Section 09: Solutions

1. Regular Expressions

Note: this question was also on the Section 8 supplemental handout.

(a) Write a regular expression that matches base 10 numbers (e.g., there should be no leading zeroes).

Solution:

$$0 \cup ((1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)^*)$$

Explanation: Either the number is exactly 0, or it starts with a non-zero digit (1–9) followed by any number of digits (0–9). This prevents leading zeroes.

(b) Write a regular expression that matches all base-3 numbers that are divisible by 3 without leading zeroes.

Solution:

$$0 \cup ((1 \cup 2)(0 \cup 1 \cup 2)^*0)$$

Explanation: In base-3, a number is divisible by 3 exactly when its last digit is 0 or is 0. The regex matches either 0, or a string starting with 1 or 2, followed by any base-3 digits, and ending in 0 to prevent leading zeroes.

(c) Write a regular expression that matches all binary strings that contain the substring "111", but not the substring "000".

Solution:

$$(01 \cup 001 \cup 1^*)^*(0 \cup 00 \cup \varepsilon)111(01 \cup 001 \cup 1^*)^*(0 \cup 00 \cup \varepsilon)$$

Explanation: All allowed prefixes and suffixes must avoid producing 000, so they can contain at most two consecutive zeroes. The middle 111 enforces that the string contains the substring 111. The parts around it enumerate all zero-runs of length at most 2 without introducing 000.

(d) Write a regular expression that matches all binary strings that do not have any consecutive 0's or 1's.

Solution:

$$((01)^*(0 \cup \varepsilon)) \cup ((10)^*(1 \cup \varepsilon))$$

Explanation: These strings must alternate 0s and 1s to avoid consecutive. The regex captures all alternating sequences that start with 0 (like 0101…) and all alternating sequences that start with 1 (like 1010…), allowing a final optional symbol.

(e) Write a regular expression that matches all binary strings of the form $1^k y$, where $k \geq 1$ and $y \in \{0,1\}^*$ has at least k 1's.

Solution:

$$1(0 \cup 1)^* 1(0 \cup 1)^*$$

Explanation: While it may seem like we need to keep track of how many 1's there are, it turns out that we don't. Convince yourself that strings in the language are exactly those of the form $1x$, where x is any binary string with at least one 1. Hence, x is matched by the regular expression $(0 \cup 1)^* 1 (0 \cup 1)^*$.

2. CFGs

Write a context-free grammar to match each of these languages.

(a) All binary strings that start with 11.
Solution:

$$\mathbf{S} \to 11\mathbf{T}$$
$$\mathbf{T} \to 1\mathbf{T} \mid 0\mathbf{T} \mid \varepsilon$$

Explanation: The start symbol forces the string to begin with 11. After that, **T** generates any (possibly empty) sequence of 0s and 1s using the usual "any binary string" grammar.

(b) All binary strings that contain at most one 1.
Solution:

$$\mathbf{S} \to \mathbf{ABA}$$
$$\mathbf{A} \to 0\mathbf{A} \mid \varepsilon$$
$$\mathbf{B} \to 1 \mid \varepsilon$$

Explanation: The string is divided into three parts: zeros before the (possible) 1, the middle symbol **B** which is either a single 1 or empty, and zeros after it. **A** generates any number of 0s. **B** ensures we have *either* one 1 *or* zero 1s.

(c) All strings over 0, 1, 2 with the same number of 1s and 0s and exactly one 2.
Solution:

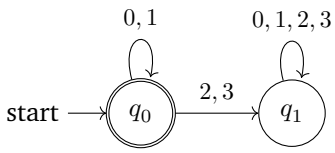
$$\mathbf{S} \to 2\mathbf{T} \mid \mathbf{T}2 \mid \mathbf{ST} \mid \mathbf{TS} \mid 0\mathbf{S}1 \mid 1\mathbf{S}0$$
$$\mathbf{T} \to \mathbf{TT} \mid 0\mathbf{T}1 \mid 1\mathbf{T}0 \mid \varepsilon$$

Explanation: **S** matches a 2 at the beginning or end. The rest of the string must then match **T** since it cannot have another 2. If neither the first nor last character is a 2, then it falls into the usual cases of matching 0s and 1s, so we can mostly use the same rules as **T**. The main change is that **SS** becomes **ST** | **TS** to ensure that exactly one of the two parts contains a 2. The other change is that there is no ϵ since a 2 must appear somewhere.

3. DFAs, Stage 1

Construct DFAs to recognize each of the following languages. Let $\Sigma = \{0, 1, 2, 3\}$.

(a) All binary strings.
Solution:

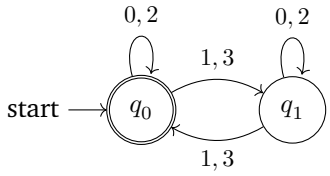


q_0: binary strings

q_1: strings that contain a character which is not 0 or 1.

Explanation: The DFA stays in q_0 as long as the input is only 0s and 1s. The moment a 2 or 3 appears, it moves to q_1, which traps all non-binary strings. Only q_0 is accepting, so the machine accepts exactly the binary strings.

(b) All strings whose digits sum to an even number.
Solution:

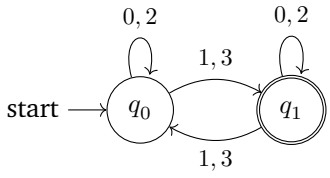


q_0: sum of digits seen so far is even

q_1: sum of digits seen so far is odd

Explanation: Digits 0 and 2 keep the parity the same, so each state loops on them. Digits 1 and 3 flip the parity, so they send you to the opposite state. Since q_0 is accepting, this DFA accepts exactly the strings with an even digit-sum.

(c) All strings whose digits sum to an odd number.
Solution:



q_0: sum of digits seen so far is even

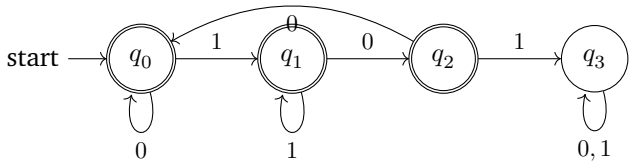
q_1: sum of digits seen so far is odd

Explanation: This DFA is the same as the previous one, but now q_1 is the accepting state. Thus, it accepts exactly the strings whose digit-sum is odd.

4. DFAs, Stage 2

Construct DFAs to recognize each of the following languages. Let $\Sigma = \{0, 1\}$.

(a) All strings which do not contain the substring 101.
Solution:



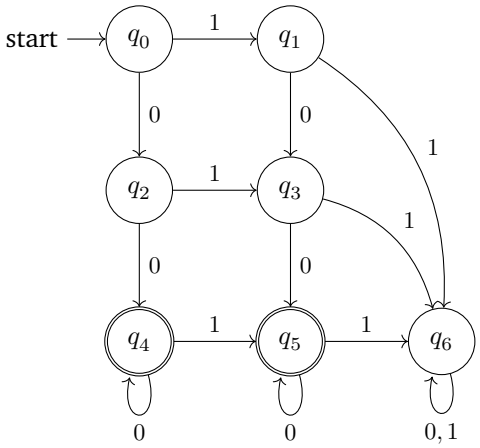
q_3: string that contain 101.

q_2: strings that don't contain 101 and end in 10.

q_1: strings that don't contain 101 and end in 1.

q_0: ε, 0, strings that don't contain 101 and end in 00.

(b) All strings containing at least two 0's and at most one 1.
Solution:



q_1: 0 zeros, 1 one

q_2: 1 zero, 0 ones

q_3: 1 zero, 1 one

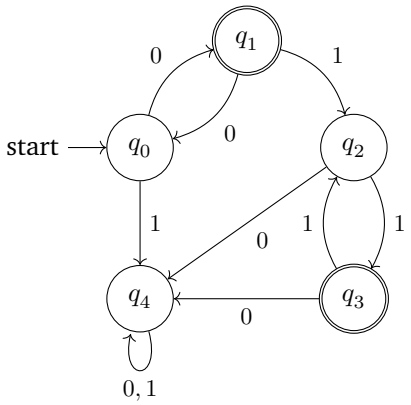
q_4: 2+ zeros, 0 ones (accept!)

q_5: 2+ zeros, 1 one (accept!)

q_6: more than one 1, so reject forever (trap, it here)

Explanation: The DFA counts zeros until it reaches at least two, and separately tracks whether a 1 has already appeared. Once two 0s have been seen, the string is acceptable as long as no second 1 occurs. State q_6 is the trap for strings with two or more 1s.

(c) All strings containing an even number of 1's and an odd number of 0's and not containing the substring 10.
Solution:



q_0: even 1s, even 0s (initial)

q_1: even 1s, odd 0s (accept!)

q_2: odd 1s, odd 0s

q_3: even 1s, odd 0s (accept!)

q_4: trap state - substring 10 occurred or we have an even number of 0's and a 1 already occurred

Explanation: The machine tracks parity of 0s and 1s using four states. The extra constraint "no 10" forces all transitions that would introduce a 10-pattern into a trap state q_4. The accepting states are exactly those with even 1s and odd 0s.

5. All The Models!

Construct a valid regular expression, CFG, and DFA for the following languages.

(a) All strings whose base-6 representation is divisible by 3 (leading zeros are ok). Let $\Sigma = \{0, 1, 2, 3, 4, 5\}$.

Solution:

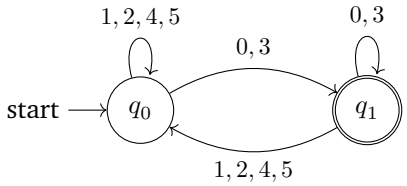
Regular Expression:

$$(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5)^*(0 \cup 3)$$

Explanation: A base-6 number is divisible by 3 exactly when its last digit is 0 or 3. Since leading zeros are allowed, any sequence of digits may come before that. The regex captures "any digits, then a final 0 or 3."
CFG:

$$\mathbf{S} \to \mathbf{T}0 \mid \mathbf{T}3$$
$$\mathbf{T} \to 0\mathbf{T} \mid 1\mathbf{T} \mid 2\mathbf{T} \mid 3\mathbf{T} \mid 4\mathbf{T} \mid 5\mathbf{T} \mid \varepsilon$$

Explanation: T generates any sequence of base-6 digits (including empty). Then S appends either 0 or 3 at the end, ensuring the whole string ends with a digit that makes it divisible by 3.
DFA:



The DFA tracks divisibility mod 3 using two states:

q1: ends with a 0 or 3

q0: ends with any other digit

(b) All binary strings of 0s capped by a 1 on either side.

Solution:

We are working with binary strings, therefore $\Sigma = \{0, 1\}$.

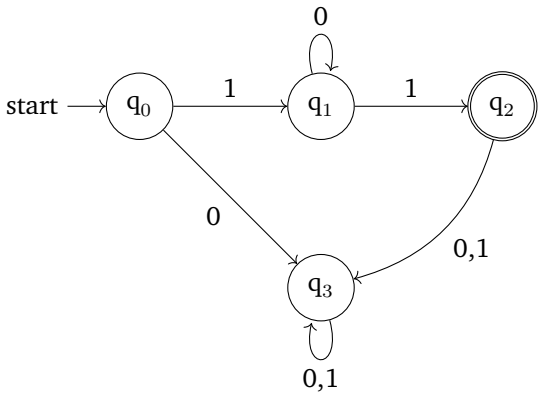
Regular Expression:

$$1(0)^*1$$

CFG:

$$\mathbf{S} \rightarrow 1\mathbf{T}1$$
$$\mathbf{T} \rightarrow 0\mathbf{T} \mid \varepsilon$$

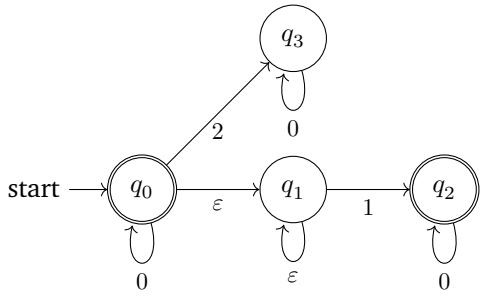
DFA:



6. NFAs, Stage 1

Note: This content will be covered in lecture on Nov 21st and 24th. Here is a video walkthrough posted in place of the section that would be on Thanksgiving week! Check the course calendar for slides.

(a) What language does the following NFA accept?

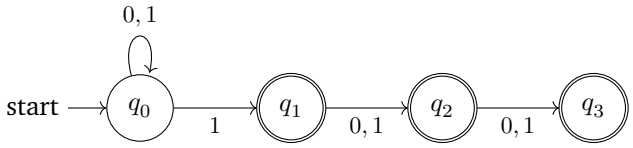


Solution:

All strings of only 0's and 1's not containing more than one 1.

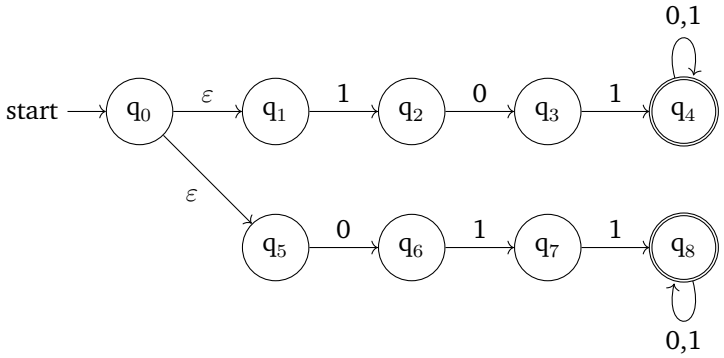
(b) Create an NFA for the language "all binary strings that have a 1 as one of the last three digits".
Solution:

The following is one such NFA:



Explanation: We loop freely until we see the potential last 1, then the NFA tracks the next 0–2 characters to ensure that at least one of the final three characters is 1.

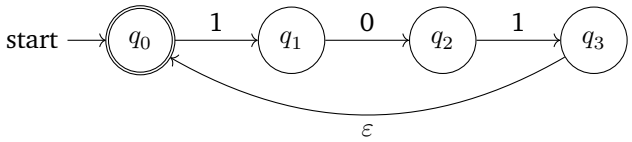
(c) Create an NFA for the language "all binary strings that start with 101 or 011".
Solution:



Explanation: The two ε-branches let the NFA choose to match either prefix (101 or 011), and once one of those is matched, the accepting state loops to allow any remaining suffix. These ε branches are very helpful when we have options or an OR in the language!

(d) Create an NFA for the language "all strings composed of 0 or more occurrences of 101 only".

Solution:



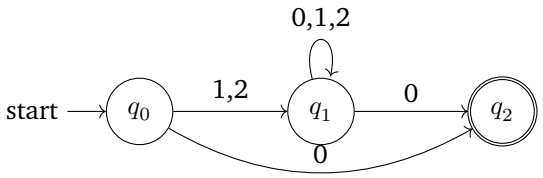
Explanation: This NFA forces every block to be exactly the substring 101, and an ε-transition from the end returns you to the start so the pattern can repeat zero or more times. Note that if that if we wanted at least one occurrence of 101, we could just make q_3 a final state, and not q_0. ε transitions are also great for a kind of "loop" situation like what we have here!

7. NFAs, Stage 2

Note: This content will be covered in lecture on Nov 21st and 24th. Here is a video walkthrough posted in place of the section that would be on Thanksgiving week! Check the course calendar for slides.

Hint: Note that we already found the regex for these in Question 1. Referencing your answers for that can be helpful!

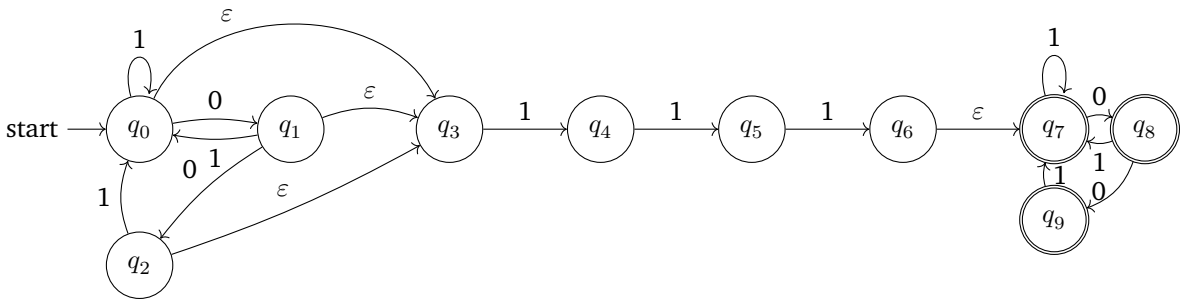
(a) Create an NFA for the language "all base 3 numbers without leading zeros that are divisible by 3". **Solution:**



Explanation: From the start state, this NFA branches on the first character. If it's a zero, then the entire string must only be zero because we don't want any leading zeroes, and so we can jump straight to our final state. Otherwise, our string is some nonzero number. However, to ensure that the number is divisible by 3, we enforce that it must end in 0 by only allowing zero to transition to the final state.

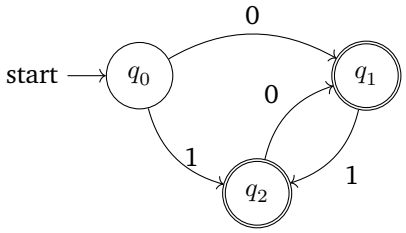
(b) Create an NFA for the language "all binary strings that contain the substring '111', but not the substring '000'".

Solution:



Explanation: Observe that strings accepted by this language take the form $x111y$ where x, y are possibly empty binary strings that do not contain '000'. Then, q_0, q_1, q_2, q_3 are states corresponding to a smaller NFA that accepts substrings that do not contain '000'. q_3, q_4, q_5, q_6 enforce the presence of 111 by making the accepting states only available via a sequence of 3 consecutive '1' transitions. Lastly, q_7, q_8, q_9 are our accepting states but also allow for characters after '111' by forming an NFA for substrings that do not contain '000'.

(c) Create an NFA for the language "all binary strings that do not have any consecutive 0's or 1's". **Solution:**



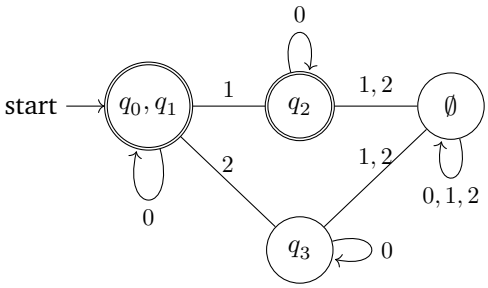
Explanation: In this NFA, q_1 represents the state at which the last character we saw was a 0. In this state, the only valid transition is one in which we see a 1 next. Similarly, q_2 represents the state at which the last character we saw was a 1. In q_0 then, we can route the string to correct state based on the first character that we see.

8. DFAs & Minimization

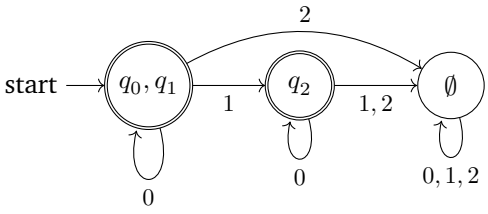
Note: We will not test you on minimization, although you may optionally read the extra slides and do this problem for fun

(a) Convert the NFA from 6a to a DFA, then minimize it.

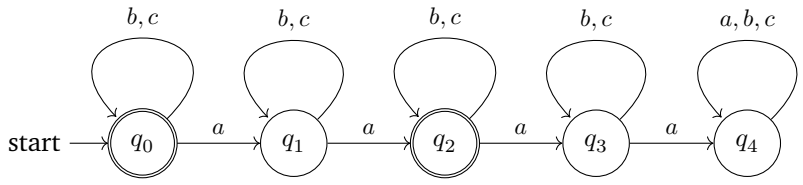
Solution:



Here is the minimized form:



(b) Minimize the following DFA:



Solution:

Step 1: q_0, q_2 are final states and the rest are not final. So, we start with the initial partition with the following groups: group 1 is $\{q_0, q_2\}$ and group 2 is $\{q_1, q_3, q_4\}$.

Step 2: q_1 is sending a to group 1 while q_3, q_4 are sending a to group 2. So, we divide group 2. We get the following groups: group 1 is $\{q_0, q_2\}$, group 3 is $\{q_1\}$ and group 4 is $\{q_3, q_4\}$.

Step 3: q_0 is sending a to group 3 and q_2 is sending a to group 4. So, we divide group 1. We will have the following groups: group 3 is $\{q_1\}$, group 4 is $\{q_3, q_4\}$, group 5 is $\{q_0\}$ and group 6 is $\{q_2\}$.

The minimized DFA is the following: