CS 250 Course Study Guide

Sarmistha Sarna Gomasta, Kyla Levin, Alexander Yeung

Fall 2023

Please Fill Out Our Feedback Form: https://forms.gle/1ieupiHHHSuTEjXe9

Notes on Intended Use:

We hope you find this study guide useful, however, we would like to make a few notes on its intended use.

1. This study guide is not intended to be exhaustive. Its intended purpose is for you to ascertain whether or not you have a basic understanding of each topic. *Please use other resources (lecture notes, past exams, etc.) to study for the final.*
2. This resource was made by graduate students as a learning supplement and may not perfectly reflect exam-level difficulty.
3. We advise you to use this to first to gauge your basic understanding of a topic before filling in any gaps in your knowledge with other materials to improve practice exam performance.

Contents

Chapter 1: Sets and Predicate Logic

Q1.1

Solve the following syllogism. Report the answer as a logical implication, as well as a naturally-spoken English sentence.

- The only foods that my doctor recommends are ones that aren't very sweet.
- Nothing that agrees with me is good for dinner.
- Cake is always very sweet.
- My doctor recommends all foods that are good for dinner.

Q1.1 Answer

We will use the following variables: $D =$ doctor recommends it, $S =$ sweet, $A =$ agrees with me, $G =$ good for dinner

- This can be read as "If my doctor recommends it, then it's not very sweet." $D \rightarrow \neg S$
- This can be read as "There is nothing such that if it agrees with me, then it's good for dinner." $A \rightarrow G$
- This can be read as "If it's cake, then it's very sweet." $C \rightarrow S$
- This can be read as "If it's good for dinner, then my doctor recommends it." $G \rightarrow D$

$C \rightarrow S \rightarrow \neg D \rightarrow \neg G$
Transitively, $C \rightarrow \neg G$
In English, "If it's cake, then it's not good for dinner." Or, more naturally, "Cake is not good for dinner."

Q1.2

a. What values of a and b make the following statement true? Prove using a truth table.

$$(a \lor \neg b) \land (b \land \neg a)$$

b. What values of a, b, and c make the following statement true? Prove without using a truth table.

$$(a \lor \neg b) \land (b \land \neg a) \rightarrow (\neg c \land (a \lor b))$$

Q1.2 Answer

a. What values of a and b make the following statement true? Prove using a truth table.

$$(a \vee \neg b) \wedge (b \wedge \neg a)$$

a	b	$\neg a$	$\neg b$	$(a \vee \neg b)$	$(b \wedge \neg a)$	$(a \vee \neg b) \wedge (b \wedge \neg a)$
T	T	F	F	T	F	F
T	F	F	T	T	F	F
F	T	T	F	F	T	F
F	F	T	T	T	F	F

To fill in the truth table, begin with the primitive values a and b in the leftmost columns. From there, use those values to fill in the successive $\neg a$ and $\neg b$ values in the next two columns. Continue rightways, adding columns with increasingly complex expressions using the truth values from the simpler columns until you have constructed the final expression.

As a reminder, for the binary operator "OR" ($\vee$), *at least one* of the input needs to be true. For the binary operator "AND" ($\wedge$), *both* of the input need to be true. And the unary operator "NOT" ($\neg$) will flip the truth value of its input.

As seen in the final column of the truth table, there is no combination of a and b values that make $(a \vee \neg b) \wedge (b \wedge \neg a)$ true.

b. What values of a, b, and c make the following statement true? Prove without using a truth table.

$$(a \vee \neg b) \wedge (b \wedge \neg a) \rightarrow (\neg c \wedge (a \vee b))$$

All possible values of a, b, and c will make this statement true. We know from part (a) that the antecedent of the implication step is always false. There are no a and b values that make it true. Therefore, there is no value that the consequent could have to make the whole implication evaluate to true. If the consequent is true, then $F \rightarrow T$ evaluates to true. If the consequent is false, then $F \rightarrow F$ still evaluates to true.

Q1.3

Complete a deductive sequence proof based on the following propositions:

1. $\neg p \wedge q$
2. $r \rightarrow p$
3. $\neg r \rightarrow s$
4. $s \rightarrow t$

Your answer should either be a conjunction of all of the variables p,q,r,s,t which enumerates their truth values, or you may explicitly state the truth value of each variable.

Q1.3 Answer

1. Assume $\neg p$
2. q (Premise 1, right separation)
3. $\neg p \to \neg r$ (Contrapositive of Premise 2)
4. $\neg r$ (Modus Ponens of Line 3)
5. s (Modus Ponens of Premise 3)
6. t (Modus Ponens of Premise 4)

Then our final solution is $\neg p \land q \land \neg r \land s \land t$

Q1.4

Prove by contradiction that the sum of a rational number and an irrational number is irrational. Recall that a rational number is one which can be expressed as the quotient of two integers. An irrational number is one which cannot be represented as the quotient of two integers. You may assume that the operations addition, subtraction, and multiplication on rationals are well-defined. In other words, the sum, difference, or product of two rationals is always rational.

Q1.4 Answer

Assume to the contrary, $\exists s$ such that $s = r + i$ where $r \in \mathbb{Q}$, the rational numbers, and $i \notin \mathbb{Q}$, or i is irrational. Well, then $i = s - r$. But we know that the difference between two rational numbers is always rational. Then i must be a rational number. But this is a contradiction because we assumed i to be irrational!

Q1.5

Prove that if n is divisible by 6, then $n + 10$ is not divisible by 6.

Q1.5 Answer

First, we must assume the antecedent to prove the consequent, so assume that n is divisible by 6.
Now, for the sake of contradiction, assume that $n + 10$ *is* divisible by 6. If this is the case, then both $6|n$ and $6|(n + 10)$ are true, therefore $6|(n + 10 - n)$
If you require proof of this property, it can be observed below:

> Claim: If $a|b$ and $a|c$, then $a|(b - c)$
> If $a|b$, then by the definition of divisibility, $\exists n \in \mathbb{Z} : an = b$
> If $a|c$, then by the definition of divisibility, $\exists m \in \mathbb{Z} : am = c$
> Subtracting one equation from the other: $an - am = a(n - m) = b - c$
> Since $n - m \in \mathbb{Z}$, then by definition of divisibility, $a|(b - c)$

Back to the original proof: If $6|(n + 10 - n)$, then this simplifies to $6|10$, however, this is not true. 6 does not divide 10, therefore we have a contradiction.

Additional Resources for Sets and Predicate Logic

1. Translating Statements to Propositional Logic
2. Excellent Discrete Math Playlist
3. Playlist Containing Many Worked Out Deductive Proofs
4. MIT Video of Proof by Contradiction
5. Proof by Contrapositive Intro
6. Rules of Inference Video

Chapter 2: Quantifiers and Relations

Q2.1

Consider the set $A = \mathbb{N}$
Identify whether the following relations are reflexive, antireflexive, symmetric, antisymmetric, or transitive.

- x relates to y if and only if $x = 2y$
- x relates to y if and only if $x\%2 = y\%2$

Q2.1 Answer

- x relates to y if and only if $x = 2y$
  This is not reflexive, because for most numbers, $x \neq 2x$. However, it is also not antireflexive, since $x = 2x$ is true for $x = 0$. It is not symmetric, since if $x = 2y$, then $y \neq 2x$. The only case when this is true is for $x = y = 0$, when x and y are the same. Therefore, this relation is antisymmetric. It is not transitive, since if $x = 2y$ and $y = 2x$, then $x = 4z$, which is not always equal to $2z$.

- x relates to y if and only if $x\%2 = y\%2$
  This is reflexive, since $x\%2 = x\%2$ for all x. Because it is reflexive, it cannot be antireflexive. It is symmetric, since if $x\%2 = y\%2$, then $y\%2 = x\%2$. It is not antisymmetric, since this can be true of different x and y values. It is transitive, since if $x\%2 = y\%2$ and $y\%2 = z\%2$, then $x\%2 = z\%2$.

Q2.2

Let A, B, C, D be nonempty sets.

a) Prove that $A \times B \subseteq C \times D$ if and only if $A \subseteq C$ and $B \subseteq D$.

b) What happens to the result in part (a) if any of the sets A, B, C, D is empty?

Q2.2 Answer

(a) Assume that $A \times B \subseteq C \times D$ and let $a \in A$ and $b \in B$. Then $(a, b) \in A \times B$, and since $A \times B \subseteq C \times D$ we have $(a, b) \in C \times D$. But $(a, b) \in C \times D$ implies $a \in C$ and $b \in D$. Hence, $a \in A$ implies $a \in C$, so $A \subseteq C$, and $b \in B$ implies $b \in D$, so $B \subseteq D$. Conversely, suppose that $A \subseteq C$ and $B \subseteq D$, and that $(x, y) \in A \times B$. Then $x \in A$ and $y \in B$ implies $x \in C$ (since $A \subseteq C$) and $y \in D$ (since $B \subseteq D$) implies $(x, y) \in C \times D$. Consequently, $A \times B \subseteq C \times D$.

(b) If any of the sets A, B, C, D is empty we still find that

$$\{(A \subseteq C) \wedge (B \subseteq D)\} \Rightarrow \{A \times B \subseteq C \times D\}.$$

However, the converse need not hold. For example, let $A = \emptyset$, $B = \{1, 2\}$, $C = \{1, 2\}$ and $D = \{1\}$. Then $A \times B = \emptyset$ — if not, there exists an ordered pair (x, y) in $A \times B$, and this means that the empty set A contains an element x. And so $A \times B = \emptyset \subseteq C \times D$ — but $B = \{1, 2\} \not\subseteq \{1\} = D$.

Q2.3

Determine whether or not each of the following relations is a function. If a relation is a function, find its range.

a) $\{(x, y) \mid x, y \in \mathbb{Z}, y = x^2 + 7\}$, a relation from $\mathbb{Z}$ to $\mathbb{Z}$

b) $\{(x, y) \mid x, y \in \mathbb{R}, y^2 = x\}$, a relation from $\mathbb{R}$ to $\mathbb{R}$

c) $\{(x, y) \mid x, y \in \mathbb{R}, y = 3x + 1\}$, a relation from $\mathbb{R}$ to $\mathbb{R}$

d) $\{(x, y) \mid x, y \in \mathbb{Q}, x^2 + y^2 = 1\}$, a relation from $\mathbb{Q}$ to $\mathbb{Q}$

e) $\mathcal{R}$ is a relation from A to B where $|A| = 5$, $|B| = 6$, and $|\mathcal{R}| = 6$.

Q2.3 Answer

(a) Function: Range = $\{7, 8, 11, 16, 23, \ldots\}$

(b) Relation, not a function. For example, both (4,2) and (4, -2) are in the relation.

(c) Function: Range = the set of all real numbers.

(d) Relation, not a function. Both (0,1) and (0, -1) are in the relation.

(e) Since $|\mathcal{R}| > 5$, $\mathcal{R}$ cannot be a function.

Q2.4

Determine whether each of the following statements is true or false. If the statement is false, provide a counterexample.

a) $|a| = [-a]$ for all $a \in \mathbb{Z}$.

b) $|a| = [a]$ for all $a \in \mathbb{R}$.

c) $|a| = [a] - 1$ for all $a \in \mathbb{R} \setminus \mathbb{Z}$.

d) $[a] = [-a]$ for all $a \in \mathbb{R}$.

Q2.4 Answer

(a) True

(b) False: Let $a = 1.5$. Then $\lfloor 1.5 \rfloor = 1 \neq 2 = \lfloor a \rfloor$

(c) True

(d) False: Let $a = 1.5$. Then $\lfloor -a \rfloor = -2 \neq -1 = \lfloor -a \rfloor$.

Q2.5

Suppose you have sets $A = \{3, 5, 7, 11, 13, 17\}$ and $B = \{0, 1, 2, 3, 5, 8, 13\}$ and some function $f : A \to B$.

a. How many one-to-one functions f are possible? Give one example.

b. How many onto functions f are possible? Give one example.

Q2.5 Answer

a. A "one-to-one" function is one where each input leads to a distinct output. So any example where no two elements of A lead to the same element of B is a valid answer. Think of this as selecting an output for each input. For the first input, $3 \in A$, there are 7 possible outputs in B. Since the outputs must be distinct in a one-to-one function, there are 6 possible outputs for the next element of A, 5 for the next, and so on and so forth. So for 6 possible inputs, there are $7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 = 5,040$ possible one-to-one functions. For the simplest example, $f = \{(3,0),(5,1),(7,2),(11,3),(13,5),(17,8)\}$

b. An "onto" function is one where every possible output has a corresponding input. This is, however, not possible since there are more possible outputs than there are possible inputs. To achieve every possible output, at least two inputs would have to lead to the same output. This can also not be the case, since this would not be a function. So no onto functions f are possible.

Additional Resources for Quantifiers and Relations

1. Set Theory Playlist
2. Equivalence Relations and Equivalence Classes Explained
3. Bijective, Injective, Surjective Explained
4. Existential and Universal Quantifiers Explained
5. Negating Quantifiers
6. Negating Multiple Quantifiers
7. Reflexive, Symmetric, Transitive

Chapter 3: Number Theory

Q3.1

Prove that for some relatively prime a and n, multiplying all numbers in the set $\{1, 2, ..., n-2, n-1\}$ by a will output a permutation of $\{1, 2, ..., n-2, n-1\}$.
Hint: There are two parts to this proof—existence and uniqueness. To show that multiplication by a permutes the elements of the set, you must show that the multiplication always produces an element in the set (existence) and you must show that that element will never be repeatedly produced via multiplication by a (uniqueness). Consider how these relate to modular arithmetic and modular inverses.

Q3.1 Answer

As stated in the hint, we must show both the existence and uniqueness of the elements from the set when multiplied by a.
First, to prove existence, we will prove that for some relatively prime a and n, and for some elements $b, c \in \{1, 2, ..., n-2, n-1\}$, $ab = c \pmod{n}$, unless $b = 0$. This is equivalent to saying that "If $ab = 0 \pmod{n}$, then $b = 0 \pmod{n}$". Since a and n are relatively prime, a has an inverse in mod n. Multiply both sides by $a^{-1} \pmod{n}$ to get $aba^{-1} = 0(a^{-1}) \pmod{n}$. aa^{-1} will cancel out to 1, showing that $b = 0 \pmod{n}$.
Next, to prove uniqueness, we will show that for some relatively prime a and n, if $ab = ac \pmod{n}$, then $b = c$. Once again, because a and n are relatively prime, we know a must have some inverse $a^{-1} \pmod{n}$. Multiplying both sides by a^{-1} gives $aba^{-1} = aca^{-1} \pmod{n}$, leaving $b = c \pmod{n}$.
Putting the two pieces together, we know that multiplying any element of $\{1, 2, ..., n-2, n-1\}$ by some constant a, which is relatively prime to n, will never result in 0 and therefore will always produce some number in the range of 1 to $n-1$. Furthermore, no multiplication will result in the same number. Therefore, the output will be a permutation of the original set $\{1, 2, ..., n-2, n-1\}$.

Q3.2

Let $a = 473$ and $b = 47$. Run Euclid's Algorithm on a and b to find their GCD. What can you conclude about a and b from your run of Euclid's Algorithm?

Q3.2 Answer

$$\begin{aligned}
473 &= 10 \times 47 + 3 \\
47 &= 15 \times 3 + 2 \\
3 &= 1 \times 2 + 1 \\
2 &= 2 \times 1 + 0
\end{aligned}$$

The GCD of 473 and 47 is 1 since the algorithm ends with a remainder of 1. Since their GCD is 1, the two numbers are relatively prime.

Q3.3

Use the Chinese Remainder Theorem to find the smallest positive integer x that satisfies the following three simultaneous congruences:

$$\begin{aligned} x &\equiv 2 \pmod 3 \\ x &\equiv 3 \pmod 5 \\ x &\equiv 2 \pmod 7 \end{aligned}$$

Q3.3 Answer

$$\begin{aligned}
n_1 &= 3\\
n_2 &= 5\\
n_3 &= 7\\
N &= n_1 n_2 n_3 = 105
\end{aligned}$$

Calculate the modular multiplicative inverses:

$$\begin{aligned}
y_1 &\equiv 35^{-1} \pmod{3} \quad (\text{Since } N_1 = N/n_1 = 35)\\
y_2 &\equiv 21^{-1} \pmod{5} \quad (\text{Since } N_2 = N/n_2 = 21)\\
y_3 &\equiv 15^{-1} \pmod{7} \quad (\text{Since } N_3 = N/n_3 = 15)
\end{aligned}$$

Compute t_i terms:

$$\begin{aligned}
t_1 &= 2 \cdot 35 \cdot y_1 \pmod{105}\\
t_2 &= 3 \cdot 21 \cdot y_2 \pmod{105}\\
t_3 &= 2 \cdot 15 \cdot y_3 \pmod{105}
\end{aligned}$$

Sum and reduce mod N to get:

$$\begin{aligned}
x &= t_1 + t_2 + t_3 \pmod{105}\\
&= 14 + 63 + 28 \pmod{105}\\
&= 105 \pmod{105}
\end{aligned}$$

However, since we want the smallest positive integer, $x = 105 \pmod{105}$ gives $x = 0$. Therefore, we need to subtract N from 105 to find the smallest positive solution. So the corrected smallest positive integer solution is $x = 105 - 105 = 0$. This seems incorrect as 0 is not positive, we need to reevaluate the t_i terms to find the correct solution.

Additional Resources for Number Theory

1. Euclid's Algorithm
2. Chinese Remainder Theorem

Chapter 4: Induction and Recursion

Q4.1

Explain the error in the following induction proof:
Claim: All students are the same height.
Base case: Consider a set of students of size 1. Any student is the same height as themselves, and so all students in the set are the same height.
Inductive hypothesis: All students in a set of size n have the same height.
Inductive step: Now suppose we have a set of $n+1$ students, labeled s_1 through s_{n+1}: $\{s_1, s_2, s_3, ..., s_{n-1}, s_n, s_{n+1}\}$.
Consider the entire set of $n+1$ students as two individual sets of n students each.
The first set contains students s_1 through s_n: $\{s_1, s_2, s_3, ..., s_{n-1}, s_n\}$
The second set contains students s_2 through s_{n+1}: $\{s_2, s_3, ..., s_{n-1}, s_n, s_{n+1}\}$
Both sets contain n students, and so by the inductive hypothesis, all students in each set must be of the same height. And if all students in each set have the same height, then all students in the full $n+1$ set must have the same height.
In conclusion, all students in any set of size $n \geq 1$ must have the same height.

Q4.1 Answer

The error in the proof can be attributed to the overlap between the two sets in the inductive step. The base case is easy to observe for some set $\{s_1\}$, but when the inductive step is applied to ”step” from $n = 1$ to $n = 2$, the two sets described in the inductive step would be $\{s_1\}$ and $\{s_2\}$, in which may be two students of differing heights. So while s_1 and s_2 are the same heights as themselves, this does not mean that they are the same height as one another.

Q4.2

Suppose we have a recursively-defined function (much like a Fibonacci sequence) $f_n = 5f_{n-1} - 6f_{n-2}$ where $f_0 = 2$ and $f_1 = 5$.
Prove inductively that $f_n = 2^n + 3^n$ for all $n \geq 0$.

Q4.2 Answer

Claim: $f_n = 2^n + 3^n$ for all $n \geq 0$
Base case: For $n = 0$: $f_0 = 2$, and $2^0 + 3^0 = 1 + 1 = 2$. For $n = 1$: $f_1 = 5$ and $2^1 + 3^1 = 2 + 3 = 5$.
Inductive hypothesis: Assume that $f_n = 2^n + 3^n$ for all $n \geq 0$
Inductive step:

$$f_{n+1} = 5f_n - 6f_{n-1}$$

$$f_{n+1} = 5(2^n + 3^n) - 6(2^{n-1} + 3^{n-1}) \text{ By inductive hypothesis}$$

$$f_{n+1} = 5(2^n) + 5(3^n) - 3(2^n) - 2(3^n)$$

$$f_{n+1} = 2(2^n) + 3(3^n)$$

$$f_{n+1} = 2^{n+1} + 3^{n+1}$$

Q4.3

Suppose you are assembling trains. There are two types of train cars that you can use: one has a length of 1, the other has a length of 2. Trains are ordered sequences of train cars: for example, if a train has a total length of 3, then it could have been built using a length-2 car, followed by a length 1. Or a length 1 car, followed by a length 2. Or a length 1 car, followed by two more 1's.
Prove that for a train with a total length of $n \geq 1$, there are F_n number of different ways to construct the train, where F_n is the n-th Fibonacci number.
As a reminder, $F_0 = 0$ and $F_1 = 1$ in this class.

Q4.3 Answer

Base cases:
$F_1 = 1$ This is true, since if the train has a length of 1, then there's only one possible way to construct it: a single length-1 car.
$F_2 = 2$ If $n = 2$, then this could be formed by 2 or by $1 + 1$
Inductive Hypothesis: Assume that F_n is the number of ways to construct a train of length n.
Inductive Step: Now consider building a train of length $n + 1$. For the first car, you must choose between a 1-car and a 2-car. Consider each option individually:
Case 1: You choose to start with a 1-car. The rest of the train is now of length n, and the I.H. tells us that there are F_n possible ways to construct this n-length train.
Case 2: You choose to start with a 2-car. The rest of the train is now of length $n - 1$, and the I.H. tells us that there are F_{n-1} possible ways to construct this $n - 1$-length train.
Since you have F_n possible train constructions if you start with a 1-car and F_{n-1} possible train constructions if you start with a 2-car, then you have $F_n + F_{n-1} = F_{n+1}$ total possibilities for the $n + 1$-length train.

Q4.4

Prove by induction that $n^2 - n$ is always even for all $n \in \mathbb{N}$.

Q4.4 Answer

Base Case: When $n = 0$, $(0)^2 - (0) = 0$ and 0 is even.
Induction Hypothesis: Assume that $n^2 - n$ is always even for $n \in \mathbb{N}$
Induction Step: If n is even as the I.H. states, then for some n, $\exists c \in \mathbb{Z} : n^2 - n = 2c$
Then, if we consider the $n + 1$ case:

$(n+1)^2 - (n+1) = n^2 + 2n + 1 - n - 1$ Distribute out the exponent

$(n+1)^2 - (n+1) = n^2 - n + 2n$ Remove the constants and rearrange terms

$(n+1)^2 - (n+1) = (2c) + 2n$ Replace $n^2 - k$ with $2c$, by I.H.

$(n+1)^2 - (n+1) = 2(c + n)$ Since $c + n \in \mathbb{Z}$, the $n + 1$ case is even.

Additional Resources for Induction and Recursion

1. Induction Playlist
2. Strong Induction
3. Induction Visualization
4. Recursively Defined Functions

Chapter 9: Graphs and Trees

Q9.1

How many different undirected graphs with n nodes are possible? Assume that two graphs with n nodes are "different" if they have different edges and give your answer as an expression of n.

Q9.1 Answer

First, we must find out how many possible edges can exist in an undirected graph with n nodes. This is the same as the number of edges that would be in the complete graph. There are many different ways to think about this:

- Consider the adjacency matrix of the graph. Since there are n nodes in the graph, the matrix is of size $n \times n$, and since the graph is complete, every entry in the matrix would be 1 except along the diagonal (since edges cannot be directed at themselves). So the number of edges would be the number of 1's in the matrix, divided by 2 so as not to double-count. This gives $\frac{(n \times n)-n}{2}$ or $\frac{n(n-1)}{2}$
- Number the nodes from 1 to n. Beginning at node 1, it can have up to $n-1$ edges with the other $n-1$ nodes. The next, up to $n-2$ edges, the next, $n-3$, and so on. This gives the sum $(n-1)+(n+2)+ ... + 2 + 1$, which is also equal to $\frac{n(n-1)}{2}$ through a combinatorial proof (not covered in this class).
- In a complete graph, all n nodes would have a degree of $n-1$. Sum this up across all nodes, and the total degree is $n(n-1)$. Then, to find the number of edges, divide by 2 to remove the duplicates, because one edge will be one degree in two separate nodes. This also gives $\frac{n(n-1)}{2}$
- Consider all n nodes as a set. From this set, one can *choose* any unordered pair of two nodes to form an edge. This is the definition of the *choose* operation, and so there are $\binom{n}{2}$ possible edges. This again is equivalent to $\frac{n(n-1)}{2}$ edges by combinatorial proof.

Now that we know that there are $\frac{n(n-1)}{2}$ possible edges, we can think of these edges as being a set of edges, and each possible graph on n nodes having some subset of those edges. To figure out how many different graphs are possible, we must figure out how many possible subsets there are for a set with $\frac{n(n-1)}{2}$ edges. This is the same as finding the size of the power set of the set of edges, which is $2^{\frac{n(n-1)}{2}}$.

Q9.2

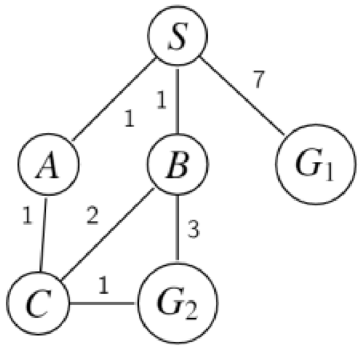
Develop a pseudocode method for a recursive implementation of DFS.

9.2 Answer

```
    DFS-Recursive(Graph, start):
    visited = set()
    Recursive-DFS(start, visited)
Recursive-DFS(node, visited):
    if node is not visited:
        visit node
        add node to visited set
        for each neighbor of node:
            if neighbor is not visited:
                Recursive-DFS(neighbor, visited)
```

Q9.3

SOURCE: UW Madison https://pages.cs.wisc.edu/ dyer/cs540/exams/exam1-f19-sol.pdf
Consider the following graph.



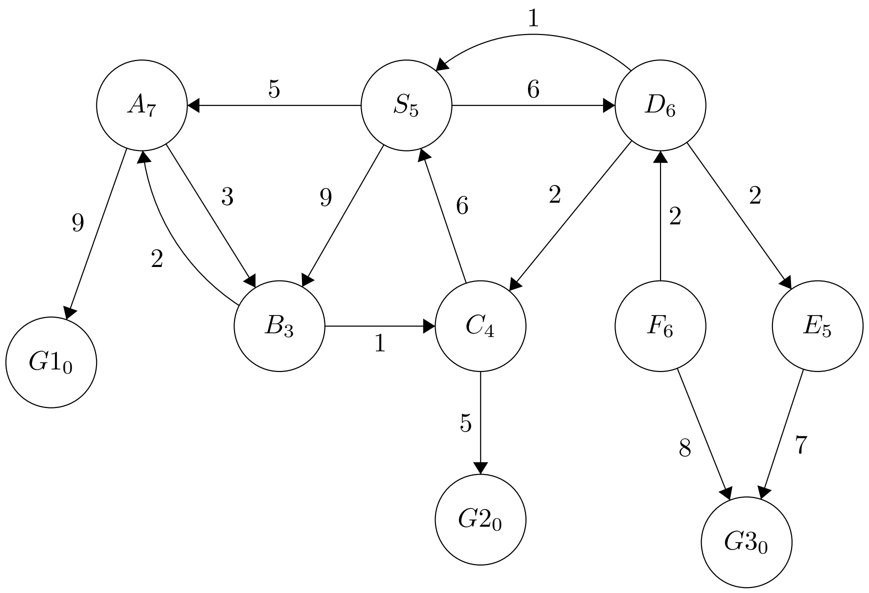
S is a start state and G1 and G2 are end states. Assume that nodes are searched in alphabetical order. Conduct a BFS and UCS on the graph.
What path does BFS return and what is its cost? What path does UCS return and what is its cost? Produce the overall results of both search methods as a rigorous proof.

9.3 Answer

BFS returns the path from S to G1 with a cost of 7. It will push all the neighboring nodes from S, but terminate upon finding that G1 is a goal state and is a neighbor of S.
UCS returns the path from S to A to C to G2 with a cost of 3. It will first check from S to A and B, being that their cost is 1. Then, it will add the path from S to A to C, because its cost is 2. It will add the paths from S to B to G2 and S to B to C, but those have costs higher than the path from S to A to C. Resuming from S to A to C, it will add the path S to A to C to G2, which will have the lowest cost of any other paths, and then will return that path with cost of 3.

9.4

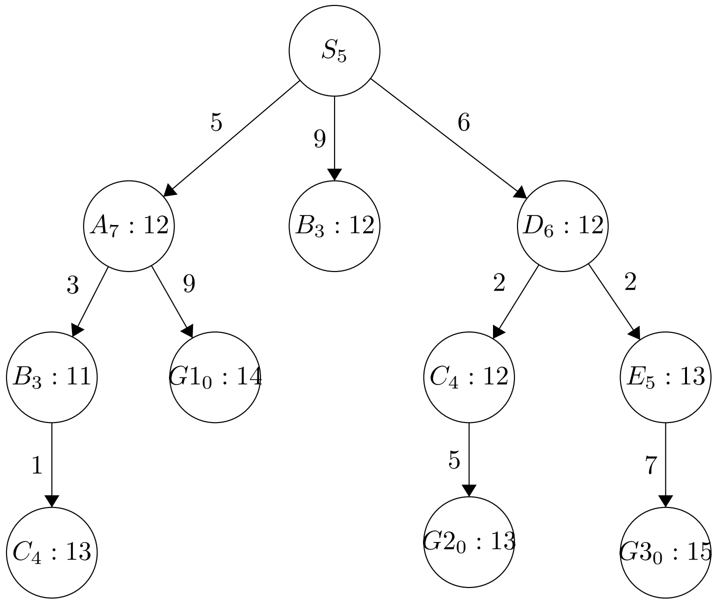
Run A^* search on the following graph, starting at node S and ending at any of the 3 possible goal G nodes. The subscripts on each node are their heuristic values, and you may assume that it is an admissible, consistent heuristic. Show the final search tree and show or explain the state of the priority queue at each step, keeping a closed list of visited nodes.



9.4 Answer

At the start of the algorithm, S is pushed onto the queue with an A^* score of 5. Its outgoing edges are to A_7, B_3, and D_6. However, since $5 + 7 = 9 + 3 = 6 + 6 = 12$, the algorithm considers all paths equal and all three nodes are pushed onto the queue with an A^* score of 12.
Starting with node A, its outgoing neighbors are B and G1, with A^* scores of 11 and 14, respectively. Both are added to the queue, and since (B, 11) has a higher priority than (B, 12), it is visited first.
B's outgoing neighbors are A, which has already been visited in a shorter path, and C, which is pushed onto the queue with an A^* score of 13.
The next highest priority is B again with a score of 12, but this is longer than the (B, 11) path that was already visited, so it goes no further.
Next priority is (D, 12), whose outgoing neighbors are S (also previously visited with a shorter path), C with a score of 12, and E with a score of 13.
Next priority is (C, 12), whose outgonig neighbors are S and G2 with a score of 13.
Next priority is (C, 13), which has already been visited by a shorter path.
Next priority is (E, 13), whose only outgoing edge is to G3 with a score of 15.
Now, only paths to the goal are on the priority queue, and so the algorithm returns the path from S to D to C to G2 as the shortest path with a total score of 13.
Below is the full priority queue and search tree with A^* scores filled in at each node.

Pop	Put	Priority Queue
	(S, 5)	(S, 5)
(S, 5)	(A, 12) (B, 12) (D, 12)	(A, 12) (B, 12) (D, 12)
(A, 12)	(B, 11) ($G1_0$, 14)	(B, 11) (B, 12) (D, 12) ($G1_0$, 14)
(B, 11)	(C, 13)	(B, 11) (B, 12) (D, 12) (C, 13) ($G1_0$, 14)
(B, 12)		(D, 12) (C, 13) ($G1_0$, 14)
(D, 12)	(C, 12) (E, 13)	(C,12) (C, 13) (E, 13) ($G1_0$, 14)
(C, 12)	($G2_0$, 13)	(C, 13) (E, 13) ($G2_0$, 13) ($G1_0$, 14)
(C, 13)		(E, 13) ($G2_0$, 13) ($G1_0$, 14)
(E, 13)	(G3, 15)	($G2_0$, 13) ($G1_0$, 14) (G3, 15)



Additional Resources for Graphs and Trees

1. DFS and BFS
2. Graph Theory Playlist
3. Uniform Cost Search
4. A* Search

Chapter 5: Regular Expressions

Q5.1

Assume the alphabet $\sum = \{0, 1\}$. If the English description is given, provide the regular expression. If the regular expression is given, give a short, English description of the accepted set of strings.

a. The set of strings that contain at least one 0 and at least one 1.

b. $(1 + \lambda)(00^*1)^*0^*$

c. The set of strings such that all pairs of adjacent 0's appear before any pairs of adjacent 1's.

Q5.1 Answer

a. This can be thought of as either a 0 appearing before the 1, or the 1 appearing before the 0. Since these are the only mandated characters, the rest of the string can be any combination of 0's and 1's that come before, in-between, or after the 0 and 1. In other words, there are two possible string formats: $(0+1)^*0(0+1)^*1(0+1)^*$ or $(0+1)^*1(0+1)^*0(0+1)^*$
Together, the full regex is $((0+1)^*0(0+1)^*1(0+1)^*) + ((0+1)^*1(0+1)^*0(0+1)^*)$

b. The language starts with either a 1 or an empty string. From there, it is followed by a 0, some number of zero's, and then a 1. Just looking at these parts, the string can start as either:
1 0 0... 1
0 0... 1
Even if the 0... is omitted, the string can only start with 0 1. 1 1 is not allowed.
Looking further, we can observe that within the (00^*1) part, removing the optional 0's, (01) the structure mandates that 0 must come before 1. So if this block repeats, the structure will looks like this: 0 1 0 1 with more 0's padding the in-between. And since only more 0's are added onto the end, we can conclude that this will be the set of all strings without consecutive 1's.

c. This can be divided into two parts: The first part of the string that does not allow 11's, but does allow 00's, and the second part of the string that does not allow 00's, but does allow 11's.
For a binary string where 11 cannot occur, this could be any combination of 0's or 10's, either ending in a 1 or not: $(0+10)^*(1+\lambda)$
The formula is much the same for a binary string where 00 cannot occur: $(1+01)^*(0+\lambda)$
Concatenating the two together gives us the final string: $(0+10)^*(1+\lambda)(1+01)^*(0+\lambda)$

Q5.2

Is the Kleene star operation distributive over the + operation? In other words, is $(S+T)^* = S^* + T^*$ true for any languages S and T? Why or why not?
If your answer is no, is this always true, or can you think of specific values for S and T that would make it true?

Q5.2 Answer

No, they are not. $(S+T)^*$ indicates choosing between S and T, and then repeatedly concatenating this any number of times. Therefore, it produces combination strings that can switch between S and T. However, $S^* + T^*$ only chooses between S and T after the repeated concatenations, and so it cannot produce strings with combinations of S and T—it can only produce strings that are either repeated concatenations of S or repeated concatenations of T.
The above is generally true, but when either S or T (or both) are the empty language, or when S and T are the same language, $(S+T)^* = S^* + T^*$.

Additional Resources for Regular Expressions

1. Formal Language Theory Playlist

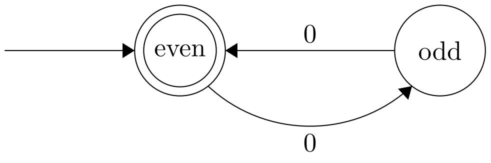
Chapter 14: Finite Automata

Q14.1

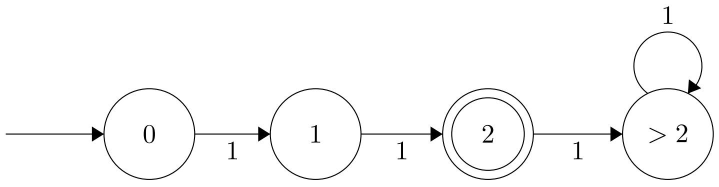
Assume $\sum = \{0, 1\}$. Draw a DFA for the language that accepts all binary strings with either an even number of 0's or contains exactly two 1's.

Q14.1 Answer

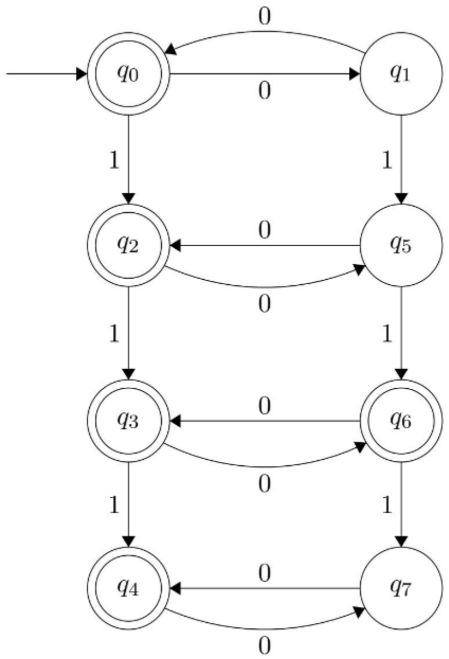
Let us first consider what states this machine will need. The machine considers strings based on two different criteria, although a string only needs one to be accepted. The first condition is that there is an even number of 0's. There are only two states for this, an even number of 0's or an odd number of 0's. Reading in 0's would switch between the two states like so:



Now let us consider the other criteria, which is "exactly two 1's". This has four states: zero 1's, exactly one 1, exactly two 1's, and more than two 1's, since beyond two 1's, that criteria will always fail. Reading in any 1 will move between these states, ending with the "more than two 1's" state in a dead end.



Since one criteria only cares about 0's being read in, and the other only cares about 1's being read in, we can combine the two into one machine with 8 states, each representing a state from the even/odd criteria *and* a state from the "how many 1's" criteria.



Q14.2

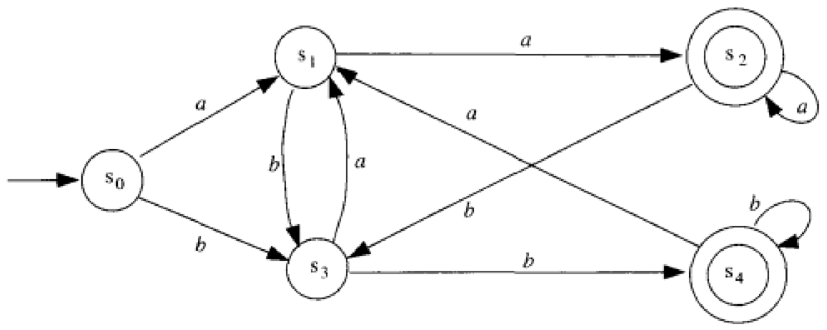
Draw a DFA that accepts the language consisting of the set of all strings of x's and y's that start with xy or yy followed by any string of x's and y's.

Q14.2 Answer

The given diagram represents a DFA comprising five states, s_0 to s_4, with s_0 as the start state and s_2, s_4 as accept states. The transitions are defined by the input symbols 'a' and 'b':

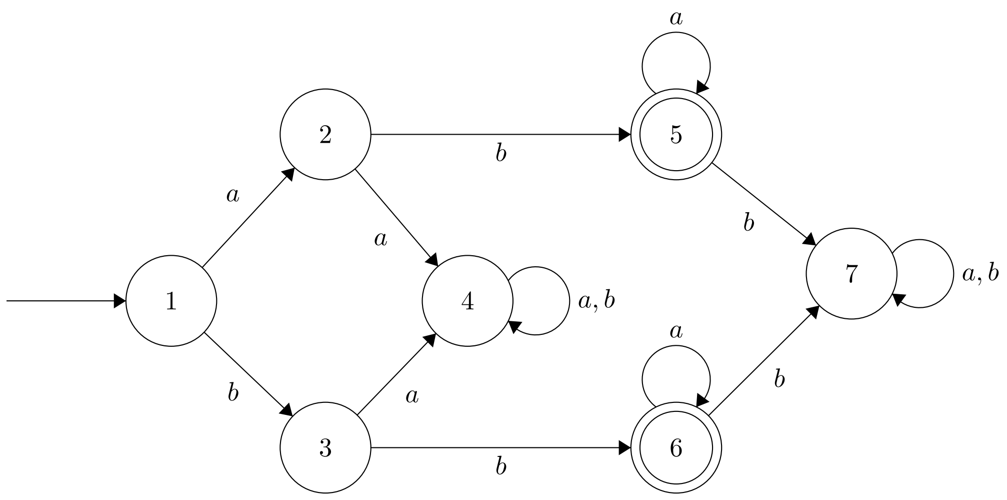
- From s_0, there is a transition to s_1 on input 'a' and a loop on 'b'.
- State s_1 transitions to s_2 on 'a' and to s_3 on 'b'.
- State s_2 has a loop on 'a'.
- State s_3 transitions to s_1 on 'a' and to s_4 on 'b'.
- State s_4 has a loop on 'b'.

This DFA accepts strings that end with 'a' leading to s_2 or strings with an even number of 'b's leading to s_4.



Q14.3

Use the DFA minimization algorithm on the following DFA:



Q14.3 Answer

The DFA minimization algorithm begins by separating the nodes into accept states and non-accept states. Each of these groups is called a "class", and each iteration of this algorithm is called a relation. The point of the algorithm is to divide each class up into smaller and smaller groups of nodes until they cannot be divided anymore, meaning we have reached the minimal DFA.

Relation 0: Class A = $\{1, 2, 3, 4, 7\}$ Class B = $\{5, 6\}$

Below, we have a table showing each node in each class and which class its edges point towards. Nodes 2 and 3 have a different behavior than the other nodes in A—when reading in a 'b', they lead to class B, not class A. So we will separate out those nodes into their own class in the next iteration.

Node	Class (a input)	Class (b input)
1	A	A
2	A	B
3	A	B
4	A	A
7	A	A
5	B	A
6	B	A

Relation 1: Class A = $\{1, 4, 7\}$ Class B = $\{5, 6\}$ Class C = $\{2, 3\}$

In the below table, we relabel the edges to reflect the new classes, and once again observe that node 1's behavior is unlike the others in Class A. Therefore, again, we put it in its own separate class.

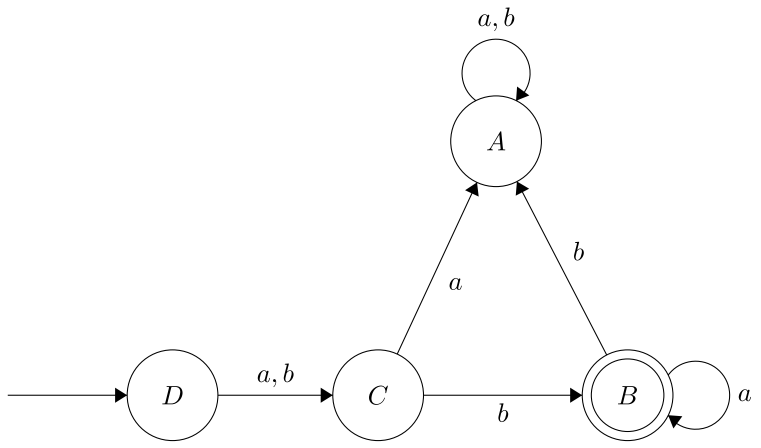
Node	Class (a input)	Class (b input)
1	C	C
4	A	A
7	A	A
2	A	B
3	A	B
5	B	A
6	B	A

Relation 2: Class A = $\{4, 7\}$ Class B = $\{5, 6\}$ Class C = $\{2, 3\}$ Class D = $\{1\}$

In the below table, we relabel the edges to reflect the new classes, and once again observe that node 1's behavior is unlike the others in Class A. Therefore, again, we put it in its own separate class.

Node	Class (a input)	Class (b input)
1	C	C
4	A	A
7	A	A
2	A	B
3	A	B
5	B	A
6	B	A

We have now reached the end of the minimization algorithm, since all behaviors in each class are the same, meaning all the nodes in each class can be condensed into a single node with one uniform output and input that is representative of all the nodes within. The minimal DFA is shown below.

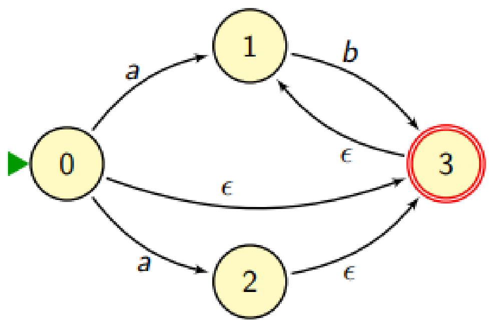

a, b
A
a
b
D
a, b
C
b
B
a

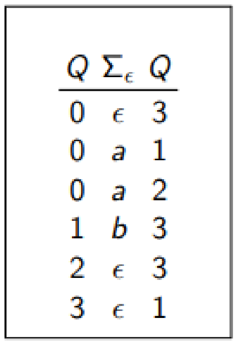
Q14.4

Convert the NFA below into a DFA using the subset construction. This should be done by:

- Creating DFA states as subsets of the NFA states.
- Defining the start state of the DFA as the epsilon-closure of NFA state 0.
- Including any subset containing NFA state 3 as accept states in the DFA.
- Determining the DFA transitions for each symbol from each state subset.

Provide the transition table for the DFA.





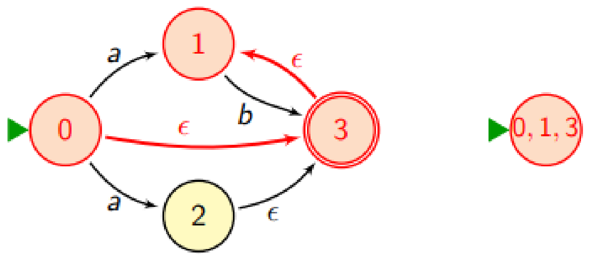
Q	Σ_ϵ	Q
0	ϵ	3
0	a	1
0	a	2
1	b	3
2	ϵ	3
3	ϵ	1

Q14.4 Answer

Each state of the DFA is a set of states of the NFA. The DFA's initial state, S_0, is the epsilon-closure of the NFA's initial state, which includes states $\{0, 1, 3\}$.

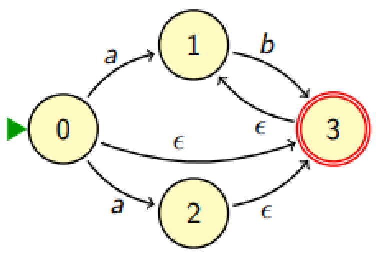
$$\varepsilon\text{-closure}\{0\} = \{0, 1, 3\} = S_0$$

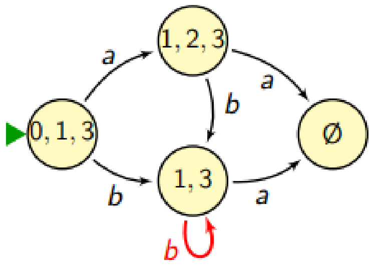
$$\epsilon\text{-CLOSE}\{0\} = \{0, 1, 3\} = S_0$$



For input a, the transition from S_0 leads to a new DFA state, S_1, determined by the epsilon-closure of the set $\{1, 2\}$, resulting in the set $\{1, 2, 3\}$. For input b, the transition from S_0 leads to state S_2, which is the epsilon-closure of the set $\{3\}$, or $\{1, 3\}$. The DFA transitions from state S_1 on input a lead to a sinking state, as there are no corresponding NFA transitions. Conversely, on input b, S_1 transitions to the previously established state S_2. The transitions from state S_2 on inputs a and b need to be determined following the same methodology, considering the epsilon-closures and transitions in the NFA. The final diagram is :

$$\begin{aligned}
\delta(S_0, a) &= \epsilon\text{-CLOSE}\{1, 2\} = \{1, 2, 3\} = S_1\\
\delta(S_0, b) &= \epsilon\text{-CLOSE}\{3\} = \{1, 3\} = S_2\\
\delta(S_1, a) &= \epsilon\text{-CLOSE}\{\} = \emptyset = S_3\\
\delta(S_1, b) &= \epsilon\text{-CLOSE}\{3\} = \{1, 3\} = S_2\\
\delta(S_2, a) &= \epsilon\text{-CLOSE}\{\} = \emptyset = S_3\\
\delta(S_2, b) &= \epsilon\text{-CLOSE}\{3\} = \{1, 3\} = S_2
\end{aligned}$$





Q14.5

Let L^R be any arbitrary regular language. Let the language $L^R = \{w^R : w \in L\}$ be the reversal of L. Show that L^R is regular.

Q14.5 Answer

Since we know that L is regular, we know that there exists some DFA or equivalent λ-NFA which accepts L. Without loss of generality, we can assume that there exists only start or end state (If a DFA had multiple accepting states, you could eliminate them down to just one by redirecting all of the arrows going into other accepting states into a single accepting state, and eliminating the redundant accepting states). Then we can construct a DFA or equivalent λ-NFA which accepts L^R by making the start state and accepting state, the accepting state a start state, and reversing every state transition (reversing the "arrows"). Then we have a machine that traverses backwards over every string in L, and have thus constructed a DFA for L^R. Since there exists a DFA / λ-NFA for L^R, we know that L^R is regular.

Q14.6

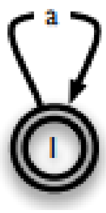
For $\Sigma = \{a, b, c, d\}$, give a regular expression that captures all strings that use their letters in reverse alphabetical order, but use at most three of the four possible letters.
Note: The strings themselves can be longer than 3 letters long, since letters can repeat. Draw an NFA that captures the regular expression from above.

Q14.6 Answer

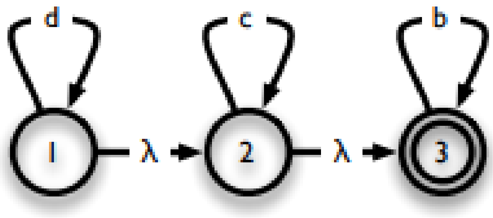
The regular expression that captures the above language is:

$$(d^*c^*b^*) + (d^*b^*a^*) + (d^*c^*a^*) + (c^*b^*a^*)$$

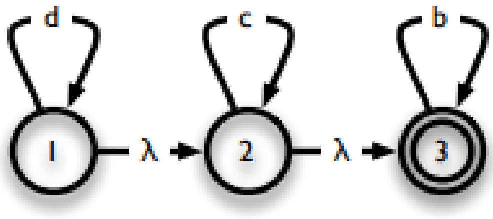
We will construct this NFA step by step. Initially, consider the NFA for a^* as follows:



The NFA for $d^*c^*b^*$ is constructed by chaining the NFAs for d^*, c^*, and b^* as shown:



Extra care is taken when connecting the NFAs sequentially; the final states of the first NFA are connected to the start state of the subsequent NFA, and the final states are made non-final to ensure the correctness of the overall NFA. Although it is safe to keep all states final in this specific case, it is generally prudent not to do so in order to avoid errors in the construction of the NFA. Now we can combine the components to construct the overall NFA.



Additional Resources for Finite Automata

1. Myhill Nerode
2. DFA Minimization
3. Regular Expression to Automaton
4. Lambda/Epsilon NFA to NFA

Chapter 15: Formal Language Theory

Q15.1

Let L be a language that is Turing Recognizable by some Turing Machine. Let L' be the language $\Sigma^* - L$, and let that language also be Turing Recognizable by some other Turing Machine. Prove that L is Turing Decidable. (Hint: Recall that a single-tape Turing Machine can be used to simulate a multi-tape Turing Machine)

Q15.1 Answer

We know this to be true by the TR / TD Theorem. But for a rigorous proof, recall that a multi-tape Turing Machine can be simulated by a single-tape Turing Machine. Then we will construct a multi-tape Turing Machine which runs the Turing Machines for L and L' in parallel. One of these is guaranteed to terminate, since $L + L' = \Sigma^*$. If, on any given string, the Turing Machine that recognizes L terminates, we know that string is in L. If the Turing Machine that recognizes L' terminates, we know that string is not in L. Then here we have constructed a Turing Machine that always decides whether or not a string is in L, and so L is Turing Decidable.

Additional Resources for Formal Language Theory

1. Turing Machines Visualized
2. TR / TD
3. Halting Problem
4. Theory of Computation Playlist
5. `https://www.geeksforgeeks.org/conversion-from-nfa-to-dfa/`