Week 13 - Friday

COMP 2230

Last time

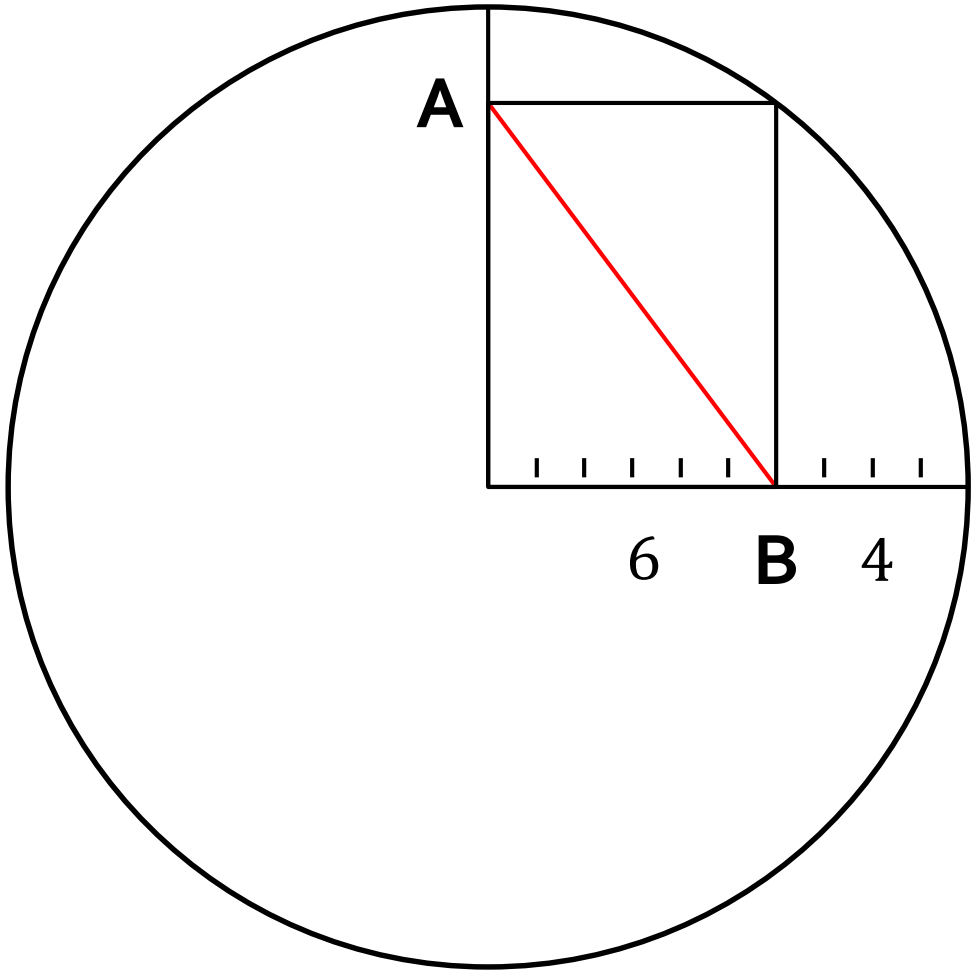
- Finite-state automata

Questions?

Assignment 6

Logical warmup

- What's the length of the rectangle's diagonal, running from **A** to **B**?



Back to Finite-State Automata

Designing automata

- Design a finite-state automaton that accepts the set of all strings of 0's and 1's such that the number of 1's in the string is divisible by 3
- Make a regular expression for this language

- Design a finite-state automaton that accepts the set of all strings of 0's and 1's that contain exactly one 1
- Make a regular expression for this language

FSA = regular expressions

- Kleene's Theorem shows that finite-state automata are equivalent to regular expressions
- That is, for every finite-state automaton there is some equivalent regular expression, and vice versa
- We won't prove it, but it should be intuitively clear because there are algorithms for building FSA's from regular expressions and vice versa

Irregular

- Languages that can be expressed as an FSA or a regular expression are called **regular**
- Some languages are *not* regular
- For example, the language consisting of strings $a^k b^k$, meaning all strings that have a positive number of a's followed by the same number of b's, is not regular
- Prove it (by contradiction)
- **Hint:** We use the pigeonhole principle to show that more than one sequence of a^p and a^q must end up the in the same state

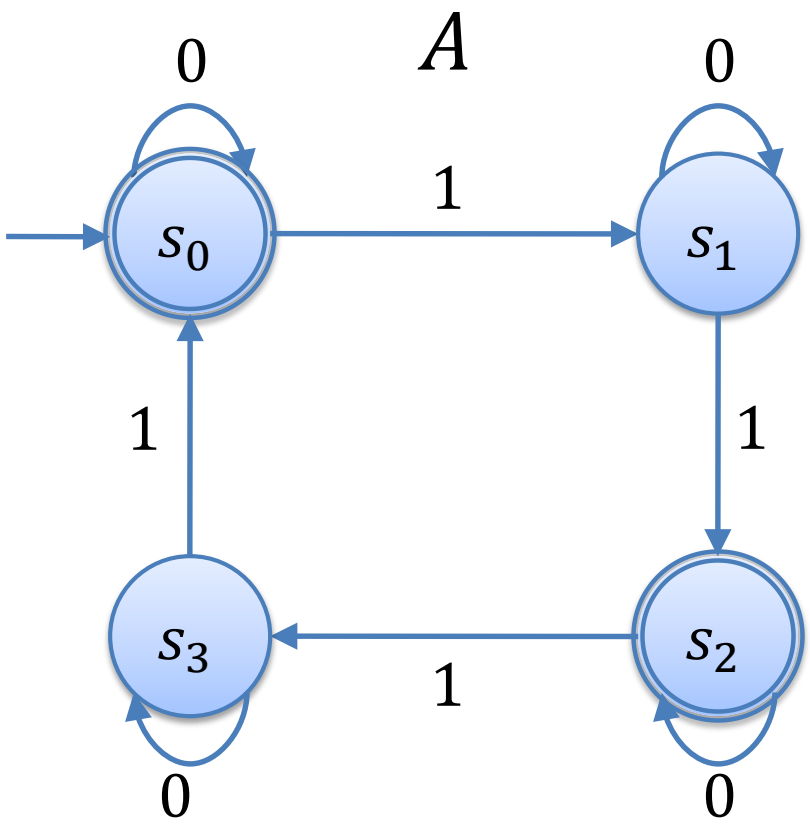
Simplifying FSAs

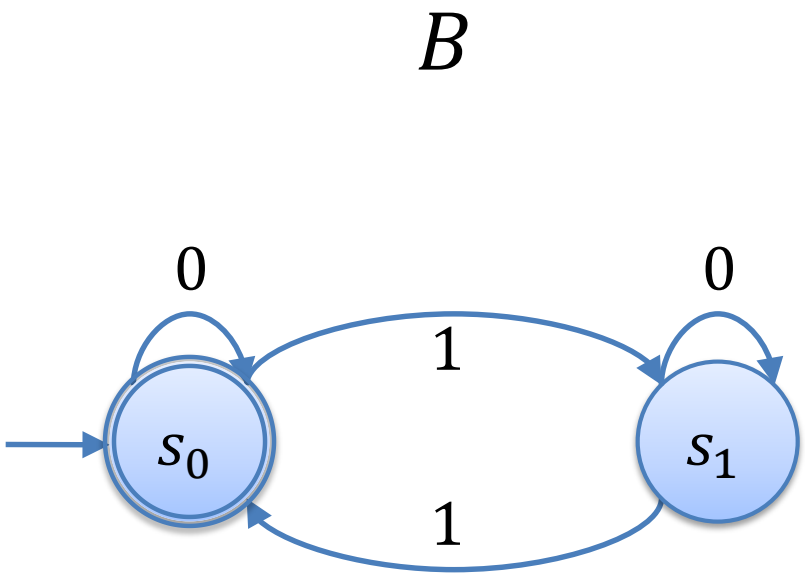
Three-Sentence Summary

Simplifying FSAs

Comparison

- List strings accepted by the FSA A
- List strings accepted by the FSA B





*-equivalence

- Two states of a finite-state automaton are $*$**-equivalent** if any string accepted by the automaton when it starts from one state is accepted when starting from the other
- Given an automaton A with eventual-state function N^*, we can formally say:
  - States s and t in A are $*$-equivalent iff $N^*(s, w)$ and $N^*(t, w)$ are both accepting states or both not
- It turns out that $*$-equivalence defines an equivalence relation

k-equivalence

- $*$-equivalence is hard to demonstrate directly
- Instead, we'll focus on equivalence after k or fewer inputs
- Given an automaton A with eventual-state function N^*, we can formally say:
  - States s and t in A are k-equivalent iff $N^*(s, w)$ and $N^*(t, w)$ are both accepting states or both not, for all strings w of length k or less

Facts about k-equivalence

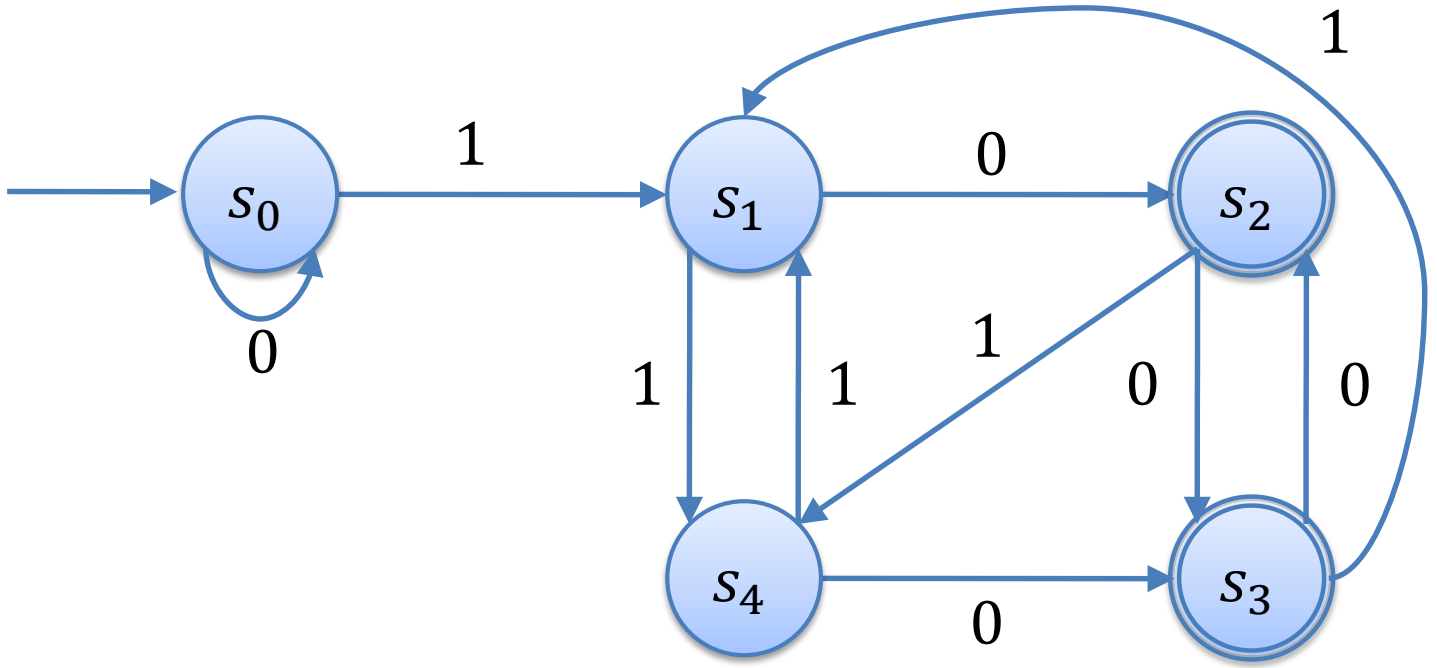
- For $k \geq 0$, k-equivalence is an equivalence relation
- For $k \geq 0$, the k-equivalence classes partition the set of all states of the automaton into a union of mutually disjoint subsets
- For $k \geq 1$, if two states are k-equivalent, they are also $(k-1)$-equivalent
- For $k \geq 1$, each k-equivalence class is a subset of a $(k-1)$-equivalence class
- Any two states that are k-equivalent for all integers $k \geq 0$ are $*$-equivalent

k-equivalence theorems

- Let A be an FSA with next-state function N
- Given any states s and t in A:
  1. s is 0-equivalent to t iff either s and t are both accepting states or they are both nonaccepting states
  2. For every integer $k \geq 1$, s is k-equivalent to t iff s and t are $(k-1)$-equivalent and for any input symbol m, $N(s,m)$ and $N(t,m)$ are also $(k-1)$-equivalent
- These theorems essentially allow us to create a recursive definition for testing k-equivalence

k-equivalence examples

- Find the 0-equivalence classes, the 1-equivalence classes, and the 2-equivalence classes for the following FSA:



Finding the $*$-equivalence classes

- Keep finding k-equivalence classes for larger and larger values of k
- If you ever find that the set of k-equivalence classes is equal to the set of $(k+1)$-equivalence classes, that is the set of $*$-equivalence classes
- This is known as a **fixed point** in mathematics

The quotient automaton

- We can build a new FSA from the $*$-equivalence classes
- Recall that $[s]$ means the equivalence class of s
- This FSA is called the **quotient automaton** A', and is defined from an FSA A with states S, input symbols I, and next-state function N as follows:
  1. The set of states S' of A' is the set of $*$-equivalent classes of states of A
  2. The set of input symbols I' of A' equals I
  3. The initial state of A' is $[s_0]$ where s_0 is the initial state of A
  4. The accepting states of A' are the states of the form $[s]$ where s is an accepting state of A
  5. The next-state function N': $S' \times I \rightarrow S'$ is:

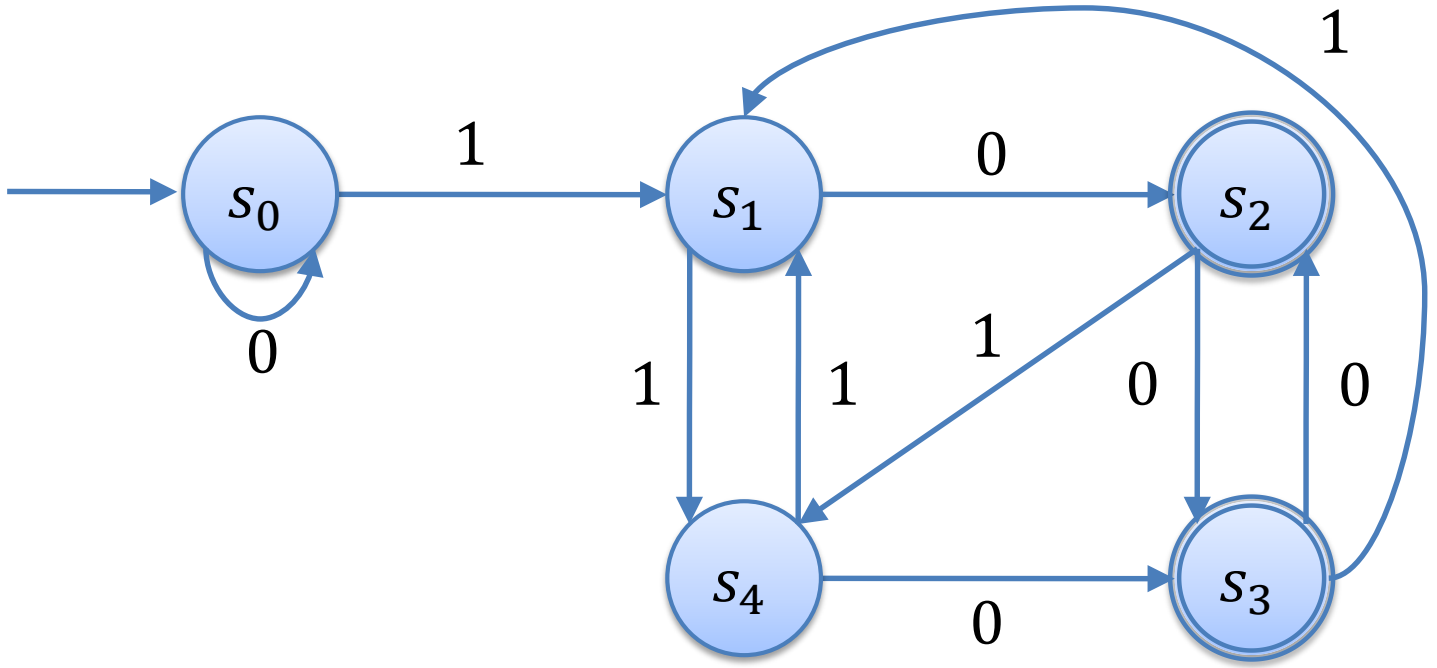
     For all states $[s]$ in S' and input symbols m, $N'([s], m) = [N(s, m)]$

Constructing a quotient automaton

- Let A be an FSA with states S, input symbols I, and next-state function N
- To build A':
  1. Find the set of 0-equivalence classes of S
  2. For each integer $k \geq 1$, find the k-equivalence classes of S until the k-equivalence classes are the same as the $(k-1)$-equivalence classes
  3. Build a quotient automaton whose states are the equivalence classes given above with transition function $N'([s], m) = [N(s, m)]$ for any input symbol m

Quotient automaton example

- Find the quotient automaton for the following FSA



Ticket Out the Door

Upcoming

Next time…

- Finishing simplifying FSAs
- Context-free grammars and pushdown automata

Reminders

- Keep working on Assignment 6