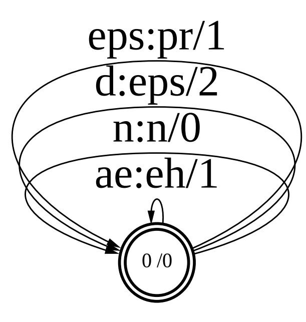
- **Graphical Representation:**



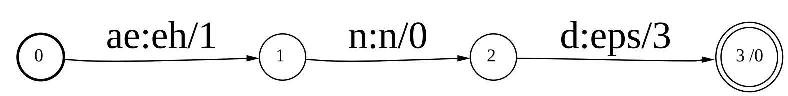
A.fsa



B.fsa



T.fst



C.fst

Generalized Weighted Edit Distance

BASEFORM(S) a_i	**PHONE(s)** b_j	**WEIGHTS** $w(a_i, b_j)$	**TYPE**
p	pcl pr	1	expansion
eh m	em	3	contraction
r eh	ax r	2	transposition
t/V'__V	dx	0	context-dependency

Language Normalization

- **M. Mohri Transducer**:

M. Mohri Phrase	**English Phrase**
indications to	directions to
computer science	theoretical computer science
neither . . . neither	neither . . . nor

- **Phrase Fragment Transducer** (T):

- **Identity Transducer** (Σ):

Language Normalization Transducer (N)

With T = Fragment Transducer and Σ = Identity Transducer:

- **Method 1:**

$$N = \Sigma^* T \Sigma^*$$

- **Method 2:**

$$N = \Sigma^* (T \Sigma^*)^*$$

- **Method 3:**

  Let

$$A = \pi_1(T)$$

  and

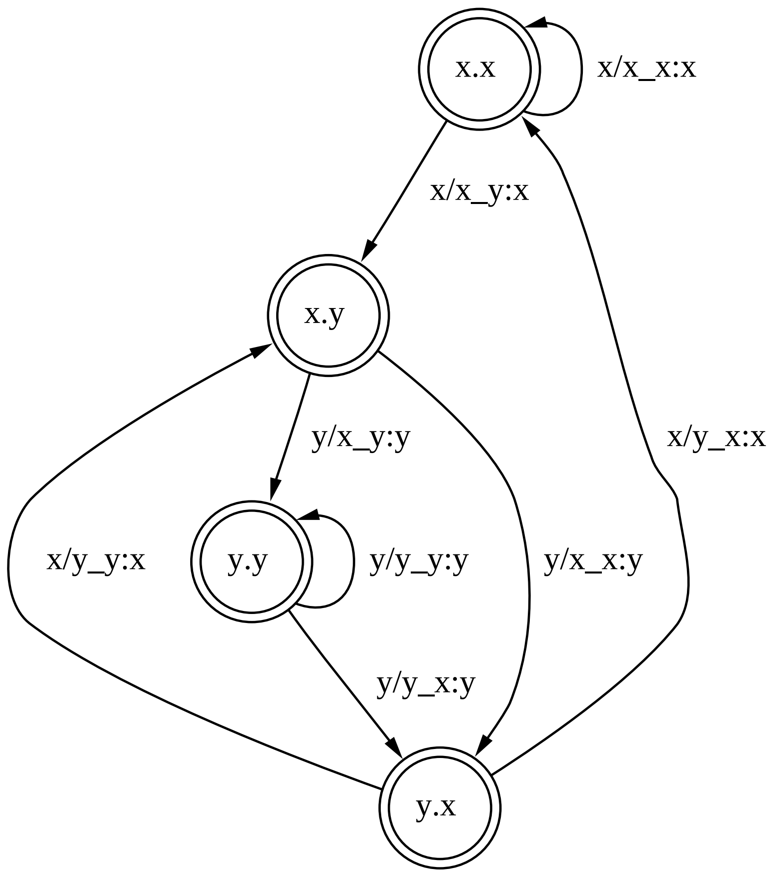
$$\Gamma = \Sigma^* - \Sigma^* A \Sigma^*,$$

  then:

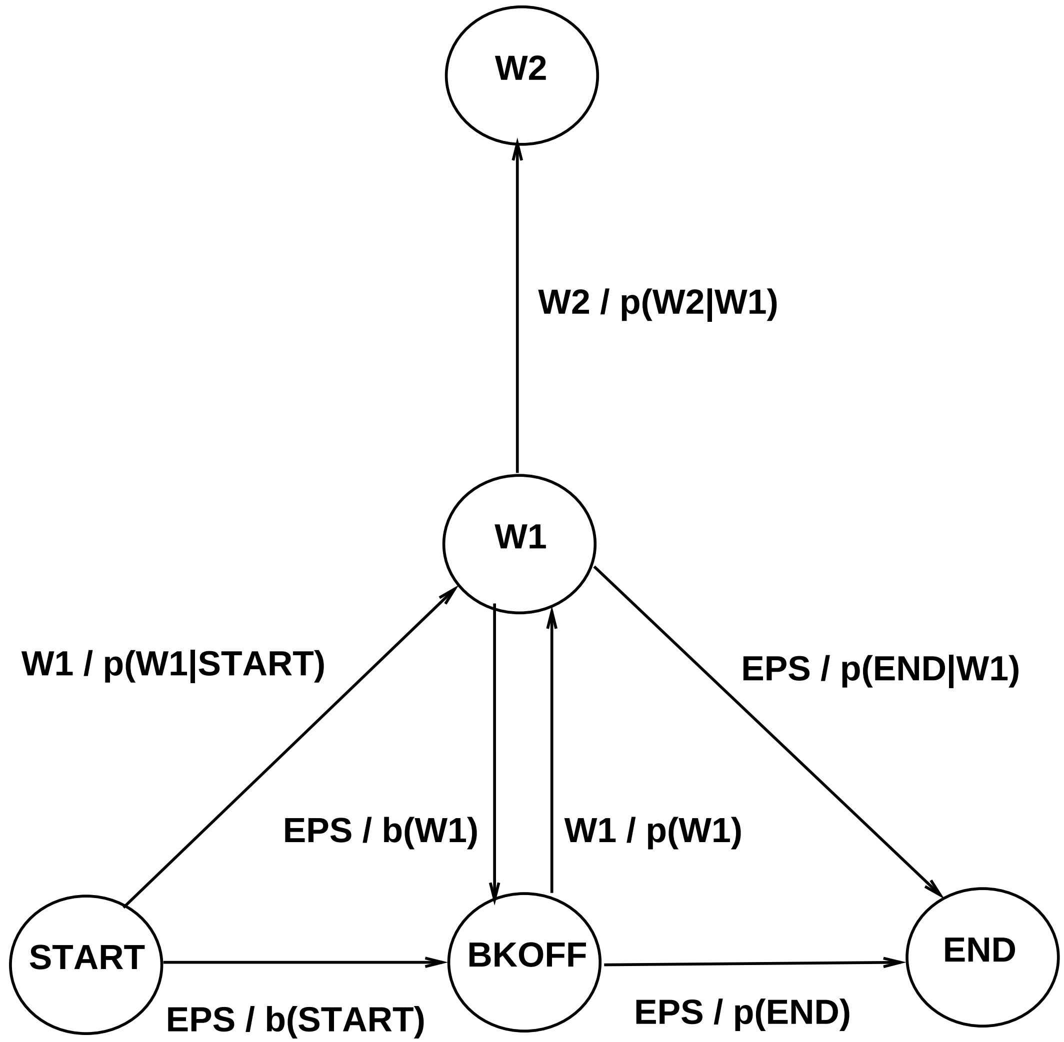
$$N = \Gamma(T\Gamma)^*$$

Context-Dependency in ASR Modeling

- **Context-dependent phone models:** Maps from CI units to CD units. Example: $ae/b\_\_d \rightarrow ae_{b,d}$
- Triphonic context transducer for two symbols x and y:



Bigram Model



N-Gram Language Model Examples

- **N-Gram Statistics:**

n-gram	# of words	# of states	# of arcs	# of eps
2	20000	19981	5066971	19979
3	20000	5046501	16795922	5046499

- **Trigram Output:**

```
fsmrandgen trigrm.fsa | fsmprint | format
```

SOCIAL PHILOSOPHY PROFESSOR AT HARVARD UNIVERSITY CHARLES BRADY OF

OPPENHEIMER AND COMPANY BELIEVES ITS MAIN POWER BROKER SHIN

ANALYSTS ARE SO MANY THE FINAL TALLY OF NEW ACCOUNTING FIRMS

SUSPENDED HIM WITHOUT PORTFOLIO SAID IT HAS NO CABINET DEPARTMENTS

- **Trigram Phonetic Output:**

```
fsmrandgen trigrm.fsa | fsmcompose lexicon.fst - |
fsmproject -1 | fsmprint | format
```

s ow sh ax l ph ih l f ih l aa s ax f iy p r ax f eh s er at t hh aa r v er d y uw n ax v er . . .

Rules of Thumb

1. Think in terms of operations on sets and relations
2. Do not *interpret* equivalent FSMs differently:
   (a) encode information on transitions not states
   (b) do not depend on the presence or absence of epsilon transitions
   (c) do not depend on the registration between input and output labels along a path
   (d) do not depend on the distribution of costs along a path
3. Reduce non-determinism for speed
   (a) ordinary non-determinism
   (b) epsilon transitions
4. Use minimization for space