CS 250 Course Study Guide

Sarna Gomasta, Kyla Levin, Alexander Yeung

Fall 2023

Chapter 1: Sets and Predicate Logic

This section has the following learning objectives:

1. Translate statements to and from propositional logic
2. Prove statements of propositional logic using truth tables and/or propositional proof rules.
3. Translate statements to and from predicate logic.
4. Prove statements of predicate logic using the four quantifier proof rules.
5. Know and work with the definitions of relations (partial orders and equivalence relations) and properties of functions (one-to-one, onto, bijective).

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.

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." $D \to \neg S$
- This can be read as "There is nothing such that if it agrees with me, then it's good for dinner." $A \to G$
- This can be read as "If it's cake, then it's very sweet." $C \to S$
- This can be read as "If it's good for dinner, then my doctor recommends it." $G \to D$

$C \to S \to \neg D \to \neg G$
Transitively, $C \to \neg G$
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.

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

1.2 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. $\neg p \land q$
2. $r \to p$
3. $\neg r \to s$
4. $s \to t$

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.

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 $\neg p \land q \land \neg r \land s \land t$

1.4

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.

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!

Chapter 2: Quantifiers and Relations

2.1

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 $x\%2 = y\%2$

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 $x\%2 = z\%2$.

2.2

TODO: Function question

Answer

Chapter 3: Number Theory

Q3.1 General Number Theory

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.

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 Euclid's Algorithm

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?

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.

Chinese Remainder Theorem (CRT)

Chapter 4: Induction and Recursion

4.1

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.

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.

4.2

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 $n \geq 0$.

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:

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

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

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

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

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

Chapter 9: Graphs and Trees

9.1

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.

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 $2^{\frac{n(n-1)}{2}}$.

Chapter 5: Regular Expressions

5.1

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.

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: $(0+10)^*(1+\lambda)(1+01)^*(0+\lambda)$

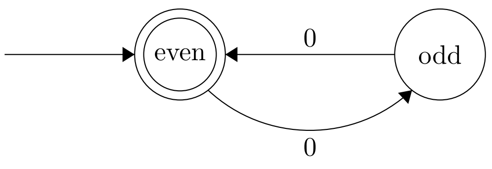
Chapter 14: Finite Automata

14.1

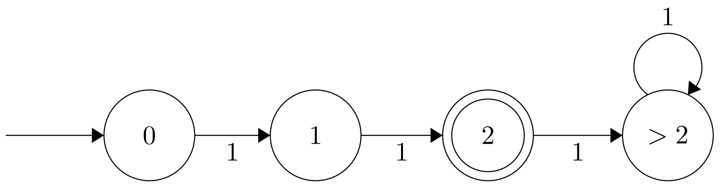
Assume $\sum = \{0, 1\}$. Draw a DFA for the language that accepts all binary strings with either an even number of 0's or contains exactly two 1's.

Answer

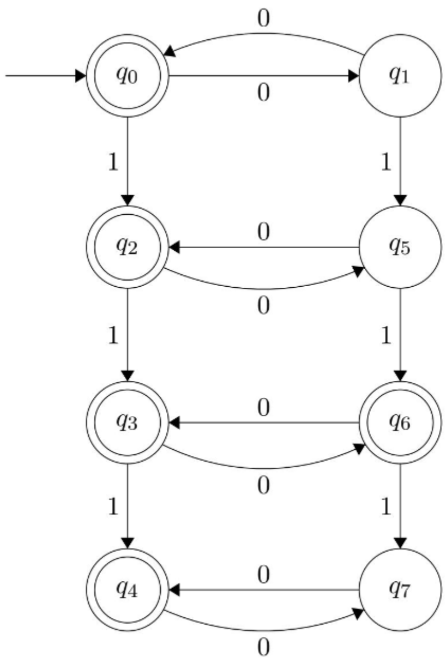
Let us first consider what states this machine will need. The machine considers strings based on two different criteria, although a string only needs one to be accepted. The first condition is that there is an even number of 0's. There are only two states for this, an even number of 0's or an odd number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



14.2

TODO: DFA Simplification question

Answer

14.3

TODO: DFA to NFA conversion question

Answer

0.1 14.4

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.

Answer

Since we know that L is regular, we know that there exists some DFA or equivalent 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 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 / NFA for L^R, we know that L^R is regular.

14.5

TODO: $\lambda-$NFA problem

Answer

Chapter 15: Formal Language Theory

15.1

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)

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.