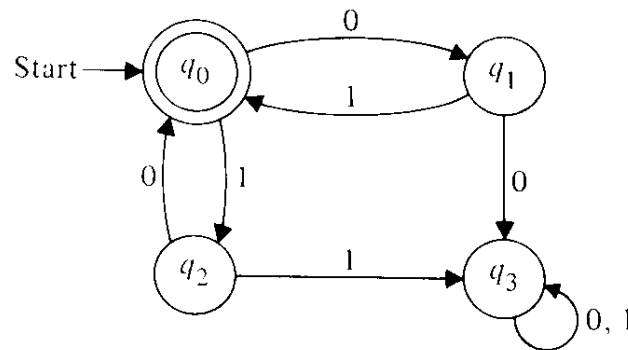


CS 562: Empirical Methods in Natural Language Processing

Unit 1: Sequence Models

Lecture 3: Finite-State Acceptors

Lecture 4: Finite-State Transducers



Week 2 -- Sep 1 & 3, 2009

Liang Huang (lihuang@isi.edu)

This Week: Finite-State Machines

- Finite-State Acceptors and Languages
 - DFAs (deterministic)
 - NFAs (non-deterministic)
- Finite-State Transducers
- Applications in Language Processing
 - part-of-speech tagging, morphology, text-to-sound
 - word alignment (machine translation)
- HW I (out on Th, due next Th): FSMs with carmel
- Next Week: putting probabilities into FSMs

Languages and Machines

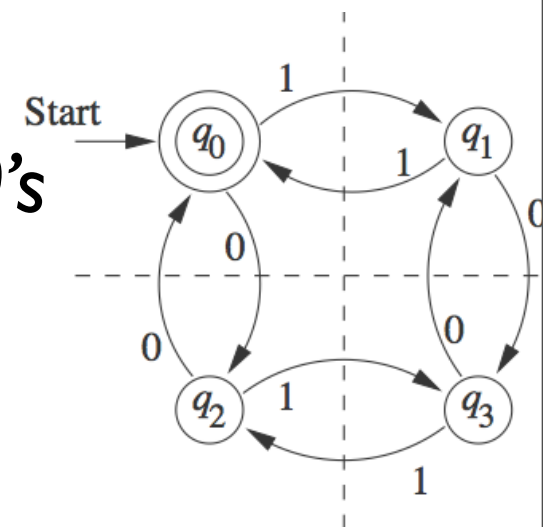
- Q1: how to formally define a *language*?
- a language is a **set of strings**
 - could be finite, but often infinite (due to recursion)
 - $L = \{ aa, ab, ac, \dots, ba, bb, \dots, zz \}$ (finite)
 - English is the set of *grammatical English sentences*
 - variable names in C is set of alphanumeric strings
- Q2: how to *describe* a (possibly infinite) language?
 - use a finite (but recursive) representation
 - finite-state acceptors (FSAs) or regular-expressions

Finite-State Acceptors

- $L1 = \{ aa, ab, ac, \dots, ba, bb, \dots, zz \}$ (finite)
 - start state, final states
- $L2 = \{ \text{all letter sequences} \}$ (infinite)
 - recursion (cycle)
- $L3 = \{ \text{all alphanumeric strings} \}$

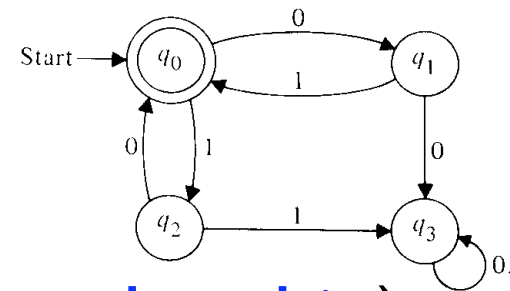
More Examples

- $L4 = \{ \text{all letter strings with at least a vowel} \}$
- $L5 = \{ \text{all letter strings with vowels in order} \}$
- $L6 = \{ \text{all } 01 \text{ strings with even number of } 0\text{'s} \text{ and even number of } 1\text{'s} \}$



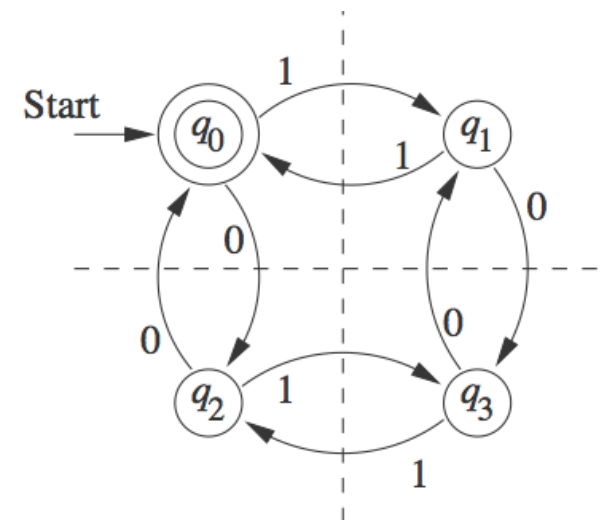
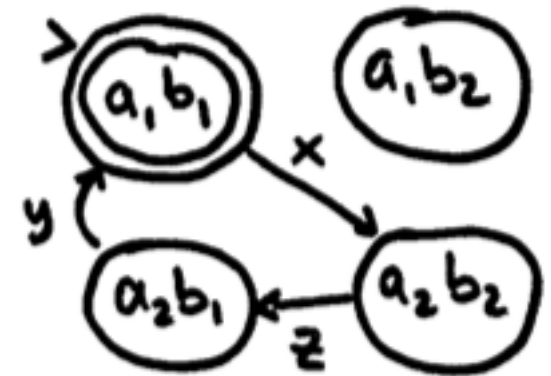
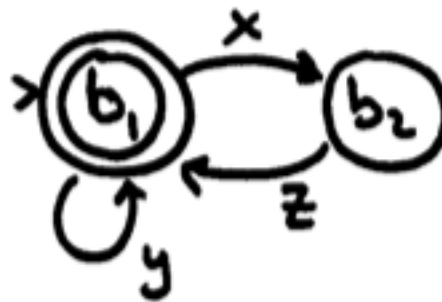
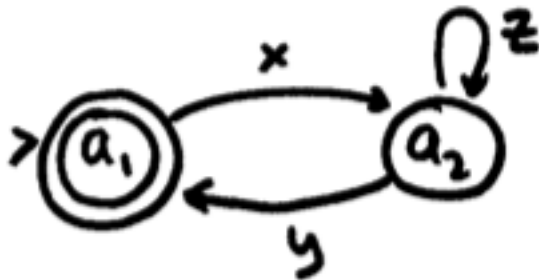
Membership and Complement

- deterministic FSA: iff no state has two exiting transitions with the same label. (**DFA**)
- the language L of a DFA D : $L = L(D)$
- how to check if a string w is in $L(D)$? (**membership**)
 - linear-time: follow transitions, check finality at the end
 - no transition for a char means “into a trap state”
- how to construct **complement DFA**? $L(D') = \neg L(D)$
 - super easy: just reverse the finality of states :)
 - note that “trap states” also become final states



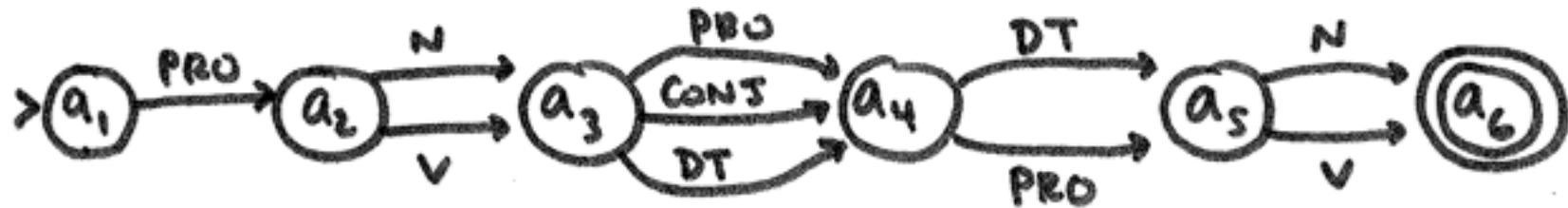
Intersection

- construct D s.t. $L(D) = L(D_1) \cap L(D_2)$
- state-pair (“cross-product”) construction
- intersected DFA: $|Q| = |Q_1| \times |Q_2|$

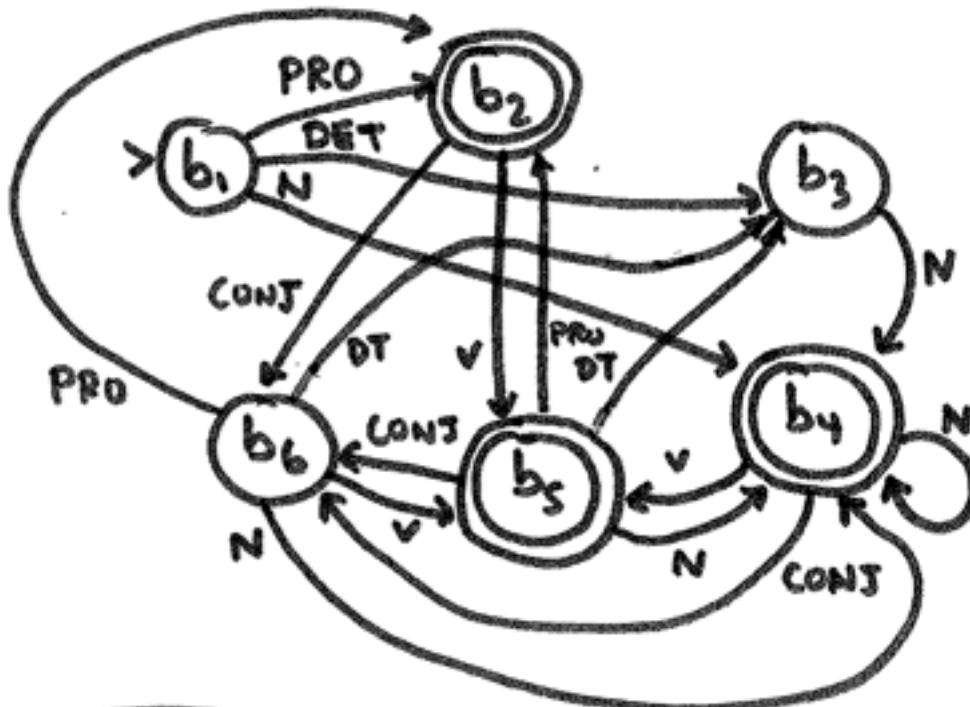


Linguistic Example

- DFA A: all interpretations of “he hopes that this works”



- DFA B: all legal English category sequences (simplified)

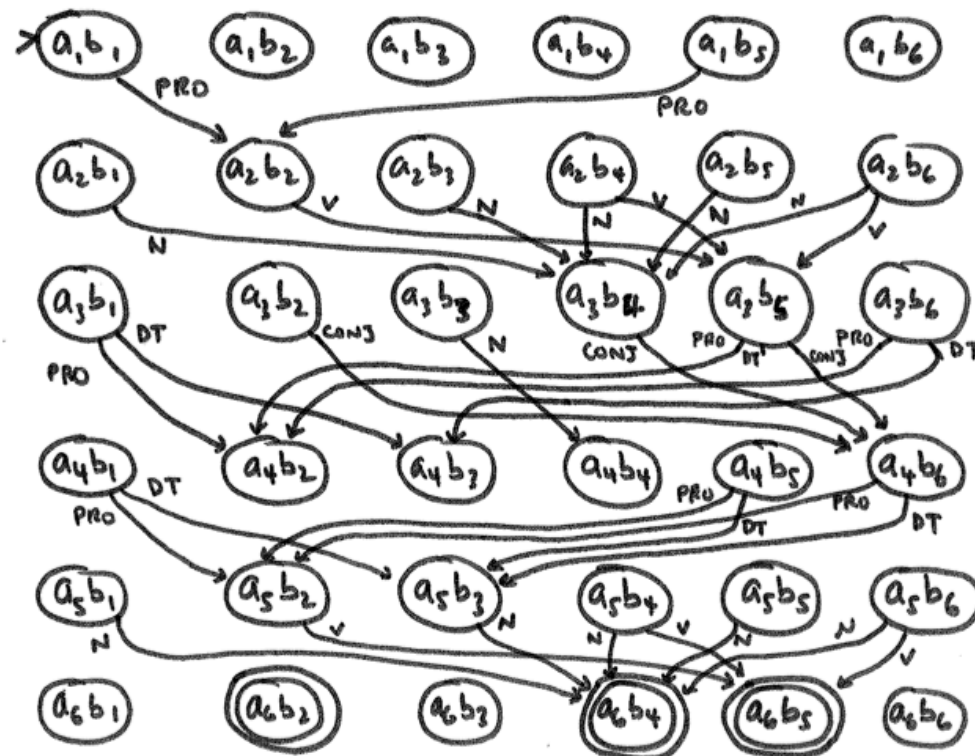
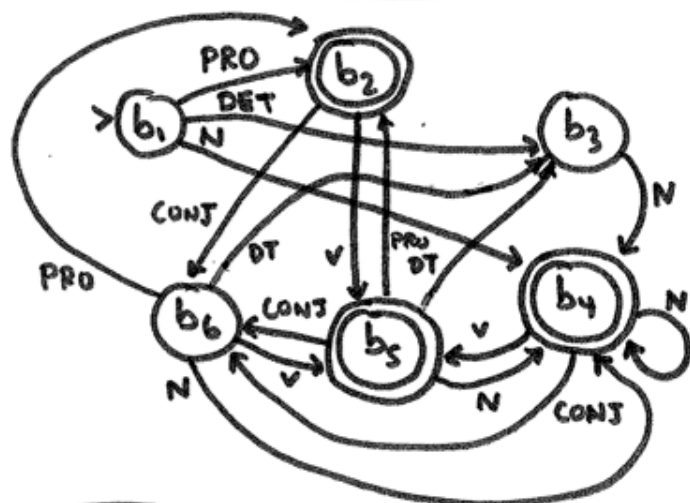
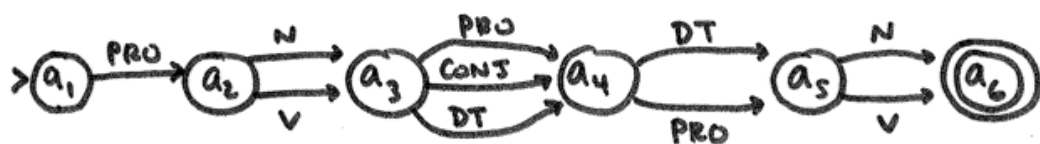


what do these states mean?

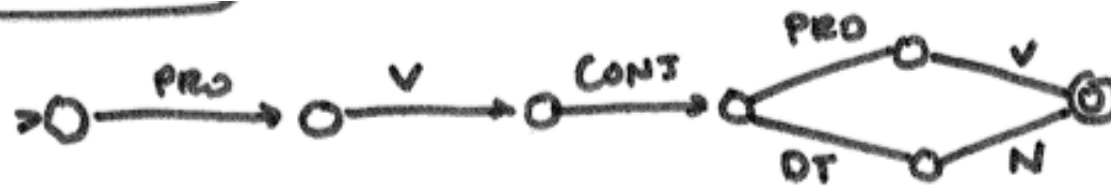
what will $A \cap B$ mean?

Linguistic Example

- intersection by state-pair (“product”) construction



- cleanup: he hopes that this works



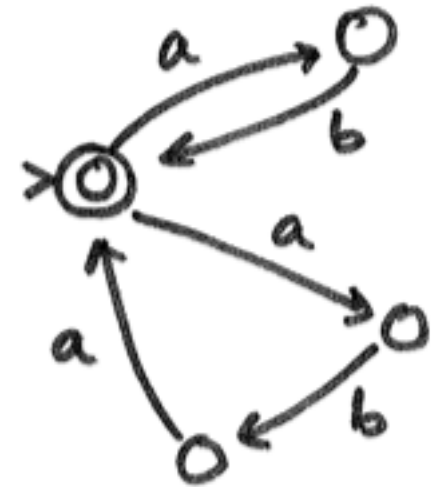
- this is **part-of-speech tagging!** (with a bigram model)

Union

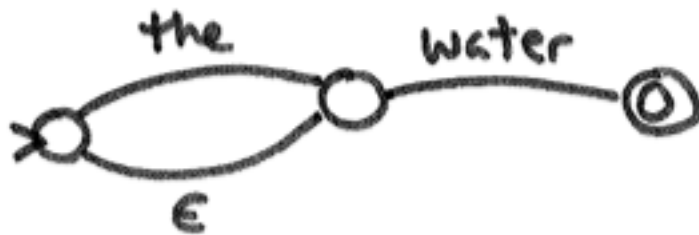
- easy, via De Morgan's Law: $L_1 \cup L_2 = \neg (\neg L_1 \cap \neg L_2)$
- or, directly, from the product construction again
- what are the final states?
 - could end in either language: $Q_2 \times F_1 \cup Q_1 \times F_2$
 - same De Morgan: $\neg ((Q_1 \setminus F_1) \cap (Q_2 \setminus F_2)) = \neg (\neg F_1 \cap \neg F_2)$

Non-Deterministic FSAs

- $L = \{ \text{all strings of repeated instances of } ab \text{ or } aba \}$
 - hard to do with a deterministic FSA!
 - e.g., abababaababa



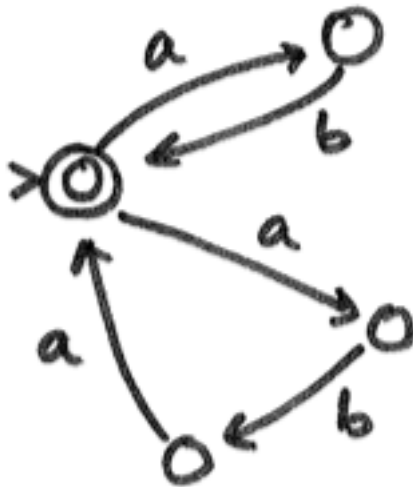
- epsilon transition (no symbol)



- there is algorithm to determinize a DFA
 - blow up the state-space exponentially

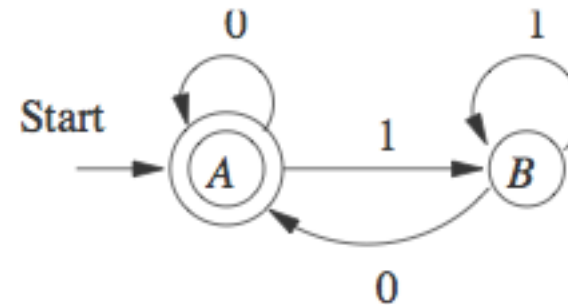
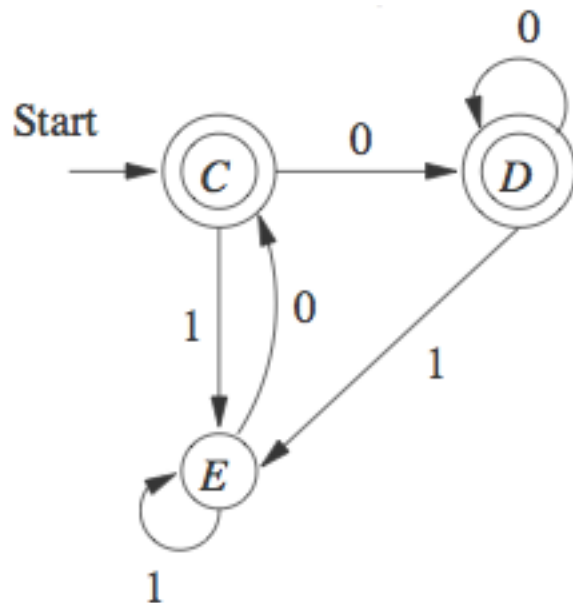
Determinization Example

- determinization by subset construction (2^n)



Minimization and Equivalence

- each DFA (and NFA) can be reduced to an equivalent DFA with minimal number of states
 - based on “state-pair equivalence test”
 - can be used to test the equivalence of DFAs/NFAs

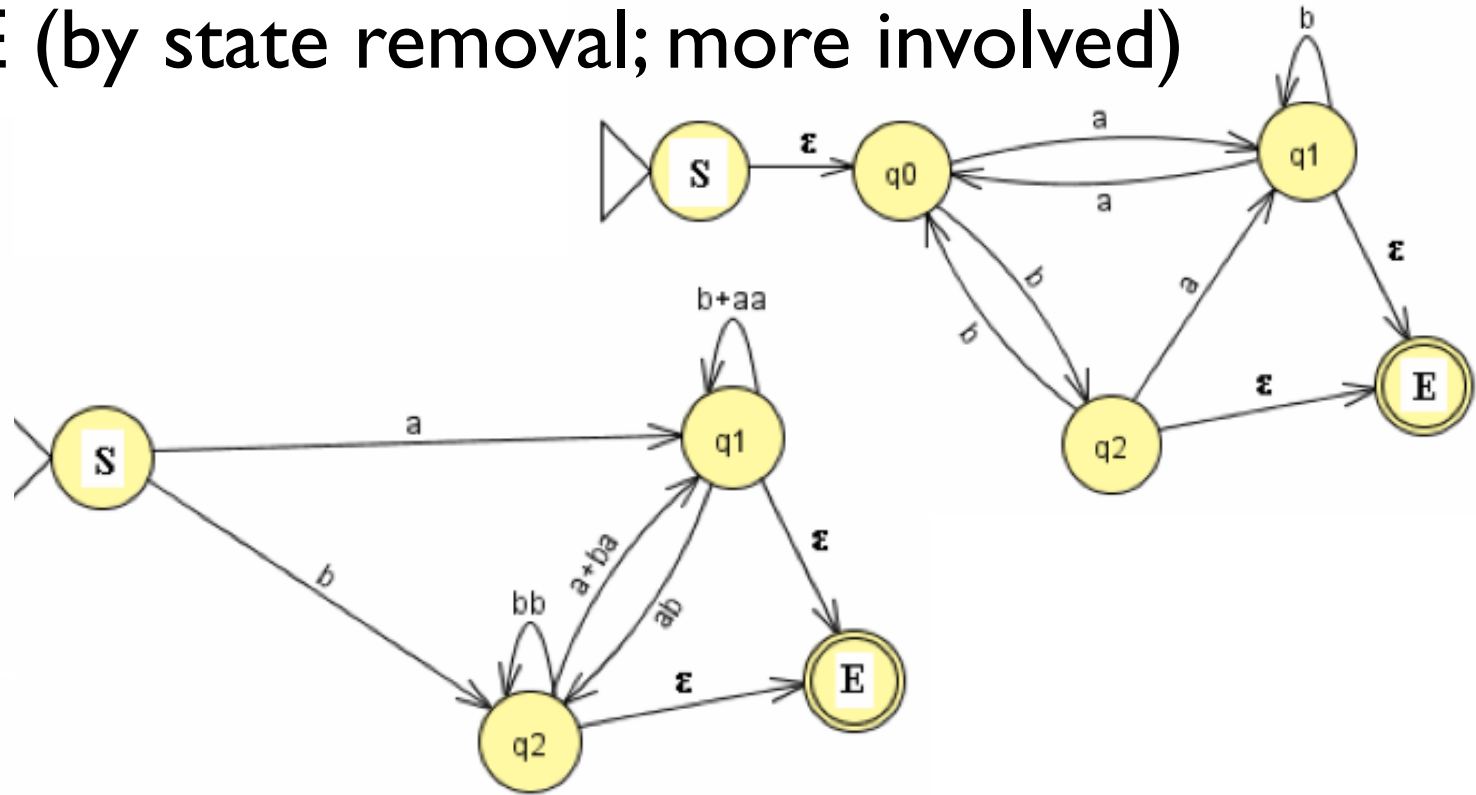
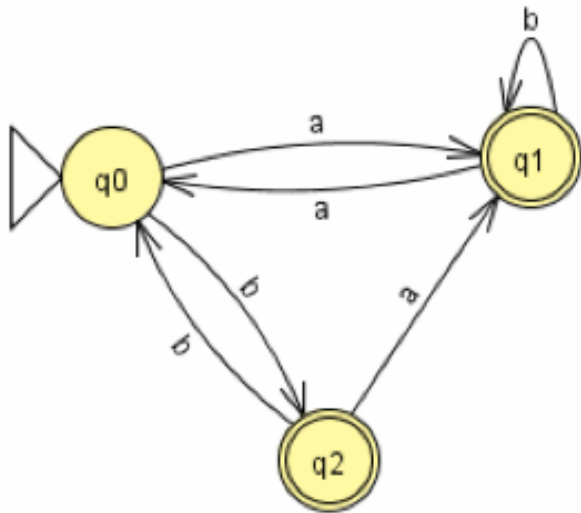


Advantages of Non-Determinism

- union (and intersection also?)
- concatenation: $L_1L_2 = \{ xy \mid x \text{ in } L_1, y \text{ in } L_2 \}$
- membership problem
 - much harder: exp. time \Rightarrow rather determinize first
- complement problem (similarly harder)
- but is NFA more expressive than DFA?
 - NO, because you can always determinize an NFA
- NFA: more “intuitive” representation of a language
- mDFA: “compact (but less intuitive) encoding”

FSA vs. Regular Expressions

- RE operators: R^* , R_1+R_2 , R_1R_2
- RE \Rightarrow NFA (by recursive translation; easy)
- NFA \Rightarrow RE (by state removal; more involved)



- RE \Leftrightarrow NFA \Leftrightarrow DFA \Leftrightarrow mDFA

Wrap-up

- machineries: (infinite) languages, DFAs, NFAs, REs
 - why and when non-determinism is useful
- constructions/algorithms
 - state-pair construction: intersection and union
 - quadratic time/space
 - subset construction: determinization
 - exponential time/space
 - briefly mentioned: minimization and $RE \Leftrightarrow NFA$
 - see Hopcroft et al textbook for details

Quick Review

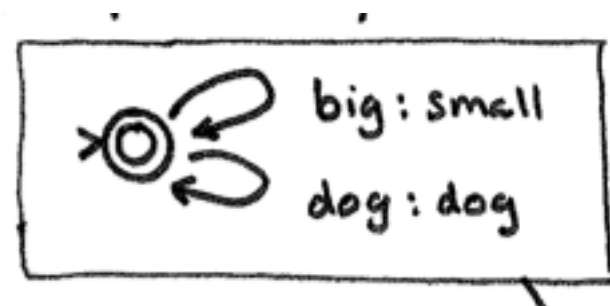
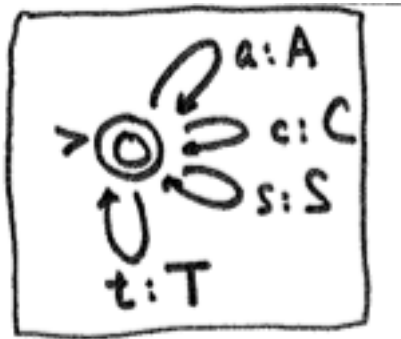
- how to detect if a DFA accepts any string at all?
 - how about empty string?
 - how about all strings?
- how about an NFA?
- how to design a *reversal* of a DFA/NFA?

Finite-State Transducers

- FSAs are “acceptors” (set of strings as a language)
- FSTs are “converters”
 - compactly encoding set of string pairs as a relation
- capitalizer: { <cat, CAT>, <dog, DOG>, ... }
- pluralizer: { <cat, cats>, <fly, flies>, <hero, heroes>... }

Formal Definition

- a finite-state transducer T is a tuple $(Q, \Sigma, \Gamma, I, F, \delta)$ such that:
 - Q is a finite set, the set of *states*;
 - Σ is a finite set, called the *input alphabet*;
 - Γ is a finite set, called the *output alphabet*;
 - I is a subset of Q , the set of *initial states*;
 - F is a subset of Q , the set of *final states*; and
 - $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q$ is the *transition relation*.

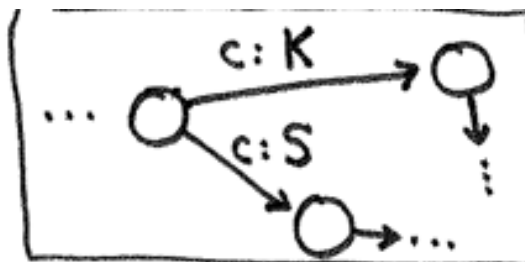


Examples

- text-to-sound: {<cat, K A E T>, <dog, D A W G>, <bear, B E H R>, <bare, B E H R>...}
- (easy for Spanish/Italian, medium for French, hard for English!)
- POS tagger: {<I saw the cat, PRO V DT N>, ...}
- transliterator: { <bus h, 布 什>, <obama, 奥 巴 马>, ... }
bu shi *ao ba ma*
- translator: { <he is in the house, el está en la casa>, <he is in the house, está en la casa>, ... }
- notice the many-to-many relation (not a function)
- but is this real translation? NO, there are no reorderings!
- FSMs are best for morphology; we need CFGs for syntax

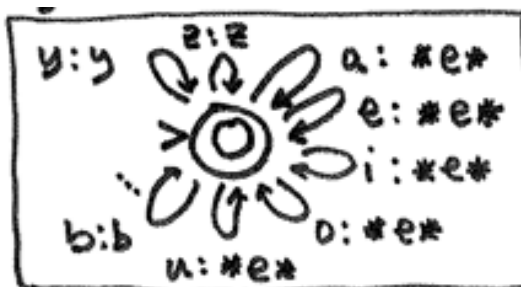
Non-Determinism in FSTs

- ambiguity



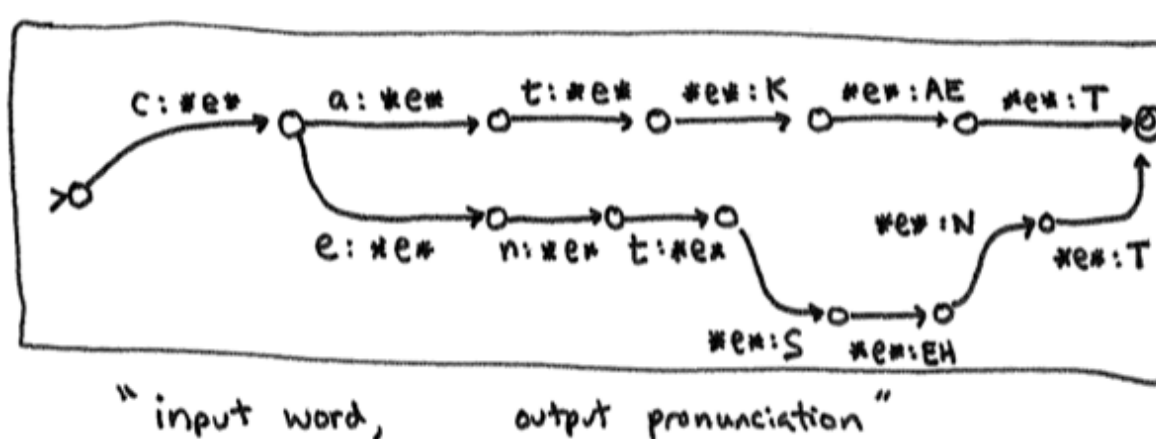
character "c"
pronounced as either
K sound or S sound

- optionality

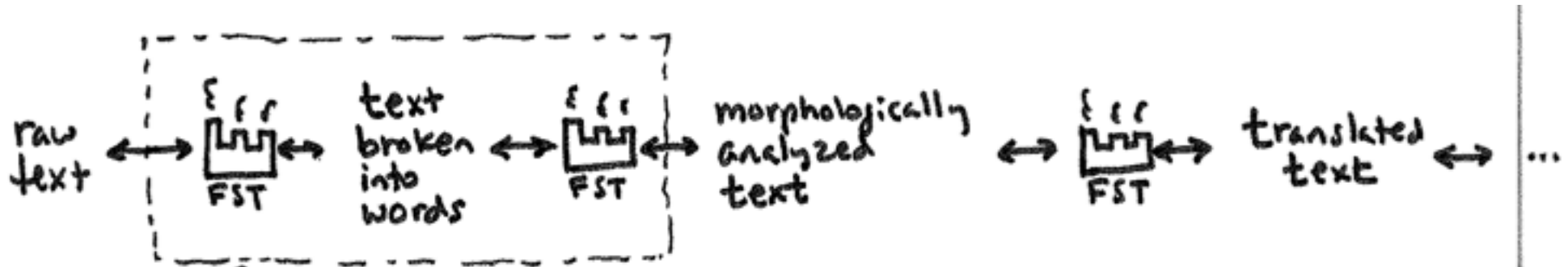


- important because in/out often have different lengths

- delayed decision via epsilon transition



Central Operation: Composition

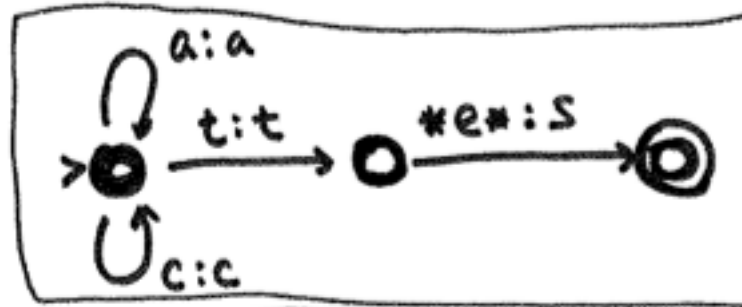


- language processing is often in cascades
 - often easier to tackle small problems separately
- each step: $T(A)$ is the relation (set of string pairs) by A
 - $\langle x, y \rangle$ in $T(A)$ means $x \sim_A y$
- compose $(A, B) = C$
 - $\langle x, y \rangle$ in $T(C)$ iff. $\exists z: \langle x, z \rangle$ in $T(A)$ and $\langle z, y \rangle$ in $T(B)$

Simple Example

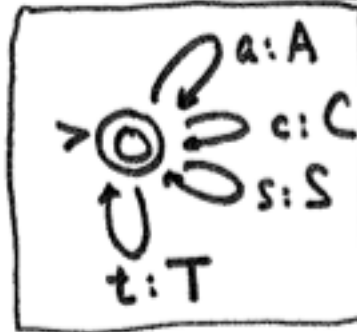
- pluralizer + capitalizer

FST A pluralizes :



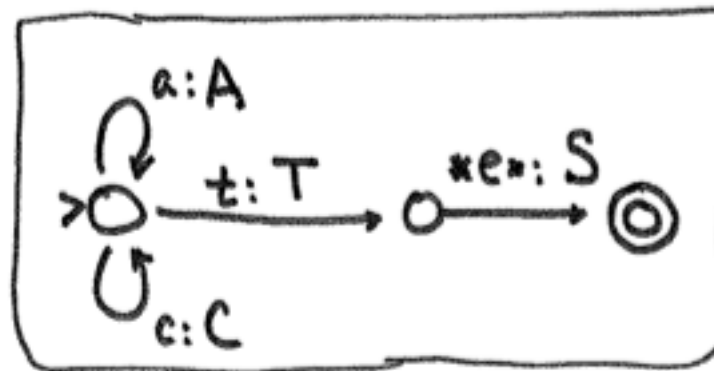
<act,	acts>
<cat,	cats>
<aat,	aats>
⋮	⋮
	}

FST B capitalizes :

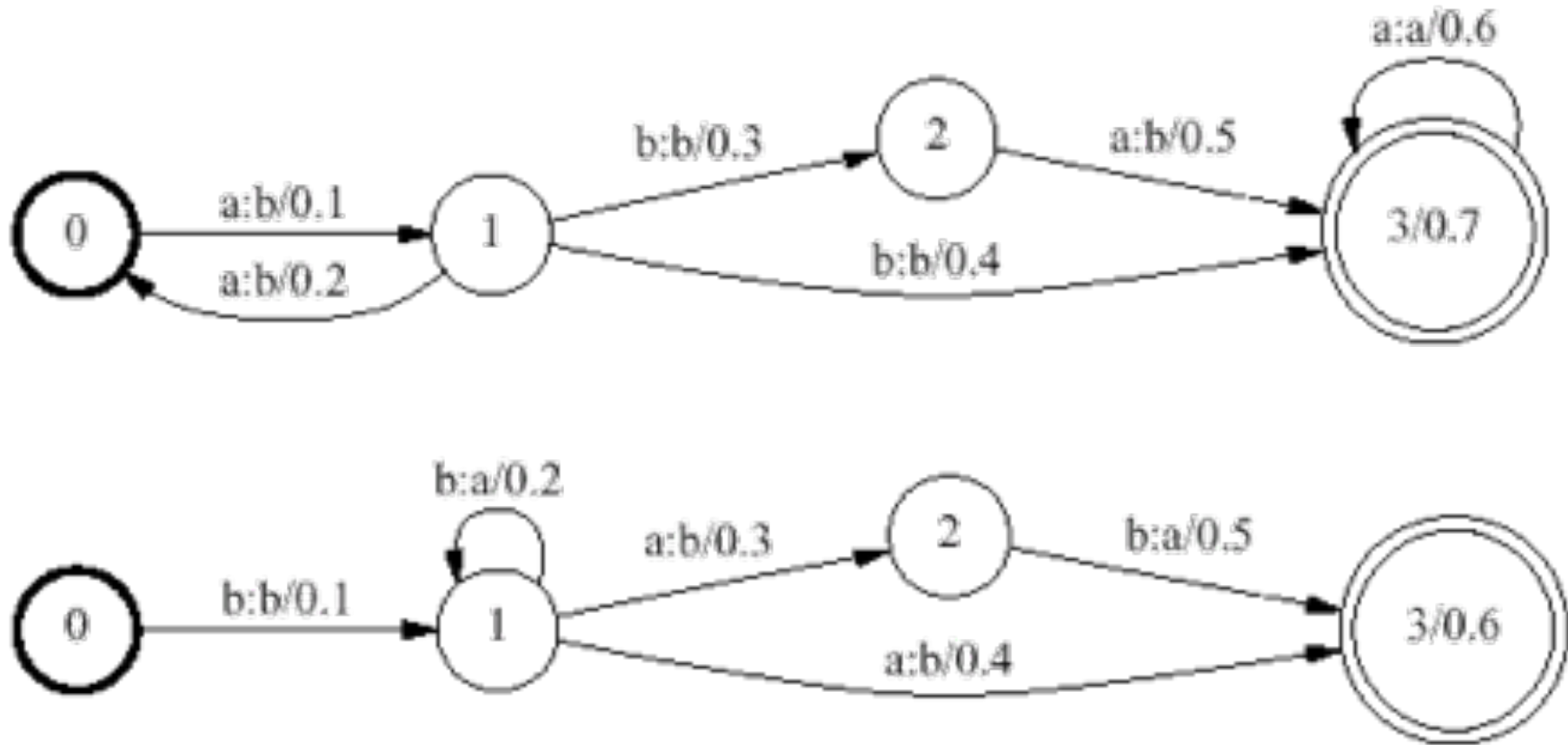


<act,	ACT>
<ccc,	CCC>
<acts,	ACTS>
⋮	⋮
	}

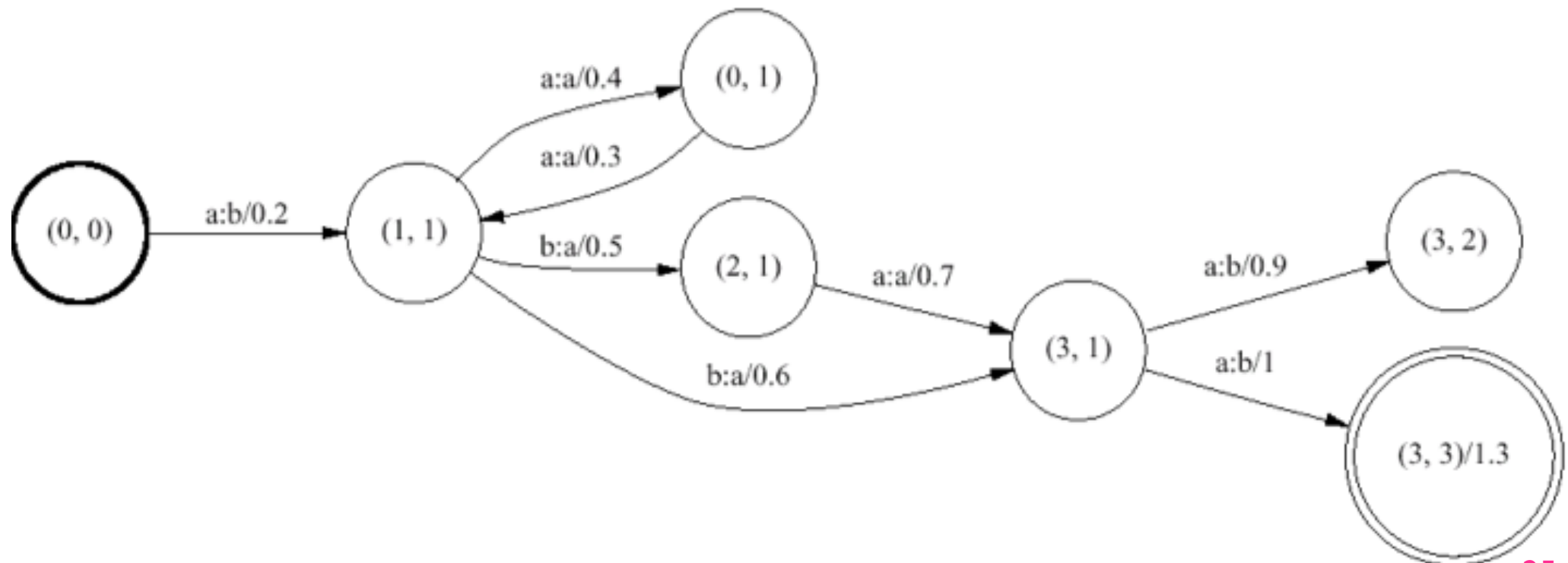
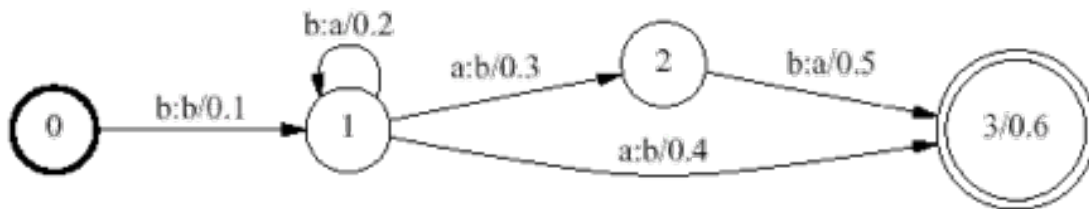
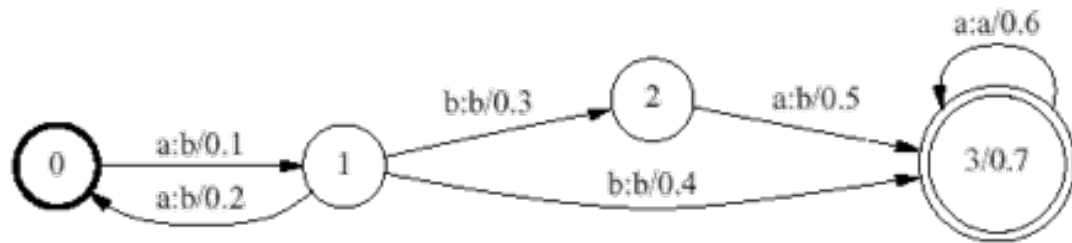
FST Compose(A, B)
does both:



How to do composition?



How to do composition?

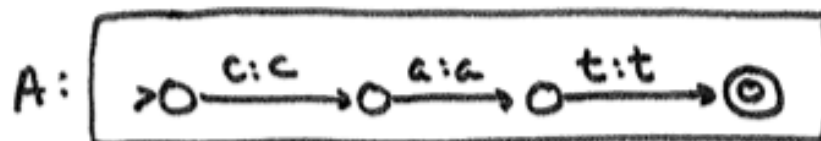


composition is like intersection?

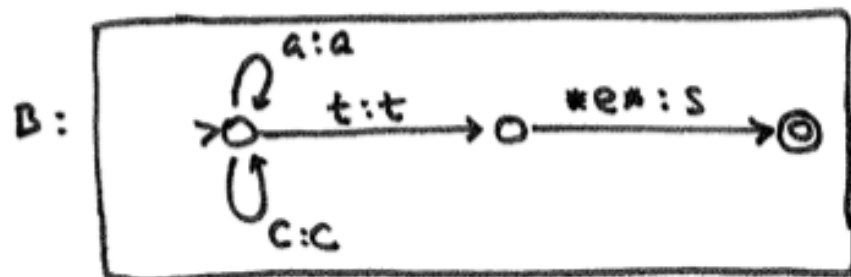
- both use cross-product (“state-pair”) construction
- indeed: intersection is a special case of composition
 - FSA is a special FST with identity output! ($a \Rightarrow a:a$)
 - $A \cap B = \text{proj}_{\text{in}} (\text{Id}(A) \diamond \text{Id}(B))$
- what about FSAs composed with FSTs?
 - FSA \diamond FST --- get output(s) from certain input(s)
 - $\langle x, z \rangle: \exists y \text{ s.t. } \langle x, y \rangle \text{ in } T(\text{Id}(A)) \text{ and } \langle y, z \rangle \text{ in } T(B)$
 - but $y=x \Rightarrow \langle x, z \rangle: x \text{ in } L(A) \text{ and } \langle x, z \rangle \text{ in } T(B)$
 - FST \diamond FSA --- get input(s) for certain output(s)

Get Output

e.g., pluralize "cat"

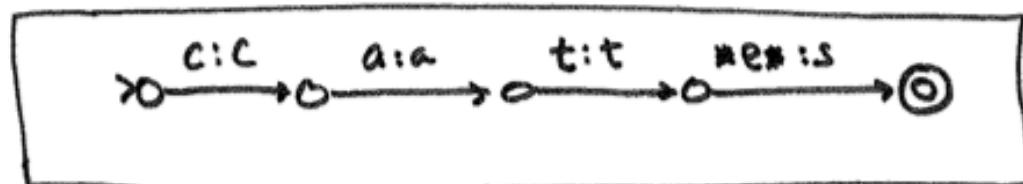


{ <cat, cat> }



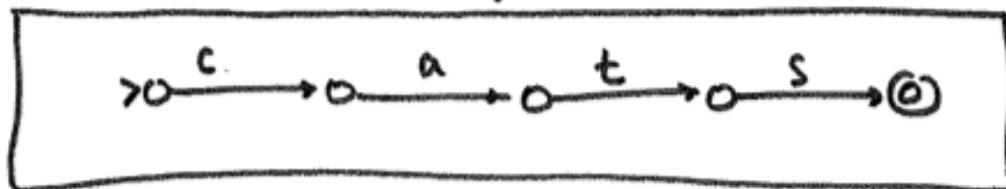
{ <cat, cats>,
<act, acts>,
... }

Compose (A, B) includes $\langle x, y \rangle$ if $\exists z: \langle x, z \rangle \in A$ & $\langle z, y \rangle \in B$



FST { <cat, cats> }

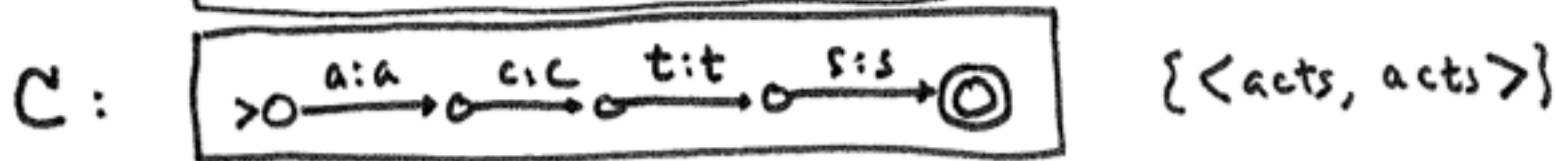
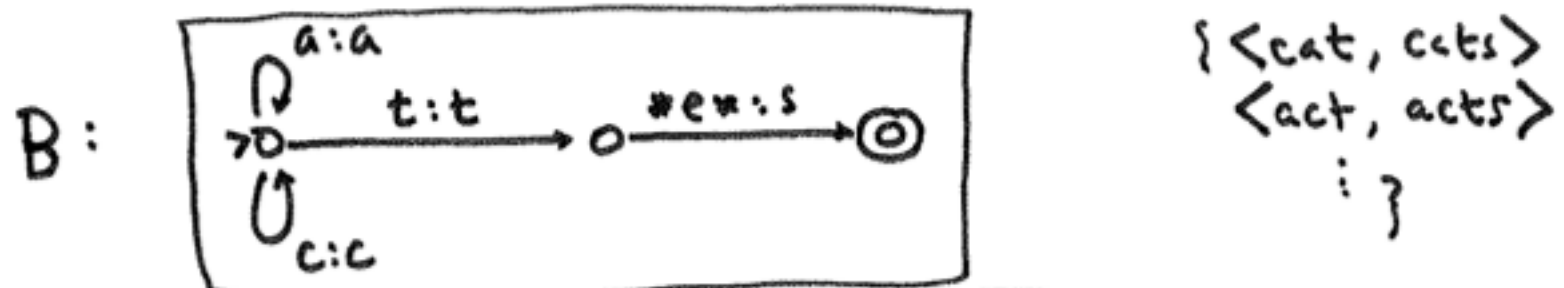
throw away input labels



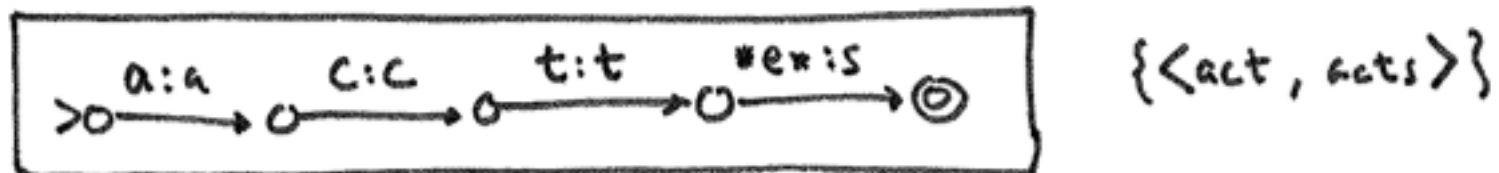
FSA { cats }

Get Input

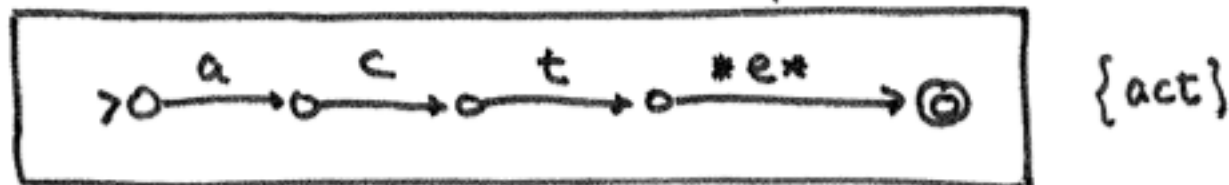
- morphological analysis (e.g. what is “acts” made from)



Compose(B, C)

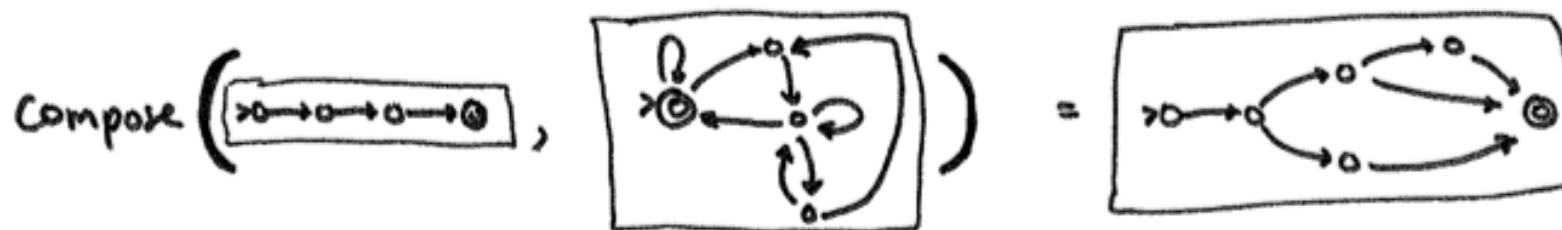


↓ throw away output labels

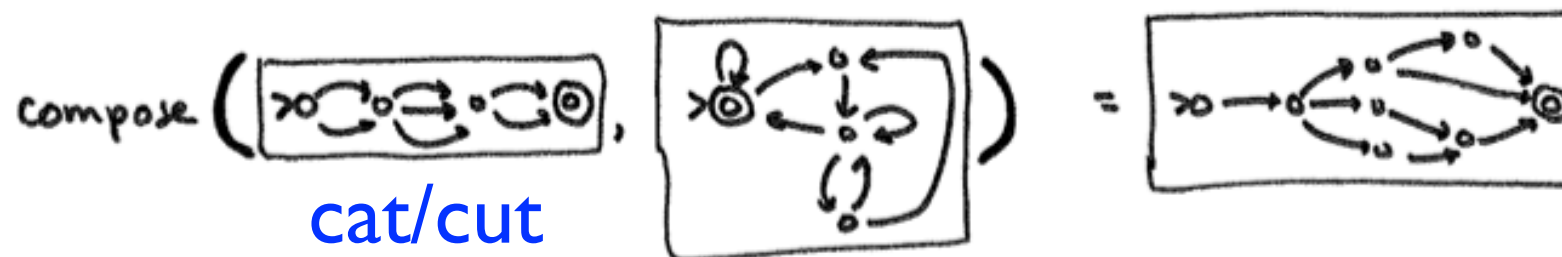


Multiple Outputs

Multiple outputs



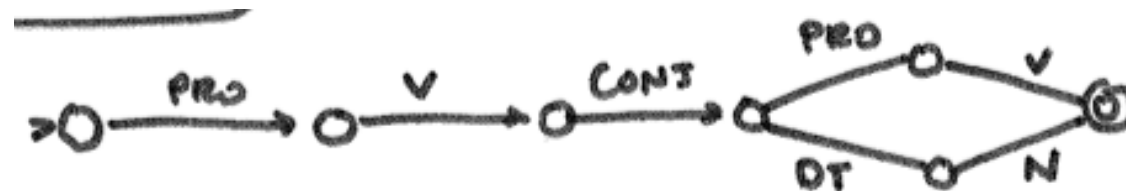
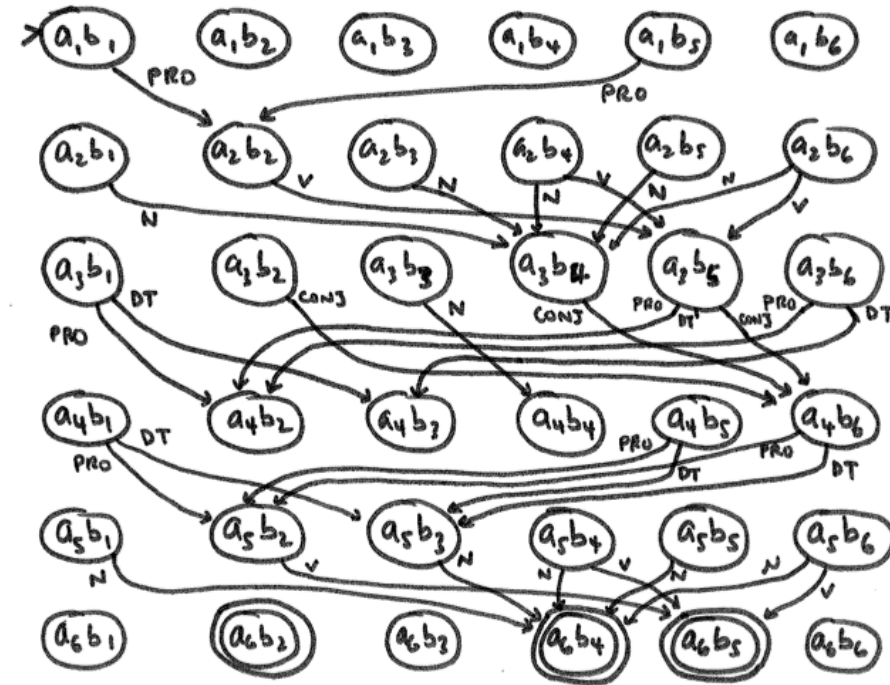
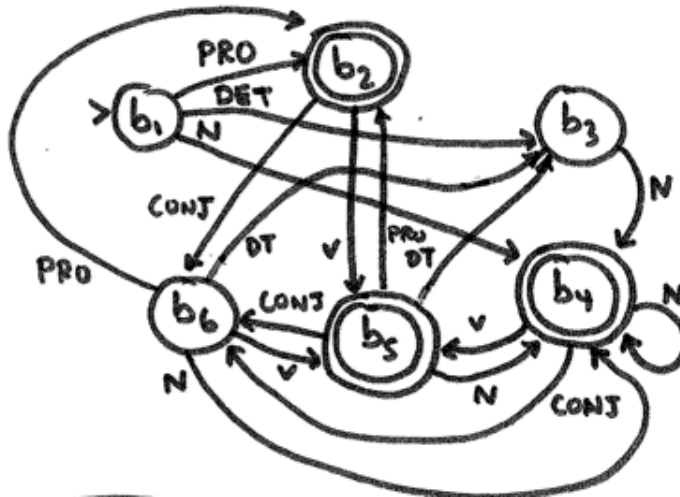
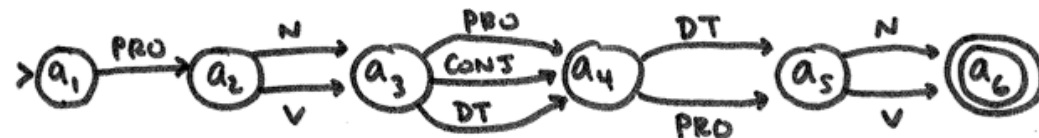
Multiple inputs & outputs



- text-to-sound: { <cat, K A E T>, <dog, D A W G>, <bear, B E H R>, <bare, B E H R>... }
- translator: { <he is in the house, el está en la casa>, <he is in the house, está en la casa>, ... }

POS Tagging Revisited

- he hopes that this works

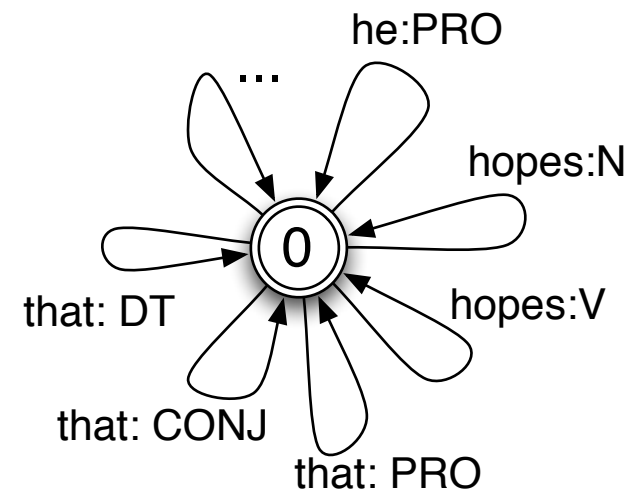


Redo POS Tagging via composition

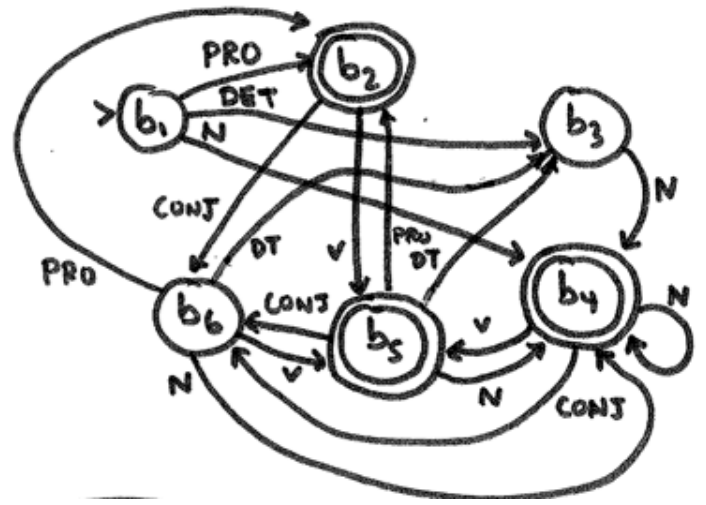
FST A: sentence



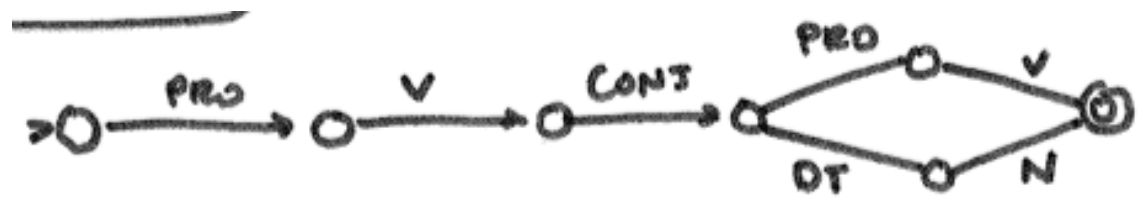
FST B: lexicon



FST C: POS bigram LM



$$\text{proj}_{\text{out}} (A \diamond B \diamond C) =$$



Q: how about $A \diamond (B \diamond C)$? what is $B \diamond C$?