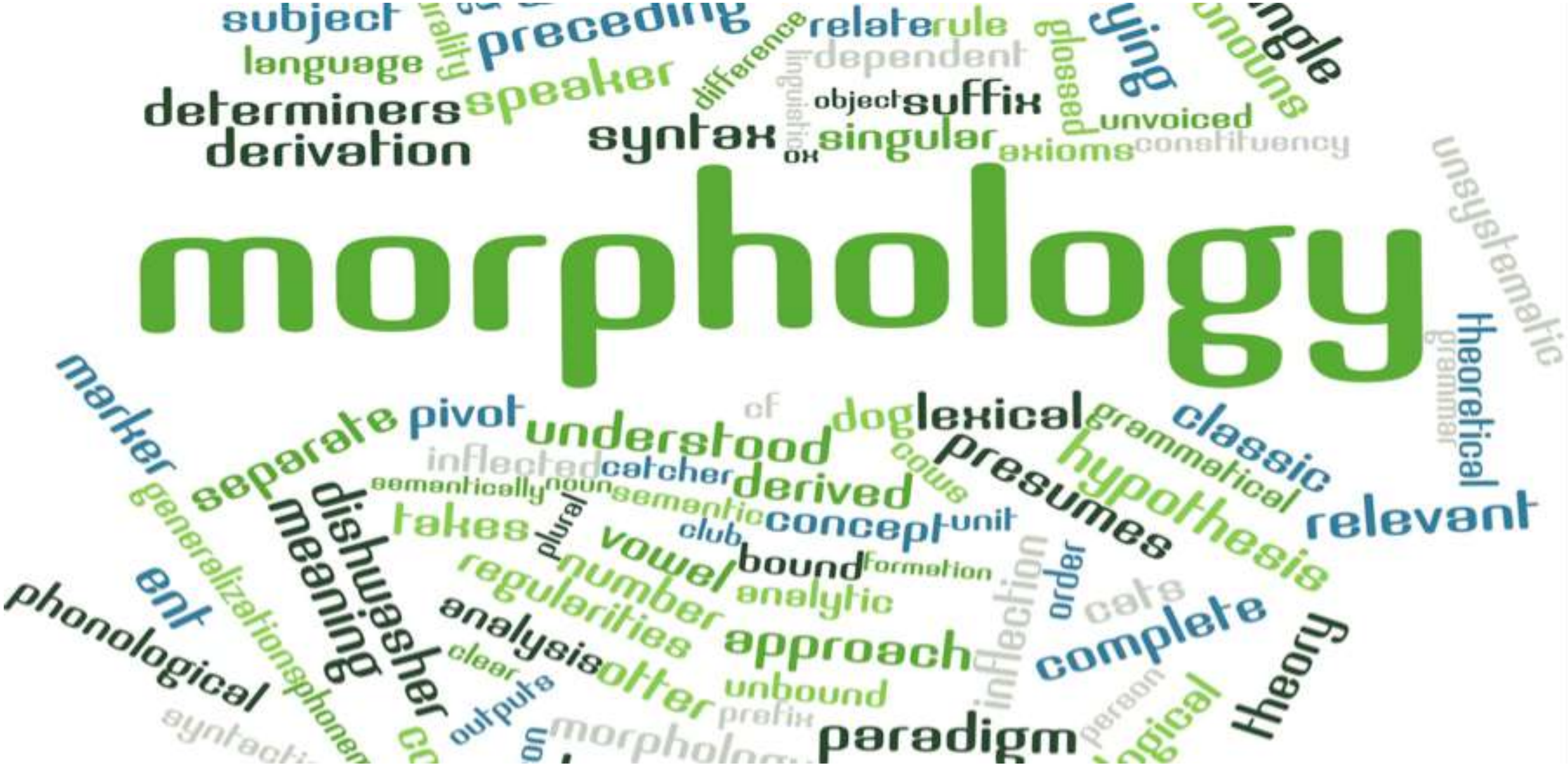
Finite State Morphology

by

David R. Mortensen



One Specialty: Morphology

Related specialty: phonology

What is Linguistic Morphology?

- Morphology is the study of the internal structure of words.
  - **Derivational morphology.** How new words are created from existing words.
    - *[become]*
    - *[[becom]ing]*
    - *[un[becom]ing]]*
  - **Inflectional morphology.** How features relevant to the syntactic context of a word are marked on that word.
    - **Der Mann** schläft. “The man is sleeping.”
    - Ich sehe **den Mann**. “I see the man.”
  - ***Compounding.*** Creating new words by combining existing words (not discussed much in this lecture).

Morphemes

- A venerable way of looking at morphology.
- **Morphemes.** Minimal pairings of form and meaning.
  - **Roots.** The “core” of a word that carries its basic meaning.
    - *apple* : ‘apple’
    - *walk* : ‘walk’
  - **Affixes** (**prefixes**, **suffixes**, **infixes**, and **circumfixes**). Morphemes that are added to a base (a root or stem) to perform either derivational or inflectional functions.
    - *un-* : ‘NEG’
    - -*s* : ‘PLURAL’

Morphologies of Languages Vary Widely

- Some traditional categories which are problematic but still useful:
  - **Analytic** or **isolating.** Some languages, like English and especially Chinese, have relatively few affixes (especially inflectional affixes).
  - **Agglutinative.** Most languages (including Hungarian, Finnish, Turkish, Swahili, Japanese, Korean, and most native languages of the Americas, Australia, and New Guinea) perform derivation and inflection by the concatenation of easily-segmentable affixes before, after, and in the root.
  - **Fusional** or **flexional**. Some languages, like German, Spanish, Russian, and Greek, inflect words with the addition of difficult-to-segment *desinences*. For example, the same sequence of segments added to nouns may realize both case and number features.

Isolating Languages: Chinese

- Morphological analyzers and generators are not very useful for isolating languages like English, Vietnamese, Yoruba, and Chinese.

- These languages have little morphology other than compounding.

- Chinese has few—if any—affixes (prefixes and suffixes):
  - 们： 我们， 你们， 他们， 。。。同志们
    mén: wǒmén, nǐmén, tāmén, tóngzhìmén
    plural: we, you (pl.), they comrades, LGBT people
  - A couple of “suffixes” that mark aspect: 着 *-zhě* ‘continuous aspect’

Agglutinative Languages: Swahili

Swahili	English
***m**-tu **a**-li-lala*	'The person slept'
***m**-tu **a**-**ta**-lala*	'The person will sleep'
***wa**-tu **wa**-**li**-lala*	'The people slept'
***wa**-tu **wa**-**ta**-lala*	'The people will sleep'

- Words written without hyphens or spaces between morphemes.
- Orange prefixes mark noun class (like gender, except **Swahili** has nine instead of two or three).
  - Verbs agree with nouns in noun class.
  - Adjectives also agree with nouns.
  - Very helpful in parsing.
- Black prefixes indicate tense.

Fusional Languages: A New World Spanish

	Singular			Plural		
	1st	2nd	3rd formal 2nd	1st	2nd	3rd
Present	am-o	am-as	am-a	am-a-mos	am-áis	am-an
Imperfect	am-ab-a	am-ab-as	am-ab-a	am-áb-a-mos	am-ab-ais	am-ab-an
Preterit	am-é	am-aste	am-ó	am-a-mos	am-asteis	am-aron
Future	am-aré	am-arás	am-ará	am-are-mos	am-aréis	am-arán
Conditional	am-aría	am-arías	am-aría	am-aría-mos	am-aríais	am-arían

Root-and-Pattern Morphology

- **Root-and-pattern**. A special kind of fusional morphology found in Arabic, Hebrew, and their cousins.
- Root usually consists of a sequence of consonants.
- Words are derived and, to some extent, inflected by patterns of vowels intercalated among the root consonants.
  - **kitaab** ‘book’
  - **kaatib** ‘writer; writing’
  - **maktab** ‘office; desk’
  - **maktaba** ‘library’

Other Non-Concatenative Morphological Processes

- Root-and-pattern morphology is an extreme case of the general class of morphology that is most difficult to model using computational methods.
- **Non-concatenative morphology** involves operations other than the concatenation of affixes with bases.
  - Infixation.
  - Reduplication. Can be prefixing, suffixing, and even infixing.
  - Internal change (tone change; stress shift; apophony, such as umlaut and ablaut).
  - Root-and-pattern morphology.
  - And more...

Polysynthetic Morphologies

- Polysynthetic morphologies are an interesting special case of agglutinative languages.

- Not only do they allow the formation and inflection of words with multiple, segmentable morphemes, they allows the creation of full "sentences" by morphological means.

- They often allow the incorporation of nouns into verbs.

- They may also have affixes that attach to verbs and take the place of nouns.

- Yupik Eskimo
  untu-ssur-qatar-ni-ksaite-ngqiggte-uq
  reindeer-hunt-FUT-say-NEG-again-3SG.INDIC
  'He had not yet said again that he was going to hunt reindeer.'

Language Comparison wrt FSTs

- Morphologies of all types can be analyzed using finite state methods.
- Some present more challenges than others.
  - **Analytic languages**. Trivial, since there is little or no morphology (other than compounding).
  - **Agglutinating languages**. Straightforward—finite state morphology was “made” for languages like this.
  - **Polysynthetic languages**. Similar to agglutinating languages, but with blurred lines between morphology and syntax.
  - **Fusional languages.** Easy enough to analyze using finite state method as long as one allows “morphemes” to have lots of simultaneous meanings and one is willing to employ some additional tricks.
  - **Root-and-pattern languages.** Require some very clever tricks.

Why Compute When Storage is Cheap?

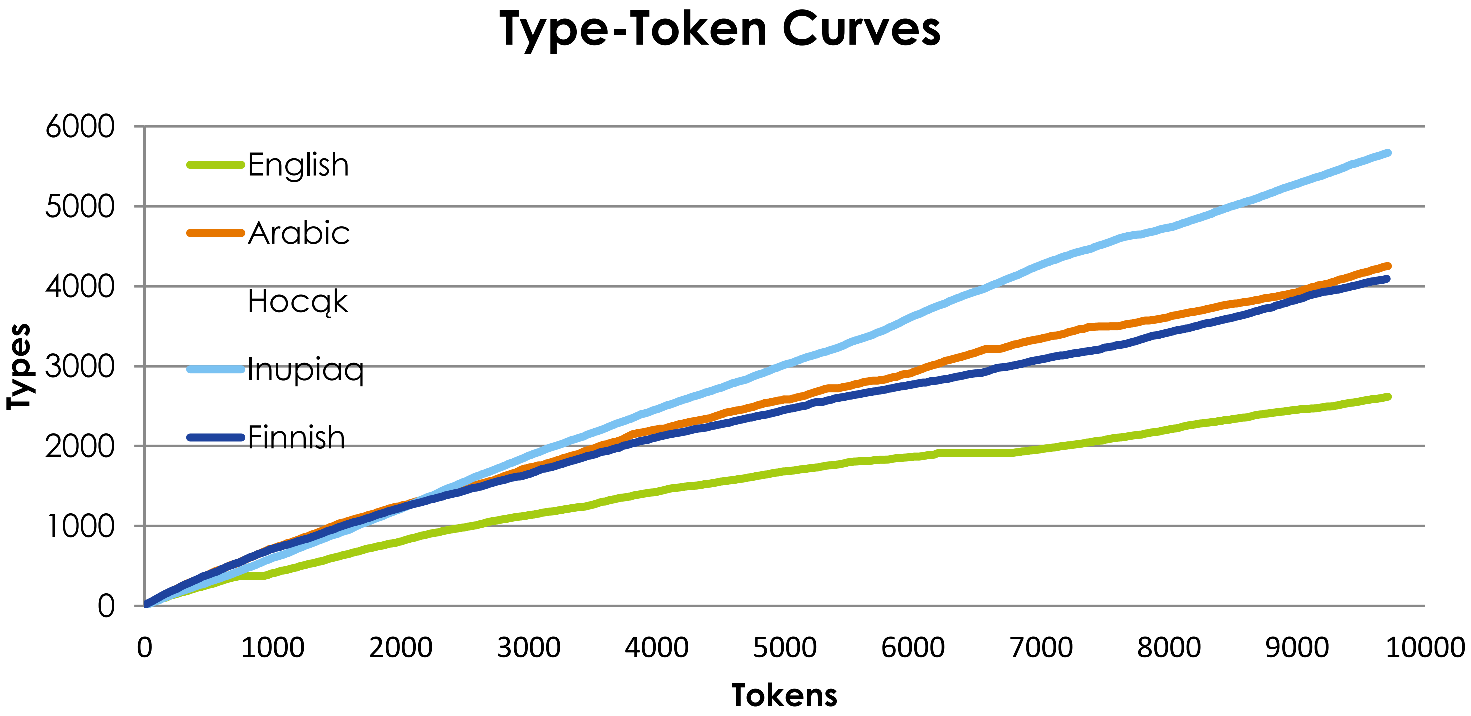
English is Easy

- You could get by performing sig n without implementing much En
- You can store most of the derive ost of their inflected forms.
- English **does** have productive morphological constructions, so you could encounter some out of dictionary issues, but for the most part, things would come out in the wash.

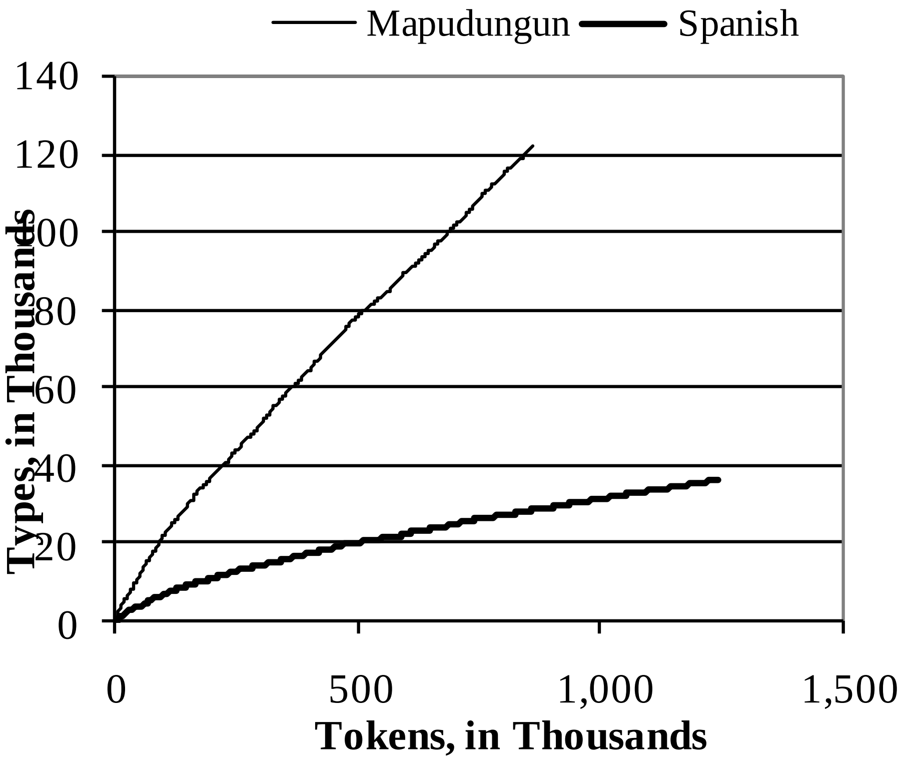
Morphological patterns that can be employed to derive or inflect new words.

- Chinese is even easier.
- **However, there are other languages.**

Inupiaq Needs an Analyzer/Generator



Mapathungun Is No Easier



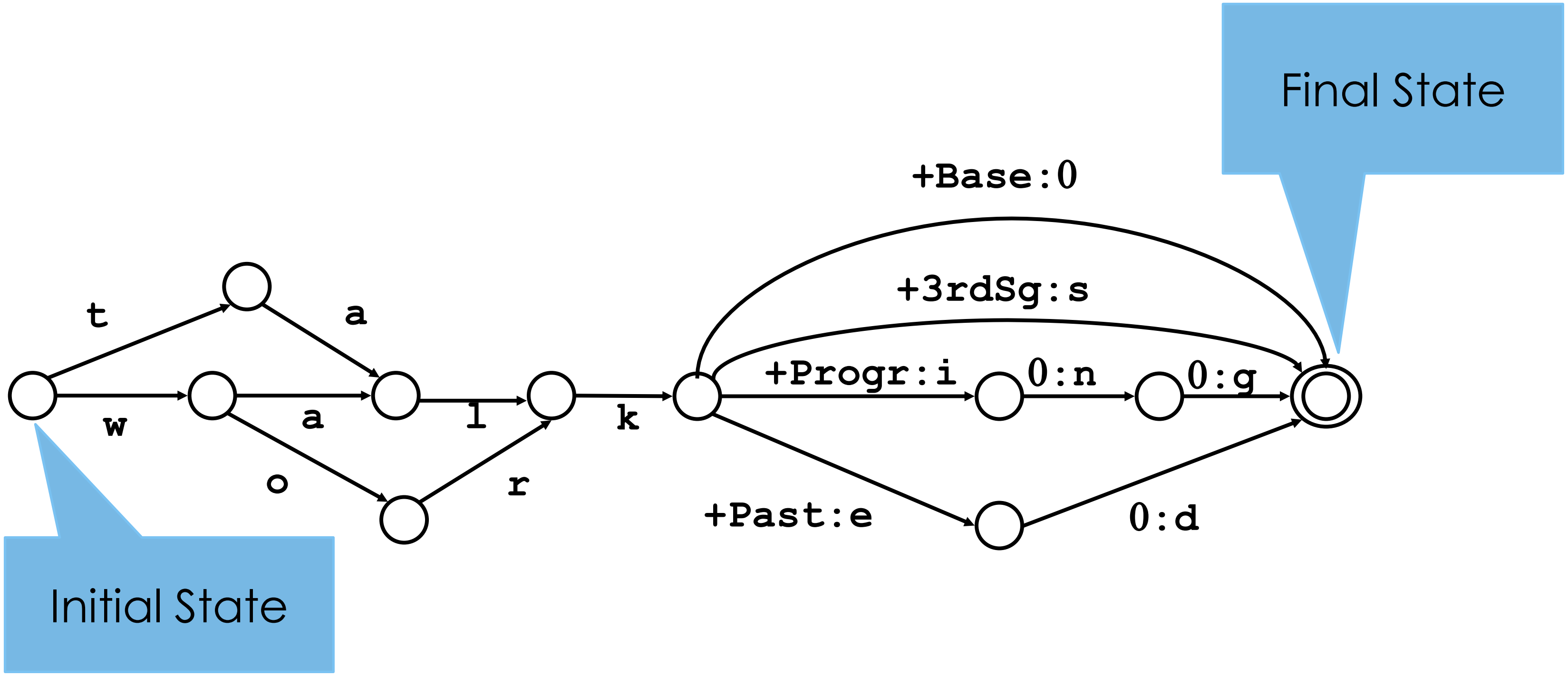
The Good News

- More than almost any other problem in computational linguistics, morphology is a solved problem (as long as you can afford to write rules by hand).
- Finite state morphology is one of the great successes of natural language processing.
- One brilliant aspect of using FSTs for morphology: the same code can handle both analysis and generation.

Finite State Morphology and Finite State POS Tagging

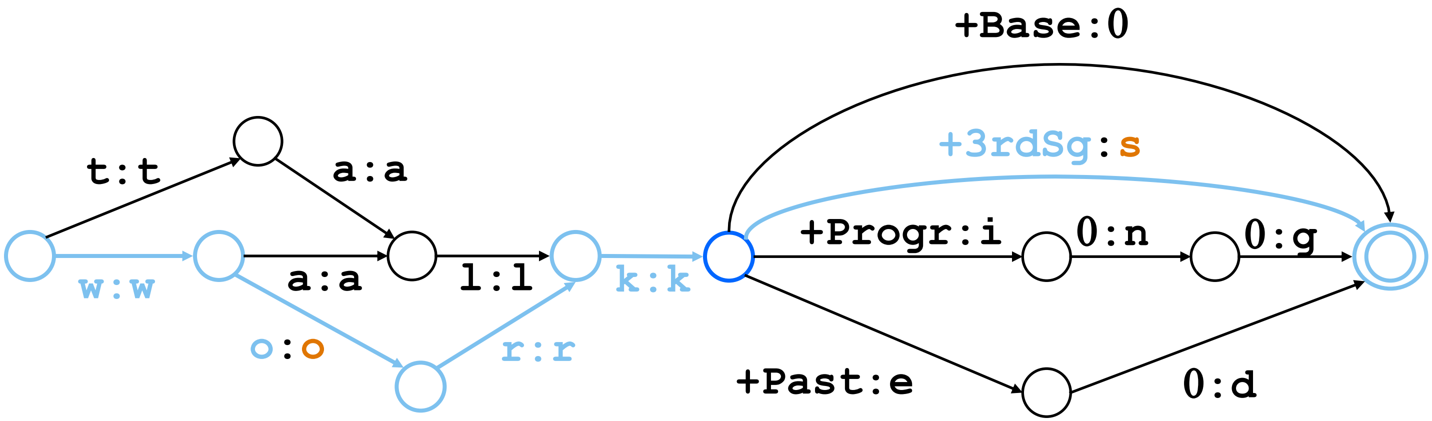
- **Morphological analysis** is not the same as **part of speech tagging**, though they are related in some ways.
- **Part of speech tagging**.
  - **Tokenization**. Break text into tokens.
  - **Lexical lookup**. Lookup each tag for each token.
  - **Disambiguation**. Choose tag for token, often on basis of statistical model (e.g. HMM).
- **Morphological analysis**.
  - Transduce each token as stem/translation and a sequence of morphological properties.
  - ***amaríamos*** → *love+Conditional+1P+Plural*

A Finite State Transducer



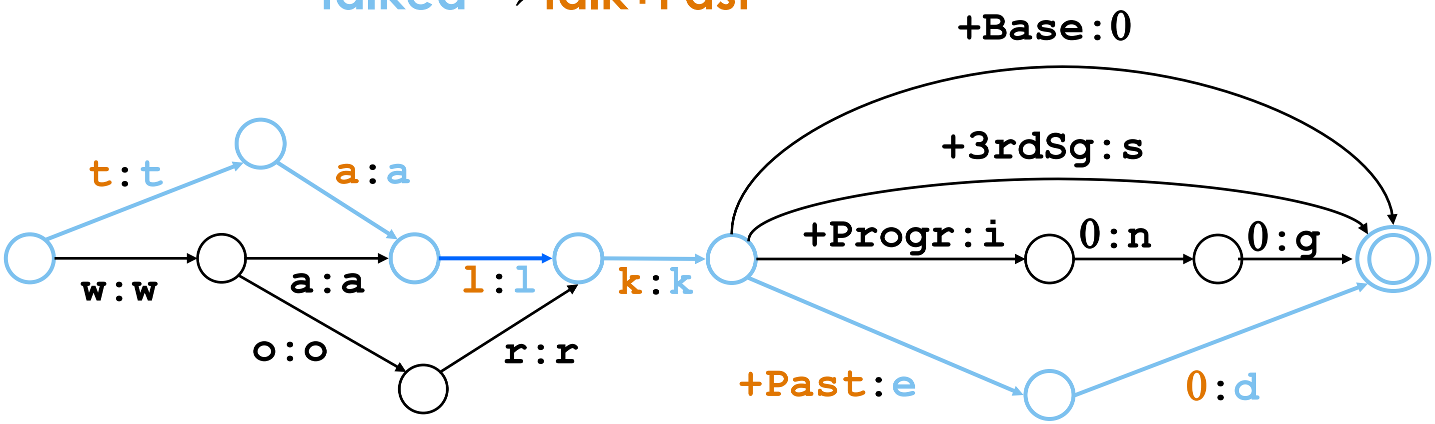
Generation

work+3rdSg → works

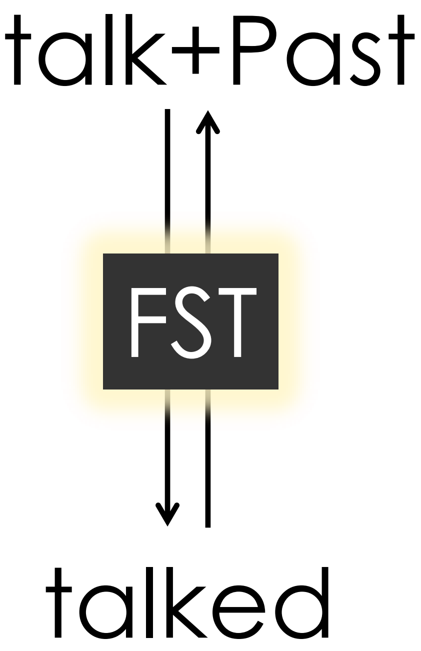


Analysis

talked → talk+Past



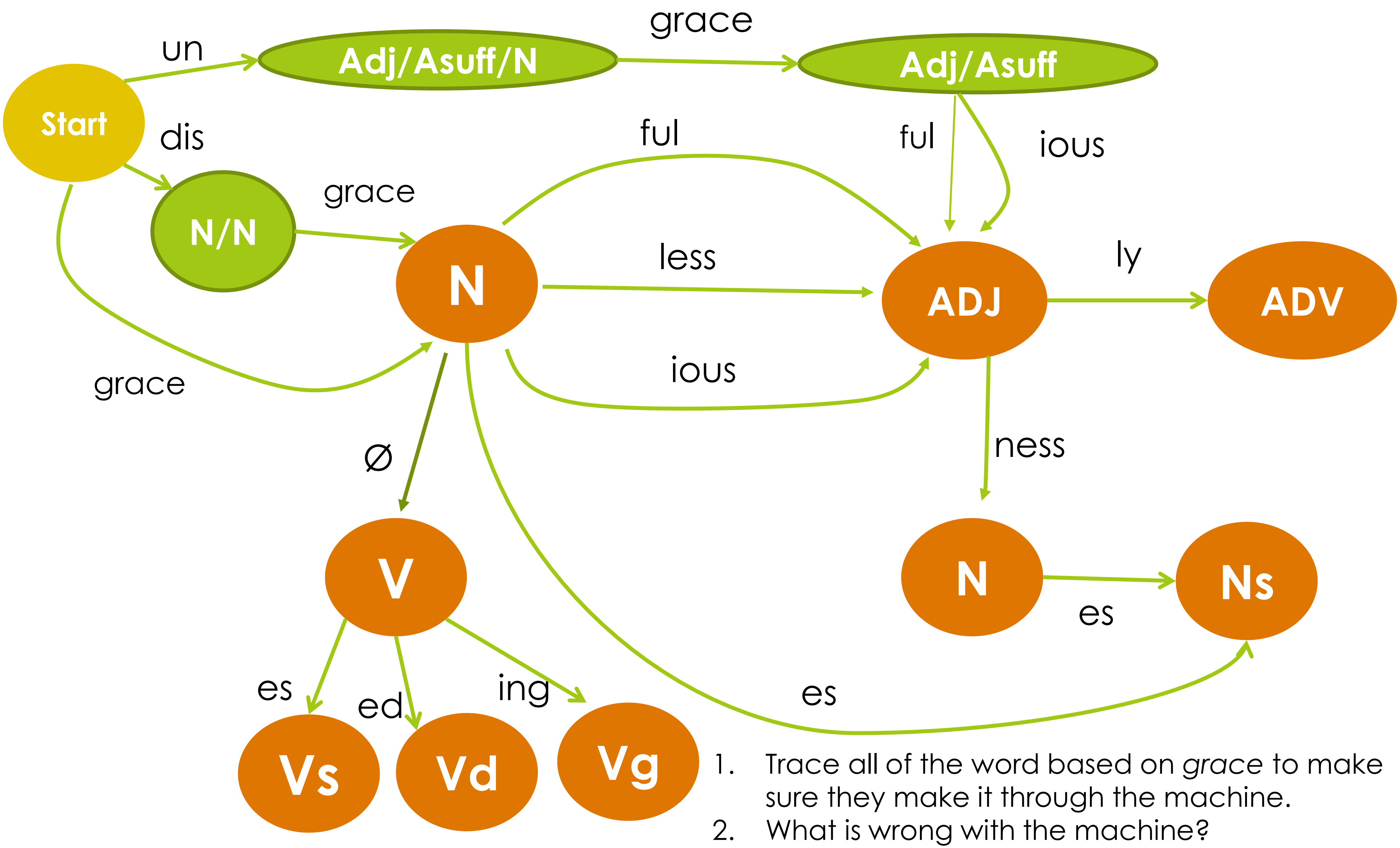
Upper Side/Lower Side



talk+Past

FST

talked



1. Trace all of the word based on *grace* to make sure they make it through the machine.
2. What is wrong with the machine? Overgeneration? Undergeneration? Redundancy? Does not capture structure and ambiguity?

Tools

- There are special finite state toolkits for building morphological tools (and other linguistic tools).
- The best-known of these is the **Xerox Finite State Tool** or **XFST**, which originated at Xerox PARC.
- There are open source reimplementations of XFST called **HFST** (Helsinki Finite State Technology) and **Foma**, which are not as fully optimized as XFST but which are sometimes more pleasant to use.
- None of these tools allow the construction of weighted FSTs.

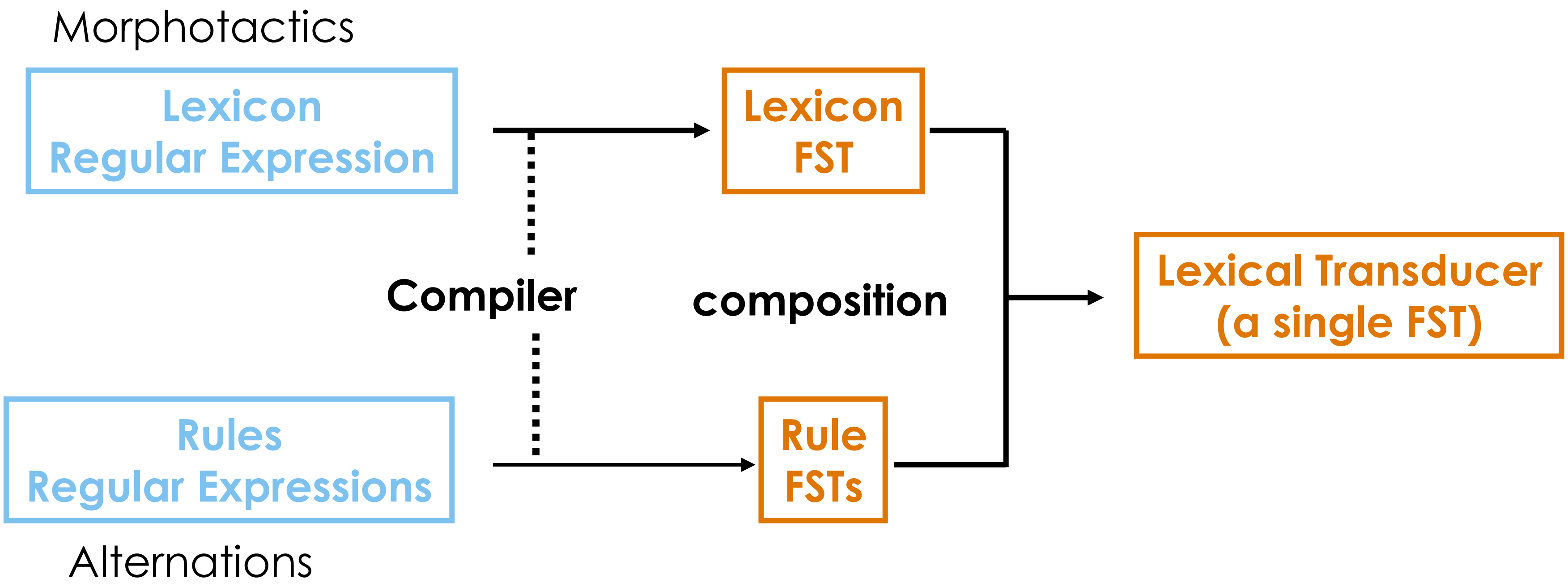
A Convenient Division I

- XFST makes a useful division:
  - **lexc:** constructs finite state transducers from lexical rules defining morphemes and how they can be concatenated.
  - **xfst:** constructs finite state transducers that implement phonological alternations (changes in the segments—vowels and consonants—in a word that come about when morphological operations change their context). The most common phonological alternations occur at morpheme boundaries; however, phonological alternations can also be non-local.
    - /san + pat/ → [sa**m**pat]
    - /dɑg + z/ → [dɑg**z**], /kæt + z/ → [kæt**s**], /hɔɹs + z/ → [hɔɹs**əz**]

A Convenient Division II

- The FSTs produced by lexc and xfst have the same data formats and can be straightforwardly composed, etc.
- lexc is designed to define lexicons and the relationship between **morphemes** and their **properties**.
- FSTs can do most, but not quite all, of the things the phonological rules traditionally used by linguists can do. **Specialty of xfst**.
- **This distinction may be useful in future tools for finite state morphology** even though it is not algorithmic in nature.

How Lexical Transducers are Made



Morphotactics

Lexicon
Regular Expression

Lexicon
FST

Compiler

composition

Lexical Transducer
(a single FST)

Rules
Regular Expressions

Rule
FSTs

Alternations

Phonological Rule Ordering

- Traditional phonological rules, in a formalism proposed by Morris Halle and Noam Chomsky, are ordered rewrite rules.
- **a → b / X_Y** (where **a** is rewritten as **b** between strings **X** and **Y**)
- **Orderings and interactions:**
  - The way rules are ordered relative to other interacting rules produces interesting patterns.
  - Rules can also interact with themselves, assuming that they apply either RTL or LTR instead of simultaneously.

Kinds of Interactions I

- **Feeding**

  Rule_i creates environments in which Rule_{i+n} can apply.

- **Bleeding**

  Rule_i destroys environments in which Rule_{i+n} could apply.

- These interactions are very common in natural language.

Kinds of Interactions II

- **Counter-feeding**

  Rule_{i+n} would create environments in which Rule_i would apply if Rule_{i+n} were ordered before Rule_i.

- **Counter-bleeding**

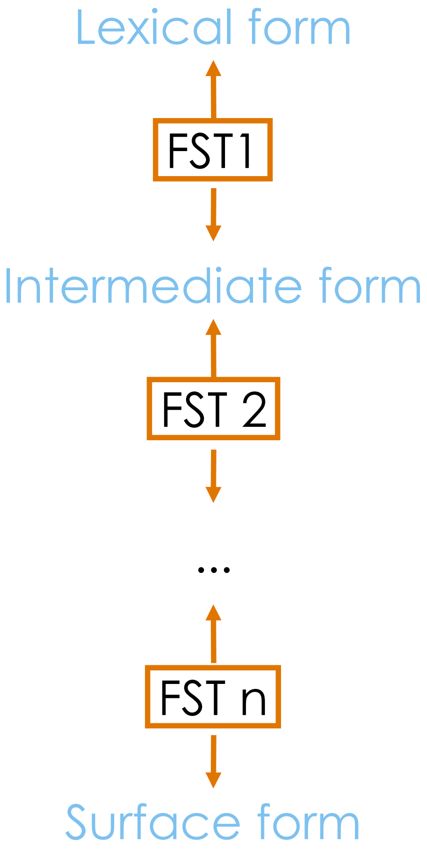
  Rule_{i+n} would destroy environments in which Rule_i would apply if Rule_{i+n} were ordered before Rule_i.

- What kind(s) of interaction that is/are possible with the Chomsky and Halle formalism can't be modeled via the composition FSTs?

Self-Interactions

- Interactions between two or more rules are always possible via composition of FSTs.
- However, self-interactions where a rule feeds or bleeds itself cannot be implemented with FSTs.
- Fortunately, relatively few phonological alternations are implemented using self-feeding (and even fewer, using self-bleeding).
  - **Spreading vowel harmony**. When vowels change to be more like neighboring vowels. This can apply iteratively.
  - **Tone spreading**. When tones change to match neighboring tones. This can also apply iteratively.

Ordered Rules Modeled as Cascade of FSTs



Lexical form

FST1

Intermediate form

FST 2

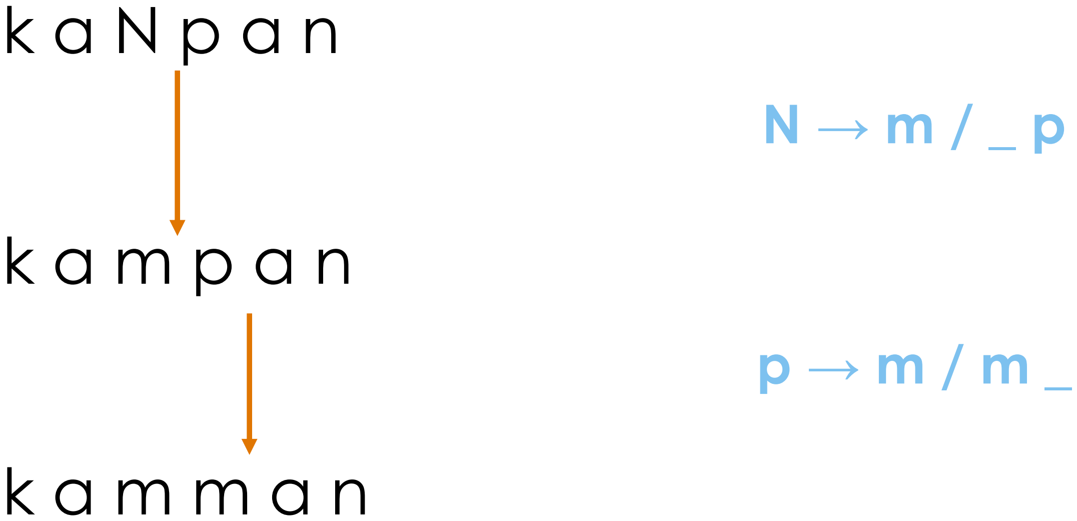
…

FST n

Surface form

Ordered sequence
of rewrite rules
(Chomsky & Halle 1968)
can be modeled
by a cascade of
finite-state transducers
(Johnson 1972;
Kaplan & Kay 1981)

Sequential Application



kaNpan

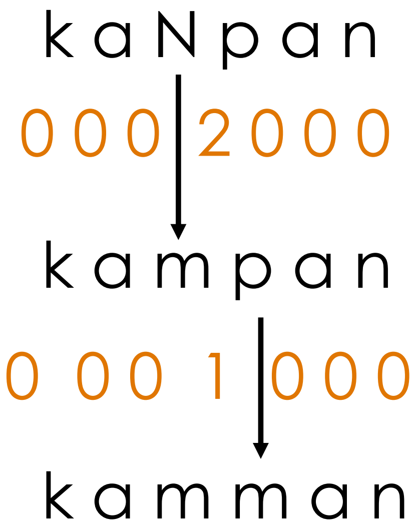
N → m / _ p

kampan

p → m / m _

kamman

Sequential Application in Detail



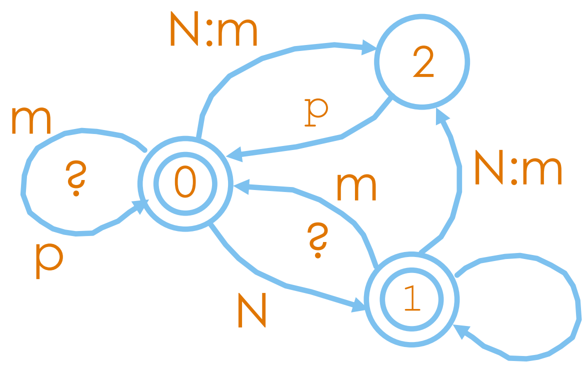
kaNpan

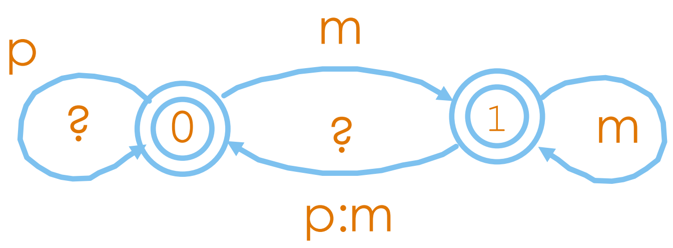
000 2000

kampan

000 1 000

kamman



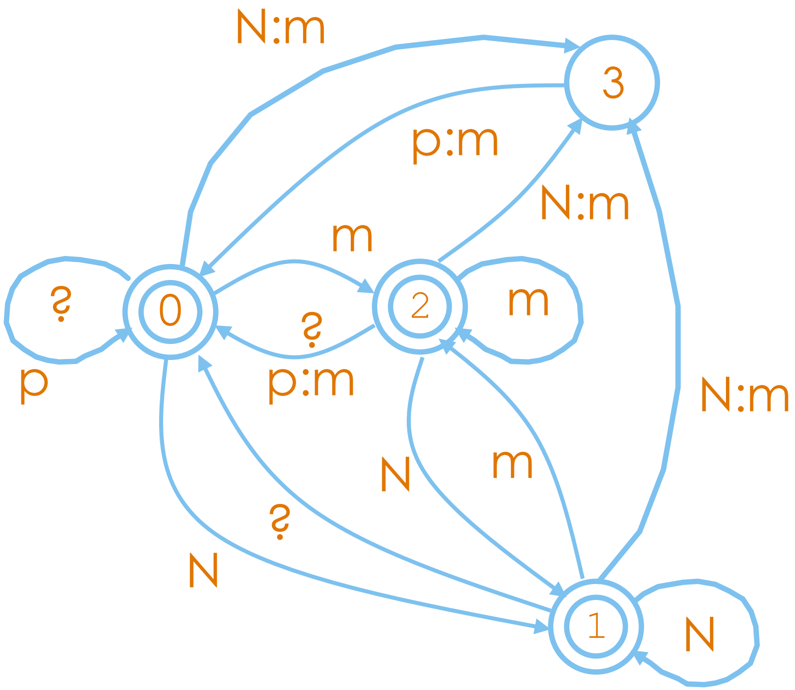


Composition

kaNpan

0 0 0 3 0 0 0

kamman



A Cheat: Flag Diacritics

- Sometimes, it is easier to write FSTs for morphology when the FST formalism is enhanced with **flag diacritics**.
- These essentially add memory to finite state machines.
- They are allowed in **XFST**, **Foma**, and **HSFT**. Elsewhere, too.
  - If you look at XFST (lexc) scripts, you will see strings like @U.CASE.ACC@, with alphanumeric characters and full stops enclosed in "@" characters.
  - This is a unification flag; there are other types of flags.
  - "Traversing an arc labeled with a Flag Diacritic is like an epsilon transition but is conditioned on the success or failure of an OPERATION specified by the Flag Diacritic. The result depends on the state of a feature register that is initialized in the beginning of the analysis or generation and is contiguously updated along each path that is being explored." (Beesley and Karttunen, 2003)

Flag Diacritic Example: Arabic

‘book’	**analysis**
kitaab+u	definite nominative
kitaab+a	definite accusative
kitaab+i	definite genitive
kitaab+uN	indefinite nominative
kitaab+aN	indefinite accusative
kitaab+iN	indefinite genitive
al+kitaab+u	definite prefix + definite nominative
al+kitaab+a	definite prefix + definite accusative
al+kitaab+i	definite prefix + definite genitive

A Constraint

- The definite prefix *al*- cannot co-occur with the indefinite case markers *-uN*, *-an*, and *-in*.
- It is possible to capture this distribution with pure finite state technology.
- Not terribly fun.
- Instead...
  - Set a feature register using a flag diacritic @U.ART.PRESENT@ when you consume the definite prefix .
  - Insure that indefinite case suffixes cannot be present if the feature register is set accordingly.

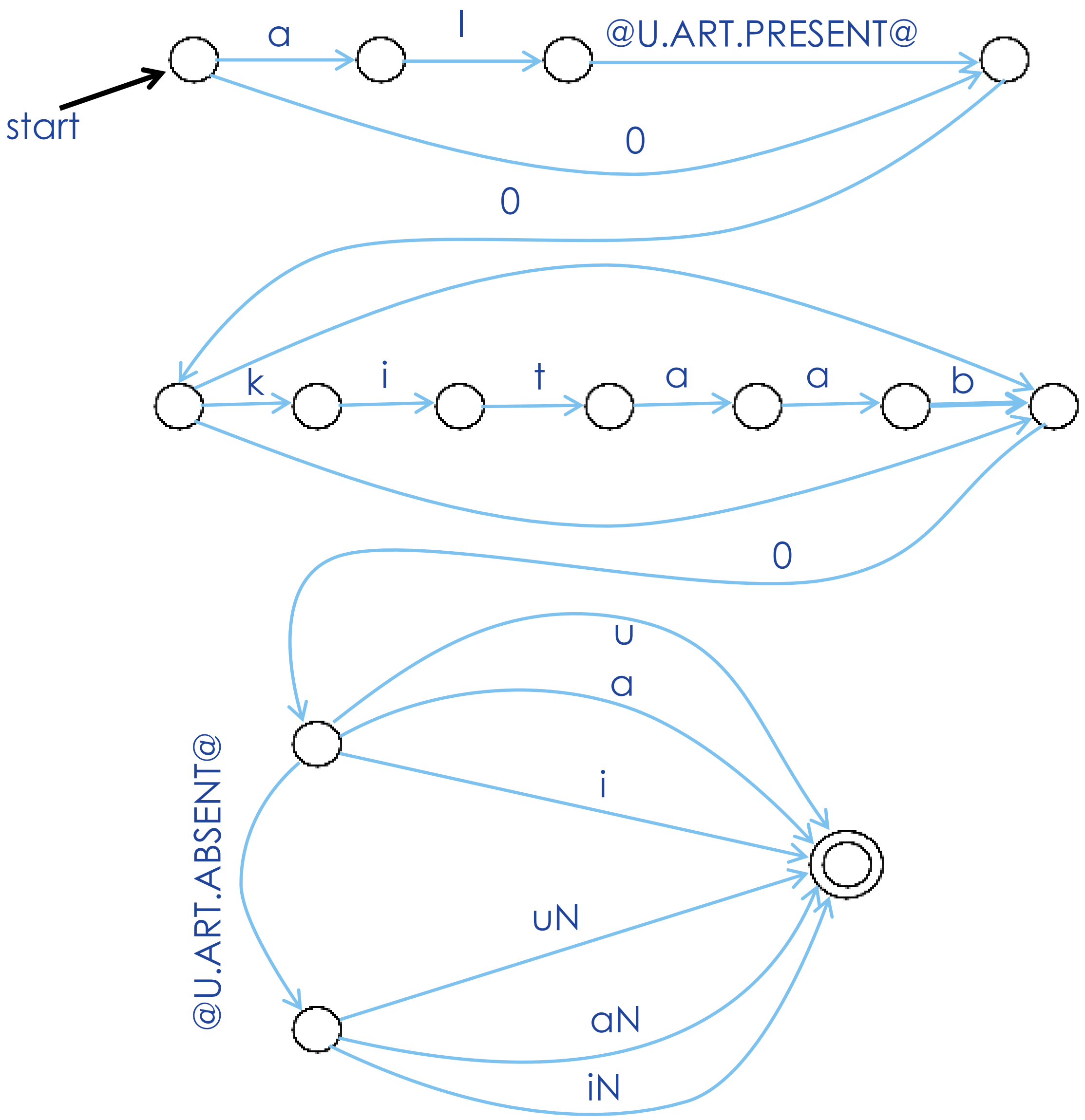
Arabic Definiteness in XFST’s lexc I

```
Multichar_Symbols @U.ART.PRESENTS@ @U.ART.ABSENT@ uN aN iN

LEXICON Root
        Article ;

LEXICON Article
al@U.ART.PRESENT@ Stem ; ! optional article prefix
          Stem ; ! empty string entry

LEXICON Stems
kitaab    Case ; ! one stem to represent > 10,000
```

Arabic Definiteness in XFST’s lexc II

```
LEXICON Case
u        # ;
a        # ;
i        # ;
@U.ART.ABSENT@ IndefCase ;

LEXICON IndefCase
uN       # ;
aN       # ;
iN       # ;
```

Arabic FSM Fragment



Upshot

While flag diacritics are not a fundamental part of the finite state paradigm, they are a useful addition in cases where humans will manually code finite state morphological analyzers or generators. They will be used in our finite state implementations of non-concatenative morphology.

Concatenation is Regular

- The most common kind of morphological operation involves the concatenation of morphemes.
  - Agglutinative and polysynthetic languages use concatenation primarily or exclusively.
  - Concatenation is straightforwardly regular.
  - We can easily implement it with finite state methods.

More Than Mere Concatenation

- **Reduplication.** Repeating all or part of a word as a morphological operation.
- **Infixation.** Inserting an affix into a base.
- **Root-and-pattern morphology.** Interdigitation in Semitic and its relatives.
- Others. **Apophony**, including the umlaut in English *tooth* → *teeth*; **subtractive morphology**, including the **truncation** in English nickname formation (*David* → *Dave*); and so on.

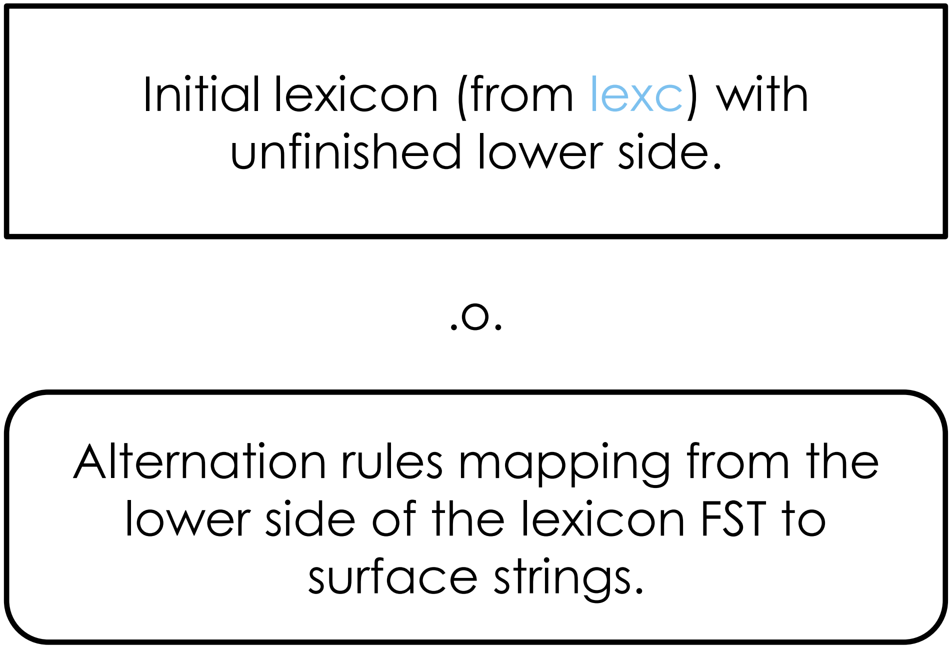
The Problem

- Non-concatenative morphology cannot be addressed with finite state methods without considerable ingenuity.
- However, many languages feature some non-concatenative processes.
  - **Reduplication**—extremely widespread.
  - **Root-and-pattern**—occurs in two very important languages.

A Solution

- Antworth (1990) found a way of capturing reduplication as long as the reduplicant always had the same number of segments (vowels and consonants). **Unsatisfying**.
- Beesley and Karttunen (2000, 2003) proposed the **compile-replace** algorithm, a clever solution that allows purely finite state methods to deal with a wide class of non-concatenative morphology.
- **Summary of compile-replace:**
  - Regex replacements generate new regex, which encodes meta-morphotactic information.
  - New regex is compiled to FSA.
  - FSA describes complete lower side.

Concatenative Morphology Using XSFT: Simplest Case

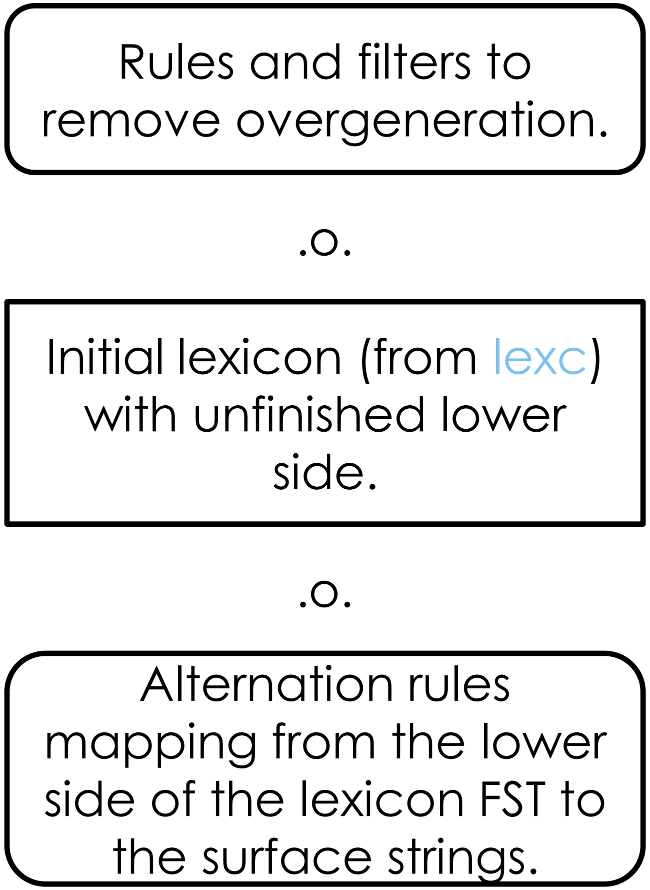


Initial lexicon (from lexc) with unfinished lower side.

.o.

Alternation rules mapping from the lower side of the lexicon FST to surface strings.

Concatenative Morphology Using XFST: Realistic Case



Rules and filters to remove overgeneration.

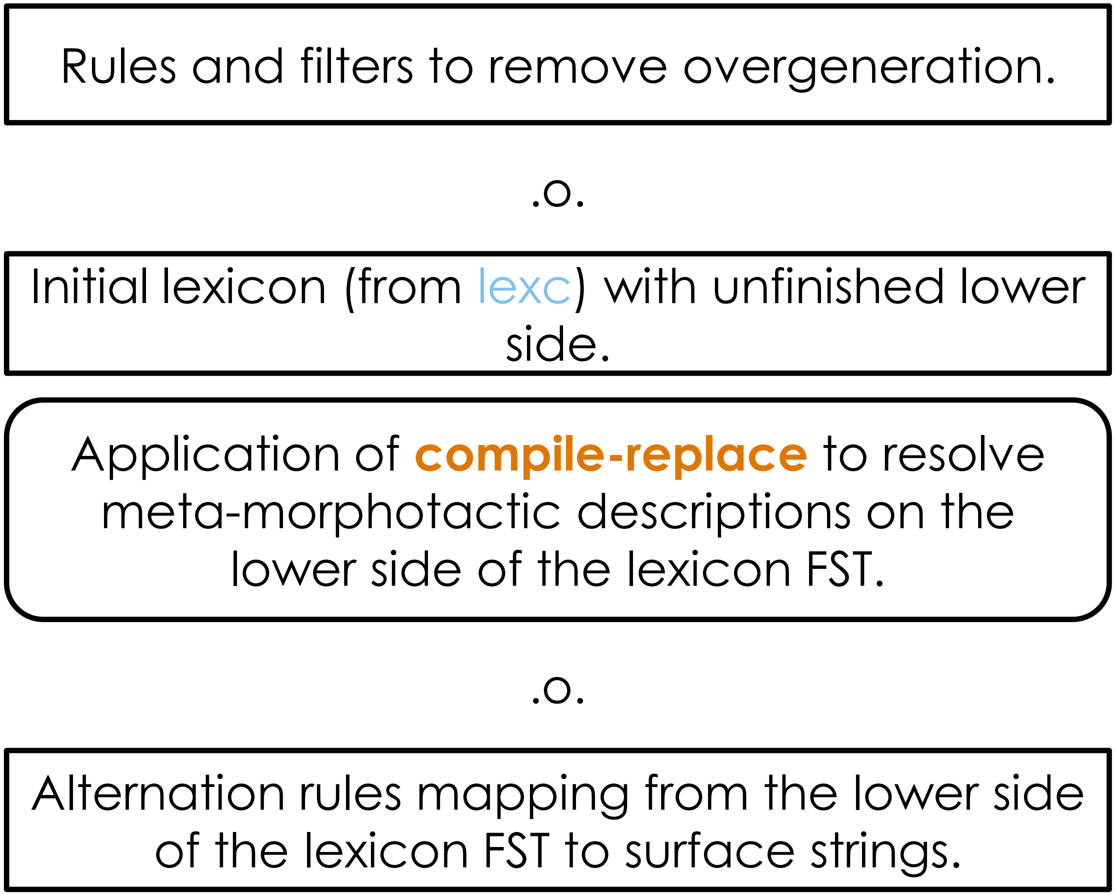
.o.

Initial lexicon (from lexc) with unfinished lower side.

.o.

Alternation rules mapping from the lower side of the lexicon FST to the surface strings.

Adding Compile-Replace to XSFT



Rules and filters to remove overgeneration.

.o.

Initial lexicon (from lexc) with unfinished lower side.

Application of **compile-replace** to resolve meta-morphotactic descriptions on the lower side of the lexicon FST.

.o.

Alternation rules mapping from the lower side of the lexicon FST to surface strings.

Total Reduplication in Malay

Root	Gloss	Reduplication	Gloss
anak	‘child’	*anakanak*	‘children’
lembu	‘cow’	*lembulembu*	‘cows’
buku	‘book’	*bukubuku*	‘books’
basikal	‘bicycle’	*basikalbasikal*	‘bicycles’

Script in lexc for Malay Reduplication

```
Multichar_Symbols ^[ ^] +Noun +Unmarked +Plural

LEXICON Root
0:^[{  NRoots ;
       NRoots ;

LEXICON NRoots
buku      NSuff ;
lembu     NSuff ;
pelabuhan NSuff ;

LEXICON NSuff
+Noun:0    Num ;

LEXICON Num
+Unmarked:0   # ;
+Plural:}^2^] # ;
```

e Script

In retrospect, that was not

Though when all the morphological and phonological alternations in Mal incorporate sim

This is not trivial but is nevertheless left as an exercise to the viewer. See, also, Beesley and Karttunen (2003).

Upper
Lower

Upper
Lower

Upper
Lower:

But What about Arabic?

Pattern:		a		a	
Template:	C	V	C	V	C
Root:					

How would you model this kind of pattern using the **compile-replace** algorithm (or some other finite state method)?

One Solution (Beesley & Karttunen, 2003)

- Start with a network that constructs regexes that, when compiled, "merges" roots and patterns into templates.
- ^[{ktb}.m>.{CVCVC}.<m.[a+]^]
  - .m>. rightward merge
  - .<m. leftward merge
- Only works if C and V are defined as "super-symbols":
  - list C b t y k l m n f w r z d s
  - list V a i u
- .m>. "spreads" k, t, and b LTR across the Cs (which match).
- .<m. "spreads" a RTL across the Vs (which match).

More Compile-Reduce

- Just as with Malay total reduplication, **compile-replace** comes to the rescue.
  - Network generates a regular expression as it consumes the upper side.
  - **Compile-replace** compiles all (or part) of this expression.
  - The resulting FSA defines a language consisting of one word.

I Like to Merge It, Merge It

```
Multichar_Symbols ^[ ^] +3P +Masc +Fem +Sg +Act +Pass +FormI +FormIII

LEXICON Root
        LBound ;

LEXICON LBound
[:^[{   Roots ;

LEXICON Roots
ktb     MergeRight ;
drs     MergeRight ;

LEXICON MergeRight
0:}.m%>.{ Template ; ! % literalizes the >
```

I Like to Merge It, Merge It

```
LEXICON Template
+FormI:CVCVC   MergeLeft ;
+FormII:CVVCVC MergeLeft ;

LEXICON MergeLeft
0:}.%<m.  Vocalization ; ! % literalizes the <

LEXICON Vocalization
+Act:[a+]   RBound ;
+Pass:[u*i] RBound ;

LEXICON RBound
]:^] PerfEnd ;

LEXICON PerfEnd
+3P+Masc+Sg:a # ;
+3P+Fem+Sg:at # ;
```

Conclusion

- Finite state methods provide a simple and powerful means of generating and analyzing words (as well as the phonological alternations that accompany word formation/inflection).
- Straightforward concatenative morphology is easy to implement using finite state methods.
- Other phenomena are easiest to capture with extensions to the finite state paradigm.
  - Co-occurrence restrictions—flag diacritics.
  - Non-concatenative morphology—compile-replace algorithm. Pure finite state, but computed in a novel fashion.