

B.Tech.

Fifth Semester Examination

Theory of Automata Computation (CSE-305F)

Short Questions :

Q. 1. (a) Explain the terms alphabet and string.

Ans. Alphabet : An alphabet is defined as a finite, non-empty set of symbols and/or characters. It can be explained with the following examples :

- (i) $A = \{a, b\}$ is an alphabet.
- (ii) $A = \{1, 2, 3, \dots\}$ is not an alphabet as it is infinite.
- (iii) $A = \{\}$ is not an alphabet as it is an empty set.
- (iv) If Σ is an alphabet containing all the 26 characters used in English language, Σ is finite and non-empty set and $\Sigma = \{a, b, c, \dots, z\}$ or $\Sigma = \{A, B, C, D, \dots, Z\}$.

String : A string is a finite sequence of symbols taken from some fixed alphabet string is also known as word.

Ex. For alphabet $\Sigma = \{a, b\}$, 'abab' is string.

Q. 1. (b) Define the Grammar.

Ans. Grammar : A grammar is a finite set of rules over some fixed notations and rules that are used for constructing legal statements for some language. A grammar or phrase structured grammar is defined by 4-tuples and is represented as

$$G = (V_n, \Sigma, P, S) \text{ where}$$

- (i) V_n is a finite, non-empty set of variables or non-terminals. **Ex. :** A, B, C, \dots etc.
- (ii) Σ is finite non-empty set of terminals. **Ex. :** a, b, c, d, \dots
- (iii) P = finite non-empty set of production rules.
- (iv) $S \in V_n$ and is known as start symbol.

Q. 1. (c) What do you understand by DFA (Deterministic Finite Automata) and how is it represented ?

Ans. DFA : A finite automata is said to be deterministic finite automata, if corresponding to an input symbol, there is single resultant state, i.e., there is only one transition. A DFA M is described by 5 tuples

$$M = (Q, \Sigma, S, q_0, F)$$

Q = finite non-empty set of states that are in finite control

Σ = non-empty set of input alphabet

S = is a transition function which maps $Q \times \Sigma \rightarrow Q$, i.e., the head reads a symbol in its present state and moves into next state.

$q_0 \in Q$, is the initial state or starting state

$F \subseteq Q$ is set of final states.

Q. 1. (d) What is Kleene closure ?

Ans. Kleene Closure : Let Σ be some alphabet, the Kleene closure of Σ is denoted by Σ^* . Kleene closure is also known as reflexive transitive closure. For a set of alphabet Σ the language in which any string of letters from Σ is a word, including the null string, is known as the closure of these alphabets. The length of Kleene closure of any alphabet is infinite. It is defined as follows :

$$\Sigma^* = \{\text{set of all words over } \Sigma\}$$

Ex. if $\Sigma = \{a\}$ then $\Sigma^* = \{\epsilon, a, aa, aaa, \dots\}$

Q. 1. (e) What is context free grammar ?

Ans. Context Free Grammar : A grammar G described as $G = (V_n, \Sigma, P, S)$ is said to be CFG where

V_n = set of variables or non-terminals

Σ = set of terminals

P = set of production

$$A \rightarrow \alpha \quad \Rightarrow \quad \begin{array}{l} A = \text{variable} \\ \alpha \in (V_n \cup \Sigma)^* \end{array}$$

Ex. : $G = (V_n, \Sigma, P, S)$

(a) $V_n = \{E\}$

(b) $\Sigma = \{+, *, (,), id\}$

(c) production

(d) $S = \{E\}$

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow id$$

Q. 1. (f) What is ambiguity ? Show that $S \rightarrow aS \mid Sa \mid a$ is an ambiguous grammar.

Ans. Ambiguity : A grammar G is said to be ambiguous if there exists some strings, $W \in L(a)$ for which are two or more distinct derivation trees or there are two or more distinct leftmost derivations.

The given productions are :

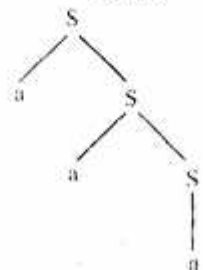
$$S \rightarrow aS$$

$$S \rightarrow Sa$$

$$S \rightarrow a$$

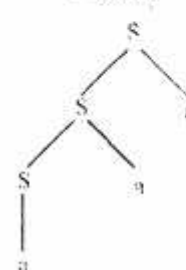
$w = aaa$

$$\begin{array}{l} S \rightarrow aS \\ \rightarrow aaS \\ \rightarrow aaa \end{array}$$



(a) First derivation tree above grammar

$$\begin{array}{l} S \rightarrow Sa \\ \rightarrow Saa \\ \rightarrow aaa \end{array}$$



(b) Second derivation is ambiguous tree.

Q. 1. (g) What is need of push down automata ?

Ans. Need of Pushdown Automata : The finite automata is not capable to recognize all context free languages such as $\{WCWR \mid W \in (a+b)^*\}$. Because while reading the string in finite automata from left to right, it has to remember the string encountered before the symbol 'C' and then compare it with string after 'C'. But finite automata does not have any memory element to remember it.

To remember the string we need a memory element and if we add a memory element to a finite automata to accept the CFL, it is known as PDA. Thus, for making all the strings accepted, pushdown automata is required.

Q. 1. (h) Define turing machine.

Ans. Turing Machine : A turing machine is described by 7 tuples as

$$M = (Q, \Sigma, \Gamma, \delta, q_0, b, F)$$

where Q = finite non-empty set of states

Σ = is the non-empty set of input symbols

Γ = is non-empty set of tape symbols

δ = is the transition function which maps $(Q \times \Gamma) \rightarrow Q \times \Gamma \times \{L, R\}$

$q_0 = q_0$ is initial state

$b \in \Gamma$ = blank symbol

$F \subseteq Q$ = is set of final states.

Q. 1. (i) Explain universal Turing Machine.

Ans. Universal Turing Machine : Each turing machine is capable of performing a single algorithm. This means that we might need to build a separate turing machine for each and every algorithm that we want to execute but it is highly impractical. A universal turing machine is a specific turing machine that can simulate the behaviour of any turing machine. The key idea is to think of the description of the turing machine itself as an input for some other machine. Universal turing machines can run many algorithms at one time. A UTM can be taken as reprogrammable turing machine. A UTM takes as input the description of a TM along with initial tape contents and simulates the input on that turing machine.

Q. 1. (j) What is post correspondence problem ?

Ans. Post Correspondence Problem (PCP) : A correspondence problem P is a finite set of ordered pairs of non-empty strings over some alphabet. Let Σ be an alphabet, the P is finite subse. of $\Sigma^+ \times \Sigma^+$.

A match or solution of P is any string $w \in \Sigma^+$ such that pairs $(U_1, V_1)(u_2, v_2) \dots (U_n, V_n) \in P$ and

$$w = u_1 u_2 \dots u_n = V_1 V_2 \dots V_n \text{ for some } n > 0$$

Ex. Consider a correspondence system

$$P = \{(b, a), (ba, ba), (bab^3, b^3)\}$$

P can be represented as.

i	u_i	V_i
1	b	a
2	ba	ba
3	bab^3	b^3

$$\begin{aligned} u_1 &= b \\ u_2 &= ba \\ u_3 &= bab^3 \\ V_1 &= a \\ V_2 &= ba \\ V_3 &= b^3 \end{aligned}$$

We have a solution,

$$\begin{aligned} W &= u_3 u_1 V_2 = V_2 V_3 V_3 V_1 \\ &= bab^3 b^3 a \end{aligned}$$

Section—A

Q. 2. What do you understand non-deterministic finite automata ? Design a finite automata which accepts language : $L = \{ w : w \text{ has both an even number of 0's and an even number of 1's} \}$ over alphabet $\Sigma = \{0, 1\}$.

Ans. Non-deterministic Finite Automata : A finite automata is said to be non-deterministic if we have more than one possible transition on the same input symbol from some state. A NDFA M is described by 5 tuples as follows :

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q is finite, non-empty set of all the states

Σ is an input alphabet

δ = transition function which maps

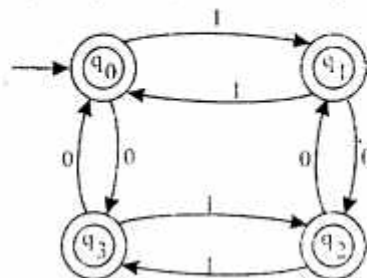
$$Q \times \Sigma \rightarrow 2^Q$$

$q_0 \in Q$, known as initial state.

$F \subseteq Q$, known as set of final states.

We have to construct a DFA that accepts all strings that contain an even number of 0's and an even number of 1's

$$L = \{00, 11, 0101, 0011, 00100111, \dots\}$$



Transition diagram

∴ The DFA will be.

$$\begin{aligned} M &= \{Q, \Sigma, \delta, q_0, F\} \\ Q &= \{q_0, q_1, q_2, q_3\} \\ \Sigma &= \{0, 1\} \\ q_0 &= \{q_0\} \\ F &= \{q_0\} \end{aligned}$$

δ	0	1
$\rightarrow q_0$	q_3	q_1
q_1	q_2	q_0
q_2	q_1	q_3
q_3	q_0	q_2

Q. 3. Consider the following grammar :

$$G = \{S \rightarrow ASb, AS, \epsilon, A \rightarrow \alpha A, \epsilon\}$$

State true or false for the following :

- (i) The string "ababb" is generated by the grammar G.
- (ii) Every string of form $a^n b^m$, $n, m \geq 0$ is generated by G.
- (iii) The language is ambiguous.
- (iv) The language generated by G is regular.

Ans. (i) False : string "ababb"

$$S \rightarrow AS \Rightarrow aAS \Rightarrow a \in S \Rightarrow aAS \Rightarrow aaAS \dots$$

$$S \rightarrow ASb \Rightarrow aASb \Rightarrow a \in Sb \dots$$

$$S \rightarrow ASb \Rightarrow \epsilon Sb \dots$$

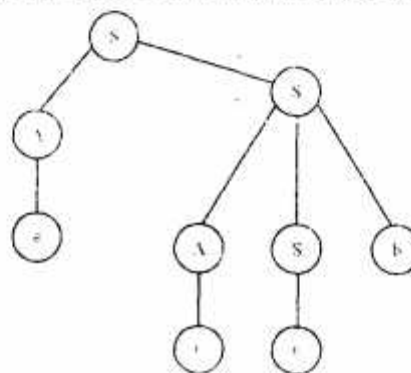
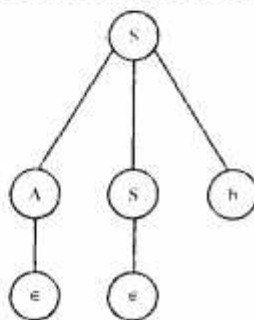
$$S \rightarrow AS \Rightarrow \epsilon S \dots$$

But we cannot get symbol 'b' after symbol 'a' at the beginning of the given string.

(ii) True : The string generated by grammar G is the form

$$a^n b^m, n, m \geq 0$$

(iii) True : For ambiguous string $w = b$, the two distinct derivation trees are shown :



(iv) True : The language generated by the grammar G

$$a^n b^m : n, m \geq 0 \text{ is regular}$$

Section—B

Q. 4. (a) (i) State the Myhill-Nerode Theorem.

Ans. Myhill-Nerode Theorem : A given language L is a regular grammar if and only if the set of equivalence classes of L is finite or we can also state it as follows :

- (i) Language L can be divided into set of all possible strings into separate classes.
- (ii) If L is a regular language then number of classes created is finite.
- (iii) If number of classes that L has, is finite then L is regular language.

Q. 4. (a) (ii) Explain Regular sets.

Ans. Regular Sets : A set represented for a regular expression is called a regular set. Regular languages are also represented in set notation. We use curly brackets ($\{ \}$) to represent a set.

Ex. : if $\Sigma = \{a, b\}$ be an alphabet, then

Regular Set	Regular Expression
$\{ \}$	ϕ
$\{a\}$	a
$\{\epsilon, a, aa, aaa, \dots\}$	a^*
$\{a, a, aaa, \dots\}$	a^+
$\{01, 00, 10, 11\}$	$01 + 00 + 10 + 11$

Q. 4. (b) Prove that the set of strings over alphabet $\{0\}$ of the form 0^n , where n is not prime, is not a regular language.

Ans. We use Pumping Lemma and prove it by contradiction.

Step I : Let us assume that the given language L is regular and accepted by DFA M with n states.

Let n be a prime number.

Step II : Let $w = 0^n$

By Pumping Lemma $w \in L$ such that $|w| \geq n$ and $w = xy^iz$ such that $|xyz| \leq n$ and $|y| > 0$ and x, y, z are strings of 0's.

Step III : Let $y = 0^k$ for some $K \geq 1$ and $K \leq n$ then

$$\begin{aligned}
 i &= n + 1 \\
 |xy^iz| &= |xyz| + |y|^{i-1} \\
 &= n + (i - 1)k \\
 &= n + nk \\
 &= n(1 + k)
 \end{aligned}$$

Thus, $n(1 + k)$ is not prime number. So $xy^iz \notin L$. This is contradiction. Thus, L is not regular.

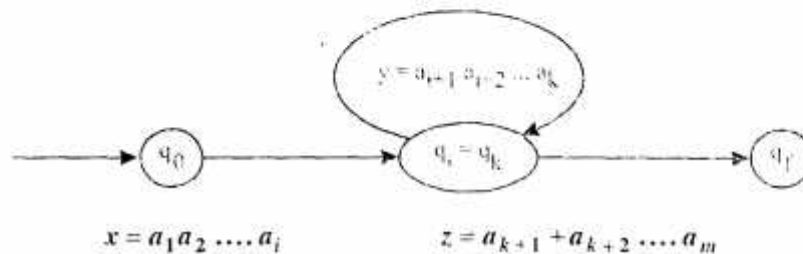
Q. 5. State and prove Pumping lemma for Regular language.

Ans. Pumping Lemma for Regular Sets : Pumping Lemma is a powerful tool for proving certain languages which are non-regular.

It is useful because it gives a method for pumping many sub-strings from a given long string. It gives necessary conditions to prove a set of strings is not regular.

Statement : Let $M = \{Q, \Sigma, \delta, q_0, F\}$ be a finite automata having n states. Let L be a regular language accepted by the finite automata M . A long string $w \in L$ such that $|w| \geq n$ and $w = xyz$, where $y \neq \epsilon$ then $xy^iz \in L$ for $i \geq 0$.

Proof : Finite automata M recognizes L and hence L is a regular languages, if $w \in L$ such that $w = xyz$, we have substring y optional in w and $|w| \geq n$ where n is the number of states in finite automata. Consider the conceptual transition diagram of finite automata M shown below



Let us consider the strings recognized by M strings $w = xy^iz$ for $i \geq 0$ is accepted by M . It is also obvious that substring y is optional and obtained by travelling the loop in state q_i state q_f is the final state. When loop is not travelled, we get $w = xz$ accepted by M .

Let $w = a_1 a_2 a_3 \dots a_i a_{i+1} a_{i+2} \dots a_j a_{j+1} a_{j+2} \dots a_n$

Where $x = a_1 a_2 a_3 \dots a_i$

$y = a_{i+1} a_{i+2} \dots a_j$

$z = a_{j+1} a_{j+2} \dots a_n$

The length of w is m and $m \geq n$. It means, when $m > n$, it indicates that there is some loop in transition diagram. Let y be string obtained from the edges involved in looping as shown in figure.

Case I : When $w = xy^0z = xz$ for $i = 0$, xz is accepted and $xz \in L$.

Case II : $w = xy^iz$ for $i \geq 1$

It means that the control of finite automata M decides to go i -times into the loop with label y and xy^iz is accepted. following are the list of strings accepted by M .

$$\begin{aligned}
 L(M) &= \{xz, xyz, xy^2z, \dots\} \\
 &= \{xy^iz, xy^iz, xy^2z, \dots\} \\
 &= \{xy^iz \mid i \geq 0\}
 \end{aligned}$$

So for all values of $i \geq 0$; $w = xy^iz$ is accepted by M .

Hence statement of Pumping Lemma is true.

Section—C

Q. 6. (i) Define context free grammar. Find a CFG for following language :

$$L = \{a^i b^j c^k \mid j \geq i + k; i, j, k \text{ are non-negative integers}\}.$$

Ans. Context Free Grammar : A grammar G is context free, if it contains only type 2 productions
A type 2 production is of the form $A \rightarrow \alpha$ where

$A \in V_n$ and $\alpha \in (V_n \cup \Sigma)^*$, where

V_n = set of non-terminals

Σ = set of terminals

L

The given language is

$$L = \{a^i b^j c^k \mid j \geq i + k, i, j, k \text{ are non-negative integers}\}$$

Let CFG G generates L

$$G = (V_n, \Sigma, P, S)$$

where, $V_n = \{S, A, B\}$

$$\Sigma = \{a, b, c\}$$

P is the set of production rules and it contains :

$$P_1 : S \rightarrow AB$$

$$P_2 : A \rightarrow aAb$$

$$P_3 : A \rightarrow \Lambda$$

$$P_4 : B \rightarrow bBC$$

$$P_5 : B \rightarrow \Lambda$$

$$S = \{S\}$$

Q. 6. (ii) Let L_1 be a context free language and L_2 be a regular language, then prove that $L_1 \cap L_2$ is CFG.

Ans. We can prove it by taking the examples of context free language and regular language

Consider CFL

$$L_1 = \{a^n b^n \mid n \geq 1\}$$

& regular language $L_2 = \{a^p b^q \mid p, q \geq 1\}$

Then intersection of L_1 and L_2 is a context free language.

Let us express the strings of L_1 and L_2

$$L_1 = \{ab, aabb, aaabbb, \dots\}$$

$$L_2 = \{ab, aab, aabb, abbb, aabbb, \dots\}$$

So $L_1 \cap L_2 = \{ab, aabb, aaabbb, \dots\}$

$$= \{a^k b^k \mid k \geq 1\}$$

Hence $L_1 \cap L_2$ is a context free language.

Q. 7. (i) Convert the following grammar to a PDA that accepts the same language

$$S \rightarrow 0 S 1 | A$$

$$A \rightarrow 1 A 0 | S | \epsilon$$

Ans. The productions of given grammar are

$$S \rightarrow 0 S 1 | A$$

$$A \rightarrow 1 A 0 | S | \epsilon$$

Let the equivalent PDA be

$$P = \{q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$$

The tuples of PDA are described as :

$$Q = \{q\}$$

$$\Sigma = \{0, 1\}$$

$$\Gamma = \{0, 1, S, A\}$$

δ is defined as :

$$\delta(q, \epsilon, S) = (q, 0S1)$$

$$\delta(q, \epsilon, S) = (q, A)$$

$$\delta(q, \epsilon, A) = (q, 1A0)$$

$$\delta(q, \epsilon, A) = (q, S)$$

$$\delta(q, \epsilon, A) = (q, \epsilon)$$

$$\delta(q, 0, 0) = (q, \epsilon)$$

$$\delta(q, 1, 1) = (q, \epsilon)$$

$$q_0 = \{q\}$$

$$Z_0 = S$$

$$F = \emptyset$$

Q. 7. (ii) What is Halting problem of Turing Machine.

Ans. Halting Problem of Turing Machine : The Halting problem of a Turing machine is a decision problem which can be simply stated as :

For a given description of algorithm and a description of its initial arguments (or inputs), determines whether the algorithm, when executed with these arguments, ever halts or the alternative is that given algorithm runs forever without halting.

If a Turing machine recognizes a language L , we assume that the Turing machine halts whenever the input string is accepted. However, there may be several strings for which the same Turing machine will never halt. The halting problem of Turing machine is unsolvable problem, but we can run the machine with a given input. If it halts then a specific case is answered. If the machine does not halt the conclusion is that the machine will never halt by recognizing some pattern with the computation.

Section—D

Q. 8. Explain in detail. CHOMSKY classification of grammar and also define Chomsky Hierarchy.

Ans. Chomsky Classification of Grammar : There are four types of grammars :

- (i) (Type 0) Unrestricted Grammar
- (ii) (Type 1) Context Sensitive Grammar
- (iii) (Type 2) Context Free Grammar
- (iv) (Type 3) Regular Grammar

Unrestricted (type 0) Grammar : An unrestricted grammar G consist of 4 tuples

$$G = (V, \Sigma, P, S)$$

V = non-terminals

Σ = terminal

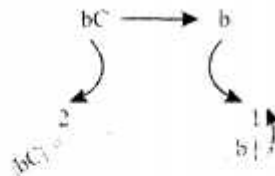
P = production rule

S = starting symbol

A grammar is called an unrestricted grammar if it contains a contracting production rules.

It is a production rule which is in form of $w_{i+1} \rightarrow w_i$ i.e., w_i produce shorter length than w_{i+1} .

Ex. :



Context Sensitive Grammar (or Type 1) : A context sensitive grammar G consist of 4 tuples

$$G = (V, \Sigma, P, S)$$

A context sensitive grammar is a grammar in which each production contain a non-contracting production rules.

Non-Contracting Production : A production $\alpha \rightarrow \beta$ satisfying $|\alpha| \leq |\beta|$ is known as non-contracting production.

Context Free Grammar (or Type 2) : A context free grammar G consist of 4 tuples

$$G = (V, \Sigma, P, S)$$

The grammar may also contain a production rule $S \rightarrow \epsilon$, if S is starting symbol and left hand side of production rule always contains a single non-terminal symbol.

Regular Grammar (Type 3) : A regular Grammar G consist of 4-tuples

$$G = (V, \Sigma, P, S)$$

V = set of non-terminal

Σ = set of terminal

P = production rules are in either of the following form

Form 1 : A production rule in the form of

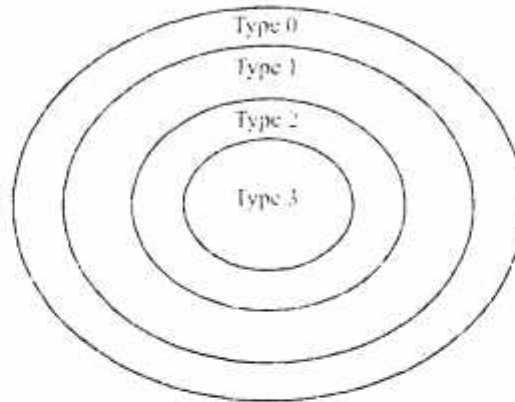
$$\left. \begin{array}{l} A \rightarrow aB \\ A \rightarrow a \end{array} \right\} \begin{array}{l} A \in (V \cup S) \\ B \in V \\ a \in \Sigma \end{array} \quad \text{right linear production}$$

Form 2 : A production rule in the form of

$$\left. \begin{array}{l} A \rightarrow Ba \\ A \rightarrow a \end{array} \right\} \begin{array}{l} A \in (V \cup S) \\ B \in V \\ a \in \Sigma \end{array} \quad \text{left linear production}$$

S = starting symbol.

Chomsky Hierarchy :



Production rules of all types of grammars we will find some relation among them.

Type 1 \subseteq Type 0 ;

Type 2 \subseteq Type 1

Type 3 \subseteq Type 2

Q. 9. What is primitive Recursive function show that $f(x, y) = x + y$ is a primitive recursive function ?

Ans. Primitive Recursive Function : A total function f is said to be primitive recursive if (i) it is any one of the three initial functions or (ii) it can be obtained by the application of composition and recursion repeated a finite number of times to the set of initial functions.

Or A total function is called primitive recursive if it can be obtained by the application of composition and recursion a finite number of times repeated to initial functions.

$\Rightarrow f(x, y) = x + y$ is a primitive recursive function

We know that a function f of $(n + 1)$ variables is defined by recursion. If there exists a function g of n variables and another function h of $(n + 2)$ variables such that

$$f(x_1, x_2, \dots, x_n, 0) = g(x_1, x_2, \dots, x_n) \quad \dots(i)$$

$$f(x_1, x_2, \dots, x_n, y + 1) = h(x_1, x_2, \dots, x_n, y, f(x_1, x_2, \dots, x_n, y)) \quad \dots(ii)$$

In given question, f is of 2 variables so $f(x, y)$ is defined by recursion if we have a function g of one variable and a function h of 3 variables.

$$f(X, 0) = X + 0 = X = g(X) \quad \text{by equation (i)} \quad \dots(iii)$$

$$\text{i.e.,} \quad g(X) = X = U_1'(X) \quad \dots(iv)$$

$$\begin{aligned} \text{Also} \quad f(x, y + 1) &= X + (Y + 1) = (X + y) + 1 \\ &= f(x, y) + 1 \quad \dots(v) \end{aligned}$$

Comparing equations (ii) & (v)

$$\begin{aligned} h(x, y, f(x, y)) &= f(x, y) + 1 \\ &= S(f(x, y)) \\ &= S(U_3^{-3}(x, y, f(x, y))) \end{aligned}$$

Now define $h(x, y, z) = S_3^{-3}(x, y, z)$

$g = U_1^{-1}(x)$, we can take g as initial function and then h is obtained from initial function U_3^{-3} and S by composition and f by recursion using g and h . Therefore f is obtained by application of composition and recursion a finite number of times to initial function U_1^{-1} , U_3^{-3} and S . Thus, f is primitive recursive.

Alternative method

We know that

$$X + (Y + 1) = (x + y) + 1$$

$$\begin{aligned} \text{or } f(x, y + 1) &= (x + y) + 1 = f(x, y) + 1 = S(f(x, y)) \\ f(X, 0) &= X \end{aligned}$$

Defining $f(x, y)$ as

$$f(X, 0) = X = g(X) = U_1^{-1}(X)$$

$$f(x, y + 1) = S(U_3^{-3}(x, y, f(x, y)))$$

So f is primitive recursive.