Vendredi 9 juin 2023 — 13 Modeling Computation

Three types of structures used in models of computation : (I.) Grammars ← Grammars are used to generate words of a language and determine whether a word is in a language. Important in the construction and theory of compilers.
(II.) Finite-state machines (FSM) ← Set of states, input alphabet, transition function that assigns a next state to every pair of state and an input. The states of a FSM give it a limited memory capabilities.
Can be used to model many kinds of machines : Vending machines, delay machines, binary adders, language recognizers. (FSM without input but have final states)
Sets can be generated by a certain type of grammar
(III.) Turing machines ← Can be used to to recognize sets, to compute number-theoric functions.
Church-Turing thesis states that every effective computation can be carried out using a Turing machine. Turing machines are used to classify problems as (teractable vs intractable) and (solvable vs unsolvable)

13.1 Languages and Grammars

We are concerned only with the syntax or form and not its semantics or meaning of a sentence.

Two classes of problems that arise most frequently in applications to programming language:
(I.) How can we determine whether a combination of words is a valid sentence in a formal language?
(II.) How can we generate the valid sentence of a formal language?

Definition 1. V: vocabulary is a finite non-empty set of elements called symbols.
w: word or sentence over V is a string of finite length of elements of V
$\lambda|\varepsilon$: empty string or null string is the string containing no symbols. $\{\lambda\} \neq \emptyset$ the empty set. ($\{\lambda\}$ the set of empty string.)
V^*: set of all words over V a language over V is a subset of $V^* = \{w_0, w_1, \dots\}$
Rules for V^*:
T: terminals some of the elements of the vocabulary that cannot be replaced by other symbols
N: nonterminals members of the vocabulary, which can be replaced $N = V - T$
S: start symbol is a special member of V that we always begin with
P: productions is another set all strings of element in the vocabulary that specify when we can replace a string from V^* $z_0 \to z_1$

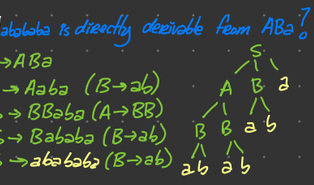
Definition 2. A phrase-structure grammar
$G = (V, T, S, P)$
* must contain at least one nonterminal on its left side

Definition 3. Let $w_0 \to l\, z_0\, r$, $w_1 \to l\, z_1\, r$
If $z_0 \to z_1$ a production of G, $w_0 \Rightarrow w_1$. (w_0 directly derivable)
If $w_0, w_1, \dots, w_n$ are strings over V such that $w_0 \Rightarrow w_1$, $w_1 \Rightarrow w_2$, $\dots$, $w_{n-1} \Rightarrow w_n$, we say that w_n is derivable from w_0
We write $w_0 \overset{*}{\Rightarrow} w_n$
The sequence of steps used to obtain w_n from w_0 is called a derivation

Definition 4. $L(G)$: the language generated by G or the language of G is the set of all strings of terminal that are derivable from the starting state S.
T^* : $T^* \subseteq V^*$
$L(G) = \{w \in T^* \mid S \overset{*}{\Rightarrow} w\}$

Example 1 | $G = (V, T, S, P)$, where $V = \{a, b, A, B, S\}$
$T = \{a, b\}$
$P = \{ S \to ABa,\ A \to BB,\ B \to ab,\ AB \to b \}$

Example 2 | abaBaba is directly derivable from ABa?
$S \to ABa$
$S \to Aaba$ $(B \to ab)$
$S \to BBaba$ $(A \to BB)$
$S \to Bababa$ $(B \to ab)$
$S \to abababa$ $(B \to ab)$



Example 3 | Let G be the grammar with $V = \{S, A, a, b\}$, $T = \{a, b\}$, S starting symbol, $P = \{ S \to aA \mid b,\ A \to aa \}$. What is $L(G)$, the language of this grammar?
From S,
$S \to aA$
$S \to aaa$ $(A \to aa)$
or $S \to b$
No additional words can be derived.
Hence, $L(G) = \{b, aaa\}$

Example 4 | Let G with $V = \{S, 0, 1\}$, $T = \{0, 1\}$, starting symbol S, $P = \{ S \to 11S \mid 0 \}$. What is $L(G)$, the language of this grammar?
From the start S,
$S \to 0$
or $S \to 11S$
$S \to 110$ $(S \to 0)$
or $S \to 11S$
$S \to 1111S$ $(S \to 11S)$
$S \to 1111\dots0$ $(S \to 0)$ ($(11)^{2n}$)
We surmise that $L(G) = \{0, 110, 11110, \dots\}$ The set of all strings that begin with an even number of 1s and end with a 0.

Example 5 | Give a phrase-structure grammar that generates the set $\{0^n 1^n \mid n = 0, 1, 2, \dots\}$ (The set of number of 0s and 1s are the same.)
$G = (V, T, S, P)$, where $V = \{S, 0, 1\}$, $T = \{0, 1\}$, starting state S, $P = \{ S \to 0S1 \mid \lambda \}$
Context-free language (Type 2)

Example 6 | Find a phrase-structure grammar to generate the set $\{0^m 1^n \mid m$ and n are nonnegative integers$\}$ (The number of 0s and 1s may differ)
$G_1 = (V_1, T_1, S_1, P_1)$ with $V_1 = \{S_1, 0, 1\}$, $T_1 = \{0, 1\}$, $P_1 = \{ S_1 \to 0S_1 \mid S_1 1 \mid \lambda \}$
Regular grammar (Type 3)
Or $G_2 = \{V_2, T_2, S_2, P_2\}$ where $V_2 = \{S_2, A, 0, 1\}$, $T = \{0, 1\}$, starting state S_2, $P_2 = \{ S_2 \to 0S \mid 1A \mid 1 \mid \lambda$, $A \to 1A \mid 1 \}$

Example 7 | $\{0^n 1^n 2^n \mid n = 0, 1, 2, 3, \dots\}$
$G = (V, T, S, P)$
where $V = \{0, 1, 2, S, A, B, C\}$
$T = \{0, 1, 2\}$
Starting state S
$P = \{ S \to C \mid \lambda,$
$C \to 0CAB$
$BA \to AB$
$0A \to 01$
$1A \to 11$
$1B \to 12$
$2B \to 22 \}$
Context-sensitive grammar (Type 1)

Type of phrase structured grammars:

(I.) Type 0: A grammar that has no restriction on its productions

(II.) Type 1: $w_1 \to w_2$

where $w_1 = lAr$, $w_2 = lwr$

and A: nonterminal symbol
l, r: are strings of 0 or more terminal or nonterminal symbols
w: is a nonempty string of terminal or nonterminal symbols

*If also have the production $S \to \lambda$ as long as S do not appear on the right-hand side of any other production.

Type 1 are called context-sensitive grammars because w_1 can be replaced by w_2 only when it is surrounded by the strings l and r. → Generate context-sensitive language

In other words, $w_1 \to w_2$, $l(w_1) \le l(w_2)$ and $S \to \lambda$ possible } Noncontracting or monotonic

Context-sensitive grammar have the most complicated definition. Noncontracting. It follows that the lengths of the strings in a deviation in a context-sensitive language are non-decreasing unless the production $S \to \lambda$ is used.
This means that the only way for the empty string to belong to the language is for the production $S \to \lambda$ to be part of the grammar.

(III.) Type 2: $w_1 \to w_2$, where w_1 is a single symbol that is not a terminal symbol. $w_1 \in N$

Type 2 are called Context-free grammar because a nonterminal symbol that is on the left side of a production can be replaced in a string whenever it occurs no matter what else is the string. → Generate a language that is called context-free language

(IV.) Type 3: $w_1 \to w_2$ with $w_1 \to A$ and either $w_2 = aB$ or $w_2 = a$, A,B nonterminal symbols a terminal symbol
or with $w_1 = S$ and $w_2 = \lambda$

Play important role in programming languages

In other words, $A \to aB \mid a$ or $S \to \lambda$

Type 3 are called regular grammars → Generate regular language.
*There's relationship between regular languages and FSM.

Regular grammar are used to search text for a certain pattern and in lexical analysis which is the process of transforming an input stream into a stream of tokens for use by a parser

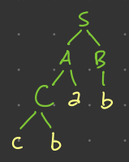
Derivation tree

A derivation in the language generated by a context-free grammar (type 2) can be represented graphically using an ordered root tree, called a derivation or parse tree
The root is the starting symbol,
The internal vertices are the nonterminal symbols that arise in the derivation
The leaves are terminal symbols that arise
The problem of determining whether a string is in the language generated by a context-free grammar arises in many applications, such as in the construction of compilers.

*There is another notation that is sometimes used to specify a type 2 grammar called the Backus-Naur form (BNF) it is used in the specification of the programming language ALGOL, Java, LISP, SQL, XML

Example 12 Determine whether the word cbab belongs to the language generated by the grammar $G = (V, T, S, P)$
where $V = \{a, b, c, S, A, B, C\}$, $T = \{a, b, c\}$

and $P = \{ S \to AB,\ A \to Ca,\ B \to Ba,\ B \to Cb,\ B \to b,\ C \to cb,\ C \to b \}$



13.3 Finite-State Machines with No Output

(Specifically designed for recognizing languages. Instead of producing output, these machines have final states.)

One of the most application of FSM is in language recognition (Design and construction of compilers for programming language)
A string is recognized iff it takes the starting state to one of its final states.

Definition 1 | Suppose that A and B are subset of V^* ($A, B \subseteq V^*$), where V is a vocabulary.
The concatenation of A and B is the set of all string of the form xy, where $x \in A$ and $y \in B$
(x string, y string)

Definition 2 | Suppose that A is a subset of V^*. Then the Kleene closure of A, denoted by A^* is the set consisting of concatenations of arbitrarily many strings from A.
That is,

$$A^* = \bigcup_{k=0}^{\infty} A^k = A^0 A^1 \cdots A^\infty$$

Definition 3 | A finite-state automaton $M = (S, I, f, s_0, F)$. ($G = (V, T, S, P)$)

- S: finite set of states
- I: finite input alphabet
- f: transition function ($f: S \times I \to S$)
- s_0: initial or start state
- F: subset of S consisting of final or accepting states.

We can represent finite-state automata using either state tables or state diagrams
Final states are indicated in state diagrams by using double circles

Formally, we can define the extended transition function f recursively for $M = (S, I, f, s_0, F)$

(I) $f(s, \lambda) = s$, $\forall s \in S$

(II) $f(s, xa) = f(f(s, x), a)$ $\forall s \in S, x \in I^*, a \in I$

Definition 4 | The language recognized or accepted by the machine M, denoted L(M), is the set of all strings that are recognized by M
Two finite-state automata are called equivalent if they recognize the same language.

Example 1 | Let $A = \{0, 11\}$ and $B = \{1, 10, 110\}$. Find AB and BA

$AB = \{01, 010, 0110, 111, 1110, 11110\}$
$BA = \{10, 111, 100, 1011, 1100, 11011\}$

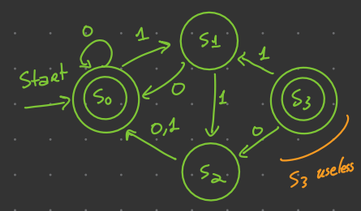
Example 2 | Let $A = \{1, 00\}$ Find A^n for n = 0, 1, 2 and 3.

$A^0 = \{\lambda\}$
$A^1 = A^0 A = \{\lambda\} A = \{1, 00\}$
$A^2 = AA = \{11, 100, 001, 0000\}$
$A^3 = A^2 A = \{111, 1100, 1001, 10000, 0011, 00100, 00001, 000000\}$

Example 3 | What are the Kleene closures of the sets $A = \{0\}$ $B = \{0, 1\}$ $C = \{11\}$?

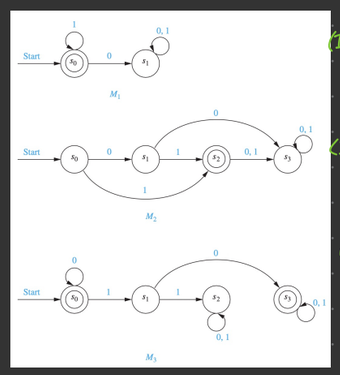
$A^* = \{0^n \mid n = 0, 1, 2, \ldots\}$
The Kleene closure of B is the concatenation of an arbitrary number of strings where each string is either 0 or 1. This is the set of all strings over the alphabet $V = \{0, 1\}$. That is $B^* = \{(0+1)^*\} = V^*$
$C^* = \{1^{2n} \mid n = 0, 1, 2, \ldots\}$

Example 4 | Construct the state diagram for $M = (S, I, f, s_0, F)$, where $S = \{s_i \mid i = 0, 1, 2, 3\}$, $I = \{0, 1\}$, $F = \{s_0, s_3\}$ and the transition function f



f	Input 0	Input 1
s_0	s_0	s_1
s_1	s_3	s_2
s_2	s_1	s_0
s_3	s_2	s_1

Example 5 | Determine the languages recognized by M_1, M_2, M_3



(I) $L(M_1) = \{1^n \mid n = 0, 1, 2, \ldots\} = \{1^*\}$

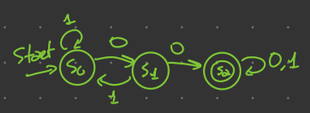
(II) $L(M_2) = \{1, 01\} = \{(1+01)\}$

(III) $L(M_3) = \{0^n, 0^n 10x \mid n = 0, 1, 2, \ldots$ and x is any string$\} = \{0^* + (0^* 10(0+1)^*)\}$

Example 6 | Construct deterministic finite-state automata that recognize each of the languages.

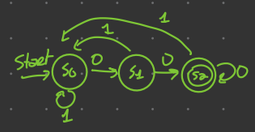
(a) the set of strings that begin with two 0s.

(b) the set of bit string that contain two consecutive 0s.



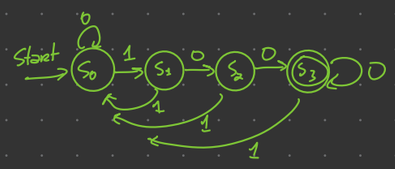
(c) do not contain two consecutive 0s

(d) end with two 0s

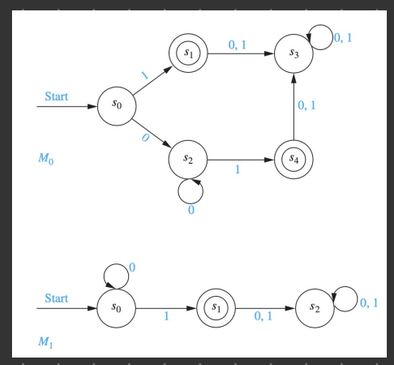


(e) contain at least two 0s

Example 7 | Construct a finite-state automata that recognizes the set of bit string that contain an odd number of 1s and that end with at least two consecutive 0s



Example 8 | Show that the M_0 and M_1 are equivalent.



$L(M_0) = \{1 + 0^*1\}$

$L(M_1) = \{0^*1 + 1\}$

$L(M_0) = L(M_1) \Rightarrow M_0$ and M_1 are equivalent

Algorithms used to construct finite-state automata to recognize certain languages may have many more states than necessary

Using unnecessarily large finite-state machine to recognize languages can make both hardware and software applications inefficient and costly
This problem arises when finite-state automata are used in compilers, which translate computer programs to a language a computer can understand (object code)

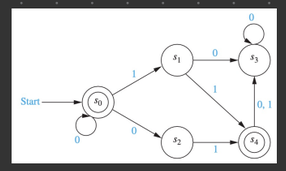
machine minimization

Reduces the number of states by replacing states with equivalence classes of states with respect to an equivalence relation in which two states are equivalent if every input string either sends both states to a final state or sends both to a state that is not final.

* Before the minimizing procedure begins, all states that cannot be reached from the start state using any input string are first Removed.

Definition 5 Nondeterministic automaton $M = (S, I, f, s_0, F)$ Where $f: S \times I \to P(S)$

Example 11 Find the language recognized by the following nondeterministic finite-state automaton.



$L(M) = \{0^n, 0^n01, 0^n11 \mid n \geq 0\}$
$= \{0^* + 0^*01 + 0^*11\}$
$= \{0^*(\lambda + 01 + 011)\}$

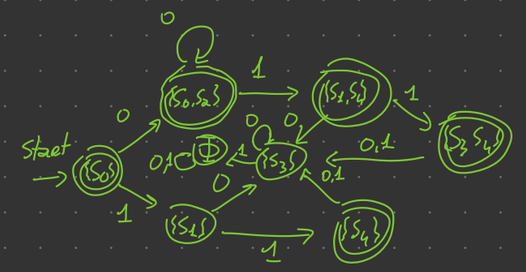
Theorem 1 If the language L is recognized by a nondeterministic finite-state automaton M_1, then L is also recognized by a deterministic finite-state automaton M_2.

Example 12 Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in example 10.

State	f Input 0	1
s_0	s_0, s_2	s_1
s_1	s_3	s_4
s_2	$\emptyset$	s_4
s_3	s_3	$\emptyset$
s_4	s_3	s_3

→

State	f input 0	1
$\{s_0\}$	$\{s_0, s_2\}$	$\{s_1\}$
$\{s_0, s_2\}$	$\{s_0, s_2\}$	$\{s_1, s_4\}$
$\{s_1\}$	$\{s_3\}$	$\{s_4\}$
$\{s_1, s_4\}$	$\{s_3\}$	$\{s_3, s_4\}$
$\{s_3\}$	$\{s_3\}$	$\emptyset$
$\{s_4\}$	$\{s_3\}$	$\{s_3\}$
$\{s_3, s_4\}$	$\{s_3\}$	$\{s_3\}$



Final states: $\{s_0\}, \{s_0, s_2\}, \{s_1, s_4\}$
$\{s_4\}, \{s_3, s_4\}$

13.4 Language Recognition

Stephen Kleene showed that there is a finite-state automaton that recognizes a set if and only if this set can be built up from the null set, the empty string, and singleton strings by taking concatenations, unions, and Kleene closures, in arbitrary order. Sets that can be built up in this way are called **regular sets**.

Definition 1. The **regular expressions** over a set I are defined recursively by:
- (I_R) the symbol $\emptyset$ is a regular expression (empty set, with no strings)
- (II_R) the symbol λ is a regular expression (set containing the empty string, $\{\lambda\}$)
- (III_R) the symbol x is a regular expression whenever $x \in I$ ($\{x\}$ one symbol x)
- (IV_R) the symbols (AB), $(A \cup B)$, and A^* are regular expressions whenever A and B are regular expressions.

Theorem 1. Kleene's Theorem
A set is regular if and only if it is recognized by a finite-state automaton.

Theorem 2. A set is generated by a regular grammar if and only if it is a regular set

Example 1 What are the strings in the regular sets specified by the regular expression 10^*, $(10)^*$, $0 \cup 01$, $0(0 \cup 1)^*$, and $(0^*1)^*$?

TABLE 1

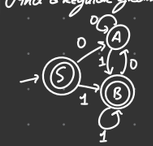
Expression	*Strings*
10^*	a 1 followed by any number of 0s (including no zeros)
$(10)^*$	any number of copies of 10 (including the null string)
$0 \cup 01$	the string 0 or the string 01
$0(0 \cup 1)^*$	any string beginning with 0
$(0^*1)^*$	any string not ending with 0

Example 2 Find a regular expression that specifies each of these sets:
(a) the set of bit strings with even length. $(00 \cup 01 \cup 10 \cup 10)^*$
(b) the set of bit strings ending with a 0 and not containing 11. $(0 \cup 10)^*(0 \cup 10)$
(c) the set of bit strings containing and odd number of 0s. $1^*01^*(01^*01^*)^*$

Example 3 Construct a nondeterministic finite-state automaton that recognizes the regular set 1^*01

Example 4 Construct a nondeterministic finite-state automaton that recognizes the language generated by the regular grammar $G = (V, T, S, P)$, where $V = \{0, 1, A, S\}$, $T = \{0, 1\}$, and the productions in P are $S \to 1A \mid 0 \mid \lambda$, $A \to 0A \mid 1A \mid 1$

Example 5 Find a regular grammar that generates the regular set recognized by the finite-state automaton:



The grammar $G = (V, T, S, P)$ generates the set recognized by this automaton where
$V = (0, 1, S, A, B)$,
$T = \{0, 1\}$,
S is the start symbol,
and $P = \{ S \to 0A \mid 1B \mid 1 \mid \lambda$,
$A \to 0A \mid 1B \mid 1$,
$B \to 0A \mid 1B \mid 1$
$\}$