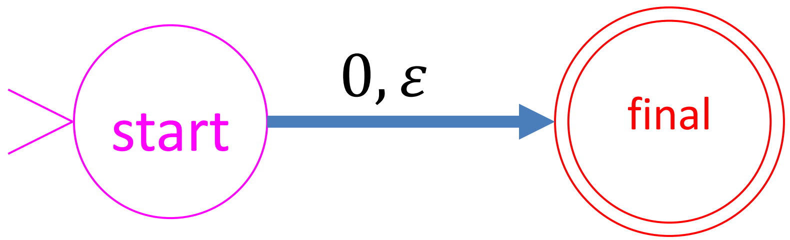
CS3102 Theory of Computation

Warm up:

What's the language of this NFA?



Logistics

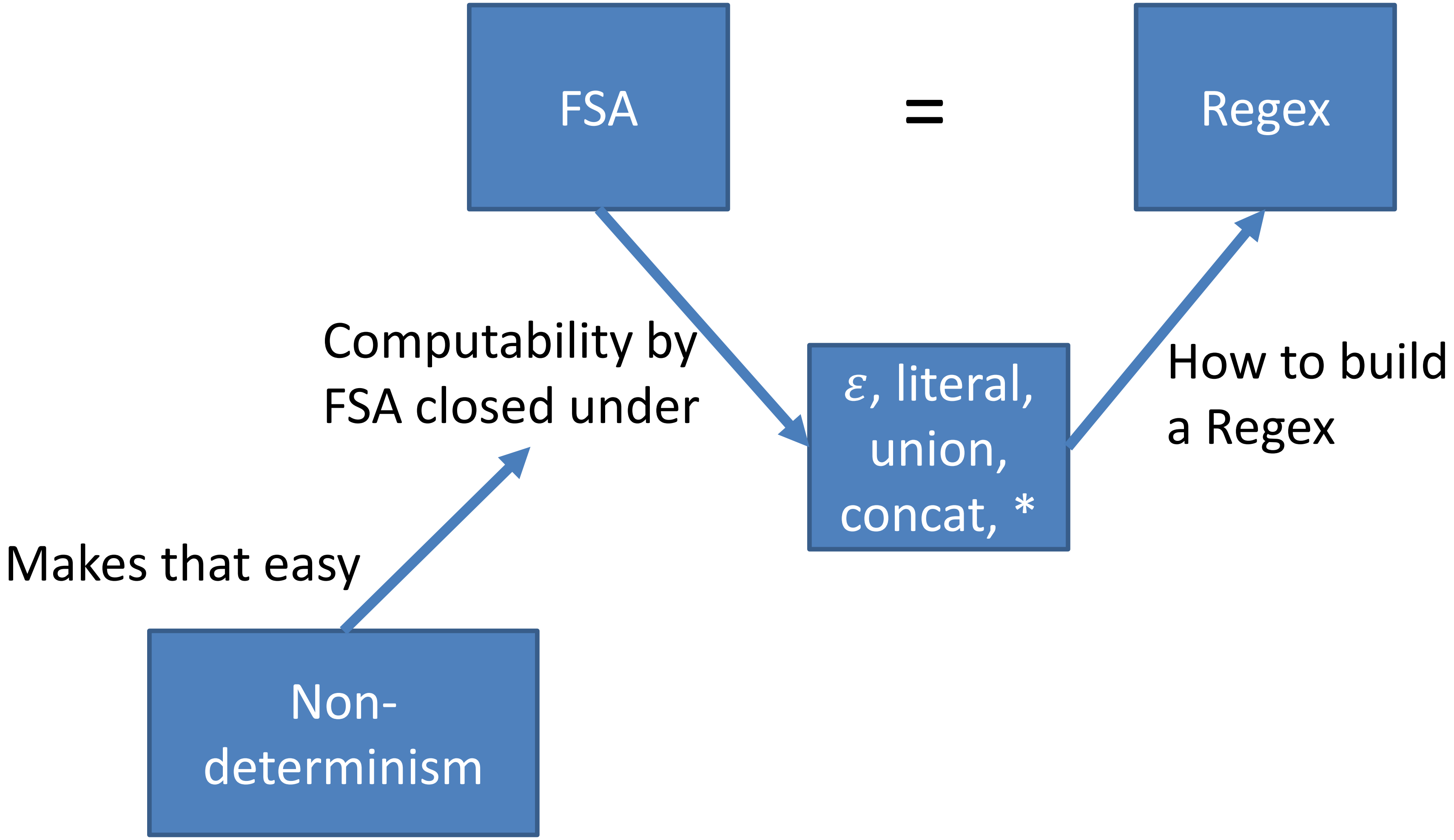
Last Time

- Non-determinism

Showing Regex $\leq$ FSA

- Show how to convert any regex into a FSA for the same language
- Idea: show how to build each "piece" of a regex using FSA

Proof “Storyboard”



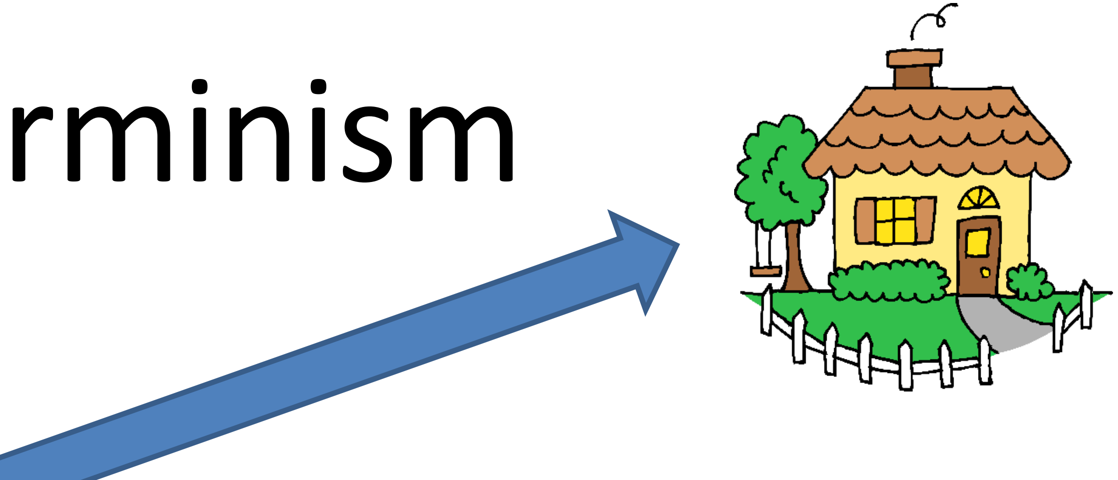
“Pieces” of a Regex

- ~~Empty String:~~
  - ~~Matches just the string of length 0~~
  - ~~Notation: ε or “”~~
- ~~Literal Character~~
  - ~~Matches a specific string of length 1~~
  - ~~Example: the regex a will match just the string a~~
- ~~Alternation/Union~~
  - ~~Matches strings that match at least one of the two parts~~
  - ~~Example: the regex $a|b$ will match a and b~~
- Concatenation
  - Matches strings that can be dividing into 2 parts to match the things concatenated
  - Example: the regex $(a|b)c$ will match the strings ac and bc
- Kleene Star
  - Matches strings that are 0 or more copies of the thing starred
  - Example: $(a|b)c^*$ will match a, b, or either followed by any number of c’s

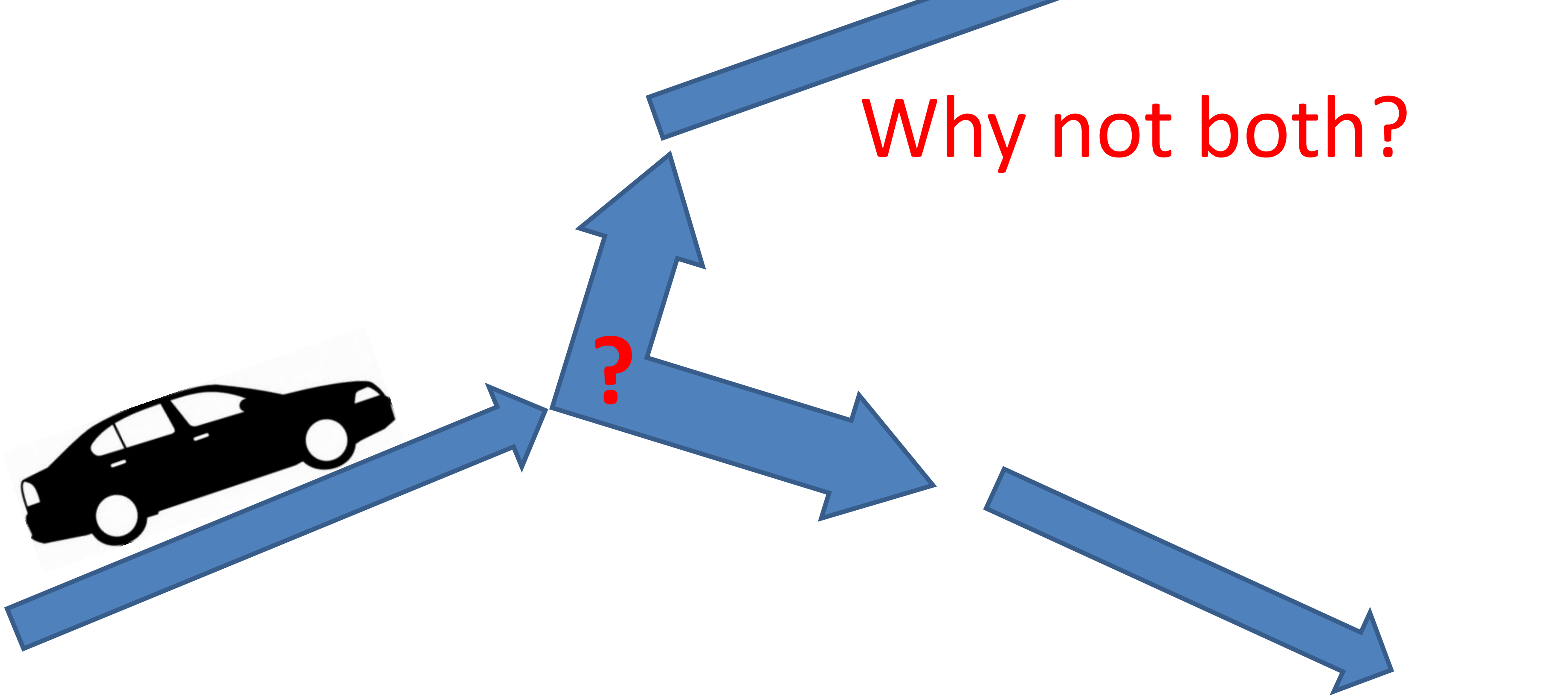
Non-determinism

- Things could get easier if we “relax” our automata
- So far:
  - Must have exactly one transition per character per state
  - Can only be in one state at a time
- Non-deterministic Finite Automata:
  - Allowed to be in multiple (or zero) states!
  - Can have multiple or zero transitions for a character
  - Can take transitions without using a character
  - Models parallel computing

Nondeterminism



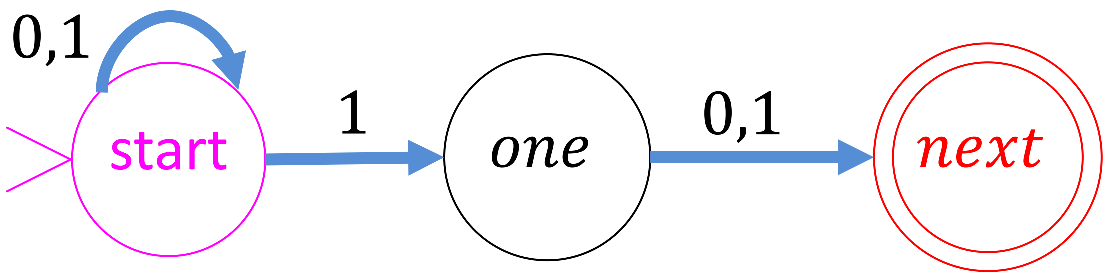
Driving to a friend's house
Friend forgets to mention a fork in the directions
Which way do you go?



Why not both?

Example Non-deterministic Finite Automaton

- $SecondLast1 =$
  $\{w \in \{0,1\}^* \mid \text{the second from last character is a } 1\}$



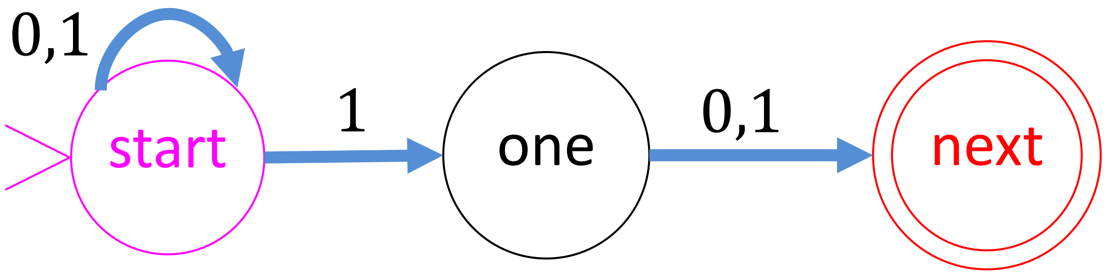
Non-Deterministic Finite State Automaton

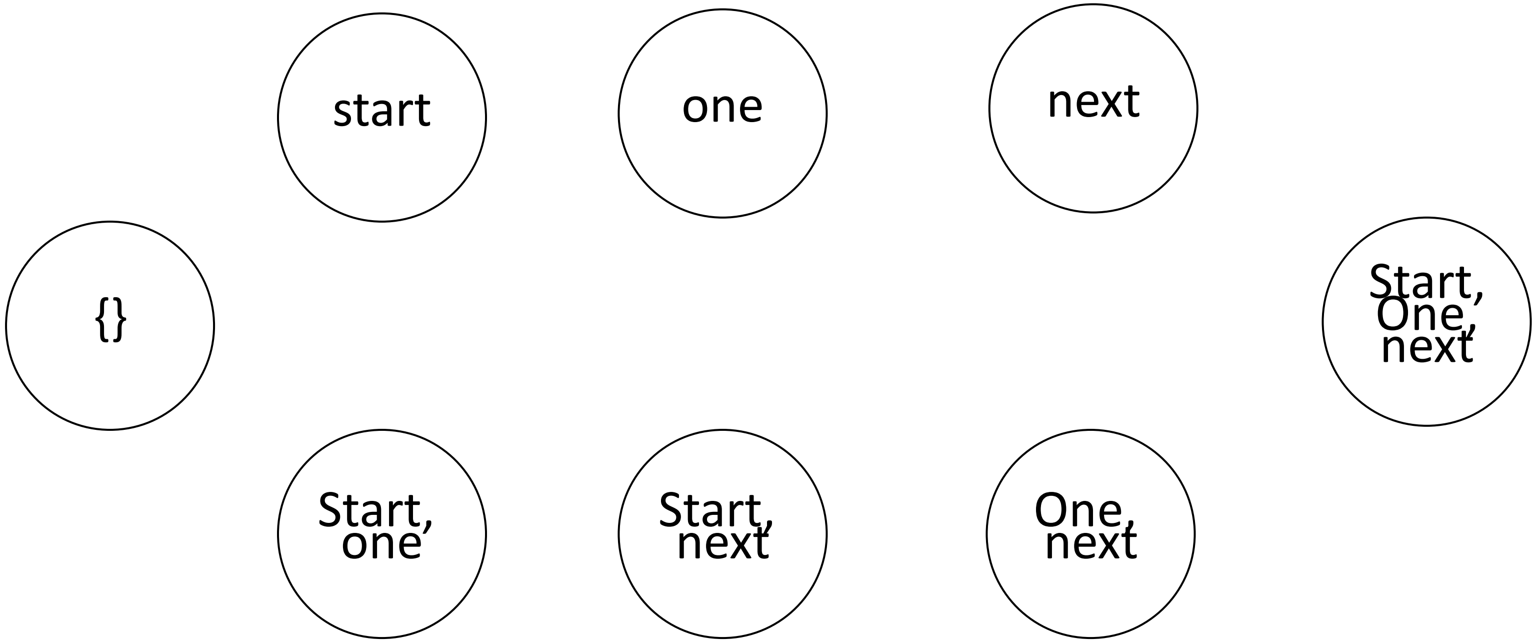
- Implementation:
  - Finite number of states
  - One start state
  - “Final” states
  - Transitions: (partial) function mapping **state-character (or epsilon) pairs to sets of states**
- Execution:
  - Start in the initial “state”
  - **Enter every state reachable without consuming input (ε-transitions)**
  - Read each character once, in order (no looking back)
  - Transition to new **states** once per character (based on current states and character)
  - **Enter every state reachable without consuming input (ε-transitions)**
  - Return True if **any** state you end is final
    - Return False if **every** state you end in is non-final

NFA = DFA

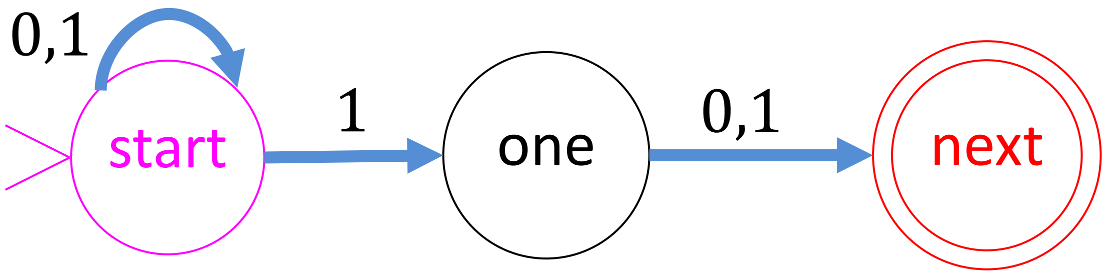
- DFA $\leq$ NFA:
  - Can we convert any DFA into an NFA?
- NFA $\leq$ DFA:
  - Can we convert any NFA into a DFA?
  - Strategy: NFAs can be in any subset of states, make a DFA where each state represents a set of states

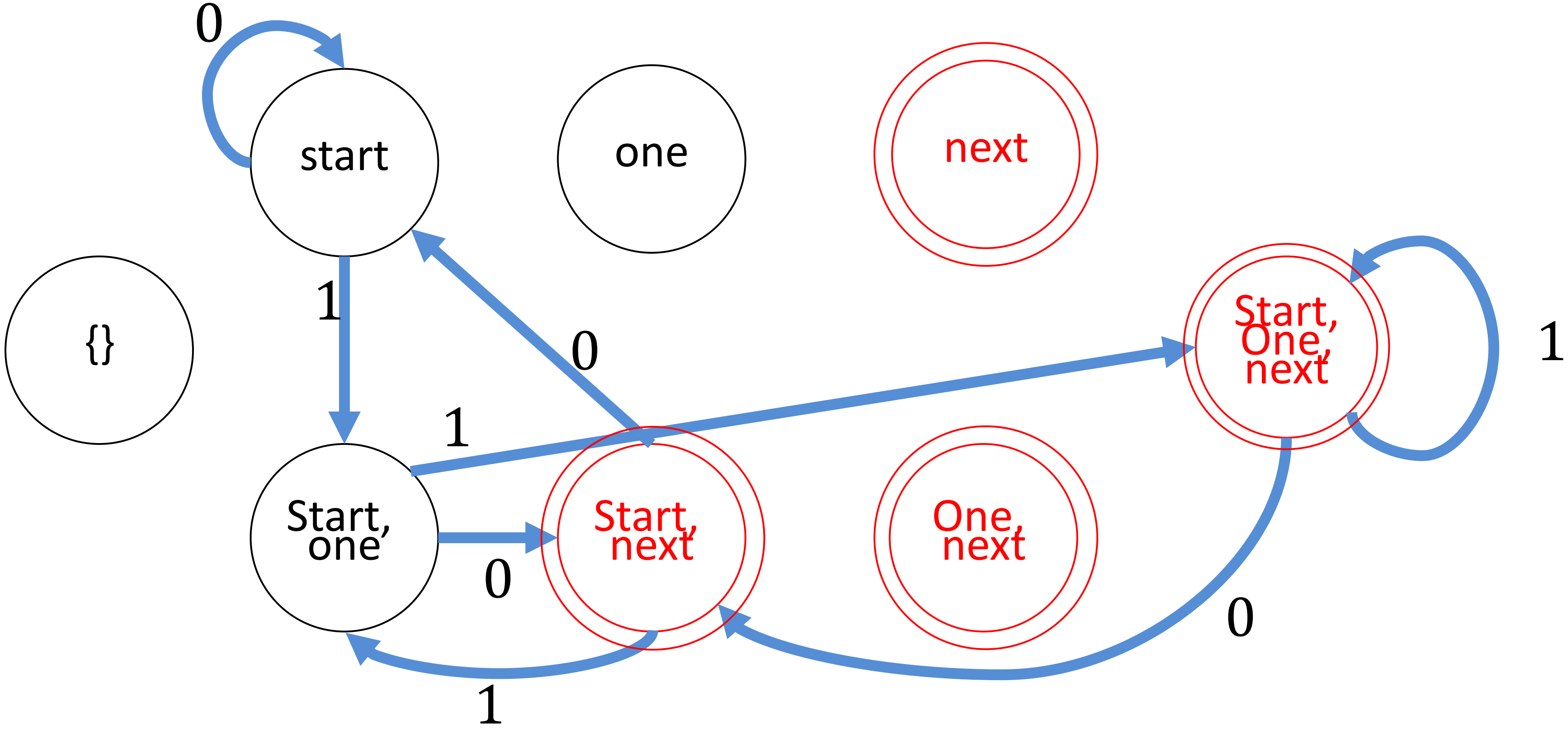
Powerset Construction





Powerset Construction

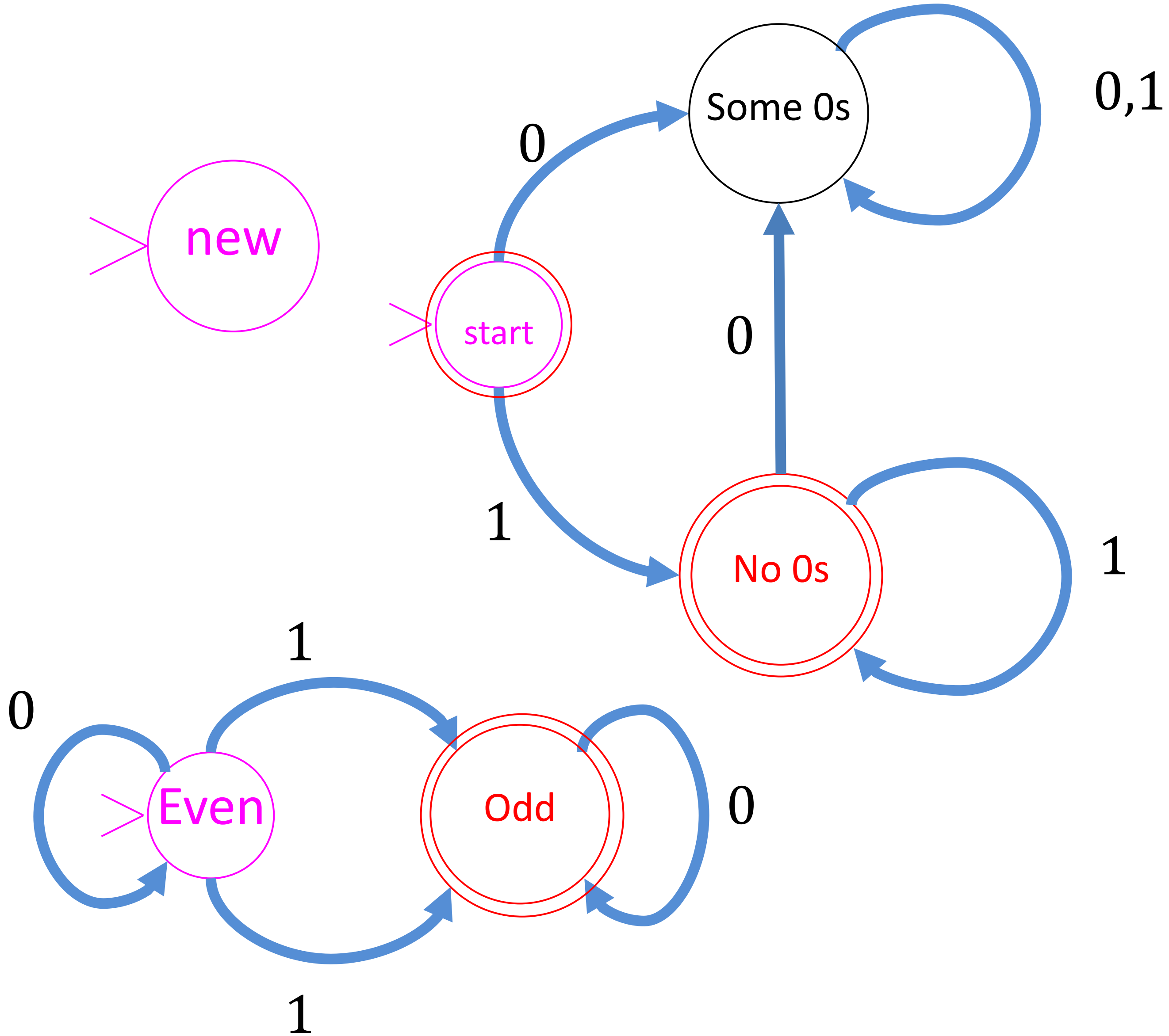




Powerset Construction (symbolic)

- NFA $M = (Q, \Sigma, \delta, q_0, F)$
- As a DFA:
  - $M_D = (2^Q, \Sigma, \delta_D, q_D, F_D)$
    - $q_D = \{q_0\} \cup \delta(q_0, \varepsilon)$
      - start state and everything reachable using empty transitions
    - $F_D = \{s \in 2^Q | \exists q \in s\,.\,q \in F\}$
      - All states with a start state in them
    - $\delta_D(s, \sigma) = \bigcup_{q \in s} \delta(s, \sigma)$
      - Transition to the stateset of everything any current state transitions to

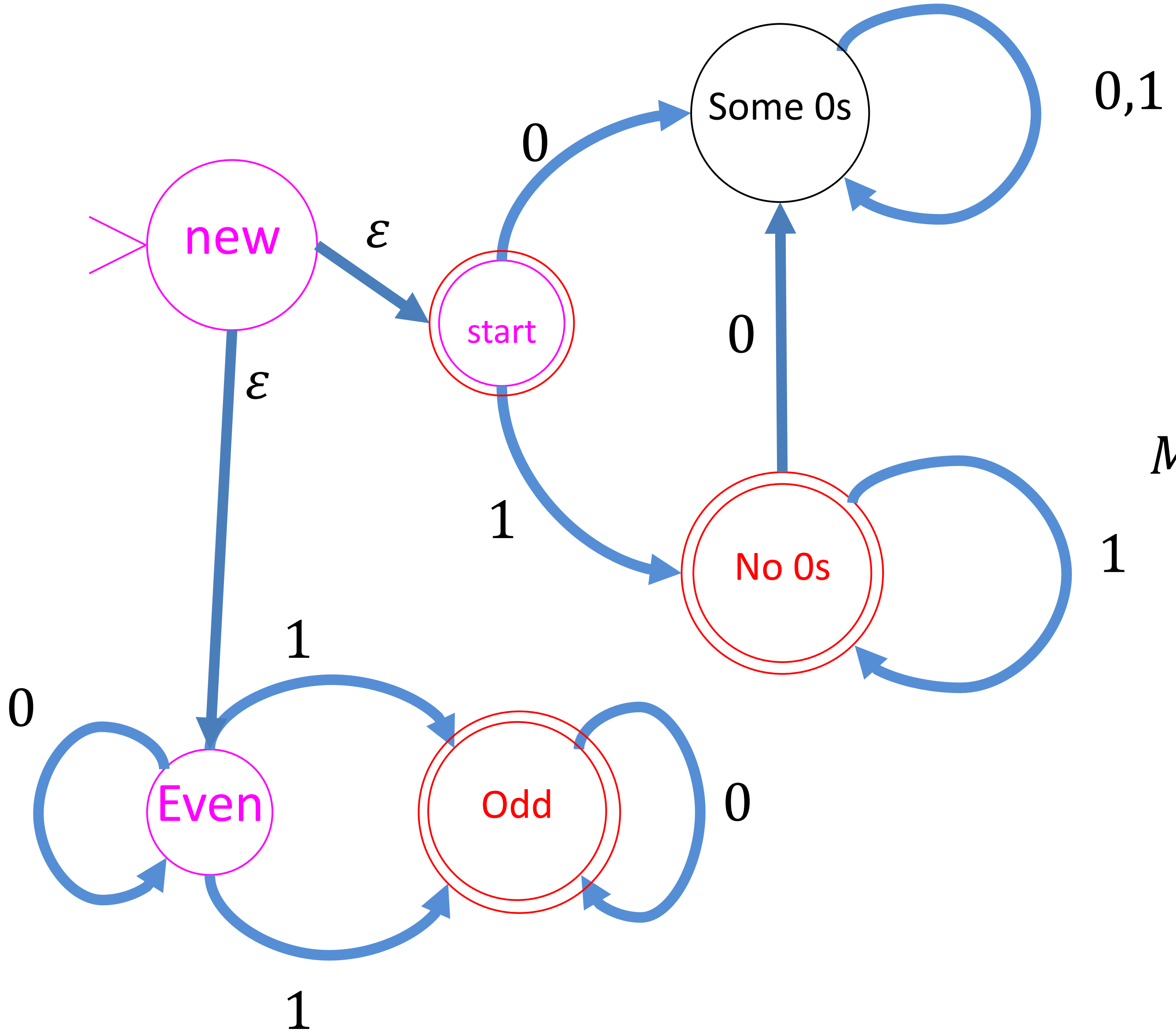
Union Using Non-Determinism

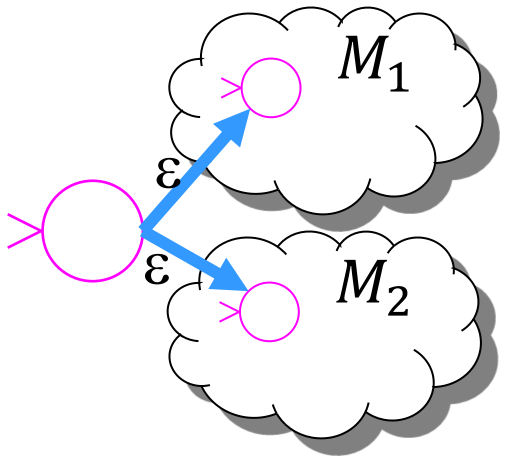


Goal: Return 1 if either machine returns 1

Strategy: Run both machines in parallel (non-deterministically) by transitioning to the start states for both without using input

Union Using Non-Determinism





$$M_\cup = (Q_1 \cup Q_2 \cup \{new\}, \Sigma, \delta_\cup, new, F_1 \cup F_2)$$

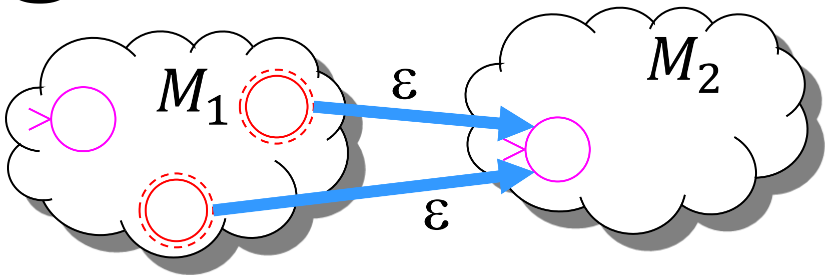
$$\delta_\cup(q,\sigma) = \begin{cases} \{\delta_1(q,\sigma)\} \text{ if } q \in Q_1 \\ \{\delta_2(q,\sigma)\} \text{ if } q \in Q_2 \end{cases}$$

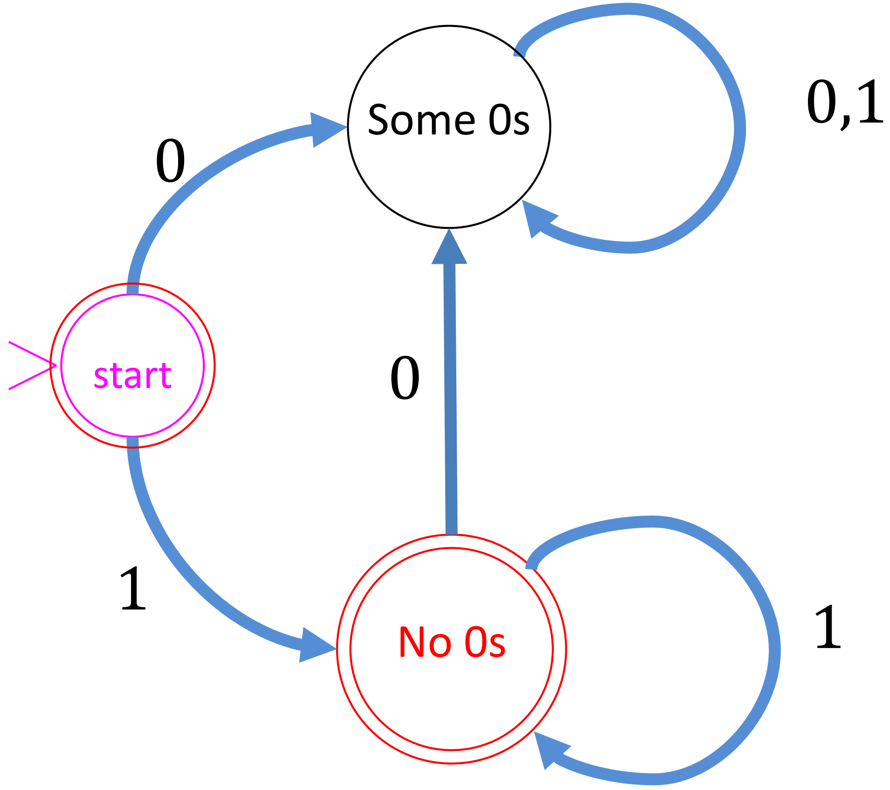
$$\delta_\cup(new, \varepsilon) = \{start, even\}$$

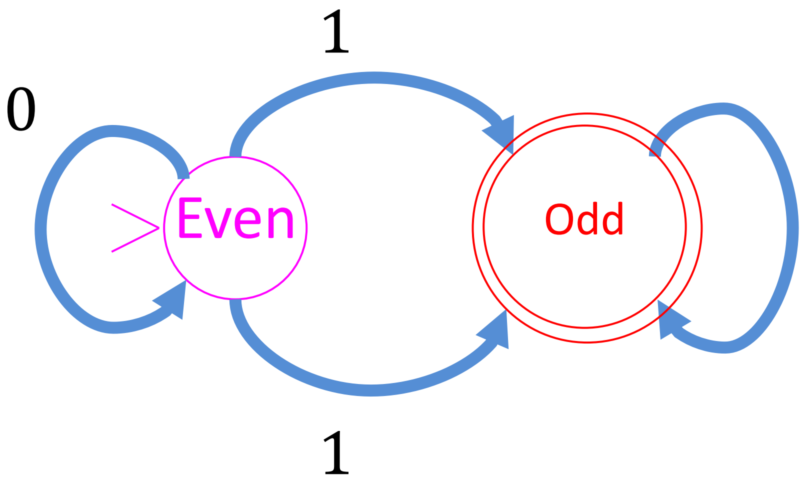
Language Concatenation

- $L_1L_2 = \{w \in \Sigma^* | \exists x \in L_1, \exists y \in L_2. xy = w\}$
- The set of all strings I can create by concatenating a string from L_1 with a string from L_2 (in that order)
- $\{\varepsilon, 0, 10\} \cdot \{0,00\} = \{0, 00, 000, 100, 1000\}$

Concatenation using NFA



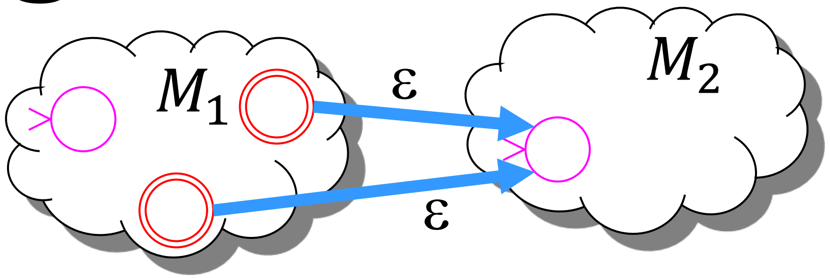




Goal: Return 1 if the input can be broken into 2 chunks, M_1 returns 1 on the first chunk, M_2 on the second

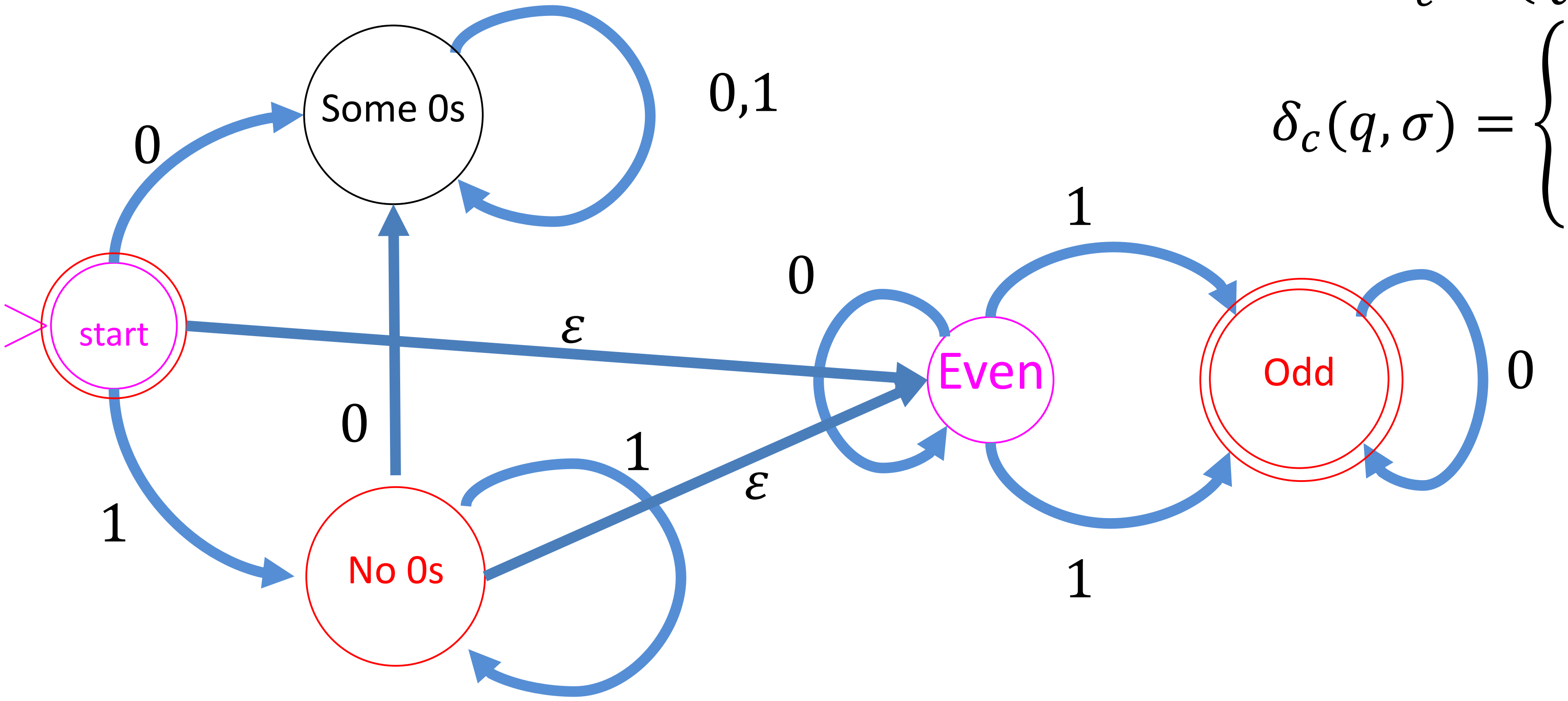
Strategy: Every time we enter a final state in M_1, non-deterministically run the rest of the string on M_2. Return 1 if M_2 does.

Concatenation using NFA



$$M_c = (Q_1 \cup Q_2, \Sigma, \delta_c, q_{01}, F_2)$$

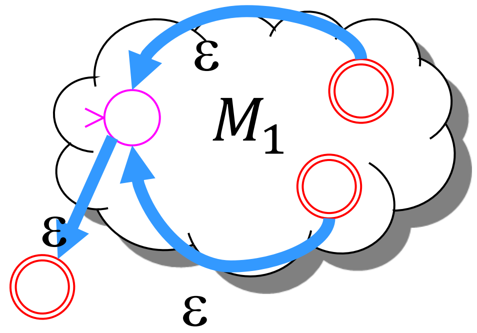
$$\delta_c(q,\sigma) = \begin{cases} \{\delta_1(q,\sigma)\} \text{ if } q \in Q_1 - F_1 \\ \{\delta_1(q,\sigma), q_{0,2}\} \text{ if } q \in F_1 \\ \{\delta_2(q,\sigma)\} \text{ if } q \in Q_2 \end{cases}$$

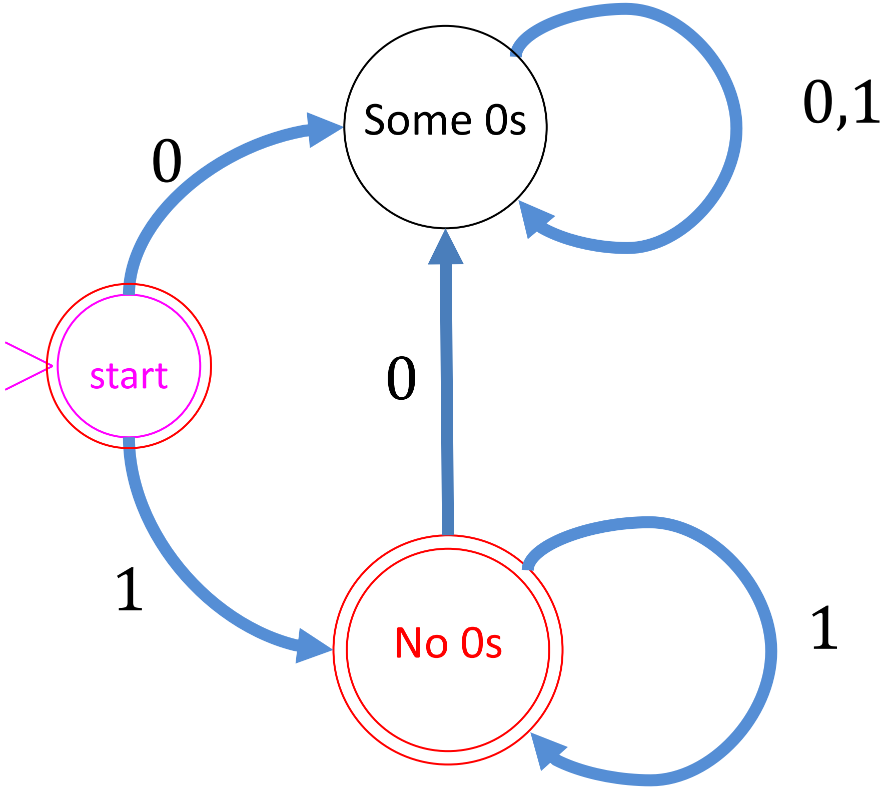


Kleene Star

- $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$
- $L^0 = \{\varepsilon\}$
- $L^k = (L \text{ concatenated } k \text{ times})$
- $\{00, 11\}^* = \{\varepsilon, 00, 11, 0011, 1100, 0000, 1111, 110011, \ldots\}$

Kleene Star using NFA

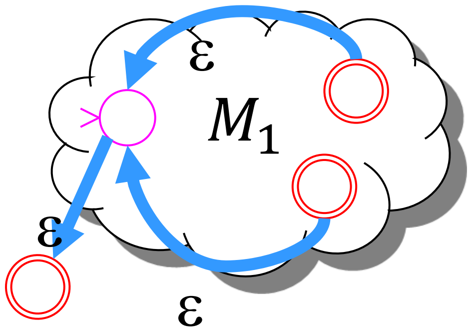


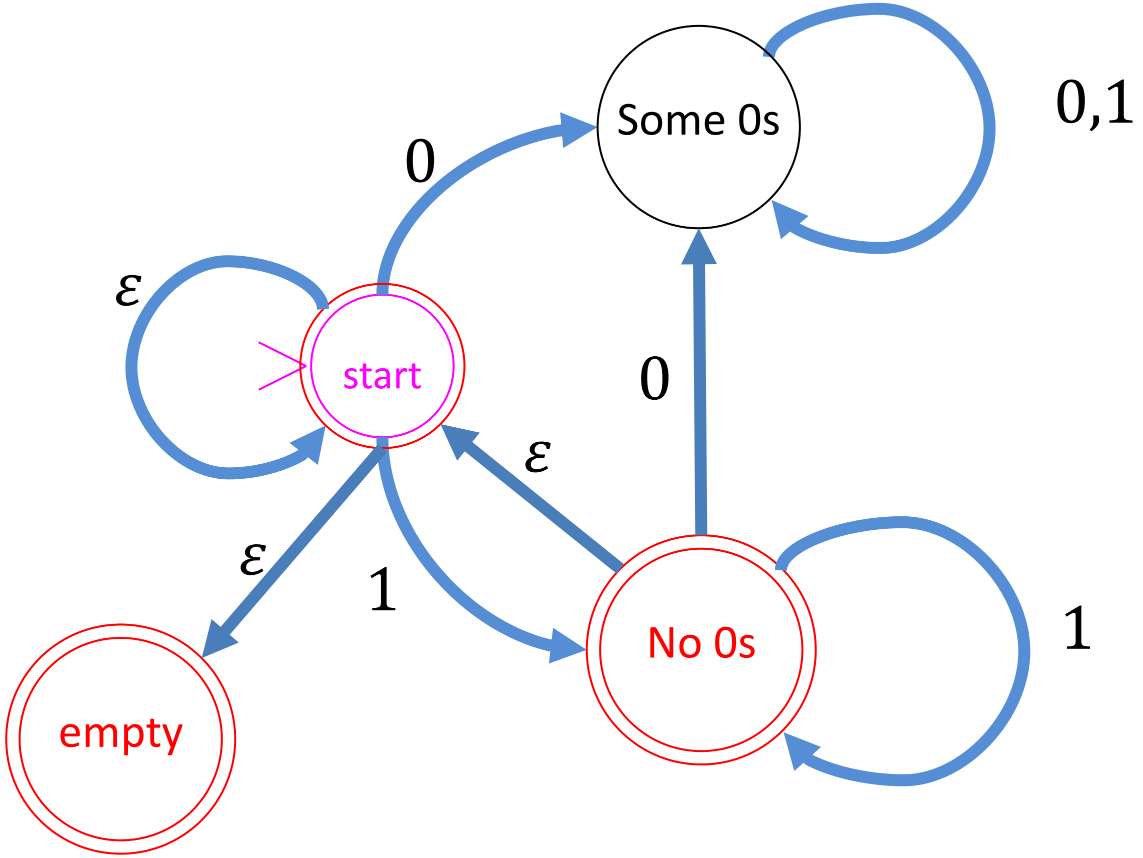


Goal: Return 1 if the input can be broken into chunks such that M_1 returns 1 for every chunk

Strategy: Every time we enter a final state in M_1, non-deterministically "restart" the machine to run on the rest of the string, make sure we return 1 on ε

Kleene Star using NFA





Goal: Return 1 if the input can be broken into chunks such that M_1 returns 1 for every chunk

Strategy: Every time we enter a final state in M_1, non-deterministically "restart" the machine to run on the rest of the string, make sure we return 1 on ε

Conclusion

- Any language expressible as a regular expression is computable by a NFA
- Any language computable by a NFA is computable by a DFA
- NFA = Regex = DFA
- Call any such language a “regular language”

Characterizing What's computable

- Things that are computable by FSA:
  - Functions that don't need "memory"
  - Languages expressible as Regular Expressions
- Things that aren't computable by FSA:
  - Things that require more than finitely many states
  - Intuitive example: Majority

Majority with FSA?

- Consider an inputs with lots of 0s

000...0000 111...1111	000...0000 111...1111	000...0000 111...1111
×49,999 ×50,000	×50,000 ×50,000	×50,000 ×50,001

- Recall: we read 1 bit at a time, no going back!
- To count to 50,000, we'll need 50,000 states!