Lecture 17: FST, Morphology with foma

Ling 1330/2330 Intro to Computational Linguistics
Na-Rae Han, 10/26/2023

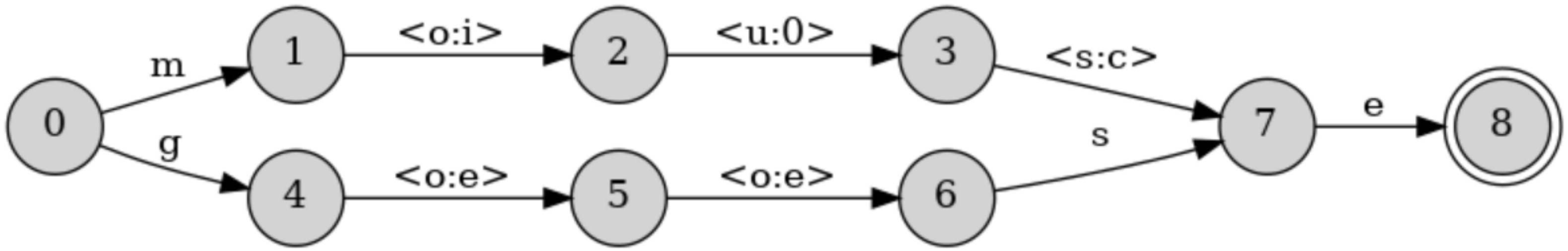
Outline

- Morphology and FST
  - Jurafsky & Martin (2nd Ed!) Ch.3 Words and Transducers
  - Hulden (2011) Morphological analysis with FST

    ← foma!

Introducing: foma

- https://fomafst.github.io/
- A compiler of finite-state machines (FSA and FST)
  - FSA: you already know
  - FST: Finite-State **Transducer**

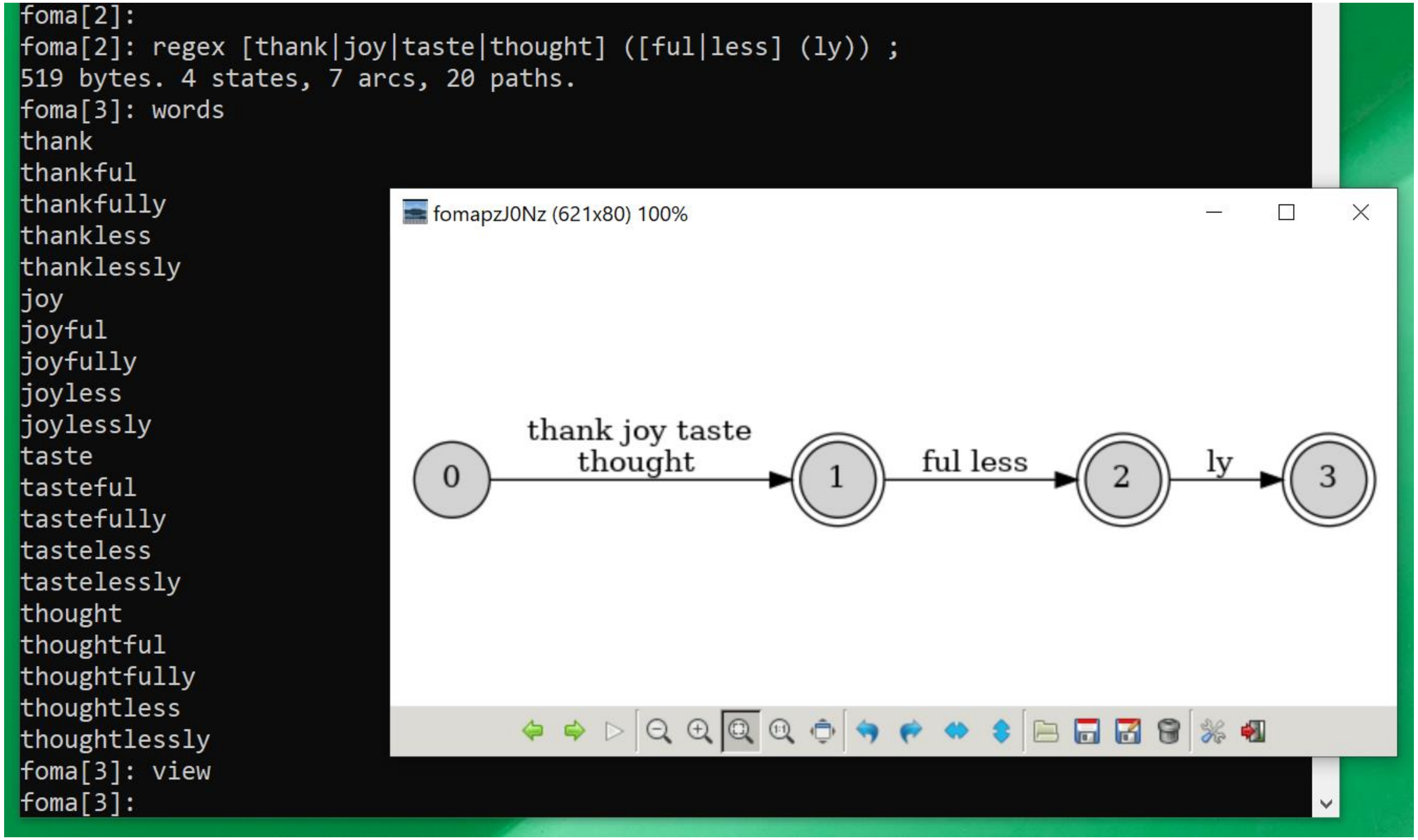


  - A modern incarnation of Xerox's classic FST suite: XFST and LEXC.

regex in foma: pitfalls

- Foma takes regular expression syntax from Xerox's FST tools, which incorporate many linguistic rule conventions
- foma's regex syntax differ from the standard (Perl, Python) syntax in some key aspects, most notably:
  - ? vs. ()
  - () vs. []
- Additionally, foma adopts multi-character symbols; SPACE is meaningful.
  - "abc" is a single symbol, "a b c" is three symbols concatenated
- Refer to:
  - https://github.com/mhulden/foma/blob/master/foma/docs/simpleintro.md#regex-basics

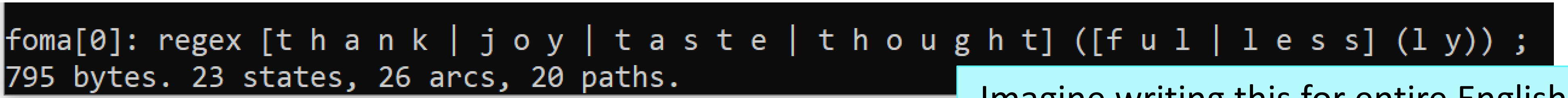
English morpho-syntax as FSA



```
foma[2]:
foma[2]: regex [thank|joy|taste|thought] ([ful|less] (ly)) ;
519 bytes. 4 states, 7 arcs, 20 paths.
foma[3]: words
thank
thankful
thankfully
thankless
thanklessly
joy
joyful
joyfully
joyless
joylessly
taste
tasteful
tastefully
tasteless
tastelessly
thought
thoughtful
thoughtfully
thoughtless
thoughtlessly
foma[3]: view
foma[3]:
```

- Here, "thank", "ful", etc. are construed as distinct **multi-character symbol units**.
- When building a morphological parsers, we don't normally treat morphemes as such. (WHY?)

Introducing: LEXC format for lexicon



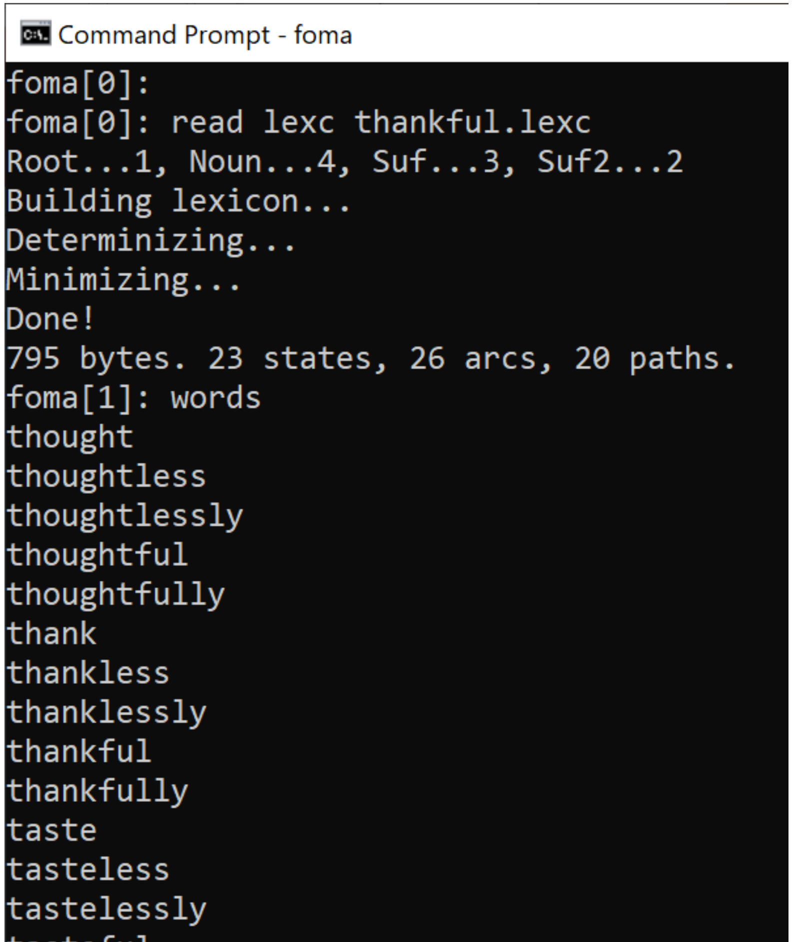
```
foma[0]: regex [t h a n k | j o y | t a s t e | t h o u g h t] ([f u l | l e s s] (l y)) ;
795 bytes. 23 states, 26 arcs, 20 paths.
```

Imagine writing this for entire English nouns... foma is ill-suited!

```
LEXICON Root
Noun;

LEXICON Noun
thank           Suf;
joy             Suf;
taste           Suf;
thought         Suf;

LEXICON Suf
#;
ful             Suf2;
less            Suf2;

LEXICON Suf2
#;
ly              #;
```

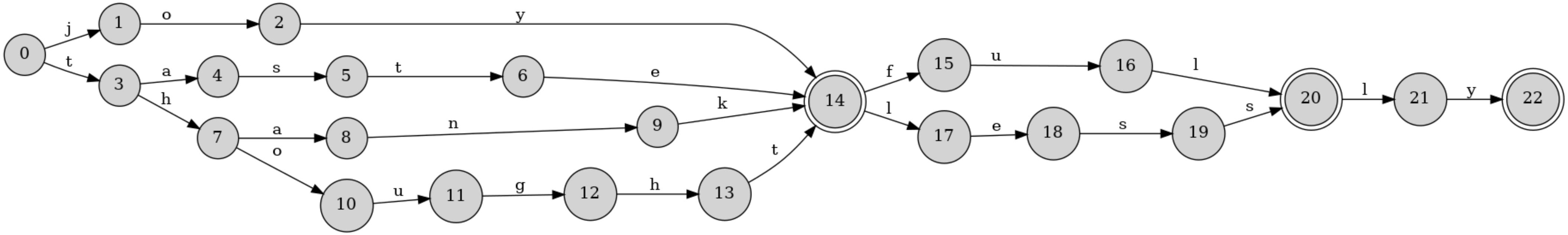
'thankful.lexc' file, In **LEXC format**. Optimal for lexicon building.

Compile through `read lexc` command.



```
foma[0]:
foma[0]: read lexc thankful.lexc
Root...1, Noun...4, Suf...3, Suf2...2
Building lexicon...
Determinizing...
Minimizing...
Done!
795 bytes. 23 states, 26 arcs, 20 paths.
foma[1]: words
thought
thoughtless
thoughtlessly
thoughtful
thoughtfully
thank
thankless
thanklessly
thankful
thankfully
taste
tasteless
tastelessly
tasteful
```

"thankfully" as a proper FSA



- Here, arc labels are **individual letters**.

  ➔ "thank" is NOT construed as a single, multi-character symbol but as concatenation of 't', 'h', 'a'…

- This example is just FSA and not a true FST, because the upper side and the lower side are the same.

Continuing from Exercise 8

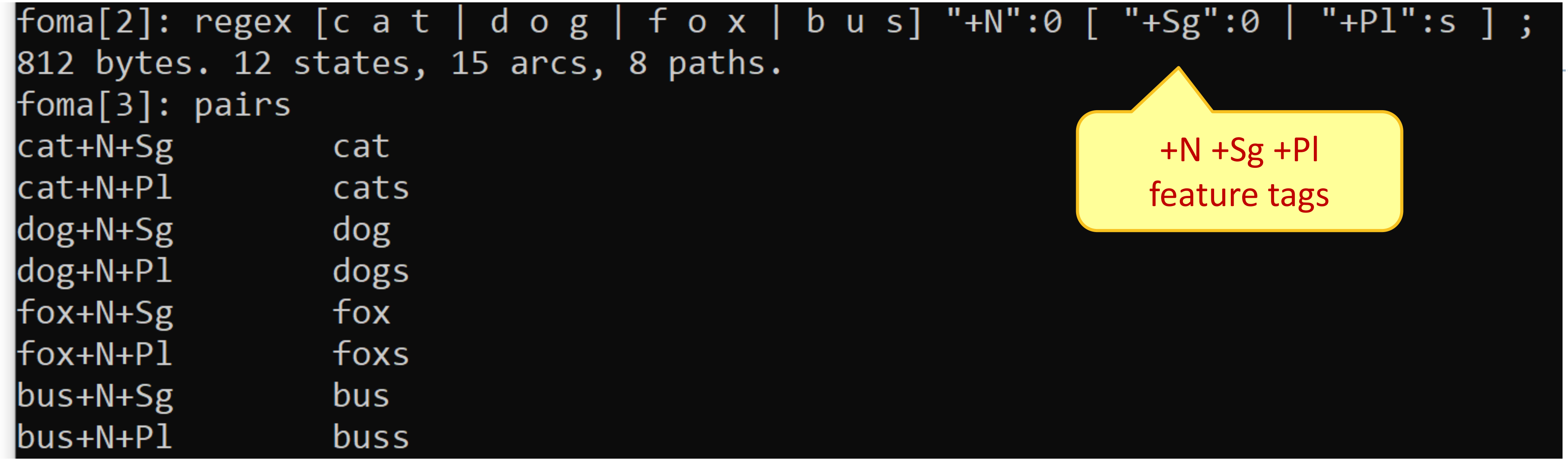
- Goal: build an FST that handles these nouns:

cat+N+Sg	cat		cat+N+Pl	cats
dog+N+Sg	dog		dog+N+Pl	dogs
fox+N+Sg	fox		fox+N+Pl	foxes
bus+N+Sg	bus		bus+N+Pl	buses

- Multi-char symbols:
  - +N denotes "noun" POS
  - +Pl denotes "plural" feature
  - +Sg denotes "singular" feature
- Morpheme boundary:
  - Let's use ^ this time: `cat^s`, etc.

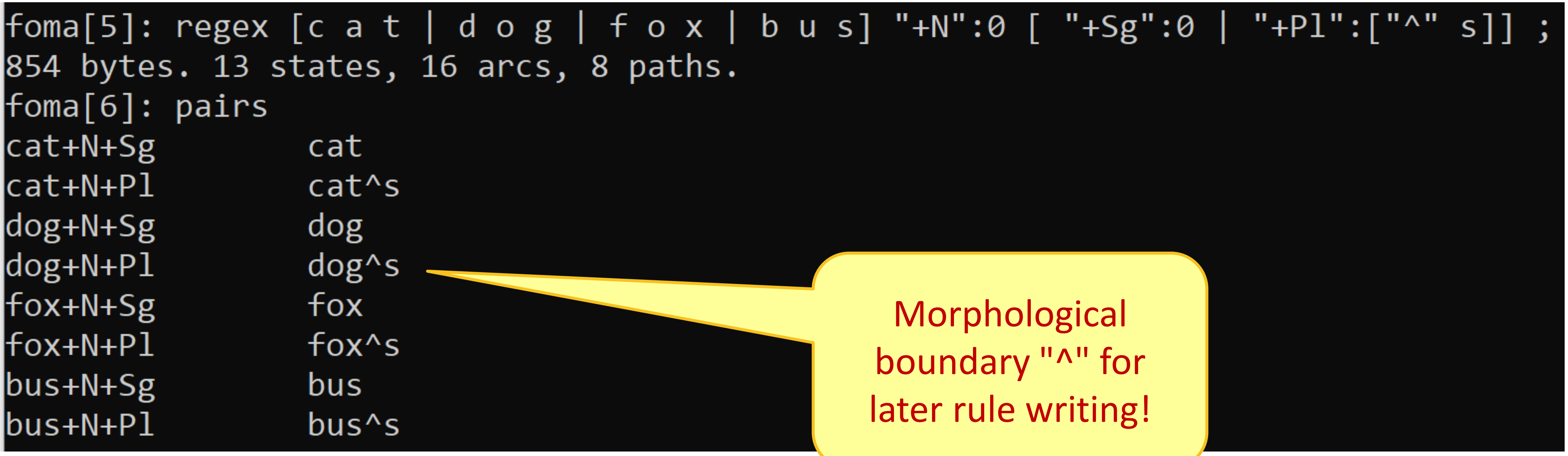
+ is part of grammatical tags, not a morpheme boundary!

^ is special char in foma, need to use `"^"`



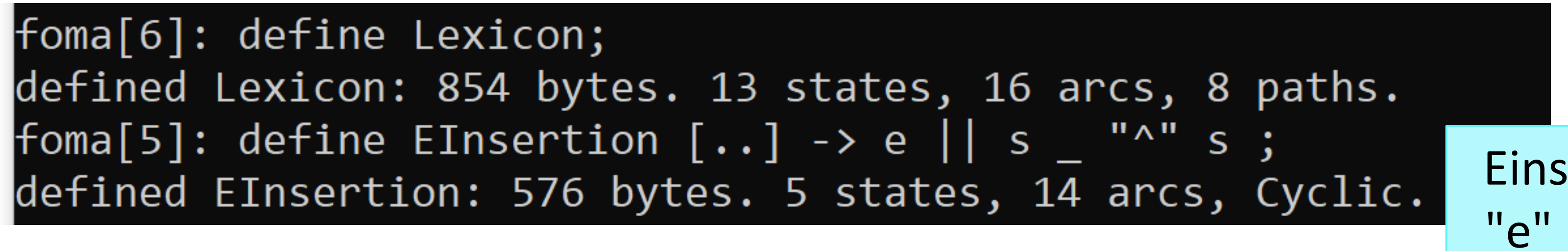
```
foma[2]: regex [c a t | d o g | f o x | b u s] "+N":0 [ "+Sg":0 | "+Pl":s ] ;
812 bytes. 12 states, 15 arcs, 8 paths.
foma[3]: pairs
cat+N+Sg        cat
cat+N+Pl        cats
dog+N+Sg        dog
dog+N+Pl        dogs
fox+N+Sg        fox
fox+N+Pl        foxs
bus+N+Sg        bus
bus+N+Pl        buss
```

+N +Sg +Pl
feature tags



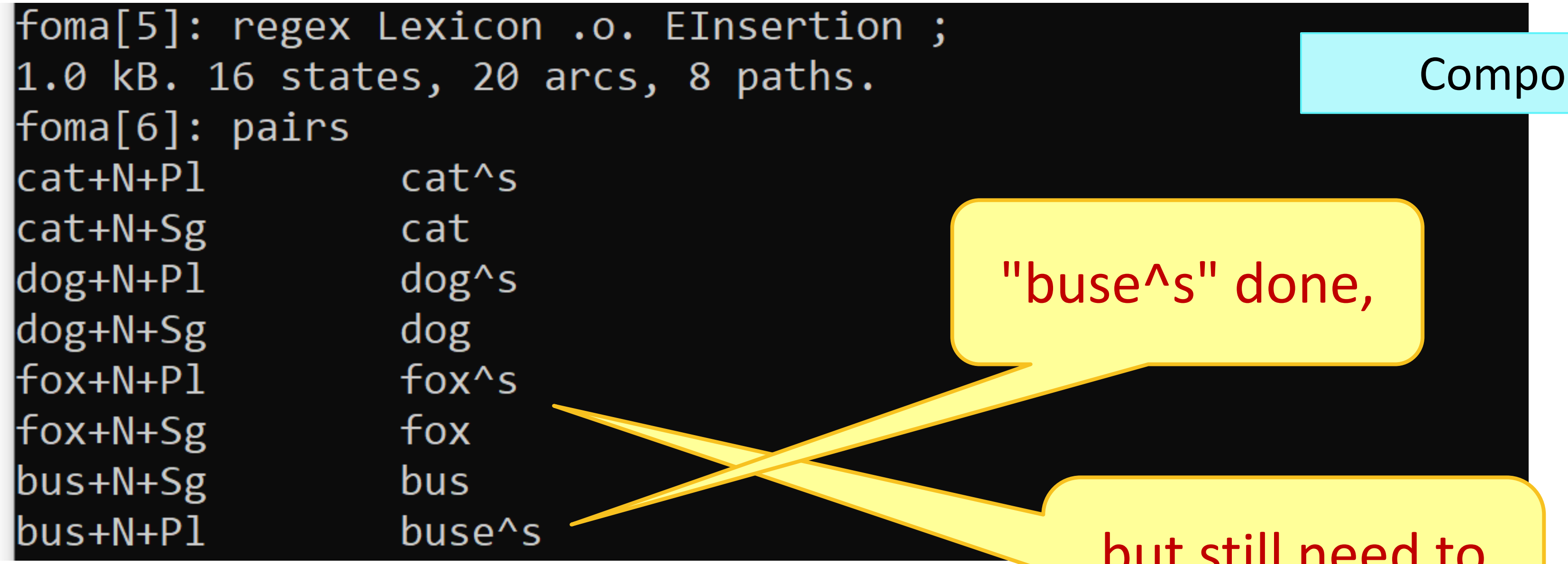
```
foma[5]: regex [c a t | d o g | f o x | b u s] "+N":0 [ "+Sg":0 | "+Pl":["^" s]] ;
854 bytes. 13 states, 16 arcs, 8 paths.
foma[6]: pairs
cat+N+Sg        cat
cat+N+Pl        cat^s
dog+N+Sg        dog
dog+N+Pl        dog^s
fox+N+Sg        fox
fox+N+Pl        fox^s
bus+N+Sg        bus
bus+N+Pl        bus^s
```

Morphological boundary "^" for later rule writing!



```
foma[6]: define Lexicon;
defined Lexicon: 854 bytes. 13 states, 16 arcs, 8 paths.
foma[5]: define EInsertion [..] -> e || s _ "^" s ;
defined EInsertion: 576 bytes. 5 states, 14 arcs, Cyclic.
```

Einsertion rule inserts "e" between s and ^ s

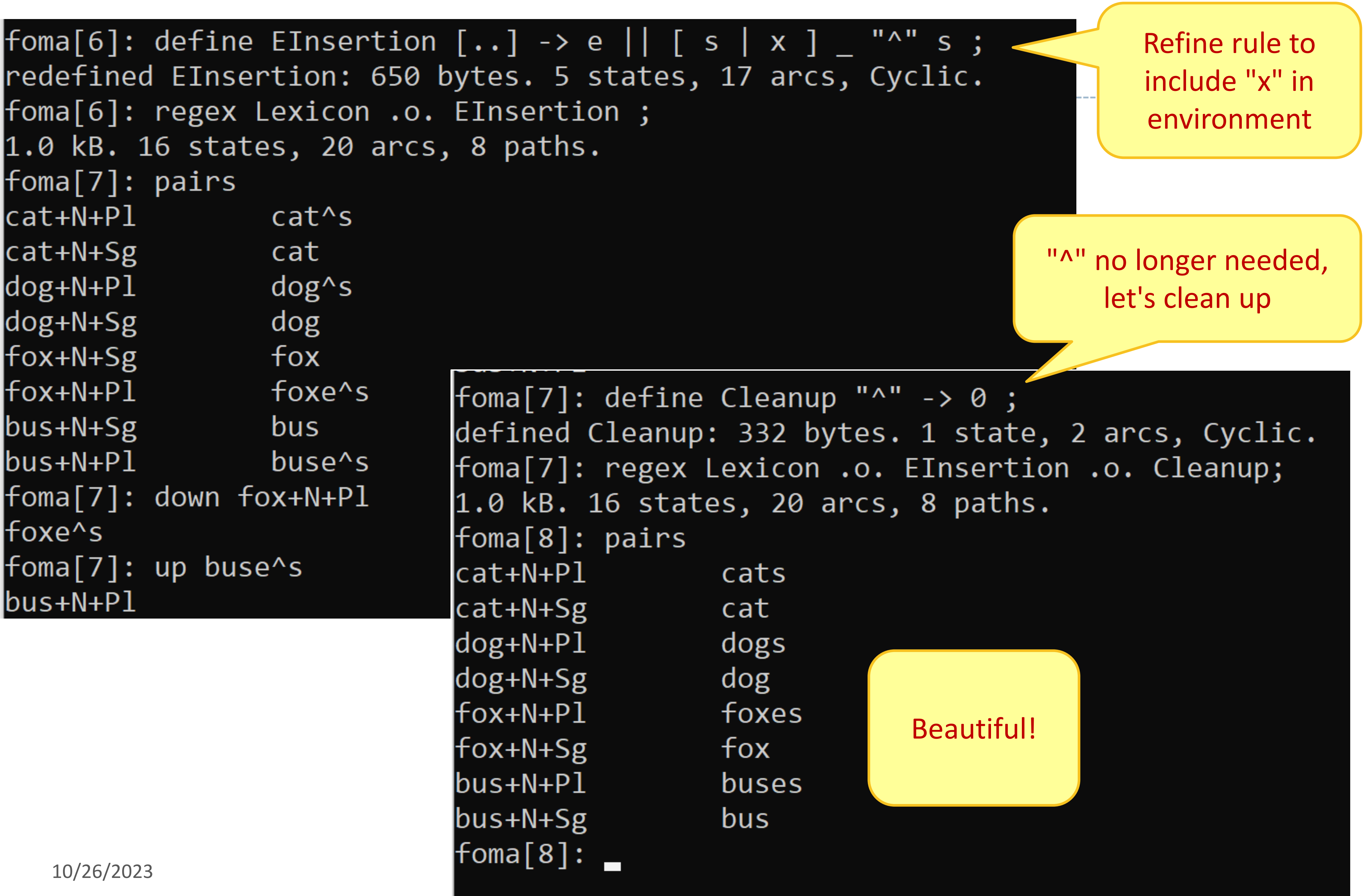


```
foma[5]: regex Lexicon .o. EInsertion ;
1.0 kB. 16 states, 20 arcs, 8 paths.
foma[6]: pairs
cat+N+Pl        cat^s
cat+N+Sg        cat
dog+N+Pl        dog^s
dog+N+Sg        dog
fox+N+Pl        fox^s
fox+N+Sg        fox
bus+N+Sg        bus
bus+N+Pl        buse^s
```

Compose!

"buse^s" done,

but still need to handle "fox^s"



```
foma[6]: define EInsertion [..] -> e || [ s | x ] _ "^" s ;
redefined EInsertion: 650 bytes. 5 states, 17 arcs, Cyclic.
foma[6]: regex Lexicon .o. EInsertion ;
1.0 kB. 16 states, 20 arcs, 8 paths.
foma[7]: pairs
cat+N+Pl        cat^s
cat+N+Sg        cat
dog+N+Pl        dog^s
dog+N+Sg        dog
fox+N+Sg        fox
fox+N+Pl        foxe^s
bus+N+Sg        bus
bus+N+Pl        buse^s
foma[7]: down fox+N+Pl
foxe^s
foma[7]: up buse^s
bus+N+Pl
```

Refine rule to include "x" in environment

"^" no longer needed, let's clean up

```
foma[7]: define Cleanup "^" -> 0 ;
defined Cleanup: 332 bytes. 1 state, 2 arcs, Cyclic.
foma[7]: regex Lexicon .o. EInsertion .o. Cleanup;
1.0 kB. 16 states, 20 arcs, 8 paths.
foma[8]: pairs
cat+N+Pl        cats
cat+N+Sg        cat
dog+N+Pl        dogs
dog+N+Sg        dog
fox+N+Pl        foxes
fox+N+Sg        fox
bus+N+Pl        buses
bus+N+Sg        bus
foma[8]: _
```

Beautiful!

As a LEXC script file

cats.lexc

```
Multichar_Symbols +N +Sg +Pl

LEXICON Root
Noun;

LEXICON Noun
cat                Nsuf;
dog                Nsuf;
tiger              Nsuf;
fox                Nsuf;
bus                Nsuf;

LEXICON Nsuf
+N+Sg:0            #;
+N+Pl:^s           #;
```

Multicharacter symbols (tags) must be declared.

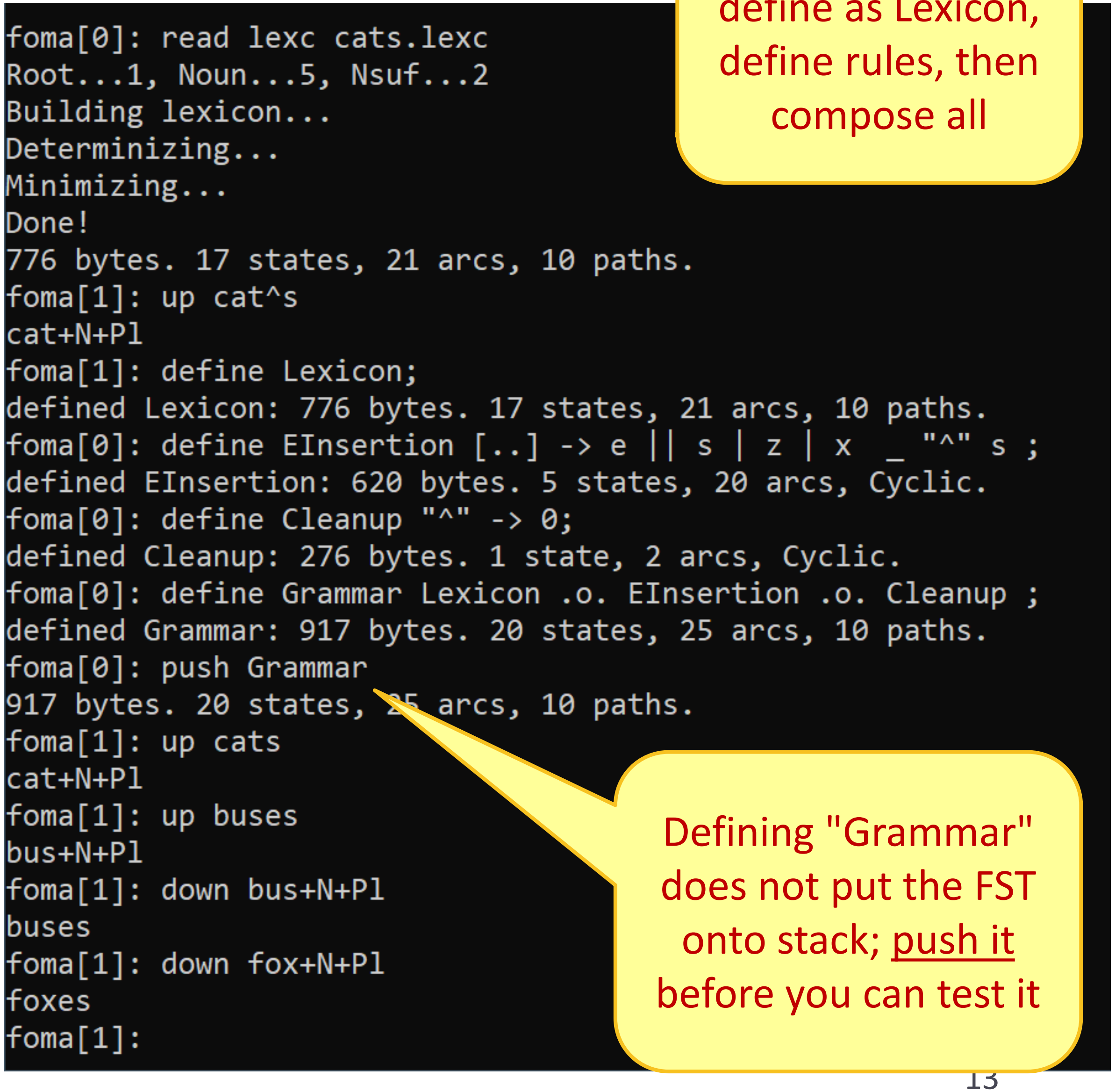
In LEXC, there is no need to space out characters.

Assumption: "abc" is three concatenated symbols *unless otherwise declared*

LEXC + cascading rules

cats.lexc

```
Multichar_Symbols +N +Sg +Pl

LEXICON Root
Noun;

LEXICON Noun
cat             Nsuf;
dog             Nsuf;
tiger           Nsuf;
fox             Nsuf;
bus             Nsuf;

LEXICON Nsuf
+N+Sg:0         #;
+N+Pl:^s        #;
```



```
foma[0]: read lexc cats.lexc
Root...1, Noun...5, Nsuf...2
Building lexicon...
Determinizing...
Minimizing...
Done!
776 bytes. 17 states, 21 arcs, 10 paths.
foma[1]: up cat^s
cat+N+Pl
foma[1]: define Lexicon;
defined Lexicon: 776 bytes. 17 states, 21 arcs, 10 paths.
foma[0]: define EInsertion [..] -> e || s | z | x _ "^" s ;
defined EInsertion: 620 bytes. 5 states, 20 arcs, Cyclic.
foma[0]: define Cleanup "^" -> 0;
defined Cleanup: 276 bytes. 1 state, 2 arcs, Cyclic.
foma[0]: define Grammar Lexicon .o. EInsertion .o. Cleanup ;
defined Grammar: 917 bytes. 20 states, 25 arcs, 10 paths.
foma[0]: push Grammar
917 bytes. 20 states, 25 arcs, 10 paths.
foma[1]: up cats
cat+N+Pl
foma[1]: up buses
bus+N+Pl
foma[1]: down bus+N+Pl
buses
foma[1]: down fox+N+Pl
foxes
foma[1]:
```

Read in LEXC file, define as Lexicon, define rules, then compose all

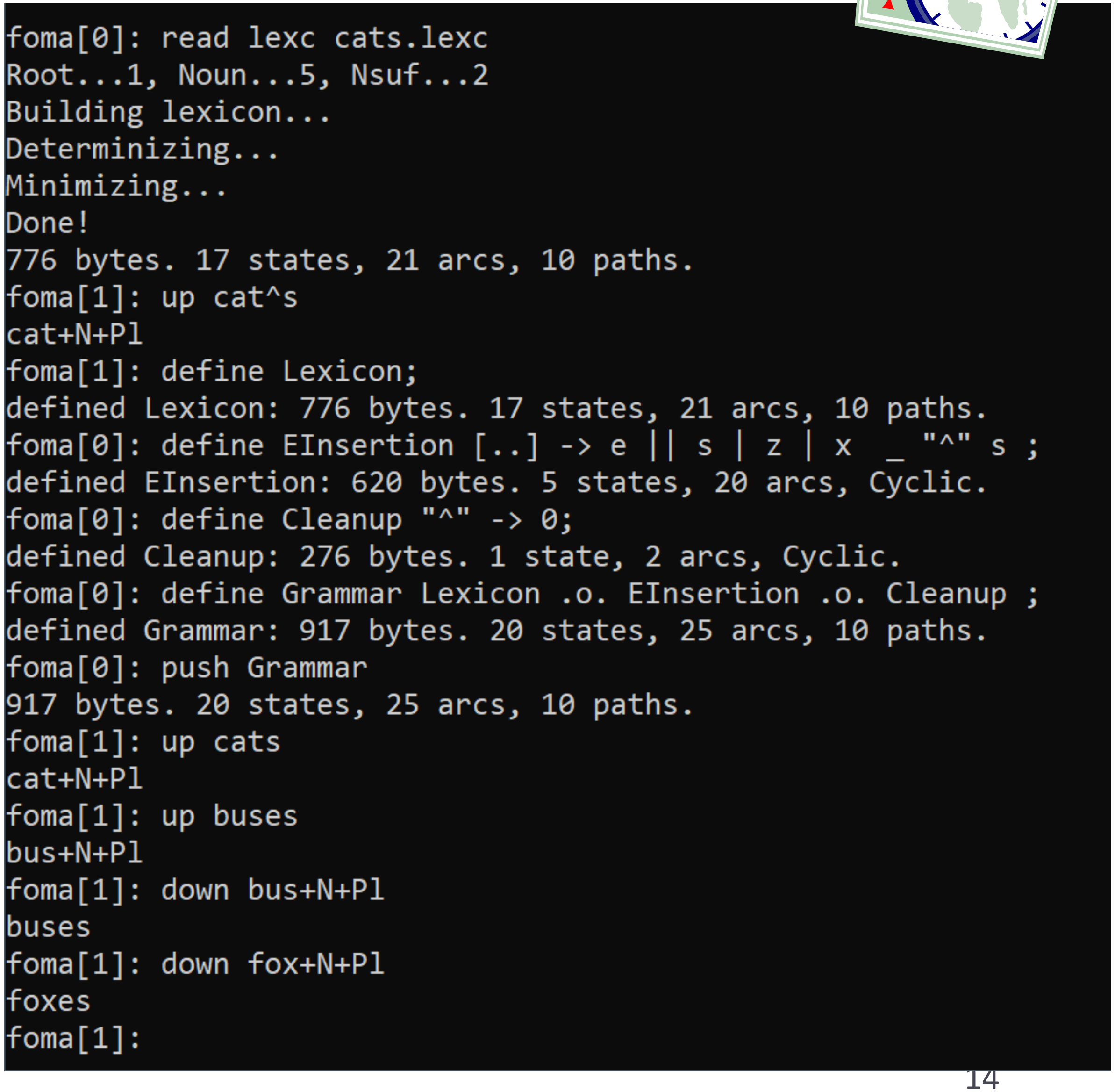
Defining "Grammar" does not put the FST onto stack; push it before you can test it

LEXC + cascading rules

Try out:
5 minutes

cats.lexc

```
Multichar_Symbols +N +Sg +Pl

LEXICON Root
Noun;

LEXICON Noun
cat             Nsuf;
dog             Nsuf;
tiger           Nsuf;
fox             Nsuf;
bus             Nsuf;

LEXICON Nsuf
+N+Sg:0         #;
+N+Pl:^s        #;
```



```
foma[0]: read lexc cats.lexc
Root...1, Noun...5, Nsuf...2
Building lexicon...
Determinizing...
Minimizing...
Done!
776 bytes. 17 states, 21 arcs, 10 paths.
foma[1]: up cat^s
cat+N+Pl
foma[1]: define Lexicon;
defined Lexicon: 776 bytes. 17 states, 21 arcs, 10 paths.
foma[0]: define EInsertion [..] -> e || s | z | x  _ "^" s ;
defined EInsertion: 620 bytes. 5 states, 20 arcs, Cyclic.
foma[0]: define Cleanup "^" -> 0;
defined Cleanup: 276 bytes. 1 state, 2 arcs, Cyclic.
foma[0]: define Grammar Lexicon .o. EInsertion .o. Cleanup ;
defined Grammar: 917 bytes. 20 states, 25 arcs, 10 paths.
foma[0]: push Grammar
917 bytes. 20 states, 25 arcs, 10 paths.
foma[1]: up cats
cat+N+Pl
foma[1]: up buses
bus+N+Pl
foma[1]: down bus+N+Pl
buses
foma[1]: down fox+N+Pl
foxes
foma[1]:
```

Mac users & plain text files

- File extensions don't strictly matter: you can name your files cats.lexc.txt and cats.foma.txt
  - Just make sure to call the "...txt" file name within foma
- Mac users: if you are using TextEdit, you must save your file as a plain text file, not "RTF" (rich text format) file!
- If the "save as" option does not show UTF8/plaintext option, you should first convert your file as a plain text file through a menu.
- Stuck? Tianyi can show you how.

LEXC + foma script

cats.lexc

```
Multichar_Symbols +N +Sg +Pl

LEXICON Root
Noun;

LEXICON Noun
cat             Nsuf;
dog             Nsuf;
tiger           Nsuf;
fox             Nsuf;
bus             Nsuf;

LEXICON Nsuf
+N+Sg:0         #;
+N+Pl:^s        #;
```

cats.foma

```
### cats.foma ###
read lexc cats.lexc
define Lexicon;

# E insertion rule
define EInsertion [..] -> e || s | z | x _ "^" s ;

# Cleanup: remove morpheme boundaries
define Cleanup "^" -> 0;

# Compose rules
define Grammar Lexicon     .o.
               EInsertion  .o.
               Cleanup;
```

The second half in a script file!!

The big composition operation builds our FST, names it "Grammar"

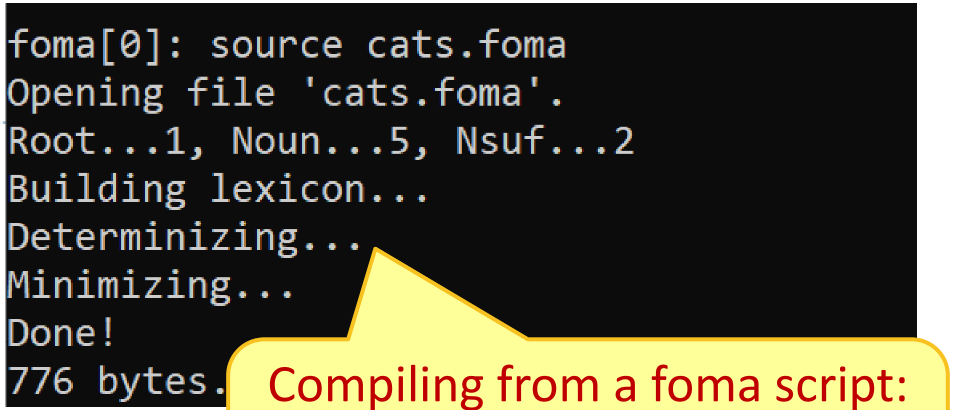
Running a foma script

cats.lexc

```
Multichar_Symbols +N +Sg +Pl

LEXICON Root
Noun;

LEXICON Noun
cat             Nsuf;
dog             Nsuf;
tiger           Nsuf;
fox             Nsuf;
bus             Nsuf;

LEXICON Nsuf
+N+Sg:0         #;
+N+Pl:^s        #;
```

cats.foma

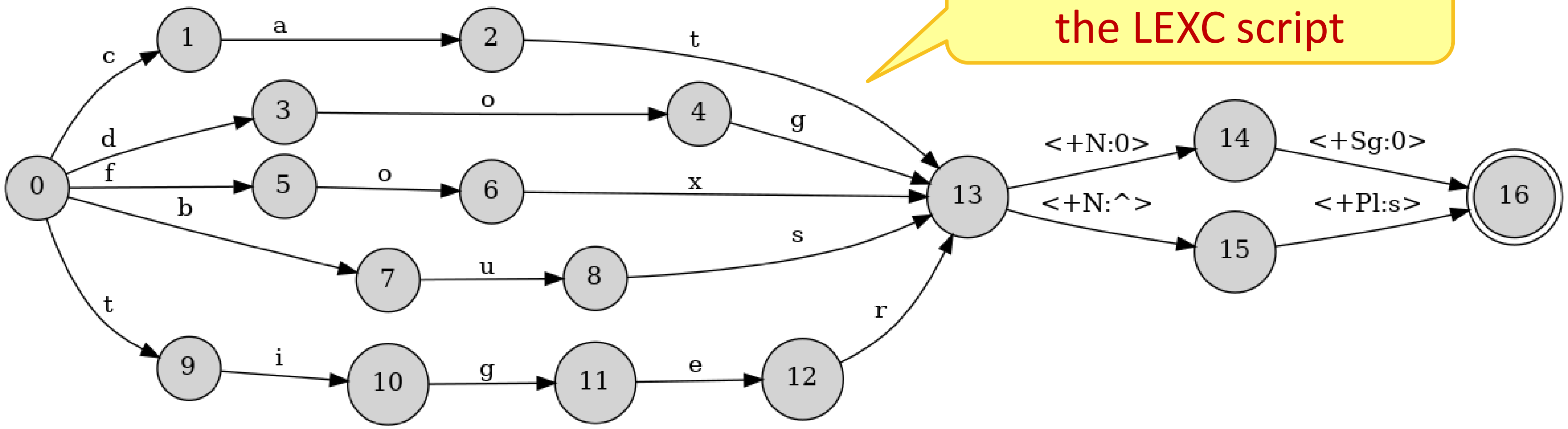
```
### cats.foma ###
read lexc cats.lexc
define Lexicon;

# E insertion rule
define EInsertion [..] -> e || s | z | x _ "^" s ;

# Cleanup: remove morpheme boundaries
define Cleanup "^" -> 0;

# Compose rules
define Grammar Lexicon     .o.
               EInsertion  .o.
               Cleanup;
```



```
foma[0]: source cats.foma
Opening file 'cats.foma'.
Root...1, Noun...5, Nsuf...2
Building lexicon...
Determinizing...
Minimizing...
Done!
776 bytes.
```

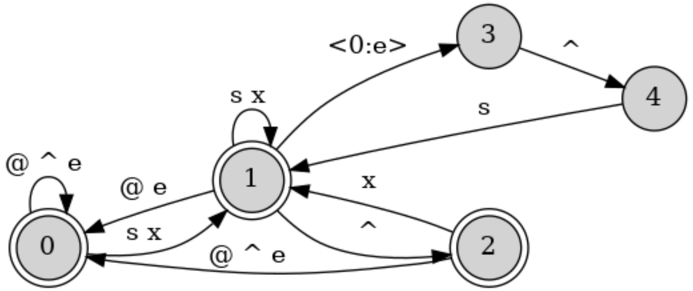
Compiling from a foma script: use `source FOMAFILE` command

Original lexicon from the LEXC script



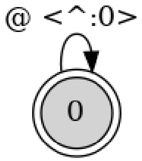
.o.

"e" insertion rule as FST

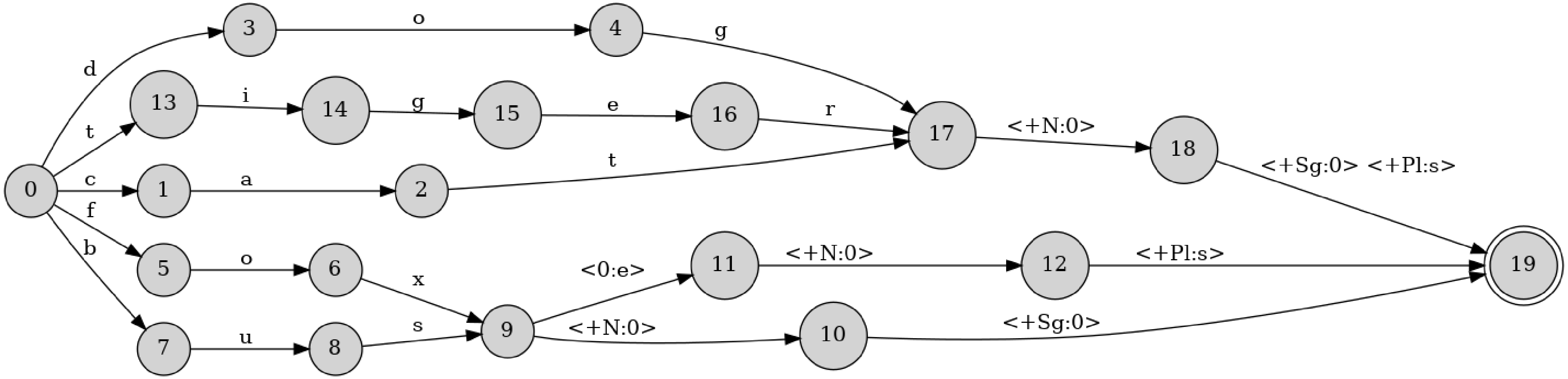


.o.

"^" cleanup rule



The resulting FST



- The output FST from the composition operation.
- Analyses ("fox+N+Pl") on the upper level, surface forms ("foxes") on the lower level.
- Used as a morphological analyzer/generator.
- FST operations are fast, efficient, and computationally elegant.

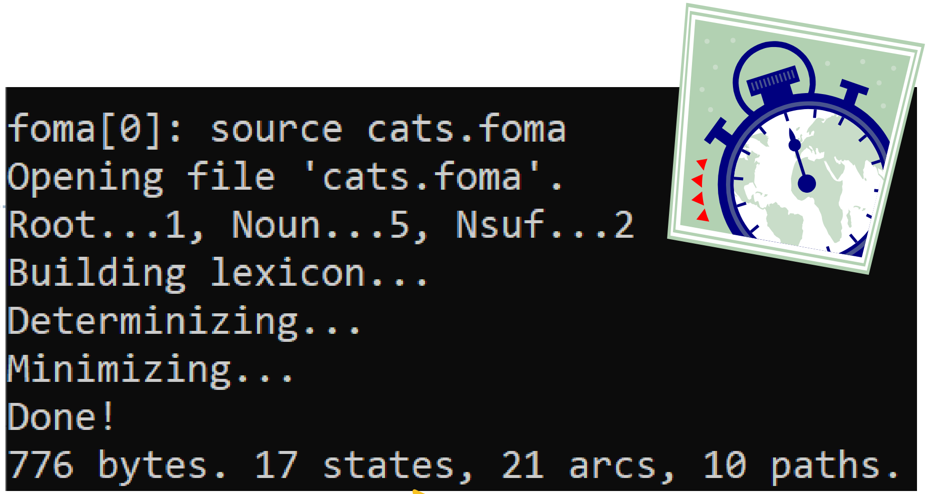
Try out

cats.lexc

```
Multichar_Symbols +N +Sg +Pl

LEXICON Root
Noun;

LEXICON Noun
cat             Nsuf;
dog             Nsuf;
tiger           Nsuf;
fox             Nsuf;
bus             Nsuf;

LEXICON Nsuf
+N+Sg:0         #;
+N+Pl:^s        #;
```

cats.foma

```
### cats.foma ###
read lexc cats.lexc
define Lexicon;

# E insertion rule
define EInsertion [..] -> e || s | z | x _ "^" s ;

# Cleanup: remove morpheme boundaries
define Cleanup "^" -> 0;

# Compose rules
define Grammar Lexicon     .o.
               EInsertion  .o.
               Cleanup;
```



```
foma[0]: source cats.foma
Opening file 'cats.foma'.
Root...1, Noun...5, Nsuf...2
Building lexicon...
Determinizing...
Minimizing...
Done!
776 bytes. 17 states, 21 arcs, 10 paths.
```

To test out the FST, run:
`push Grammar`

QUESTION:
How to add "teach" and "teaches"?

Adding a new POS category

cats.lexc

```
Multichar_Symbols +N +Sg +Pl +V +3P

LEXICON Root
Noun;
Verb;

LEXICON Noun
cat             Nsuf;
…
bus             Nsuf;

LEXICON Nsuf
+N+Sg:0         #;
+N+Pl:^s        #;

LEXICON Verb
teach           Vsuf;

LEXICON Vsuf
+V:0         #;
+V+3P+Sg:^s  #;
```

cats.foma

```
### cats.foma ###
read lexc cats.lexc
define Lexicon;

# E insertion rule
define EInsertion [..] -> e || s | z | x | c h _ "^" s ;

# Cleanup: remove morpheme boundaries
define Cleanup "^" -> 0;

# Compose rules
define Grammar Lexicon     .o.
               EInsertion  .o.
               Cleanup;
```

NOT a LEXC script,
a foma script!
Characters need
spacing out.

Wrapping up

- Homework 6 out
  - Due Tuesday

- Next week
  - FST morphology review
  - Part-of-speech tagging