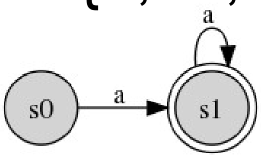
Creating LRs with FSTs
Part II

Compiling automata and transducers

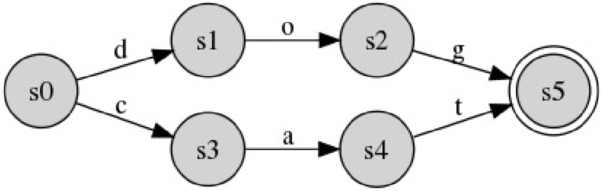
Mans Hulden
(University of Helsinki)
Iñaki Alegria
(University of The Basque Country)

Recap: finite automata

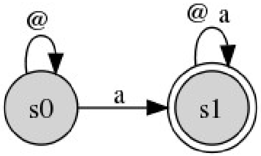
“one or more as”: {a,aa,...}:



the words “cat” and “dog”:



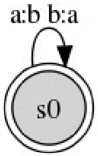
any word that contains at least an a:



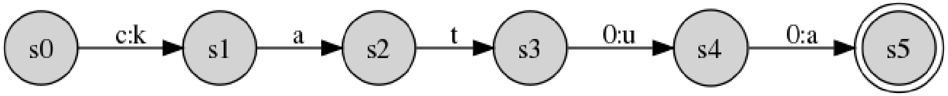
@ = any symbol outside the defined alphabet

Recap: finite transducers

Translates all a-symbols to b and vice versa

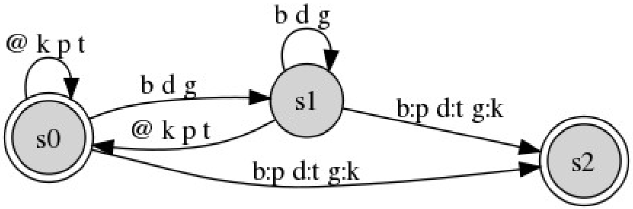


Translates “cat” to “katua”



Devoice end-of-word stops:
xleb → xlep, rad → rat, etc.

*Convention: a single symbol on an arc (a) is shorthand for an identity pair (a:a)



Birds-eye view

Generative phonology/morphology tends to model word-formation processes and allomorphy by minimizing different lexical forms of morphemes

Eg.:

cat → cats

fox → foxes

The plural morpheme s can be held to be invariant, while surface-variation is introduced by phonological rules

Birds-eye view

The theoretical mechanisms for such word-formation processes include a *lexicon component* (that guarantees proper morpheme ordering) and a *phonological component* (usually a set of ordered alternation rules)

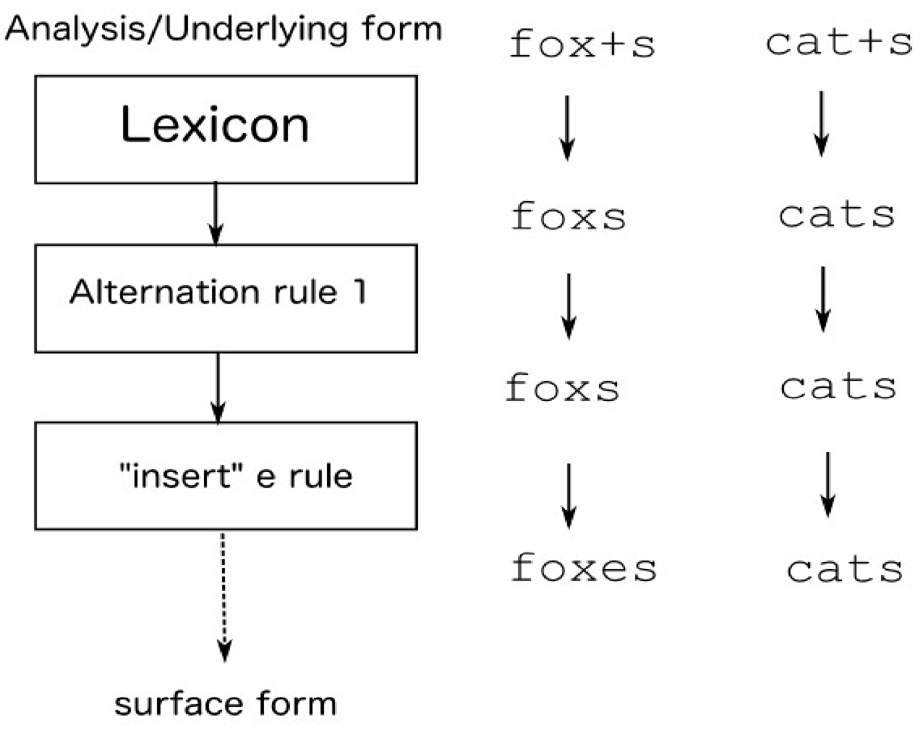
Words are “derived” by

(1) Constructing a morphotactically correct “underlying” form

(2) Subjecting this underlying form to various rewriting rules

Birds-eye view

Two different derivations



Analysis/Underlying form	fox+s	cat+s
Lexicon	foxs	cats
Alternation rule 1	foxs	cats
"insert" e rule	foxes	cats
surface form		

Birds-eye view

- The different stages of derivation are modeled through transducers
- The transducers are joined together by composing, yielding a monolithic transducer with only a relation between the surface and underlying forms
- Transducers are built by a special type of regular expressions...

Introduction to *foma*

A general-purpose tool for constructing and manipulating automata and transducers

Contains a regular expression compiler to convert expressions (including “rewrite rules”) to automata and transducers

Contains a lexc-parser to construct transducers from lexicon descriptions

Interface and regular expression formalism somewhat compatible with the commercial xfst and lexc tools by Xerox

Available at http://foma.sf.net

API available (in C) for integration with other programs

[source & binaries for Linux, Mac, and Windows]

Introduction to *foma*

- Unix-style command-line tool with interface
- Installation & starting
- Download appropriate files from http://foma.sf.net
- Standard fare: place “foma” in your /usr/local/bin or /usr/bin (Linux and Mac), etc.
  - Experimental support for FSM visualization (Linux and Mac)
  - Linux: visualization requires “GraphViz” and “gqview”
    - Ubuntu example:
    - sudo apt-get install graphviz
    - sudo apt-get install gqview
  - Mac:
    - Visualization requires GraphViz for OSX from http://www.pixelglow.net

foma: hands-on

Compiling regular expressions: regex

```
regex a+;
regex c a t | d o g;
regex ?* a ?*;
```

```
regex [a:b | b:a]*;
regex [c a t]:[k a t u a];
regex b -> p , g -> k, d -> t || _ .#.;
```

[demo]

foma: hands-on

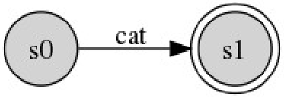
<table>
<tr><td>(space)</td><td>concatenation</td></tr>
<tr><td>|</td><td>union</td></tr>
<tr><td>*</td><td>Kleene star</td></tr>
<tr><td>&</td><td>Intersection</td></tr>
<tr><td>~</td><td>Complement</td></tr>
</table>

foma: ordinary symbols

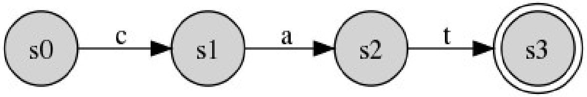
Single-character symbols:
a, b, c, Ω, ب , β, etc.

Multi-character symbols:
[Noun], +3pSg, @a_symbol@, cat, dog

```
foma[0]: regex cat;
168 bytes. 2 states, 1 arcs, 1 path.
```



```
foma[1]: regex c a t;
257 bytes. 4 states, 3 arcs, 1 path.
```



foma: special symbols

0	the empty string (epsilon)
?	"any" symbol (similar to . in grep/perl/awk/sed-regexes, or Σ in "formal language" regexes)

foma: contd.

testing automata against words:

```
foma[0]: regex ?* a ?*;
261 bytes. 2 states, 4 arcs, Cyclic.
foma[1]: down
apply down> ab
ab
apply down> xax
xax
apply down> bbx
???
apply down>^D
foma[1]:
```

foma: contd.

running transducers:

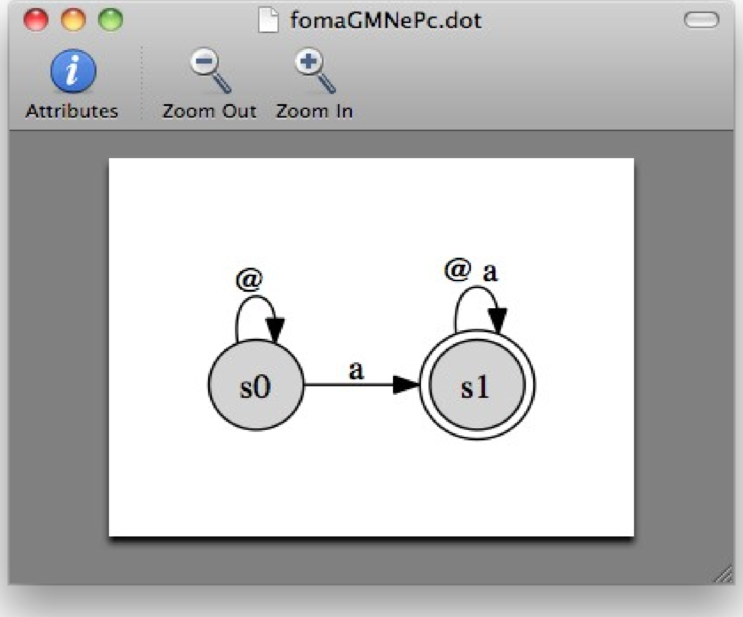
```
foma[0]: regex [c a t]:[k a t u a];
317 bytes. 6 states, 5 arcs, 1 path.
foma[1]: down
apply down> cat
katua
apply down> dog
???

foma[1]: up
apply up> katua
cat
```

Examining FSMs

```
foma[0]: regex ?* a ?*;
261 bytes. 2 states, 4 arcs, Cyclic.
foma[1]: net
Sigma: @ a
Size: 1.
Net: 41A7
Flags: deterministic pruned minimized epsilon_free
Arity: 1
Ss0:   @ -> s0, a -> fs1.
fs1:   @ -> fs1, a -> fs1.
foma[1]:
```

Examining FSMs visually

```
foma[0]: regex ?* a ?*;
261 bytes. 2 states, 4 arcs, Cyclic.
foma[1]: view
foma[1]:
```



More about foma

Labeling FSMs: the define command

```
foma[0]: define V [a|e|i|o|u];
defined V: 317 bytes. 2 states, 5 arcs, 5 paths.

foma[0]: define StartsWithVowel [V ?*];
defined StartsWithVowel: 429 bytes. 2 states, 11 arcs, Cyclic.
foma[0]:
```

Define contd.

```
foma[0]: define V [a|e|i|o|u];
redefined V: 317 bytes. 2 states, 5 arcs, 5 paths.

foma[0]: define C [b|d|g|k|m|n|p|s|t|v|z];
defined C: 497 bytes. 2 states, 11 arcs, 11 paths.

foma[0]: define Syllable [C* V+ C*];
defined Syllable: 1.0 kB. 3 states, 43 arcs, Cyclic.

foma[0]: define PhonologicalWord Syllable+;
defined PhonologicalWord: 887 bytes. 2 states, 32 arcs, Cyclic.

foma[0]: print defined
V                 317 bytes. 2 states, 5 arcs, 5 paths.
StartsWithVowel   429 bytes. 2 states, 11 arcs, Cyclic.
C                 497 bytes. 2 states, 11 arcs, 11 paths.
Syllable          1.0 kB. 3 states, 43 arcs, Cyclic.
PhonologicalWord  887 bytes. 2 states, 32 arcs, Cyclic.
```

Transducer operations

Composition (operator: .o.)

```
foma[0]: define EngBasque [c a t]:[k a t u a];
defined EngBasque: 317 bytes. 6 states, 5 arcs, 1 path.
foma[0]: define BasqueFinn [k a t u a]:[k i s s a];
defined BasqueFinn: 331 bytes. 6 states, 5 arcs, 1 path.
foma[0]: regex EngBasque .o. BasqueFinn;
345 bytes. 6 states, 5 arcs, 1 path.
foma[1]: down
apply down> cat
kissa
apply down>
```

Replacement rules

Simple replacement:

```
foma[0]: regex a -> b ;
290 bytes. 1 states, 3 arcs, Cyclic.
foma[1]: down
apply down> a
b
apply down> axa
bxb
apply down>
```

Replacement rules

Conditional replacement

```
foma[0]: regex a -> b || c _ d;
526 bytes. 4 states, 16 arcs, Cyclic.
foma[1]: down
apply down> cadca
cbdca
apply down>
```

Replacement rules

Conditional replacement w/ multiple contexts.

```
foma[0]: regex a -> b || c _ d , e _ f;
890 bytes. 7 states, 37 arcs, Cyclic.
foma[1]: down
apply down> cadeaf
cbdebf
apply down> a
a
apply down>
```

Replacement rules

“Parallel” rules, the .#.-symbol

Example: devoice some word-final stops

```
foma[0]: regex b -> p , g -> k , d -> t || _ .#. ;
634 bytes. 3 states, 20 arcs, Cyclic.
foma[1]: down
apply down> cab
cap
apply down> dog
dok
apply down> dad
dat
```

Replacement rules & composition

We can define multiple different rules and compose them into one single transducer:

```
foma[0]: define Rule1 a -> b || c _ ;
defined Rule1: 384 bytes. 2 states, 8 arcs, Cyclic.
foma[0]: define Rule2 b -> c || _ d;
defined Rule2: 416 bytes. 3 states, 10 arcs, Cyclic.
foma[0]: regex Rule1 .o. Rule2;
574 bytes. 4 states, 19 arcs, Cyclic.
foma[1]: down
apply down> cad
ccd
apply down> ca
cb
apply down> ad
ad
```

Review of basic *foma commands*

- Compile regex:

  `regex regular-expression;`

- Name a FST/FSM using a regex:

  `define name regular-expression;`

- View (visually) a compiled regex:

  `view` or `view net`

- View (in text form) a compiled regex:

  `net` or `print net`

- Run a word through a transducer:

  `down <word>` or `apply down <word>`

- In the inverse direction:

  `up <word>` or `apply up <word>`

- Print all the words an automaton accepts:

  `words` or `print words`

- Only lower side words (for a transducer):

  `lower-words` or `print lower-words`

- Only upper-side words (for a transducer):

  `upper-words` or `print upper-words`

Review of basic *foma regexes*

- Special symbols 0 (epsilon) and ? (the “any” symbol)
- [and] are grouping symbols
- _ is a context separator (don't use in definitions)
- .#. is a special symbol indicating left or right word boundary in replacement rules
- Reserved symbols (operators) need to be quoted if used as symbols: eg. a “&” b;

space	concatenation
|	union
*	Kleene star
+	Kleene plus
&	Intersection
~	Complement
(A)	Optionality (identical to A | 0)

Transducer-related:

:	Cross-product
A -> B	Replacement rules
A -> B || C _ D	Context-conditioned replacement rules
.o.	Composition