Chapter Three: Closure Properties for Regular Languages

Once we have defined some languages formally, we can consider combinations and modifications of those languages: unions, intersections, complements, and so on. Such combinations and modifications raise important questions. For example, is the intersection of two regular languages also regular—capable of being recognized directly by some DFA?

Outline

Language Complement

- For any language L over an alphabet Σ, the *complement* of L is

$$\overline{L} = \left\{ x \in \Sigma^* \mid x \notin L \right\}$$

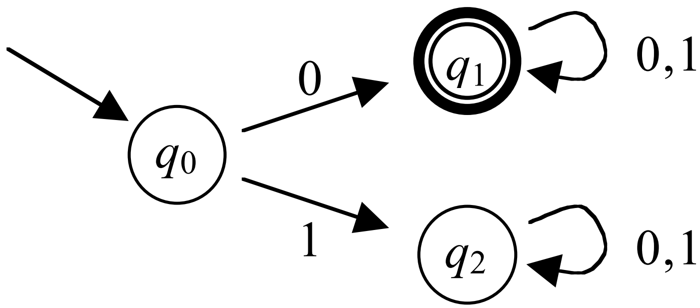
- Example:

$$L = \left\{ 0x \mid x \in \{0,1\}^* \right\} = \text{ strings that start with 0}$$

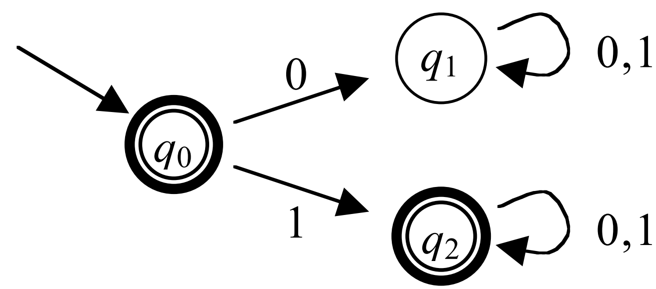
$$\overline{L} = \left\{ 1x \mid x \in \{0,1\}^* \right\} \cup \{\varepsilon\} = \text{ strings that } \textit{don't} \text{ start with 0}$$

- Given a DFA for any language, it is easy to construct a DFA for its complement

Example



$$L = \left\{ 0x \mid x \in \{0,1\}^* \right\}$$



$$\overline{L} = \left\{ 1x \mid x \in \{0,1\}^* \right\} \cup \{\varepsilon\}$$

Complementing a DFA

- All we did was to make the accepting states be non-accepting, and make the non-accepting states be accepting
- In terms of the 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, all we did was to replace F with Q-F
- Using this construction, we have a proof that the complement of any regular language is another regular language

Theorem 3.1

The complement of any regular language is a regular language.

- Let L be any regular language
- By definition there must be some DFA $M = (Q, \Sigma, \delta, q_0, F)$ with $L(M) = L$
- Define a new DFA $M' = (Q, \Sigma, \delta, q_0, Q\text{-}F)$
- This has the same transition function δ as M, but for any string $x \in \Sigma^*$ it accepts x if and only if M rejects x
- Thus $L(M')$ is the complement of L
- Because there is a DFA for it, we conclude that the complement of L is regular

Closure Properties

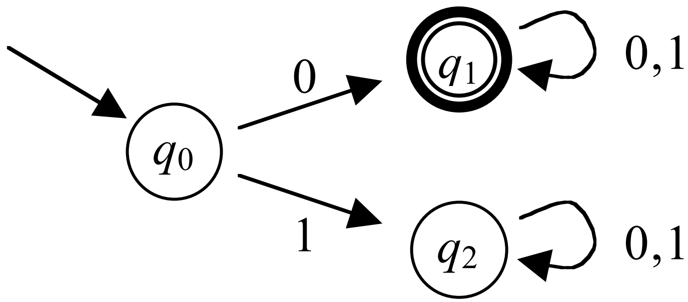
- A shorter way of saying that theorem: the regular languages are *closed under complement*
- The complement operation cannot take us out of the class of regular languages
- Closure properties are useful shortcuts: they let you conclude a language is regular without actually constructing a DFA for it

Outline

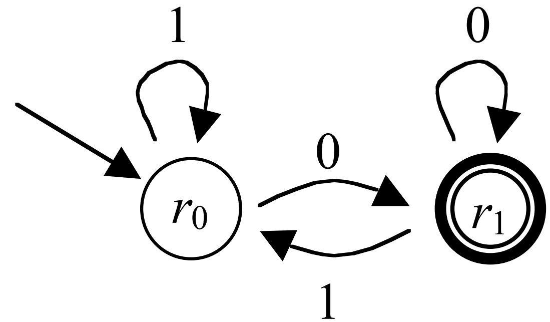
- 3.1 Closed Under Complement
- 3.2 Closed Under Intersection
- 3.3 Closed Under Union
- 3.4 DFA Proofs Using Induction
- 3.5 A Mystery DFA

Language Intersection

- $L_1 \cap L_2 = \{x \mid x \in L_1 \text{ and } x \in L_2\}$
- Example:
  - $L_1 = \{0x \mid x \in \{0,1\}^*\}$ = strings that start with 0
  - $L_2 = \{x0 \mid x \in \{0,1\}^*\}$ = strings that end with 0
  - $L_1 \cap L_2 = \{x \in \{0,1\}^* \mid x \text{ starts and ends with } 0\}$
- Usually we will consider intersections of languages with the same alphabet, but it works either way
- Given two DFAs, it is possible to construct a DFA for the intersection of the two languages

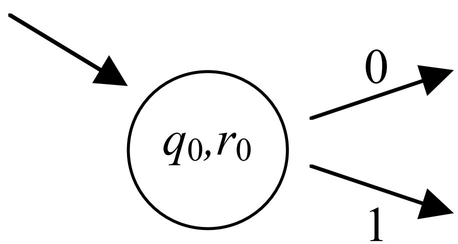


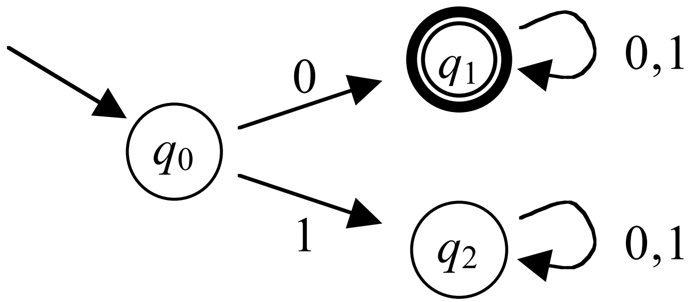
$\{0x \mid x \in \{0,1\}^*\}$



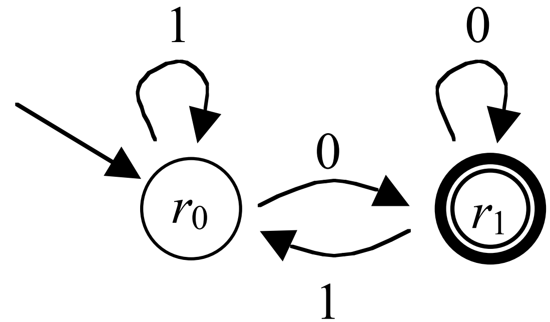
$\{x0 \mid x \in \{0,1\}^*\}$

- We'll make a DFA that keeps track of the pair of states (q_i, r_j) the two original DFAs are in
- Initially, they are both in their start states:



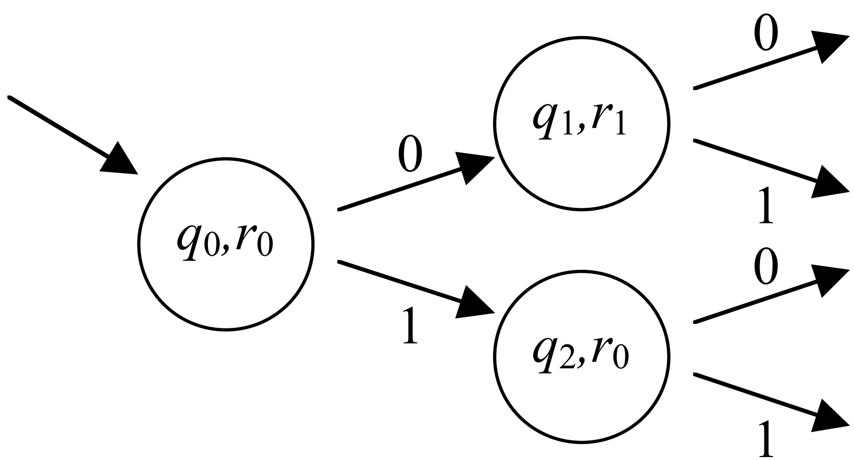


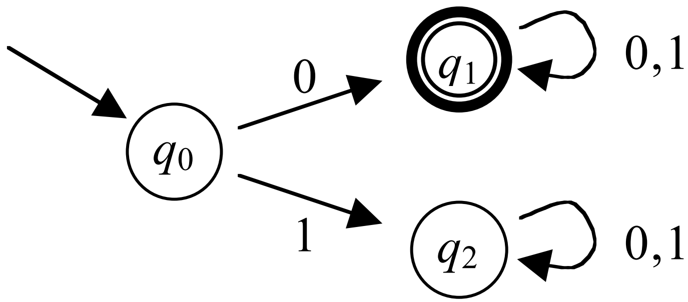
$\{0x \mid x \in \{0,1\}^*\}$



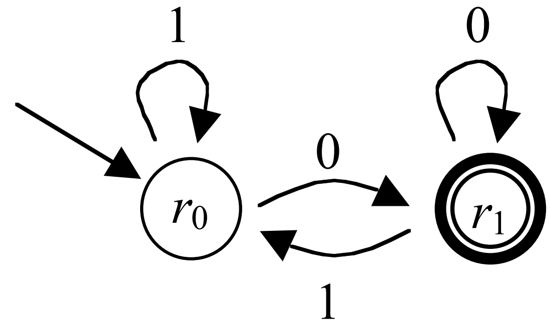
$\{x0 \mid x \in \{0,1\}^*\}$

- Working from there, we keep track of the pair of states (q_i, r_j):



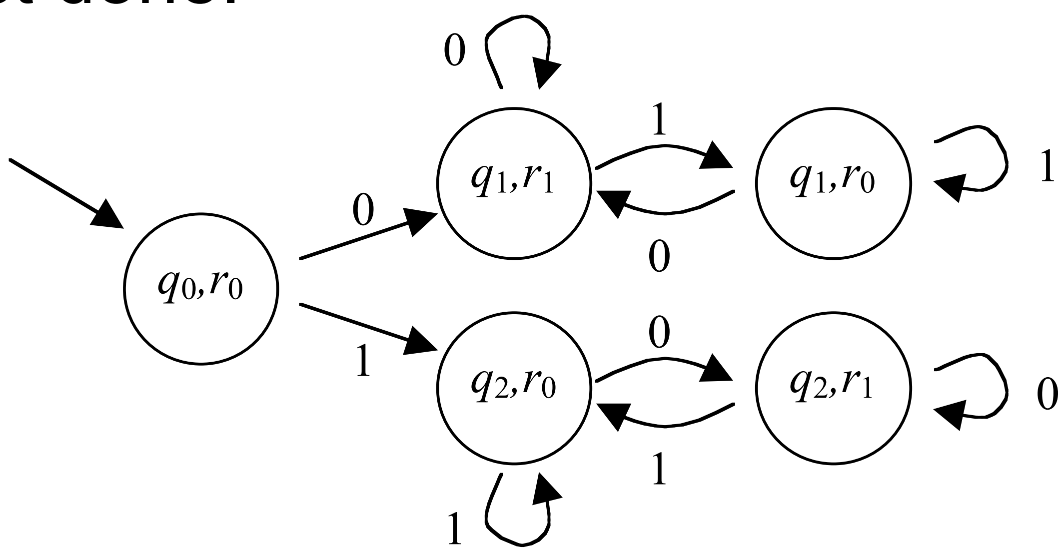


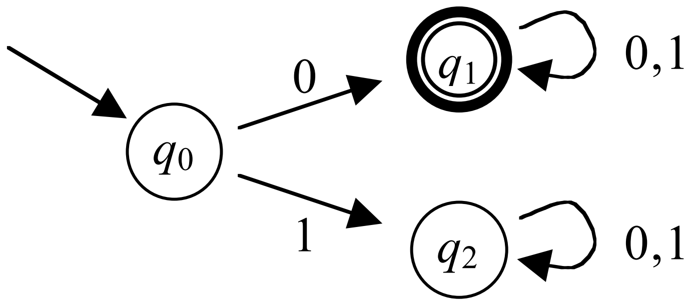
$\{0x \mid x \in \{0,1\}^*\}$



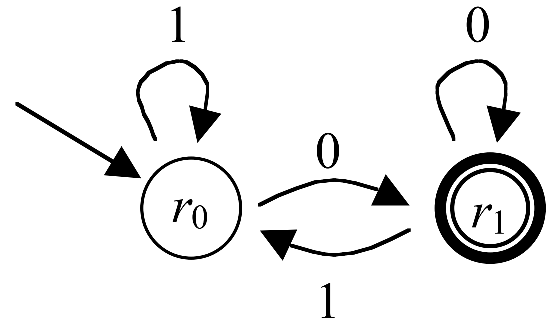
$\{x0 \mid x \in \{0,1\}^*\}$

- Eventually state-pairs repeat; then we're almost done:



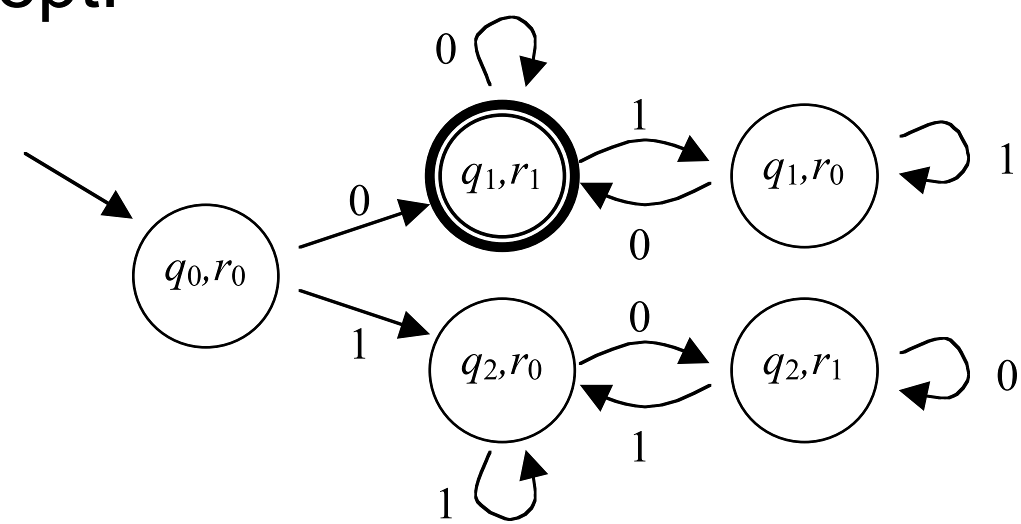


$\{0x \mid x \in \{0,1\}^*\}$



$\{x0 \mid x \in \{0,1\}^*\}$

- For intersection, both original DFAs must accept:



Cartesian Product

- In that construction, the states of the new DFA are pairs of states from the two originals
- That is, the state set of the new DFA is the *Cartesian product* of the two original sets:

$$S_1 \times S_2 = \{(e_1, e_2) \mid e_1 \in S_1 \text{ and } e_2 \in S_2\}$$

- The construct we just saw is called the *product construction*

Theorem 3.2

If L_1 and L_2 are any regular languages, $L_1 \cap L_2$ is also a regular language.

- Let L_1 and L_2 be any regular languages
- By definition there must be DFAs for them:
  - $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ with $L(M_1) = L_1$
  - $M_2 = (R, \Sigma, \delta_2, r_0, F_2)$ with $L(M_2) = L_2$
- Define a new DFA $M_3 = (Q \times R, \Sigma, \delta, (q_0,r_0), F_1 \times F_2)$
- For δ, define it so that for all $q \in Q$, $r \in R$, and $a \in \Sigma$, we have $\delta((q,r),a) = (\delta_1(q,a), \delta_2(r,a))$
- M_3 accepts if and only if both M_1 and M_2 accept
- So $L(M_3) = L_1 \cap L_2$, so that intersection is regular

Notes

- Formal construction assumed that the alphabets were the same
  - It can easily be modified for differing alphabets
  - The alphabet for the new DFA would be $\Sigma_1 \cap \Sigma_2$
- Formal construction generated all pairs
  - When we did it by hand, we generated only those pairs actually reachable from the start pair
  - Makes no difference for the language accepted

Outline

- 3.1 Closed Under Complement
- 3.2 Closed Under Intersection
- 3.3 Closed Under Union
- 3.4 DFA Proofs Using Induction
- 3.5 A Mystery DFA

Language Union

- $L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2 \text{ (or both)}\}$
- Example:
  - $L_1 = \{0x \mid x \in \{0,1\}^*\}$ = strings that start with 0
  - $L_2 = \{x0 \mid x \in \{0,1\}^*\}$ = strings that end with 0
  - $L_1 \cup L_2 = \{x \in \{0,1\}^* \mid x \text{ starts with 0 or ends with 0 (or both)}\}$
- Usually we will consider unions of languages with the same alphabet, but it works either way

Theorem 3.3

If L_1 and L_2 are any regular languages,
$L_1 \cup L_2$ is also a regular language.

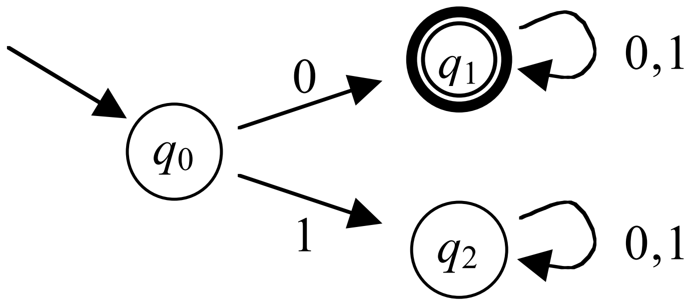
- Proof 1: using DeMorgan's laws
  - Because the regular languages are closed for intersection and complement, we know they must also be closed for union:

$$L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$$

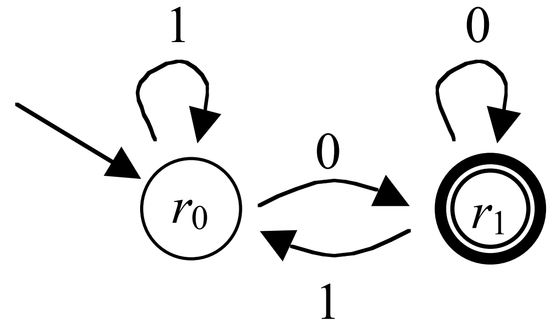
Theorem 3.3

If L_1 and L_2 are any regular languages,
$L_1 \cup L_2$ is also a regular language.

- Proof 2: by product construction
  - Same as for intersection, but with different accepting states
  - Accept where either (or both) of the original DFAs accept
  - Accepting state set is $(F_1 \times R) \cup (Q \times F_2)$

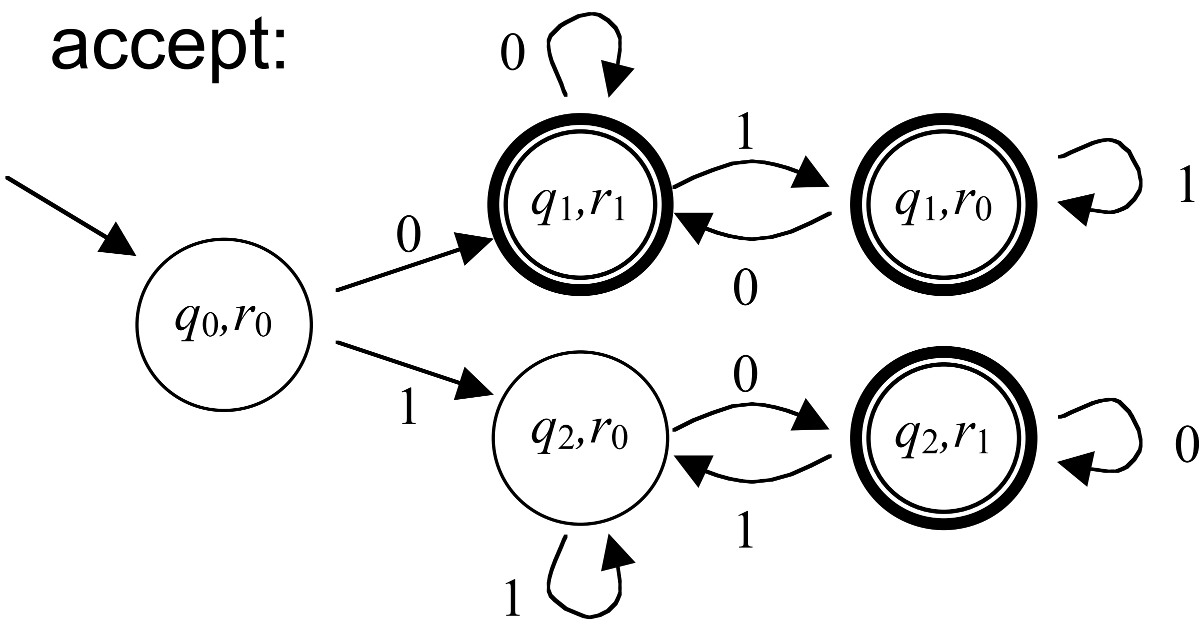


$\{0x \mid x \in \{0,1\}^*\}$



$\{x0 \mid x \in \{0,1\}^*\}$

- For union, at least one original DFA must accept:



Outline

- 3.1 Closed Under Complement
- 3.2 Closed Under Intersection
- 3.3 Closed Under Union
- 3.4 DFA Proofs Using Induction
- 3.5 A Mystery DFA

Proof Technique: Induction

- Mathematical induction and DFAs are a good match
  - You can learn a lot about DFAs by doing inductive proofs on them
  - You can learn a lot about proof technique by proving things about DFAs
- We'll start with an example
- Consider again the proof of Theorem 3.2...

Review: Theorem 3.2

If L_1 and L_2 are any regular languages, $L_1 \cap L_2$ is also a regular language.

- Let L_1 and L_2 be any regular languages
- By definition there must be DFAs for them:
  - $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ with $L(M_1) = L_1$
  - $M_2 = (R, \Sigma, \delta_2, r_0, F_2)$ with $L(M_2) = L_2$
- Define a new DFA $M_3 = (Q{\times}R, \Sigma, \delta, (q_0,r_0), F_1{\times}F_2)$
- For δ, define it so that for all $q \in Q$, $r \in R$, and $a \in \Sigma$, we have $\delta((q,r),a) = (\delta_1(q,a), \delta_2(r,a))$

  (big step)
- M_3 accepts if and only if both M_1 and M_2 accept
- So $L(M_3) = L_1 \cap L_2$, so that intersection is regular

A Big Jump

- There's a big jump between these steps:
  - For δ, define it so that for all $q \in Q$, $r \in R$, and $a \in \Sigma$, we have $\delta((q,r),a) = (\delta_1(q,a), \delta_2(r,a))$
  - M_3 accepts if and only if both M_1 and M_2 accept
- To make that jump, we need to get from the definition of δ to the behavior of δ^*
- We need a lemma like this (Lemma 3.4):

In the product construction, for all $x \in \Sigma^*$,
$\delta^*((q_0,r_0),x) = (\delta_1^*(q_0,x), \delta_2^*(r_0,x))$

Lemma 3.4, When $|x| = 0$

In the product construction, for all $x \in \Sigma^*$,
$\delta^*((q_0,r_0),x) = (\delta_1^*(q_0,x), \delta_2^*(r_0,x))$

- It is not hard to prove for particular fixed lengths of *x*
- For example, when $|x| = 0$:

$$
\begin{aligned}
&\delta^*((q_0,r_0), x) && \\
&\quad = \delta^*((q_0,r_0), \varepsilon) && \text{(since } |x| = 0\text{)} \\
&\quad = (q_0,r_0) && \text{(by the definition of } \delta^*\text{)} \\
&\quad = (\delta_1^*(q_0, \varepsilon), \delta_2^*(r_0, \varepsilon)) && \text{(by the definitions of } \delta_1^* \text{ and } \delta_2^*\text{)} \\
&\quad = (\delta_1^*(q_0, x), \delta_2^*(r_0, x)) && \text{(since } |x| = 0\text{)}
\end{aligned}
$$

Lemma 3.4, When $|x| = 1$

In the product construction, for all $x \in \Sigma^*$,
$\delta^*((q_0,r_0),x) = (\delta_1^*(q_0,x), \delta_2^*(r_0,x))$

- Assuming we have already proved the case $|x| = 0$
- Now, $|x| = 1$:

$\delta^*((q_0,r_0), x)$
$= \delta^*((q_0,r_0), ya)$ (for some symbol a and string y)
$= \delta(\delta^*((q_0,r_0), y), a)$ (by the definition of δ^*)
$= \delta((\delta_1^*(q_0, y), \delta_2^*(r_0, y)), a)$ (using Lemma 3.4 for $|y| = 0$)
$= (\delta_1(\delta_1^*(q_0, y), a), \delta_2(\delta_2^*(r_0, y), a))$ (by the construction of δ)
$= (\delta_1^*(q_0, ya), \delta_2^*(r_0, ya))$ (by the definitions of δ_1^* and δ_2^*)
$= (\delta_1^*(q_0, x), \delta_2^*(r_0, x))$ (since $x = ya$)

Lemma 3.4, When $|x| = 2$

In the product construction, for all $x \in \Sigma^*$,

$$\delta^*((q_0,r_0),x) = (\delta_1^*(q_0,x), \delta_2^*(r_0,x))$$

- Assuming we have already proved the case $|x| = 1$
- Almost no change for $|x| = 2$ (changes in red):

$\delta^*((q_0,r_0), x)$

$= \delta^*((q_0,r_0), ya)$	(for some symbol a and string y)
$= \delta(\delta^*((q_0,r_0), y), a)$	(by the definition of δ^*)
$= \delta((\delta_1^*(q_0, y), \delta_2^*(r_0, y)), a)$	(using Lemma 3.4 for $\|y\| = 1$)
$= (\delta_1(\delta_1^*(q_0, y), a), \delta_2(\delta_2^*(r_0, y), a))$	(by the construction of δ)
$= (\delta_1^*(q_0, ya), \delta_2^*(r_0, ya))$	(by the definitions of δ_1^* and δ_2^*)
$= (\delta_1^*(q_0, x), \delta_2^*(r_0, x))$	(since $x = ya$)

A Never-Ending Proof

- We could easily go on to prove the lemma for |*x*| = 3, 4, 5, 6, and so on
- Each proof would use the fact that the lemma was already proved for shorter strings
- But what we need is a finite proof that Lemma 3.4 holds for all the infinitely many different lengths of *x*

Inductive Proof Of Lemma 3.4

- Our proof of Lemma 3.4 has two parts:
  - Base case: show that it holds when $|x| = 0$
  - Inductive case: show that whenever it holds for some length $|x| = n$, it also holds for $|x| = n+1$
- By induction, we conclude it holds for all $|x|$

In the product construction, for all $x \in \Sigma^*$,
$\delta^*((q_0,r_0),x) = (\delta_1^*(q_0,x), \delta_2^*(r_0,x))$

Proof: by induction on $|x|$.

Base case: when $|x| = 0$, we have:

$\delta^*((q_0,r_0), x)$

$= \delta^*((q_0,r_0), \varepsilon)$	(since $|x| = 0$)
$= (q_0,r_0)$	(by the definition of δ^*)
$= (\delta_1^*(q_0, \varepsilon), \delta_2^*(r_0, \varepsilon))$	(by the definitions of δ_1^* and δ_2^*)
$= (\delta_1^*(q_0, x), \delta_2^*(r_0, x))$	(since $|x| = 0$)

Inductive case: when $|x| > 0$, we have:

$\delta^*((q_0,r_0), x)$

$= \delta^*((q_0,r_0), ya)$	(for some symbol a and string y)
$= \delta(\delta^*((q_0,r_0), y), a)$	(by the definition of δ^*)
$= \delta((\delta_1^*(q_0, y), \delta_2^*(r_0, y)), a)$	(by inductive hypothesis, since $|y| < |x|$)
$= (\delta_1(\delta_1^*(q_0, y), a), \delta_2(\delta_2^*(r_0, y), a))$	(by the construction of δ)
$= (\delta_1^*(q_0, ya), \delta_2^*(r_0, ya))$	(by the definitions of δ_1^* and δ_2^*)
$= (\delta_1^*(q_0, x), \delta_2^*(r_0, x))$	(since $x = ya$)

Inductive Proof

- Every inductive proof has these parts:
  - One or more base cases, with stand-alone proofs
  - One or more inductive cases whose proofs depend on…
  - …an inductive hypothesis: the assumption that the thing you're trying to prove is true for simpler cases
- In our proof, we had:
  - $|x| = 0$ as the base case
  - $|x| > 0$ as the inductive case
  - For the inductive hypothesis, the assumption that the lemma holds for any string y with $|y| < |x|$

Induction And Recursion

- Proof with induction is like programming with recursion
- Our proof of Lemma 3.4 is a bit like a program for making a proof for any size *x*

```
void proveit(int n) {
  if (n==0) {
    base case: prove for empty string
  }
  else {
    proveit(n-1);
    prove for strings of length n, assuming n-1 case proved
  }
}
```

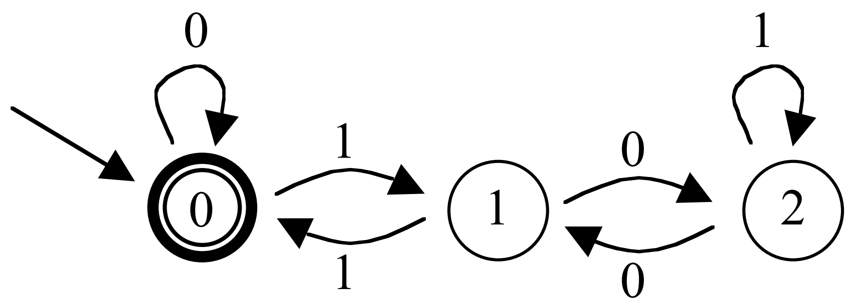
General Induction

- Our proof used induction on the length of a string, with the empty string as the base case
- That is a common pattern for proofs involving DFAs
- But there are as many different patterns of inductive proof as there are patterns of recursive programming
- We will see other varieties later

Outline

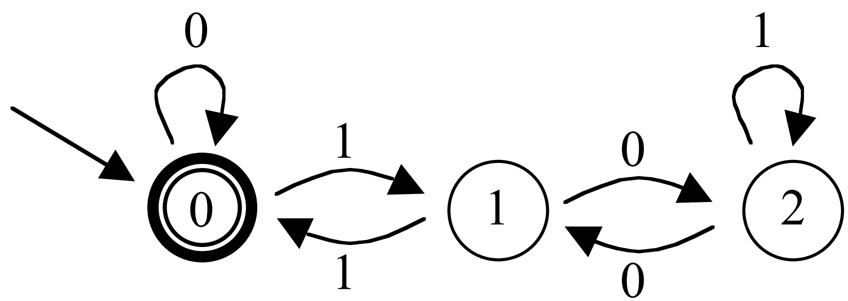
- 3.1 Closed Under Complement
- 3.2 Closed Under Intersection
- 3.3 Closed Under Union
- 3.4 DFA Proofs Using Induction
- 3.5 A Mystery DFA

Mystery DFA



- What language does this DFA accept?
- We can experiment:
  - It rejects 1, 10, 100, 101, 111, and 1000…
  - It accepts 0, 11, 110, and 1001…
- But even if that gives you an idea about the language it accepts, how can we prove it?

Transition Function Lemma



Lemma 3.5.1: for all states $i \in Q$ and symbols $c \in \Sigma$,

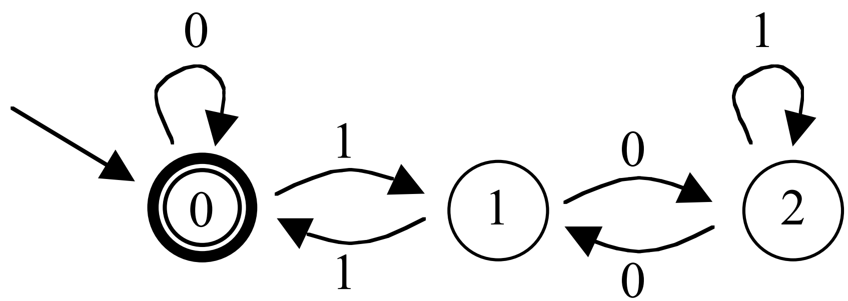
$$\delta(i, c) = (2i+c) \bmod 3$$

- Proof is by enumeration:
  - $\delta(0, 0) = 0 = (2\times0+0) \bmod 3$
  - $\delta(0, 1) = 1 = (2\times0+1) \bmod 3$
  - $\delta(1, 0) = 2 = (2\times1+0) \bmod 3$
  - $\delta(1, 1) = 0 = (2\times1+1) \bmod 3$
  - $\delta(2, 0) = 1 = (2\times2+0) \bmod 3$
  - $\delta(2, 1) = 2 = (2\times2+1) \bmod 3$

Function val For Binary Strings

- Define val(x) to be the number for which x is an unsigned binary representation
- For completeness, define val(ε) = 0
- For example:
  - val(11) = 3
  - val(111) = 7
  - val(000) = val(0) = val(ε) = 0
- Using val we can say something concise about $\delta^*(0,x)$ for any x…

Off To A Bad Start...



Lemma 3.5.2, weak: $L(M) = \{x \mid \text{val}(x) \bmod 3 = 0\}$

- This is what we ultimately want to prove: *M* defines the language of binary representations of numbers that are divisible by 3
- But proving this by induction runs into a problem

Lemma 3.5.2, weak: $L(M) = \{x \mid \text{val}(x) \bmod 3 = 0\}$

Proof: by induction on $|x|$.

Base case: when $|x| = 0$, we have:

$\delta^*(0, x)$
$= \delta^*(0, \text{e})$ (since $|x| = 0$)
$= 0$ (by definition of δ^*)

so in this case $x \in L(M)$ and $\text{val}(x) \bmod 3 = 0$.

Inductive case: when $|x| > 0$, we have:

$\delta^*(0, x)$
$= \delta^*(0, yc)$ (for some symbol c and string y)
$= \delta(\delta^*(0, y), c)$ (by definition of δ^*)
$= ???$

The proof gets stuck here: our inductive hypothesis is not strong enough to tell us what $\delta^*(0, y)$ is, when $\text{val}(y)$ is not divisible by 3

Proving Something Stronger

- We tried and failed to prove

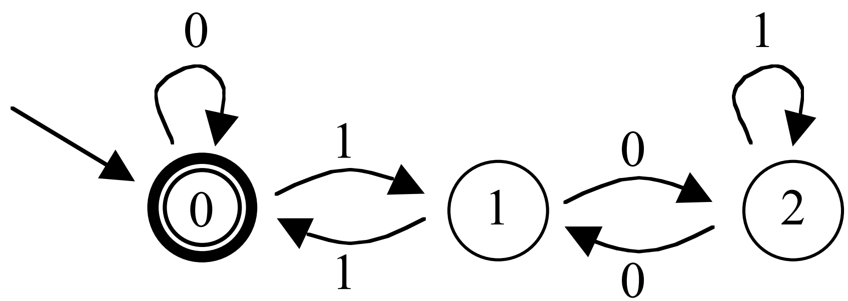
  $L(M) = \{x \mid \text{val}(x) \bmod 3 = 0\}$

- To make progress, we need to prove a broader claim:

  $\delta^*(0,x) = \text{val}(x) \bmod 3$

- That implies our original lemma, but gives us more to work with
- A common trick for inductive proofs
- Proving a strong claim can be easier than proving a weak one, because it gives you a more powerful inductive hypothesis

The Mod 3 Lemma



Lemma 3.5.2, strong: $\delta^*(0,x) = \text{val}(x) \bmod 3$

- This follows from Lemma 3.5.1 by induction
- Proof is by induction on the length of the string x

Lemma 3.5.2, strong: $\delta^*(0,x) = \mathrm{val}(x) \bmod 3$

Proof: by induction on $|x|$.

Base case: when $|x| = 0$, we have:

$\delta^*(0, x)$

$= \delta^*(0, \varepsilon)$ (since $|x| = 0$)

$= 0$ (by definition of δ^*)

$= \mathrm{val}(x) \bmod 3$ (since $\mathrm{val}(x) \bmod 3 = \mathrm{val}(\varepsilon) \bmod 3 = 0$)

Inductive case: when $|x| > 0$, we have:

$\delta^*(0, x)$

$= \delta^*(0, yc)$ (for some symbol c and string y)

$= \delta(\delta^*(0, y), c)$ (by definition of δ^*)

$= \delta(\mathrm{val}(y) \bmod 3, c)$ (using the inductive hypothesis)

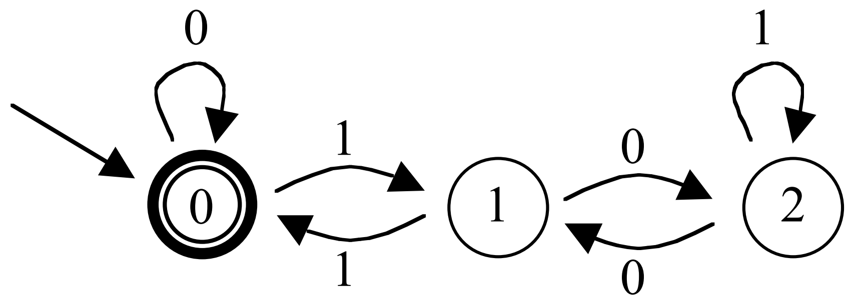
$= (2(\mathrm{val}(y) \bmod 3)+c) \bmod 3$ (by Lemma 3.5.1)

$= 2(\mathrm{val}(y)+c) \bmod 3$ (using modulo arithmetic)

$= \mathrm{val}(yc) \bmod 3$ (using binary arithmetic: $\mathrm{val}(yc) = 2(\mathrm{val}(y))+c$)

$= \mathrm{val}(x) \bmod 3$ (since $x = yc$)

Mystery DFA's Language



- Lemma 3.5.2, strong: $\delta^*(0, x) = \text{val}(x) \bmod 3$
- That is: the DFA ends in state *i* when the binary value of the input string, divided by 3, has remainder *i*
- So *L*(*M*) = the set of strings that are binary representations of numbers divisible by 3
- Those examples again:
  - It rejects 1, 10, 100, 101, 111, and 1000…
  - It accepts 0, 11, 110, and 1001…