Finite Automata

Part Two

Recap from Last Time

Formal Language Theory

- An ***alphabet*** is a set, usually denoted Σ, consisting of elements called ***characters***.
- A ***string over Σ*** is a finite sequence of zero or more characters taken from Σ.
- The ***empty string*** has no characters and is denoted ε.
- A ***language over Σ*** is a set of strings over Σ.
- The language **Σ*** is the set of all strings over Σ.

DFAs

- A ***DFA*** is a
  - ***D***eterministic
  - ***F***inite
  - ***A***utomaton
- DFAs are the simplest type of automaton that we will see in this course.

DFAs

- A DFA is defined relative to some alphabet Σ.
- For each state in the DFA, there must be ***exactly one*** transition defined for each symbol in Σ.
    - This is the "deterministic" part of DFA.
- There is a unique start state.
- There are zero or more accepting states.

The Language of an Automaton

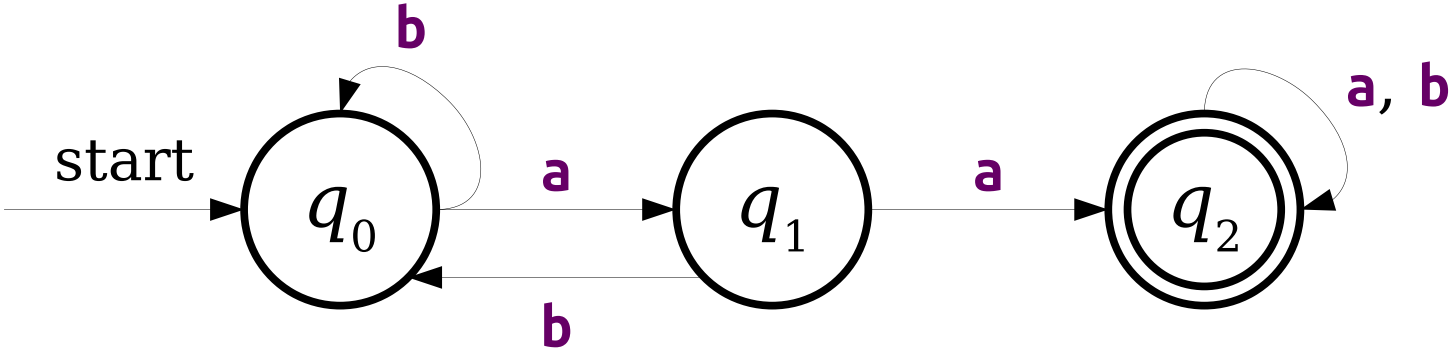
- If D is a DFA that processes strings over Σ, the ***language of D***, denoted $\mathscr{L}(\boldsymbol{D})$, is the set of all strings D accepts.
- Formally:

$$\mathscr{L}(\boldsymbol{D}) = \{\ \boldsymbol{w} \in \Sigma^* \mid \boldsymbol{D} \textbf{ accepts } \boldsymbol{w}\ \}$$

New Stuff!

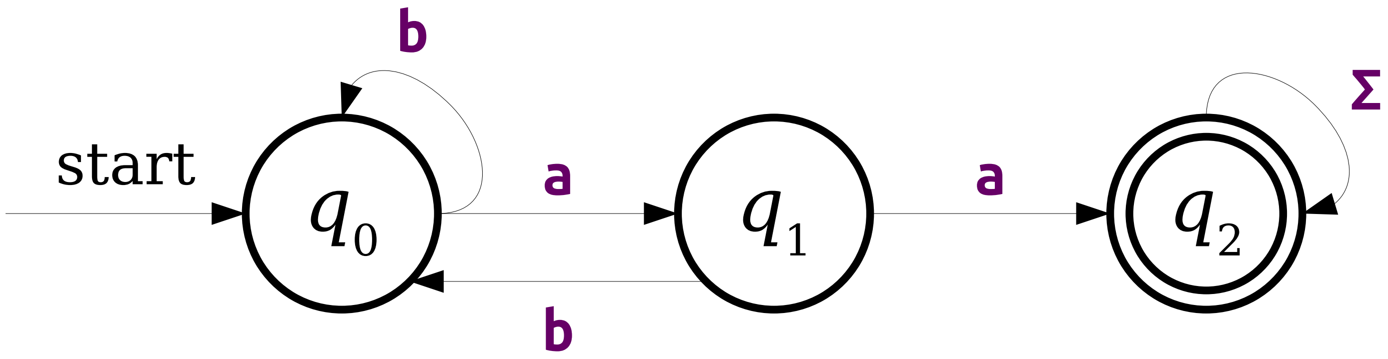
Recognizing Languages with DFAs

$L = \{\ w \in \{$**a**, **b**$\}^* \mid w$ contains **aa** as a substring $\}$



Recognizing Languages with DFAs

$L = \{\ w \in \{\texttt{a}, \texttt{b}\}^* \mid w \text{ contains } \texttt{aa} \text{ as a substring }\}$



More Elaborate DFAs

$L = \{ w \in \{\mathbf{a}, \mathbf{*}, \mathbf{/}\}^* \mid w \text{ represents a C-style comment } \}$

Let's have the **a** symbol be a placeholder for "some character that isn't a star or slash."

Let's design a DFA for C-style comments. Those are the ones that start with /* and end with */.

Accepted:

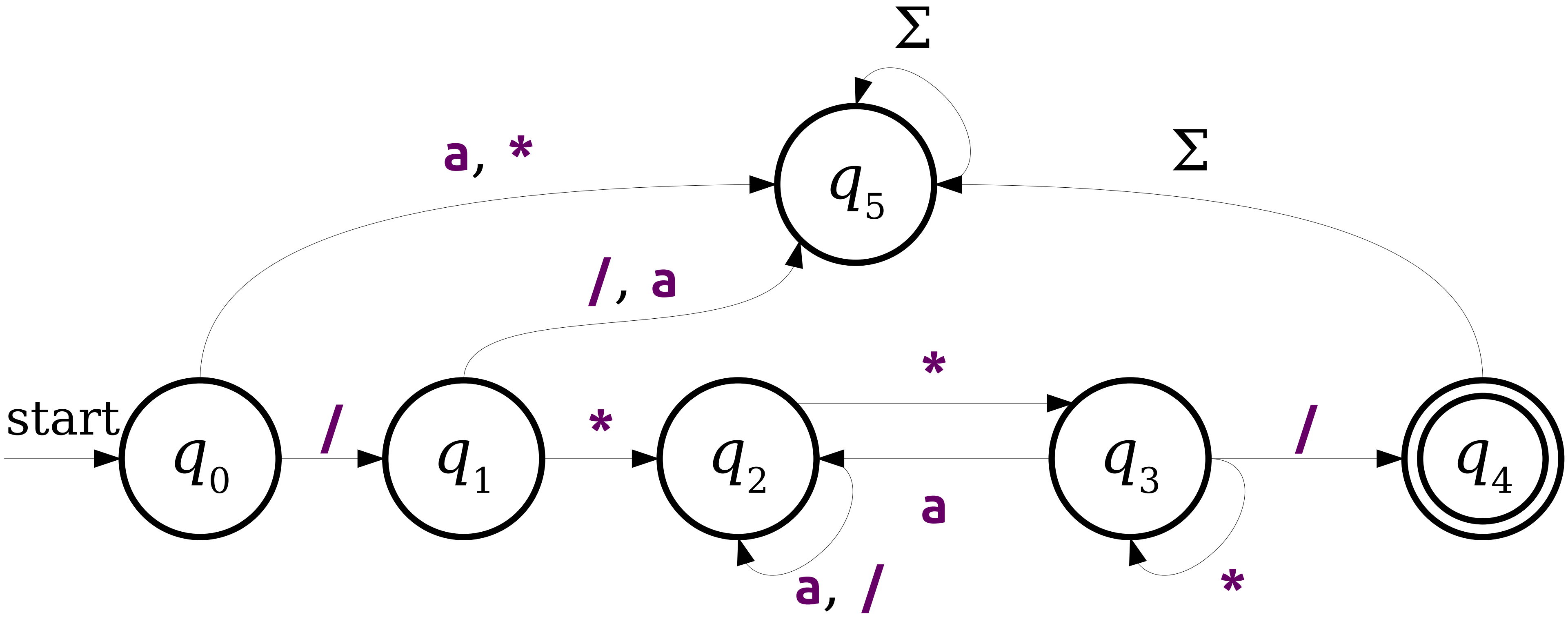
/*a*/
//**
/*/**
/*aaa*aaa*/
/*a/a*/

Rejected:

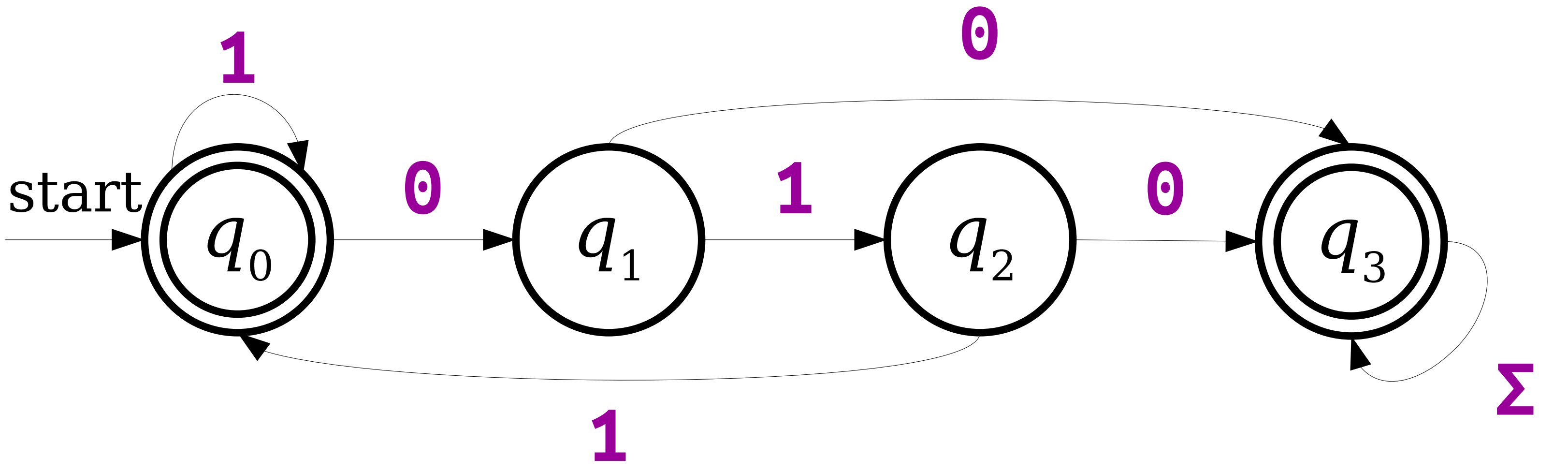
/**
//a/*aa*/**
aaa//aa**
/*/
/a/**
//aaaa

More Elaborate DFAs

$L = \{\ w \in \{$**a**, *****, **/**$\}^* \mid w$ represents a C-style comment $\}$



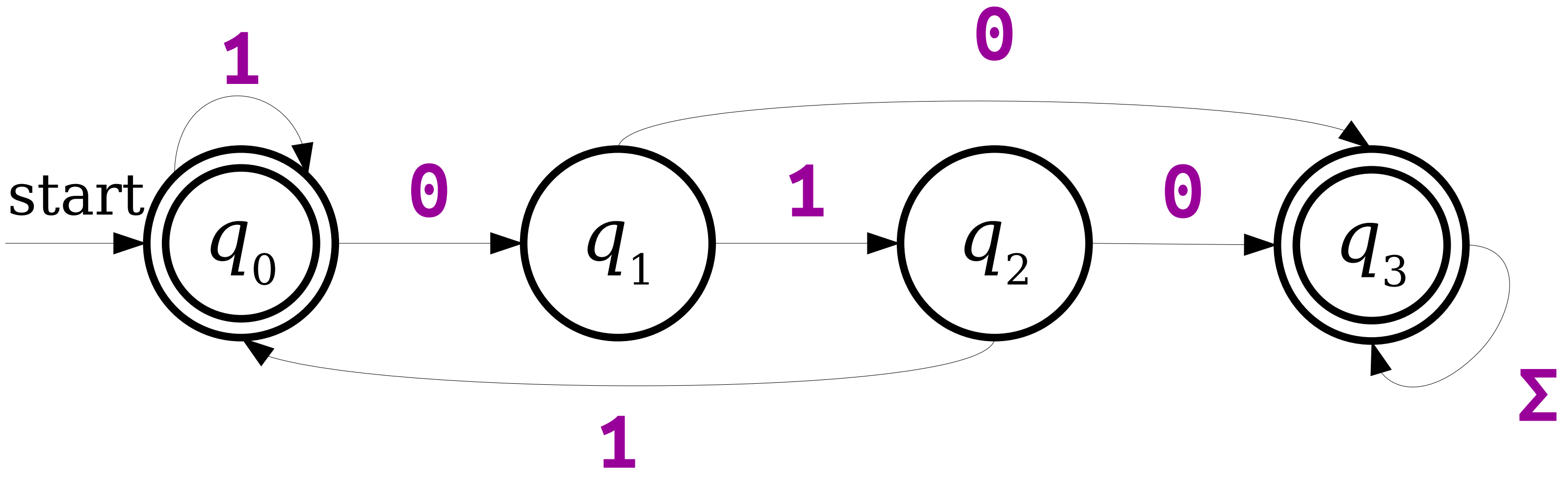
Tabular DFAs



	0	1
$*q_0$	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
$*q_3$	q_3	q_3

These stars indicate accepting states.

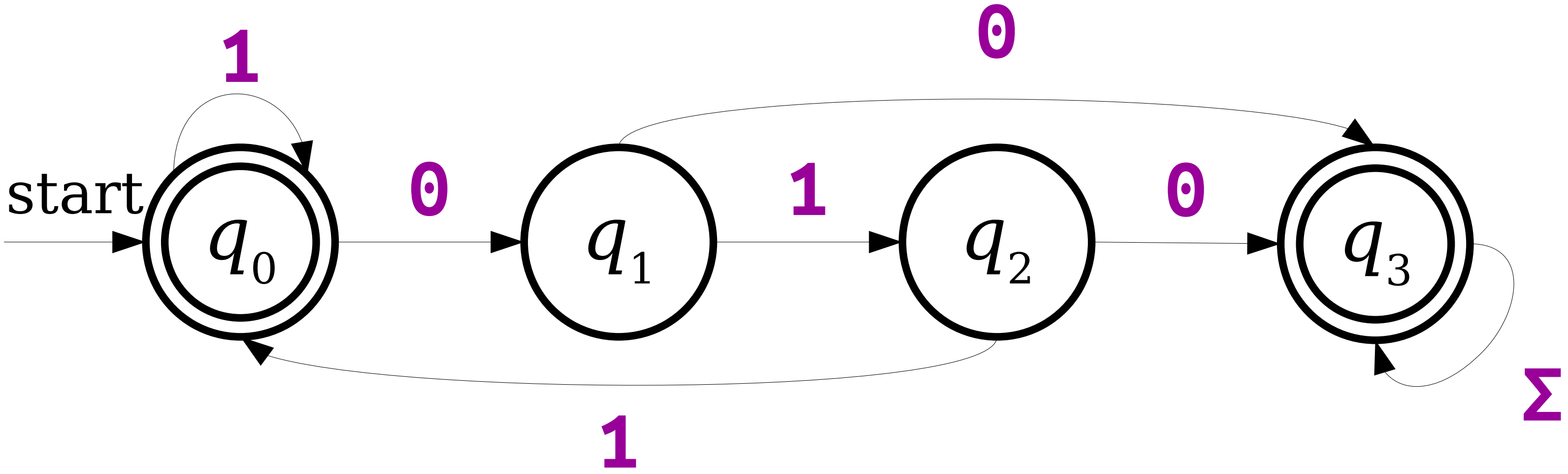
Tabular DFAs



	0	1
$*q_0$	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
$*q_3$	q_3	q_3

Since this is the first row, it's the start state.

Tabular DFAs



	0	1
$*q_0$	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
$*q_3$	q_3	q_3

Question to ponder: Why isn't there a column here for Σ?

Code? In a Theory Class?

```
int kTransitionTable[kNumStates][kNumSymbols] = {
     {0, 0, 1, 3, 7, 1, …},
      …
};
bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    …
};
bool SimulateDFA(string input) {
    int state = 0;
    for (char ch: input) {
        state = kTransitionTable[state][ch];
    }
    return kAcceptTable[state];
}
```

The Regular Languages

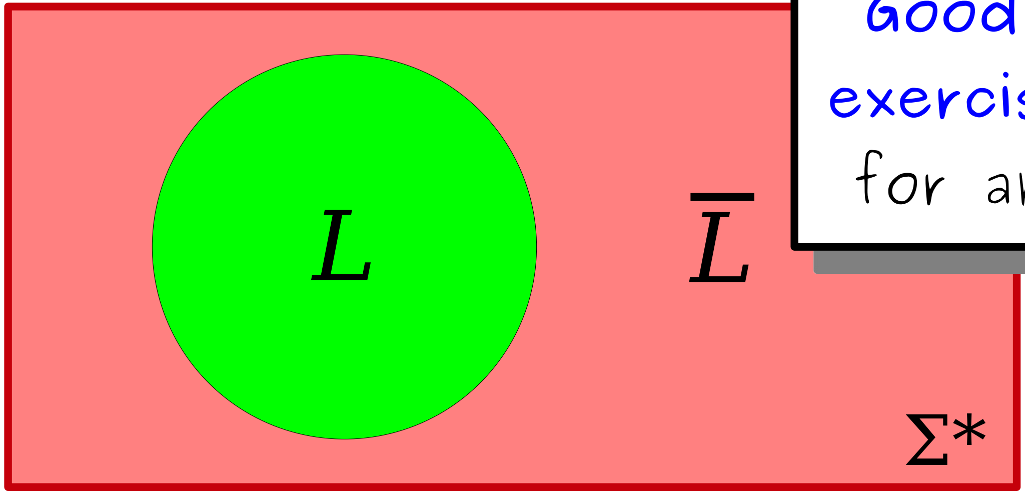
A language L is called a ***regular language*** if there exists a DFA D such that $\mathscr{L}(D) = L$.

If L is a language and $\mathscr{L}(D) = L$, we say that D ***recognizes*** the language L.

The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the ***complement*** of that language (denoted $\overline{L}$) is the language of all strings in Σ^* that aren't in L.
- Formally:

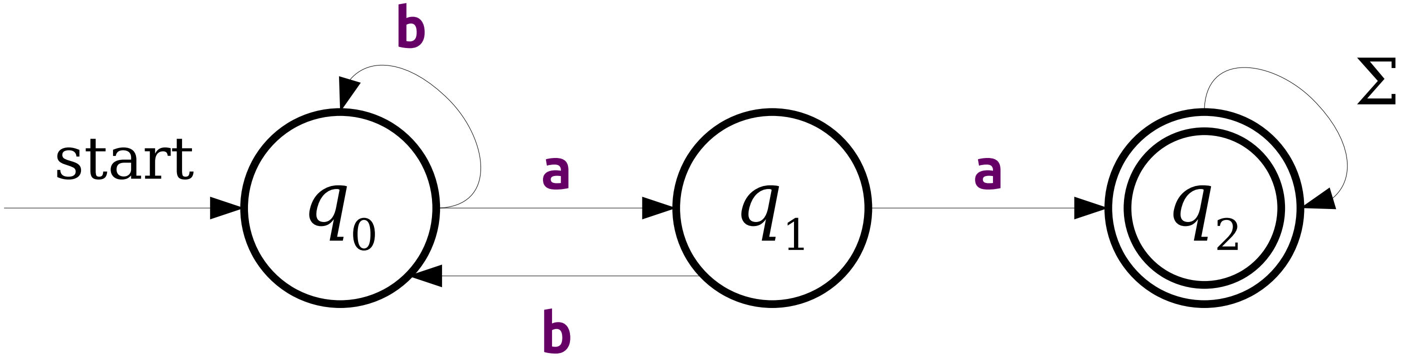
$$\overline{L} = \Sigma^* - L$$



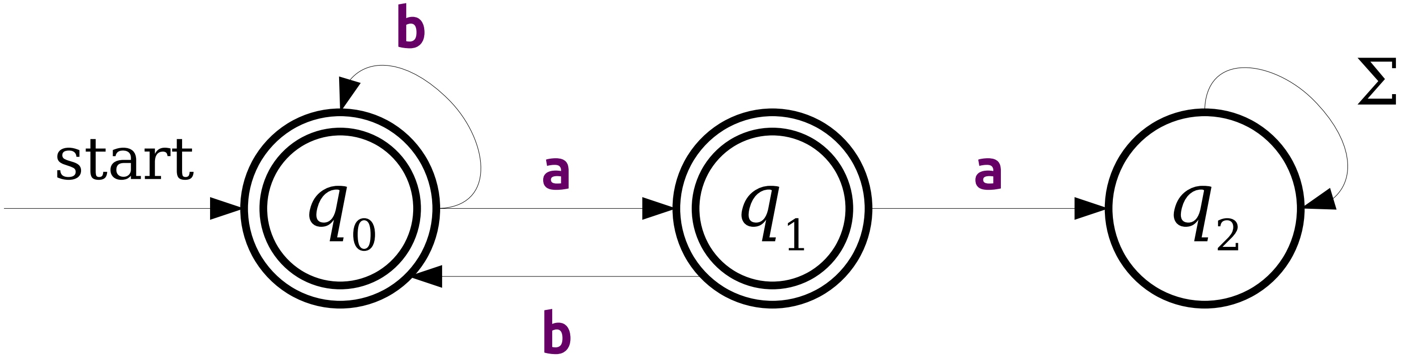
Good proofwriting exercise: prove $\overline{\overline{L}} = L$ for any language L.

Complementing Regular Languages

L = { w ∈ {a, b}* | w contains aa as a substring }

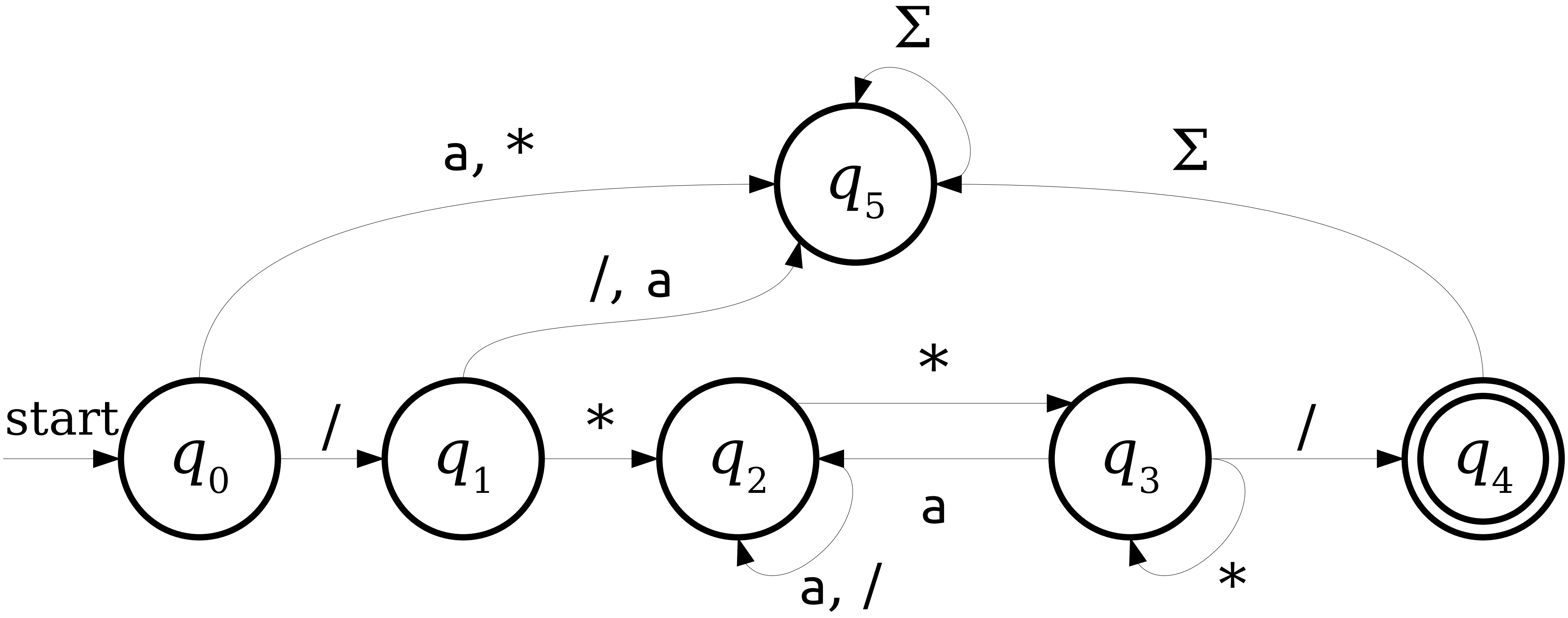


$\overline{L}$ = { w ∈ {a, b}* | w ***does not*** contain aa as a substring }



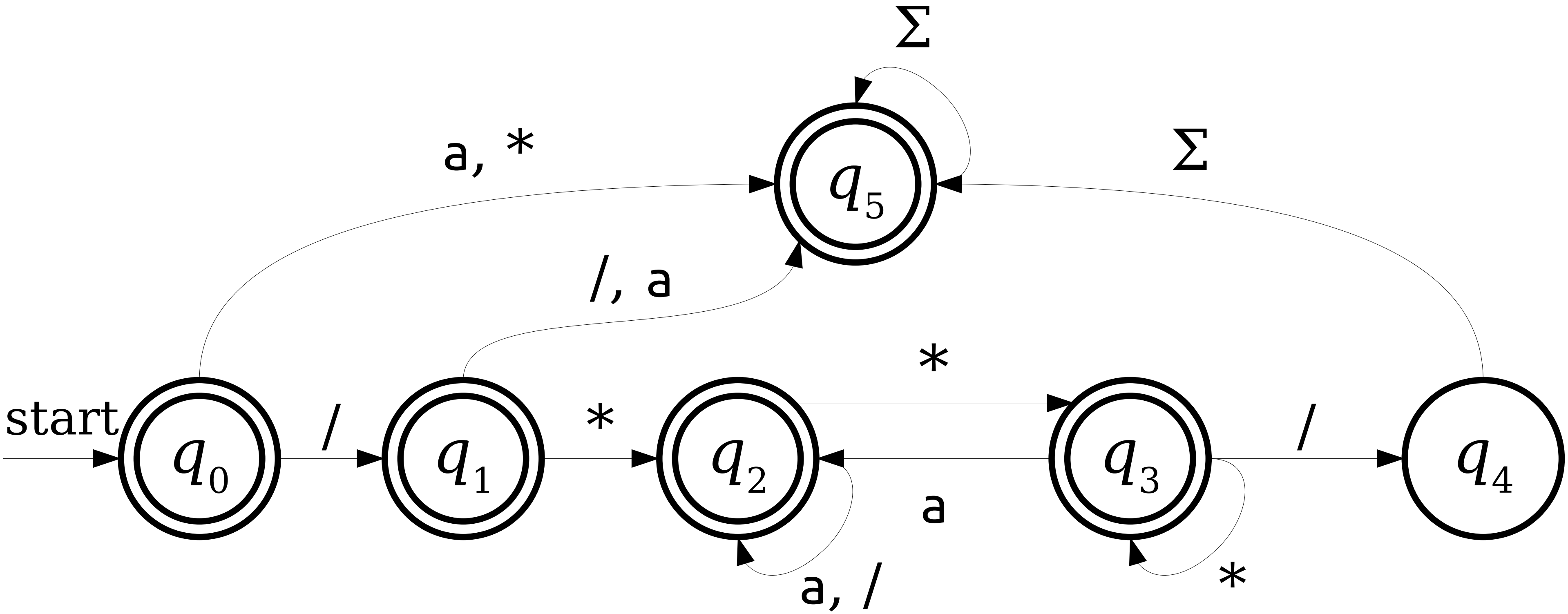
Complementing Regular Languages

$L = \{\ w \in \{$**a**$,$ *****$,$ **/**$\}^* \mid w$ represents a C-style comment $\}$



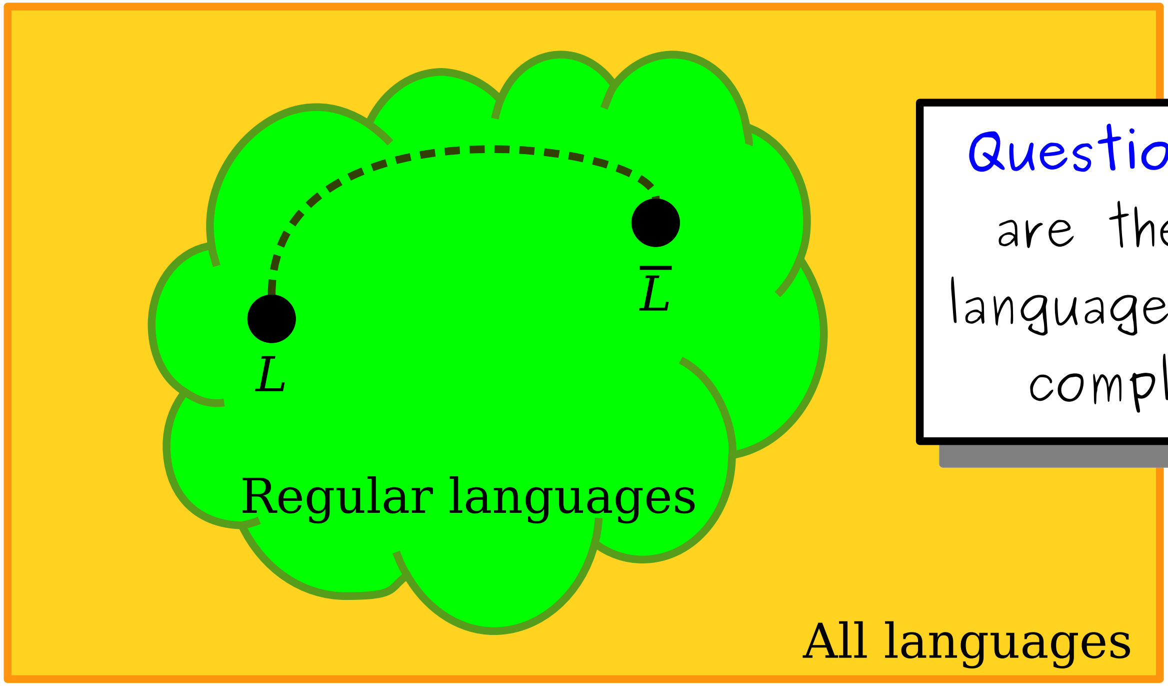
Complementing Regular Languages

$\overline{L}$ = { w ∈ {**a**, *****, **/**}* | w ***doesn't*** represent a C-style comment }



Closure Properties

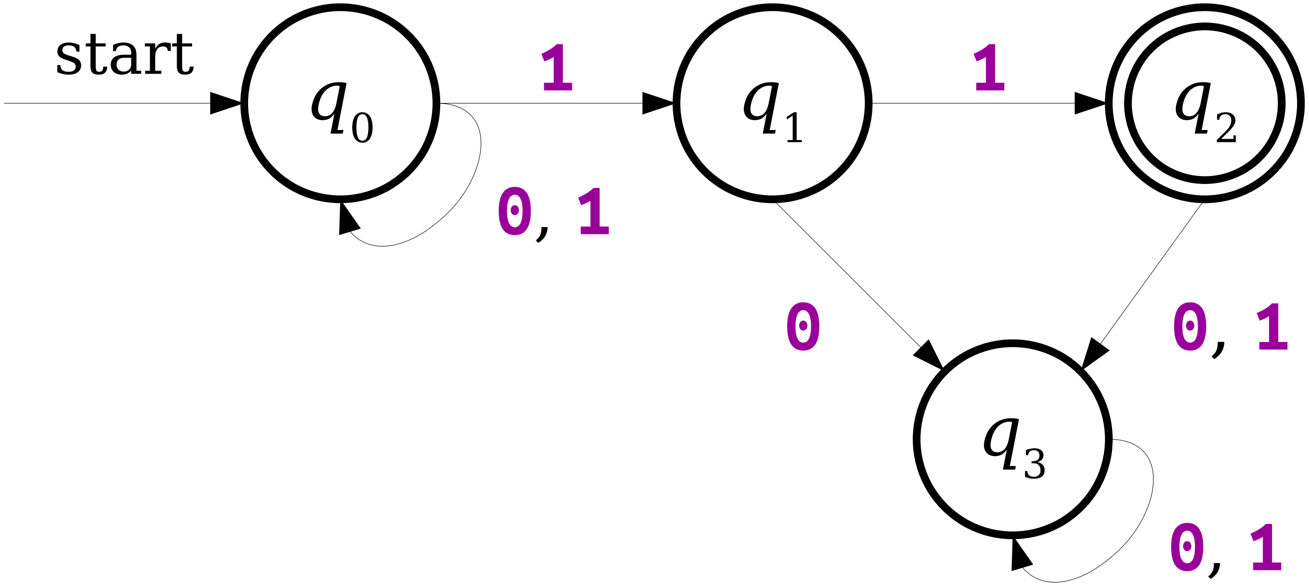
- ***Theorem:*** If L is a regular language, then $\overline{L}$ is also a regular language.
- As a result, we say that the regular languages are ***closed under complementation***.



Question to ponder: are the *nonregular* languages closed under complementation?

NFAs

Revisiting a Problem



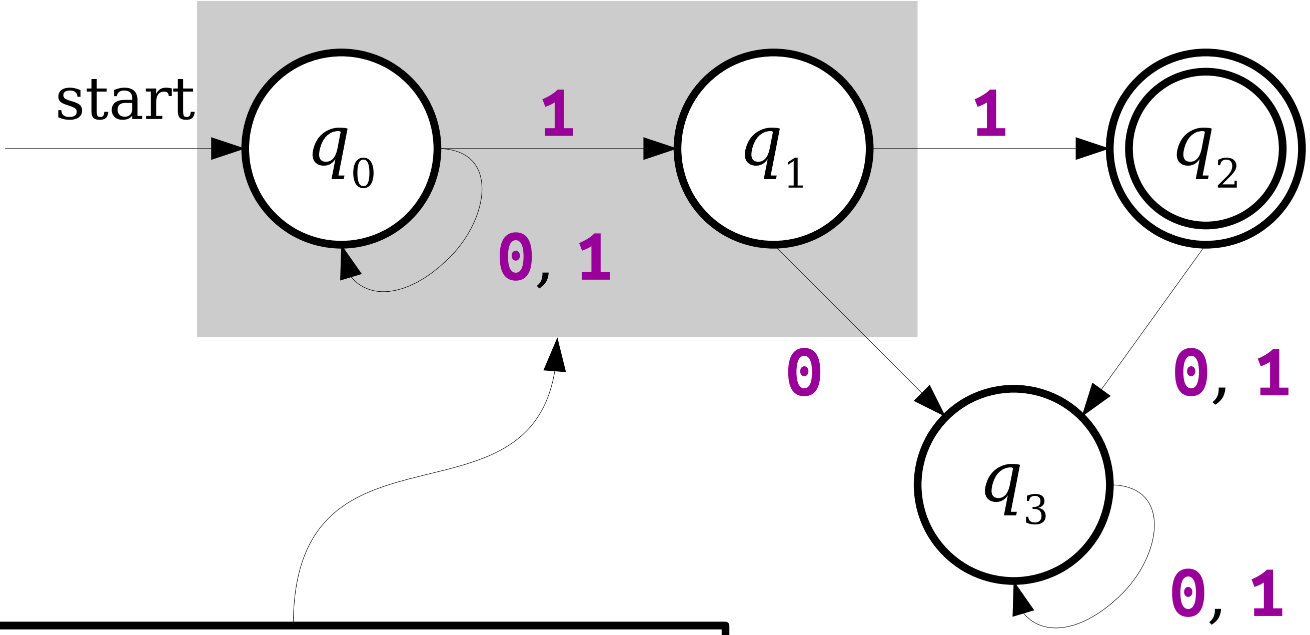
NFAs

- An ***NFA*** is a
  - ***N***ondeterministic
  - ***F***inite
  - ***A***utomaton
- Structurally similar to a DFA, but represents a fundamental shift in how we'll think about computation.

(Non)determinism

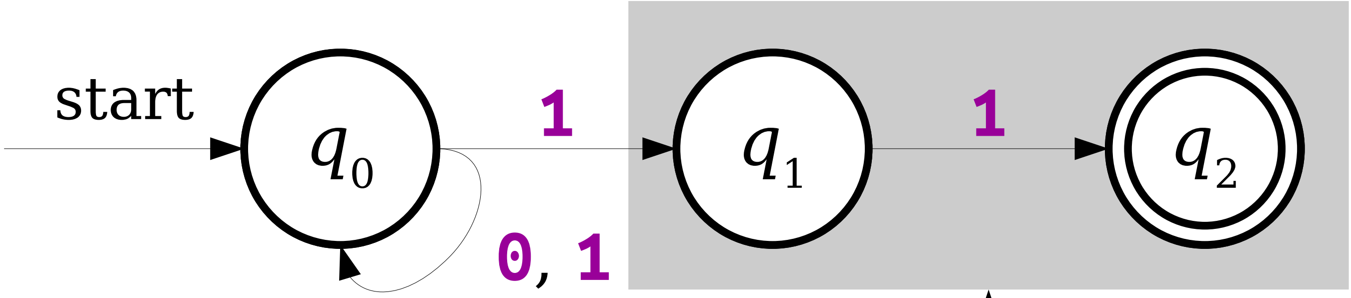
- A model of computation is ***deterministic*** if at every point in the computation, there is exactly one choice that can make.
  - The machine accepts if that series of choices leads to an accepting state.
- A model of computation is ***nondeterministic*** if the computing machine has a finite number of choices available to make at each point, possibly including zero.
- The machine accepts if ***any*** series of choices leads to an accepting state.
  - (This sort of nondeterminism is technically called ***existential nondeterminism***, the most philosophical-sounding term we'll introduce all quarter.)

A Simple NFA



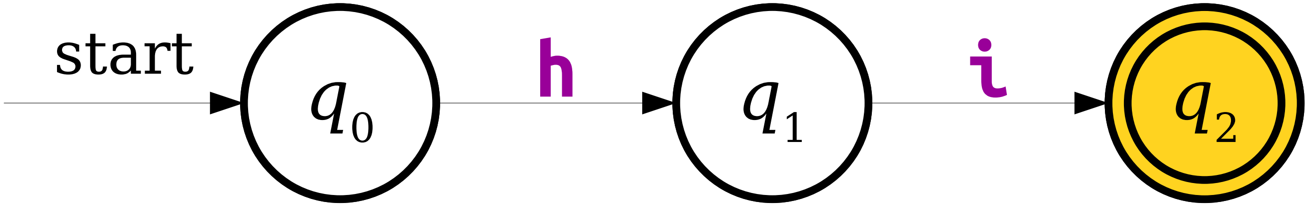
q_0 has two transitions defined on 1!

A More Complex NFA



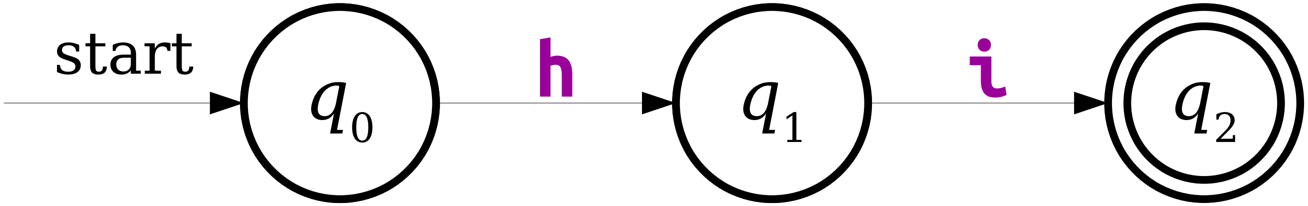
If a NFA needs to make a transition when no transition exists, the automaton **dies** and that particular path does not accept.

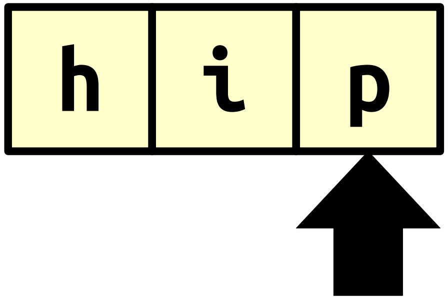
Hello, NFA!



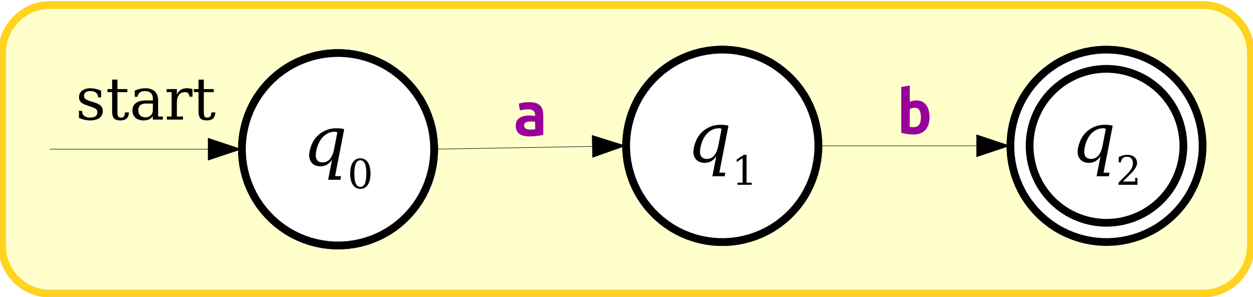
h	i

Tragedy in Paradise



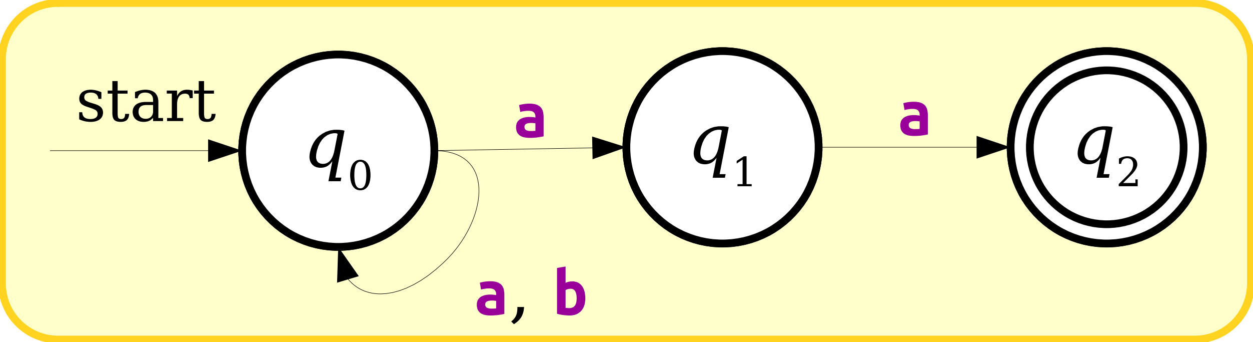


h	i	p

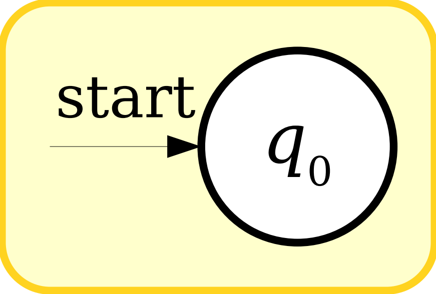


{ab}

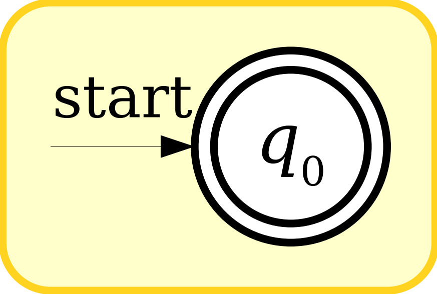
Question to ponder:
Why is the answer
{ $w \in \Sigma^*$ | w ends in **aaa** }
not correct?



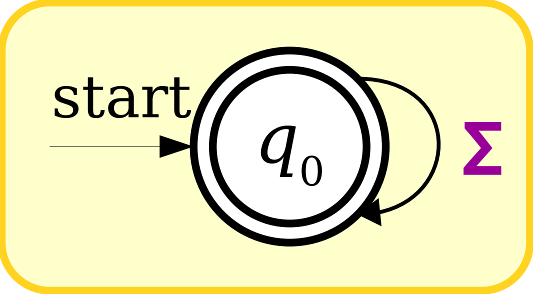
{ $w \in \Sigma^*$ | w ends in **aa** }



Ø



{**ε**}



Σ*

Note that flipping the accept and reject states of an NFA doesn't always give an NFA for the complement of the original language. *(Why?)*

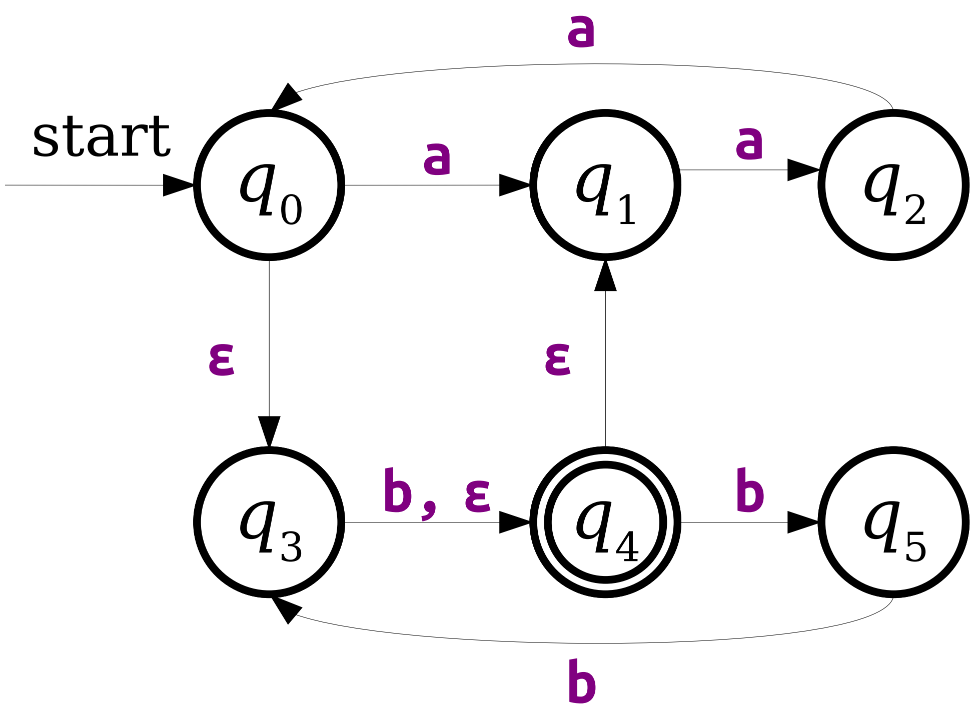
The ***language of an NFA*** is

$\mathscr{L}(N)$ = { $w \in \Sigma^*$ | N accepts w }.

What is the language of each NFA?
(Assume Σ = {a, b}.)

ε-Transitions

- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

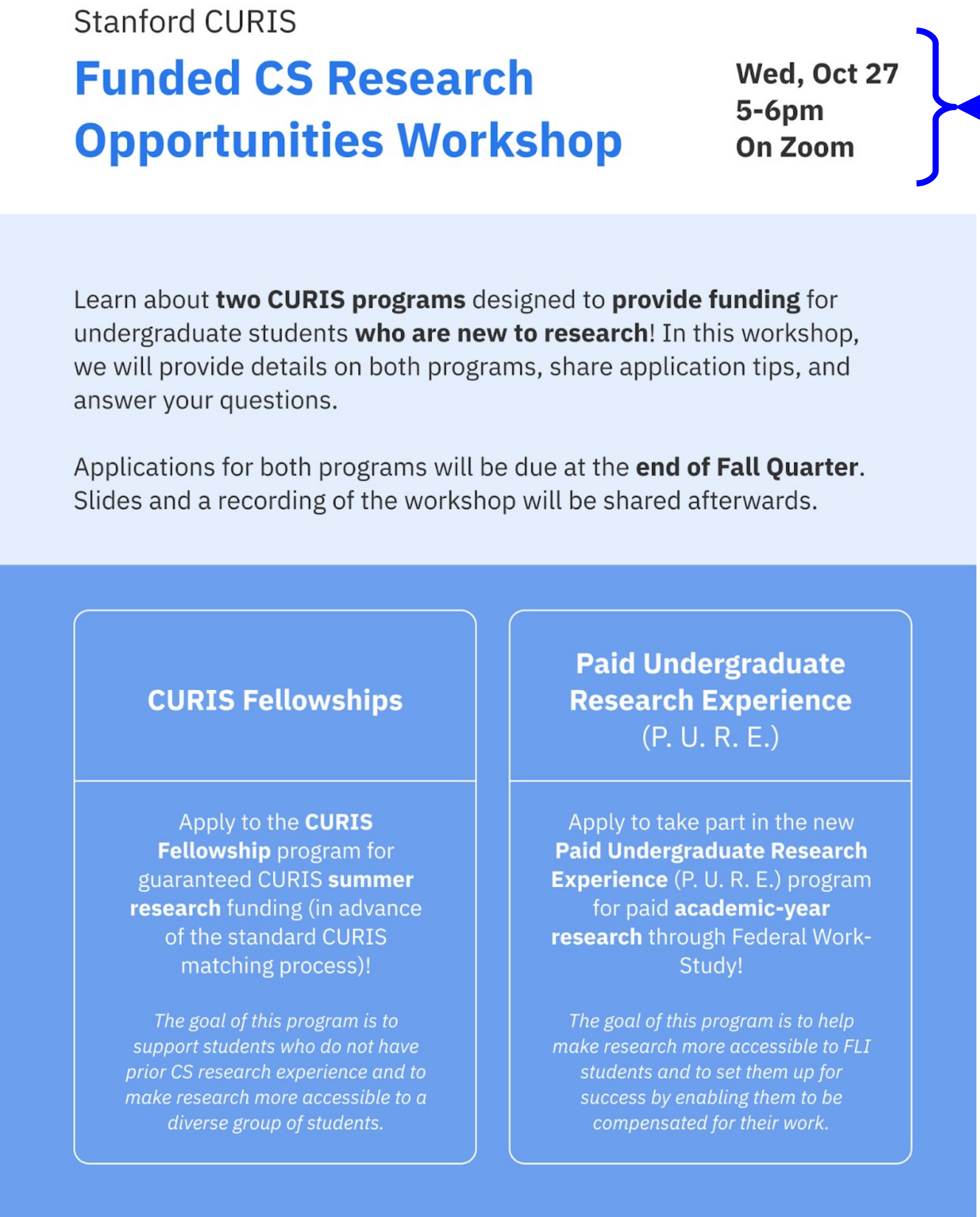


Not at all fun or rewarding exercise: what is the language of this NFA?

ε-Transitions

- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
- NFAs are not *required* to follow ε-transitions. It's simply another option at the machine's disposal.

Time-Out For Announcements!



Stanford CURIS

Funded CS Research Opportunities Workshop

Wed, Oct 27
5-6pm
On Zoom

Learn about **two CURIS programs** designed to **provide funding** for undergraduate students **who are new to research**! In this workshop, we will provide details on both programs, share application tips, and answer your questions.

Applications for both programs will be due at the **end of Fall Quarter**. Slides and a recording of the workshop will be shared afterwards.

CURIS Fellowships

Apply to the **CURIS Fellowship** program for guaranteed CURIS **summer research** funding (in advance of the standard CURIS matching process)!

The goal of this program is to support students who do not have prior CS research experience and to make research more accessible to a diverse group of students.

Paid Undergraduate Research Experience (P. U. R. E.)

Apply to take part in the new **Paid Undergraduate Research Experience** (P. U. R. E.) program for paid **academic-year research** through Federal Work-Study!

The goal of this program is to help make research more accessible to FLI students and to set them up for success by enabling them to be compensated for their work.

Right after class!

https://stanford.zoom.us/j/
99396091443?
pwd=WUxRekt2SVppb3JBKzZQ
U3JVTXd1QT09

Your Questions

"I love all the interesting problems we covered in lectures so far, but I sometimes can't relate what we learn in class about writing proof and discrete math to the CS programming side. I know this is only halfway through the quarter, but will it be clearer about what we learn in class connect to real programming?"

The content for the rest of this quarter contains a bunch of gems that are immediately useful in programming. Automata, for example, are used in the design of UI elements, network controllers, compilers and interpreters, etc.

More generally, the techniques you've learned so far – how to formalize concepts, edge cases in formal logic, etc. – are surprisingly useful in designing complex systems. Building big systems is largely about getting the abstractions right. Concepts like functions, graphs, and the like are supremely useful here. MapReduce is a good example of this, as are the protocols that power the internet. And knowing FOL is useful for figuring out what to do in edge cases in systems.

Back to CS103!

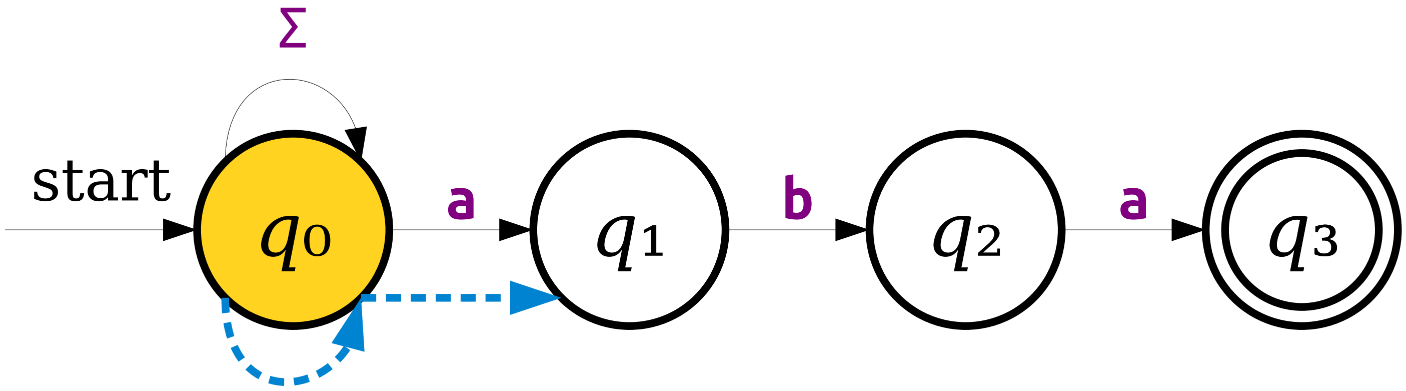
Intuiting Nondeterminism

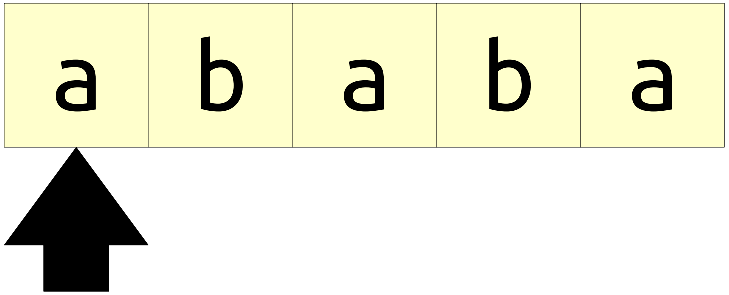
- Nondeterministic machines are a serious departure from physical computers. How can we build up an intuition for them?
- There are two particularly useful frameworks for interpreting nondeterminism:
  - ***Perfect positive guessing***
  - ***Massive parallelism***

Perfect Positive Guessing

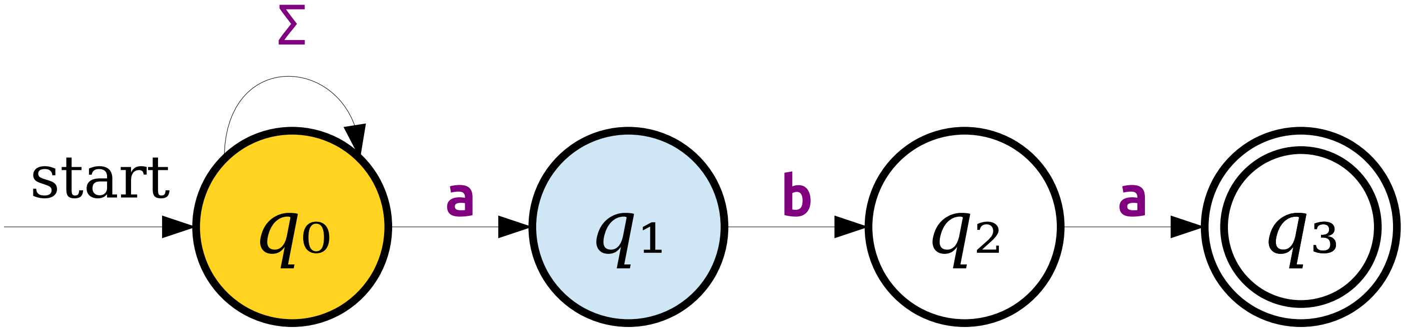
- We can view nondeterministic machines as having ***Magic Superpowers*** that enable them to guess choices that lead to an accepting state.
  - If there is at least one choice that leads to an accepting state, the machine will guess it.
  - If there are no choices, the machine guesses any one of the wrong guesses.
- There is no known way to physically model this intuition of nondeterminism – this is quite a departure from reality!

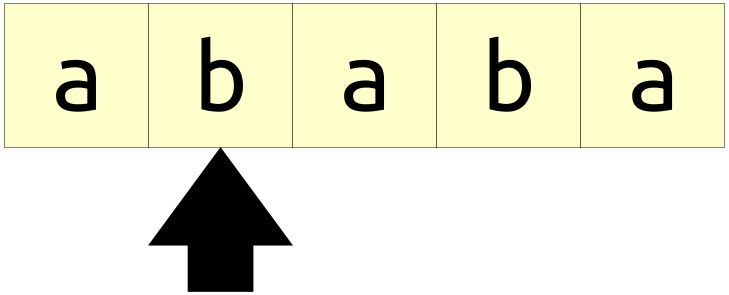
Massive Parallelism



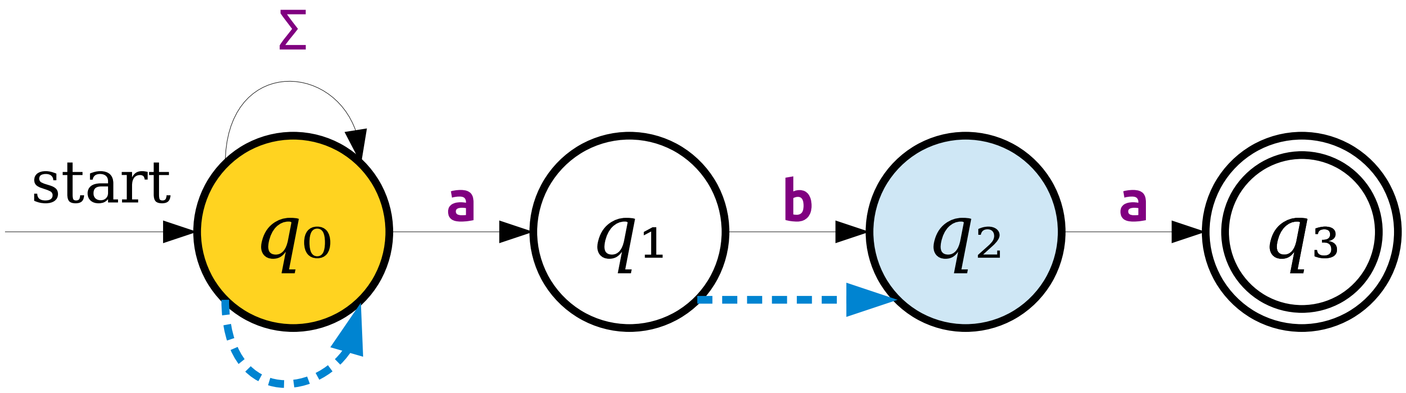


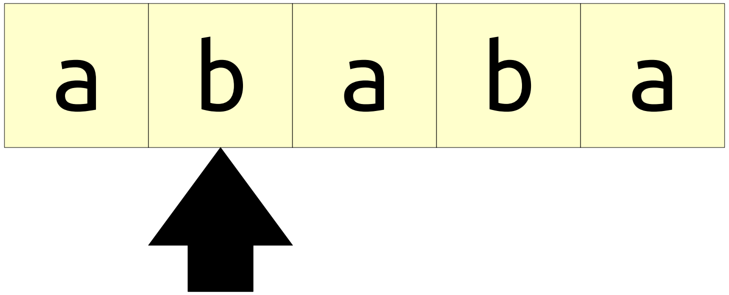
Massive Parallelism



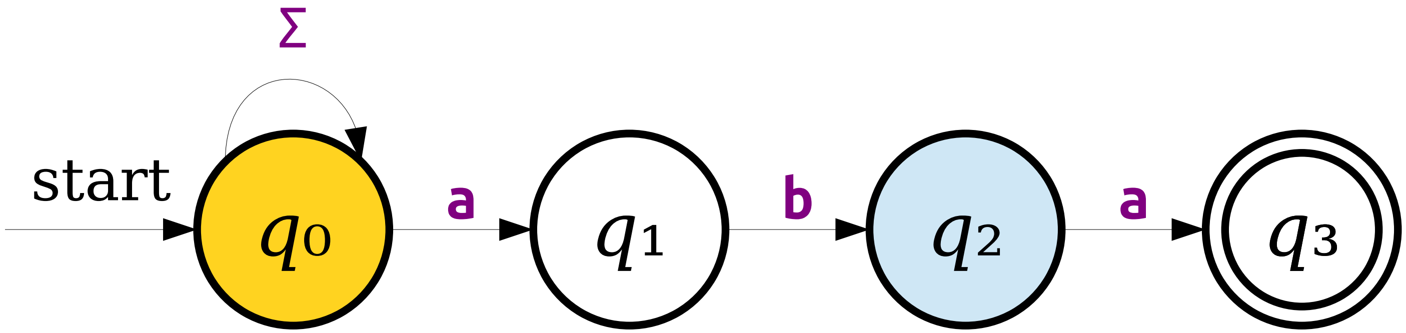


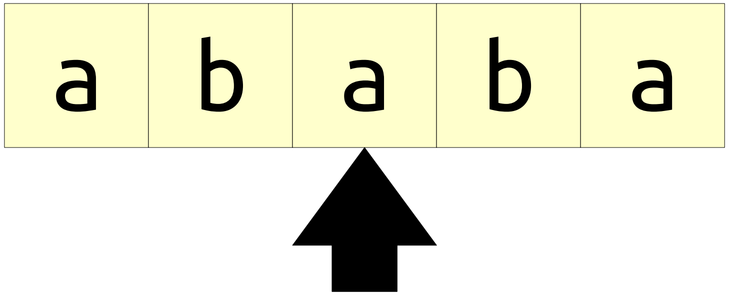
Massive Parallelism



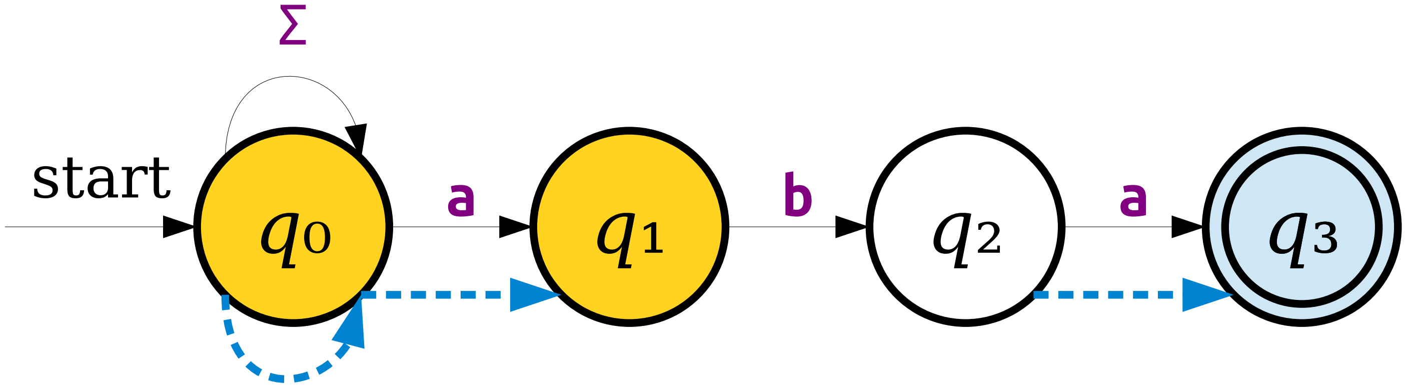


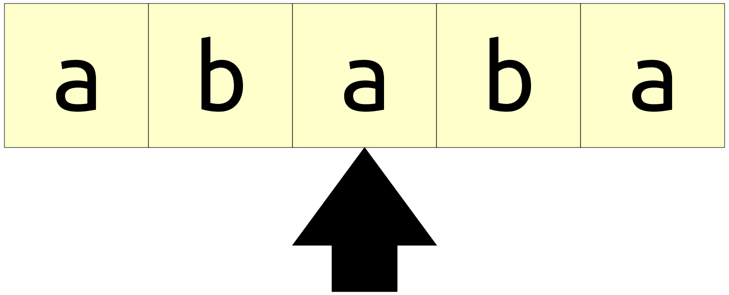
Massive Parallelism



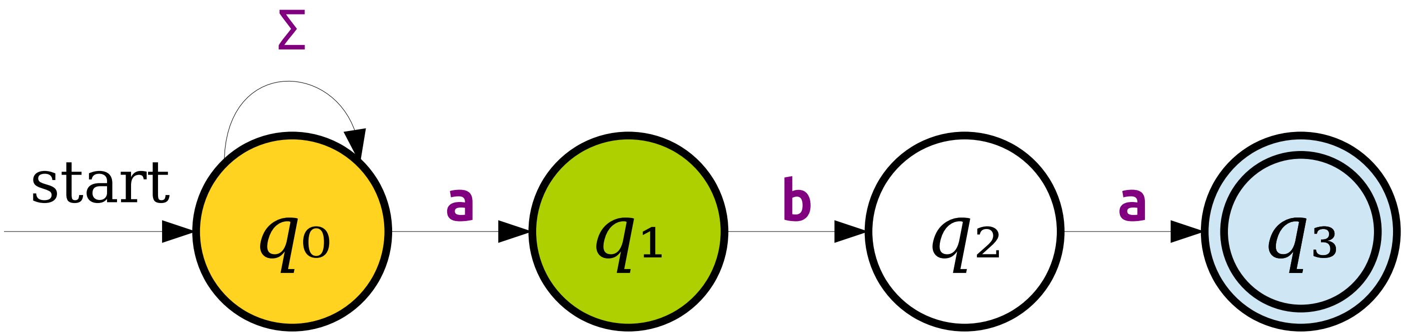


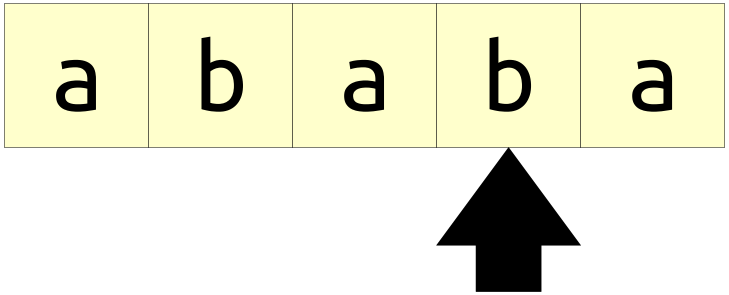
Massive Parallelism





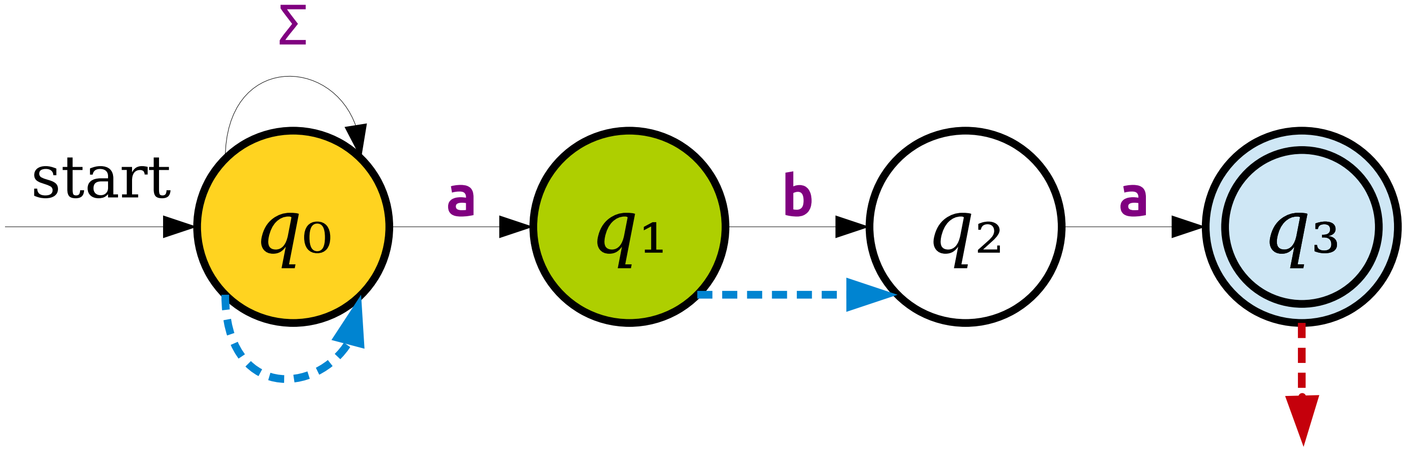
Massive Parallelism

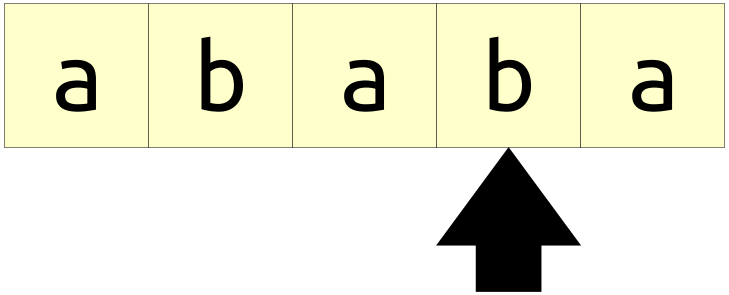




a	b	a	b	a

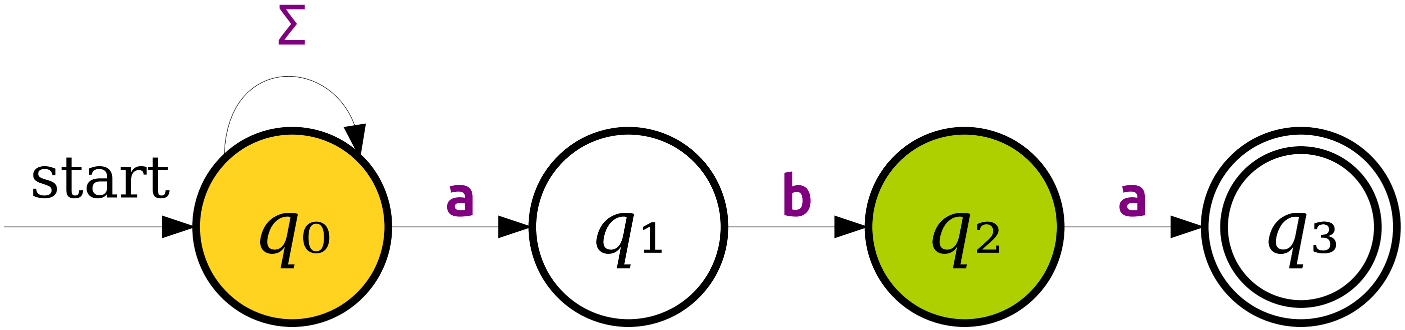
Massive Parallelism

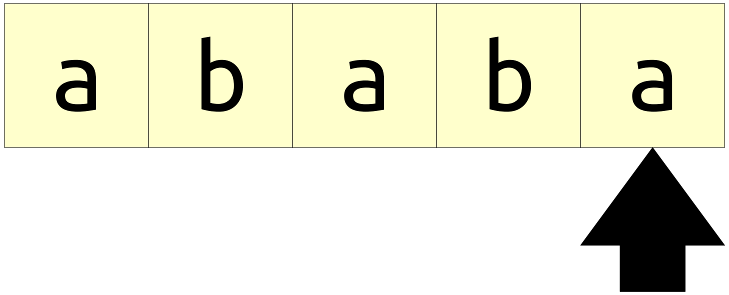




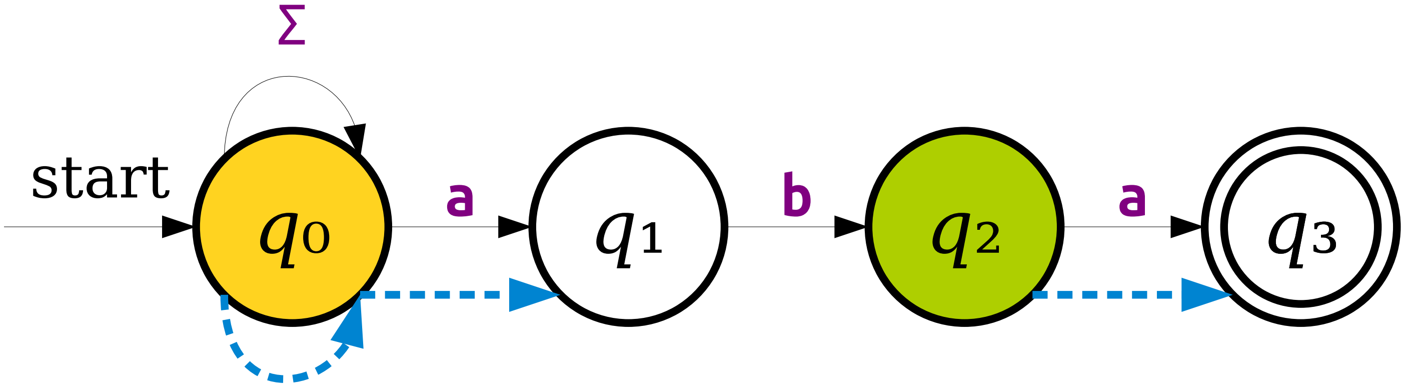
a	b	a	b	a

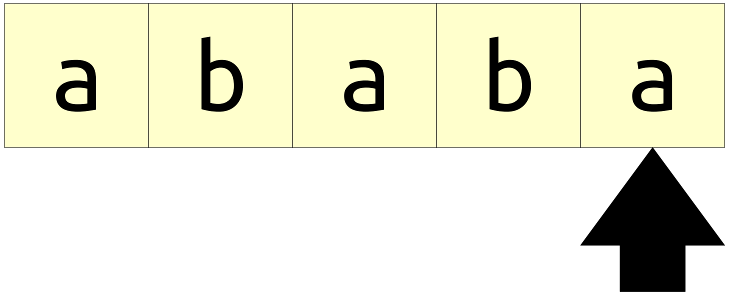
Massive Parallelism



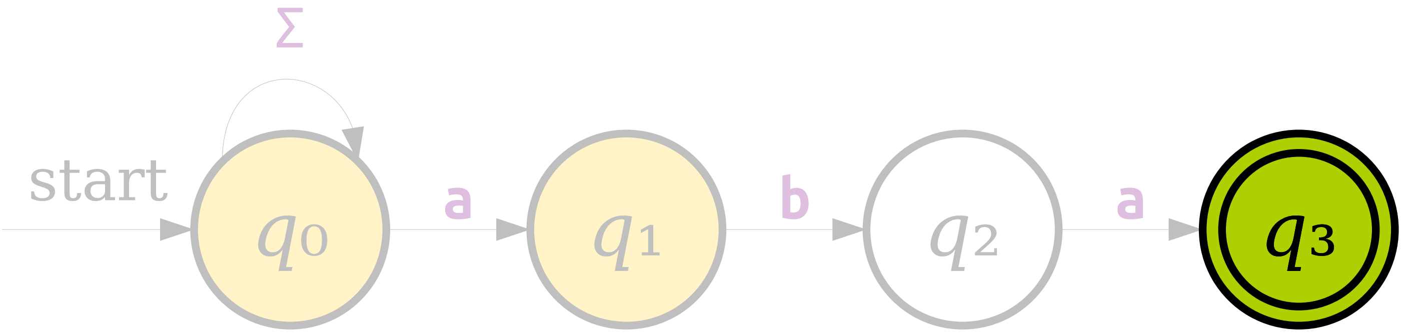


Massive Parallelism





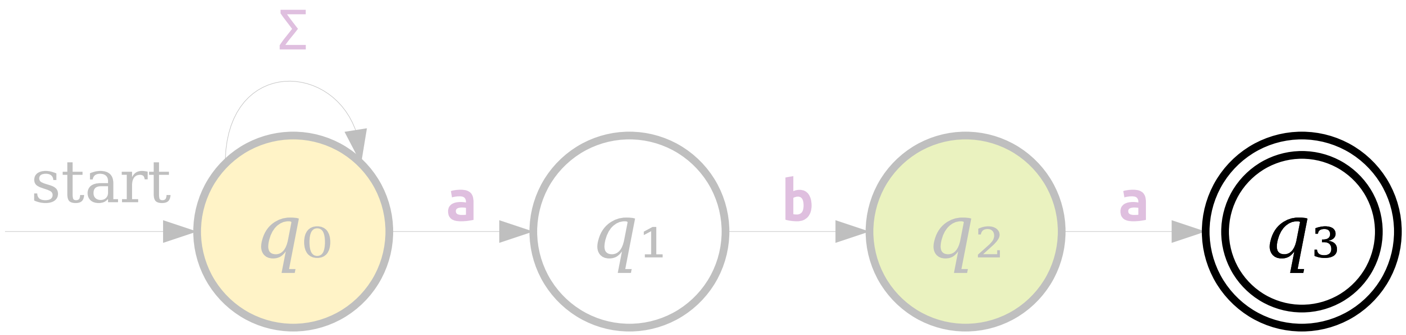
Massive Parallelism



We're in at least one accepting state, so there's some path that gets us to an accepting state.

a	b	a	b	a

Massive Parallelism



a	b	a	b

We're not in any accepting state, so no possible path accepts.

Massive Parallelism

- An NFA can be thought of as a DFA that can be in many states at once.
- At each point in time, when the NFA needs to follow a transition, it tries all the options at the same time.
- (Here's a rigorous explanation about how this works; read this on your own time).
  - Start off in the set of all states formed by taking the start state and including each state that can be reached by zero or more ε-transitions.
  - When you read a symbol `a` in a set of states *S*:
    - Form the set *S'* of states that can be reached by following a single `a` transition from some state in *S*.
    - Your new set of states is the set of states in *S'*, plus the states reachable from *S'* by following zero or more ε-transitions.

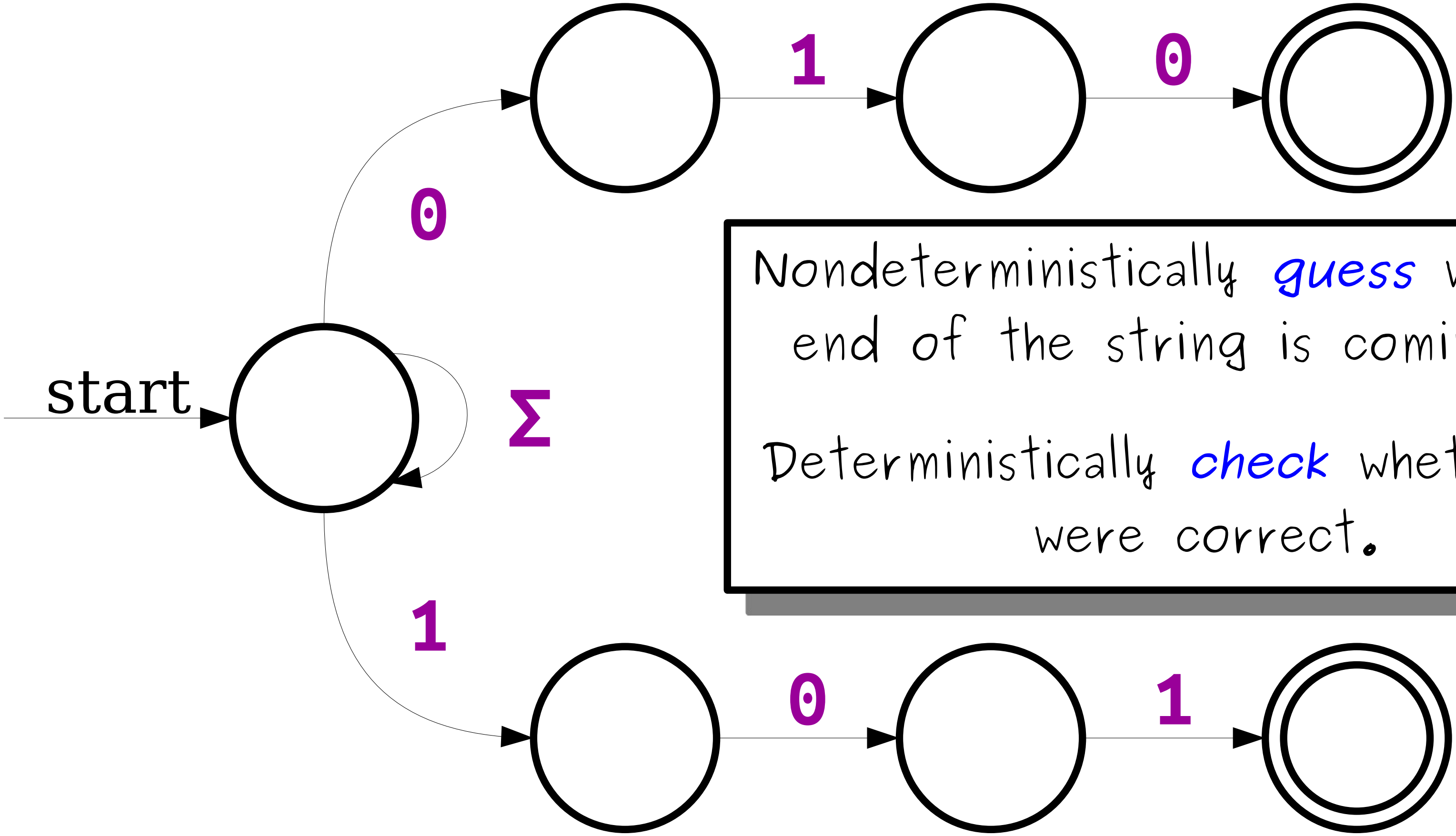
Designing NFAs

Designing NFAs

- ***Embrace the nondeterminism!***
- Good model: ***Guess-and-check***:
  - Is there some information that you'd really like to have? Have the machine *nondeterministically guess* that information.
  - Then, have the machine *deterministically check* that the choice was correct.
- The *guess* phase corresponds to trying lots of different options.
- The *check* phase corresponds to filtering out bad guesses or wrong options.

Guess-and-Check

$L = \{ w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \text{ ends in } \mathtt{010} \text{ or } \mathtt{101} \}$

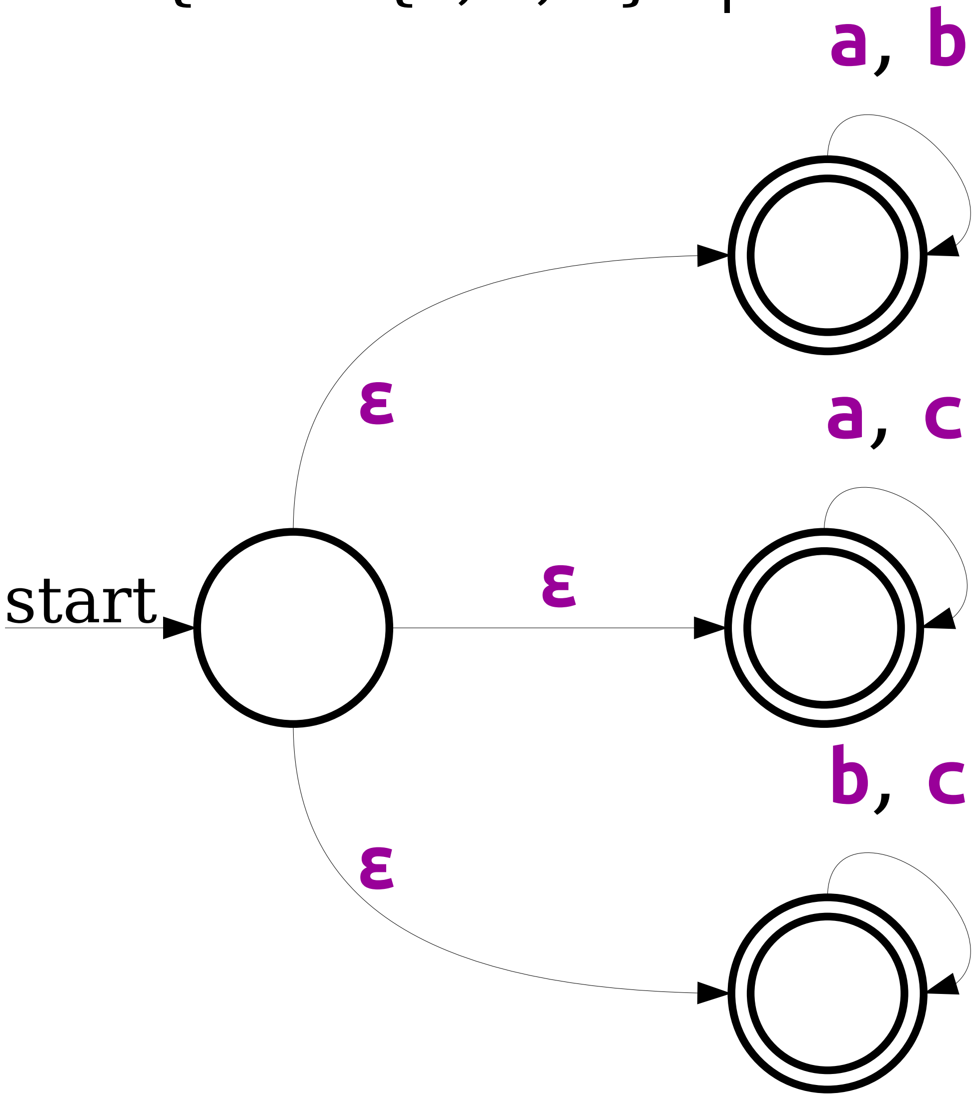


Nondeterministically *guess* when the end of the string is coming up.

Deterministically *check* whether you were correct.

Guess-and-Check

$L = \{ w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid \text{at least one of } \mathtt{a}, \mathtt{b}, \text{ or } \mathtt{c} \text{ is not in } w \}$



Nondeterministically *guess* which character is missing.

Deterministically *check* whether that character is indeed missing.

Just how powerful are NFAs?

Next Time

- ***The Powerset Construction***
  - So beautiful. So elegant. So cool!
- ***More Closure Properties***
  - Other set-theoretic operations.
- ***Language Transformations***
  - What's the deal with the notation Σ*?