Computational Linguistics

Greg Kobele

Universität Leipzig

Winter Semester, 2020

Expressivity

Which languages are describable as FSAs?

- we can do many things with FSAs

Questions

- are all aspects of human languages FS?
- are any aspects of human language FS?

Tails

Tails

Given a language L,

what are the *grammatical continuations* of a word?

- w is a tail of x just in case $xw \in L$
- $T_L(x)$ is the set of all tails of x

Basic Facts

if $x \in L$, then

$\epsilon \in T_L(x)$

$T_L(\epsilon)$

is the whole language

Having the same tails

$x \equiv_L y$

means 'x and y have the same tails'

$$T_L(x) = T_L(y)$$

$\equiv_L$ is an equivalence relation

- $x \equiv_L x$
- if $x \equiv_L y$ then $y \equiv_L x$
- if $x \equiv_L y$ and $y \equiv_L z$ then $x \equiv_L z$

Each state in a DFA represents a set of tails

For each state q,

$L(q) = \{w : \delta^*(q, w) \in F\}$

- the set of words that, from q, take you to a final state

if $\delta^*(q_0, u) = q$ and $\delta^*(q_0, v) = q$

then $T_L(u) = L(q) = T_L(v)$

If a DFA represents L

then L must have a finite set of distinct tails ($\equiv_L$ is of finite index)

From tails to DFA

Construct an automaton:

1. $Q = \{T_L(w) : w \in \Sigma^*\}$
2. $q_0 = T_L(\epsilon)$
3. $F = \{q \in Q : \epsilon \in q\}$
4. $\delta(T_L(w), a) = T_L(wa)$

Finite (i.e. a DFA) just in case

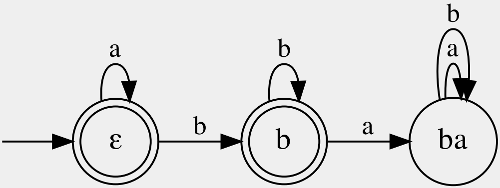
L has a finite set of distinct tails ($\equiv_L$ is of finite index)

Myhill-Nerode Theorem

A language L is finite state iff its relation $\equiv_L$ is of finite index

Example: $L = a^*b^*$

1. $T_L(\epsilon) = T_L(a) = T_L(aa) = \ldots = a^*b^*$
2. $T_L(b) = T_L(ab) = T_L(aaabb) = \ldots = b^*$
3. $T_L(ba) = \emptyset$



Example: $L = a^n b^n$

1. $T_L(\epsilon) = a^n b^n$
2. $T_L(a) = a^n b^{n+1}$
3. $T_L(aa) = a^n b^{n+2}$

   $\vdots$

4. $T_L(ab) = T_L(aabb) = T_L(a^n b^n) = \ldots = \{\epsilon\}$
5. $T_L(aab) = T_L(aaabb) = T_L(a^{n+1} b^n) = \{b\}$
6. $T_L(aaab) = T_L(aaaabb) = T_L(a^{n+2} b^n) = \{bb\}$
7. $T_L(aaaab) = T_L(aaaaabb) = T_L(a^{n+3} b^n) = \{bbb\}$

   $\vdots$

8. $T_L(b) = \emptyset$

EXAMPLE: $L = a^n b^n$

- $T_L(a^i) = \{a^j b^{i+j}\}$, for each i
- $T_L(a^{i+j} b^i) = \{b^j\}$, for each i and j
- $T_L(b) = \emptyset$

Not finite state!!!

$a^n b^n$ cannot be described by a DFA

Hypothesis:

all phonotactic patterns are finite state

Algebraic manipulation

Notation

Languages

$L, L_1, L_2, \ldots$

Machines

$M, M_1, M_2, \ldots$

Cross-Product Machine

Given M_1 and M_2

The *cross-product* machine simulates M_1 and M_2 running in parallel

states are pairs $\langle q_1, q_2 \rangle$

I am in state q_1 in M_1, and in q_2 in M_2

transitions are *pointwise*:

If I am in state q_1 in M_1, and in q_2 in M_2 and read an "a" then I go to state $\delta_1(q_1, a)$ in M_1 and $\delta_2(q_2, a)$ in M_2

$$\delta(\langle q_1, q_2 \rangle, a) = \langle \delta_1(q_1, a), \delta_2(q_2, a) \rangle$$

Complement

If L is finite state, so is the set of L-ungrammatical words!

$$w \in \overline{L} \text{ iff } w \notin L$$

Algorithm:

run M on w, accept if M rejects, and reject if M accepts

Construction

take M, but exchange final and non-final states!

Union

If L_1 and L_2 are finite state, so is their union!

$$w \in L_1 \cup L_2 \text{ iff } w \in L_1 \text{ or } w \in L_2$$

Algorithm:

run both M_1 and M_2 on w, accept if either computation accepts

Construction

take *cross-product* machine of M_1 and M_2; state $\langle q_1, q_2 \rangle$ is final iff either of q_1 or q_2 are

Intersection

If L and L' are finite state, so is their intersection!

$$w \in L \cap L' \text{ iff } w \in L \text{ and } w \in L'$$

Algorithm:

run both M and M' on w, accept if both computations accept

Construction

take *cross-product* machine; state $\langle q_1, q_2 \rangle$ is final iff both of q_1 and q_2 are

Concatenation

If L and L' are finite state, so is their concatenation!

$$w \in LL' \text{ iff } w = uv, \text{ and } u \in L \text{ and } v \in L'$$

Algorithm:

run M on w. When you pass through a final state, you can stop, and run M' on whatever is left.

Construction

take M and M'. The start state is the same as in M. The final states are the same as in M'. At each final state of M, add an empty transition to the start state of M'.
(this is a non-deterministic machine)

Reversal

If L is finite state, then so is L^r

$$w \in L^r \text{ iff } w^r \in L$$

(w^r is just w written backwards)

Algorithm

start in a final state, read transitions backward, and accept if you end in a start state

Construction

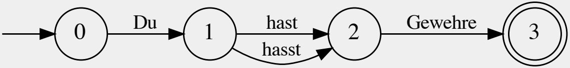
take M. start states are M's final states, final states are M's start states, transitions are reversed: $q' \in \delta^r(q, a)$ iff
$q \in \delta(q', a)$
(this is a non-deterministic machine)

Intersection

grammatical complex pattern as the conjunction of simpler constraints

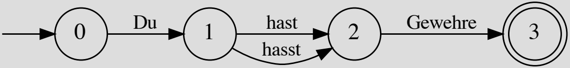
processing represent uncertainty about input!

- *Du hast Gewehre* vs *Du hasst Gewehre*



Parsing as intersection

Input



Intersect with Grammar

well-formed iff intersection grammar non-empty

Testing for emptiness

M is empty

'means' M doesn't accept any strings

$$L(M) = \emptyset$$

Idea

is there a path from a start state to an empty state?

Algorithm

- close the set of start states under transitions
- is at least one final state in the result?