Generating Lexers

Lexer generators

Spec = RE1 { Action1 }
        | RE2 { Action2 }
        | RE3 { Action 3 }
          ... // in decreasing priority

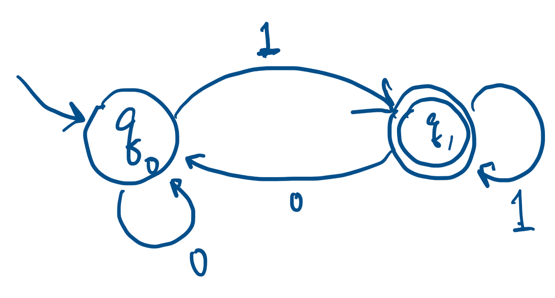
Idea: Spec --> NFA --> DFA --> code = loop + table

Review: DFA

machine with finite # of states
four-tuple $(q0, \Sigma, \mathrm{F}, \delta)$

final states

$\delta(q, x) = q'$



DFA for recognizes odd bin numbers

Easy to implement efficiently

```
q = q0
while (i ≤ n) {
    q = δ(q, input[i])
    ++i
}
if (q ∈ F) return accept
else return fail
```

How to generate δ from regexes?

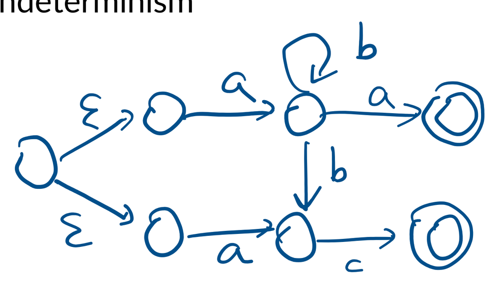
R1 | R2 | ...   R1 IF
                     R2 identifiers

"i"

Idea: NFA, and then NFA --> DFA

Review: NFA

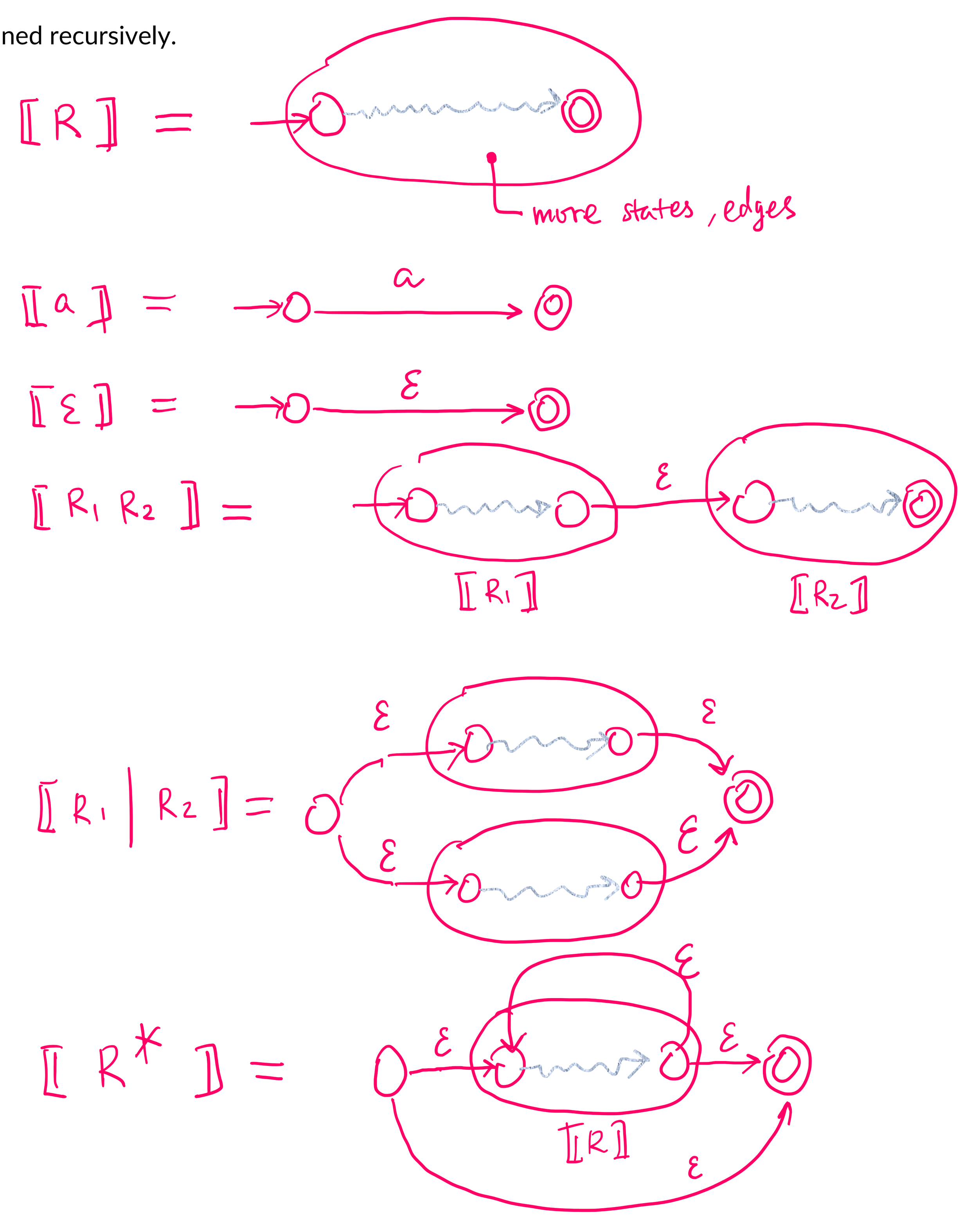
DFA + ...
- arrows can be ε
- exit arrows from a state can share same input symbol
- angelic nondeterminism



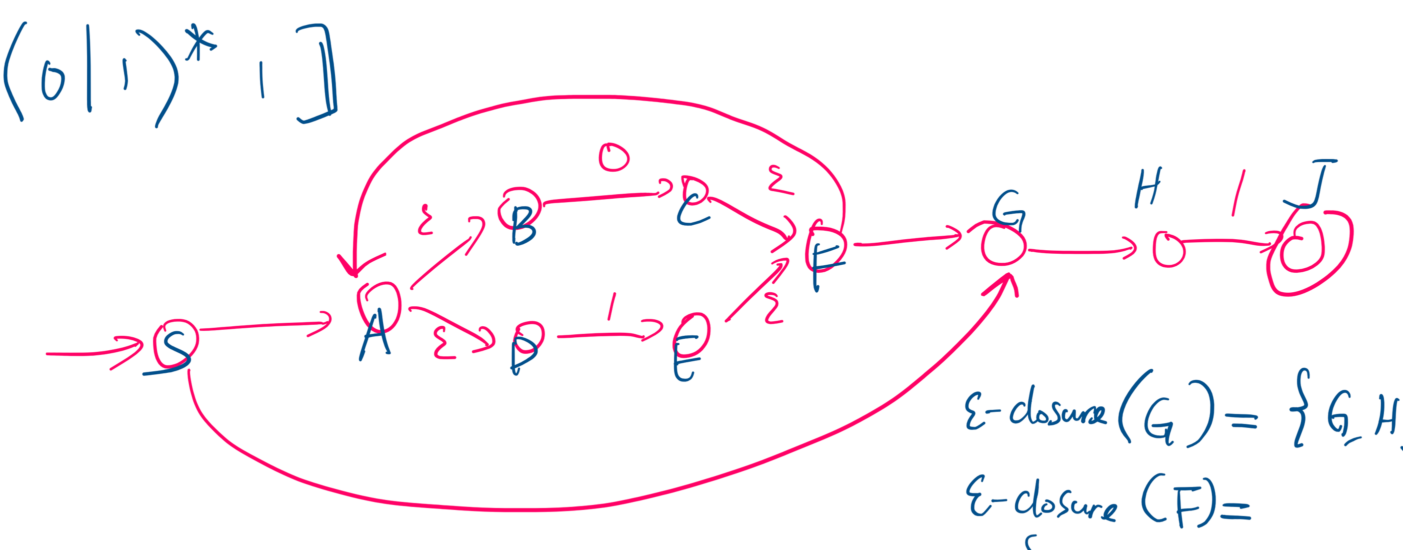
abba ✓
abac ✗

RE --> NFA

Defined recursively.



$[\![(0|1)^* 1]\!]$



$\varepsilon\text{-closure}(G) = \{G, H\}$

$\varepsilon\text{-closure}(F) = \{F, G, H, A, B, D\}$

NFA --> DFA

Idea: a DFA state = set of all NFA states that can possibly be reached by reading the same sequence of input symbols.

ε-closure(q) = Set of states reachable from q using zero or more ε edges
Inductively defined.

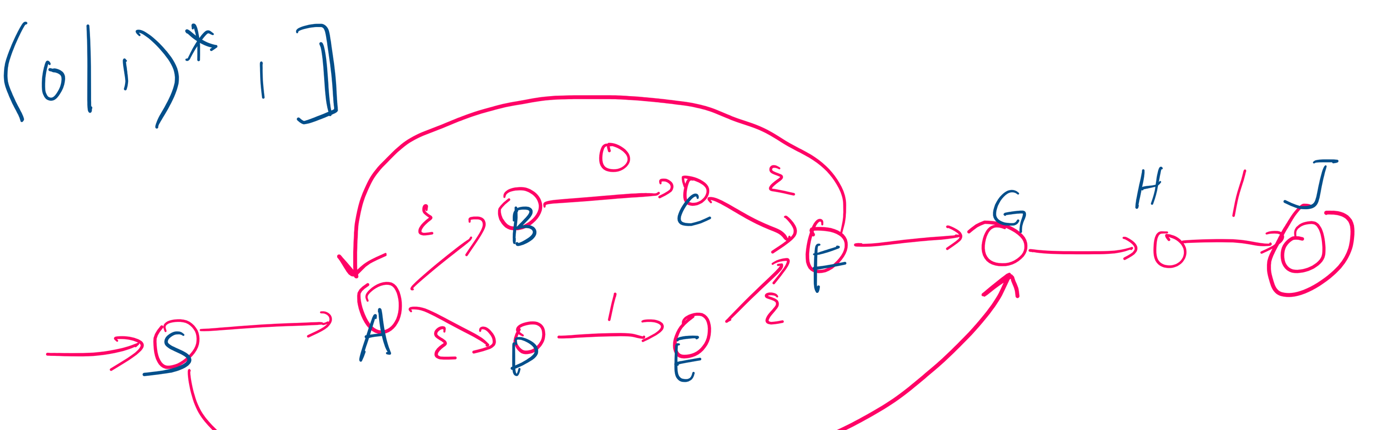
$$\frac{}{q \in \varepsilon\text{-closure}(q)} \qquad \frac{q'' \in \varepsilon\text{-closure}(q) \quad q'' \xrightarrow{\varepsilon} q'}{q' \in \varepsilon\text{-closure}(q)}$$

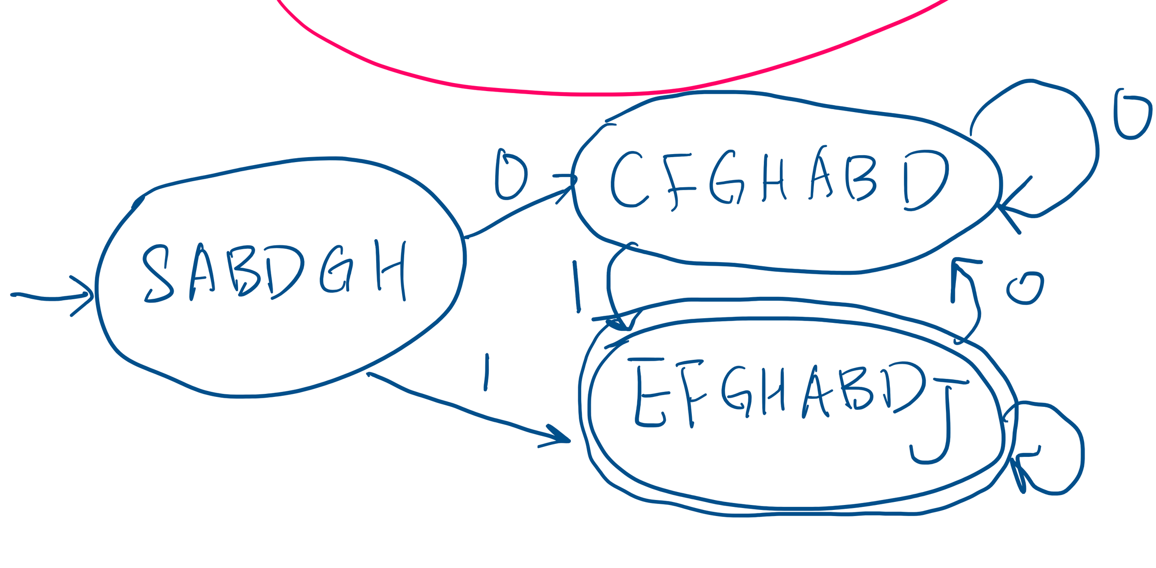
Construct DFA from NFA:

Idea: a DFA state = set of all NFA states that can possibly be reached by reading the same sequence of input symbols

- Initial states: $Q_0 = \varepsilon\text{-closure}(q_0)$
- States and transitions: $\delta(Q, x) = \varepsilon\text{-closure}\left(\bigcup_{q \in Q} \{q' \mid q \xrightarrow{x} q'\}\right)$
- Final state: Q is a final state of the DFA if any $q \in Q$ is a final NFA state

$[\![(0|1)^* 1]\!]$





Worklist algorithm for computing ε-closure(q):

$$\varepsilon\text{-closure}(q)[q'] \in \mathbb{B}$$

$$\varepsilon\text{-closure}(q)[q'] = \text{true} \quad \text{if } q = q'$$

$$\varepsilon\text{-closure}(q)[q'] = \bigvee_{q'' \xrightarrow{\varepsilon} q'} \varepsilon\text{-closure}(q)[q'']$$

```
ε-closure(q) = { q1 ↦ false, ... }
worklist = { q }  // empty on termination
                  // Invariant: equations about q' not satisfied
                  //   ⇒ q' ∈ worklist

while (worklist not empty) {
    remove q' from worklist
    update ε-closure(q)[q'] per equations
    if ε-closure(q)[q'] changed :
        for each q'' s.t. q' -ε-> q''
            add q'' to worklist
}
```

$O(N^2)$

N

Correctness?

Partial correctness ✓ + Termination ✓

DFA minimization

Some DFA states can be equivalent after conversion.

$$\frac{Q_1 \in \text{Final} \quad Q_2 \notin \text{Final}}{Q_1 \not\equiv Q_2} \qquad \frac{Q_1 \notin \text{Final} \quad Q_2 \in \text{Final}}{Q_1 \not\equiv Q_2} \qquad \frac{\delta(Q_1, x) \not\equiv \delta(Q_2, x)}{Q_1 \not\equiv Q_2}$$

Algorithm: start with every pair of states being equivalent, and unequate them.

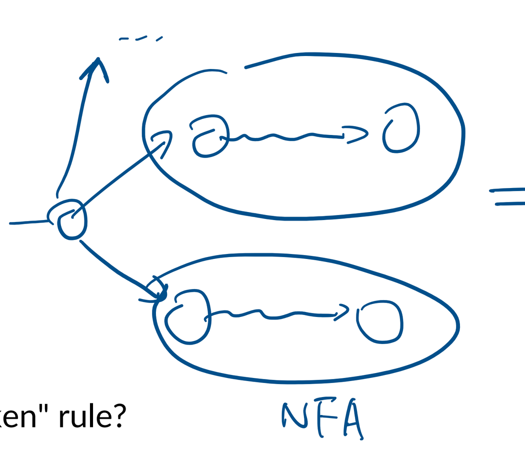
When to unequate two states?

Myhill–Nerode theorem: for any DFA, there is a unique minimal DFA that accepts the same input.

Putting it all together

Spec = R1 { Action1 }
        | R2 { Action2 }
        | R3 { Action 3 }
          ... // in decreasing priority

$\Rightarrow$



$\Rightarrow$ DFA $\Rightarrow$ code

How to implement the "Longest-matching token" rule?

$R1 = abc$

$R2 = (abc)^* d$

Input = abc | abc | ... abc x

$O(n^2)$

$n + n-1 + n-2 + \dots = O(n^2)$