ask_automata

<b>Automata Theory</b> is a branch of theoretical computer science that involves designing abstract machines, and solving computational problems that can be solved through the usage of these machines.

ask_automaton

You can represent an automaton using a 5-tuple (Q, , , q0, F), where:\nQ is a finite set of states.\n\n is a finite set of symbols, called the alphabet of the automaton.\n\n is the transition function.\n\nq0 is the initial state from where any input is processed (q0 Q).\n\nF is a set of final state/states of Q (F Q).

ask_alphabet

An <b>alphabet</b> is any finite set of symbols. \nExample   = {a, b, c, d} is an alphabet set where a, b, c, and d are symbols.

ask_string

A <b>string</b> is a finite sequence of symbols taken from .\nExample aabacd is a valid string on the alphabet set = {a, b, c, d}. \n\n A string S accepted by automaton (DFA/NDFA) can be represented as *(q0, S) F \n\n A string S' not accepted by automation (DFA/NDFA) can be represented as *(q0, S) F ask_string_length

The length of the string is the number of symbols in a string denoted <b>|S|</b>. \n Examples: If S = babca, |S|= 5 ask_empty_string

An <b>empty string</b> is a string where its length <b>|S|</b> is equal to 0. \n Denoted using or .

ask_language

A <b>language</b> refers to a set of strings composed of symbols from a given alphabet. \n Example: \n\n A language that can take all possible strings of length 2 over the alphabet = {a, b} is L = { aa, ab, ba, bb }\n\n The language L accepted by automaton(DFA/NDFA) can be represented as {S | S * and *(q0, S) F} \n\n The language L' not accepted by automaton(DFA/NDFA) can be represented as {S | S * and *(q0, S) F} ask_kleene

The Kleene closure is the infinite set of all possible strings with all possible lengths over excluding .

\n Example - If = { a, b } , + = { a, b, aa, ab, ba, bb,..}

ask_transition_function

Denoted by . A <b>transition function</b> is a function where : Q   Q.\n It defines the rules for moving from one state to another based on the current state and the input symbol being read.

ask_ndfa

A <b>NDFA</b> is an automaton where, for a given input symbol, the machine can move to any combination of the states present in the machine. In other words, the exact state to which the machine would move can't be determined.

ask_dfa

A <b>DFA</b> is an automaton that consists of a finite set of states, and a transitition function that can map each state and input symbol to a single next state.

ask_dfa_vs_ndfa

<b>DFA</b>: The transition from a state is to a single particular next state for each input symbol. Hence it is called deterministic. Empty string transitions are not seen in DFA. Backtracking is allowed in DFA. Requires more space. A string is accepted by a DFA, if it transits to a final state. <b>NDFA</b>: The transition from a state can be to multiple next states for each input symbol. Hence it is called non-deterministic. NDFA permits empty string transitions. In NDFA, backtracking is not always possible. Requires less space. A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.

ask_ndfa_to_dfa

1) Create a table from the given NDFA. \n2) Create a blank table under possible input alphabets for the DFA. \n Mark the start state for both the NDFA and the DFA. \n 4) Trace the combination of States {Q0, Q1,... , Qn} for each possible input alphabet. \n5) Keep applying step 4 whenever you generate a new DFA state under the input alphabet columns. \n6) The states that have the corresponding final states of the NDFA are the final states for the NFA.

ask_dfa_minimization

You can follow these steps to acheive DFA Minimization:\n 1) Draw a table for all pairs for all states (Qi, Qj)\n 2) Examine every state pair (Qi, Qj) in the given DFA where Qi F and Qj F or vice versa, then mark them. (F is the set of final states)\n 3) Repeat step 2) until there aren't any states left to mark.\n *it's worth noting that if there's an unmarked pair (Qi, Qj),you should mark it if the pair { (Qi, A), (Qi, A)} is marked for some input alphabet. 4) Combine the remaining unmarked pairs(Qi, Qj), and turn them into a single state in the reduced DFA.

ask_mealy_machine

A Mealy Machine is a Finite State Machine whose output depends on both the present state, and the present input.\n It can be described as a 6 tuple 6 tuple (Q, , O, , X, q0):\n - Q is the finite set of states. \n - is the input alphabet. \n -O is the output alphabet. \n - resembles the input transition function where : Q Q \n -X resembles the output transition function where <b>X: Q O</b> \n -q0 is the initial state.

ask_moore_machine

A Moore Machine is a Finite State Machine whose output depends solely on the present state.\n It can be described as a 6 tuple 6 tuple (Q, , O, , X, q0):\n - Q is the finite set of states. \n - is the input alphabet. \n -O is the output alphabet. \n - resembles the input transition function where : Q   Q \n -X resembles the output transition function where <b>X: Q O</b> \n -q0 is the initial state.

ask_grammar

A Grammar <b>G</b> is a formal system that describes the syntax of a Language. \n A grammar can be considered as a 4-tuple (N, T, S, P): \n - N is a set of variables(non-terminal symbols). \n - T or is a set of Terminal Symbols. \n - S resembles a special variable refered to as the Start symbol where S N. \n P is the production rule(s) for Terminals and Non-terminals. \n Example: G= ({S, A, B}, {a, b}, S, {S AB, A a, B b}) \n -S,A,B N (non terminal symbols). \n - a, b are terminal symbols. \n S is the start symbol, and S N. \n S AB, A a, B b is the production rule.

ask_derivation_grammar

You can derive strings from other strings using the production rules in a given grammar. For example: grammar G = ({S, A}, {a, b}, S, {S aAb, aA aaAb, A   }) \n S aAb (rule S aAb) \n aaAbb (using rule aA aAb) \n aaaAbbb (using rule aA aaAb) \n aaabbb (using rule A) ask_language_from_grammar

The set of all the strings that can be derived from a grammar is called the language generated from a grammar.