

# Module 2

## Finite Automata

The simplest computers for the simplest languages

CS 360: Introduction to the Theory of Computing

Collin Roberts, University of Waterloo

2.1

### Topics of Module 2

- Deterministic finite automata
- Nondeterministic finite automata and the equivalence to DFAs
- $\epsilon$ -NFAs, and their equivalence to DFAs.

2.2

## 1 Deterministic finite automata

### Deterministic finite automata

*Finite automaton*: a simple computing machine

- Given a finite input word, it moves from one program state to another.
- Each move is based on one input letter
- At the end of the input, the machine either *accepts* or *rejects* the input, depending on the machine state.

Vital limitation of a finite automaton:

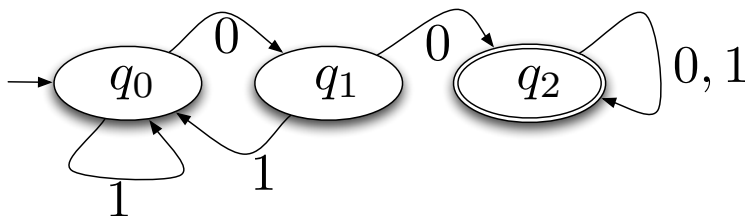
- It cannot look back in its input.
- The only memory is in the state; aside from that, it has forgotten everything.

2.3

### Definitions

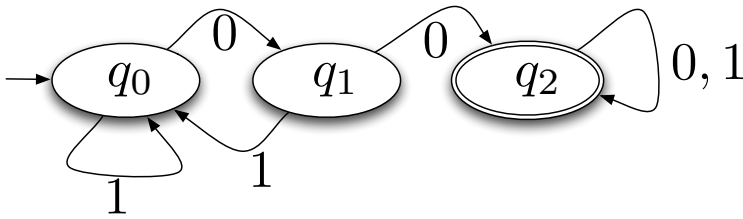
Finite automaton described by 5 parameters:

- $Q$  = set of computation states
- $\Sigma$  = finite input alphabet
- $\delta$  = transition function
  - **Important:** In a DFA, there must be a transition defined for *every state* and for *every possible alphabet character*.
- $q_0$  = start state
- $F$  = accept states



2.4

This DFA



In this finite automaton:

- $Q = \{q_0, q_1, q_2\}$ ,
- $\Sigma = \{0, 1\}$ ,
- $\delta$  is a function from  $Q \times \Sigma \rightarrow Q$ . It includes  $(q_0, 0) \rightarrow q_1$
- $q_0 = q_0$ ,
- $F = \{q_2\}$ .

This DFA *accepts* the *language* of words with 00 as a substring. **Question:** How would you prove that?

2.5

### Acceptance, extension

Given a DFA  $M$ , what does “ $M$  accepts  $w$ ” mean?

- Starting at  $q_0$ , follow transition function  $\delta$  for each letter in  $w$ , in order.
- String  $w$  accepted by  $M$  if at the end of  $w$ 's transitions, we wind up in a state in  $F$ .

A more formal definition of acceptance comes by looking at the *extended transition function*,  $\hat{\delta}$ .

- $\hat{\delta}(q, w)$ : state we finish in if we start at state  $q$  and follow  $\delta$  for each letter in the word  $w$  in turn.

Function  $\hat{\delta}$  is a function from  $Q \times \Sigma^* \rightarrow Q$ . It usually cannot be written down in a closed form, which leads us to the following technique.

2.6

### Formal definition of extended transition function

Formally,  $\hat{\delta}(q, w)$  is defined recursively:

- $\hat{\delta}(q, \varepsilon) = q$ , for all states  $q$ .
- If  $|w| > 0$ , then we can write  $w = xa$ , where  $|a| = 1$ .
- Then define  $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$ .

Unpack that:

- Let  $q_1 = \hat{\delta}(q, x)$ .
- In words,  $q_1$  is the state that we reach starting from  $q$  after we have read the prefix  $x$ .
- Then, process the single letter  $a$ :  $\delta(q_1, a) = \delta(\hat{\delta}(q, x), a)$ .

This can be defined from the other end:

- If  $w = ax$ , then  $\hat{\delta}(q, w) = \hat{\delta}(\delta(q, a), x)$ .
- The textbook gives the first definition, so we will stick with that.

2.7

### Lemma about extended transition function

*Lemma:* Let  $D = (Q, \delta, \Sigma, q_0, F)$  be a DFA, with extended transition function  $\hat{\delta}$ . Let  $q \in Q$  and  $a \in \Sigma$  be arbitrary. Then  $\hat{\delta}(q, a) = \delta(q, a)$ , i.e.  $\hat{\delta}$  agrees with  $\delta$  for any state  $q$  and on any single alphabet symbol  $a$ .

**Proof:**

$$\begin{aligned}\hat{\delta}(q, a) &= \hat{\delta}(q, \varepsilon a) \\ &= \delta(\hat{\delta}(q, \varepsilon), a) \\ &= \delta(q, a).\end{aligned}$$

□

2.8

## Language of an DFA

The *language* of the DFA  $M$ : all words  $M$  accepts.

Formally, acceptance of a word:

- $M = (Q, \Sigma, \delta, q_0, F)$  accepts  $w \in \Sigma^*$  exactly when  $\hat{\delta}(q_0, w) \in F$ .

Language of the DFA: all accepted words.

- $L(M) = \{w \in \Sigma^* \text{ where } \hat{\delta}(q_0, w) \in F\}$ .
- (or just  $\{w \in \Sigma^* \text{ where } M \text{ accepts } w\}$ .)

Terminology:  $L(M)$  can be called:

- The language of the DFA  $M$
- The language *accepted* by the DFA  $M$
- The language *recognized* by the DFA  $M$

2.9

## Do DFAs compute?

In a certain sense, yes.

Example:

- “Is  $x$  a multiple of 3?” can be answered by a DFA.
  - The input word  $w$  is the binary representation of  $x$ .
  - Then the machine accepts  $w$  if  $x$  is a multiple of 3.
- This language,  $L = \{w \mid w \text{ is the binary representation of a number } x \text{ that is divisible by } 3\}$ , is accepted by an DFA.
  - $L$  includes 11, 110, 1111, 0, and does not include 10,  $\epsilon$ .
- So in that sense, yes, they compute.

2.10

## Divisibility DFA

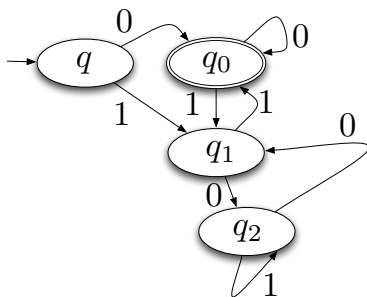
Reminder of conventions:

- start state has an unlabelled arrow
- accept states are double circles
- label arrows with values for  $\delta$

We will have a state for each remainder (0, 1, and 2), plus a special start state, so we do not accept

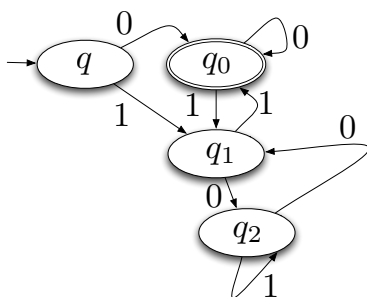
$\epsilon$ .

This gives this DFA:



2.11

## Why does that DFA work?



- Adding a new symbol (0 or 1): double and add the new symbol.
- Double a multiple of 3 and add 0: a new multiple of 3,
- Double a multiple of 3 and add 1: a number with remainder 1,
- Double a number with remainder 1 and add 1: a multiple of 3,
- *etc.*
- We are in state  $q_i$  when  $i$  is the remainder considering what we have already read.

This machine only accepts multiples of 3.

2.12

## Enhancements to DFAs

We are going to prove a theorem soon that FAs accept a specific class of languages, called *regular languages*.

- That should be robust: changes to an FA should not invalidate the property.
- So we will change FAs in a variety of small and large ways.
- The first major change is *nondeterminism*.

2.13

## 2 Nondeterministic Finite Automata

### Nondeterminism

How does a DFA work?

- In a given state, for each input letter, there is exactly one choice for what to do, and
- The machine accepts if after all letters have been read, it is in an accept state.

What if there were choices, instead?

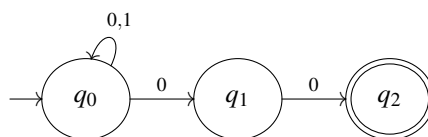
- In a given state, for each letter, there may be a *choice* of what to do.
- The machine accepts if *some* sequence of choices results in an accept state.

Nondeterministic FAs allow a huge blow-up in computation: there will be lots going on in parallel. As such they may not be very realistic models of any kind of computers.

2.14

### An example

$L = \{\text{all words with } 00 \text{ as the last two symbols}\}$



- Nondeterministic FA: Transition from a state, given an alphabet symbol, is to a *set* of possible states.
- *Note*: sometimes a state has no outgoing transition for a given symbol; the second state has no output labelled 1.
- If a thread reaches a state which has no outgoing transition for the given input symbol, then that thread *crashes*; it proceeds no further.

2.15

### Formal definition

NFA defined by 5 parameters

- $Q$  = set of computation states
- $\Sigma$  = finite input alphabet
- $\delta$  = transition function
  - **Important**: In an NFA, there need *not* be a transition defined for every state and for every possible alphabet character.
- $q_0$  = start state

- $F$  = accept states

Differences from DFAs:

- $\delta$ : function from  $Q \times \Sigma \rightarrow \{\text{subsets of } Q\}$ .
  - Recall notation:  $2^Q = \{\text{subsets of } Q\}$ .
  - Based on a state in  $Q$  and an input letter from  $\Sigma$ , which states are now active in  $Q$ ?
  - (Different from DFA, where it is from  $Q \times \Sigma \rightarrow Q$ .)
- Accepts whenever *any* state path from  $q_0$  is to an accept state.

2.16

### New form for extended transition function

We must enhance the definition for our extended transition function. We now need to allow the output of the transition function to be a set of states instead of a single state.

- $\hat{\delta}(q, w)$  = all states that we can reach from start state  $q$  processing the input word  $w$ .
- $\hat{\delta}$ : function  $Q \times \Sigma^* \rightarrow 2^Q$ .
- Base case:  $\hat{\delta}(q, \epsilon) = \{q\}$ .
- Recursive case: If  $|w| > 0$ , then we may write  $w = xa$ , where  $|a| = 1$ .
- Then can we define  $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ ?
  - Well, no.
  - The function  $\delta$  is  $Q \times \Sigma \rightarrow 2^Q$ .
  - But  $\hat{\delta}(q, x)$  is in  $2^Q$  instead of in  $Q$ , so  $\delta(\hat{\delta}(\dots), a)$  is not allowed.
  - We really need to define  $\hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q, x)} \delta(p, a)$  instead.
  - *Note*: This definition handles threads that crash correctly.
    - \* A thread that crashes has no outgoing transition from state  $p$  for input symbol  $a$ .
    - \* In other words,  $\delta(p, a) = \emptyset$ .
    - \* But then,  $\delta(p, a)$  contributes nothing to the union of sets of states, reflecting the fact that the thread has crashed and proceeds no further.

2.17

### Acceptance by an NFA and the language of an NFA

Acceptance of an NFA:

- NFA  $M$  accepts a word  $w$  if  $\hat{\delta}(q_0, w) \cap F$  is nonempty.
- There is a path from  $q_0$ , labelled by letters of  $w$ , winding up in an accept state from  $F$ .

Language of an NFA:

- The language of the NFA  $M = (Q, \Sigma, \delta, q_0, F)$  is:

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}.$$

- That is, all words accepted by the NFA.

2.18

### Where are we?

DFAs:

- Model of computation: finite states, follow single transitions
- Acceptance of a word: transitions lead to an accept state from  $M$
- Language: All accepted words

NFAs:

- Model of computation: finite states, possibly many transitions per letter, or possibly none.
- Acceptance: any path of transitions leads to an accept state.
- The extended transition function is more complicated.
- Language: All accepted words.

Are these different from each other in terms of power?

Both are limited in that they only read each input letter once, left-to-right.

2.19

## NFAs are no more powerful than DFAs

Theorem: Let  $L$  be a language that is accepted by an NFA. Then  $L$  is accepted by a DFA.

- *Note:* The opposite direction is easy: DFAs are NFAs!  
(Well, must change  $\delta$  trivially, so that the NFA transitions to the 1-element set corresponding to the transition in the DFA.)
- Need to show: Given an NFA, we can construct a DFA that accepts its language.

2.20

## Outline of proof

Here is how we will do this:

- Given an NFA  $N$ , we will construct a DFA,  $D$ .
- Then, we will have to show that  $L(D) = L(N)$ .
- As this is an equality of sets, we need to show that every word in  $L(N)$  is in  $L(D)$ , and every word in  $L(D)$  is in  $L(N)$ .

You have seen a little of this in CS 241.

First, let's build the DFA  $D$ , using the same alphabet  $\Sigma$  that  $N$  uses.

*Remember:* The machine  $D$  does *not* have to have the same states as  $N$ , just the same alphabet and language!

2.21

## Computation in an NFA

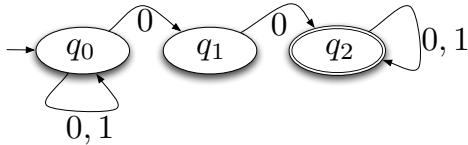
How does the NFA  $N$  work?

- If  $N$  is in state  $q$ , after processing one letter  $a$ ,  $N$  could be in any state from the set  $\delta(q, a)$ .
- Then,  $N$  processes the next letter and winds up in any of another set of states.  
(That is what is built into the extended transition function,  $\hat{\delta}_N$ .)
- In the DFA  $D$ , a single state represents a *set of states* in the NFA  $N$ .
- When  $D$  reads a new letter in, we jump from one state in the DFA to another (corresponding to the appropriate sets of states in the NFA  $N$ ).

2.22

## Sketch of the process of the proof

This NFA accepts the language  $L = \{\text{words with } 00 \text{ as a substring}\}$ . How would you prove that?



- Its only accept state is  $q_2$ .
- Suppose we have processed some letters, and the NFA could be in either state  $q_0$  or  $q_1$ , and the next input letter is 1.
  - From  $q_0$  we go to  $q_0$ .
  - From  $q_1$ , no transition labelled 1.
- The new DFA, if it is in the *set* state  $\{q_0, q_1\}$  and reads in a 1, must go to the state corresponding to the set  $\{q_0\}$ .

2.23

## Sketch of the process of the proof

Here is idea of the algorithm for the *subset construction* for the transition function of  $D$ :

- For each subset  $S \subseteq Q$  of states from  $N$ :
  - Recall that  $S$  corresponds to a single state in  $D$ .
  - For each alphabet symbol,  $a$ :
    - \* For each state  $p \in S$ , consider  $\delta(p, a)$  (recall, this is a set of states).
    - \* Gather all of these together. Let  $T = \bigcup_{p \in S} \delta(p, a)$ .
    - \* Then  $T$  also corresponds to a single state in  $D$ .
    - \* Add the transition  $S \xrightarrow{a} T$  to the transition function  $\delta_D$  for  $D$ .

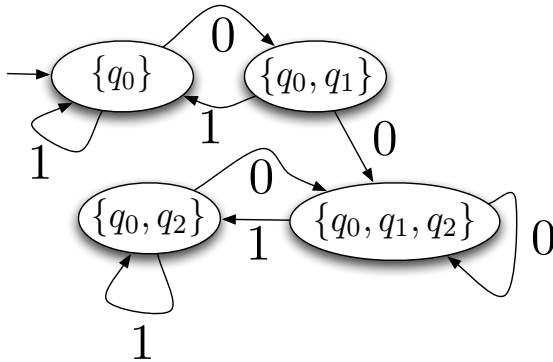
2.24

### This procedure may make many new states

This 3-state NFA turns into a 4-state DFA.

- In general, if the NFA  $N$  has  $k$  states, the DFA  $D$  could have  $2^k$  states.

Here is the DFA:

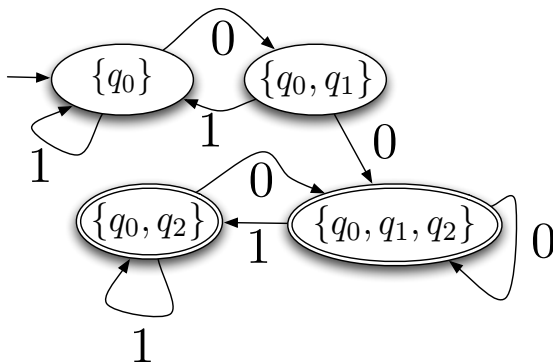


And what are accept states?

- $F_D = \{\text{States in } D \text{ that represent sets of states in } N \text{ that include at least one accept state from } F_N\}$ .

2.25

### The full DFA



- Convince yourself that accepts the same language as the original construction.
- Interestingly, the DFA state  $\{q_0, q_2\}$  is not needed, since we only get to it from another accept state.
- We really only needed 3 states.

Now, let's generalize this idea.

2.26

### Subset construction (formal)

Given NFA  $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ , construct a new DFA  $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ , with these parameters:

- $Q_D = 2^{Q_N}$ .
- Let  $S \in Q_D$ , i.e. think of  $S$  as a set of states from the definition of  $N$ . Then
- $F_D = \{S \in Q_D \mid S \cap F_N \neq \emptyset\}$
- That is, each acceptance state in  $D$  corresponds to a set of states in  $N$  with at least one accept state.
- And a more complicated transition function:

$$\begin{aligned} \delta_D(S, a) &= \{\text{all states reachable in } N \text{ from } S \text{ when we read } a\} \\ &= \bigcup_{p \in S} \delta_N(p, a). \end{aligned}$$

- Let  $D$ 's initial state be  $\{q_0\}$ , where  $q_0$  is the initial state of  $N$ .

This is a DFA, not an NFA. We know which state  $D$  is in after reading any alphabet symbol.

2.27

### The languages are equal

Now we must show that the languages of the NFA  $N$  and the DFA  $D$  equal. Think about which words are in the languages of  $N$  and of  $D$ .

- $w \in L(N) \Leftrightarrow \hat{\delta}_N(q_0, w) \cap F_N \neq \emptyset$ .
- $w \in L(D) \Leftrightarrow \hat{\delta}_D(\{q_0\}, w) \in F_D$ .

By the definition of  $F_D$ , the second statement is equivalent to:

- $w \in L(D) \Leftrightarrow \hat{\delta}_D(\{q_0\}, w) \cap F_N \neq \emptyset$ .

We must show these are the same: that if  $w \in L(D)$ , then  $w \in L(N)$ , and vice versa.

2.28

### Proof

Must show:  $w \in L(N) \Leftrightarrow w \in L(D)$ .

- That is:  $\hat{\delta}_N(q_0, w) \cap F_N \neq \emptyset \Leftrightarrow \hat{\delta}_D(\{q_0\}, w) \cap F_N \neq \emptyset$ .
- It suffices to show:  $\hat{\delta}_N(q, w) = \hat{\delta}_D(\{q\}, w)$ , for any state  $q$  from the definition of  $N$ .
- (If one of these sets has a non-empty intersection with  $F_N$ , then so does the other)

Proof: By induction on  $|w|$ .

- Base case ( $|w| = 0$ ): Thus  $w = \epsilon$ . Then  $\hat{\delta}_N(q, \epsilon) = \{q\} = \hat{\delta}_D(\{q\}, \epsilon)$ .
- Inductive case ( $|w| > 0$ ): The inductive hypothesis is that, for every  $x$  with  $|x| < |w|$ , we have  $\hat{\delta}_N(q, x) = \hat{\delta}_D(\{q\}, x)$ , for any state  $q$  from  $N$ .
- Since  $|w| > 0$ , we may write  $w = xa$ , where  $|a| = 1$ .
- Then the induction hypothesis applies to  $x$ , so  $\hat{\delta}_N(q, x) = \hat{\delta}_D(\{q\}, x)$ .

2.29

### Inductive case of the proof

- The induction hypothesis is that  $\hat{\delta}_N(q, x) = \hat{\delta}_D(\{q\}, x)$ .
- We need to show that  $\hat{\delta}_N(q, xa) = \hat{\delta}_D(\{q\}, xa)$ .
- So now we process the last character,  $a$ , on both sides.

$$\begin{aligned}
 \hat{\delta}_D(\{q\}, xa) & \stackrel{\text{Definition of } \hat{\delta}_D \text{ for the DFA } D}{=} \delta_D(\hat{\delta}_D(\{q\}, x), a) \\
 & \stackrel{\text{induction hypothesis}}{=} \delta_D(\hat{\delta}_N(q, x), a) \\
 & \stackrel{\text{Definition of } \delta_D}{=} \bigcup_{p \in \hat{\delta}_N(q, x)} \delta_N(p, a) \\
 & \stackrel{\text{Definition of } \hat{\delta}_N}{=} \hat{\delta}_N(q, xa).
 \end{aligned}$$

Reading one more symbol in  $N$ , the set of states of  $N$  we can be in corresponds with the state we will be in in  $D$  (recall that a single state of  $D$  represents a set of states of  $N$ ).

- The NFA  $N$  accepts the same language as the DFA  $D$ .

The class of languages accepted by NFAs is the same as for DFAs.

2.30



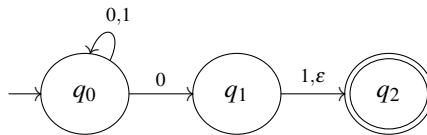
### 3 epsilon-NFAs

#### Another expansion to NFA's: $\epsilon$ -transitions

Sometimes in designing an NFA, it is handy to have transitions that happen automatically, without reading any letters of the input.

- Machine models that allow this are  $\epsilon$ -NFAs.
- An  $\epsilon$ -NFA is a 5-tuple, like an NFA.
- The only difference is transition function:
- $\delta$  is now a function:  $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ .
- (Transitions may exist that do not consume input letters.)

**Example:** The  $\epsilon$ -NFA



Accepts binary words ending with 0 or with 01. (We could do this without  $\epsilon$ -transitions, by making the second state an accept state.)

2.31

#### Formality about $\epsilon$ -NFAs

How do their extended transition functions look?

- Which states could I be in after processing the word  $x$ ?
- Any states I could be in after just  $x$ , with no  $\epsilon$ -transitions, plus
- Any states after just  $x$ , and *one*  $\epsilon$  transition, plus
- Any states after just  $x$ , and *two*  $\epsilon$  transitions, and so on ...
- ...until we exhaust all possible  $\epsilon$ -transitions.

Unroll this to get a recursive definition for  $\hat{\delta}$ .

2.32

#### Defining the $\epsilon$ -closure of a state

The definitions should be fairly similar:

- $\hat{\delta}(q, y)$ : states we can get to after processing  $y$  and having  $\epsilon$ -transitions.

Let's start with  $\epsilon$ -transitions:

- Let  $\text{ECLOSE}(p)$  = all states reachable starting from  $p$  only using  $\epsilon$ -transitions.
- We can define  $\text{ECLOSE}(p)$  recursively:

- $p \in \text{ECLOSE}(p)$
- If  $q \in \text{ECLOSE}(p)$ , then so are all of the states in  $\delta(q, \epsilon)$ .

The set  $\text{ECLOSE}(p)$  is called the  $\epsilon$ -closure of the state  $p$ .

We will also allow  $\text{ECLOSE}(S)$  to be defined for a set  $S$  of states via:

$$\text{ECLOSE}(S) = \bigcup_{s \in S} \text{ECLOSE}(s).$$

2.33

#### Defining the $\epsilon$ -closure of a state

**Lemma:** For an  $\epsilon$ -NFA  $E$ , subset  $S \subseteq Q$  and decomposition  $S = \bigcup_i S_i$ ,

$$\bigcup_i \text{ECLOSE}(S_i) = \text{ECLOSE}\left(\bigcup_i S_i\right),$$

(i.e. taking  $\varepsilon$ -closure commutes with taking set unions). **Proof:**

$$\begin{aligned}
 \bigcup_i \text{ECLOSE}(S_i) & \stackrel{\text{Definition of ECLOSE}(S_i)}{=} \bigcup_i \left( \bigcup_{s \in S_i} \text{ECLOSE}(s) \right) \\
 & \stackrel{S = \bigcup_i S_i}{=} \bigcup_{s \in S} \text{ECLOSE}(s) \\
 & \stackrel{\text{Definition of ECLOSE}(S)}{=} \text{ECLOSE}(S) \\
 & \stackrel{S = \bigcup_i S_i}{=} \text{ECLOSE} \left( \bigcup_i S_i \right). \quad \square
 \end{aligned}$$

2.34

### The definition of the extended transition function

We can define  $\hat{\delta}$  for  $\varepsilon$ -NFA's, also recursively.

- Base case:  $\hat{\delta}(q, \varepsilon) = \text{ECLOSE}(q)$ : states reachable from  $q$  with  $\varepsilon$ -transitions
- Inductive case: Suppose that  $w = xa$ , where  $a \in \Sigma$ . (*Note:  $a$  cannot be  $\varepsilon$ , which is not a member of  $\Sigma$ .*)
  - We know that  $P = \hat{\delta}(q, x)$  is the set of all states in  $Q$  that we can get to by following either edges for the letters of  $x$  or  $\varepsilon$ -transitions (including  $\varepsilon$ -transitions at the end of  $x$ ).
  - Then, we must follow the transitions for the alphabet symbol  $a$ : Let  $R = \bigcup_{p \in P} \delta(p, a)$ ; then  $R$  has all of the states we can get to from  $P$  after following a transition for  $a$ .
  - Last, we might have some more  $\varepsilon$ -transitions.
- So,  $\hat{\delta}(q, w) = \text{ECLOSE}(R) = \text{ECLOSE} \left( \bigcup_{p \in \hat{\delta}(q, x)} \delta(p, a) \right)$

2.35

### Languages and power of $\varepsilon$ -NFAs

- Language of an  $\varepsilon$ -NFA:  $L = \left\{ w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset \right\}$
- That is,  $\hat{\delta}(q_0, x)$  includes an accept state.

Are  $\varepsilon$ -NFAs more powerful?

- No.
- Theorem: Given an  $\varepsilon$ -NFA  $E$ , there exists an ordinary DFA  $D$  such that  $L(D) = L(E)$ .
- This is not very surprising: we must show that we can include the  $\varepsilon$ -transitions of  $E$  in the transition function  $\delta_D$  for  $D$ .
- (The other direction is just by definition: a DFA is an  $\varepsilon$ -NFA, once we make some trivial changes to the structure of  $\delta$  so that it produces a 1-element set when it reads a symbol and the empty set when it reads in  $\varepsilon$ .)

To prove the theorem, we must construct a DFA  $D$  accepting language  $L(E)$ .

2.36

### The equivalent DFA

- Both machines use the same alphabet, of course.
- We use the subset construction, as when we built the DFA for an NFA.
- Starting state is  $q_D = \text{ECLOSE}(q_0)$ . Thus, we start having implicitly taken  $\varepsilon$ -transitions from the starting state  $q_0$  of  $E$ .

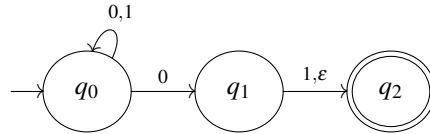
The complexity comes in the transition function and the accept states.

- Transition function:
  - From one DFA state,  $S$  (corresponding to a set of states in  $E$ ), if we process one letter  $a$  in the new DFA, we should mimic this behaviour from  $E$ :

- \* follow any edges labelled  $a$
- \* take any  $\varepsilon$ -transitions
- From any one state from  $E$ , say  $q$ , this then takes us to:
  - \*  $\delta_E(q, a)$
  - \*  $\text{ECLOSE}(\delta_E(q, a))$
- And we therefore want the union over all states  $q \in S$ :  $\delta_D(S, a) = \bigcup_{q \in S} \text{ECLOSE}(\delta_E(q, a))$ .

### Example of Subset Construction

- Here we demonstrate one step in the subset construction for the earlier small example of an  $\varepsilon$ -NFA:



- The subset construction gives us

$$Q_D = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$$

$$q_D = \text{ECLOSE}(q_0) = \{q_0\}$$

- Now we determine  $\delta_D(S, a)$ , where  $S = \{q_0, q_1, q_2\}$  (the state we reach from  $\{q_0\}$  upon reading 0) and  $a = 1$ .
- Computing  $\text{ECLOSE}(\delta_E(p, a))$  for each  $p \in S$  gives

$$\begin{aligned} \text{ECLOSE}(\delta_E(q_0, 1)) &= \text{ECLOSE}(\{q_0\}) = \{q_0\} \\ \text{ECLOSE}(\delta_E(q_1, 1)) &= \text{ECLOSE}(\{q_2\}) = \{q_2\} \\ \text{ECLOSE}(\delta_E(q_2, 1)) &= \text{ECLOSE}(\emptyset) = \emptyset \end{aligned}$$

### Example of Subset Construction

- Hence the target state coming from the subset construction is  $\{q_0, q_2\}$ .
- The construction says that we need to add to the transition function for  $D$ :  $\delta_D(\{q_0, q_1, q_2\}, 1) = \{q_0, q_2\}$ .
- Now to complete the transition function for  $D$ , we do this same construction for each of the 8 choices for  $S$  (on the previous slide) and each alphabet symbol from  $\Sigma = \{0, 1\}$ .

### Picking the accept states, equality of languages

- We need to figure out which states are accepting states:
  - $F_D = \{S \mid S \in Q_D \text{ and } S \cap F_E \neq \emptyset\}$ .
- Declare a word accepted by  $D$  if  $D$  is in an accept state (according to this recipe) when  $D$  finishes processing the word.
- Now, to show equality of the languages of the two automata, we must show that if  $x$  is accepted by  $E$ , then  $x$  is accepted by  $D$ , and vice versa.
  - (One concern: do we do the right thing for the word  $\varepsilon$ ?)
  - To show  $L(E) = L(D)$ , can show that  $\hat{\delta}_E(q_0, x) = \hat{\delta}_D(q_D, x)$ ?
  - If so, then we will be in the same set of  $\varepsilon$ -NFA states after reading  $x$ , and our definitions of  $F_D$  and  $F_E$  will guarantee that both machines will accept exactly the same words.

## Transition functions

We want to show:  $\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$ , for all strings  $w$ . The proof is by induction on  $|w|$ .

Base case ( $|w| = 0$ ): In this case,  $w = \varepsilon$ . We have  $\hat{\delta}_E(q_0, \varepsilon) = \text{ECLOSE}(\{q_0\})$ , by definition of  $\hat{\delta}_E$  in the  $\varepsilon$ -NFA.

- We therefore have

$$\begin{aligned}
 \hat{\delta}_E(q_0, \varepsilon) & \underbrace{=}_{\varepsilon\text{-NFA rule}} \text{ECLOSE}(\{q_0\}) \\
 & \underbrace{=}_{\text{singleton set property}} \text{ECLOSE}(q_0) \\
 & \underbrace{=}_{\text{Definition of } q_D} q_D \\
 & \underbrace{=}_{D \text{ is a DFA}} \hat{\delta}_D(q_D, \varepsilon) \\
 & \underbrace{=}_{\text{Definition of } q_D} \hat{\delta}_D(\text{ECLOSE}(q_0), \varepsilon).
 \end{aligned}$$

- So the base case holds.

2.41

## Transition functions

Inductive case ( $|w| > 0$ ):

- We need to argue that  $\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$ .
- The induction hypothesis is that  $\hat{\delta}_E(q_0, x) = \hat{\delta}_D(q_D, x)$ , for all strings  $x$  where  $|x| < |w|$ .
- Write  $w = xa$ , where  $a$  is a single character.
- The induction hypothesis applies to  $x$ .
- Thus we may let  $\hat{\delta}_E(q_0, x) = \hat{\delta}_D(q_D, x) = S$ .
- Now we compute

2.42

## Transition functions

$$\begin{aligned}
 \hat{\delta}_E(q_0, w) & \underbrace{=}_{w=xa} \hat{\delta}_E(q_0, xa) \\
 & \underbrace{=}_{\text{Definition of } \hat{\delta}_E} \text{ECLOSE} \left( \bigcup_{p \in \hat{\delta}_E(q_0, x)} \delta_E(p, a) \right) \\
 & \underbrace{=}_{\text{Definition of } S} \text{ECLOSE} \left( \bigcup_{p \in S} \delta_E(p, a) \right) \\
 & \underbrace{=}_{\text{Lemma}} \bigcup_{p \in S} \text{ECLOSE}(\delta_E(p, a)) \\
 & \underbrace{=}_{\text{Definition of } \delta_D} \delta_D(S, a) \\
 & \underbrace{=}_{\text{Definition of } S} \delta_D(\hat{\delta}_D(\text{ECLOSE}(q_0), x), a) \\
 & \underbrace{=}_{\text{Definition of } \hat{\delta}_D} \hat{\delta}_D(\text{ECLOSE}(q_0), xa) \underbrace{=}_{w=xa} \hat{\delta}_D(\text{ECLOSE}(q_0), w).
 \end{aligned}$$

2.43

## Transition functions

- **Remark:** You should convince yourself that the base case handles the input word  $\varepsilon$  correctly.
- This argument did not use any special properties of  $q_0$ .
- We could re-run the argument with any state,  $q$ .
- Thus we could have proved, for any state  $q$  and any word  $w \in \Sigma^*$ :

$$\hat{\delta}_E(q, w) = \hat{\delta}_D(\text{ECLOSE}(q), x).$$

- We have proved by induction that the two extended transition functions agree on all input words.
- Therefore, as we argued earlier, this shows that the two automata accept precisely the same languages.
- So we are done.

2.44

## End of module 2

Hence, the class of languages accepted by  $\varepsilon$ -NFAs is the same as the class accepted by ordinary NFAs, which is the same as the class of languages accepted by DFAs.

We have now seen a collection of types of automata:

- Deterministic finite automata
- Nondeterministic finite automata
- $\varepsilon$ -NFAs

All three accept the same class of languages. But what is that class?

They are the *regular languages*.

2.45