Basix

28 March 2021 06:54 AM

The subject of theory of computation deals with the answers to the question "What is the fundamental capabilities and limitations of computer?"

This same question has been answered in three different ways (or by considering three different aspects of problem)

Complexity - certain problems are theoretically solvable by the computer but the cost is so high (time complexity or space complexity is so high) that it becomes practically impossible to solve them.

Computability - certain problems can not be solved by a computer altogether for example ascertaining the correctness of a mathematical statement.

Automata - this deals with the properties and definitions of mathematical models of computation.

Mathematical Notions and terminology

28 March 2021 07:18 AM

(selected concepts)

Sets

Proper subset : A is a proper subset of B if A is a subset of b and is not equal to B, written as

:n $A \subsetneq B$,

Usually an identical elements in a set are overlooked but when it is required to be considered we call that set a multiset

Sequences and tuples

A sequence of number in specific order is called sequence

When a sequence is finite up to k elements in it. It is called k-tuple

Cross product / cartesian product of set

If $A = \{1, 2\}$ and $B = \{x, y, z\}$,

$$A \times B = \{ (1, x), (1, y), (1, z), (2, x), (2, y), (2, z) \}.$$

Functions and relations

A special type of binary relation, called an ***equivalence relation***, captures the notion of two objects being equal in some feature. A binary relation R is an equivalence relation if R satisfies three conditions:

1. R is ***reflexive*** if for every x, xRx;
2. R is ***symmetric*** if for every x and y, xRy implies yRx; and
3. R is ***transitive*** if for every x, y, and z, xRy and yRz implies xRz.

Graphs

Represented as a pair two set vertices and edges

$$\big(\{1,2,3,4,5,6\},\ \{(1,2), (1,5), (2,1), (2,4), (5,4), (5,6), (6,1), (6,3)\}\big).$$

Strings and language

Alphabets - a set containing list of symbols to be used for that language

String over alphabets - a sequence of symbols from the set of alphabets

lexicographical order - dictionary order

Language - set of strings

Boolean Logic/Algebra

SUMMARY OF MATHEMATICAL TERMS

Alphabet	A finite set of objects called symbols
Argument	An input to a function
Binary relation	A relation whose domain is a set of pairs
Boolean operation	An operation on Boolean values
Boolean value	The values TRUE or FALSE, often represented by 1 or 0
Cartesian product	An operation on sets forming a set of all tuples of elements from respective sets
Complement	An operation on a set, forming the set of all elements not present
Concatenation	An operation that sticks strings from one set together with strings from another set
Conjunction	Boolean AND operation
Connected graph	A graph with paths connecting every two nodes
Cycle	A path that starts and ends in the same node
Directed graph	A collection of points and arrows connecting some pairs of points
Disjunction	Boolean OR operation
Domain	The set of possible inputs to a function
Edge	A line in a graph
Element	An object in a set
Empty set	The set with no members
Empty string	The string of length zero
Equivalence relation	A binary relation that is reflexive, symmetric, and transitive
Function	An operation that translates inputs into outputs
Graph	A collection of points and lines connecting some pairs of points
Intersection	An operation on sets forming the set of common elements
k-tuple	A list of k objects
Language	A set of strings
Member	An object in a set
Node	A point in a graph
Pair	A list of two elements, also called a 2-tuple
Path	A sequence of nodes in a graph connected by edges
Predicate	A function whose range is {TRUE, FALSE}
Property	A predicate
Range	The set from which outputs of a function are drawn
Relation	A predicate, most typically when the domain is a set of k-tuples
Sequence	A list of objects
Set	A group of objects
Simple path	A path without repetition
String	A finite list of symbols from an alphabet
Symbol	A member of an alphabet
Tree	A connected graph without simple cycles
Union	An operation on sets combining all elements into a single set
Vertex	A point in a graph

Definitions proofs and theorems

06 April 2021 11:37 AM

Omitted because topics are naive

Types of proofs

06 April 2021 11:38 AM

Proof by construction

These method aims to prove the existence of object by actually constructing it.

Proof by contradiction

We first assume that the theorem is false and then it leads us to a contradiction which impels our assumption must be wrong.

Proof by induction

We use mathematical induction to show that a particular statement is true for all values

Basix

07 April 2021 12:20 PM

Terminology

Automata - (plural of automation) it is a system (or machine) where there are multiple states and depending upon the input given to it ,it changes its state.

Start state - the state from which the machine starts is the start state.

Accept state - the state which is considered true aka final state

Reject state - the state which is considered false

Transition - the transitioning from one state to another

State diagram:

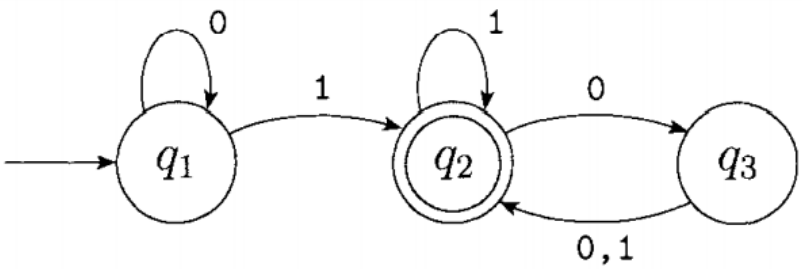


FIGURE 1.4

A finite automaton called M_1 that has three states

Language of machine - if a set of string A is accepted by a machine as input it is called the language of machine or machine M recognizes language A.

Remark: there is a technical difference between recognizing a language and accepting a string, a machine may accept several strings but recognizes only one language.

Regular language - if some finite automaton recognizes a language it is called regular language.

Equivalent machines - two machines are equivalent if they recognize the same regular language.

Finite automata

07 April 2021 12:32 PM

DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.[2]

F may be null
There is at least one transition corresponding to every state\

Example

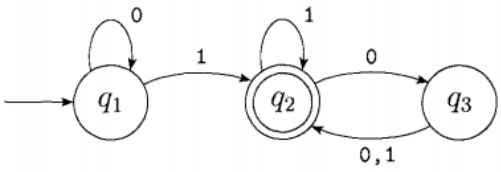


FIGURE 1.6
The finite automaton M_1

We can describe M_1 formally by writing $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 is the start state, and
5. $F = \{q_2\}$.

Features of a finite automaton
- Finite (limited) memory
- Finite number of states
- Finite control

Computation

07 April 2021 01:24 PM

Informally computation is the phenomenon when the machine recognizes a language and computes an input in the language and reacts by changing its state.

Formal definition

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

Condition 1 says that the machine starts in the start state. Condition 2 says that the machine goes from state to state according to the transition function. Condition 3 says that the machine accepts its input if it ends up in an accept state. We say that M ***recognizes language*** A if $A = \{w | M \text{ accepts } w\}$.

The regular operations

07 April 2021 01:35 PM

Similar additions subtractions … in mathematics there are three fundamental operations on regular languages, these are called regular operations.

DEFINITION 1.23

Let A and B be languages. We define the regular operations ***union***, ***concatenation***, and ***star*** as follows.

- **Union**: $A \cup B = \{x | \ x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy | \ x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \ldots x_k | \ k \geq 0 \text{ and each } x_i \in A\}$.

Union - takes to set of languages and makes a new set the union of two
concatenation - makes a set by appending the alphabets of A in front of B in all possible ways.
Star - it attaches any number of alphabets belonging to a in front of one another.\

Let the alphabet Σ be the standard 26 letters $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$. If $A = \{\mathtt{good}, \mathtt{bad}\}$ and $B = \{\mathtt{boy}, \mathtt{girl}\}$, then

$$A \cup B = \{\mathtt{good}, \mathtt{bad}, \mathtt{boy}, \mathtt{girl}\},$$

$$A \circ B = \{\mathtt{goodboy}, \mathtt{goodgirl}, \mathtt{badboy}, \mathtt{badgirl}\}, \text{ and}$$

$$A^* = \{\varepsilon, \mathtt{good}, \mathtt{bad}, \mathtt{goodgood}, \mathtt{goodbad}, \mathtt{badgood}, \mathtt{badbad}, \mathtt{goodgoodgood}, \mathtt{goodgoodbad}, \mathtt{goodbadgood}, \mathtt{goodbadbad}, \ldots\}.$$

Properties / theorem

- The class of regular languages is closed under union operations
- The class of regular languages is closed under concatenation operations
  - The proofs are worth a look sipser pg 45

Non determinism

07 April 2021 01:55 PM

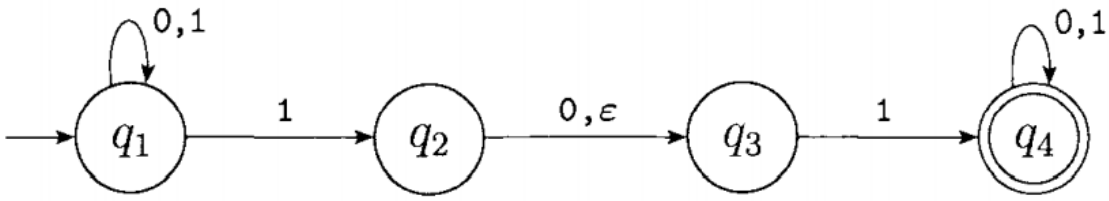


FIGURE 1.27
The nondeterministic finite automaton N_1

Consider the automaton here in each step for a given input the automaton may proceed in a number of ways
from q1 if input 1 is given it has two choices it may remain in state q1 or it may transit to state q2.
from q2 if input 0 is given or no input is given it always goes to state q3.

Thus whenever the automation has more then one choice of states for a particular input it is called nondeterministic automata.

NFA - non deterministic finite automata
DFA - deterministic finite automata

Computation in NFA
The computation proceeds as in DFA but when two or more possible choice of states are found the machine splits itself into that possible cases and each of the "forked threads" follow these different choices of states. This forking continues at each point whenever a machine encounters a choice of states until one of the threads reaches the accept states, then all the parallel forked threads or machines dies out and the machine accepts the input however if non of the threads reaches the accept states the machine reject the input, this is like parallel processing.

Representation of NDA

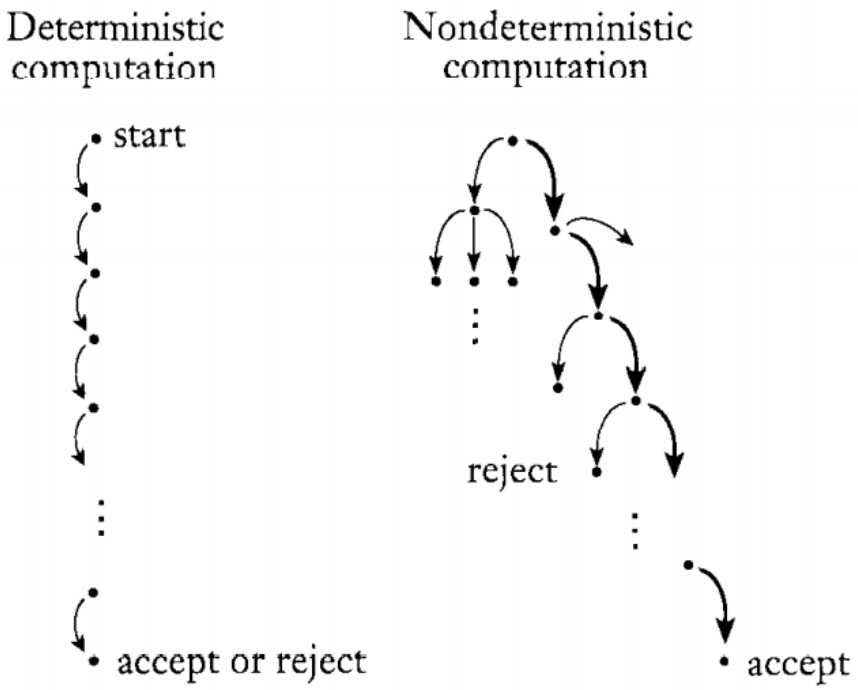


FIGURE 1.28
Deterministic and nondeterministic computations with an accepting branch

Non deterministic automata

08 April 2021 08:58 AM

DEFINITION 1.37

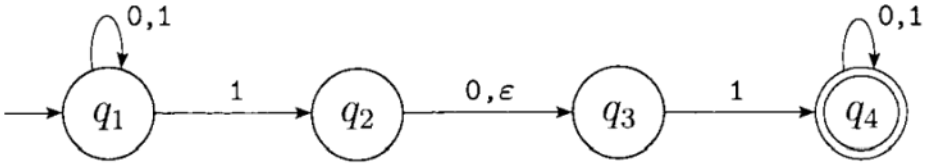
A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

The transition function in NDA may take an empty string an addition to a string from alphabet set and instead of producing a single state as in DFA it produces a set of state.

Example

Recall the NFA N_1:



The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4. q_1 is the start state, and
5. $F = \{q_4\}$.

Equivalence of NFAs and DFAs

08 April 2021 10:50 AM

Theorems

For every NFA there is an equivalent DFA
A language is regular if some NFA recognizes it.

Proofs in sipser pg 55

Converting a NFA to DFA seems important and is given in sipser

Closure under regular operations

08 April 2021 11:08 AM

Omitted due to complexity will be done later on

Regular expressions

08 April 2021 11:09 AM

Informal notions

In mathematics we combine operations to form expressions eg: (3 + 5) x 8

In regular language also we combine operations to form expressions called regular expressions

eg ({0} U {1}) . {0}*

The given expression in short hand is also written as

(0 U 1)0* the dot and set brackets are omitted

Solving this :

={0,1}{0}*

={0,1,00,10,000,1000,….}

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

In items 1 and 2, the regular expressions a and ε represent the languages $\{a\}$ and $\{\varepsilon\}$, respectively. In item 3, the regular expression $\emptyset$ represents the empty language. In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R_1 and R_2, or the star of the language R_1, respectively.

Don't confuse the regular expressions ε and $\emptyset$. The expression ε represents the language containing a single string—namely, the empty string—whereas $\emptyset$ represents the language that doesn't contain any strings.

Precedence of operations

* star

. concatenation

U union

Theorem

A language is regular if and only if some regular expression describes it.

The regular expression is equivalent with finite automata

Omitted topics

Generalized NFA (GNFA)

conversion of DFA to regular expression

Non regular languages

08 April 2021 11:42 AM

Languages which are not regular are called non regular languages.
All regular language follows the pumping lemma, if a language does not follow pumping lemma it is guaranteed to be non regular.

Pumping lemma

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Recall the notation where $|s|$ represents the length of string s, y^i means that i copies of y are concatenated together, and y^0 equals ε.

When s is divided into xyz, either x or z may be ε, but condition 2 says that $y \neq \varepsilon$. Observe that without condition 2 the theorem would be trivially true. Condition 3 states that the pieces x and y together have length at most p. It is an extra technical condition that we occasionally find useful when proving certain languages to be nonregular. See Example 1.74 for an application of condition 3.

EXAMPLE 1.73

Let B be the language $\{0^n1^n | n \geq 0\}$. We use the pumping lemma to prove that B is not regular. The proof is by contradiction.

Assume to the contrary that B is regular. Let p be the pumping length given by the pumping lemma. Choose s to be the string 0^p1^p. Because s is a member of B and s has length more than p, the pumping lemma guarantees that s can be split into three pieces, $s = xyz$, where for any $i \geq 0$ the string xy^iz is in B. We consider three cases to show that this result is impossible.

1. The string y consists only of 0s. In this case the string $xyyz$ has more 0s than 1s and so is not a member of B, violating condition 1 of the pumping lemma. This case is a contradiction.
2. The string y consists only of 1s. This case also gives a contradiction.
3. The string y consists of both 0s and 1s. In this case the string $xyyz$ may have the same number of 0s and 1s, but they will be out of order with some 1s before 0s. Hence it is not a member of B, which is a contradiction.

Thus a contradiction is unavoidable if we make the assumption that B is regular, so B is not regular. Note that we can simplify this argument by applying condition 3 of the pumping lemma to eliminate cases 2 and 3.

In this example, finding the string s was easy, because any string in B of length p or more would work. In the next two examples some choices for s do not work, so additional care is required.

Basix

21 May 2021 07:53 AM

CFL is a more powerful way of describing a language just like regular languages or DFA
The grammar of CFL called context free grammar is such that it allows recursive features to be described
Applications of CFL

- A grammar acts as reference for people trying to learn a programming language
- Given a grammar a parser can be generated automatically by certain tools

An collection of languages associated with CFG are called context free languages these include regular languages as well as some other languages.
A class of machine recognizing those languages are called pushdown automata

Context free grammars

21 May 2021 08:09 AM

The following is an example of a context-free grammar, which we call G_1.

$$A \to \mathtt{0}A\mathtt{1}$$
$$A \to B$$
$$B \to \mathtt{\#}$$

A grammar consists of a collection of ***substitution rules***, also called ***productions***. Each rule appears as a line in the grammar, comprising a symbol and a string separated by an arrow. The symbol is called a ***variable***. The string consists of variables and other symbols called ***terminals***. The variable symbols often are represented by capital letters. The terminals are analogous to the input alphabet and often are represented by lowercase letters, numbers, or special symbols. One variable is designated as the ***start variable***. It usually occurs on the left-hand side of the topmost rule. For example, grammar G_1 contains three rules. G_1's variables are A and B, where A is the start variable. Its terminals are `0`, `1`, and `#`.

You use a grammar to describe a language by generating each string of that language in the following manner.

1. Write down the start variable. It is the variable on the left-hand side of the top rule, unless specified otherwise.
2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.
3. Repeat step 2 until no variables remain.

For example, grammar G_1 generates the string `000#111`. The sequence of substitutions to obtain a string is called a ***derivation***. A derivation of string `000#111` in grammar G_1 is

$$A \Rightarrow \mathtt{0}A\mathtt{1} \Rightarrow \mathtt{00}A\mathtt{11} \Rightarrow \mathtt{000}A\mathtt{111} \Rightarrow \mathtt{000}B\mathtt{111} \Rightarrow \mathtt{000\#111}$$

You may also represent the same information pictorially with a ***parse tree***. An example of a parse tree is shown in Figure 2.1.

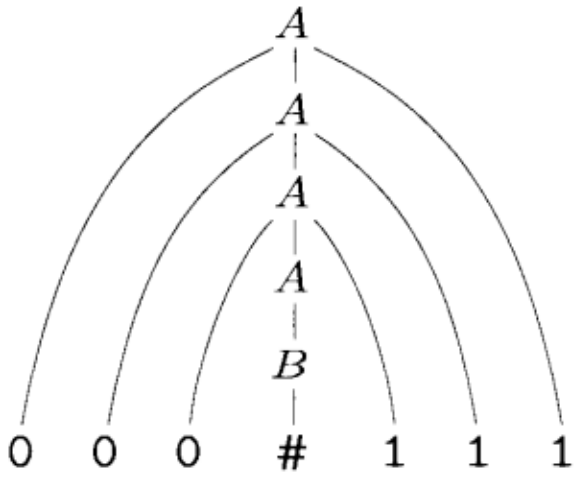


FIGURE 2.1
Parse tree for 000#111 in grammar G_1

All strings generated in this way constitute the ***language of the grammar***. We write $L(G_1)$ for the language of grammar G_1. Some experimentation with the grammar G_1 shows us that $L(G_1)$ is $\{0^n\#1^n \mid n \geq 0\}$. Any language that can be generated by some context-free grammar is called a ***context-free language*** (CFL). For convenience when presenting a context-free grammar, we abbreviate several rules with the same left-hand variable, such as $A \rightarrow 0A1$ and $A \rightarrow B$, into a single line $A \rightarrow 0A1 \mid B$, using the symbol " | " as an "or."

Formal definition

21 May 2021 08:16 AM

DEFINITION 2.2

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

If u, v, and w are strings of variables and terminals, and $A \to w$ is a rule of the grammar, we say that uAv ***yields*** uwv, written $uAv \Rightarrow uwv$. Say that u ***derives*** v, written $u \overset{*}{\Rightarrow} v$, if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$

Derivation of String

03 June 2021 09:03 AM

Left Most derivation

The left most non-terminal is substituted with corresponding production to derive a string

S -> aAB|a
A -> aBA|a
B -> b

Then LMD of aabab is S -> aAB -> aaBaB -> aabAB -> aabaB -> aabab

Right Most derivation

The right most non-terminal is substituted with corresponding production to derive a string

S -> aAB|a
A -> aBA|a
B -> b

Then RMD of aabab is S -> aAB -> aAb -> aaBAb -> aaBab -> aabab

Parse tree derivation

Defined in CFG page

Designing CFG

21 May 2021 08:31 AM

Rule 1

As with the design of finite automata, discussed in Section 1.1 (page 41), the design of context-free grammars requires creativity. Indeed, context-free grammars are even trickier to construct than finite automata because we are more accustomed to programming a machine for specific tasks than we are to describing languages with grammars. The following techniques are helpful, singly or in combination, when you're faced with the problem of constructing a CFG.

First, many CFLs are the union of simpler CFLs. If you must construct a CFG for a CFL that you can break into simpler pieces, do so and then construct individual grammars for each piece. These individual grammars can be easily merged into a grammar for the original language by combining their rules and then adding the new rule $S \rightarrow S_1 \mid S_2 \mid \cdots \mid S_k$, where the variables S_i are the start variables for the individual grammars. Solving several simpler problems is often easier than solving one complicated problem.

Rule 2 and 3

Second, constructing a CFG for a language that happens to be regular is easy if you can first construct a DFA for that language. You can convert any DFA into an equivalent CFG as follows. Make a variable R_i for each state q_i of the DFA. Add the rule $R_i \rightarrow aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in the DFA. Add the rule $R_i \rightarrow \varepsilon$ if q_i is an accept state of the DFA. Make R_0 the start variable of the grammar, where q_0 is the start state of the machine. Verify on your own that the resulting CFG generates the same language that the DFA recognizes.

Third, certain context-free languages contain strings with two substrings that are "linked" in the sense that a machine for such a language would need to remember an unbounded amount of information about one of the substrings to verify that it corresponds properly to the other substring. This situation occurs in the language $\{0^n 1^n \mid n \geq 0\}$ because a machine would need to remember the number of 0s in order to verify that it equals the number of 1s. You can construct a CFG to handle this situation by using a rule of the form $R \rightarrow uRv$, which generates strings wherein the portion containing the u's corresponds to the portion containing the v's.

Finally, in more complex languages, the strings may contain certain structures that appear recursively as part of other (or the same) structures. That situation occurs in the grammar that generates arithmetic expressions in Example 2.4. Any time the symbol a appears, an entire parenthesized expression might appear recursively instead. To achieve this effect, place the variable symbol generating the structure in the location of the rules corresponding to where that structure may recursively appear.

Ambiguity

21 May 2021 08:36 AM

Sometimes a grammar can generate the same string in several different ways. Such a string will have several different parse trees and thus several different meanings. This result may be undesirable for certain applications, such as programming languages, where a given program should have a unique interpretation.

If a grammar generates the same string in several different ways, we say that the string is derived *ambiguously* in that grammar. If a grammar generates some string ambiguously we say that the grammar is *ambiguous*.

DEFINITION 2.7

A string w is derived ***ambiguously*** in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ***ambiguous*** if it generates some string ambiguously.

- If there exist a string derived from the grammar which has more than one LMD or RMD or parse tree then that grammar is called ambiguous
- If every string derived from the grammar has exactly one LMD or RMD or parse tree then that grammar is called unambiguous
- If there does not exist any equivalent unambiguous grammar for an ambiguous grammar then such a grammar is called inherently ambiguous.

Chomsky Normal Form

21 May 2021 08:40 AM

DEFINITION 2.8

A context-free grammar is in ***Chomsky normal form*** if every rule is of the form

$$A \to BC$$
$$A \to a$$

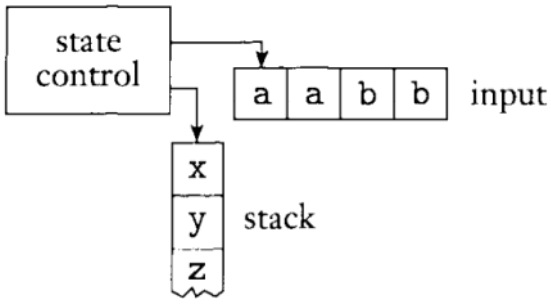
where a is any terminal and A, B, and C are any variables—except that B and C may not be the start variable. In addition we permit the rule $S \to \varepsilon$, where S is the start variable.

Rules for conversion to be studied

Pushdown Automata

22 May 2021 07:07 AM

A push down automata (PDA) for CFL is analogous to DFA or NDFA for regular languages.
A PDA uses a stack of unbound length where it can write on or read from
A PDA is more powerful than NFA or NDFA



Formal Definition

DEFINITION 2.13

A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Example

EXAMPLE 2.14

The following is the formal description of the PDA (page 110) that recognizes the language $\{0^n1^n \mid n \geq 0\}$. Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$,

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

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and

δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0			1			ε		
Stack:	0	$	ε	0	$	ε	0	$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

We can also use a state diagram to describe a PDA, as shown in the Figures 2.15, 2.17, and 2.19. Such diagrams are similar to the state diagrams used to describe finite automata, modified to show how the PDA uses its stack when going from state to state. We write "$a,b \to c$" to signify that when the machine is reading an a from the input it may replace the symbol b on the top of the stack with a c. Any of a, b, and c may be ε. If a is ε, the machine may make this transition without reading any symbol from the input. If b is ε, the machine may make this transition without reading and popping any symbol from the stack. If c is ε, the machine does not write any symbol on the stack when going along this transition.

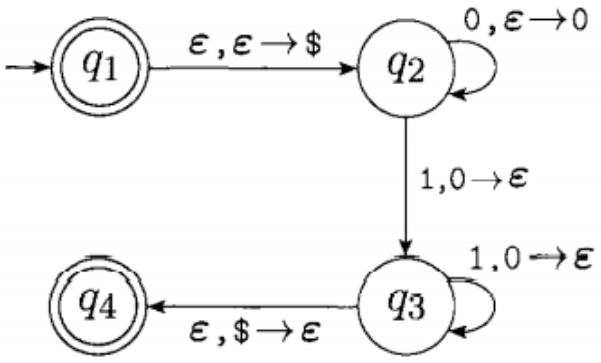


FIGURE 2.15
State diagram for the PDA M_1 that recognizes $\{0^n 1^n \mid n \geq 0\}$

Equivalence of PDA with CFL

22 May 2021 07:39 AM

Turing machine

21 May 2021 07:53 AM

A turing machine consist of a control(finite control) and an infinite tape with the head of the control pointing to a tape alphabet.

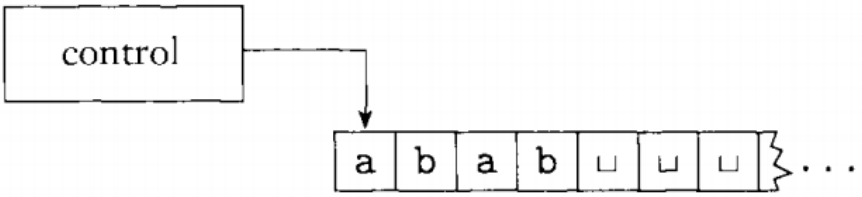


FIGURE 3.1
Schematic of a Turing machine

Differences between TM and other machines

- The TM can both write on tape and read from it
- A TM has infinite tape
- The head can move both left as well as right of the tape
- TM cant accept epsilon

Formal definition

DEFINITION 3.3

A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Transition function

$D(q,a) = (q_a,b,L)$ => when TM was at q state it read a tape alphabet a reached state q_a replaced a with b and moved to left

The current configuration of a TM is represented by uqv where it represents that tape string is uv TM is in q state with its head on the first letter of v

Language of TM

25 May 2021 09:24 AM

The collection of strings that is accepted by M is called the language of M or the language recognized by M.

Similarly a language accepted by TM is called TM recognizable

Given a language and TM there are three possible outcome the machine may reject , accept or loop continuously(does not halt)

It is some times difficult to tell the difference between machines that is simply taking long time and machine that is looping

TM that halts on all inputs or never loop[are called deciders, the language they accepts are called decidable

Variants of Turing machines

30 May 2021 08:26 AM

There are vast variety of turing machine all with same power some important ones are

Multitape turing machine
Nondeterministic turing machine
Enumerators

Church Turing thesis

30 May 2021 08:31 AM

Algorithms and turing algorithms are equal