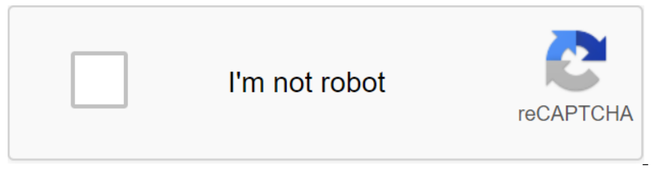

Continue

Introduction to automata theory solution manual pdf

Introduction to Automata Theory, Languages and Computing Solutions for Chapter 2 Solutions for Chapter 3 Solutions for Chapter 4 Solutions for Chapter 5 Solutions for Chapter 6 Solutions for Chapter 7 Solutions for Chapter 8 Solutions for Chapter 9 Solutions for Chapter 10 Solutions for Chapter 11 Academia.edu No longer Supports the Internet Explorer.To Browse Academia.edu and the wider Internet is faster and more secure, please take a few seconds to update the browser. Academia.edu uses cookies to personalize content, adapt ads, and improve user experience. Using our website, you agree to our collection of information using cookies. To learn more, review our privacy policy.Introduction to Machine Theory, languages and computing solutions for Chapter 2 Revised 9/6/01. Decisions under Section 2.2 Exercise 2.2.1 (a) states correspond to eight combinations of switch positions, and must indicate whether the previous roll was published in D, i.e. whether the previous entry was accepted. Let 0 represent the position on the left (as in the chart) and 1 position to the right. Each state can be represented by a sequence of three 0 or 1, representing the directions of the three switches, in order to be left to right. We follow these three bits, indicating that it is a state of acceptance, or r, pointing to a failure. Из 16 возможных состояний, оказывается, что только 13 доступны из первоначального состояния, 000r. Вот таблица перехода: A B -000r 100r 011r 000a 100r 011r 001a 101r 000a 010r 110r 001a 010a 110r 001a 011r 111r 010a 100r 010r 111r 100a 010r 111r 101r 011r 100a 101a 011r 100a 110 000a 101a 110a 000a 101a 111r 001a 110a Упражнение 2.2.2 В следующем, мы используем dhat для 'дельта-шляпа.' Заявление, которое будет доказано, является dhat (q,xy) - dhat (dhat(q,x), и мы приступаем к индукции по длине y. Basis: Если y эпсилон, то заявление dhat (q,x) - dhat (dhat(q,x), epsilon). Это утверждение вытекает из основы в определении dhat. Обратите внимание, что при применении этого определения, мы должны относиться к dhat (q ,x), as if it were just a state, say, r. Then, a statement to be proven p th dhat (p, epsilon), which is easy to recognize as the basis in the definition of dhat. Induction: Suppose that the statement for the strings is shorter than that of the one, and break y q za, where is the last symbol y. Steps of conversion dhat (q,x),y) in dhat (q,xy) are summed up in the following table: Expression Of the Reason dhat (q,x), y) Beginning dhat (dhat (q,x), treatment dhat (q,x) as delta state (dhat (q,xz), a) Inductive hypothesis dhat (q,xza) Definition of dhat dhat (q,xy) y'za Exercise 2.2.4 (a) Intuitive value of states A, B and C in that, that line, seen so far, ends in 0, 1 or at least 2 zeros. 0 1 - A B A BCA C 2.2.6 (a) The trick is to understand that reading another either multiplies the number seen so far by 2 (if it's 0), or multiplied by 2, and then adds 1 (if it's 1). We don't need to remember all the number we've seen and --- the rest of it when divided into 5. That is, if we have any amount of form 5a'b, where b is the balance, between 0 and 4, then 2 (5a'b) and 10a'2b. Since 10a is definitely divided into 5, the remaining 10a'2b is the same as the balance of 2b when divided into 5. Since b, it's 0, 1, 2, 3 or 4, we can easily identify the answers. The same idea takes place if we want to consider what happens to 5a'b if we multiply by 2 and add 1. The table below shows this machine. The state of qi means that the contribution has seen so far the remainder of I when divided into 5. 0 1 - q0 q1 q2 q2 q2 q4 q0 q3 q2 q2 q4 q3 q4 q4 There is a small matter, however, that this machine takes the strings with the leading 0's. Since the problem requires taking only those lines that start at 1, we need an additional state of s, a state of start and an additional 'dead state' d. If in a state of s, we see 1 first, we act as q0; i.e. we are going to the state q1. However, if the first entry is 0, we should never accept, so we go to state d, which we never leave. Full vending machine: 0 1 - s d q1 q0 q0 q1 q1 q2 q2 q2 q2 q4 q0 q3 q1 q2 q4 q4 q4 dd d Exercise 2.2.9 Part (a) is a light induction by length of W, starting at length 1. Base: No 1. Then dhat (q_0,w) - dhat (q_f,w) because w is one symbol, and dhat agrees with the delta on single symbols. Induction: Let the et zh, so the inductive hypothesis refers to z. Then dhat (q_0,w) - dhat (q_0,za) - delta (q_0,z), a) - delta (q_f,z), a((inductive hypothesis) - dhat (q_f,za) - dhat (q_f,w). For our part (b), we know that dhat (q_0,x) and q_f. Since xepsilon, we know by part (a) that dhat (q_f,x) and q_f. Then it is a simple induction on k to show that dhat (q_0,x'k) q_f. Induction: Suppose the statement for K-1; i.e. dhat (q_0,x'k-1) - q_f. Use Exercise 2.2.2, dhat (q_0,x'k) - dhat (q_0,x'k-1), x) - dhat (q_f,x) (according to inductive hypothesis) - q_f (a) . Exercise 2.2.10 Automaton shows whether the number 1 is seen even (state A) or the odd (state B) taking in the latter case. It's a light induction on to show that dh'A,w - A if and only if w has a pretty number 1's. Base: No 0. Then, the empty line certainly has a pretty number of 1's, namely zero 1's, and dhat (A,w) and A. Induction: Suppose the statement for the strings is shorter than w. Then w th za, where either 0 or 1. Case 1: a No 0. If w has quite number 1's, so does z. According to the inductive hypothesis, dhat (A,z) - A. DFA transitions tell us dhat (A,w) If w has an odd number 1's, then so does z. According to the inductive hypothesis, dhat (A,z) - B, and DFA tell us dhat (A,w) - B. So, in this case, dhat (A,w) - if and only if w has number 1's. Case 2: a No 1. If w has an even number 1's, z has an odd number 1's. According to the inductive hypothesis, dhat (A,z) - B. DFA transitions tell us dhat (A,w) If w has the odd number 1's, then z has an even number 1's. According to the inductive hypothesis, dhat (A,z) - A, and the transitions of the DFA tell us dhat (A,w) - B. So, in this case, as well as, dhat (A,w) - A if and only if w has a pretty number 1's. Solutions for Section 2.3 Exercise 2.3.1 Here are the NFA state sets submitted by each of the DFA states through H: A B.S.P., apt; C q p,r; D.S.P., sq.r. E.S.P., q,s;; F.P., sq.r.,s;; G.S.P., r.s;; H.S.P., p. 0 1 - A B A B BDC CE A DF C q E F F F F E H H Exercise 2.3.4 (a) The idea is to use the state of qi, for i q 0.1,...,9 to represent the idea that we saw the entry I and guessed that this is a repeated figure at the end. We also have a state of qs, an initial state, and a qf, final state. We stay in the state of qs all the time; it does not constitute an assumption of what has been done. Transition table: 0 1 ... 9 - qs,q0 qs,q1... «qs,q9» q0 «qf» (q0)... (q0) q1 (q1) ... . q9 (q9) ... (qf) {} ... Solutions for Section 2.4 Exercise 2.4.1 (a) We will use q0 as a start state. q1, q2 and q3 recognize abc; q4, q5 and q6 recognize abd, and q7 through q10 recognizes aacd. Transition Table: B c d-q0 (q0,q1,q4,q7) (q0) (q0) Exercise 2.4.2(a) q1 (q2) (q6) (q6) (q6) (q6) (q7) (q8) (q8) (q9) (q9) (q9) (q9) (q9) (q10) (q10) (q10) (q10) (q8) (q9) (q9) (q9) (q9) (q10) (q10) (q10) (q10) (q10) (q10) (q8) (q9) (q9) (q9) (q9) (q9) (q10) (q10) (q10) (q10) (q10) (q8) (q9) (q9) (q9) (q9) (q10) (q10) (q10) (q10) (q10) (q8) (q9) (q9) (q9) (q9) (q10) (q10) (q10) (q10) (q8) (q9) (q9) (q9) (q9) (q10) (q10) (q A subset of construction gives us the following states, each of which is a subfacence of the NFA. B (q0,q1,q4,q7); C q0,q1,q4,q7,q8;; D q0,q2,q5;; E q0,q9; F (q0,q3); G (q0,q6); H q0,q10. Note that F, G and H can be combined into one acceptance state, or we can use these three states to signal the recognition of abc, abd and aacd, respectively. a b c d ->A B A A A BCDAA CCDE A DBAF G EBAAH *F B A A A *G B A A A *H B A A A Solutions for Section 2.5 Exercise 2.5.1 For part (a): the closure of p is just {p}; for q it's p,q and for r it's p,q,r. For (b), start by noticing that it always leaves the state unchanged. So we can think about the impact of b's and c's lines only. To begin with, note that the only ways to get from p to d for the first time, using only b, c, and epsilon transitions bb, bc, and c. After receiving r, we can go back to r reading or b or c. So each line of length 3 or less, consisting only of b's and c's, is accepted except for line b. However, we must allow And. When we try to insert a's into these lines while keeping the length to 3 or less, we find that each b's line, and c's with no more than one a is accepted. Also Consisting of one C and up to 2 a's are accepted; other lines are rejected. There are three DFA states available from the original state, which is an epsilton close p, or p. Let the AP, B and S. then the transition table: b c -A B C B B B BB C C C C C Introduction to Automation Theory, languages, and computational solutions for Chapter 3 Solutions for Section 3.1 Exercise 3.1.1(a) The simplest approach is to consider those lines in which the first a precedes the first b separately from those where the reverse occurs. Expression: c'a (a'c) b (a'b'c) Here is one such expression: (10'0) (Epsilon-1) To understand why this expression works, the first part consists of all lines in which each 1 follows 0. To that, we should only add the possibility that there is one at the end that will not follow 0. That is the work of os (Epsilon No.1). Now we can rethink the question as asking for strings that have a set-top box without an adjacent one followed by a suffix without an adjacent 0's. The first is the expression we have developed, and the second is the same expression, with 0 and 1 interchangeable. Thus, the solution to this problem is that (10'0) (Epsilon-1) (01'1) (Epsilon-0). Note that the term Epsilon-1 is not really needed in the middle, as 1 corresponding factor can be derived from the coefficient (01'1). Exercise 3.1.4 (a) This expression is another way to write 'no adjacent 1's.' you should compare it to the different kind of expression we developed in tackling Exercise 3.1.2(a). The argument is why it works similar. (00'1) says that every one is preceded by at least one 0. 0 at the end allows 0 after the final 1, and (Epsilon-1) at the beginning allows the initial one, which should be either the only symbol of the string or followed by 0. Exercise 3.1.5 Language of regular Epsylon expression. Note that Epsilon means string language consisting of any number of blank lines sweeter, but this is only a set containing an empty string. Solutions for Section 3.2 Exercise 3.2.1 Part (a): Below are all expressions R ' (0); We list only the signings. R11 - Epsilon No.1; R12 No 0; R13 and fi; R21 No 1; R22 and Epsilon; R23 No 0; R31 and fi; R32 No 1; R33 and Epsilon-0. Part (b): Here are all R' (1) names; we re-list only the subscripts. R11 and 1 euro; R12 and 1'0; R13 and fi; R21 and 11 euros; R22 and Epsilon 11'0; R23 No 0; R31 and fi; R32 No 1; R33 and Epsilon-0. Part (e): Here's the transition chart: If we eliminate the q2 state, we'll get: Applying the formula in the text, Expression for ways to receive from q1 to q3 is: 1 01 00 (010) 1100 (0'10) Exercise 3.2.4 (a) Exercise 3.2.6 (a) (Revised 1/16/Exercise) LL or L. 3.2.6 (b) Set suffixes of strings in L. Exercise 3.2.8 Let R ' (k)_ijm number of paths from I to J-length, we can calculate these numbers, for all states I and J, and for m no more than, by induction to k. Basis: R' (0)_ij1 is the number of arcs (or, more precisely, the label arc) from the state I to the state of J. R' (0)_ii0 and 1 and all the other R' (0)_ijm are 0. Induction: RD (c)_ijm is the amount of R-K-1) _ijm and the amount on all lists (p1,p2,...,pr) of positive integrators, that amount up to M, RS (k-1)_ikp1 e RH (to-1)_kkp2 (K-1)_kjpr _kkp3 y... RS (c1)_kkp (p-1) Note R should be at least 2. The answer is the amount of R'k) _1jn where k is the number of states, 1 is the state of origin, and j is any acceptable state. Solutions for Section 3.4 Exercise 3.4.1 (a) Replacing R with A and S with b. Then the left and right become the union b (b) a. That is, a,b b,a. Because order doesn't matter in sets, both languages are the same: a language consisting of lines A and b. Exercise 3.4.1 (f) Replacing R to a. The right side becomes a, that is, all lines A, including an empty line. The left side is all strings consisting of the concate of strings strings a. But this is only a set of a's strings, and therefore equals to the right side. Exercise 3.4.2 (a) Not the same. Replace R with a and S with b. The left side becomes all strings a and b (mixed), while the right side consists only of strings a (one) and b's strings (one). A line like AB on the left side, but not to the right. Exercise 3.4.2 (c) is also not the same. Replace R with a and S with b. The right side consists of all lines consisting of zero or more cases of the appearance of lines form a... AB, that is, one or more a ended one b. However, each line on the language of the left side should end in AB. For example, Epsilon is in the language on the right, but not on the left. Introduction to Automata Theory, Languages and Computing Solutions for Chapter 4 Solutions for Section 4.1 Exercise 4.1.1 (c) Let it be pumping the lemma standing (note that this is not relevant to what is a local variable in the definition of L language). Choose w'n10'n. Then when we write w and xyz, we know that 0, so we have to choose untidy w. Since w should consist of pairs 00 and 11, the opponent can choose u to be one of these pairs. Then no matter what we choose, xy'iz will consist of pairs 00 and 11, and therefore belongs in the language. Solutions for Section 4.2 Exercise 4.2.1 (a) aabbaa. Exercise 4.2.1 (c) Regular Expression Language A(ab) .b. Exercise 4.2.1 (e) Each b must come from 1 or 2. However, if the first b comes from 2 and the second comes from 1, then they will both need between them as part h(2) and h(1), respectively. Thus, the reverse consists of strings (110, 102, 022). Exercise 4.2.2 Start with DFA A for L. Create a new DFA B, that is exactly the same as A, except that the state q is a state of acceptance B, if and only if the delta (q,a) is a state of acceptance A. Then B takes the input line w, if only if A takes wa; that is, L (B) - L/a. Exercise 4.2.5(b) We will use D_a for 'derivative in relation to a.' that if the Epsilon is not in L(R), the RS derivative will always remove from the part of the line that comes from R. However, if the epsilon is in L(R), then the line may have nothing from R and will remove from the beginning of the line in L (S) (which is also a line in L(RS). The rule we want is: If the Epsilon is not in L (R), then D_a (RS) - (D_a (R))S. Otherwise, D_a (RS) - D_a (R)S D_a (S). Exercise 4.2.5 (e) L may not have a line that starts with 0. 4.2.5 (f) that whenever 0w is in L, the w is in L, and vice versa. Thus, L should be a form of L(0')M for some M languages (not necessarily a regular language) that does not have a line starting from 0. In proof, note first that D_0 D_0 (L(L(0)M and D_0 (L(0))M union D_0 (M) There are two reasons for the last step. D_0 D_0 (M) is a blank set (this is part (e).D_0 , N can not include a line that is not in L (0)M. If x were such a string, then we can remove all 0 at the beginning of x and get some strings off that also in N. But u also have to be in M. Exercise 4.2.8 Let there be AFA for L. We are building AFA B for half (L). The B state has a q,S form, where: q is the A state in which it will be after reading any B input read so far. S is a set of states A that A can get from these states into acceptance by reading any typing line that is the same length as the length of the B line that has been read so far. It is important to understand that B does not necessarily know how much input it has read so far; he keeps this information up to date every time he reads a new character. The rule that keeps things up to date is: delta_B (s,S),' No. delta_A (q,a), T is a set of states p A in a way that there is a transition from p to any state S at any entry of the symbol. Thus, the first component continues to A, while the second component now represents states that can reach a state of acceptance along a path that is one longer than the path presented by S.S. To complete the construction of B, we should only specify: Initial state q_0,F, that is, the initial state of A and the acceptance of states A. This choice reflects a situation where A read 0 input: it is still in its original state, and the adoption states are exactly the ones that can reach the state of acceptance in the way of a length of 0. By taking provisions b those provisions q,s such that q is in S. The rationale is that it is these states that have reached some sort of string of n lengths, and there are some other n length lines that will take the state q to the adoption of the state. Exercise 4.2.13 (a) Start by adding this language. The result is a language consisting of all the 0's and 1's lines that are not in O'1, as well as the lines in the L_0n1n. If we cross with 0'1, the result is definitely L_0n1n. Because the addition and intersection with the regular set retain a pattern, if the language is regular, it would be L_0n1n. Since we know that the latter is false, we conclude that this language is not regular. Exercise 4.2.14 (c) Change the acceptance of states to be those for which the first component is the acceptance state of A_L and the second is the non-acceptance state A_M. Then the result is DFA accepts if and only if the entry in L - M. Decisions for section 4.3 Exercise 4.3.1 Let it be pumping lemma constant. Check all the lines of length between n and 2n-1 for L membership. The reason is that the pumping lemma is applied to such a line, and it can be pumped to show an infinite sequence of lines in L. Suppose, however, that in L there are no strings that are in the n to 2n-1 range. We claim that in L lengths 2n and no more strings, and thus in L there is only a finn number of rows. In proof, suppose W is a string in L length at least 2n, and w is as short as any line in L, which has a length of at least 2n. Then the pumping lemma applies to w and we can write w q xyz, where xz is also in L. How long can xz be? It can't be as long as 2n, because it's shorter than w, and w as short as any string in L lengths 2n or more. n, because xz is no more n shorter w. So xz has a length between n and 2n-1, which is a contradiction, since we assumed that there were no rows in L with length in that range. Solutions for Section 4.4 Exercise 4.4.1 Revised 10/23/01. Бэкс Кх х X х х Екс х х х х х х Г3 х х х х х Н|х х х х х х х ----------A B C D E F G 0 1 ->AG BF AG BF AG CE CE D BF *D D AG H AG D Note, however, that state H is inaccessible, so it should be removed, leaving the first four states as the minimum-state DFA. Introduction to machine theory, languages and computational solutions for Chapter 5 Revised 11/11/01. для раздела 5.1 Упражнение 5.1.1(a) S -' 0S1 01 Упражнение 5.1.1 (b) S A B C D E - - - - . . . . . CD aa epsilon bBc E. cD aCb E. aa cd epsilon b To understand how this grammar works, observe the following: D generates zero or more c's. E generates one or more b's. B first generates an equal amount of b's and c's, then produces one or more b's (via E) or one or more c's (via cD). That is, B generates strings in b'c with an unequal number b's and c's. Similarly, C generates unequal numbers a's, then b's. Thus, AB generates lines in a'b'c with an unequal number b's and c's, while the CD generates strings in a'b'c with an unequal number of a's and b's. Exercise 5.1.2 (a) Left: S e A1B 0A1B 00A1B 001B 001B 0010B 00101B 00101B 00101 Right Biggest: S - A1B - A1 00B - A101B - A101 - 0A101 - 00A101 - Exercise 5.1.5 S - S SS (S) 0 1 cm phi e The idea is that these productions for S allow any expression to be, Accordingly, amount (union) of two expressions, the connotation of two expressions, expression, expression brackets or one of the four main expressions: 0, 1, phi and epsilon. Solutions for Section 5.2 Exercise 5.2.1 (a) / A / 0 A / 0 A e S No 1 B / 0 B / 1 B e In the aforementioned tree e means Epsilon. Solutions for Section 5.3 Exercise 5.3.2 B - BB (B) (BBC) Epsilon Exercise 5.3.4 (a) Changing production (5) to: ListItem - Doc Solutions for Section 5.4 Exercise 5.4.1 Here's a/A S -1 - S b S e S / / S s s e e Two left derivatives are: S aS aS asbS - aabS - aabs and S aSbS - aaSbS - aabS aabS. The two right derivatives are: S aS asbS - aaSb - aab and S aSbS - aSb asb aaSb aaSb aab. Exercise 5.4.3 The idea is to introduce another non-terminal T that cannot generate an unbalanced A. This strategy conforms to the usual rule in programming languages that 'else' is associated with the nearest previous, unsurpassed 'then' . Here we make b match the previous unsurpassed. Grammar: S -AS aTbS Epsilon T - aTbT Epsilon Exercise 5.4.6 Alas, it is not. We must have three non-terminals corresponding to three possible strong expressions: 1. Factor cannot be broken by any operator. These are basic expressions, brackets of expression, and these expressions follow one or more s. 2. The term can only be broken on K. For example, consider 01, where 0 and 1 are shaken, but if we follow it from the side, it becomes 0 (1), and the concatenation has been 'broken'. 3. The expression may be broken by concataten or q, but not by q. An example may be the expression 0'1. Please note that if we co-believe (say) 1 or follow the word, we will clarify the expression 0 (11) or 0 (1), and in any case the union has been broken. Grammar: E - T- | F-I Fz (E) 0 y 1 cm phi e Introduction to the theory of automation, languages, and computational solutions for Chapter 6 Solutions for Section 6.1 Exercise 6.1.1 (a) In the following, e means Epsilon, empty line, and means the initial symbol Z_0. (q,01,) - (q,1,X) -- (q,e,X) -- (p,e,q) -- (p,1,) -- (p,e,e) Solutions for Section 6.2 Exercise 6.2.1 (a) We will take an empty stack. The X symbol will be used to count 0 at the entrance. In the state q, the state of the beginning where we saw the No.1, we add X to the stack for every 0 seen. The first X replaces the Z_0, the launch symbol. When we see one, we go to the p state and then only pop the stack, one X for each input 1. Formally, the PDA is (q,p,0,1), X,Z_0, delta, q.Z_0). Rules: 1. 2. 3. 4. delta (q.0.Z_0) - Delta (q.0,X) - delta (q.0,X) - delta (q,XX) delta (q.1,X) - delta (p,epsilon) Start with the beginning of the q0 state, from the beginning of the Z_0 symbol, and immediately guess whether to check on: 1. i'j'0 (state1). 2. i'j'0 (State q2). 3. j'k (state q3). We will accept the final state; as smaller, q1 and q3 adoption positions. Rules and their explanations (again, Z_0 e means Epsilon Z_0 Z_0 Z_0): delta (q1,c,Z_0 Z_0) In case (1), we assume that there is no a's or b's, and we consume all c's. State number one will be one of our host states. Delta (q2,a,Z_0 XZ_0) These rules start the case (2). We use X to calculate the number of read from input while remaining in q2. delta (q2,b,X) - Delta (q4,b,X) When b's seen, we go to the position of q4 and pop x vs. b's. Delta (q4,e,e,Z_0 Z_0) If we reach the bottom stack of the marker in the state q4, we have seen an equal amount of a's and b's. We go spontaneously to the state of q1, which will take and consume all c's, continuing to take. delta (q3,a,Z_0 Z_0) This rule begins the case (3). We consume everything from input. Since j'k-0 is possible, the q3 state should be a state of acceptance. Delta (q3,b,Z_0 XZ_0) When b's arrives, we start counting them and go to the state q5, which is not a welcome state. delta (q5,b,X) We continue to count B's. Delta (q5,c,X) - Delta (q6.c,X) When c's arrives, we go to the state q6 and match c's against b's. Delta (q6,e,Z_0) When the marker of the bottom stack is exposed in the state q6, we saw an equal amount of b's and c's. We spontaneously take a state of q3, but we pop the stack so we can't take after reading more a's. Exercise 6.2.4 Introduction of the new state q, which becomes the original state. When entering the Epsilon and the starting P symbol, the new PDA has the choice of popping the stack (thus taking epsilon), or going into the state of the beginning of P. Exercise 6.2.5 (a) We again use e represent epsilon. (q0,bab,Z_0) -- (q2,ab,BZ_0) -- (q3,b,Z_0) -- (q1,b,AZ_0 Z_0 Z_0) (f,e,e) Exercise 6.2.8 Suppose there is a rule that (p,X1X2... Xk) is a choice in the delta (q, a, q). We create k-2 new states r1,r2,...,r'k-2 that mimic this rule but do so by adding one character at a time to the stack. That is to replace (p,X1X2... Xk) in the rule (r'k-2',X'k-1'Xk. Then create new delta rules (r'k-2,e,X'k-1) - (r'k-1), X'k-2'X'K-1) and so on, Up to Delta (r2,e,X3) - (r1,X2X3) and Delta (r1.X2) Solutions for Section 6.3 Exercise 6.3.1 (z.0.1), 0.1,A,S,delta,q,s), where the delta is determined: 1. 2. 3. 4. delta (q,e,S) - q(q,0S1), (q,A) delta (q,e,A) - q(q,1A0), (q,S), (q,e) delta (q.0.0) Exercise 6.3.3 In the next, S is the starting symbol, e means empty line, and q is used instead of Z_0. 1. S-s.s.s.s. (q'p) The next four productions come from the rule (1). 2. 3. 4. 5. qq - 1'qxq'q'q'-1'q'p'p' (q'p'p) 1 'qXq'q'p' (q'p) - 1 'qXp'p'p' The next four productions come from the rule (2). 6. 7. 8. 9. qXq - 1'qXq'qxq' qXq' qXq - 1 'qXp'pXq' qXp - 1 'qXp' - 1 'qXp'p'pXp) The next two productions come from the rule (3). 10. «qXq» -» 0'pXq 11. qXp - 0 pXp The next production comes from rule (4). 12. qXq - e The next production comes from the rule (5). 13. pXp - 1 The next two productions come from the rule (6). 14. (p'q) - 0'qq 15. Exercise 6.3.6 Converting P to CFG and then converting CFG into PDA using two designs data in section 6.3. The result is a one-state CPC equivalent to P. Solutions for Section 6.4 Exercise 6.4.1 (b) Not DPDA. For example, rules (3) and (4) give a choice when in a q state, with 1 as the next entry symbol, and with X at the top of the stack, either using 1 (not making any other changes) or making a move on an epsilon input that pops up the stack and is going to state p. Exercise 6.4.3 (a) Suppose DPDA P takes both w and wx empty stack where x is not epsilon (i.e. N(P) has no prefix properties). Then (q_0.wxZ_0) (q,x,epsilon) for some public q, where q_0 and Z_0 are the starting state and symbol of P. It is not possible that (q,x, epsilon) (p, epsilon, epsilon) for some state p, because we know x not epsilon, and KPC can not move with the blank. This observation contradicts the assumption that wx is in N(P). Exercise 6.4.3 (c) Change P' as follows to create a DPDA P: 1. Add a new start state and a new start symbol. P, with this state and symbol, pushes the starting P' symbol over the stack and moves to the state of the beginning of the P'. The purpose of the new launch symbol is to make sure that P doesn't accidentally take an empty stack. 2. Add a new 'appear state' to P. In this state, P appears the symbol he sees on the stack using epsilon input. 3. If P' enters the adoption state, P enters the popping state instead. As long as L (P') has a prefix property, any line that P' accepts is the final state, P will take an empty stack. Introduction to machine theory, languages and computational solutions for Chapter 7 Revised 3/11/01. Solutions for Section 7.1 Exercise 7.1.1 A and C are clearly generating, as they have productions with terminals of bodies. Then we can discover the S generates because of the production of S-CA, whose body is made up of only the characters that generate. However, B does not generate. Excluding B, leaves grammar S -' CA A - a C -' b Since S, A and C are each achievable from S, all other characters are useful, and above grammar is the answer to the question. Exercise 7.1.2 Revised 6/27/02. a) Only S is invalid, so we have to choose at every point where the S occurs in the body to eliminate it or not. Since there is no body that consists only of S, we should not refer to the rule about not eliminating the whole body. Received grammar: S - ASB AB A- AAS AA A B - SbS bS Sb b n A bb b) The only unit of production is B - A. Thus, it is enough to replace this organ A with the bodies of all A-productions. Result: S - ASB AB AA AA AA B - SbS bS Sb b-n aas a a bb c) Refer to those that A and B receive terminal lines, and therefore, as well as S. So, there are no useless symbols. d) Introduce variables and stagings C - a and D - b, and use new variables in all bodies that are not a single terminal: S - ASB AB A - CAS KA A B - SDS DS B B QA B - A D - A D - b Finally, there are bodies 3; one of them, CAS, appears twice. Introduce new E, F and G variables, to divide these bodies, which gives the grammar of CNF: S A B C D E F - - - - - - . . . . . AB CF CA SG DS SD b-kF KA a DD b SB AS DS Exercise 7.1.10 It's impossible. The reason is that the light induction by the number of steps in the output shows that each directional shape has an odd length. Thus, it is impossible to find such grammar for the language as simple as {00}. To understand why, let's say we start by launching the S symbol and try to choose the first production. If we choose a production with one terminal as a body, we get a line of lengths of 1 and done. If we choose a body with three variables, then since the variable can't get the epsilon, we have to get a string of lengths of 3 or more. Exercise 7.1.11 (b) Statement about the entire design can be a little tricky since you need to use the construction part (c) in although we do not publish the solution (c). Design for b) by induction on I, but it should be a stronger statement that if A_i-production has a body starting with A_j, J j we use part (c) to eliminate the possibility that i'j). Base: For i No 1, we simply apply the design (c) for i No. 1. Induction: If there is a production form of A_i - A_1 ... use construction (a) to replace A_1. This gives us a situation where all A_i production facilities start at least A_2 or terminal. Similarly, replace A_2 use (a) to A_3 the lowest possible A_i production variable. Thus, we ultimately ensure that the body of each A_i production either starts with the terminal or with A_j, for some J and i. Using the design from (c) excludes the possibility that i j. Exercise 7.1.11 (d) According to the hint, we do reverse induction on i that the body A_i can be made to start with the terminals. Base: For i q k, there is nothing to do, since there are no variables with an index higher than k to start the body. Induction: Suppose that the approval for indices is greater than i. If A_i starts with a variable, it should be A_j for some j and i. According to the induction hypothesis, all A_jproductions have bodies starting with terminals now. Thus, we can use the design (a) to replace the original A_j, second only to A_i whose bodies start from the terminals. After fixing all the A_i productions for all I am, it's time to work on B_i-products. Since they have organs that start with terminals or A_j for some j, and the latter variables have only bodies that start with terminals, the application of construction (a) fixes B_j's. Solutions for Section 7.2 Exercise 7.2.1 (a) Let n be a constant pumping lemma and consider the line z We can write z q uvwxy, where v and x can be pumped up, and vwx B.C. is the production of G, then in the new grammar we have - B.C., A - B- B. If A - a is the production of G, then the new grammar has A-a, A' - a, and A' - epsilon. Exercise 7.3.1 (b) The design is similar to the part design (a), but now A' must be designed to generate the W string, if and only if A generates va. That is, the language A is the result of applying /a to the language A. If G has production A - y BC, then the new grammar has A - BC and A' - B.C. If g has A -- b for some b! a, the new grammar has A-b, but we don't add any production for A'. If G has A -A, the new grammar has A-a and A' - epsilon. Exercise 7.3.3 (a) Consider L and A'ib'jc i AcB. Then the solution for the PCP instance gives the string off so that the GBR generated and yR generated. Thus, G generates a palindrome ycyR. However, any palindrome generated by G must have a C in the middle and thus implies a PCP instance solution, i.e. a y line that appears in L (GA), while yR appears in L (GBR) hence, y appears in L (GB). Solutions for Section 10.1 Exercise 10.1.1 (a) MWST MWST then be a line of 1 to 2 to 3 to 4. Exercise 10.1.3 For each P problem in the NP there would be some polynomial p, which is limited by the opening time (and therefore the duration of the exit) reduction from P to NP-full problem in question. That would mean an algorithm for P, which ran in time O (p(n) The first term, p/n, can be neglected. exhibitor k magazine n for some permanent k. Also, p(n)k is no more nk ' for some other permanent k'. So we can say that there would be some permanent C such that the P problem can be solved in time O (nc log2n). ,A,B), build the G1 and G2 to be G, with the beginning of the symbols A and B, respectively. Since this conversion essentially copies, it can be performed in linear time, and therefore is certainly a reduction in polynomial time. Exercise 10.1.5 (c) Absolutely nothing! Part of any evidence of NP-completeness is the part that shows the problem to be in the NP. These problems are, in fact, irrefutable, and therefore certainly not in the NP. Exercise 10.1.6 (b) Test membership in one language and then, if you don't enter the first, test for membership in the second. The faded time does not exceed the amount of time taken to recognize each language. Since both are ar in P, each of them can be recognized in polynomial time, and the amount of polynomial is polynomial. Thus, their union is in P. Exercise 10.1.6 (c) Let L1 and L2 be the languages in P, and assume that we want to recognize their concatetion. Suppose we are given a n. Length entry. For each I between 1 and n-1, check whether positions 1 through I holds the cord in L1 and positions i'1 through n hold the cord in L2. If so, accept; entry to L1L2. If the test fails for all i, reject input. Время времени времени времени времени времени времени времени времени работы распознателей для L1 и L2. Since the latter are both polynomials, so the running time for TM is just described. Exercise 10.1.6 (f) Given the polynomial time OF TM M for L, we can change the M to accept the L supplement as follows: 1. Make each acceptable M condition an unaccepted state from which there are no moves. So if M accepts, the new TM will stop without taking. 2. Create a new q state, which is the only state of adoption in the new TM. For each stateymbol combination that has no stroke, the new TM enters the q state, after which it takes and stops. Solutions for Section 10.2 Exercise 10.2.1 (a) Choice x - y z 1 makes the expression enjoyable. So the expression is in the SAT. Exercise 10.2.2 (a) There are actually only three different Hamilton schemes once we take into account the differences in direction and differences in starting point. These three chains (1,2,3,4), (1,3,2,4) and (1,3,4,2). We can existence of one of these three schemes (using simplified notation section 10.3) by: x12x23x34x14 and x13x23x24x14 and x13x34x24x12. x13x34x24x12. For section 10.3 Exercise 10.3.1 (a) The next, -x means x-bar, supplement x. Let's start with using the design to put it in the CNF. xy is already a product of reservations (x)(y), and x'z is a product of reservations (z). When we use the OR design to combine these, we get (x'u) ('-x'u))(z'-u). Now, to put this expression in 3-CNF, we only have to expand the four positions, each of which has only two letters of the word, by introducing four new variables and doubling the number of reservations. Result: (x'u'v1) (x'u'-v1) (y'u'v2) (y'u'-v2) (x-u-u-v3) (yu-u-v4). Exercise 10.3.3 (a) It is satisfying. Let any two variables be assigned to TRUE, say x1 and x2, and let the other two variables be assigned to FALSE. Then in any set of three variables there should be at least one true and at least one false. Thus, none of the provisions can be false. Solutions for Section 10.4 Exercise 10.4.1 For Part (a): There are triangles (3-clicks) such as 1,2,3. However, there is no 4-click, as there are only 4 nodes, and one edge is missing. So to No. 3 is the answer. For part (b): All pairs of nodes should have an advantage between them, and the number of pairs of k knots chooses 2, or k(k-1)/2. For our part

(c): We will reduce NC to CLIC as follows. Suppose we are given a copy (G,k) of NC. Build a copy (G',n-k) of CLIS, where n is the total number of G knots, and G' with a set of added edges; that is, G' has the advantage (u,v) if and only if G does not have this edge. We have to show that the G has a K-size cover node if and only if G' has a n-k size click. First, let the C be a node cover of G size k. We claim that C's, supplement nodes in C, is a clique in the G' size n-k. Of course, C' has a n-k size. Suppose it's not a click. Then there are a couple of knots (u,v) that don't have an edge in the G'. Thus, this edge is in G. But neither u nor v is in C, which contradicts the assumption that is the node cover. Conversely, if C' is a n-k size clique in G', then we claim that C supplement C', is a C size node in G. The argument is similar: if (u,v) is the G edge not covered by C, then you and V are in C', but the edge (u,v) is not in G', contrary to the assumption that C' is a click. Exercise 10.4.2 For each position, we add one node, and connect it so that it can only be painted in one of the no.1 colors available if the position is made true. Suppose the clause consists of letters with xi, xj, and xk variables, perhaps nullified. The node for the reservation is connected to: 1. xm for all m 0, 1,..., n, except i, j and k. So the only possible colors for the nodes are those that are used for it literal. 2. If the literal with xi is positive (not Connect the reservation node to the xi node. If the literal is nullified, plug in the for the si bar. 3. Connect to xj and xk nodes, similarly. Now, if at least one of the clause letters is the correct purpose, where the c0 color corresponds to the truth, then this literal will not be colored for its variable, and we can use that color for the reservation node. However, if the purpose of the truth makes all the three-letter words false, then the reservation node is connected to the nodes of all n'1 colors, and we cannot complete the coloring. Thus, the coloration of the full graphics colors No.1 is possible, if and only if there is a satisfying purpose of truth for the expression 3-CNF. Exercise 10.4.3 (a) Yes; Hamilton's chain can be found by circling the inner circle, say, 11 to 20, clockwise, then up to 10, around the outer circle counterclockwise, to 1, and then back to 11. Exercise 10.4.4(f) Let (G,k) be an example of a click problem, and let's assume G has n nodes. We produce an example of a half-clique problem, as follows: 1. If k n/2, just produce G. Please note that G has half a click if and only if it has a size-to-click in this case. 2. If k n/2, add 2k - n insulated nodes (nodes without incident edges). The resulting graph has half a click (the size of which should be (n q (2k-n))/2 q 2k, if and only if G has a k. 3. if k n/2, add n - 2k nodes, and connect them in every way possible to each other and to the original G nodes. Conversely, if the new graph has half a click, it should include at least (n-k) - (n-2k) k G graph nodes, implying that G has a k size clique. , since the number of new nodes and edges, at least square the original number of nodes, and the rules for adding nodes and edges are easy to hold. Exercise 10.4.5 (a) After the tip, select any x node on the G chart. Then add new nodes that are adjacent to x and y respectively, and no other nodes. Call the received G'. We argue G'does Hamilton's way if and only if G has a Hamilton scheme. If G has a Hamilton scheme, then Hamilton's next path is in G': start with u, go to x, follow Hamilton's track, but finish on y instead of x, and then go to v. If G' has a Hamilton path, it should start with you and end with v, or vice versa (which is actually the same path. Hamilton's way. Thus, the problem of Hamilton's path boils down to the question of The graph has a covering tree with only 2 knots of leaves. Of course, then Hamilton's path comes down to a more general problem, stated in the question where the number of knots of leaves is the parameter of the problem. Solutions for Section 11.1 Exercise 11.1.1 (a) The problem is the NP. We only need to check if the expression is correct when all variables are correct (polynomial time, deterministic step) and then guess and check some other destinations. Note that if the expression is not true when all variables are correct, then it is certainly not in TRUESAT. The TRUE-SAT supplement consists of all inputs that are not well-formed expressions, inputs that are well-formed expressions, but which are false when all variables are correct, and well-formed expressions that are true only when all variables are correct. We will show the TRUE-SAT is NP-full, so it is unlikely that the supplement is in the NP. To show TRUE-SAT is NP-full, we reduce the SAT to it. Suppose we are given an E expression with x1, x2,..., xn variables. E to E' conversion as follows: 1. First, check if E is true when all variables are correct. If so, we know E is satisfied and therefore convert it into a certain expression x'y that we know is in TRUE-SAT. 2. Otherwise, let E' and E and x1x2... xn, of course, reducing polynomial time. Of course, E' is true when all variables are correct. If E is in the SAT, then it is satisfied with some purpose of the truth of others all-true, because we checked everything true and found E to be false. Thus, E' is in TRUESAT. Conversely, if E' is in TRUE-SAT, then with x1x2... xn true only for an all-true destination, E should be enjoyable. Exercise 11.1.2 There are three things to show. The language is in the NP, in the co-NP, and not in P. 1. To show the language in NP, guess z, calculate f(z) deterministically at polynomial time, and check whether f (z) x. When the guess z is correct, we have f-1 (x). Compare it to y, and take a pair (x,y) if z q y. 2. To show the language to be in the co-NP, we must show the supplement --- a set of inputs that are not a form (x,y), where f -1 (x) - y, is in the NP. It's easy to check for poorly formed inputs, so the hard part is checking whether f-1 (x) is. However, the trick with the part (1) works. Guess z, calculate f(z), check if f (z) x, and then check if z q. If both tests are done, then we have established that f -1 (x) - y, so (x,y) is in the add-on language. 3. Finally, we have to show that the language is not in. We can show that if it was in P, then with n tests for membership in the language, we could binary search to find the exact value of f -1 (x). If one test takes which is polynomial in n, then n times that amount is also polynomial in n. Start by testing the vapor (x,2n-1), i.e. a rough middle point in the range of n-bit whole. If the answer is yes, the next test (x,2n-2); If the answer is no, the test (x,3'2n-2) is next. So we can install one bit a bit f -1 (x) with each test, and after n tests, we know f-1 (x) for sure. Solutions for Section 11.3 Exercise 11.3.2 Suppose M is a TM with a polynomial space bound p(n), and W is an entrance to M length n. We have to show how to take M and W, and write down, in polynomial time, the regular expression E, which is Sigma, if and only if M does not take the f. Technically this design reduces L (M) to the kit that is in question that is to a set of regular expressions that are not equivalent to Sigma. However, the easy consequence of the theorem 11.4 is that because determinant, polynomial time TM can be done to stop, PS closes under the supplement; simply change the acceptance of states to cease, but the non-acceptance of states, add an acceptable state and make every stopped, unaccepted state, moving to this acceptable state instead of immediately stopping. Thus, it can be assumed that M is actually TM to supplement the L language in PS in question. We then actually shorten L to the language of regular expressions equivalent to Sigmay, as suggested. To build a regular E expression, we'll write E and F and H, where the three sub-expressions E, F, and H define sequences of M identifiers that aren't 'start right' and 'finish right', 'respectively' Think of accepting M calculations as a sequence of characters that are the signature of the M ID, each of which is preceed by a special symbol marker. Alphabet Sigma for E will th plus all the tapes and state symbols of M, which we can assume without losing the general kind of disjointed. Each ID is exactly p(n) 1 characters long, since it includes state and exactly p(n) tape symbols, even if many at the end are empty. 1. H: Ends incorrectly. M does not accept if the sequence does not have an adoption ID. Thus, let H (Sigma - F) where the HF is a set of acceptance states of M. 2. F: Starts out wrong. Any line in which the symbols of the first p(n) 2 are not q, q_0 (start state), w, and p(n) - n spaces, is not the beginning of the adoption of calculations, and therefore should be in L/E). We can write F as the amount of terms: Sigma-Sigma, i.e. all lines that don't start with q. Sigma (Sigma-q_0) Sigma, i.e. all lines that don't have q_0 as a second character. i'1 - Sigma (Sigma-a_i) Sigma, where a_i is ith position w. Note Sigmak means Sigma written k times, but this expression takes only polynomial time to write. I Sigma (Sigma-B-Sigma), for all n-3 n introduction to automata theory solution manual pdf. solution manual of introduction to automata theory languages and computation. introduction to automata theory languages solution manual