Tutorialspoint formal language and automata theory pdf

I'm not robot

reCAPTCHA

Continue

The term Automaton comes from the Greek word αὐτόματα, which means self-ifing. Automata in plural is an abstract self-propelled computing device that automatically follows a given sequence of operations. The machine with the final number of states is called Finite Automaton (FA) or Finite State Machine (FSM). The formal definition of the Finite Automaton A machine can be represented by a 5-blunt machine (to, ∑, δ, q0, F), where q is the final set of states. ∑ is the ultimate set of symbols called the machine alphabet. δ is a transitional function. q0 is the initial state in which any input is processed (q0 ∈ q). F is a set of final states/states q (F ⊆ q). Similar terminology Alphabet Definition - the alphabet of any final set of characters. An example ∑ a, b, c, d is an alphabet in which the symbols are letters a, b, c and d. String definition is the ultimate sequence of characters taken from ∑. Example - 'cabcad' is the actual line on the alphabetical set ∑ - a, b, c, d' length of definition of a line - is the number of characters present in a line. (Designated SH). Examples : If S and 'cabcad', SH 6 If it is called an empty line (denoted in ε) Kleene Star Definition - Kleene Star, ∑, is the operator on a set of characters or strings, ∑, which gives an endless set of all possible lines of all possible lengths more ∑ including. The office - ∑ - ∑0 ∪ ∑1 ∪ ∑2 ∪....... where ∑p is a set of all possible lines of length of r. Example: If ∑, a, b, ∑, a, b, aa, ab, ba, bi,........... Kleene Closure / Plus Definition - a set of ∑ - is an endless set of all possible lines of all possible lengths for ∑ except q. Representation - ∑ - ∑1 ∪ ∑2 ∪ ∑3 ∪....... ∑ ∑ - Example - If ∑ - a, b, ∑, a, b, a, ab, ba, bb,........... The definition of language - language is a subset of ∑ for some alphabetical ∑. It can be finite or infinite. Example: If a language takes all possible lines of length 2 within ∑ years, then L and AB, aa, ba, bb - the term Automata comes from the Greek word αὐτόματα, which means self-eph. Automata in plural is an abstract self-propelled computing device that automatically follows a given sequence of operations. The machine with the final number of states is called Finite Automaton (FA) or Finite State Machine (FSM). The formal definition of the Finite Automaton A machine can be represented by a 5-blunt machine (to, ∑, δ, q0, F), where q is the final set of states. ∑ is the ultimate set of symbols called the machine alphabet. δ is a transitional function. q0 is the initial state in which any input is processed (q0 ∈ q). F is a set of final (F ⊆ q). Similar terminology Alphabet Definition - the alphabet of any final set of characters. An example ∑ a, b, c, d is an alphabet in which the symbols are letters a, b, c and d. String definition is the ultimate sequence of characters taken from ∑. Example - 'cabcad' is the actual line on the alphabetical set ∑ - a, b, c, d' length of definition of a line - is the number of characters present in a line. (Designated SH). Examples : If S and 'cabcad', SH 6 If it is called an empty line (denoted in ε) Kleene Star Definition - Kleene Star, ∑, is the operator on a set of characters or strings, ∑, which gives an endless set of all possible lines of all possible lengths more ∑ including. The office - ∑ - ∑0 ∪ ∑1 ∪ ∑2 ∪....... where ∑p is a set of all possible lines of length of r. Example: If ∑, a, b, ∑, a, b, aa, ab, ba, bi,........... Kleene Closure / Plus Definition - a set of ∑ - is an endless set of all possible lines of all possible lengths for ∑ except q. Representation - ∑ - ∑1 ∪ ∑2 ∪ ∑3 ∪....... ∑ ∑ - Example - If ∑ - a, b, ∑, a, b, a, ab, ba, bb,........... The definition of language - language is a subset of ∑ for some alphabetical ∑. It can be finite or infinite. Example: If a language takes all possible lines of length 2 within ∑ - a, b, then L and AB, aa, bb - Deterministic Finite Automaton Finite Automaton can be classified into two types - The Deterministic End Machine (DFA) Undeminated Finite Automaton (NDFA/NFA) Deterministic Finite Automaton (DFA) Hence, it is called a deterministic automaton. Since it has a finite number of states, the machine is called a deterministic end machine or a deterministic Finite Automaton. The formal definition of DFA A DFA can be represented by a 5-tuple (δ ∑ yap. ∑ is the final set of characters called the alphabet. δ is a transitional function, δ: q0 × ∑ → q0 is the initial state from which any input is processed (q0 ∈ q). F is a set of final states/states q (F ⊆ q). The DFA A DFA graphic view is represented by digraphs called state diagrams. Verticals represent states. The arcs marked with the input alphabet show transitions. The original state is marked by an empty single arc. The final state is indicated in double circles. Example Let the deterministic end machine be → - z, b, c, ∑ - 0, 1, q0, F and the transition function δ as shown in the following table - Present State Next State for Entering 0 Next State for Entering 1 a b b c c With His graphic view will be as follows: Non-dedeint Finite Automaton In NDFA, for a specific input symbol, the machine can move to any combination of states in the machine. In other words, the exact state in which the machine is moving cannot be determined. Therefore, it is called an indefinable machine. Because it has a finite number of states, the machine is called the indefinable Ultimate Machine or the Indefinable Ultimate Automaton. The formal definition of NDFA An NDFA can be presented 5-blunt (to, ∑, δ, q0, F), where q is the final set of states. ∑ is the ultimate set of symbols called alphabets. δ is a transitional function in which δ: ∑ → 2 (here was taken a power kit (2) because in the case of NDFA, from a state, the transition can occur to any combination of q0 states, is the initial state from which any input is processed (q0 ∈). F is a set of final states/states q (F ⊆ q). NDFA Graphic View: (just like DFA) NDFA is represented by digraphs called state diagrams. Verticals represent states. The arcs marked with the input alphabet show transitions. The original state is marked by an empty single arc. The final state is indicated in double circles. Example Let the indefinable end machine → and b, c' ∑ - 0, 1 q0 - F - C- Transition function δ as shown below - Present State Next state for entering 0 Next state for entering 1 a, b b c, c b, c c, His graphic view will be as follows: DFA vs. NDFA The following table lists the differences between DFA and NDFA. DFA NDFA Transition from state to state to one specific following state for each input character. That's why they call it deterministic. The transition from the state can be in several states for each type character. That's why they call it indefinable. Empty line transitions are not visible in DFA. The NDFA allows empty line transitions. Rollback is allowed in DFA In NDFA, rollback is not always possible. More space is needed. Less space is needed. The line is accepted by the DFA if it transits to the final state. The line is accepted by the NDFA if at least one of all possible transitions ends in final condition. Receivers, classifiers and Transducers Acceptor (Recognizer) The Boolean-calculated automaton is called the Receiver. All receiving States either accept or reject the material. Classifier A has more than two finite states and gives one yield when it is terminated. A transducer machine that produces outputs based on current input and/or a previous state is called a predo. Predictors can be of two types - Mealy Machine - output depends on both the current state and the current input. Moore machine - Exit depends only on the current state. DFA and NDFA A acceptability accepted by DFA/NDFA iff DFA/NDFA in the original state ends in the state of acceptance (any of the final states) after reading the line in full. Line S is accepted by DFA/NDFA (I, ∑, δ, q0, F), iff δ (q0, S) ∈ F L language adopted by DFA/NDFA, is S S ∈ ∑ and δ (q0, S) ∈ F' row S' not accepted DFA/NDFA (I, ∑, δ, q0, F), iff δ (q0, S) ∉ F L language not adopted by DFA/NDFA (addition of adopted language L) is the S example of the ∈ ∑ S δ (q0, S) ∉ F' D CONSIDERFA, shown in Figure 1.3. You can get acceptable lines from DFA. Lines taken above DFA: 0, 00, 11, 010, 101, ... Lines not taken above DFA: No. 1, 011, 111, ... NDFA to DFA conversion Let X (x, ∑, qx, q0, Fx) be the NDFA that adopts the language of L(X). We have to develop the equivalent of DFA Y (zi, ∑, yu, q0, Fy) in a way that L (Y) and L (X). The following procedure converts the NDFA into the equivalent of DFA - Input Algorithm - NDFA Exit - equivalent to DFA Step 1 - Creating a state table from a given NDFA. Step 2 - Create an empty state table under possible input alphabets for the DFA equivalent. Step 3 - Separate the DFA start state by q0 (the same as the NDFA). Step 4 - Learn a combination of states (0, No.1,... , for each possible alphabet input. Step 5 - Every time we generate a new DFA state under the alphabet input columns, we must apply step 4 again, otherwise go to step 6. Step 6 - States, which contain any of the final states of the NDFA are the final states of the equivalent of DFA. Example Consider the NDFA shown below. q δ (q.0) δ (q.1) (a,b,c,d,e) d,e (a,b,c,d,e) a,b,e a,b,c,d,e ∅ ∅ (b) ,d,e) d,e ∅ b,d,e e. ∅ (b) (b) (b) (b) (b) ∅ (b) DFA State Chart is as follows: DFA Minimization Input - DFA Exit - Minimized DFA Step 1 - Draw a table for all pairs of states (Ci, th) not necessarily connected directly (All unmarked initially) Step 2th) in DFA, where qi ∈ F and th ∉ F or vice versa and mark them. Here's f - this is a set of final states Step 3 - Repeat this step until we can mark more states if there is an unmarked pair (qi, th), mark it if the pair δ (qi, A), δ (Ci, A) is marked for some input alphabet. Step 4 - Combine all unmarked pairs (Ci, ij) and make them a single state in a reduced DFA. Example Let's use algorithm 2 to minimize the DFA shown below. Step 1 - We draw a table for all pairs of states. Step 2 - We celebrate state couples. b c d e f b c ✔ ✔ d ✔ ✔ e ✔ ✔ f ✔ ✔ ✔ Step 3 - We will try to mark the state couples, green colored check sign, transit. If we enter one to learn 'a' and 'f', it will go to learn 'c' and 'f' respectively. c, f) already tagged, so we'll celebrate a couple (a, f). Now we enter 1 to learn 'b' and 'f'; it will go to the state 'd' and 'f' respectively. (d, f) already tagged, so we'll celebrate a couple (b, f). b c d e f a b c ✔ ✔ d ✔ ✔ e ✔ ✔ f ✔ ✔ ✔ ✔ ✔ After Step 3, we have combinations of states that are not marked. We can recombinate the C, d, e'd, e' in c, d, e From here we got two combined states like a, b and c, d, e So the final minimized DFA will contain three states,f, a, a, b and c, d, e DFA Minimization using the Equivalency Theorem If X and Y are two states in DFA, we can combine these two states in DFA , we can combine these two states in DFA, we can combine these two states in DFA, we can combine these two states, IF they are not discernible. Two states are distinguishable if there is at least one S line, so one of the δ (X, S) and δ (Y, S) takes and the other does not take. Thus, the DFA is minimal if and only if all states are distinguishable. Algorithm 3 Step 1 - All states are divided into two sections - final and unfinished states and designated P0. All states in the section are the 0th equivalent. Take the K counter and initiate it with 0. Step 2 - Increment to 1. For each section in Pk, divide the states into Pk into two sections if they are k-distinguishable. The two states in this section are X and Y k-different if there is an input S, so δ (X, S) and δ (Y, S) are (k-1)-different. Step 3 - If Pk ≠ Pk-1, repeat step 2, otherwise go to step 4. Step 4 - Combine equivalent kth kits and make them new states of reduced DFA. Example Consider the following DFA q δ (q.0) δ (q.1) a b b a d a d e e e e f f f F Let's take the above algorithm to the aforementioned DFA - P0 (c,d,e), (a,b,f) P1 (c,d,e), (a,b) (e), (f) (a,b), (a,b), (a,b), (a,b),b.e. (a,b),(a,b),b, (a,b), (a,b),b.e. (a,b) (a,b), (a,b), (a,b) (a,b), (a,b), (a,b) (a,b), (a,b), (a,b) (a,b), (a,b), (a,b), (a,b), (a,b), (a,b), (a,b,e), (a,b), (a,b) (e,) (a,b) (a,b) (a,b) (a,b) (a,b) (e,) (a,b) (a,b) (a,b),00 P1 and P2. There are three states in the abbreviated DFA. The decrease in DFA is as follows: δ (q.0) δ (q.1) (a, b) (a, b) (c,d,e) (c,d,e) (c,d,e) (f) (f) (f) (f) Moore and Mealy Machines Finite automata may have exits corresponding to each transition. There are two types of state-owned machines that generate power output - the Mealy Machine Moore Mealy Machine A Mealy Machine is an FSM whose output depends on the current state as well as present input. This can be described as 6 tuple (I, ∑, O, δ, X, q0), where q is the final set of states. ∑ is the final set of symbols called the introductory alphabet. O is the final set of characters called the output alphabet. δ the input transition function where δ: × ∑ → is a function of going to the exit where × ∑ → O q0 is the initial state from which any input is processed (q0 ∈). Below is a map of the state of the Mealy Machine State Next State Input No. 0 input - 1 state output of state production → b x1 c x1 b x2 d x3 c x3 c x1 d x x3 d x2 Mealy State Chart - this Is Moore's Machine - is the FSM, the results of which depend only on the current state. Moore's machine can be described as 6 tuple (I, ∑, O, δ, X, q0), where q is the final set of states. ∑ is the final set of symbols called the introductory alphabet. O is the final set of characters called the output alphabet. δ the input transition function where δ: × ∑ → q X is a output transition function where X: No → O q0 is the initial state from which any input is processed (q0 ∈). Moore's Machine Status Table is shown below - Present State Next State Exit Entry No. 0 Entry No. 1 → b x2 b x1 c c d x2 d d d x3 State Chart above Moore Machine - Mealy Machine vs. Moore Machine Next Table highlights points that differentiate the Mealy machine from Moore's machine. Mealy Machine Moore Machine Output depends on both the current state and the current input. Output depends only on the current state. Typically, it has fewer states than The Mealy Machine. The value of the output function depends on transitions and changes when the input logic is made in the present state. The output function is a function of the current state and changes around the edges of the watch whenever the state changes. The power of the machine reacts faster to the entrances. They usually react in the same hours cycle. Moore machines require more logic to decode output, leading to increased chain delays. They usually react one hour later. Moore Machine to Mealy Machine Algorithm 4 Input - Moore Machine Output - Mealy Machine Step 1 - Take the empty format of the Mealy Machine transition table. Step 2 - Copy all Moore Machine transition states to this table format. Step 3 - Check the present status and relevant outputs in the Moore Machine status table; if for the state of qi output m, copy it to the weekend columns of the Table of The State Of Mealy Machine, where qi appears in the following state. Example Consider Moore's next machine - Present State Next State Exit No 0 a 1 → a d b 1 b d 0 c c 0 d b a 1 Now we use Algorithm 4 to convert it to Mealy Machine. Step 1 - 2 - Present State Next State A No 0 A - 1 State Weekend State Exit → a d b a d c c c c c d a Step 3 - Present State Next State A 0 a 1 State Exit State Exit qgt; d 1 b 0 b 1 d 1 c c 0 c 0 d b 0 a 1 Mealy Machine to Moore Machine Algorithm 5 Input - Mealy Machine Output - Moore Machine Step 1 - Calculate the number of different outputs for each state (qi) that are available in the Mealy. Step 2 - If all qi exits are the same, copy the state of qi. If it has n various outlets, break qi in n states like zin where n q 0, 1, 2....... Step 3 - If the original state is 1, insert a new starting state at the beginning that gives 0 output. Example Let's consider the following Mealy Machine - Present State Next State No. 0 a 1 Next State Exit Next State Exit → d 0 b 1 b 1 d 0 c 1 c 0 d 0 0 1 Here, States 'a' and 'd' give only 1 and 0 exits respectively, so we keep the states 'a' and 'd'. But the states 'b' and 'c' produce different outlets (1 and 0). Thus, we divide B into b0, b1 and c by c0, c1. Present State Next State Exit No 0 a 1 → a d b1 1 b0 a d 0 b1 a d 1 c0 c1 c1 c1 c1 c0 1 d b0 0 Introduction to grammar n literary meaning term, grammar denotes syntactical rules for conversation in natural languages. Linguistics has been trying to define grammars since the creation of natural languages such as English, Sanskrit, Mandarin, etc. Noam Chomsky gave a mathematical model of grammar in 1956, which is effective for writing computer languages. Grammar A Grammar G can be officially written as a 4-tuple (N, T, S, P), where N or VN is a set of variable or non-terminal characters. T ∑ is a set of Terminal symbols. S is a special variable called Start Symbol, S ∈ N P - it's the rules of production of terminals and neve terminals. The production rule is shaped α → β, α and β strings on the VN ∪ ∑ and at least one α symbol belongs to VN. Example G1 Grammar (S, A, B, A, B, S, S, S → AB, A → aA, A → a, B → b)) Here, S, A and B are non-terminal symbols; a and b are symbols of Terminal S is a symbol of Start, S ∈ N Productions, P : S → AB, A → a, B → b G2 Grammar ((S, A, A, BH, S, → aAb, aA → aaAb, A → ε) Here S and A are non-contour symbols ε. S is a symbol of start, S ∈ N Production P : S → aAb, aA → aaAb, A → ε Derivations from a Grammar Strings can be obtained from other lines using productions in grammar. If Grammar G has a production α → β, we can say that x α y gets x β y in G. This conclusion is written as x α y ⇒G x β y Sample Consider Grammar S → aAb, aA → aaAb, A → ε) Some lines that can be obtained are S ⇒ aAb using the production of S → aab ⇒ aaab using aA → aaab aab ⇒ aaaAbbb using aa → aab aab ⇒ aabab using A → ε Language-generated grammar set of all lines that can derive from grammar is said to be a language generated from this grammar. The language generated by grammar G is a subset formally defined by L(G) W ∈ ∑, S ⇒G BS If L (G1) - L (G2), грамматике G2. Пример Если есть грамматика G: N, A, A, B'T, b' P и S → AB, A → a, B → b' Here S производит AB, и мы можем заменить A a, и B b. Здесь единственной принятой строкой является ab, т.е. L(G) - пример ab Предположим, что у нас есть следующая грамматика: G: N, A, B b' P - S → AB , A → aA'a, B → bB'b Язык, генерируемый этой грамматикой - L(G) ≥ ≥ - ab, a2b, ab2, a2b2, ... производит эти языки. Пример Проблемы - Предположим, L (G) - am bn m ≥ 0 и n > 0. Мы должны выяснить грамматику G, которая производит L(G). Решение Since L(G) - am bn m ≥ 0 и n > 0 набор принятых строк может быть переписан как L (G) - b, ab,bb, aab, abb, ...' Здесь, стартовый символ должен взять по крайней мере одно 'b' предшествует любое количество 'a', включая null. Чтобы принять набор строк 3б, АБ, bb, aab, abb, ..., мы взяли постановки - S → aS , S → B, B → b и B → BB S → B → b (Принято) S → B → bB → bb (Принято) S → aS → aB → ab (Принято) S → aS → aaS → aaB → aab (Принято) S → aS → aB → abB → abb (Принято) Таким образом , мы можем доказать, что каждая строка в L(G) принимается языком, генерируемым производственным набором. Таким образом, грамматика G: (S, A, B, a, b, S, S → aS B , B → b bB) Пример Проблема - Предположим, L (G) - am bn m ≥ 0 и n ≥ 0. Мы должны выяснить грамматику G, которая производит L(G). Решение - Так как L (G) - am й м > 0 и n ≥ 0, набор принятых строк может быть переписан как L(G) - a, aa, ab, aaa, aab,abb, ... Чтобы принять набор строк 3a, aa, АБ, ааа, ааб, abb, ..., мы взяли постановки - S → aA, A → aA , A → B, B → bB ,B → S → aA → aB → a→ a (Accepted) S → aA → aaA → aaB → aaa → a (Accepted) S → AA → aB → abB → абе → аб (Принято) S → aA → aaA → aaaA → aaaB → aaa→ aaa (Принято) S → aA → aaA → aaB → aabB → aab' → aab (Accepted) S → aA → aB → abB → abbB → abb' → abb (Accepted) Таким образом, мы можем доказать, что каждая строка в L(G) принимается языком, генерируемым производственным набором. Таким образом, грамматика G: (S, A, B, a, b, S, S → aA, A → aA B, B → - BB в) Хомский Классификация грамматики В соответствии с Ноам Хомоски, Есть четыре типа грамматики - Тип 0, Тип 1, Тип 2, и Тип 3. Следующая таблица показывает, чем они отличаются друг от друга - Грамматика Тип Грамматика Принятый язык Принятый Automaton Тип 0 Неограничено грамматика Рекурсивно enumerable язык Тьюринг машина Тип 1 Контекст чувствительных грамматики Контекст чувствительных Linear Limited Automaton Type 2 Context-Free Grammar Context-Free Language Pushdown Machine Type 3 Regular Grammar Regular Language Ultimate State Machine Take a look at the following illustration. It shows the scale of each type of grammar - Type - 3 Grammar Type-3 grammars generate regular languages. Type-3 grammars must have one non-terminal on the left side and a right side consisting of one terminal or one terminal followed by one non-venel. Productions must be in the form of X → A or X → aY, where X, Y ∈ N (Not Terminal) and ∈ T (Terminal) Rule S → ε allowed if S does not appear on the right side of any rule. Example X → ε X → aY Y → b Type - 2 Grammar Type-2 Grammar generate context of free languages. The productions should be in the form of A → γ, where ∈ N (not terminal) and γ ∈ (T ∪ N) (String terminals and non-terminals). These languages generated by these grammars are recognized as an indefinable push-up machine. Example S → X X → X → aX X → abc X → ε Type - 1 Grammar Type-1 Grammar generate context of sensitive languages. The productions should be in the form of α A β → α γ β, where A ∈ N (Non-terminal) and α, β, γ ∈ (T ∪ N) (Terminals and non-terminals) lines α and β may be empty, but γ should be empty. The S → ε rule is allowed if the S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear limited machine. The example of AB → AbBc A → bcA B → Type B - 0 Grammar Type-0 Grammar generate recursively enumerable languages. Productions have no restrictions. They are any phased grammar structure, including all formal grammars. They generate languages that are recognized by the Turing machine. The productions can be in the form of α → β where the α is a string of terminals and nonterminals, at least one non-terminal and α can not be zero. β is a string of terminals and non-terminals. The example of S → ACaB Bc → acB CB → DB aD → Db Regular Expressions Regular Expression can be redefined as follows: ε is a regular expression indicating a language containing an empty line. (L (ε) - ε) φ a regular expression denoting a blank language. (L (φ) is a regular expression in which L and X, if X is a regular expression, denoting the language L'X and Y, is a regular expression, denoting the language L (Y), then X and Y is a regular expression corresponding to the language L'X) ∪ L(Y), where L (X'Y ∪) X. Y is a regular expression corresponding to the language L.X. L(Y), where L (X.Y) - L (X) . L(Y) R is a regular expression corresponding to L(R), where L (R) - (L(R)) If we apply any of the rules several times from 1 to 5, they are regular expressions. Some RE Examples of Regular Expressions Regular (0 and 10) L No 0, 1, 10, 100, 1000, 10000, ... (0*10*) L No 1, 01, 10, 010, 0010, ... (0 + ε) (1 + ε) L (ε, 0, 1, 01) (a'b) A's and B strings of any length, including zero string. So L ε, a, b, aa, ab, bb, ba, aaa...... So L and Abbot, aabb, babb, aaabb, ababb, ... (11) Set consisting of the nod number 1, including an empty line, thus ε, 11, 1111, 11111, ... (aa) (bb) b a set of strings consisting of an even number a's, followed by the odd number b's, so L q b, aabbb, aabbb, aaaab, aaaabbb, ... (aa-ab-ba-ba-BBC) A and b even length can be obtained by consatening any combination of strings aa, ab, ba and bb, including null, so L-aa, ab, ba, bb, aaab, aaba,................. Regular sets Any set representing the value of a normal expression is called a regular set. Properties of regular property sets 1. Combining two regular sets is a regular one. Proof : Let's take two regular expressions RE1 and a(aa) and RE2 (aa,.....) (Odd length strings, with the exception of Null) and L2 ε, aa, aaa, aaaaa, aaaaaaaaa,....... (Strings of even length, including Null) L1 ∪ L2 - ε a, aa, aaa, aaaa, aaaaa, aaaaaa, aaaaaaaaa,....... (Rows of all possible lengths, including Null) RE (L1 ∪ L2) - aa (which in itself is a regular expression) is thus proven. Real Estate 2. Crossing two regular sets is regular. Proof : Let's take two regular expressions RE1 and a(a) and RE2 (aa) So, L1 - a,aa, aaa, aaaa, ... (Strings of all possible lengths, except Null) L2 - ε, aa, aaaaa, aaaaaaa,....... (Strings of even length, including Null) L1 ∩ L2, aaaaa, aaaaa, aaaaaaaaa,....... (Line of even length, except Null) RE (L1 ∩ L2) - aa (aa), which in itself is a regular expression. Hence, proven. Real Estate 3. The regular recruitment supplement is regular. Proof : Let's take the regular expression - RE (aa) So, L and ε, aaaa, aaaaaa, ... (Strings of even length, including Null) Supplement L are all strings that are not in L. So, L' Za, Aaaaa, ... (Odd length lines, except Null) RE (L') - a (aa) that is a regular expression of yourself. Hence, proven. Real Estate 4. The difference in two regular sets is regular. Proof : Let's take two regular expressions: RE1 (a) and RE2 (aa) So, L1, aa, aaa, aaaa, ... (Strings of all possible lengths, except Null) L2 - ε, aa, aaaaa, aaaaaaa,....... (Strings of even length, including Null) L1 - L2 Za, Aaa, aaaaaaa, aaaaaaaaaaa,..... (Rows of all odd lengths except Null) RE (L1 - L2) - a (aa) which is a regular expression. Hence, proven. Real Estate 5. The reversal of the normal set is regular. Proof - We must prove LR is also regular if L is a regular Пусть, L No 01, 10, 11, 10 10 (L) No 01, 10 x 11, 10 LR - 10, 01, 11, 01 RE (LR) - 01 - 10 - 11 - 10, which is a regular thus proven. Real Estate 6. Closing the usual set is regular. Proof - If L - a, aaa, aaaaa, ... (Odd length strings excluding Null), i.e. RE (L,...............) (All lengths, except Null) RE (L) - a (aa) Hence proven. Real Estate 7. The concate of two regular sets is a regular one. Proof - Let RE1 (0'1) and RE2 - 01 (0'1) - Here, L1 - 0, 00, 10,000, 010, ... (A set of lines ending in 0) and L2 (01, 010 011,.....) (A set of lines starting at 01) Then L1 L2 - 001 0010 0011 0001010 000111 1001 10010,............ A set of lines containing 001 as a sub-construction that can be represented by RE (0 x 1) 001 (0 and 1) is thus proven. Identifiers associated with regular expressions, given R, P, L, As regular expressions, the following identifiers are held ∅ the RRS RRH ε ε ε R - ∅ - ∅ - R (union identification) R ε - ε R (identification for concating) ∅ L and L ∅ - ∅ (concatcation annihilator) R and R (Idempotent law) L (M.I.N) - L.M. L. (Left Distribution Right) (M and N) L y ML th NL (Right Distribution Law) ε - RR , ε - Arden's THEOrem RR In order to learn the regular expression of the end machine, we use Theorem Arden along with the properties of regular expressions. Statement : Let P and q be two regular expressions. If P does not contain a zero line, then R - RP has a unique solution that is R - P- Proof - R - (RP)P (After we put the value R - RP) - R q (ε P and P2 , P3 ....) R - PP How PL presents (ε P - P2 and P3 ....) - Hence, proven. Suggestions for applying the Arden Transition Chart theorem should not have null transitions It should have only one original state Of Step 1 Method - Create equations as the next form for all DFA states having n states with the original state q1. q1 q1R11 q2R21 ... qnRn1 ε q2 q1R12 - q2R2 ... q2R22 ... qnRn2

qn q1R1n q2R2n ... qnRn Rij presents a set of edge marks from qi to qj, if such an edge does not exist, then Rij No ∅ Step - Solve these equations to get the equation for the end state in terms of Rij Problem Build a regular expression corresponding to the machines below : Solution - Here is the initial state and the final state of q1. The equations for the three states q1, q2 and q3 are as follows: q1 q1a q1a q3a ε (ε move is because q1 is the original state0 q1b q2b q3b q3 q3 q2a Теперь, мы решим эти три уравнения: q2 - q1b - q2b - q3b - q1b - q2b (q2a)b (Замена значения q3) ε q1a - q2aa - ε (Замена значения q3) - q1a q1b (b)aa (ε (Замена значения q2) ε ε - q1 (a(b)aa) регулярное выражение: (a q b(b и ab) . Проблема Создать регулярное выражение, соответствующее автомату, приведенное ниже: Решение - Здесь начальное состояние q1, а окончательное состояние q2 Теперь мы записываем уравнения ε - q1 - q10 - q10 , q2, q11, q20 q3, q21, q30 и q31 Now, мы решим эти три уравнения: q1 - 0 Как, R R Таким образом, q1 - 0 q2 01 - q20 Так, q2 - 0'1 (0) (Теорема Ардена) Таким образом, регулярное выражение составляет 0 10. Строительство FA от RE Мы можем использовать строительство Томпсона to learn Finite Automaton from regular expression. We dare regular expression into the smallest regular expressions and convert them to the NFA and finally to the DFA. Some of the main expressions of RA are the following - Case 1 - For the regular expression 'a', we can build the following FA - Case 2 - Case 3 - For regular expression (a'b), we can build the next FA - Case 4 - For regular expression (a'b) , we can build the next FA - Step 1 With Step 2 Remove zero transition from the NFA and convert it to the NFA. The problem of converting the next RA into its equivalent of DFA No. 1 (0 x 1) 0 Solution We will concatenate three expressions 1, (0) and 0 Now we will remove ε transitions. Once ε with conversions from the NDFA, we get the following: This is the NDFA corresponding to RE Nos. 1 (0 and 1) 0. If you want to convert it to DFA, just apply the NDFA conversion method to DFA discussed in Chapter 1. Finite Automata with Null Moves (NFA-ε) The final zero-speed machine (FA-ε) passes not only after entering from the set of the alphabet, but also without any input symbol. This transition without input is called zero movement. NFA-ε is officially represented by 5-tuple (∑, δ, q0, F) consisting of q - the final set of states ∑ - the final set of input symbols δ - the function of transition δ : × ε (∑ ∪)) → 2 q0 - the initial state of q0 ∈ f - the final set of states/states (F⊆). Above (FA-ε) takes a set of rows No. 0, 1, 01 Removing null Moves from Finite Automata If in NDFA, there is ε-movement between the top of X to the top of Y, we can remove it with the following steps - Find all the outgoing edges from Y. Copy all these edges, starting with X without X without Edge labels. If X is the starting state, make Y also the original state. If Y is the final state, make X also the final state. The problem of converting the following NFA-ε nFA without zero move. Solution Step 1 - Here ε between q1 and q2, so let q1 X and qf is Y. Here are the outgoing edges from qf to qf for entries 0 and 1. Step 2 - Now we will copy all these edges from q1 without changing the edges from qf and get the next FA - Step 3 - Here q1 is the initial state, so we make qf also the initial state. So the FA becomes Step 4 - Here qf is the final state, so we make q1 also the final state. So the FA becomes - Pumping Lemma for regular grammar Let L be a regular language. Then there is a constant 'c' such that for each line w in L ≥ c We can break w into 3 cords, w and xyz, such that zgt;0 ≤ c For all k ≥ 0, the xykz line is also found in L. Lemma Pumping Lemma pumping apps must be applied to show that some languages are not regular. It should never be used to show the language is regular. If L is a regular, it satisfies the pumping Lemma. If L doesn't satisfy Pumping Lemma, it's not regular. The method to prove that the L language is not regular at first, we must assume that L is regular. Thus, the pumping lemma should hold for L. Use pumping lemma to get the contradiction - Choose w so that ≥ c Select y so that ≥ 1 Select x is such that ≤ c Assign the remaining line z. Select K so that the received line is not in L. Hence L is not regular. The problem is prove that L and aibi I ≥ 0 is not a regular. Solution - First, we assume that L is regular and n is a number of states. Let it be anbn. So, 2n ≥ n. By pumping lemma, let w and xyz, where ≤ n. Let x th ap, y th ac, and z y arbn, where r y r th, p ≠ 0, q ≠ 0, p ≠ 0. So ≠ 0. Let k No 2. Then xy2z and apa2qarbn. Number as q (p q 2q y r) Since ≠ 0, xy2z is not a form of anbn. Thus, xy2z is not in L. Therefore L is not a regular. DFA Supplement If (to, ∑, δ, q0, F) be the DFA that adopts the L language, then the DFA supplement can be obtained by replacing it with the adoption of states with its hostile states and vice versa. We will take an example and explain it below the ∑ - This DFA adopts the language L yah, aa, aaa, ... Now we will change his acceptance states with its non-attraction states and vice versa, and will get the following - This DFA adopts the language of Ľ , ε b, ab,bb,ba, ... ∑... Context-Free Grammar Introduction Definition - Context Free Grammar (CFG) consisting of the final set of grammar rules is the quadruple (N, T, P, S), where N is a set of non-terminal characters. T is a set of terminals where N ∩ T and NULL. P is a set of rules, P: N → (N ∪ T), i.e. the left side of the production rule P has any right context or left context. S is the starting symbol. Example Grammar (Az, For, B, SD, A), : → AA, A → abc. Grammar (S, a, b, a, b, b, P, S), P: S → aSa, S → bSb, S → ε Grammar (S, F, No0, 1), P: S → 00S 11F, F → 00F ε generation of tree derivatives A or tree parsing is an orderly root tree that graphically presents information derived from free context. The technique of presenting the Root Peak - should be marked with a starting symbol. Vertex - Marked with an unborn symbol. Leaves marked with a terminal symbol or ε. If S → x1x2...... xn is the production rule in the CFG, then the tree disassembly/derivative tree will be as follows: There are two different approaches to draw a tree derivative - Top down Approach - Starts with the starting symbol S descends to the leaves of the trees using the productions from the bottom up Approach - begins with the leaves of the tree Proceeds up to the root, which is the starting symbol of the S Withdrawal or Harvest tree output or exit from the parsley tree is the final line However, if all Null leaves are derived Null. Example Let CFG (N,T,P,S) be N - S, T - a, b, Starting symbol - S, P - S → SSb ε One conclusion from the above CFG is abaabb S → SS → aSbS → abs → abasb → abaaSbb → abaabb Sentential Form and Partial Tree Derivative Partial Tree Derivative is an under-tree tree production/disassemble in such a way that either all of its children are under the back, or none of them is in the undergva. Example If in any CFG productions are S → AB, → aaA ε, B → Bb ε partial tree derivative may be as follows: If a partial tree derivative contains a root S, it is called a sentential form. The aforementioned suspension tree is also in sentential form. Leftmost and Rightmost Derivative Derivative Leftmost - Left derivative is derived by applying production to the left variable at each turn. Right derivative - The most right output is obtained by applying production to the most right variable at every turn. Example Let any set of production rules in CFG be X → X'X XX XK over the alphabet. The leftmost derivation for the string a+a*a may be − X → X+X → a+X → a + X*X → a+a*X → a+a*a The stepwise derivation of the above string is shown as below − The rightmost derivation for the above string a+a*a may be − X → X*X → X*a → X+X*a → X+a*a → a+a*a The stepwise derivation of The above line is shown as below - Left and right recursive grammar In the context of free G grammar, if there is production in the form of X → Xa, where X is non-terminal and as well is the terminal string, it is called left recurrical products. Grammar with left recursive products is called left recursive grammar. And if in the context of free grammar G, if there is production in the form of X → aX, where X is a non-terminal and rather is a string of terminals, it is called the right recursive production. Grammar, having the right recursive production is called right recursive grammar. Ambiguity in the context of free grammar If the context of free grammar G has more than one tree derivative for some lines w ∈ L (G), this is called ambiguous grammar. There are some of the most right or left derivatives for some lines generated from this grammar. The problem is to check whether grammar G with production rules - X → X'X XX XXX A is ambiguous or not. Solution Let's find out the output of the tree for the line a'a. It has two left derivatives. Conclusion 1 - X → X-X → a X → a'X → a'a'X → a'a Parse tree 1 - Conclusion 2 - X → X X → a X'X → a'a'X → a'a Parse tree 2 - Since there are two parsing trees for one line aaa, grammar G is ambiguous. Languages free of CFL closure context are closed under Union Concatenation Kleene Star Operation Union Let L1 and L2 are two languages free of context. Then L1 ∪ L2 also context free. Example Let L1 - anbn, n zgt; 0 . Appropriate grammar G1 will have P: S1 → aAb'ab Let L2 - cmdm, m ≥ 0 . Appropriate G2 grammar will have P: S2 → cBb ε L1 and L2, L and L1 ∪ → L2, and ∪ see S2 Concatenation If L1 and L2 are languages free of context, then L1L2 is also free of context. Example Union languages L1 and L2, L and L1L2 - anbncmdm - Appropriate Grammar G will have additional production of S → S1 S2 Kleene Star If L is a language free of context, then L- is also free of context. Example Let L and anbn, n ≥ 0. Appropriate grammar G will have P: S → aAb ε Kleene Star L1 - th anbn - Appropriate G1 grammar will have additional productions S1 → SS1 ε languages free of context, not closed in accordance with the intersection - If L1 and L2 are languages free of context, L1 ∩ L2 is not necessarily free of context. Crossing with the usual language - if L1 is a normal language and L2 is a language free of context, L1 ∩ L2 is a language free of context. Supplement: If L1 is a language free of context, then L1' cannot be context-free. CfG simplification in CFG can happen that all production rules and symbols are not needed to get strings. In addition, there may be some zero Eliminating these productions and symbols is called simplification of CFGs. Simplification essentially includes the following steps - Reducing CFG Removal Unit Removal of Null Productions Reducing CFG CFGs are reduced in two phases - Phase 1 - Withdrawal of equivalent grammar, G', from CFG, G, so that each variable receives some terminal line. Withdrawal procedure - Step 1 - Include all W1 characters that get some terminal and initiate i'1. Step 2 - Include all the Wi-1 characters that get Wi. Step 3 - Increment i and repeat step 2 until Wi-1 and Wi. Step 4 - Include all the production rules that have Wi in it. Derivative Procedure - Step 1 - Turn the Origin symbol into Y1 and initiate i No. 1. Step 2 - Include all The Yi-1 characters that can be obtained from Yi and include all the production rules that have been applied. Step 3 - Increment i and repeat step 2 until Yi'1 and Yi. The problem of Finding a reduced grammar equivalent to G grammar with production rules P: S → AC B, A → A, C → C, E → aA e Decision Phase 1 - T - A, C, E - W1, A, C, E, E from Rules A → A, C → C and E → AA W2 - A, C, E - S - From Rule S → AC , C, E, S - U ∅ Since W2 and W3, we can get G' as G' , Where P: S → AC, A → A, C → C, E → aA e Phase 2 - Y1 - S - S - S, A, C - from Rule S → AC Y3 - S, A, C, A, C, C from Rules A → A and C → Y4 - S , A, C, a, c - Since Y3 and Y4, we can get G as G → → - , C → with Removal Unit Productions Any production rule in the form of A → B, where A, B ∈ non-terminal is called a unit of production. Removal Procedure - Step 1 - To remove → B, add production A → x to the grammar rule whenever B → x occurs in grammar. x ∈, x can be null Step 2 - Remove → B from grammar. Step 3 - Repeat step 1 until all units are removed. The problem is to remove the production unit from the following - S → XY, X → a, Y → B, No → M, M → N, N → a solution - There are 3 units of productions in grammar - Y → , No → M, and M → N First, we will remove M → N. As N → a, we M → A, and M → N. The production kit becomes S → XY, X → A, Y → B, No → M, M → A, N → Now we will remove M → As → a, we add Y → a, and → and is removed. The production kit becomes S → XY, X → A, Y → a b, no → A, M → A, N → a Now, M and N are hence we can remove them. The final CFG is a unit produced for free - S → XY, X → A, Y → a b Removal Null Productions In CFG, non-terminal symbol 'A' is a zero variable if there is a production → ε or there is a conclusion that starts with A and finally ends ε: → .... step 1 removal → ε - Find invalid non-terminal variables that flow ε. Step 2 - For each A → a production, build all production a → x where x is derived from 'a' by removing one or more non-terminals from Step 1. Step 3 - Combine the original productions with step 2 and remove ε- productions. The problem of removing zero production from the following - S → ASA A → B→, B → → ε → ε b ∈ solution - there are two zero-toe variables - A and B First, we will remove B → ε → ε → ε. , the production kit becomes S→ASA ab b-n S.A. AS (a) S, → BB B, B → b This is the final production set without zero transition. Chomsky Normal form CFG is in Chom's normal form if the products are in the following forms, where A, B and C are neu terminals and terminals. Chomsky Normal Form - Step 1 - If the start-up S character occurs on some right side, create a new S' starter symbol and a new production of S' → S. Step 2 - Remove zero productions. (Using the Null product removal algorithm discussed earlier) Step 3 - Remove production units. (Using Unit product removal algorithm discussed earlier) Step 4 - Replace each A → B1 product... Bn, where the 2 with → B1C, where C → B2 ... Bn. Repeat this step for all productions having two or more characters on the right side. Step 5 : If the right side of any production is in the form of A → aB, where terminal and A, B are not terminals, then production is replaced by A → XB and X → a. Repeat this step for each production, which is in the form of A → aB. The problem of converting the next CFG to CNF S → asa, → B S, B → b ε solution (1) Since then, as the S appears in R.H.S., we add to the production kit a new state of S0 and S0→S, and it becomes S0→S, S→ ASA aB, → B S, B → b ∈ (2) Now we will remove zero production - B → ∈ and A → ∈ After removal of B → ε , the production kit becomes S0→S, S→ ASA aB A, → B S -1 ∈, B → B After removal of A → ∈, production kit becomes S0→S, S→ ASA aB and AS (a) S.A. S, → B S, B → b (3) After removing the S → S, the production kit becomes S0→S, S→ ASA aB and AS (a) SA, → B S, B → B After removing S0→ S, the production kit becomes S0→ | aB a AS (a) SA, S→ ASA aB a AS (a) SA A → B B → b After removing A→ B, the production kit becomes S0 → ASA aB a B a(a) AS (a) SA, S→ ASA aB and AS (a) SA A → S b B → b After removing A→ S, production kit becomes S0 → ASA aB a b (a) AS (a) SA (a) SA , C → ASA aB a AS (a) SA A → b ASA aB a B AA (a) SA, B → b (4) Now we learn more than two variables in R.H.S here, S0→ ASA, S → ASA, A → ASA breaks two non-terminals in R.H.S. So we will apply Step 4 and Step 5 to get the next final production set, which is in CNF and S0→ AX aB and AS (a) SA S→ AX aB and AS (a) SA A → b AX aB a AS (a) SA B → b X → SA (5) We need to change the productions of S0→ AB, S→ aB, A→ ab And the final production set becomes S0→ AX YB a AS (a) SA→ SX YB a → SA A → b A → b AX YB a AS (a) SA B → b X → SA Y → Greibach Normal Form CF is in Grebachi Normal Form CF is in Grebachi If the products are in the following forms - A → b A → bD1... Dn S → ε where A, D1,....,Dn are non-terminals and b is terminal. CFG conversion algorithm to Greibach Normal Step 1 Form - If the start-up S symbol occurs on some right side, create a new S' starter symbol and a new production of S' → S. Step 2 - Remove zero productions. (Using the Null product removal algorithm discussed earlier) Step 3 - Remove production units. (Using the Unit product removal algorithm discussed earlier) Step 4 - Remove all direct and indirect left recursions. Step 5 - Make the right production substitutions to convert it to the correct form of GNF. The problem of converting the next CFG to CNF S → XY (en) Xn p X → mX m Y → Xn o Solution Here, S does not appear on the right side of any production and no unit or zero production in a set of production rules. So we can skip step 1 to step 3. Step 4 Now after replacing X in S → XY (en) Xo p with mX m we get S → mXY mY mY mY m-mo-mo P. And after replacing X in Y → Xn o on the right side X → mX m we get Y → mXn m O. Two new production o → o and P → p add to the set and then we came to the final GNF as the next S → mXY mY mXC mC p X → mX m Y → mXD mD o O → o P → p Pumping Lemma for CFG If L is the language, Free from context, there is a p pump length so that any string w ∈ L length ≥ p can be written as w q uvxyz where vy ≠ ε, ≤ p, and for all I ≥ 0, uvixyiz ∈ L. Lemma Pumping lemma apps used to check whether the grammar context is free or not. Take an example and show how it is checked. The problem is figuring out whether the language L-xnynzn n ≥ 1 context free or not. The Let L solution is context free. Then L has to satisfy the pumping lemma. First, select the n pumping lemma number. take z as 0n1n2n. Break z into uvwxy, where ≤ n and vx ≠ ε. So vwx may not include both 0s and 2s, since the last 0 and the first 2 are at least (n'1) positions apart. There are two cases - Case 1 and vwx has no 2s. Then vx has only 0s and 1s. Then uwy, which should be in L, has n 2s, but less than n 0s or 1s. Case 2 and vwx has no 0s. Here there is a contradiction. Thus, L is not a language free of context. Pushdown Automata Introduction to push machine is a way to implement contextual grammar in the same way we design DFA for regular grammar. DFA can remember the final amount of information, but the PDA can remember an infinite amount of information. Basically, the push-up machine is the ultimate state machine - the push-up machine stack has three components - input, control unit and stack with infinite size. The head of the stack scans the top of the stack. The stack does two operations - Push - a new symbol is added at the top. Pop - the top symbol is read and removed. The PDA may or may not read the input symbol, but it must read the top of the stack in each transition. PDA can be officially described as a 7-thump (yap δ ∑, q0, I, F) - this is the finical number of states ∑ entering the alphabet S - these are the symbols of the stack, δ this is a function of transition: × (∑ ∪ (ε) × S × × s q0 is the starting state (q0 ∈) I am the initial symbol of the top stack (I ∈ S) F is a set of receiveable states (F ∈) The next chart shows the transition to the PDA from the state q1 to the state q2 marked as a,b → c - This means that in the state of q1, if we encounter the input line 'a' and the top symbol of the stack is 'b', then we pop 'b', click 'c' on top of the stack and move on to the state q2. The terminology associated with the PDA Instant Description Instant Description (ID) PDA is represented by triplets (q, w, s), where q is a state w is an unconsumed entry with is a stack of content turnstile notation turnstiles used to connect pairs of ID that represent one or more moves of the PDA. The transition process is marked by a turnstile symbol ⊢. Consider PDA (I, ∑, S, δ, q0, I, F). The transition can be mathematically represented by the following turnstile notation (p, aw, T) ⊢ (q, w, qb) this means that when you go from p to state q, the input symbol 'a' is consumed, and the top of the 'T' stack is replaced by a new line 'α'. Note: If we want zero or more PDA moves, we must use the symbol (⊢) for it. Pushdown Automata Acceptance There are two different ways of determining the acceptability of PDA. The final state of Acceptability In the final state of eligibility, the PDA takes a line when, after reading the entire line, the PDA is in final condition. From the original state, we can make steps that will end up in the final state with stack values. Stack values don't matter as long as we're bordered in final condition. For PDA (I, ∑, S, δ, q0, I, F), the language adopted by the set of final states F, is L (PDA) (q0, w, I) ⊢ (q, ε, x), q ∈ F for any string stack x input. For PDA (me, ∑, S, δ, q0, I, F) languages, taken by an empty stack - L (PDA) (q0, w, I) ⊢ (q, ε, ε), q ∈ F. Example Build a PDA that takes L = 0n 1n n ≥ 0 Decision This language is taken by L = ε, 01, 0011, 000111, [illegible] Here, in this example, the number of A and B should be the same. Initially, we put a special '$' symbol in an empty stack. Then when q2 is, if we encounter the 0 and top of the Null, we press 0 into the stack. [illegible]

[illegible]

writes on all tracks. Here one head tape reads n characters from n tracks on one step. It takes recursively enumerable languages, like the usual one-current single tape the Turing machine takes. Turing's multi-thread machine can be officially described as a 6-thple (I, X, ∑, δ, q0, F), where q is the ultimate set of states X is the tape alphabet ∑ is the input of the alphabet δ is the relation to states and symbols where δ (qi, a1, a2, a3,....)) is the initial state of the F Right_shift Left_shift,.... — it is a set of final Note states - for each single Turing Turing Machine S, there is an equivalent Turing M multi-current machine, so L (S) and L/M). An undetectable Turing machine in an undetectable touring machine, for every state and character, has a group of actions that TM can have. Thus, the transitions here are not determinant. Calculating an undetectable Turing machine is a configuration tree that can be achieved from the start configuration. Entry is accepted if there is at least one tree node, which is the adoption configuration, otherwise it is not accepted. If all the branches of the computational tree stop at all inputs, a Turing's unspecified machine is called Decider, and if all branches are rejected for some input, the input is also rejected. The Turing's unspecified machine can be officially defined as a 6-duple (Yap,X, ∑, δ, q0, B, F), where q is the final set of states X is the alphabet of the ∑ is the introductory alphabet of the δ is a transitional function; δ : × X → P (× X × (Left_shift, Right_shift). q0 is the initial state of B is an empty F symbol is the set of the final states of the Semi-Infinite Tape Turing Machine Turing Machine with a semi-infinite tape has a left end, But not the right end. The left end is limited to the end of the marker. tape cells. The machine starts with the original q0 state and scans the head from the left end marker 'End'. At every turn he reads the symbol on the tape under his head. He writes a new character on this cell tape and then he moves either left or right one cell tape. The transition function determines determines must be accepted. It has two special states called to recognize the state and reject the state. If at any point in time it enters the accepted state, the entrance is accepted, and if it enters the state of deviation, the entrance is rejected by TM. In some cases, it continues to work indefinitely without being accepted or rejected for certain input characters. Note : Turing machines with semi-infinite tape are equivalent to standard Turing machines. The Linear Connected Machine A linear limited machine is a multi-threaded undetectable Turing machine with a tape of some limited finite length. Length and function (Initial entry length, constant c) Here is information about the memory ≤ with × Input Information Computing is limited to a permanent limited area. Alphabet input contains two special symbols that serve as markers of the left end and markers of the right end, which mean that the transitions do not move to the left end marker or to the right of the marker of the right end of the tape. The linear limited automaton can be defined as 8-tuple (I, X, ∑, q0, ML, MR, δ, F), where q is the final set of states X is the alphabet feed ∑ is the input of the alphabet q0 is the initial state of ML is the left marker of the end of MR is the right marker of the end, where MR ≠ ML δ is a transitional function that each pair (the marker of the right marker of the end is the end of the where MR ≠ ML δ is a transitional function, which displays each pair (ML state is the left marker of the end MR is the right marker where MR ≠ ML δ is a transitional function that displays each pair, the tape symbol) to (state, tape symbol, permanent 'c'), where c can be 0 or No. 1 or -1 F - this is a set of end states determined linear limited machine is always context-sensitive and linear. The language of Decidability Language is called decisive or recursive, if there is a Turing machine that takes and stops at each input line w. Each deciding language is Turing-Acceptable. The problem of solving P is crucial if the L language of all copies of yes P is decisive. For the decisive language, for each line of input, TM stops either when receiving or in a state of deviation, as shown in the following chart : Example 1 Find out whether the next problem is solvable or not - is the number 'm' prime? Decision Prime numbers No. 2, 3, 5, 7, 11, 13, ... Divide the number 'm' into all numbers between '2' and '√m', starting with '2'. If any of these numbers produces the remainder of the zero, it goes to a rejected state, otherwise it goes to the Accepted State. So here the answer can be made yes or No. Therefore, this is a solvable problem. Example 2 Given the usual L and W line, can we check if w ∈ L? Decision Take the DFA, which takes L and check if w is taken some more decisive problems - does the DFA take a blank language? Is L1 ∩ L2 ∅ for regular kits? Note : If the L language is decisive, then its L' addition is also decisive If the language is decisive, that is the registrar for it. Indecisive Indecisive For the hesitant language, there is no Turing machine that adopts the language and decides for each W line input (TM may decide for some input lines though). The problem of solving P is called indecisive if the L language of all copies of yes P is not decisive. Indecisive languages are not recursive languages, but sometimes they can be recursively enumerable languages. Example Stop the Turing Machine Problem Mortality Mortality Deadly Problem Matrix Column Correspondence Problem, etc. Turing Machine Stopping The Problem of Entry - Turing Machine and Input W. Problem - Does Turing Machine Complete String W calculations in the final number of steps? The answer must be either yes or no. Proof : First we assume that such a Turing machine exists to solve this problem, and then we will show that it contradicts itself. We will name this Turing machine as a stop machine that produces yes or no in the finsuute amount of time. If the machine stop ends in the finable amount of time, the exit comes as yes, otherwise as not. Below is the diagram of the Halting machine block - Now we will design an inverted stop machine (HM) as if H returns YES, then cycle forever. If H returns NO, stop. Below is the diagram of the block Inverted Stop Machine - Next, the machine (HM)2, which itself enters is built as follows: If (HM)2 stops at the entrance, cycle forever. Otherwise, stop. Here we have a contradiction. Thus, the problem of stopping is insoluble. The rice theorem of Theorem Rice argues that any non-trivial semantic property of language, which is recognized as the Turing machine, is indecisive. The property, P, is the language of all Turing machines that satisfy this property. Formal Definition If P is a non-trivial property, and the language holding the property, Lp , is recognized as a Turing M machine, then the L'lt;M)∈ of the PL is irrefutable. Description and Properties Of the Language Property, P, it's just a set of languages. If any language is owned by P (L ∈ P), he said that L satisfies property P. Property is called trivial if it is either not satisfied with any recursively enumerable languages, or if it is satisfied with all recursively enumerable languages. The non-trivial property is satisfied by some recursively enumerable languages and is not satisfied by others. Formally speaking, in a non-trivial property where L ∈ P as the following properties hold: Real Estate 1 - There is a Turing Machine, M1 and M2 that recognize the same language, i.e. either (∈ L) or (, ∉ L) Real Estate 2 - There is a Turing Machines M1 and M2 where M1 recognizes the language while M2 does not qlt'm1' qgt; qgt; m1'lt. I'm ∈ L and ∉ L Proof, let's assume the P property is non-trivial and φ P. Так как P нетривиален,</M2> </M1> </M2> </M1> </M2> </M1> </M> </M> </M> at least one language satisfies P, i.e. L/M0 ∈ P, ∋ Turing Machine M0. Let's, w be the entrance at a particular moment and N Turing machine follows - At the entrance x Running M on W If M does not take (or does not stop), does not take x (or does not stop) If M takes w then M run M0 on X. If M0 takes x, take x. The function that displays the ATM,M,w'gt;instance M accepts the input of W e to N in such a way that if M takes w and N takes the same language as M0, then L (M) L (M) L (M0) ∈ p If M does not take w and N takes φ, then L (N) - φ ∉ p Since the ATM is indecisive and can be reduced to Lp, Lp also indecisive. Postal Correspondence Problem Mail Correspondence (PCP), introduced by Emile Post in 1946, is an insoluble problem solution. The PCP problem over the alphabet ∑ is as follows: Given the following two lists, M and N non-empty lines for ∑ - M (x1, x2, x3,........, xn) N (y1, y2, y3,........, yn) We can say that there is a mailing solution if for some i1,i2,........... Ik, where 1 ≤ ij ≤ n, condition xi1 .... xik and yi1 .... yik is satisfying. Example 1 Find out if there's a pen address in the lists of M (abb, aa, aaa) and N q (bba, aaa, aa)? Solution x1 x2 x3 M Abb aa aaa N Bba aaa aa Here, x2x1x3 - 'aaabbaaa' and y2y1y3 - 'aaabbaaa' We see that x2x1x3 - y2y1y3 Hence the solution i q 2, j No 1, and k 3. Example 2 Find out if there's a penhod solution in the lists of M (ab, bab, bbaaa) and N (a, ba, bab) Solution x1 x2 x3 M ab bab bbaaa N a ba bab In this case there is no solution, because x2x1x3 ≠ y2y1y3 (The lengths are not the same) So you can say that this problem after correspondence is indecisive. Irrefutable. </M,w> formal languages and automata theory tutorialspoint pdf. formal languages and automata theory tutorialspoint