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

• $L = \{ \text{all strings of repeated instances of } \text{ab} \text{ or } \text{aba} \}$
  • 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: \( L_1 L_2 = \{ xy | x \in L_1, y \in L_2 \} \)
- 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”
FSAs vs. Regular Expressions

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

$\bullet$ 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;
  - $\Sigma$ is a finite set, called the input alphabet;
  - $\Gamma$ 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, KAET>, <dog, DAWG>,
  <bear, BEHR>, <bare, BEHR>...\}
  - (easy for Spanish/Italian, medium for French, hard for English!)

- **POS tagger:** \{<I saw the cat, PRO V DT N>, ...\}

- **transliterator:** \{<bush, 布什>, <obama, 奥巴马>, ...\}

- **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
- 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 \( A \)
  - \( <x, y> \) in \( T(A) \) means \( x \sim_A y \)
- compose \( (A, B) = C \)
  - \( <x, y> \) in \( T(C) \) iff. \( \exists z: <x, z> \) in \( T(A) \) and \( <z, y> \) in \( T(B) \)
Simple Example

- pluralizer + capitalizer

FST A pluralizes:

FST B capitalizes:

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 => a:a)
  - \( A \cap B = \text{proj}_{\text{in}} ( \text{Id}(A) \triangleleft \text{Id}(B) ) \)
- what about FSAs composed with FSTs?
  - FSA \( \triangleleft \) FST --- get output(s) from certain input(s)
    - \( <x, z>: \exists y \text{ s.t. } <x, y> \text{ in } T(\text{Id}(A)) \text{ and } <y,z> \text{ in } T(B) \)
    - but y=x \( \Rightarrow \) \( <x, z>: x \text{ in } L(A) \text{ and } <x,z> \text{ in } T(B) \)
  - FST \( \triangleleft \) FSA --- get input(s) for certain output(s)
e.g., pluralize "cat"

A:
\[ \begin{array}{c}
\rightarrow c : c \\
\rightarrow a : a \\
\rightarrow t : t \\
\rightarrow \circ
\end{array} \]

\{ \langle \text{cat, cat} \rangle \}

B:
\[ \begin{array}{c}
a : a \\
t : t \\
\rightarrow \text{ex} : s \\
\rightarrow \circ
\end{array} \]

\{ \langle \text{cat, cats} \rangle, \\
\langle \text{act, acts} \rangle, \\
\ldots \}

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

FST \{\langle \text{cat, cats} \rangle\}

throw away input labels

FSA \{\text{cats}\}
Get Input

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

Compose \( B, C \)

\[
\begin{align*}
B: & \quad \begin{array}{c}
\begin{array}{c}
0 \\
\uparrow
\end{array} \\
\begin{array}{c}
0 \\
\uparrow
\end{array} \\
\begin{array}{c}
\left\langle \text{cat, cts} \right\rangle \\
\left\langle \text{act, acts} \right\rangle
\end{array}
\end{array} \\
C: & \quad \begin{array}{c}
\begin{array}{c}
0 \\
\uparrow
\end{array} \\
\begin{array}{c}
0 \\
\uparrow
\end{array} \\
\left\langle \text{acts, acts} \right\rangle
\end{array}
\end{align*}
\]
Multiple Outputs

- text-to-sound: \{<\text{cat, K AE T}>, <\text{dog, D AW G}>,
  <\text{bear, B EH R}>, <\text{bare, B EH R}>\ldots\}

- translator: \{<\text{he is in the house, el está en la casa}>,
  <\text{he is in the house, está en la casa}>, \ldots\}
POS Tagging Revisited

- he hopes that this works
Redo POS Tagging via composition

FST A: sentence

\[ \text{he} \xrightarrow{} \text{hopes} \xrightarrow{} \text{that} \xrightarrow{} \text{this} \xrightarrow{} \text{works} \]

FST B: lexicon

\[ \text{he:PRO} \]
\[ \text{hopes:N} \]
\[ \text{that:DT} \]
\[ \text{that:CONJ} \]
\[ \text{that:PRO} \]

FST C: POS bigram LM

\[ \text{proj}_{\text{out}} \left( A \cdot B \cdot C \right) = \]

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