Finite-state automata

Prerequisites

- strings (string parts)
- general (big operators)

Prefix trees, although limited to tree structures rather than arbitrary graphs, generalize our standard notion of graphs in that they have both vertex labels (which we called *colors*) and edge labels (the actual characters of the strings). We briefly entertained the notion of generalizing prefix trees to prefix DAGs, but that did not turn out to be particularly useful for our intended application, namely a more efficient encoding of word lists. But when we take one more step and generalize prefix trees from trees to arbitrary graphs with colors and edge labels, we do get a very useful kind of object: *finite-state automata*.

1 Automata as graphs

A finite-state automaton (FSA) is a finite graph that has both edge labels and vertex labels. The edges are usually called **arcs**, and the vertices are called **states** (by now you're hopefully accustomed to one and the same thing having many different names). We will freely switch between these terms depending on how much we want to emphasize the graph-theoretic nature of FSAs.

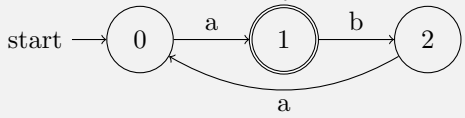
As was just said, FSAs have both edge labels and vertex labels. The edge labels are drawn from some arbitrary alphabet. The vertex labels are used to distinguish between four types of vertices, two of which are already familiar from prefix trees:

1. normal vertices,
2. **final** vertices,
3. **initial** vertices,
4. vertices that are both initial and final.

We already had normal and final vertices for prefix trees (they were color-coded as red and blue, respectively). Initial vertices are a new type. For prefix trees, it is obvious that we always want to start at the source, i.e. the root of the tree. An arbitrary graph may have multiple sources, however, or none at all, so instead the possible starting points have to be indicated explicitly by marking them as initial.

Any graph that satisfies the requirements above is a finite-state automaton. As with prefix trees, we can look at the strings that are associated with paths from an initial vertex to a final vertex and thus compute a (possibly infinite) set of strings.

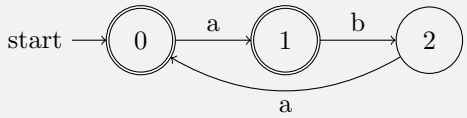
Example 1.
Consider the FSA below, where initial states are marked by an edge labeled *start* and final nodes are doubly circled.



The shortest path from an initial to a final state goes from 0 to 1, or simply $\langle 0, 1\rangle$. This path contains only an a along the way. So the string associated with this path is a.

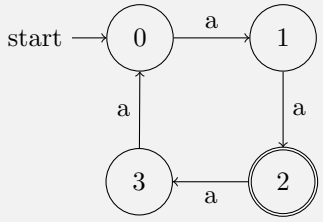
The next longer path is $\langle 0, 1, 2, 0, 1\rangle$, yielding $abaa$. After that, one can follow the path $\langle 0, 1, 2, 0, 1, 2, 0, 1\rangle$ and obtain the string $abaabaa$. In sum, all the associated strings start with an a, followed by 0 or more instances of baa.

Example 2.
In the minor variant below, the initial state is also a final state.



As a result, the empty path is a valid path from an initial state to a final state. The empty path is associated with the empty string ε. In addition, for every valid path ending in 1 there is now a valid truncated version missing the final step from 0 to 1. This also allows for the following strings: a, aba, $ababa$, and so on.

Example 3.
The automaton below produces strings over $\{a\}$ of length l such that $l \mod 4 = 2$.

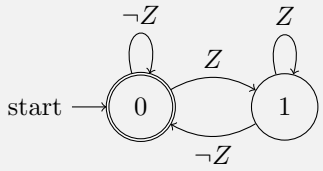


FSAs are incredibly useful for modeling natural language. For example, the n-gram grammars we have seen are all special cases of FSAs.

Example 4.
Consider the SL grammar that bans word-final voicing for German:

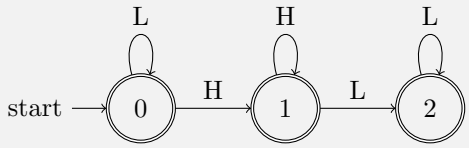
$$\{\text{b}\ltimes, \text{d}\ltimes, \text{v}\ltimes, \text{z}\ltimes\}$$

We can convert this to an FSA that will move us into a non-final state whenever a voiced consonant is encountered. Since we only consider paths that end in a final state, it is impossible for a word to end in a voiced consonant. For the sake of succinctness, we denote all voiced consonants by Z and all other sounds by $\neg Z$.



Example 5.
A *strictly piecewise* (SP) grammar is similar to an SL grammar except that each n-gram is a forbidden subsequence, rather than a forbidden substring. For example, the phenomenon of unbounded tone plateauing forbids low tones (L) from occurring between high tones (H) no matter how far apart the two high tones are. So LHLLLLL and LLLLLHL are well-formed, but not LHLLLHL. An SP-grammar can capture this by forbidding the subsequence HLH.

Equivalently, one can construct an FSA where seeing an L after an H moves us into a special part of the graph where all edges are labeled L. This way it becomes impossible to continue a string like LHLLL with an H.



The last example illustrates how vertices in an FSA serve as a limited kind of memory. The fact that we are in a specific vertex implicitly encodes that certain symbols were encountered along the path to this vertex, and by carefully placing edges from this vertex we can regulate how the computation proceeds from here. This connection between vertices and "memory states" is why the term is finite **state** automata.

2 Definition and terminology

The canonical definition of FSAs looks very different from the graph-theoretic one. This is because FSAs were invented independently, and none of the important theorems about them build on the insights of graph theory. I will first define FSAs in graph-theoretic terms, and then contrast those definitions with the canonical one from formal language theory.

Definition 1. A **finite-state automaton** (FSA) is a vertex- and edge-labeled (directed) graph $A := \langle V, E, \Sigma, c, \ell \rangle$ such that

- V is a finite set of vertices, also called **states**, and
- $E \subseteq V \times V$ is the edge relation, also called the **transition relation**, and
- Σ is a fixed, non-empty alphabet, and
- ℓ maps edges to members of $\wp(\Sigma)$, and
- c maps vertices to members of $\wp(\{I, F\})$.

A vertex v is called **initial** iff $I \in c(v)$ and **final** iff $F \in c(v)$.

The definition above isn't too different from that of graphs mostly straight-forward, except for ℓ and c. The role of c is to indicate whether a vertex is initial (mapped to $\{I\}$), final (mapped to $\{F\}$), neither (mapped to $\emptyset$), or both (mapped to $\{I < F\}$). The labeling function ℓ, on the other hand, associates each edge with a set of symbols. This may surprise you because the examples so far had edges labeled with symbols. But this was a bit of a notational trick to obscure the use of sets.

Example 6.
In the devoicing automaton above, Z and $\neg Z$ each represent multiple symbols. Each one of b, d, v, and z can take us from 0 to 1 in that automaton, which we represented with a single edge from 0 to 1 that is labeled Z. More accurately, we should have four distinct edges from 0 to 1, one labeled b, one labeled d, one labeled v, and one labeled z. But we cannot have 4 distinct edges that connect the same vertices. Each one of those edges would be exactly the same pair $\langle 0, 1 \rangle$. So instead, we say that there is a single edge from 0 to 1 that is labeled $\{b, d, v, z\}$, indicating that any one of those symbols can take us from 0 to 1.

Exercise 1.
Alternatively, we could use a different definition of labeled edges where we do not have a labeling function $\ell : E \to L$ from the set of edges to some fixed set L of edge labels, but instead E itself is a subset of $V \times L \times V$. Explain why this removes the need for sets as edge labels.

Every path through the graph is also associated with a set of strings. Intuitively, these are all the strings that can be built by following along the path.

Definition 2. With every finite path $p = \langle v_1, v_2, v_3 \ldots, v_{n-1}, v_n \rangle$ we associate a string set $L(p) := \ell(\langle v_1, v_2 \rangle) \times \ell(\langle v_2, v_3 \rangle) \cdots \times \ell(\langle v_{n-1}, v_n \rangle)$. If $p = \langle\rangle$, $L(p) = \emptyset$. If $p = \langle v_1 \rangle$, then $L(p) = \{\varepsilon\}$ if v_1 is final, and $\emptyset$ otherwise. Let P be the set of all paths from an initial state to a final state. Then the language **recognized** by A is

$\bigcup_{p \in P} L(p)$. For every stringset $L \subseteq \Sigma^*$, L is **regular** iff L is recognized by some FSA.

The canonical definition of FSAs can avoid the complication of set-labeled edges by directly treating edges as triples of the form *start*, *label*, *end*.

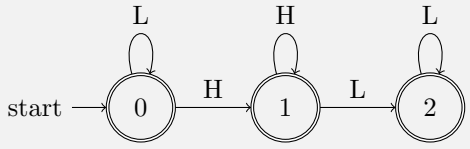
Definition 3. A **finite-state automaton** (FSA) is a 5-tuple $A := \langle \Sigma, Q, I, F, \Delta \rangle$ such that

- the alphabet Σ is a finite, non-empty set,
- Q is a finite set of **states**,
- $I \subseteq Q$ is the set of **initial** states,
- $F \subseteq Q$ is the set of **final** states,
- $\Delta \subseteq Q \times \Sigma \times Q$ is the **transition relation**.

Given a string $s := \sigma_1 \cdots \sigma_n \in \Sigma^n$ $(n \geq 0)$, a **run** of A over s is a tuple $r := \langle q_0, q_1, \ldots, q_n \rangle$ such that $q_0 \in I$ and for all $0 < i \leq n$, $\langle q_{i-1}, \sigma_i, q_i \rangle \in \Delta$. A run is **accepting** iff its last component is a final state. A string s is **recognized** or **generated** by A iff there is some accepting run of A over s. The string language $L(A)$ recognized/generated by A is the smallest set containing all strings recognized by A.

A run is just a record of which states an automaton passes through when processing a string. The run is accepting if it starts in an initial state and ends in a final state. Note that one string may allow for multiple runs. A string is recognized by the automaton iff there is at least one accepting run.

Example 7.
Consider once more the automaton for unbounded tone plateauing.

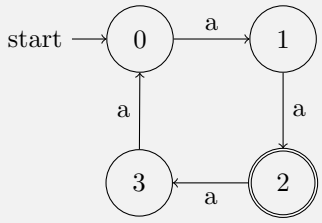


In this automaton, the string LLLHH has only one run, which is 000011. Note how the run is one symbol longer than the string. That's because we start in 0, and then the first symbol (i.e. L) moves us from 0 to 0. In more detail:

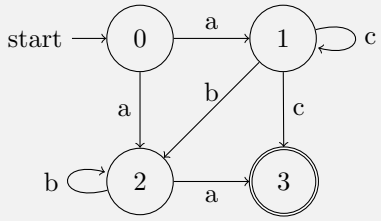
- start: 0
- L: 0
- L: 0
- L: 0
- H: 1
- H: 1

- done

Example 8.
In the automaton below, the string *aaaaaaa* receives the run 01230123.



Example 9.
Now consider the automaton below.

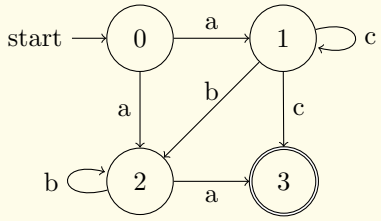


The string *aba* has two distinct runs. One is 0123, the other one is 0223.

Exercise 2.
Draw an FSA that recognizes the language a^*b^+, where a^* denotes "0 or more *a*s" and b^+ is short for "1 or more *b*s".

Exercise 3.
Draw an FSA that recognizes the language $a^+b^+a^*$.

Exercise 4.
Consider once more the following automaton:



For each one of the following strings, list all accepting runs with respect to this automaton. If there is no such run, say so.

1. *aa*

2. $acbba$
3. $abba$
4. $abab$

Exercise 5.
For each one of the following string languages, draw the smallest FSA that recognizes the language.

1. $\{aa\}$ (the string aa, and nothing else)
2. the set of all strings except aa
3. $b^+(aa)^+$ (1 or more bs followed by an even number of as, but at least 2 as)
4. $\{a, b\}^*$ (the set of all strings over a and b, including the empty string ε)
5. $(acdc)^*$ (0 or more iterations of $acdc$)
6. the set of all strings over a, b, and c such that the string contains a iff it does not contain c
7. $\{a, b\}^* c \{a, b\}^*$ (the set of all strings over a, b, and c that contain exactly one c)
8. $c\{a, b, c\}^* c$ (the set of all strings over a, b, and c that start with a c and end with another c)
9. the set of all strings over a, b, and c whose first symbol is distinct from their last symbol
10. the set of all strings over a and b such that the number of as within the first 4 symbols must not exceed the number of bs among the last 4 symbols
11. the set of all strings over a, b, and c where a never occurs immediately to the right of b (ab and bca are fine, but ba is not)
12. the set of all strings over a, b, and c where a never occurs anywhere to the right of b (ab is fine, but bca and ba are not)
13. the set of all strings over a, b, and c such that the number of cs in the string is a multiple of 3 or 5
14. the set of all strings over a and b that are worth at least 10 points, where each a is worth 2 points and each b is worth 3 points
15. the set of all strings over a and b whose point value is at most 10 or a multiple of 3, where each a is worth 2 points and each b is worth 3 points
16. the set of all strings over a, b, c, and d where a may have b somewhere to its left and a b somewhere to its right only if there is no d in the string
17. the set of all strings over a, b, and c such that whenever a b is immediately followed by at least one a, the next b to the right must be immediately followed by at least one c (for example $bbacabcbb$ is well-formed, but $bbacabbcb$ is not because the second b is immediately followed by a but the third b is not immediately followed by c)
18. the set of all strings over a, b, and c such that whenever a b has an a somewhere to its right, then the next b to the right must have at least

one *c* somewhere to its right (*bbacabbcb* is now well-formed, and so is *bbacabbabbc*, but *bbacabbbb* and *bbacabbabbb* are ill-formed)