Prerequisites

- linear algebra (not in background section yet)
  - matrices
  - matrix multiplication

Finite-state recognition as Boolean matrix multiplication

Generators and recognizers

Finite-state automata can be used as **generators** or as **recognizers**. As a generator, an FSA is used to produce well-formed strings, similar to how linguistics may use a phrase structure grammar to generate well-formed sentences. As a recognizer, the FSA is given a string as input and has to decide whether this string is well-formed. That is to say, the string has to describe a valid path from an initial state to some final state. Recognition is the more common use of FSAs.

Since recognition is a very common of FSAs in practical applications, one would like it to be as fast as possible. We already know that deterministic automata can be much faster than non-deterministic ones, so determinization can be an important step of a fast FSA implementation. However, there is a very different route which has the advantage of reducing FSA recognition to a process that has already been heavily optimized: matrix multiplication. Matrix multiplication is essential in tons of applications, in particular high-performance scientific computing. Thousands of smart minds have thought long and hard about how matrix multiplication can be done as quickly as possible. By reducing FSA recognition to matrix multiplication, we can use standard packages for matrix multiplication and harness all those speed improvements for free.

Boolean matrix multiplication

Instead of standard matrix multiplication, we will be using **Boolean matrix multiplication**. It works just like standard matrix multiplication, except that the only values we can have are the two Boolean values 0 and 1. Matrices with other values cannot be used in Boolean matrix multiplication. As long as we start with matrices that only contain 0s and 1s, almost everything will work just fine. The only problem is addition, which has to be capped at 1: whenever we would get a value above 1, we just use 1 instead.

Example 1 Consider the following formula, with $\otimes$ used for matrix multiplication.

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}$$

With normal matrix multiplication, this would yield the following matrix:

$$\begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}$$

With Boolean matrix multiplication, the 2 is capped to 1, yielding

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Exercise 1 For each one of the following equations, give its result under Boolean matrix multiplication. If the output is undefined, say so.

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Boolean matrix multiplication is actually quite a bit easier than standard matrix multiplication since we only need to determine if a cell should be filled by 0 or 1. And this is is pretty simple. Suppose we are trying to compute the product of row r in the first matrix and column c in the second matrix. As usual, we multiply the i-th value of r with the i-th value of c and then add them all up. But we actually do not need to consider each i. First of all, if the i-th value in r or c is 0, then obviously multiplication will return 0. But 0 does not add anything to the addition step. Hence the i-th values of r and c are of interest to us only if they are both 1. In that case, the i-th values contribute 1 to the addition step. But remember that addition is capped at 1, so as soon as we have at least 1 it does not matter whether the other values multiply out to 0 or 1. For this reason, Boolean matrix multiplication follows a very simple recipe:

1. Suppose you are looking at row $r := r_1 \cdots r_n$ of the first matrix and column $c := c_1 \cdots c_n$ of the second matrix.
2. For each i such that $r_i = 1$, check the value of c_i.
    1. If $c_i = 1$, the product of r and c is 1.
    2. If there is no i with $c_i = 1$, the product of r and c is 0.

Example 2 Calculating the matrix product of the two matrices below might usually take you quite a while.

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}$$

But it is very easy with our shortcut for Boolean matrix multiplication.
Row 1 & Column 1: Only the 2nd, 4th, and 6th value of Row 1 are not 0. We check the 2nd value of Column 1, which is 0. At this point, the final result is still unclear. We move on to the 4th value, which is 1. We stop and fill in 1.
Row 1 & Column 2: We already know that we only need to check the 2nd, 4th, or 6th value of Column 2. All three of them are 0, so we have to fill in 2.
Row 2: Since all values of Row 2 are 0, we can immediately fill in 0 for all columns of that row.
The final matrix is shown below:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

Exercise 2 Recompute the values for the formulas from the previous exercise, using the faster procedure now. Write down the relevant steps as in the example above. Skip any formulas that you said were not well-defined for Boolean matrix multiplication.

FSA recognition as Boolean matrix multiplication: the intuition

In order to reduce finite-state recognition to Boolean matrix multiplication, we need a method for translating a given FSA into a collection of matrices. The method is actually fairly easy, and it works for both deterministic and non-deterministic automata. However, the automaton must not contain any ε-transitions (there is an algorithm for removing such transitions).

We will start with a concrete example as the translation procedure is a little abstract when written out in formal terms.

Example 3 Consider once more the familiar automaton for the language $a(baa)^*$. To avoid confusion, the states have been renamed X, Y, and Z.

$$\begin{tikzcd}
\text{start} \arrow[r] & X \arrow[r, "a"] & Y \arrow[r, "b"] & Z \arrow[ll, bend left, "a"]
\end{tikzcd}$$

This automataon uses an alphabet with only two symbols, a and b. For each one of these symbols, we create a matrix that records how we can transition from one state into another via this symbol. Since the automaton has 3 states, we create two 3-by-3 matrices, one for a and for b. At this point, both matrices are empty, but for increased clarity we label each row and column with a state of the automaton.

$$\begin{bmatrix} a & X & Y & Z \\ X & & & \\ Y & & & \\ Z & & & \end{bmatrix} \qquad \begin{bmatrix} b & X & Y & Z \\ X & & & \\ Y & & & \\ Z & & & \end{bmatrix}$$

We now have to look at the transitions of the automaton in order to fill these matrices. We only have one transition leaving X, which goes to Y via a. We put a 1 in the first row and second column of the a-matrix.

$$\begin{bmatrix} a & X & Y & Z \\ X & & 1 & \\ Y & & & \\ Z & & & \end{bmatrix} \qquad \begin{bmatrix} b & X & Y & Z \\ X & & & \\ Y & & & \\ Z & & & \end{bmatrix}$$

Intuitively, the row of the matrix indicates which state we are currently in, and the column tells us which state we can transition into. So the a-matrix now encodes the fact that we can move from X to Y via a.
This leaves us with two transitions. One goes from Y to Z via b, so we have to add a 1 in the Y-row and Z-column of the b-matrix. The other one goes from Z to X via a, which means that the a-matrix also should have a 1 in row Z, column X.

$$\begin{bmatrix} a & X & Y & Z \\ X & & 1 & \\ Y & & & \\ Z & 1 & & \end{bmatrix} \qquad \begin{bmatrix} b & X & Y & Z \\ X & & & \\ Y & & & 1 \\ Z & & & \end{bmatrix}$$

We have looked at all the transitions. Any cell that is still empty gets a 0.

$$\begin{bmatrix} a & X & Y & Z \\ X & 0 & 1 & 0 \\ Y & 0 & 0 & 0 \\ Z & 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} b & X & Y & Z \\ X & 0 & 0 & 0 \\ Y & 0 & 0 & 1 \\ Z & 0 & 0 & 0 \end{bmatrix}$$

We are almost done with the translation. We also have to define a matrix I for the initial states and a matrix F for the final states. Intuitively, the initial state matrix records whether we can transition into a given state at the very beginning. So it uses states as columns. The final state matrix records whether we can transition out of a state at the very end. Hence it uses states as rows.

$$I := \begin{bmatrix} X & Y & Z \\ 1 & 0 & 0 \end{bmatrix} \qquad F := \begin{bmatrix} X & 0 \\ Y & 1 \\ Z & 0 \end{bmatrix}$$

And that's it. Remember that X, Y, and Z aren't part of the matrices above, they're just labels that have been added for the sake of clarity.

Given a matrix representation of an FSA, it is very easy to check whether a string is recognized by the automaton. Just replace each symbol by the corresponding matrix, put I at the very beginning of the formula, and F at the very end. Then perform Boolean matrix multiplication on all the matrices. The string is accepted iff the final product is 1.

Example 4

We already konw that the automaton above accepts the string *abaa*. Let's see how we get the same result via Boolean matric multiplication. We have to construct a formula that evaluates to 1.
First, we replace each a in the string by the matrix for a, and each b by the matrix for b:

$$\begin{bmatrix} a & X & Y & Z \\ X & 0 & 1 & 0 \\ Y & 0 & 0 & 0 \\ Z & 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} b & X & Y & Z \\ X & 0 & 0 & 0 \\ Y & 0 & 0 & 1 \\ Z & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} a & X & Y & Z \\ X & 0 & 1 & 0 \\ Y & 0 & 0 & 0 \\ Z & 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} a & X & Y & Z \\ X & 0 & 1 & 0 \\ Y & 0 & 0 & 0 \\ Z & 1 & 0 & 0 \end{bmatrix}$$

Then we add I at the front and F at the end. The result is shown below, with labels removed to simplify the matrix multiplication step:

$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

And now we just solve this formula. It's best to do so from left to right as this keeps the matrices fairly small. The end result is 1, as expected.

Exercise 3 Solve the equation in the example above in two ways:

1. proceeding from left to right, starting with the first matrix,
2. proceeding from left to right but starting with the second matrix; only multiply with the first matrix at the very end.

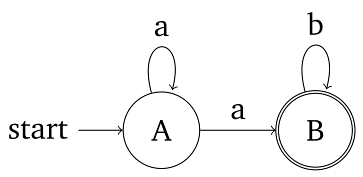
Both routes return the same result because matrix multiplication is associative. But one will be much less work for you than the other.

Exercise 4 Use Boolean matrix multiplication to determine for each one of the following strings whether it is a member of $a(baa)^*$:

1. *aab*
2. *aba*
3. *a*
4. ε

With a bit of practice, you'll notice that resolving the matrix multiplication formulas from left to right captures a specific intuition. You always end up with a 1-dimensional matrix (i.e. a vector). A 1 in column i means that the automaton might currently be in the state corresponding to this column. In a deterministic automaton like the one above, exactly one column has a 1, all others are 0. This is also called a **one-hot vector**. With non-deterministic automata, the matrix may contain multiple 1s as there are multiple states the automaton could possibly be in.

Exercise 5 Consider the non-deterministic FSA below for a^+b^*.



Construct the corresponding matrices for this automaton. Then use matrix multiplication to determine for each one of the following strings whether it is recognized by the automaton. Solve the equations from left to right and pay close attention to how the distribution of 1s reflects which states the automaton may be in.

1. *a*
2. *ab*
3. *aaab*

Exercise 6 Continuing the previous exercise, determinize the automaton. Then repeat the previous steps using the deterministic automaton instead.

FSA recognition as Boolean matrix multiplication: Formal procedure

This section specifies the translation in formal terms.

1. Suppose that we have an automaton $A := \langle \Sigma, Q, I, F, \Delta \rangle$ such that $Q := \{q_1, \ldots, q_n\}$.
2. For each $\sigma \in \Sigma$, we construct a matrix A_σ with n rows and n columns. We fill the cell $c_{i,j}$ in row i and column j with 1 if A allows us to transition from state q_i to q_j via σ ($1 \leq i, j \leq n$). Otherwise, it is filled with 0.
3. The 1-by-n matrix A_I contains a 1 in column i if $q_i \in I$, and 0 otherwise ($1 \leq i \leq n$).
4. The n-by-1 matrix A_F contains a 1 in row i if $q_i \in F$, and 0 otherwise ($1 \leq i \leq n$).
5. We recursively define a translation function t_A from strings over Σ to matrix multiplication formulas. First, $t_A(\varepsilon) := A_F$. Given $a \in \Sigma$ and $u \in \Sigma^*$, $t_A(au) := A_a \otimes t_A(u)$.
6. A string $s \in \Sigma^*$ is recognized by A iff $A_I \otimes t_A(s) = 1$.

Exercise 7 Carefully read through the definition above. Try to make sense of it based on the previously established intuition. Write down anything you do not understand, and discuss it in class.