Non-Deterministic Finite Automata

Mark Greenstreet, CpSc 421, Term 1, 2008/09

Lecture Outline

Non-Determinism

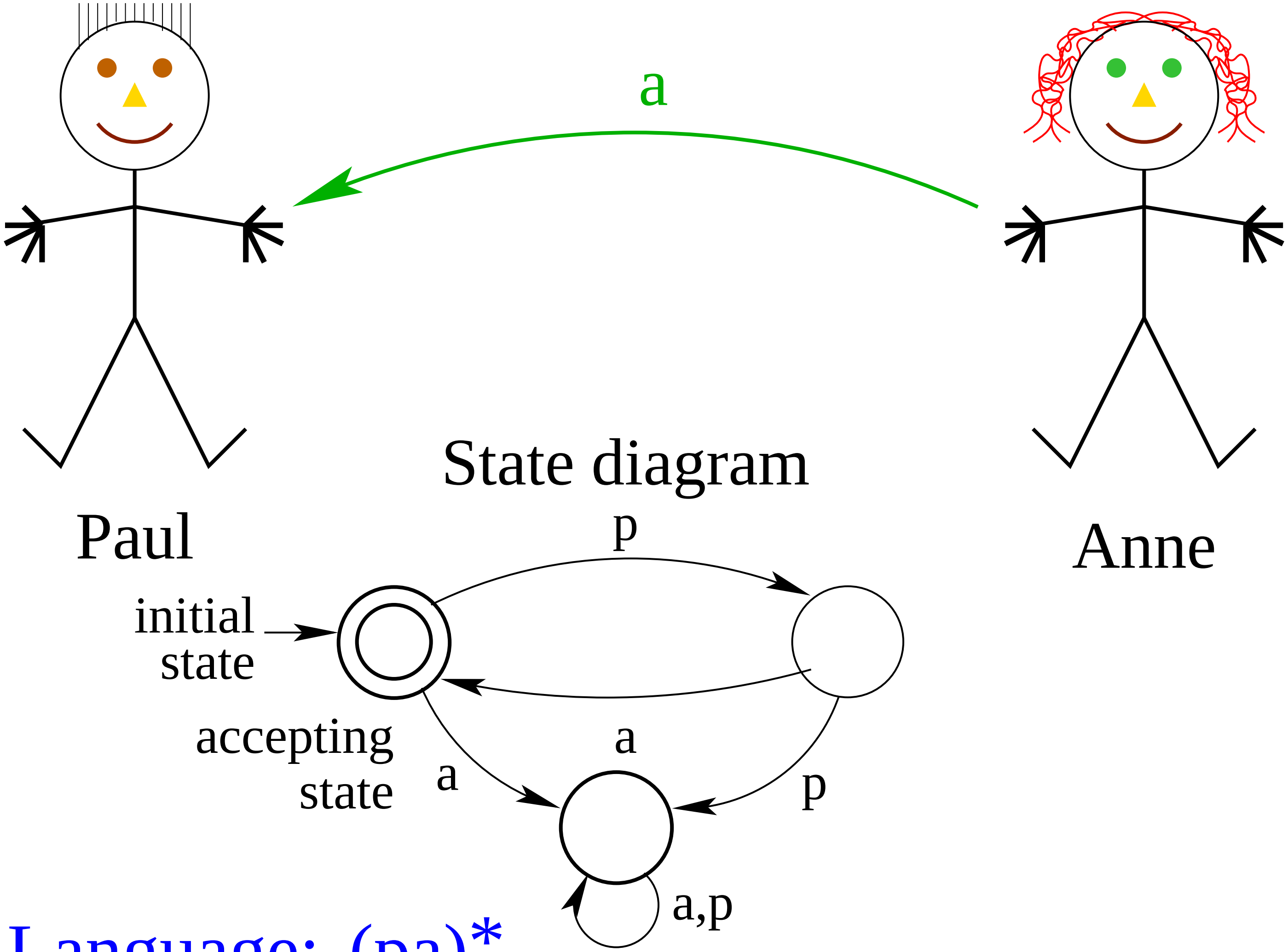
- Motivation
  - Modeling uncertainty
  - Network Protocol example
- Non-deterinisitic Finite Automata
  - Formal definition
  - Diagrams for NFAs
  - Examples

Uncertainty

When we write a program or a class or a protocol or a . . . , we must account for the fact that we don't know

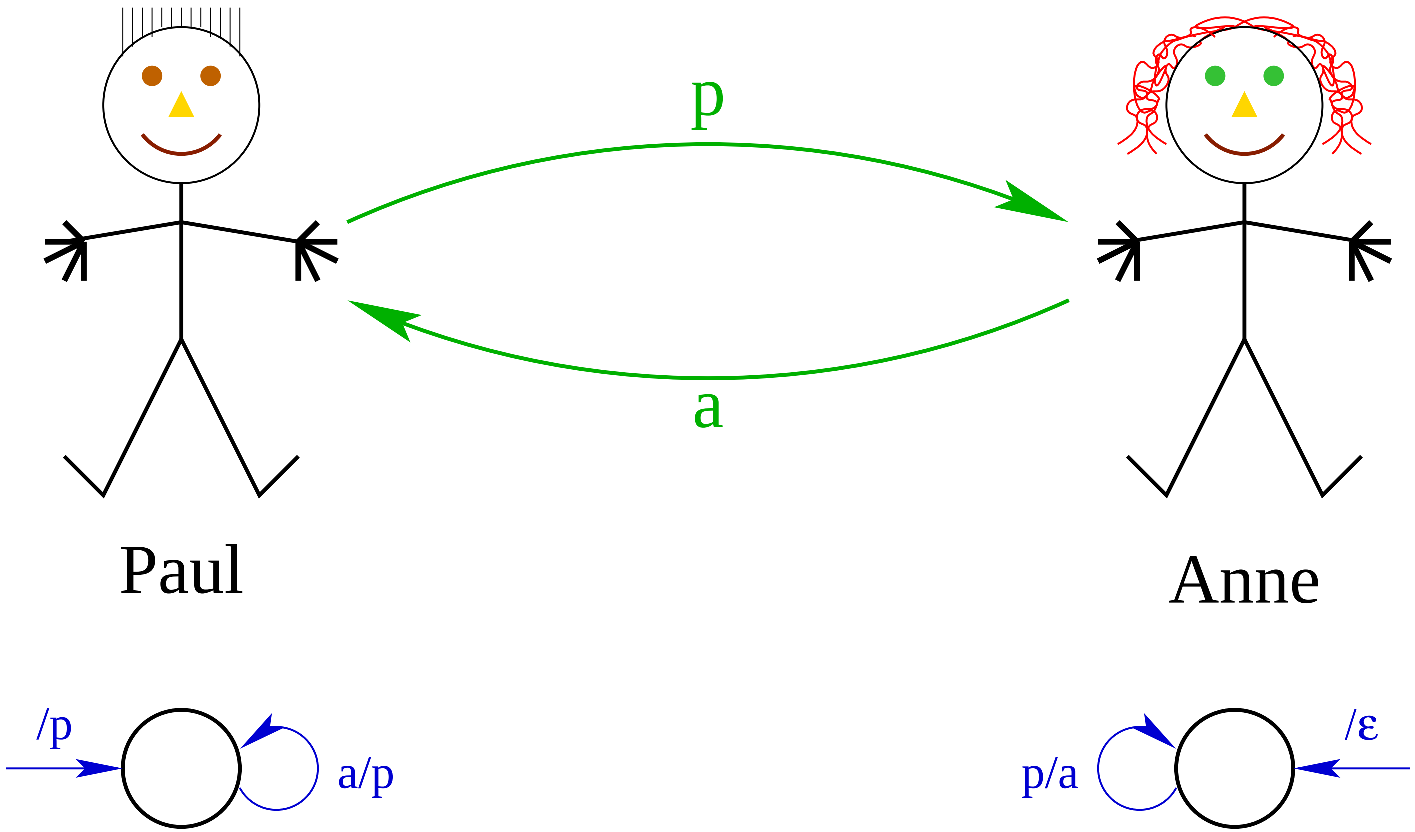
- What choices the user will make:
  - Select one of many menu options
  - Select one of many on-screen items
  - Type something on the keyboard
  - Do any of these while some other task is running and incomplete?
  - . . .
- How a class will be used:
  - What methods will be invoked in what order
  - If the code is multi-threaded, what other threads may be running concurrently,
  - . . .
- the order of events in a network
  - when a remote machine will respond
  - what requests we might get from clients
  - . . .

A Network Protocol (again)



Language: (pa)*

Finite State Transponders



a/p means ‘‘after receiving an ’a’ output a ’p’ and move to the indicated state.

Batch Acknowledgements

We can make a more efficient protocol by acknowledging groups of packets instead of individual packets.

For example, Anne could acknowledge every fourth packet.
This means Paul sends up four packets before waiting for an acknowledgement.

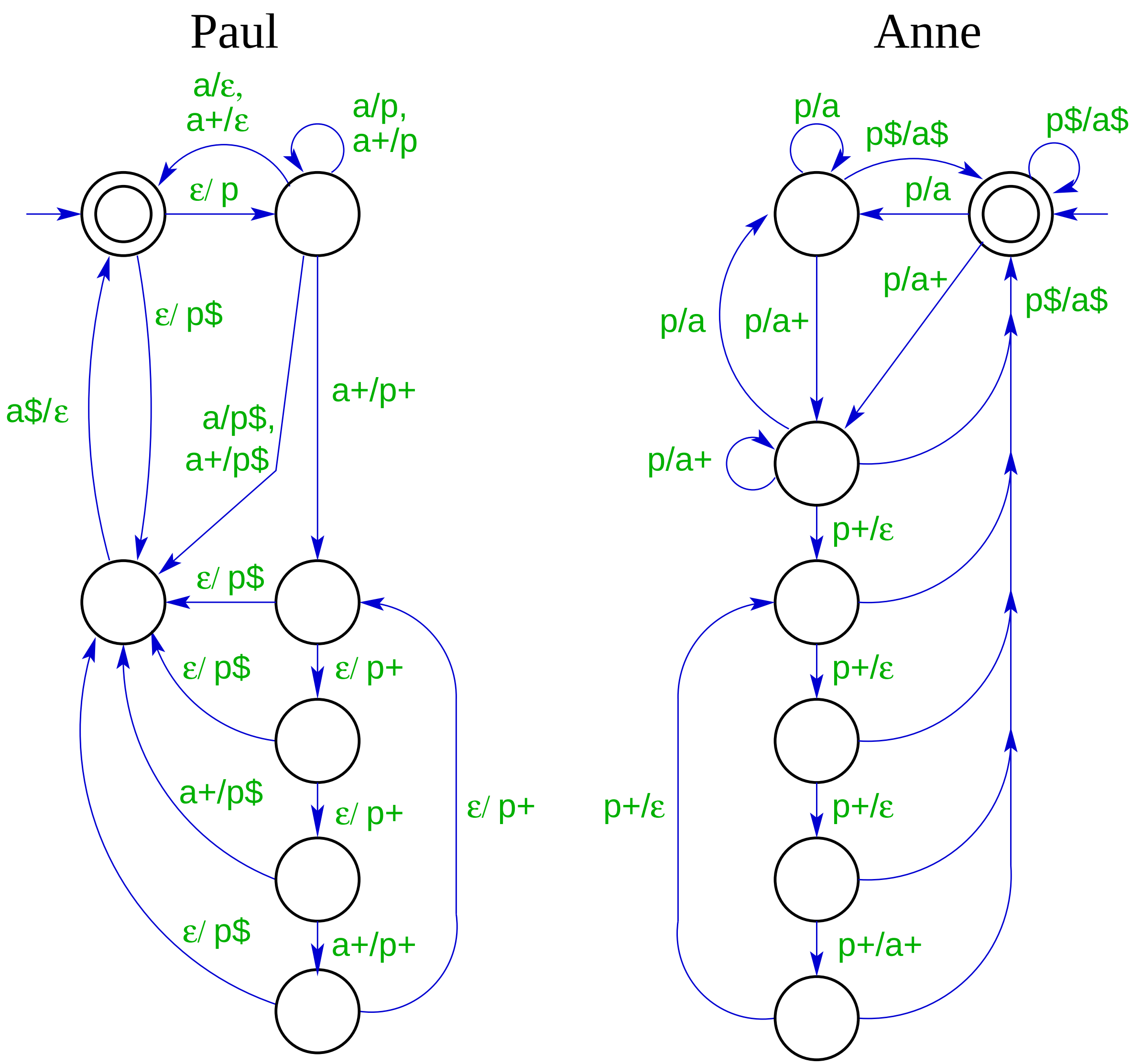
This only works if both Anne and Paul have enough memory to keep track of the number of unacknowledged packets.

We'll handle this by allowing Anne to send two kinds of acknolwedgements: a and a+. An a+ acknowledgement means she can handle batch acknowledgements.

Likewise, Paul can send two kinds of packets, p and p+, where a p+ can only be sent after receiving a a+ and indicates that Paul is assuming batch acknowledgements.

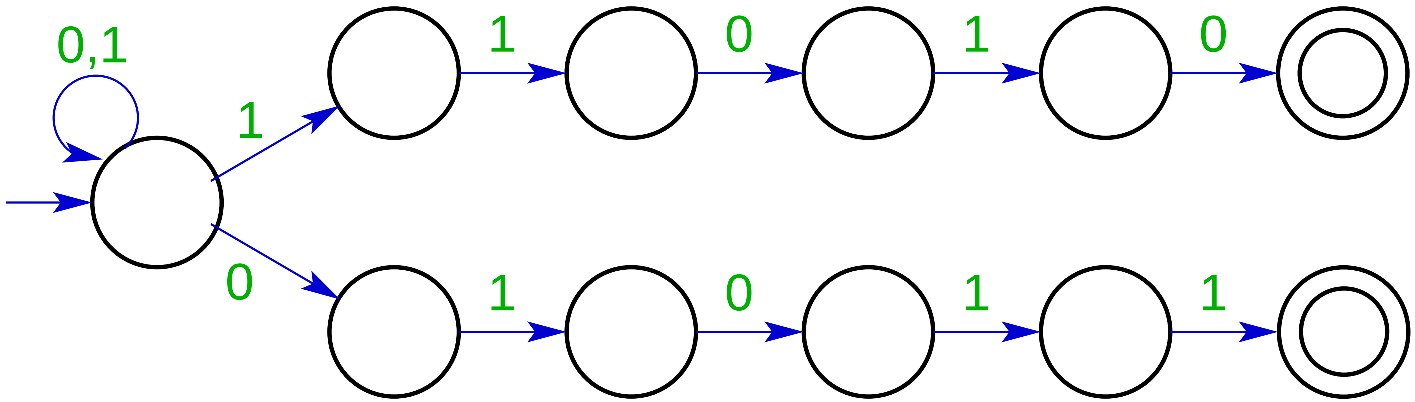
Finally, we'll add symbols p$ and a$. Paul sends p$ to indicate the last packet – note that the total number of packets sent is not necessarily a multiple of four. Anne sends a a$ to acknowledge the last packet.

Batch Acknowledgements

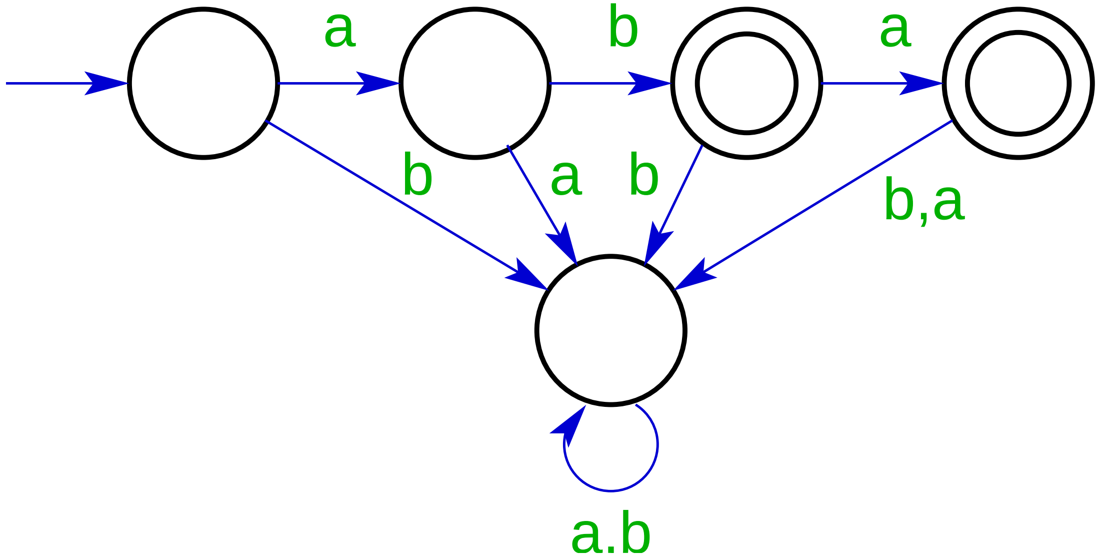


A Simple NFA

A Non-deterministic Finite Automaton (NFA) that recognizes all strings that end `01011` or `11010`.

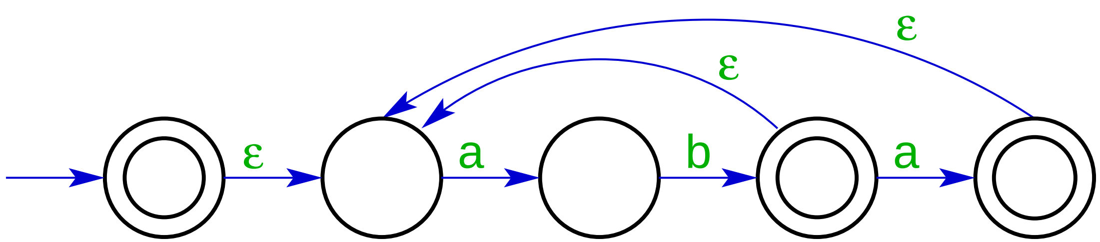


A DFA that recognizes {**ab**, **aba**}



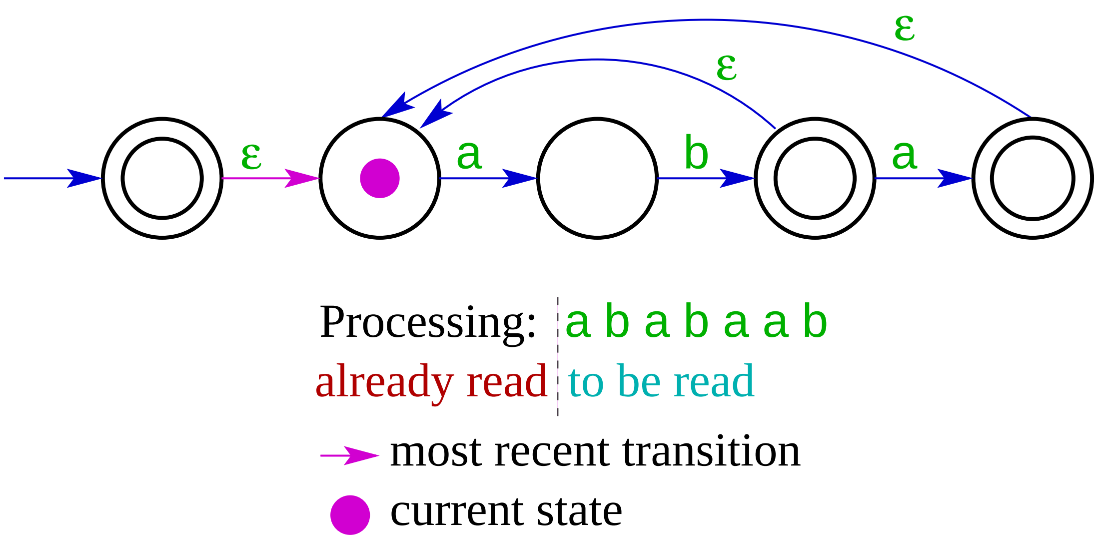
A NFA that recognizes $\{\mathbf{ab}, \mathbf{aba}\}^*$

Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{\mathtt{ab}, \mathtt{aba}\}$. Thus, the NFA recognizes a sequence of such strings.



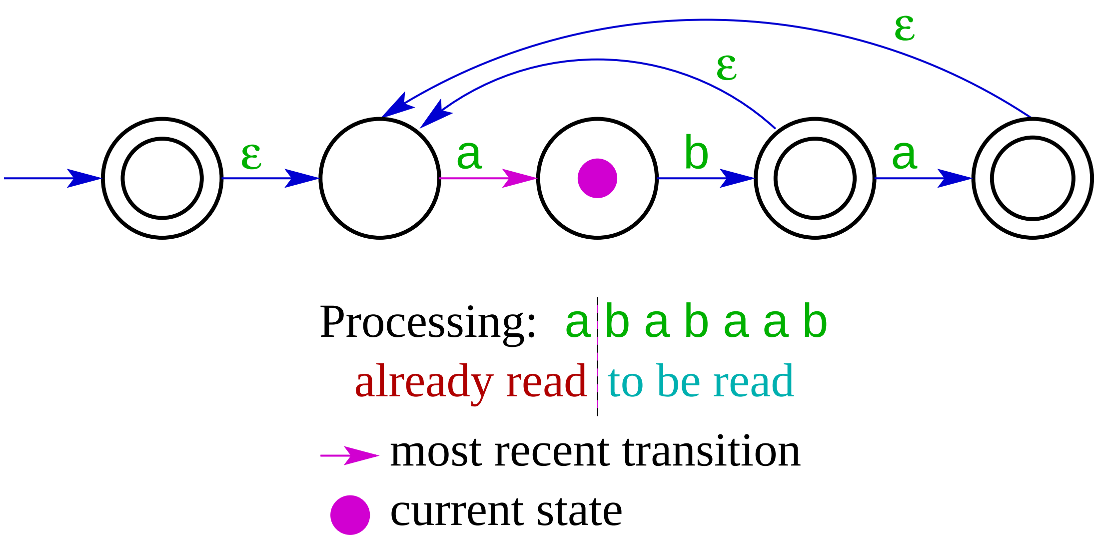
A NFA that recognizes $\{\mathbf{ab}, \mathbf{aba}\}^*$

Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{\mathrm{ab}, \mathrm{aba}\}$. Thus, the NFA recognizes a sequence of such strings.



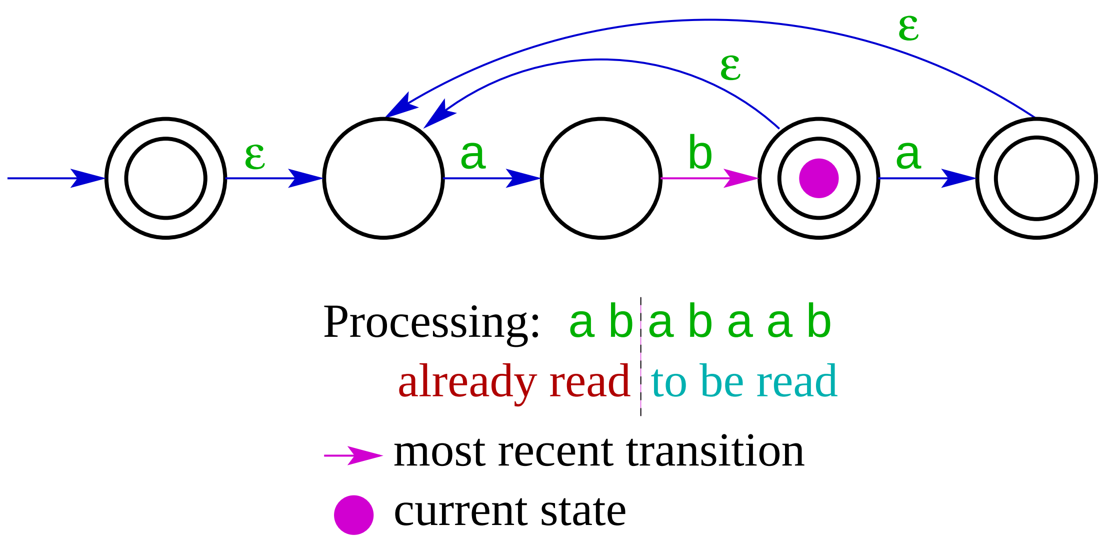
A NFA that recognizes $\{\mathbf{ab}, \mathbf{aba}\}^*$

Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{\mathtt{ab}, \mathtt{aba}\}$. Thus, the NFA recognizes a sequence of such strings.



A NFA that recognizes $\{ab, aba\}^*$

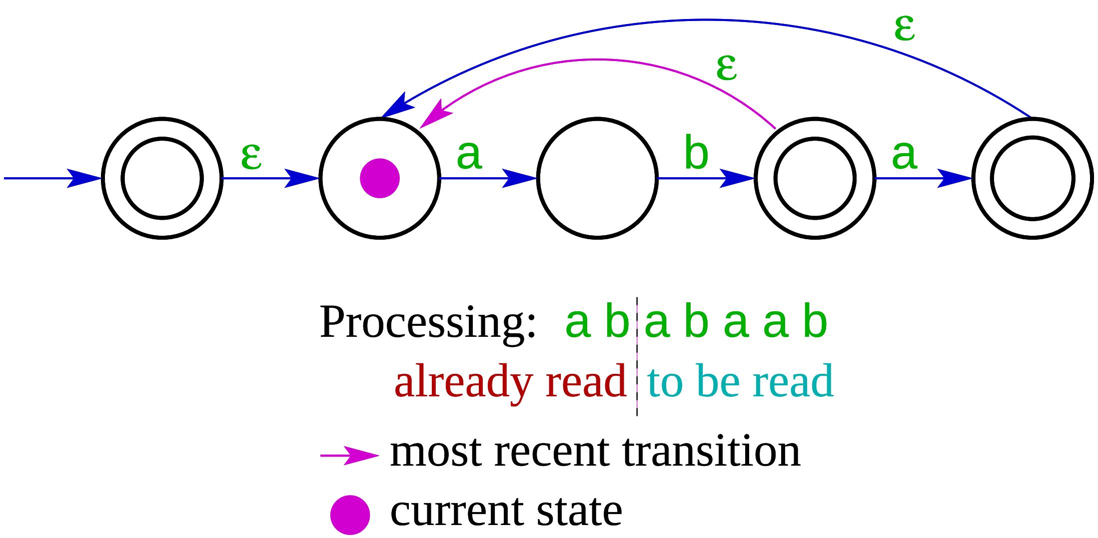
Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{ab, aba\}$. Thus, the NFA recognizes a sequence of such strings.



The NFA can choose whether to follow the a edge to the right or the ϵ edge back to the initial state. This time, the NFA goes back to the initial state.

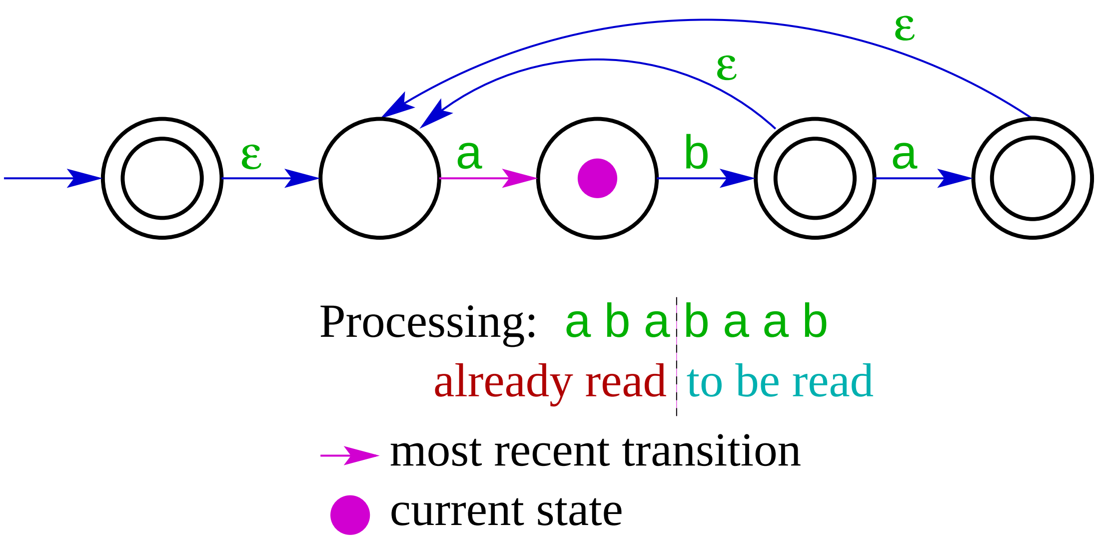
A NFA that recognizes $\{\textbf{ab}, \textbf{aba}\}^*$

Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{\text{ab}, \text{aba}\}$. Thus, the NFA recognizes a sequence of such strings.



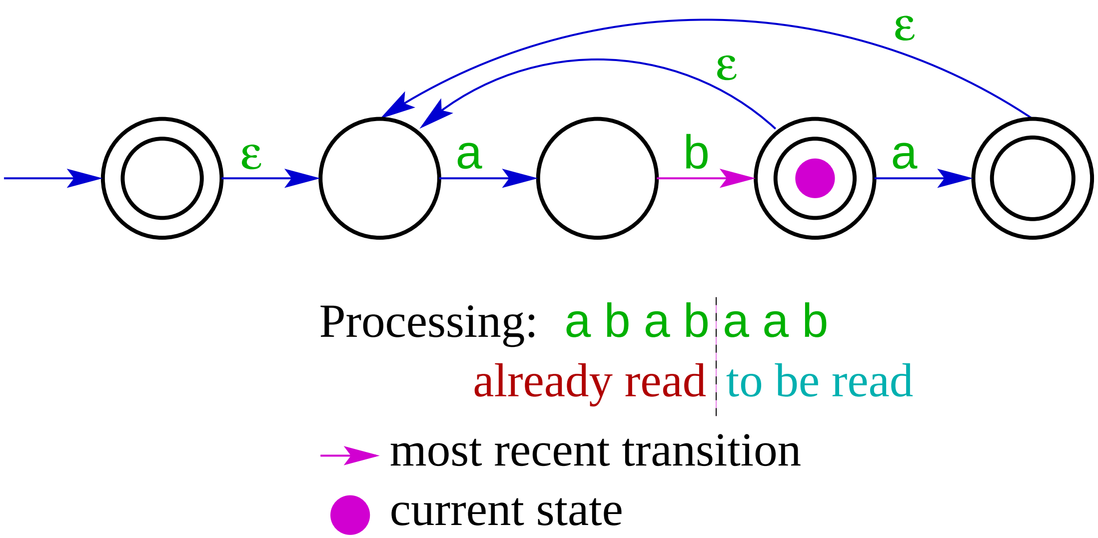
A NFA that recognizes $\{ab, aba\}^*$

Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{ab, aba\}$. Thus, the NFA recognizes a sequence of such strings.



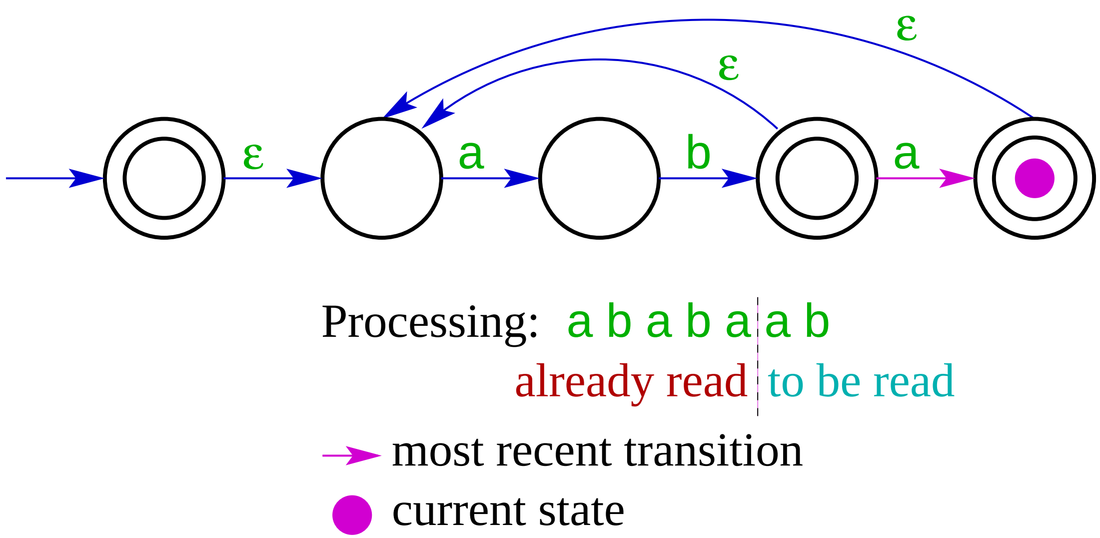
A NFA that recognizes $\{\mathbf{ab}, \mathbf{aba}\}^*$

Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{\text{ab}, \text{aba}\}$. Thus, the NFA recognizes a sequence of such strings.



A NFA that recognizes $\{\text{ab}, \text{aba}\}^*$

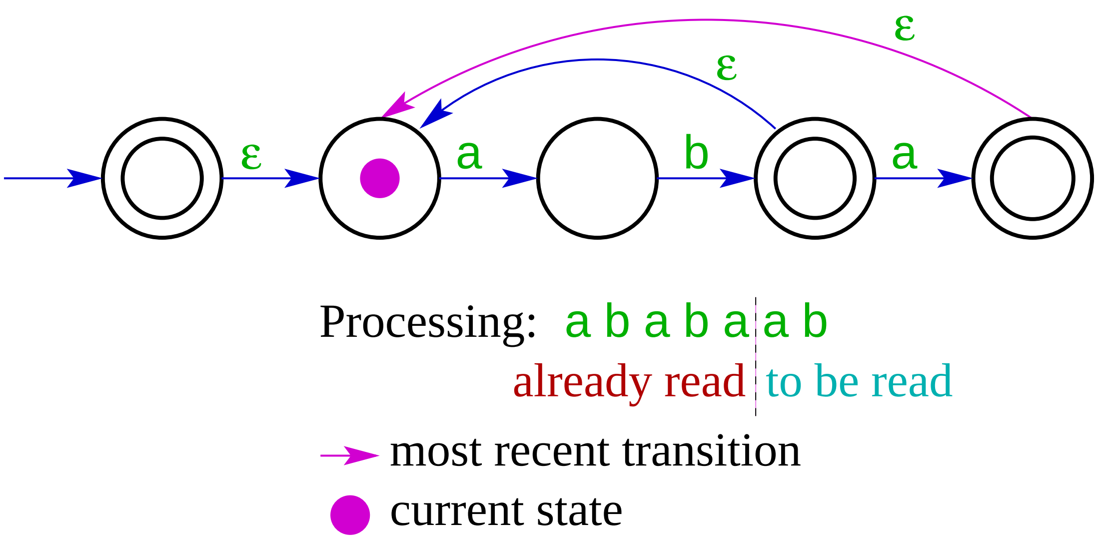
Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{\text{ab}, \text{aba}\}$. Thus, the NFA recognizes a sequence of such strings.



This time, the NFA takes the a edge to the right rather than the ϵ-edge.

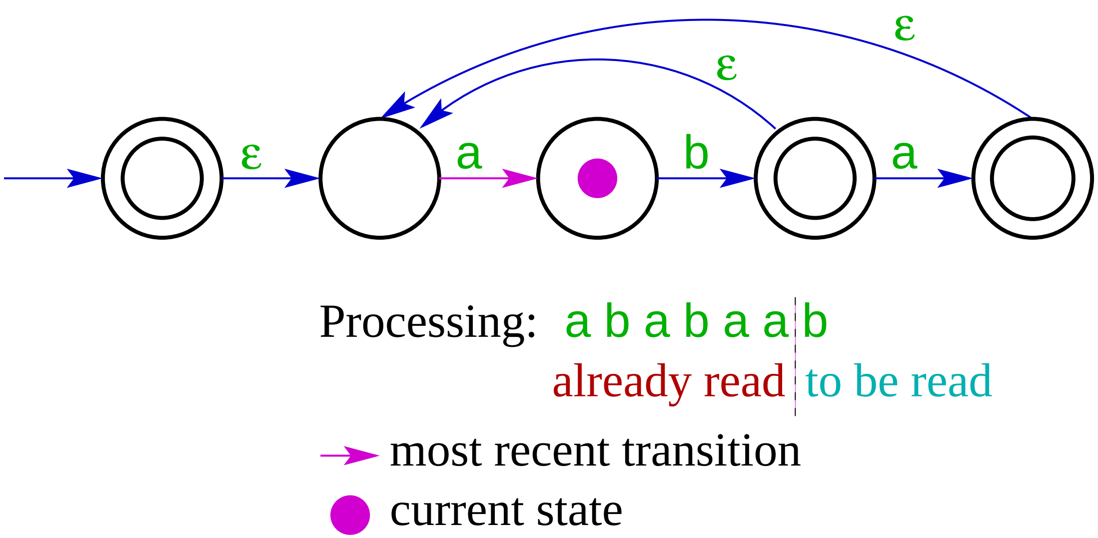
A NFA that recognizes $\{\mathbf{ab}, \mathbf{aba}\}^*$

Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{\mathtt{ab}, \mathtt{aba}\}$. Thus, the NFA recognizes a sequence of such strings.



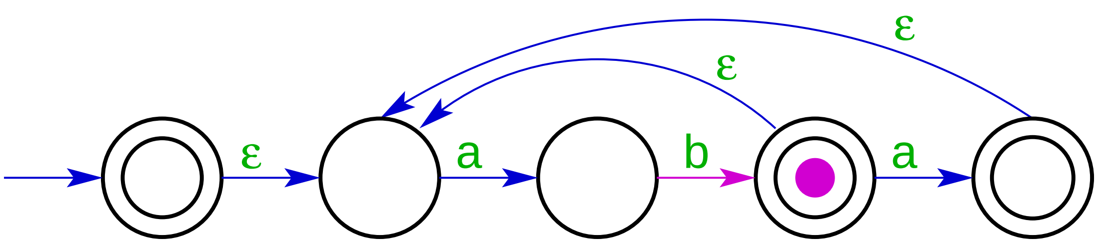
A NFA that recognizes $\{\mathbf{ab}, \mathbf{aba}\}^*$

Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{\mathsf{ab}, \mathsf{aba}\}$. Thus, the NFA recognizes a sequence of such strings.



A NFA that recognizes $\{ab, aba\}^*$

Key idea: Add ϵ-edges from each accepting state back to the initial state. Each time the NFA takes one of these edges, it has recognized a string from $\{ab, aba\}$. Thus, the NFA recognizes a sequence of such strings.



Processing: a b a b a a b
already read | to be read

most recent transition
current state

The NFA reaches the end of the string in an accepting state and accepts.

Defining NFAs

A NFA is a 5-tuple, $(Q, \Sigma, \Delta, q_0, F)$, where

- Q is a finite set of states;
- Σ is a finite alphabet;
- $\Delta \subseteq Q \times \Sigma_\epsilon \times Q$ is the transition relation;
- $q_0 \in Q$ is the initial state; and
- $F \subseteq Q$ is the set of accepting states.

- $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$.
  In English, its the set of symbols with ϵ, the empty string, added.
- If $(q_1, c, q_2) \in \Delta$, then the NFA can move from state state q_1 to state q_2 when reading symbol c from state q_q.
  - If there are states q_1, q_2, and q_3 such that (q_1, c, q_2) and (q_1, c, q_3) are both in Δ, the the NFA can move to either q_2 or q_3 when reading symbol c from state q_1.
  - If $(q_1, \epsilon, q_2) \in \Delta)$, then the NFA can move from q_1 to q_2 and stay at the same position in reading the input string.

NFA Acceptance

Let $N = (Q, \Sigma, \Delta, q_0, F)$ be an NFA.

- ϵ-closure: For any state q, we define $close_\epsilon(q)$ to be the set of all states reachable in zero or more ϵ-moves from q. Formally, $p \in close_\epsilon(q)$ iff

$$\begin{array}{l} p = q \\ \exists q' \in close_\epsilon(q).\ (q', \epsilon, p) \in \Delta \end{array}$$

  For convenience, we extend $close_\epsilon$ to sets. If $G \subseteq Q$,

$$close_\epsilon(G) \quad = \quad \bigcup_{q \in G} close_\epsilon(q)$$

- We define $step(q, c)$ to be the states that are reachable from q when reading c.
- Extending Δ to sets and strings.
- NFA N accepts s iff N can reach some accept state at the end of reading s.

NFA Acceptance

Let $N = (Q, \Sigma, \Delta, q_0, F)$ be an NFA.

- ϵ-closure: $close_\epsilon(q)$ is the set of all states reachable in zero or more ϵ-moves from q.

- We define $step(q, c)$ to be the states that are reachable from q when reading c:

$$step(q, c) \quad = \quad close_\epsilon(\{q' \mid (q, c, q') \in \Delta\})$$

  and we extend $step$ to sets just as we did for $close_\epsilon$. If $G \subseteq Q$,

$$step(G, c) \quad = \quad \bigcup_{q \in G} step(q, c)$$

- Extending Δ to sets and strings.

- NFA N accepts s iff N can reach some accept state at the end of reading s.

NFA Acceptance

Let $N = (Q, \Sigma, \Delta, q_0, F)$ be an NFA.

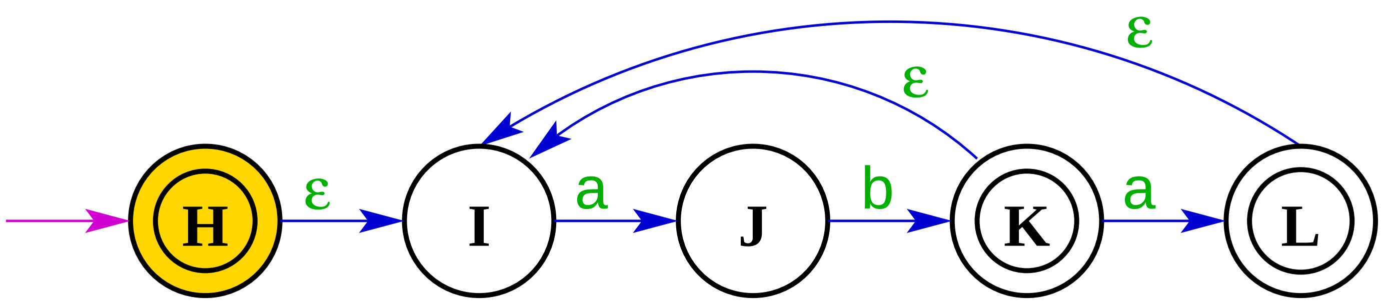
- ϵ-closure: $close_\epsilon(q)$ is the set of all states reachable in zero or more ϵ-moves from q.

- We define $step(q, c)$ to be the states that are reachable from q when reading c.
- Extending Δ to sets and strings.
  For $G \subseteq Q$ and $s \in \Sigma^*$,

$$
\begin{aligned}
\Delta(G, \epsilon) &= close_\epsilon(G) \\
\Delta(G, x \cdot c) &= step(\Delta(G, x), c)
\end{aligned}
$$

  Inituitively, $\Delta(G, s)$ is the set of all states that can be reached by starting from some state in G and reading string s. Slide 13 provides an example.
- NFA N accepts s iff N can reach some accept state at the end of reading s.
  More precisely, N accepts s iff $\Delta(\{q_0\}, s) \cap F \neq \emptyset$.

Acceptance example

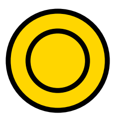


Processing: a b a b a a b

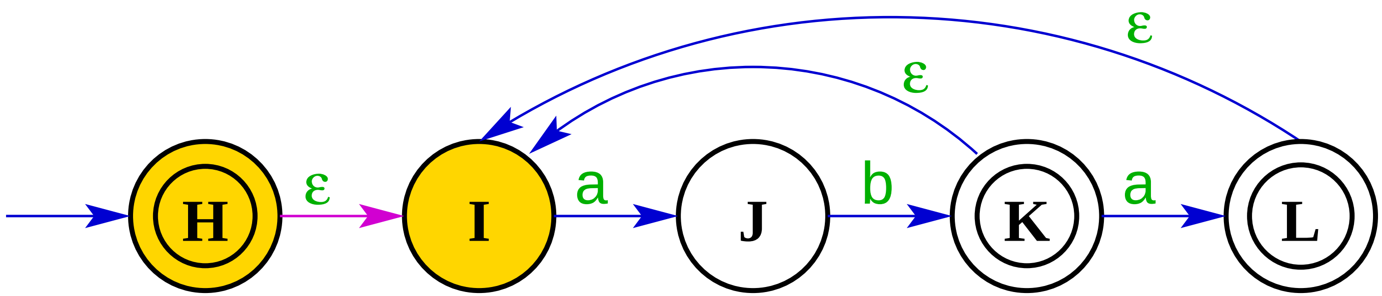
already read | to be read

$$\Delta = \{ \ (H, \epsilon, I), (I, \mathbf{a}, J), (J, \mathbf{b}, K),$$
$$(K, \epsilon, I), (K, \mathbf{a}, L), (L, \epsilon, I), \ \}$$

 enter initial state

 Initial reachable state

Acceptance example



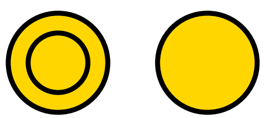
Processing: a b a b a a b

already read | to be read

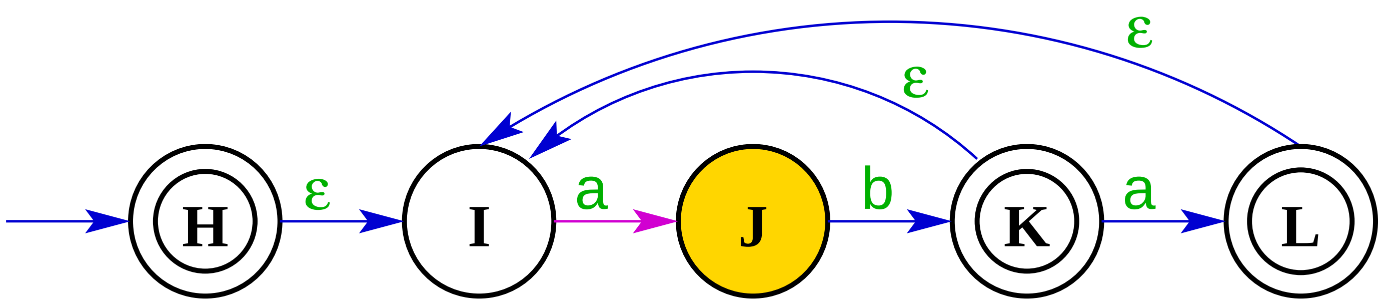
$$\Delta = \{ (H, \epsilon, I), (I, \textbf{a}, J), (J, \textbf{b}, K), \\ (K, \epsilon, I), (K, \textbf{a}, L), (L, \epsilon, I), \}$$

$$\Delta(\{H\}, \epsilon) = \{H, I\}$$

 initial ϵ-moves.

 $close_\epsilon$ of initial state.

Acceptance example



Processing: a | b a b a a b

already read | to be read

$$\Delta \;=\; \{\;(H, \epsilon, I), (I, \mathbf{a}, J), (J, \mathbf{b}, K),$$

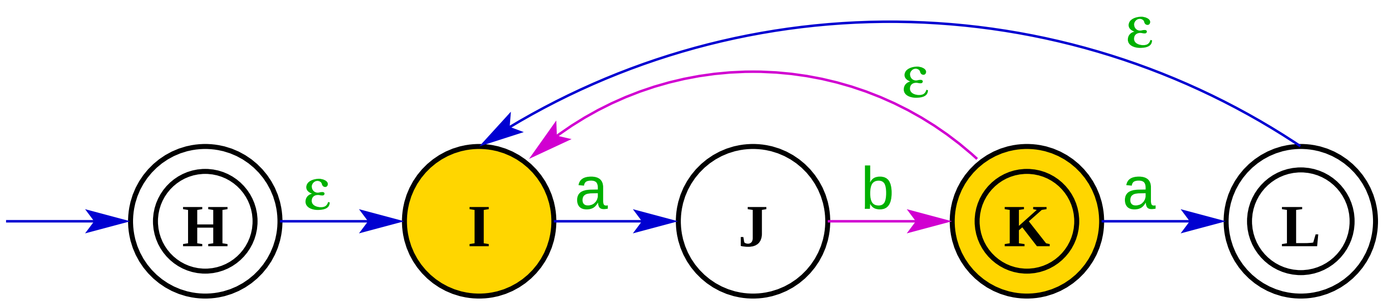
$$(K, \epsilon, I), (K, \mathbf{a}, L), (L, \epsilon, I),\;\}$$

$$\Delta(\{H\}, \mathbf{a}) \;=\; \{J\}$$

state transistion at this step.

reachable state at this step.

Acceptance example



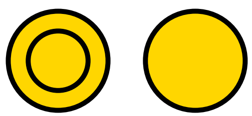
Processing: a b | a b a a b

already read | to be read

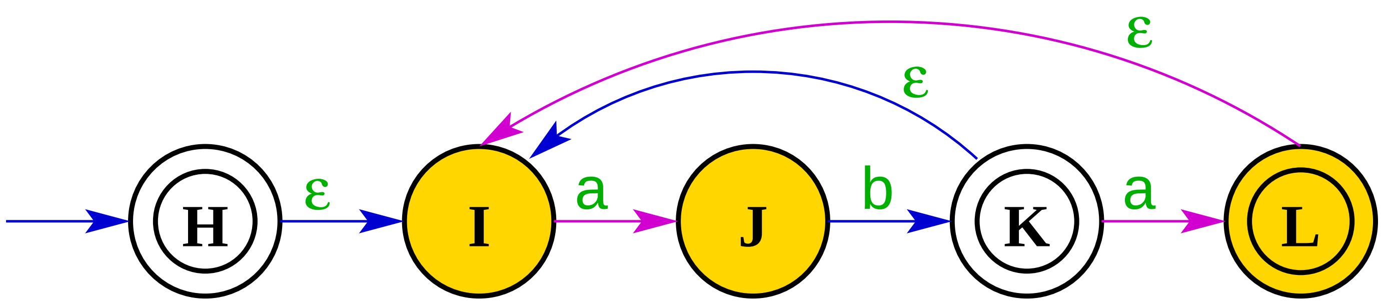
$$\Delta = \{ \ (H, \epsilon, I), (I, \mathbf{a}, J), (J, \mathbf{b}, K), \\ (K, \epsilon, I), (K, \mathbf{a}, L), (L, \epsilon, I), \ \}$$

$$\Delta(\{H\}, \mathbf{a}) = \{I, K\}$$

 possible state transistions at this step.

 reachable states at this step.

Acceptance example



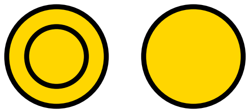
Processing: a b a | b a a b

already read | to be read

$$\Delta = \{ \ (H, \epsilon, I), (I, \mathbf{a}, J), (J, \mathbf{b}, K), (K, \epsilon, I), (K, \mathbf{a}, L), (L, \epsilon, I), \ \}$$

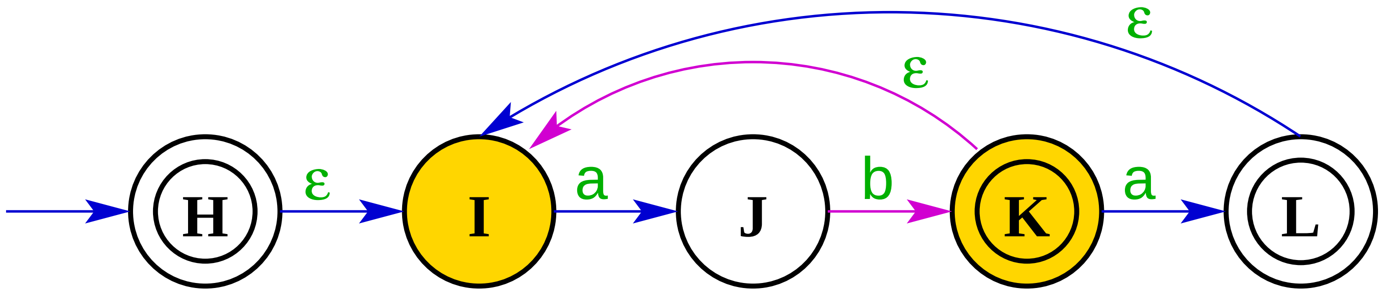
$$\Delta(\{H\}, \mathbf{a}) = \{I, J, L\}$$

possible state transistions at this step.



reachable states at this step.

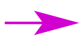
Acceptance example

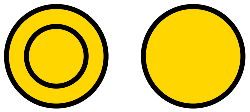


Processing: a b a b | a a b
already read | to be read

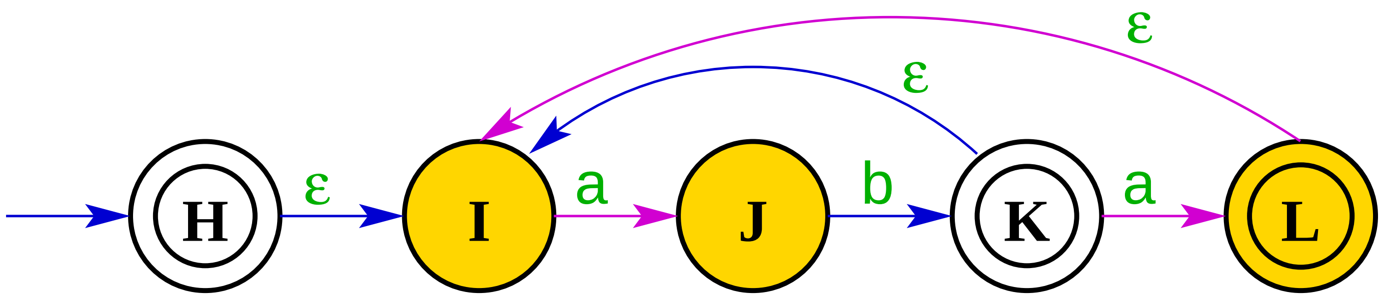
$$\Delta = \{ (H, \epsilon, I), (I, \mathbf{a}, J), (J, \mathbf{b}, K), (K, \epsilon, I), (K, \mathbf{a}, L), (L, \epsilon, I), \}$$

$$\Delta(\{H\}, \mathbf{a}) = \{I, K\}$$

 possible state transistions at this step.

 reachable states at this step.

Acceptance example



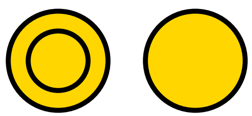
Processing: a b a b a | a b

already read | to be read

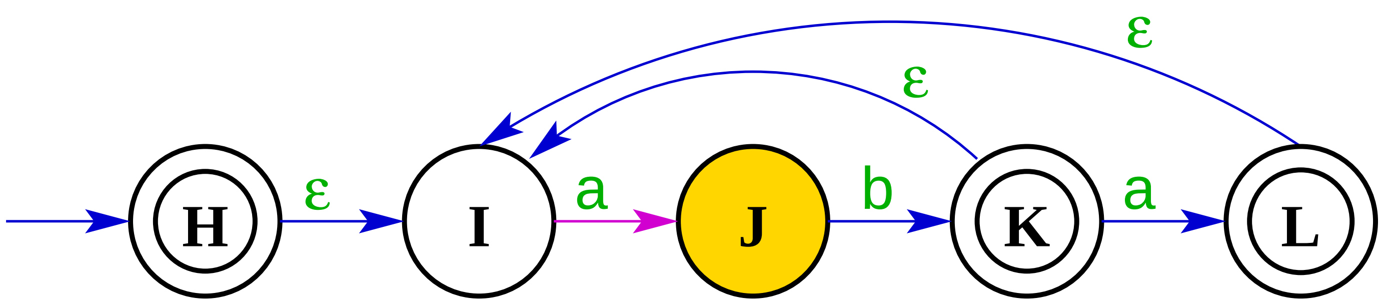
$$\Delta = \{\ (H, \epsilon, I), (I, \mathbf{a}, J), (J, \mathbf{b}, K),$$
$$(K, \epsilon, I), (K, \mathbf{a}, L), (L, \epsilon, I),\ \}$$

$$\Delta(\{H\}, \mathbf{a}) = \{I, J, L\}$$

 possible state transistions at this step.

 reachable states at this step.

Acceptance example



Processing: a b a b a a | b

already read | to be read

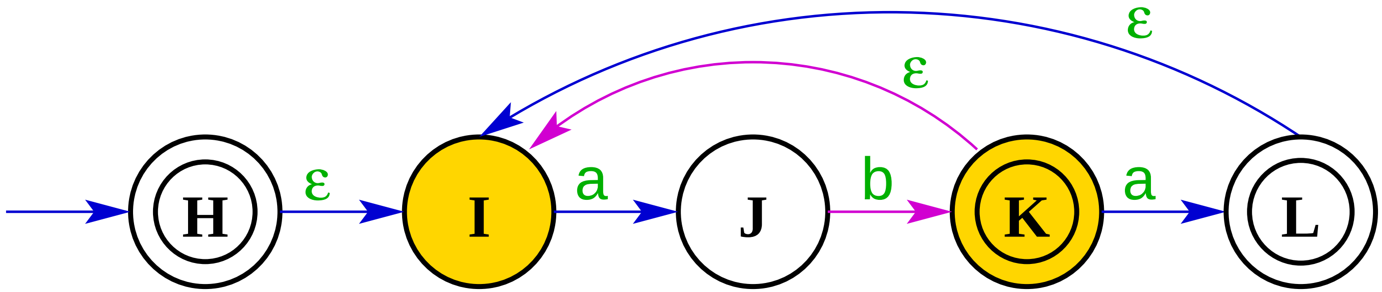
$$\Delta = \{ \ (H, \epsilon, I), (I, \mathbf{a}, J), (J, \mathbf{b}, K), \\ (K, \epsilon, I), (K, \mathbf{a}, L), (L, \epsilon, I), \ \}$$

$$\Delta(\{H\}, \mathbf{a}) \ = \ \{J\}$$

$\rightarrow$ state transistion at this step.

reachable state at this step.

Acceptance example



Processing: a b a b a a b | nothing

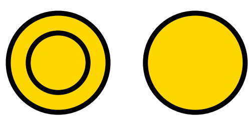
already read | left to read

ACCEPT

$$\Delta = \{ (H,\epsilon,I),(I,\mathbf{a},J),(J,\mathbf{b},K),$$
$$(K,\epsilon,I),(K,\mathbf{a},L),(L,\epsilon,I), \}$$

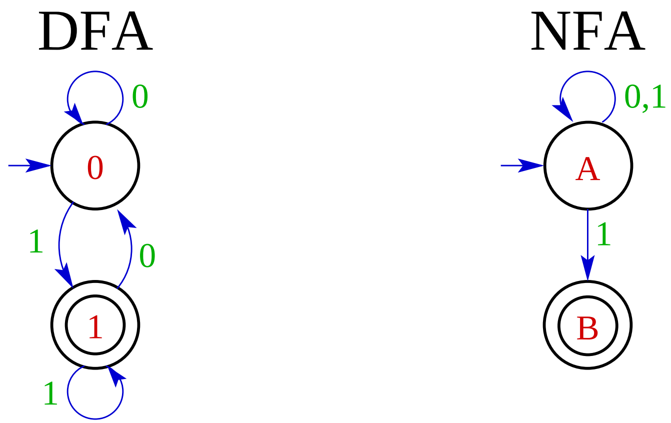
$$\Delta(\{H\},\mathbf{a}) = \{I,K\}$$


possible state transistions at this step.


reachable states at this step.

Another example: last symbol is 1

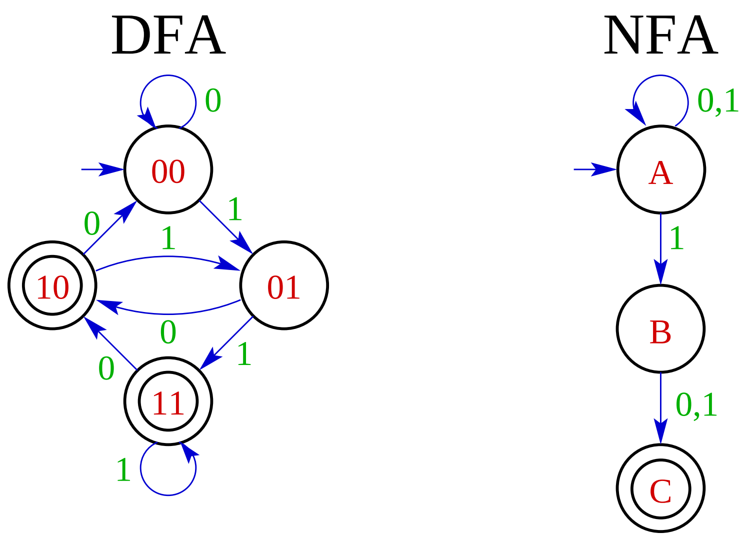
Let $\Sigma = \{0, 1\}$ and let L_1 be the set of all strings that end with a 1. Here are a DFA and a NFA that recognize L_1:



The DFA and NFA are nearly identical.

L_2: next to last symbol is 1

Let L_2 be the set of all strings that end with a 1 as the next to last symbol. Here's a DFA and a NFA that recognize L_2:

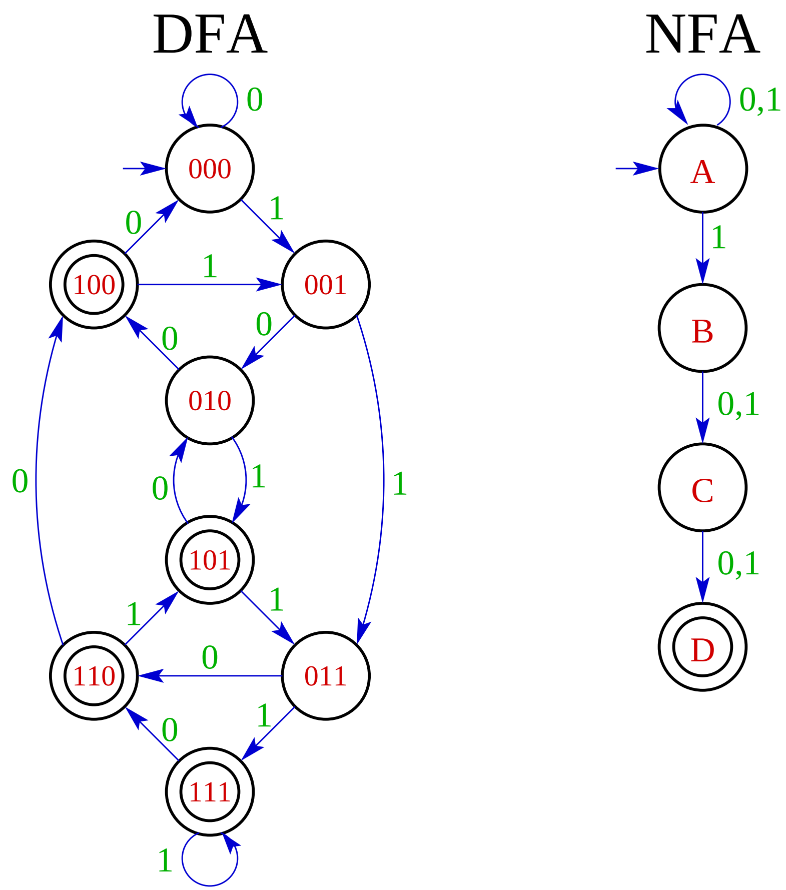


The DFA has states to keep track of the last two symbols read. State xy means that x was the next to the last symbol read, and y was the last symbol read. For example, state 10 means that the last symbol read was a 0 and the next to the last symbol read was a 1 (hence state 10 is accepting).

The NFA remains in its initial state. If the next to the last symbol is a 1, the NFA "guesses" that this is the next-to-last symbos and moves to state B. Then, when it reads the last symbol, it moves to state C and accepts.

L_3: third from last symbol is 1

Let L_3 be the set of all strings that end with a 1 as the next to last symbol. Here are a DFA and a NFA that recognize L_3:



The DFA now has states to keep track of the last three symbols read. Hence, it has

L_k: k^{th} from last symbol is 1

Any DFA that recognize L_k must have at least 2^k states.

A NFA with $k+1$ states can recognize L_k.

This example shows that a NFA can have exponentially fewer states than the smallest DFA that recognizes the same language.

Summary of the week

Monday, Sept. 8: DFAs.
Basic definitions and acceptance conditions.

Wednesday, Sept. 10: Regular Languages.
Basic definitions and the product-machine construction.

Friday, Sept. 12: NFA's
Definitions and acceptance conditions.

We now have the basic computation models for understanding regular languages.

- Next week, we'll extend these to regular-expressions, a way to describe regular languages that looks more a programming language and less like a description of hardware. We'll show that DFA, NFA, and regular expressions all give rise to exactly the same set of languages.
- In the following week, we'll wrap all of this up by showing that there are languages that aren't regular, and we'll show how to prove that a language is not regular.

The coming week

Reading: Note: this is different than the schedule in the Sept. 3 notes – we're one lecture ahead of schedule.

- September 12 (Today): Introduction to NFAs.
  - Read *Sipser* 1.2.
  - Lecture will cover through Example 1.35 (i.e. pages 47–52).
- September 15 (Monday): Equivalance of NFAs and DFAs
  - The rest of *Sipser* 1.2. (i.e. pages 53-63).
- September 17 (Wednesday): Regular Expressions
  - Read *Sipser* 1.3. Lecture will cover throud example 1.58 (i.e. pages 63-69).
- September 19 (Friday): Equivalence of DFAs and Regular Expressions
  - The rest of *Sipser* 1.3 (i.e. pages 69–76).

Homework:

- September 12 (Today): Homework 0 due. Homework 1 will be posted to the web page later today (due Sept. 19).
- September 19 (next Friday): Homework 1 due. Homework 2 goes out (due Sept. 26).

Midterm: Oct. 8