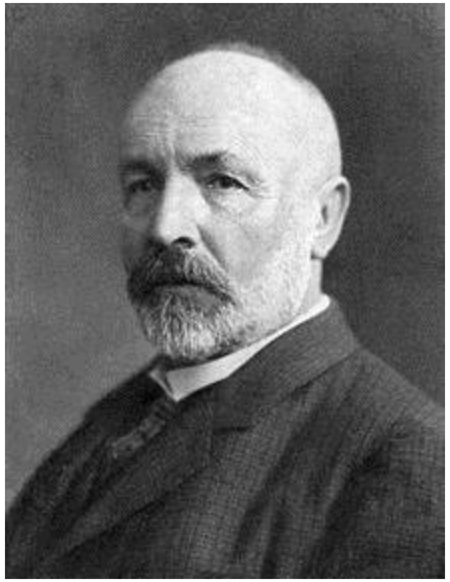
CS3102 Theory of Computation

Cantor’s Theorem



- For any set S, $|2^S| > |S|$
  - Holds when S is finite (homework)
  - What about when S is infinite?
  - If S is countably infinite: diagonalization
- Assume toward contradiction we have $f: S \leftrightarrow 2^S$
  - Let $T = \{x \in S | x \notin f(x)\}$
  - Note that $T \subseteq S$, so there must be some x_t s.t. $f(x_t) = T$
  - Is $x_t \in T$?

Continuum Hypothesis

- We know that $|\mathbb{N}| < |\mathbb{R}|$
- Is there a set S s.t. $|\mathbb{N}| < |S| < |\mathbb{R}|$?
- Answer:
  - Unanswerable

Godel’s Incompleteness Theorem

- Says any axiomatic system is at least one of:
  1. **Inconsistent**: There are false things that you can prove
  2. **Incomplete**: There are true things that you cannot prove
  3. **Weak**: You can’t talk about prime numbers
- Proof idea: Show that any system can construct the paradox “This statement cannot be proven”

Incompleteness in CS*

- Expectation Maximization Problem
  - You want to put ads on your website
  - You don't know yet who will visit your website
  - Select ads to maximize the maximum number of potential customers
- Answering this problem requires "tools" not yet addressed by set theory!

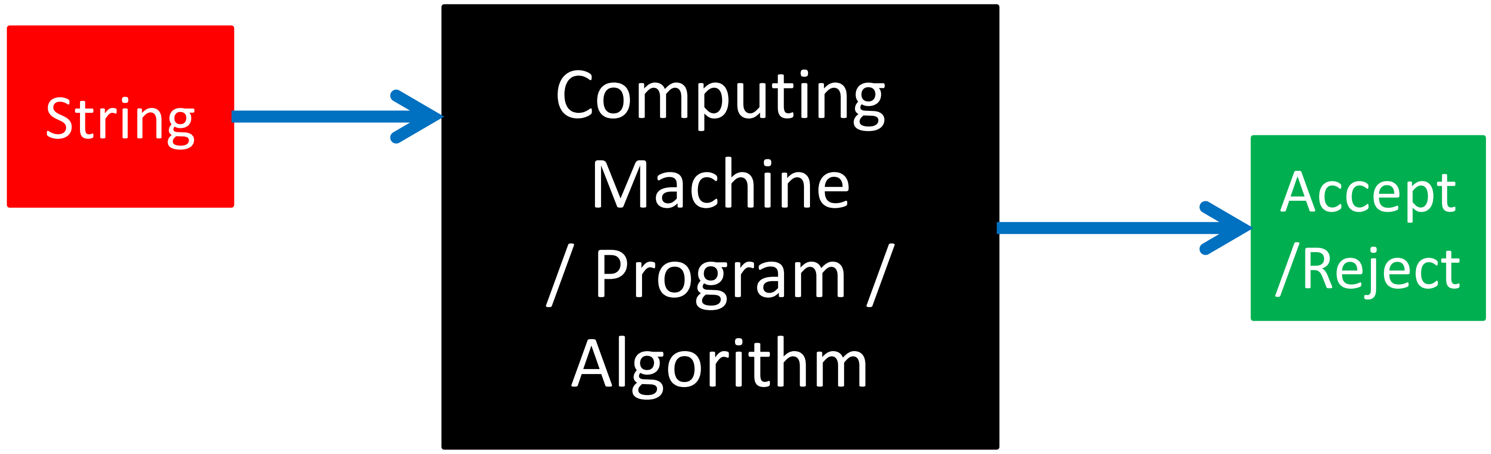
* https://www.nature.com/articles/s42256-018-0002-3

End of Phase 1

- Until now:
  - Mathematical foundations
  - Proof strategies
  - Key ideas/insights
  - Main takeaway: Some languages (and numbers) cannot be computed by Java (or anything else)
    - Why? There are more language (numbers) than there are Java programs (or even finite descriptions)

Phase 2

- Now we start filling in this box
  - First option: finite state machine



Operations on Strings

- Length
  - $|s|$= Number of characters in the string s
  - $|Ringo| = 5$
- Concatenation
  - $s \cdot t = st =$ string which has all of the characters from s followed by all of the characters from t
  - $John \cdot Paul = JohnPaul$
  - $|s \cdot t| = |s| + |t|$
- Exponentiation
  - s^k =The string created by concatenation s with itself k times
  - $(George)^5 = GeorgeGeorgeGeorgeGeorgeGeorge$
  - $\left|s^k\right| = |s| \cdot k$

Empty String (“”)

- Notation for this class: ε
  - \varepsilon in Latex
- $|\varepsilon| = 0$
- $s \cdot \varepsilon = s$
- $\varepsilon^k = \varepsilon$
- $s^0 = \varepsilon$

Operations on Languages

- Everything we can do on sets (∪,∩, −, …)
- Concatenation
- Exponentiation
- Kleene Closure

Language Concatenation

- $L_1 \cdot L_2$ or $L_1 L_2$
  - Notation is the same as string concatenation
  - Every possible way to concatenate a string from L_1 with a string from L_2 (in that order)
  - Idea: take $L_1 \times L_2$ and concatenate the strings that are paired
  - $\{john, paul\} \cdot \{george, ringo\} = \{johngeorge, jonringo, paulgeorge, paulringo\}$
  - $|L_1 L_2| \leq |L_1 \times L_2|$
  - $\{a, aa, aaa\} \cdot \{a, aa\} = \{aa, aaa, aaaa, aaaaa\}$

Language Exponentiation

- L^k
  - L concatenated with itself k times
  - $L^5 = L \cdot L \cdot L \cdot L \cdot L$
  - $\{a, b\}^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$
  - $L^0 = \{\varepsilon\}$

Kleene Closure

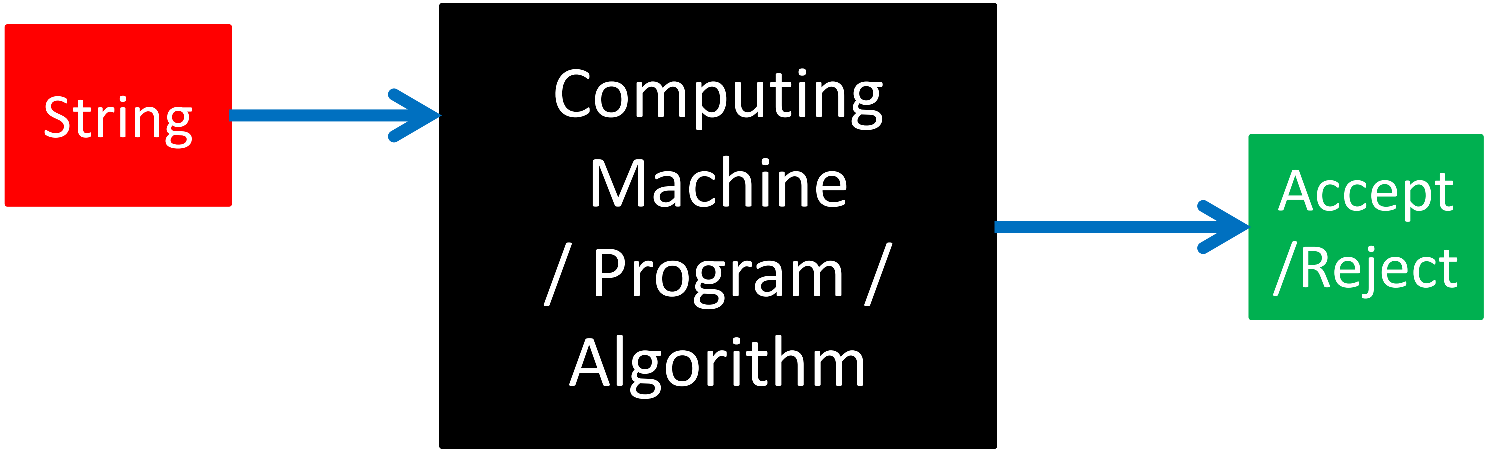
- L^*
  - L concatenated with itself 0 or more times
  - $L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \ldots$
  - $\{a, bb\}^* = \{\varepsilon, a, bb, aa, abb, bba, bbbb, aaa, \ldots\}$
  - $\emptyset^* = \{\varepsilon\}$
  - $\{\varepsilon\}^* = \{\varepsilon\}$
  - For any other language L, L^* is infinite

Sigma Star

- We denote our alphabet as Σ
  - \Sigma in Latex
- A character is just a really short string, so an alphabet is a language
- Σ^* is the set of all strings using the alphabet Σ
- 2^{Σ^*} is the set of all languages using Σ

What Shall we put in the box?

- Goal: start with something easy to prove things about
- We've talked about Java, but that's complex



Finite State Automaton

- Simple model of computation
- Represents computation without memory
- Kind of decider
  - We call the set of strings it accepts the “language” of the machine
- Our machine reads the input string only once, and one character at a time
- After reading each character, enters a new “state”
- State transition rules depend only on the current state and the current character (no looking back!)
- There are only finitely many states

Gumball Machine

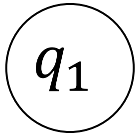
- Our gumball machine takes only pennies and nickels and does not give change
- Each gumball costs 7 cents
- $\Sigma = \{p, n\}$ (penny, nickel)
- We need to decide the language of sequences of coins adding up to at least 7 cents

Gumball Machine

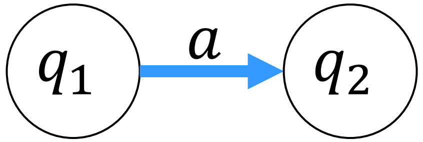
- What are all the possible “states” the machine could be in?
- 0c, 1c, 2c, 3c, 4c, 5c, 6c, 7+c
- Which “state” should the machine start in?
- Which “state” means we’ve sold a gumball?
- 6c plus a penny is always 7c, no matter how I got to 6c ($pppppp$, or pn, or np)

Finite State Automata

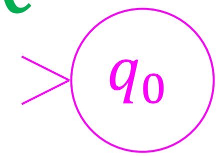
- Basic idea: a FA is a "machine" that changes states while processing symbols, one at a time.



- Finite set of states: $Q = \{q_0, q_1, \ldots q_7\}$



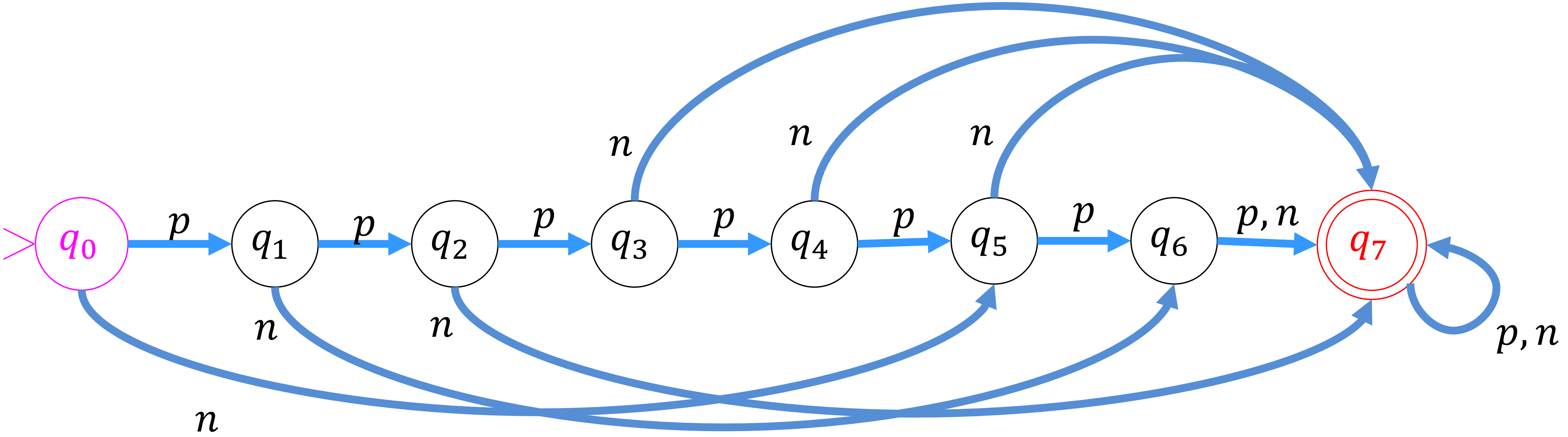
- Transition function: $\delta: Q \times \Sigma \to Q$



- Initial state: $q_0 \in Q$

- Final states: $F \subseteq Q$
- Finite state automaton is $M = (Q, \Sigma, \delta, q_0, F)$
- Accept if we end in a Final state, otherwise Reject

FSA for our Gumball Machine



Strings this accepts:

$pppppppp$

$nnnnnnn$

pnp

ppn

Strings this rejects:

ppp

n

np

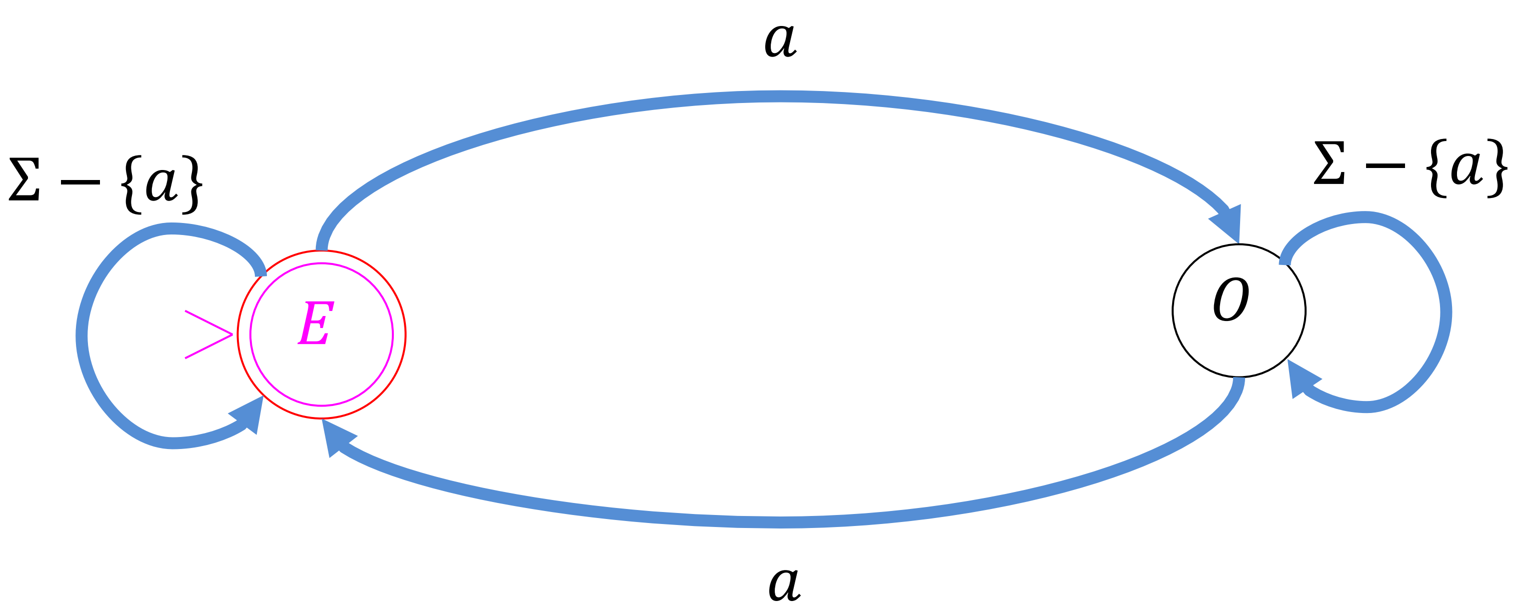
EvenA

- In HW1 you were asked to give a decider for EvenA (accepts all strings with an even number of A’s)
- How did you do it?

EvenA using FSA

1. What's our alphabet? (pick Σ)
2. What should our states be? (pick Q)
3. Which states are the accept states? (pick F)
4. Which state is the start state? (pick q_0)
5. How should we transition? (pick δ)

Let's Draw It!

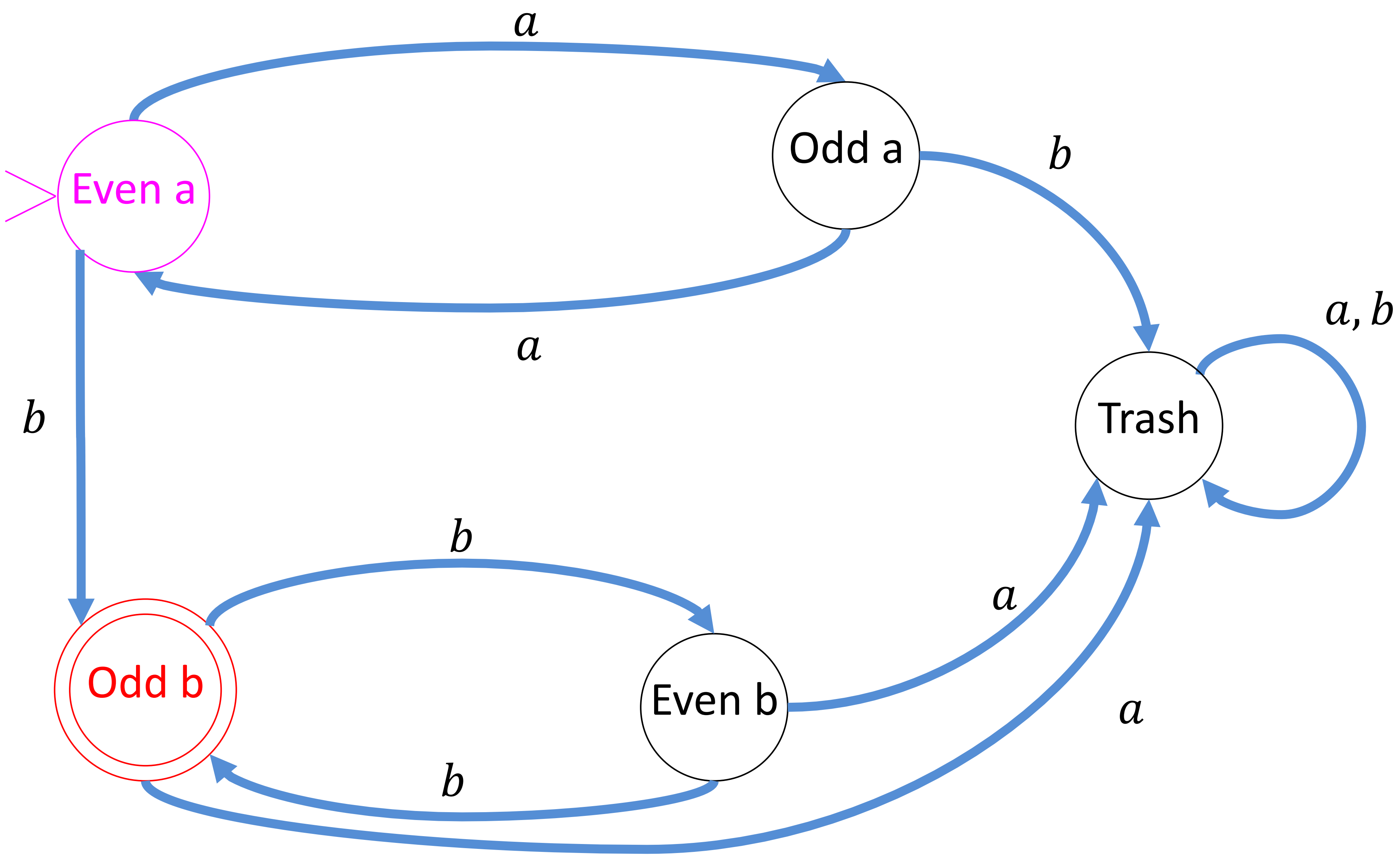


EvenAOddB

- Let's make a finite state automaton which accepts strings that have an even number of a's followed by an odd number of b's (in that order)
- It should accept:
  - $b, bbb, aab, aaaabbbbb, \dots$
- It should reject:
  - $bb, ab, baa, aba, aaabb$

EvenAoddB

Strings with an even number of a's followed by an odd number of b's



EvenAOddB using FSA

1. What's our alphabet? (pick Σ)
2. What should our states be? (pick Q)
3. Which states are the accept states? (pick F)
4. Which state is the start state? (pick q_0)
5. How should we transition? (pick δ)

Take-aways

- For a FSA M, the language of M (denoted $L(M)$) refers to the set of strings accepted by the machine
  - $L(M) = \{s \in \Sigma^* | M \text{ accepts } s\}$
- The set of all languages decided by some FSA is call the **Regular Languages**
  - Equivalent to the languages describable by regular expressions
- A particular language decided by some FSA is called a **Regular Language**
- All regular languages can be decided by a Java program using only constant memory (relative to length of word)

Closure Properties

- A set is **closed** under an operation if applying that operation to members of the set results in a member of the set
  - Integers are closed under addition
  - Integers are not closed under division
  - Σ^* is closed under concatenation
  - The set of all languages are not closed under cross product

Closure Properties of Regular Languages

- Complement
- Intersection
- Union
- Difference
- Concatenation

Closed under Complement

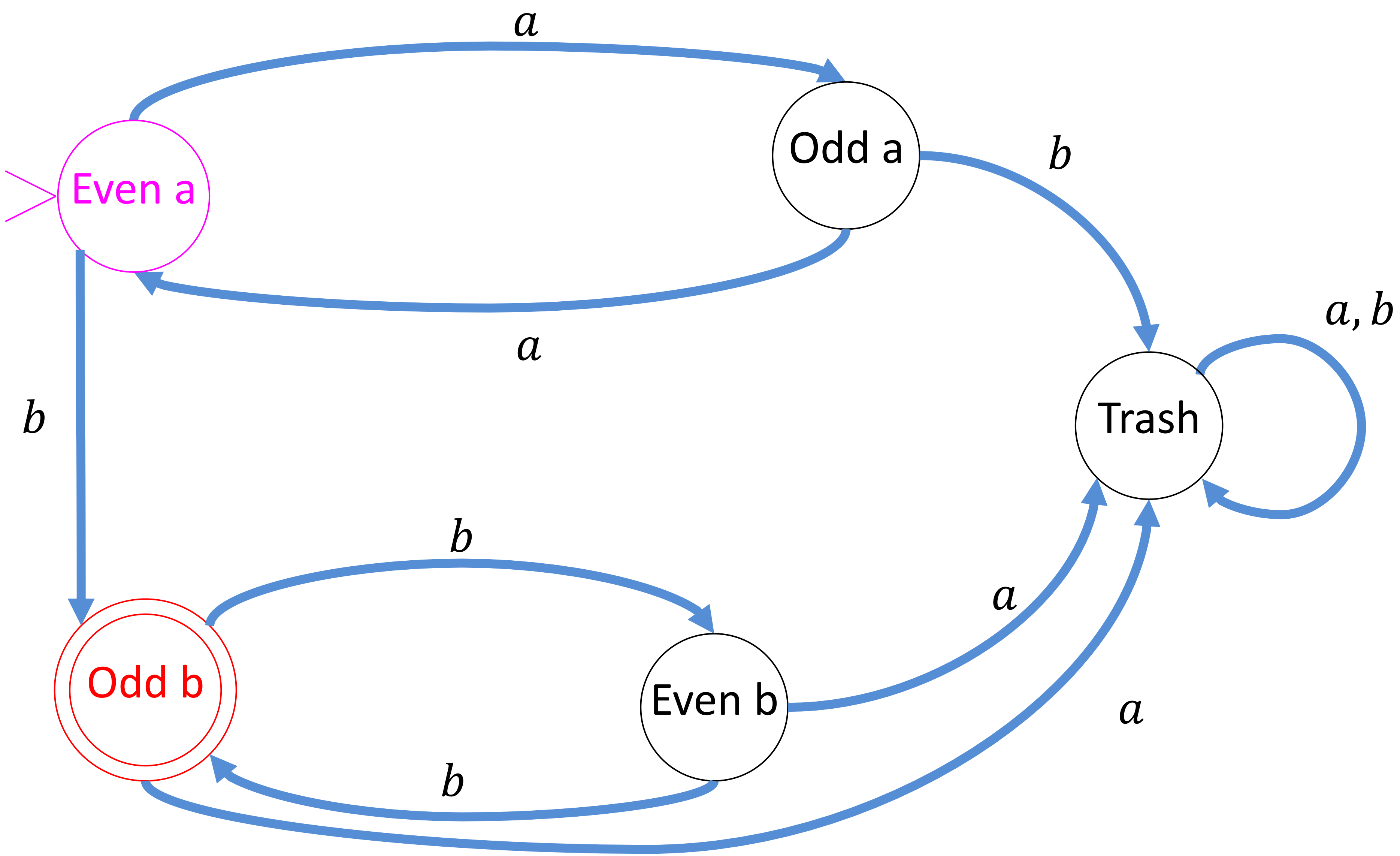
- If a language is regular then its complement is regular
- If a language has a FSA, it’s complement does as well
- If there is a FSA which accepts exactly the strings in the language, there is a FSA which accepts exactly the strings not in the language

Closed under complement

- Idea: Every string ends in some state. If that was originally an accept state then reject, else accept.
- New final states are the old non-final states

EvenAoddB

Strings with an even number of a's followed by an odd number of b's



Complement of EvenAoddB