Turing Machines and Grammars

Turing Machines

The basic concept is that the Turing machine manipulates a string on a two-way infinite tape according to transition rules. The tape contains an infinite number of cells, and each cell contains one letter. At the beginning, the tape contains the input string, and the rest of the cells contain a special tape symbol called a blank symbol. There is a head, which can read and write the content of the current cell of the tape, and can move both to the left and to the right. At the beginning, the head is over the first letter of the input string. The Turing machine also has its own inner state, which can be changed in each step. At the beginning, the inner state of the Turing machine is the initial state. The transition rules are the "program" of the Turing machine.

n each step the machine reads the letter contained by the current cell of the tape, and also reads its own inner state, then writes a letter into the current cell, changes its inner state and moves the head to the left or to the right, or stays in the same position. Sometimes, it does not change its inner state, and sometimes it does not change the content of the current cell. The operations of the Turing machine are based on the transition rules.

Definition The (nondeterministic) Turing machine (TM) is the following 7-tuple:

$$TM = (Q, T, V, q_0, \#, \delta, F)$$

where

- Q is the finite nonempty set of the states,
- T is the set of the input letters, (finite nonempty alphabet), $T \subseteq V$,
- V is the set of the tape symbols, (finite nonempty alphabet),
- q_0 is the initial state, $q_0 \in Q$,
- $\#$ is the blank symbol, $\# \in V$,
- δ is the transition function having a form

  $Q \times V \to 2^{Q \times V \times \{ \text{ Left , Right , Stay } \}}$, and
- F is the set of the final states, $F \subseteq Q$.

Language accepted by Turing machine

A configuration is called a final configuration, if the Turing machine is in a final state. Now, we can define the language accepted by the Turing machine. The input word is accepted, if the Turing machine can change its configuration from the initial configuration to a final configuration in finite number of steps.

Definition[1]

$$L(TM) = \{p \mid p \in T^*, (\lambda, q_0, p) \vdash_{TM}^* (u, q_f, av), q_f \in F, a \in V, u, v \in V^*\}$$

Deterministic Turing machine

The deterministic Turing machine, which has the same language definition power as the nondeterministic Turing machine. A Turing machine is called deterministic, if from each configuration it can reach at most one other configuration in one step. The formal definition is the following:

Definition The Turing machine $TM_d = (Q, T, V, q_0, \#, \delta, F)$ is deterministic, if the transition function $\delta(q, a)$ has at most one element for each pair (q, a), where $q \in Q$ and $a \in V$.

In other words, the Turing machine $TM_d = (Q, T, V, q_0, \#, \delta, F)$ is deterministic, if the form of the transition function δ is

$$Q \times V \to Q \times V \times \{\text{Left, Right, Stay}\}.$$

Theorem For each Turing machine TM there exists a deterministic Turing machine TM_d such that $L(TM) = L(TM_d)$.

The Church-Turing Thesis

The original reason for introducing the Turing machine was to simulate mathematical algorithms which can calculate complicated functions. Later, it has been recognized that all algorithmically computable functions can be calculated by the Turing machine, as well.

Thesis The statement which claims that a function is algorithmically computable if and only if it can be computed by the Turing machine is called Church-Turing thesis or simply Church's thesis.

Church's thesis is not a theorem, it cannot be proven, because "algorithmically computable" is not a well defined expression in the thesis. In spite of the fact that Church's thesis is not a proven theorem, the thesis is accepted among scientists.

[1] Let X and Y be configurations of the same Turing machine. Then, we say that the Turing machine can change its configuration from X to Y in finite number of steps, if $X = Y$, or there are configurations $C_0, C_1, \ldots, C_n$ such that $C_0 = X, C_n = Y$, and $C_i \vdash_{TM} C_{i+1}$ holds for each integer $0 \leq i < n$. It is denoted by $X \vdash_{TM}^* Y$.

The most important consequence of the thesis is that even the latest, and the strongest computer with a highly improved computer program can only compute the same things as a very simple Turing machine. Therefore we can use the Turing machine to show if something can be computed or not; and this is why the Turing machine plays a fundamental role in the algorithms and computational theory.

Alternative definition of complexity classes P and NP

Definition P is the set of decision problems solvable in polynomial time by a deterministic Turing machine.

Definition NP is the set of decision problems solvable in polynomial time by a non-deterministic Turing machine.

Recursive and Recursively Enumerable Languages

Recursive Languages — Definition

The language $L \in V^*$ is recursive, if there is an algorithm, which decides about any word $p \in V^*$, whether or not $p \in L$.

Recursively Enumerable Languages — Definition

The language $L \in V^*$ is recursively enumerable, if there is a procedure, which specifies all the elements of L.

Theorem The language L is recursive, if and only if both L and $\bar{L}$ are recursively enumerable.

Theorem Each context-sensitive language is recursive, but there are recursive languages which are not context-sensitive.

Theorem There exists a language L, which is not recursively enumerable.

Theorem The class of recursive languages is closed under complement operation.

Theorem The class of recursively enumerable languages is not closed under complement operation.

Theorem The class of recursively enumerable languages is closed under regular operations.

Theorem The class of recursively enumerable languages is closed under intersection.

Generative Grammar

Definition

The generative grammar (G) is the following quadruple:

$$G = (N, T, S, P)$$

where

- N is the set of the nonterminal symbols, also called variables, (finite nonempty alphabet),
- T is the set of the terminal symbols or constants (finite nonempty alphabet),
- S is the start symbol, and
- P is the set of the production rules.

The following properties hold: $N \cap T = \emptyset$ and $S \in N$.

Let us denote the union of the sets N and T by $V (V = N \cup T)$.

Then, the form of the production rules is $V^*NV^* \to V^*$.

Derivation — Definition

Let $G = (N, T, S, P)$ be a generative grammar, and let p and q be words over the joint alphabet $V = N \cup T$. We say that the word q can be derived in one step from the word p, if p can be written in a form $p = u \times w, q$ can be written in a form $q = uyw$, and $x \to y \in P$. (Denoted by $p \Rightarrow$ q.)

The word q can be derived from the word p, if $q = p$ or there are words $r_1, r_2, \ldots, r_n$ such that $r_1 = p, r_n = q$ and the word r_i can be derived in one step from the word r_{i-1}, for each $2 \leq i \leq n$. (Denoted by $p \Rightarrow^*$ q.)

Generated Language — Definition

Let $G = (N, T, S, P)$ be a generative grammar. The language generated by the grammar G is

$$L(G) = \{p \mid p \in T^*, S \Rightarrow^* p\}.$$

Chomsky Hierarchy — Definition

Let $G = (N, T, S, P)$ be a generative grammar.

- Type 0 or unrestricted grammars. Each generative grammar is unrestricted.
- Type 1 or context-sensitive grammars. G is called context-sensitive, if all of its production rules have a form

$$p_1 A p_2 \to p_1 q p_2,$$

or

$$S \to \lambda,$$

where $p_1, p_2 \in V^*, A \in N$ and $q \in V^+$. If $S \to \lambda \in P$ then S does not appear in the right hand side word of any other rule.

- Type 2 or context-free grammars. The grammar G is called context-free, if all of its productions have a form

$$A \to p,$$

where $A \in N$ and $p \in V^*$.

- Type 3 or regular grammars. G is called regular, if all of its productions have a form

$$A \to r$$

or

$$A \to rB$$

Languages — Definitions

- The language L is called recursively enumerable, if there exists an unrestricted grammar G such that $L = L(G)$.
- The language L is context-sensitive, if there exists a context-sensitive grammar G such that $L = L(G)$.
- The language L is context-free, if there exists a context-free grammar G such that $L = L(G)$.
- The language L is regular, if there exists a regular grammar G such that $L = L(G)$.
