Finite State Calculus

Computational Linguistics I

Spring 2023

Read Chapter 3 of Beesley & Karttunen

Four implementations of Fst language.

xfst foma

hfst sfst

We will use hfst, and possibly foma. The finite state calculus is used to describe *sound systems* of natural languaes.

Languages with terms denoting pluralities or sets

[Ashley and Brittany]		a+b {a,b}
[the girls]		{a,b,c}
who		{x | x is a person}
[b | c]	{“b”,“c”}	
[a b+]	{“ab”, “abb”, “abbb”, “abbb”, ...}	
[a [b | c]+]	{“ab”, “ac”, “abb”, “abc”, “acb”, “acc”, “abbb”, “abbc”, ... }	

What does a singular denote?

Ashley	{ a }
Brittany	{ b }
a	{ “a” }
b	{ “b” }

Operations on sets of strings

Intersection expressed with “&”

[a b+] & [a+ b] { “ab” }

Union expressed with “|”

[a b+] | [a+ b] {“ab”,
“abb”, “aab”,
“abbb”, “aaab”, ... }

Denotation notation

[$[_{DP}$Ashley] **]** = { a }

[$[_{DP}$Brittany] **]** = { b }

[$[_{DP}$ $[_{DP}$ Ashley] and $[_{DP}$ Brittany]] **]** = { a,b }

If *x* is a UTF8-character, then *x* is an XFST term and **[** *x* **]** is the singleton set { f }, where f is the string of length one that has *x* in position 1.

Union terms in FST

If x and y are FST terms, then $x|y$ is an FST term and

$$[x \mid y] = [x] \cup [y].$$

Intersection terms in FST

If x and y are XFST terms, then $x\&y$ is an XFST term and

$[x \mid y] = [x] \cap [y]$.

[a+ b] & [a b+]

[a+ b] | [a b+]

a & b

A	a+b+
B	ab
C	empty set

Concatenation terms

If x and y are FST terms, then x y is an FST term and

$[x\ y] =$

$\{\ c \mid$ there is an element a of $[x]$ and an element b of $[y]$ such that $c = a$^$b\ \}$

Concatenation of strings: “a”^“bb”= “abb”

Concatenation terms

If x and y are FST terms, then $x\ y$ is an FST term and

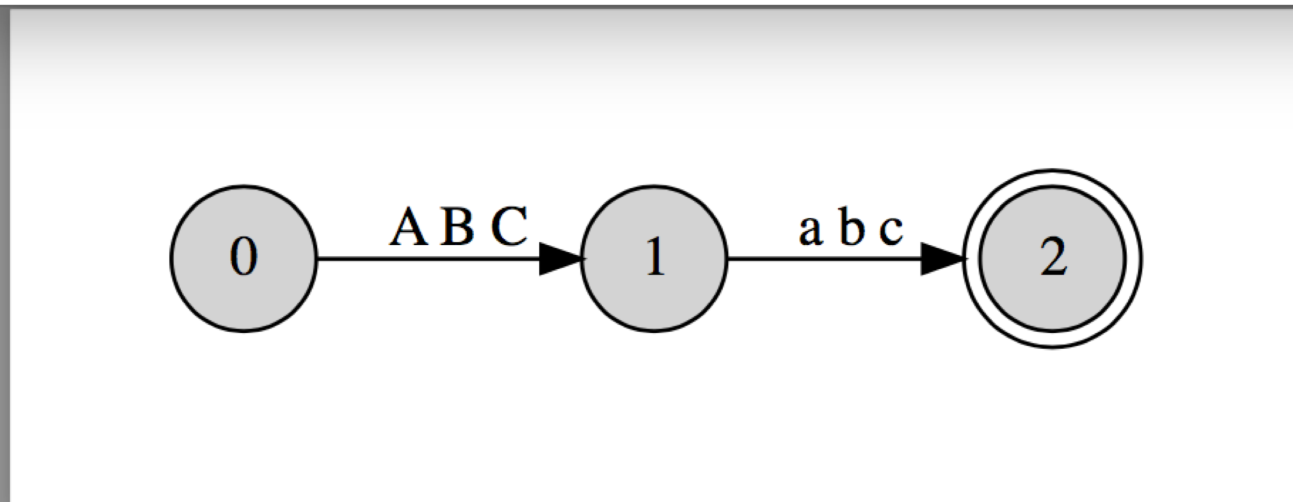
$[x\ y] =$

$\{\ c \mid$ there is an element a of $[x]$ and an element b of $[y]$ such that $c = a$^$b\ \}$

Concatenation of strings: “a”^“bb”= “abb”

The one-from each rule tends to multiply the size of the sets.

```
foma[0]: define X1 [A | B | C];
defined X1: 329 bytes. 2 states, 3 arcs, 3 paths.
foma[0]: define X2 [a | b | c];
defined X2: 329 bytes. 2 states, 3 arcs, 3 paths.
foma[0]: regex X1 X2;
455 bytes. 3 states, 6 arcs, 9 paths.
foma[1]: view net
foma[1]: 
```



Definitions

define C p | t | k ;

define V [i | u | e | o] ;

define Syl [V | [C V] | [V C] | [C V C]] ;

regex Syl;

print words;

A sequence of definitions like this is a straightline program, where things are defined in terms of things defined earlier.

syl.fst

```
define C p | t | k ;
define V [ i | u | e | o ] ;
define Syl [ V | [C V] | [V C] | [C V C] ] ;
define Dot ".";
define Word [Syl Dot]* Syl;
set print-space ON
```

```
U:---  syl.fst        All L7     (Fundamental +3)
Quit
```

4424-18 — foma — 62×15

```
[1] t i
[1] t e k . e t . e
[1] o . e
[1] i k
[1] u
[1] o t
foma[2]: source syl.fst
Opening file 'syl.fst'.
redefined C: 329 bytes. 2 states, 3 arcs, 3 paths.
redefined V: 371 bytes. 2 states, 4 arcs, 4 paths.
redefined Syl: 609 bytes. 4 states, 14 arcs, 64 paths.
redefined Dot: 202 bytes. 2 states, 1 arc, 1 path.
redefined Word: 651 bytes. 4 states, 16 arcs, Cyclic.
variable print-space = ON
foma[2]:
```

```
foma[4]: regex Word & ?^12;
2.9 kB. 40 states, 158 arcs, 6948864 paths.
foma[5]: print random-words
 [1] t u k . e k . e p . o t
 [1] t o t . o k . p e t . o
 [1] t o k . p u p . i k . e
 [1] u p . p u k . u p . t e
 [1] e t . i . t i p . p e k
 [1] o t . i . p o p . o . o
 [1] u p . e . k e t . t u k
 [1] o k . e . i p . k u . e
 [1] o k . o . p u . p i . o
 [1] t i t . u p . e p . i k
 [1] k o t . i p . t i t . e
 [1] k u p . o . e k . e . i
 [1] o t . o p . k u k . e p
 [1] o k . t i k . i p . e t
 [1] i . e p . u . i t . p o
```

Kleene operations

A+	one or more strings drawn from A concatenated
A*	one or more strings drawn from A, plus the empty string

foma commands

regex [a b*];	terminate with semicolon
regex [a	
b*];	split across lines
reg [a b*];	abbreviate commands
print words	sometimes semicolon can be omitted

grouping in terms

```
xfst[7]: reg a b * c;
380 bytes. 3 states, 3 arcs, Cir
xfst[8]: print random-words
ac
abc
ac
abbc
ac
abc
ac
ac
abbbbc
abc
abc
abbbbc
abc
ac
ac
xfst[8]:
```

```
xfst[8]: reg [a b] * c;
380 bytes. 3 states, 3 arcs, Cir
xfst[9]: print random-words
c
abababababababc
c
abc
ababababababc
c
c
abc
abc
c
abc
abababc
abc
c
abc
xfst[9]:
```

```
xfst[2]: print net
Sigma: e i k o p t u
Size: 7, Label Map: Default
Net:
Flags: deterministic, pruned, minimized, epsilon_free, loop_free
Arity: 1
s0:   e -> fs1, i -> fs1, k -> s2, o -> fs1, p -> s2, t -> s2, u -> fs1.
fs1:  k -> fs3, p -> fs3, t -> fs3.
s2:   e -> fs1, i -> fs1, o -> fs1, u -> fs1.
fs3:  (no arcs)
```

Output to file

```
abc
xfst[9]: regex Syl;
536 bytes. 4 states, 14 arcs, 64 paths. Label Map: Default.
xfst[10]: print words > syl.out
Opening 'syl.out'
Closing 'syl.out'
xfst[10]:
```

Empty string and empty set

0	unit set of the empty string
a & b	empty set

define N a & b;

Verb Roots and Inflections

```
define Root [ {eat} | {file} | {swallow} ];
define Inflection   [ [%+ {ing} %+ VG] |
    [ %+ {+ed+} %+ VBD] |
    [ %+ s %+ VBZ] ];
```

Concatenate root and inflection

```
define VerbUpper Root Inflection;

read regex VerbUpper;
print words
file+ing+VG                  file+ed+VBD
file+s+VBZ
eat+ing+VG                   eat+ed+VBD
eat+s+VBZ
swallow+ing+VG               swallow+ed+VBD
swallow+s+VBZ
```

Deletion of e in file+ing

```
define eElision e -> 0 || _ %+ [e | i] ;

define Tag [ VG | VBD | VBZ ];
define symbolElision [ Tag | %+ ] -> 0 ;
```

Relation Composition

$\|R \text{ .o. } S\| =$

$\{\langle x,z\rangle \mid \text{for some } y,$

$\langle x,y\rangle \in \|R\| \text{ and}$

$\langle y,z\rangle \in \|S\| \}$

Composed verb lexicon

define verb

VerbUpper .o.   (set coerced to relation)

  eElision .o.   (delete e in file+ed+VBD)

  symbolElision ;

Result is relation between underlying and surface forms.

Verb Relation

```
read regex verb;
print words
swallow<+:0>s<+:0><VBZ:0>
swallow<+:0>ing<+:0><VG:0>
swallow<+:0>ed<+:0><VBD:0>
        (six more)
```

Lower words

read regex verb.l;

print words

swallows swallowing swallowed

filing files filed

eats eating eated

read regex verb;

apply up

apply up> eated

eat+ed+VBD

List irregular verbs

define irregularVerb [e a t %+ e d %+ VBD] .x. [a t e];

Cartesian product of two unit sets gives unit set of a pair.

Union is wrong

read regex verb | irregularVerb;

apply down

apply down> eat+ed+VBD

eated

ate

(Or do we want two outputs?)

define verb2 [[~irregularVerb.u] .o. verb] | irregularVerb ;

read regex verb2;

print lower-words;

swallows swallowing swallowed

filing filed files

eats eating ate

verb.fst

```
define Root [ {eat} | {file} | {swallow} ];
define Inflection   [ [%+ {ing} %+ VG] |
    [ %+ {ed} %+ VBD] |
    [ %+ s %+ VBZ] ];

define VerbUpper Root Inflection;
define Tag [ VG | VBD | VBZ ];

define eElision e -> 0 || _ %+ [e | i] ;

define symbolElision [ Tag | %+ ] -> 0 ;

define verb [VerbUpper .o. eElision .o. symbolElision ] ;

define irregularVerb [e a t %+ e d %+ VBD] .x. [a t e];

define verb2 [[~irregularVerb.u] .o. verb ] | irregularVerb ;
```

To use it in the interpreter…

[xfst 5] source verb

[xfst 6] read regex verb2;

[xfst 7] print lower-words

What is a string?

domain *range*

“mommy” : $\{1,2,3,4,5\} \to \text{UTF8}$

$1 \mapsto m$

$2 \mapsto o$

$3 \mapsto m$

$4 \mapsto m$

$5 \mapsto y$

domain *range*

“a” : {1} → UTF8

1 ↦ a

XFST term *a* denotes the singleton set

$$\left\{ \begin{bmatrix} f : \{1\} \to UTF8 \\ 1 \mapsto a \end{bmatrix} \right\}$$