Book - Daniel Cohen.

DATE: 28/4/07
PAGE NO: 01

Part I - Automata Theory

Unit I - Basic concepts
Finite Automata (FA)

Unit II - Regular Expressions & Languages.

Chapters → 1 - 11

Ch.1 Background

Theory of computers — Form several mathematical model that will describe with varying degrees of accuracy, parts & types of computers & similar machines.

Mathematical model, is a game that describes some important real world behaviour.
M.M's abstract, simplify & codify to the point that the observations & conclusions that can be made about the game relate back in meaningful way to the physical world, shedding light on that which was not obvious before.

ex. chess is a mathe. m. for war, but it's a poor m. because wars are not really won by simple assassination of the leader of the opposing country.

We discover whether certain tasks can be done at all. Conclusions will be this can be done or this can never be done. (in future as well).

M.M. serve a practical purpose thro' their application to computer science, both in ① development of structures & techniques necessary & useful to comp^r. program & ② in engineering of comp^r architecture.

M.M. are called machines, we study their limitations by

analyzing the types of inputs on which they operate successfully.
Collection of these successful i/ps is called the language of the m/c. [Human can understand instructions in one language, not other].

For a new m/c, we learn its lang., & for a new lang., we find m/c that corresponds to it.

Linguistics — Questions - What is lang. & How it was developed &. How people understand it & How do people construct sentences from the ideas in their mind &.

Noam chomsky created the subject of M.M. for description of languages to answer these questions. Theory began to shed light on study of comp^n. lang. We don't know how humans understand lang. but we know how m/c's digest what they are told. Thus formulations of mathe. logic became useful to linguistics.
Thus comp^n. took linguistic abilities — Word proc^r, translator, interpreter of simple grammer, compiler of comp^n. lang.

Sub includes - Theory of ① Automata
② formal lang.
③ Turing m/cs.

In comp^n / computation theory — items that are central to it are types of tasks (algo. or prog.) that can be performed, not the mechanical nature of the physical comp^r. itself.
Comp^n. theory

We study diff. types of theoretical m/c's that are mathe. models for actual physical proc. By considering the possible inputs on which these m/c's can work, we can analyse their strengths & weaknesses.

Unit II

DATE: / /
PAGE NO.: 02

Ch. 2 Languages.

English – Letters, words, sentences.
Not all collections of letters form a valid word &
" " " " words " " sent.

Comp^{n}. Lang. –
certain char. strings → recognizable words (DO, IF, END, ...)
" str. of words → " commands
" set of cmds → prog.
prog. compilation → m/c cmds.

Def^{n} of lang. stru. → there should be rules for recognizing whether an i/p is a valid comm^{n}.
It is imp. that the prog. compiles whether or not it does what the programmer intended. If it compiles, it was a valid example of a statement or comm^{n}. in the lang. & the m/c is responsible for executing the specified sequence of instrns. We are looking for ways of determining whether the i/p is a valid comm^{n}.

(informal)
For spoken English, it is hard to state all rules, since sentences with grammatical errors are understandable.

Theory of formal languages – All rules are explicitely stated in terms of what strings of symbols can occur. No liberties are tolerated. Lang. is considered as symbols on paper & not as expressions of ideas in human minds. Lang. is not comm^{n} among intellects but the game of symbols with basic formal rules, i.e. the form of the string of symbols we are interested in, not the meaning.

Alphabet – one finite set of fundamental units (symbols), out of which we build structures. [need not be Latin letters]
Language – certain specified set of strings of char. from the alphabet.
Word – permissible string in the lang., contains finitely many symbols.
"Specify" the set of strings – is the main issue.

Empty / Null string (Λ) — string with no letters.
or NULL word (Λ).

2 words are same if all their letters are same & in same order.

Lang. with no words = null set ϕ
It is not true that Λ is a word in lang. ϕ, since this lang. has no words at all.

If a certain lang. L does not contain word Λ & we wish to add it to L, we use the union of sets operation denoted by + to form $L + \{\Lambda\}$. This lang. is not the same as L.
$L + \phi$ is same as L since no new words have been added.

If we have a method for producing a lang. & in certain instance the method produces nothing, we can say either that the method failed or it successfully produced the lang. ϕ.

Eg :- lang. ENGLISH-WORDS = { all the words in a std dictionary }
This is a finite list.
If we tried to define a lang. of infinitely many words by an infinite list, there is problem of impossibility of searching this list (alphabetically ordered) to determine whether a given word is in the lang. or not.
We don't allow the possibility of defining a lang. by an infinite dictionary.
To know all the words, does not imply the ability to create a viable sentence.
lang. ENG-WORDS does not have any grammer.
To define lang. of the Eng. sentences, alphabet is the entries in the dictionary.
Γ = { the entries in a std. dictionary, plus a blank space, capital gamma plus usual punctuation marks }.

To specify which strings of elements from Γ produce valid words in the lang. ENG-SENTENCES, we must rely on grammatical rules.
Theoretically, there are infinitely many diff. words in the lang. ex.

I ate one apple.
-11- two " s.
-11- three "
....

The trick of defining the lang. by listing all rules of grammer allows us to give a finite description of an infinite lang.

I ate three Tuesdays. - gramatically correct.
∴ allowed in formal lang.,
meaning is not considered in f. lang.
consider syntax, not semantics.

Abstract lang. will be defined in 2 ways -
① presented as an alphabet & exhaustive list of all valid words.
② presented as an alphabet & set of rules defining the acceptable words.

To be an acceptable specification of lang, set of rules must enable us to decide, in a finite amount of time, whether given string of alphabet letters is or is not a word in the lang.
It is not needed that all the letters in the alphabet need to appear in the words selected for the lang.

ex. lang. MY-PET, alphabet = {a c d g o t}
∅ The only word in the lang. is specified by:
If the Earth & the Moon ever collide, then
MY-PET = {cat}
but if they never collide, then
MY-PET = {dog}.
We can't decide which word will be in the lang.

DATE: / /
PAGE NO:

Intro to Defining Lang.

The set of lang. defining rules can be of 2 kinds :-

① they tell us how to test the given string for valid word.

② how to construct all the words in the lang. by some clear procedure.

* 1. ex. alphabet, $\Sigma = \{x\}$.

rule : any nonempty string of alphabet characters is word;

$L_1 = \{x, xx, xxx, \dots\}$ comma - optional.

or $L_1 = \{x^n \text{ for } n = 1, 2, 3, \dots\}$.

for L_1 we can define the operation of concatenation, concat

x^n concatenated with x^m is the word x^{n+m}

It is not true always that when 2 words are concatenated they produce another word in the lang.,

2. ex. $L_2 = \{x, xxx, xxxxx, \dots\}$

$= \{x^{odd}\}$

$= \{x^{2n+1} \text{ for } n = 0\ 1\ 2\ 3\}$

then,

$a = xxx$ & $b = xxxxx$ are both words in L_2, their concatenation $ab = xxxxxxxx$ is not in L_2.

For L_1 & L_2, $ab = ba$, but this does not hold for all lang.,

ex. house + boat = houseboat } diff. words.
boat + house = boathouse }

3. ex. $\Sigma = \{0, 1, 2, 3, \dots, 8, 9\}$

$L_3 = \{$any finite string of alphabet letters that does not start with the letter zero$\}$.

$L_3 = \{1, 2\ 3\ 4\ \dots\ 9\ 10\ 11\ 12\ \dots\}$

L_3 looks like set of all positive integers written in base 10. it is just a formal collection of strings of symbols

To define L_3 to include word 0,

L_3 = { any finite string of alphabet letters that, if it starts with a 0, has no more letters after first }

Function length –

length (0 x x x) = 3

length (428) = 3

$l(\wedge) = 0$

if $l(w) = 0$, then $w = \wedge$.

(w = word)

L_3 = { any finite string of alphabet letters that, if it has length more than 1, does not start with a 0 }.

There are diff. ways of specifying the same lang.

There is ambiguity in the phrase "any finite string", whether to include null string ($\wedge$).

L_3 does not include $\wedge$, ∵ lang. should look like the integers, & there is no integer with no digits.

Define L_1 to contain $\wedge$:

$L_4 = \{\wedge$ or x or xx or xxx ... $\}$

$= \{x^n$ for $n = 0, 1, 2, 3 \ldots\}$

$x^0 = \wedge$, not $x^0 = 1$ as in algebra. x^n is string of n x's.

In L_3, don't confuse with 0, which is a string of length 1, with $\wedge$.

Even when $\wedge$ is a word in the lang., it is not a letter in the alphabet.

DATE: / /
PAGE NO:

Fun. reverse –
reverse(xx) = xx
rev(145) = 541
but in L_3,
rev(140) = 041
which is not a word in L_3.

Lang. Palindrome,
$\Sigma = \{a\ b\}$
palin. = { ∧, and all strings x such that reverse(x) = x }.

= { ∧ a b aa bb aaa aba bab bbb aaaa abba ... }

Concatenation –
abba + abbaabba is palindrome,
but aa + aba is not.

KLEENE CLOSURE/Star (Σ^*)
Given an alphabet Σ, we wish to define a lang. in which any string of letters from Σ is a word, even the null string. This lang. is called closure of the alphabet. It is denoted by writing a star (an asterisk) after the name of the alphabet as a superscript.
Σ^* – known as Kleene star after the logician.

ex.1. If $\Sigma = \{x\}$ then
$\Sigma^* = L_4 = \{ \wedge\ x\ xx\ xxx\ xxxx\ ... \}$

ex.2. If $\Sigma = \{0\ 1\}$ then
$\Sigma^* = \{ \wedge\ 0\ 1\ 00\ 01\ 10\ 11\ 000\ 001\ ... \}$

ex.3. If $\Sigma = \{a\ b\ c\}$ then
$\Sigma^* = \{ \wedge\ a\ b\ c\ aa\ ab\ ac\ ba\ bb\ bc\ ca\ cb\ cc\ aaa ... \}$

Kleene star is an operation that makes an infinite lang. out of alphabet. Infinite lang. means infinitely many words, each of finite length.

Lexicographic ordering of words —
consider size of words & then alphabetical order.

If listed in alphabetical order,
$\wedge$ a aa aaa aaaa ...
would not explain real nature of lang.

Use of star operator to sets of words (S)
S^* is the set of all finite strings formed by concatenating words from S, where any word may be used as often as we like, & null str. is also included.

Do not confuse 2 lang.
ENGLISH-WORDS* & ENG-SENTENCES.
1st contains word butterbutterbutterhat, 2nd does not.
1st - concatenation - arbitrarily
2nd - grammer rules.

ex. Let $S = \{a\ ab\}$ then
$S^* = \{\wedge$ plus any word composed of factors of a & ab$\}$
$= \{\wedge$ plus all strings of a's & b's except those that start with b & those that contain a double b$\}$
$= \{\wedge\ a\ aa\ ab\ aaa\ aab\ aba\ aaaa\ aaab\ aaba\ abaa\ abab\ aaaaa\ aaaab\ aaaba\ aabaa\ aabab\ abaaa\ abaab\ ababa \ldots\}$

To prove that a certain word is in S^*, we must show how it can be written as a concatenate of words from S.
ex. ababab = (ab)(a)(ab). — unique factoring

Λ is always a word in Kleene
K.C. is used to produce

ex. factoring is not unique.

$S = \{xx\ xxx\}$

$S^* = \{\Lambda$ & all string of more than one $x\}$

$= \{\Lambda\ xx\ xxx\ xxxx\ xxxxx\ xxxxxx\ \ldots\}$

x is not in S^*.

$xxxxxxx = (xx)(xx)(xxx)$

$= (xx)(xxx)(xx)$

$= (xxx)(xx)(xx)$

Here ( ) are not letters in alphabet.

$\Sigma = \{x\ (\ )\}$

length $(xxx) = 5$

len $((xx)(xxx)) = 9$.

If the alphabet has no letters, then its closure is the lang. with the null string as its only word, because Λ is always a word in a Kleene closure.

If $\Sigma = \phi$ (the empty set),

then $\Sigma^* = \{\Lambda\}$

This is not the same as

If $S = \{\Lambda\}$,

then $S^* = \{\Lambda\}$.

which is also true but for a diff. reason, i.e. $\Lambda\Lambda = \Lambda$

The Kleene closure always produces an infinite lang.

unless the underlying set was one of the two examples above.

Kleene closure is the method of defining lang. that works only for infinite lang.

K.C. of 2 sets can end up being the same lang even if the 2 sets were not.

ex, consider 2 lang.

$S = \{a\ b\ ab\}$, $T = \{a\ b\ bb\}$.

DATE: / /
PAGE NO: 7

Ch. 3 Recursive Definitions.

⇒ A new method for defining Languages - (or sets).

3 steps -

① specify some basic objects in the set.

② give rules for constructing more objects in the set from the ones we already know.

③ to declare that no objects except those constructed in this way are allowed in the set.

✓ ex. Define set of positive even integers.

Def^n 1. → EVEN is the set of all positive whole numbers divisible by 2.

Def^n 2 → " 2n where n = 1, 2, 3, ...

Def^n 3 → The set EVEN is defined by foll-3 rules:

Rule 1 - 2 is in EVEN

2 - If x is in EVEN, then so is x+2.

3 - The only elements in the set EVEN are those that can be produced from the 2 rules above.

Def^n 3 is less popular than others; ∵ it is harder to use in most practical applications.

Prove that 14 is in EVEN. Proof by def^n 3 is lengthier.

Def^n 4. → Rule 1 - 2 is in EVEN.

2 - If x & y are both in EVEN, then so is x+y.

(x & y can be equal).

2 is in even, 2+2 = 4, 2+4 = 6, 4+4 = 8 ... is in even.

Set EVEN has def^n 1 & 2 as fine def^ns that are not recursive. There are certain sets that have no better def^n than the recursive one.

Although the second recursive def^n is harder to use (in proving that given numbers are even) than 2 non-rec. def^ns, it has some advantage.

DATE: / /
PAGE NO.:

Suppose we want to prove that the sum of 2 numbers in EVEN is also a number in EVEN. This is a trivial conclusion from 2nd def^n But to prove this from 1st def^n is harder.

Whether or not we want a rec. def^n depends on recursion or not & things: i) how easy the other possible def^{ns} are to understand, ii) what types of theorems we may wish to prove about the set.

✓ Ex. Rec. def^n of positive integers:
Rule 1: 1 is in INTEGERS.
2: If x is in INT, then so is $x+1$.

✓ Ex. Def^n of INtegers:
Rule 1: 1 is in INT.
2: If both x & y are in INT, then so are $x+y$ & $x-y$.
($1-1=0$, & for all x, $0-x=-x$)

✓ Ex. Def^n of POLYNOMIAL:
Rule 1: Any number is in POLY.
2: Variable x is in POLY.
3: If p & q are in POLY, then so are $p+q$, $p-q$, (p) & pq. (pq is multiplic^n)
Some sequence of applic^{ns} of these rules can show that $3x^2+7x-9$ is in POLY.

✓ Ex. Rule 1: x is in L_1.
2: If w is any word in L_1, then xw is also in L_1.
$L_1 = x^+ = \{x \; xx \; xxx \; \ldots\}$.

✓ 1: Λ is in L_4.
2: If x is any word in L_4, then xw is also in L_4.
$L_4 = x^* = \{\Lambda \; x \; xx \; xxx \; \ldots\}$.

1: x is in L_2

2: If xw is in L_2

$L_2 = \{x^{odd}\} = \{a\ aaa\ aaaaa \ldots\}$

1: 1 2 3 ... 9 are in INT.

2: $w_0\ w_1 \ldots w_9$ are in INT.

✓ Kleene closure.

1: If S is a lang, then all words of S are in S^*.

2: Λ is in S^*.

3: If x & y are in S^* then so is their concatenation xy.

Def^n for Arithmatic Expressions –

$\Sigma = \{0\ 1\ 2\ 3 \ldots 9\ .\ +\ -\ *\ /\ (\ )\}$

Foll: strings are not good:

$(3+5)+6)$ $2(18+9)$ $(3+(4-)8)$ $2)-(4$

Rule 1: Any number (+ve, -ve, or zero) is in AE.

2: If x is in AE, then so are

(i) (x)

(ii) $-x$ (provided x does not already start with a minus sign)

3: If x & y are in AE, then so are:

(i) $x+y$ (if first symbol in y is not + or -)

(ii) $x-y$ (—— " ——))

(iii) $x*y$

(iv) x/y

(v) $x^{**}y$ (exp).

ex: string 8/4/2 is in AE, but its meaning is doubtful.

Theorems: An AE can't contain the char. $.$.

: No AE can begin or end with the symbol /.

: No AE can contain the substring //.

DATE: / /
PAGE NO.:

Ch 4. Regular Expressions.

✓ New method to define lang.

Be careful about the phrases we use to define lang.

ex. ✓ $L_1 = \{x^n \text{ for } n = 1\ 3\ 5\ 7 \cdots\}$.

$L_2 = \{x^n \text{ for } n = 1\ 4\ 9\ 16 \cdots\}$.

✓ 3 4 8 22 ...

Difficult to find words in the lang.

In computers, less guesswork is needed. ∴ we develop some new lang. defining symbolism, much more precise than ellipsis.

✓ ex. $L_4 = \{\Lambda\ x\ xx\ xxx\ xxxx \cdots$

Indicate this set as the closure of a smaller set.

Let $S = \{x\}$, then $L_4 = S^*$.

$L_4 = \{x\}^*$.

Apply Kleene star not to a set, but directly to the letter.

& written as superscript as if it were an exponent.

x^*.

$x^* = \Lambda$ or x or x^2 or x^3 or ...

$= x^n$ for some $n = 0\ 1\ 2\ 3 \cdots$.

✓ $L_4 = \text{lang}(x^*)$

lang. lang. defining symbol.

ex. $L = \{a\ ab\ abb\ abbb\ abbbb \cdots\}$.

✓ $L = \text{lang}(ab^*)$

No string can contain a blank unless it is a character in alphabet Σ.

✓ $(ab)^* = \Lambda / ab / abab / ababab \cdots$

✓ $L_1 = \text{lang}(xx^*) = \text{lang}(x^+)$.

L_1 can be defined by,

✓ $xx^*\ x^+\ x\,x^*x^*\ x^*x\,x^*\ x^+x^*\ x^*x^+\ x^*x^*x^*x\,x^*$

✓ ex. Set of all strings of a's & b's that have at least 2 letters, that begin & end with a's & have some b's inside. $= ab^*a$

lang. $(ab^*a) = \{aa\ aba\ abba\ abbba \cdots\}$.

It is wrong to say that set of all words that

begin & end with a & have b's inbetween, because this may apply to word a, depending on interpret. Our symbolism eliminates this ambiguity.

✓ ex. lang. of expr. a^*b^* contains all str. of all a's come before all b's.

$= \{\Lambda\ a\ b\ aa\ ab\ bb\ aaa\ aab\ abb\ bbb\ aaaa \cdots\}$.

$a^*b^* \neq (ab)^*$.

✓ ex. $L_2 = \{x^{odd}\}$

$= x(xx)^*$ or $(xx)^*x$

$\neq x^*$ or x^* ∵ it includes $(xx)x(x)$.

✓ $x+y$ means either x or y where x & y are strings of char. from an alphabet. Don't confuse this with $+$ as an exponent.

✓ ex. let $\Sigma = \{a\ b\ c\}$

$T = \{a\ c\ ab\ cb\ abb\ cbb\ abbb\ cbbb\ abbbb \cdots\}$

$T = lang((a+c)b^*)$.

$=$ lang (either a or c then some b's).

can be zero b's.

It defines the lang. as follows: For each * or + used as a superscript, we must select some number of factors for which it stands. For each other +, we choose either left or right expression. for every set of choices, we have generated a particular string. The set of all str. that can be produced by this method is the lang. of the expression.

ex. Consider a finite lang. L that contains all strings of a's & b's of length 3 exactly:

$L = \{aaa\ aab\ aba\ abb\ baa\ bab\ bba\ bbb\}$.

1st letter of each word is a or b, 2nd is a/b, 3rd is a/b.

∴ $L = lang((a+b)(a+b)(a+b))$

or L = lang $(a+b)^3$.

ex. To define set of all possible strings of a's & b's, $(a+b)^+$; ∧ str. of any length, $(a+b)^*$.

ex. All words that begin with an a & end with a b, $a(a+b)^*b$ = a (arbitrary string) b.

Regular expression Def.ⁿ – (RE)

The symbols that appear in RE's are the letters of the alphabet Σ, the symbol for null string ∧, parentheses, star operator & plus sign.

[Lang. defined by RE. is called regular lang.]

Set of RE's is defined by foll. rules:

Rule 1: Every letter of Σ can be made into a RE by writing it in boldface; ∧ itself is a RE.

Rule 2: If r_1 & r_2 are RE's, then so are:

(r_1)

$r_1 r_2$

$r_1 + r_2$

r_1^*

Rule 3: Nothing else is a RE.

[$r_1^+ = r_1 r_1^*$, ∴ + is not included.].

This is lang. of lang. definers. ex. "French" is a word (adjective) & a lang. defining name (noun).

Rule 1 → confusion – a → whether letter in Σ, a, the word in Σ^*, {a}, the one word lang, or **a**, the RE for lang. Context & typography will guide us.

Parentheses is optional but is required for clear meaning. If $r_1 = aa+b$ then r_1^* technically is $aa+b^*$ (triple concatenation). but we actually mean

$r_1^* = (r_1)^* = (aa+b)^*$ – both are diff. ( ) is needed.

Boldface lambda (Λ) means RE defining word lambda.
" phi (ϕ) " " null lang

For any r we have
r + ϕ = r (B) bold.
ϕ r = ϕ (B)

ϕ* - we avoid this.

Q. RE - (a+b)* a (a+b)*
Set of all words over Σ = {a b} that have an a.
Words left are without a's (Λ & strings of b's).
Combination produces lang. of all strings.
all str. = (all str. with an a) + (all str w/o an a)
(a+b)* = (a+b)* a (a+b)* + b*
is union of 2 lang.
+ means choice.

Note -
a* = a* + a*
= a* + a* + a*
= a* + aaa.
} use of + as choice.

Ex. lang. of all words that have at least 2 a's -
(a+b)* a (a+b)* a (a+b)*.
OR b* ab* a (a+b)* or (a+b)* a b* a b* or
b* a (a+b)* ab*
∴ lang ((a+b)* a (a+b)* a (a+b)*) = lang (b* a b* a (a+b)*)

2 expressions are equivalent if they describe same lang.

All words with exactly 2 a's,
RE - b* a b* a b*.

a* b* + b* a* - bad, a

All words have at least 1 a & at least 1 b.
(a+b)* a (a+b)* b (a+b)* + (a+b)* b (a+b)* a (a+b)*
This means words all words of form some b's followed by some a's, then we can add this by b b* a a*.

$(a+b)^* a (a+b)^* b (a+b)^* + bb^* a a^*$.

ex. words that do not contain both a & b are all a's, all b's, or Λ. When these are included, we get everything.

$\therefore (a+b)^* a (a+b)^* b (a+b)^* + bb^* a a^* + a^* + b^*$
$= (a+b)^*$

defines all str of a's & b's.

Note - don't misinterpret the fact that every RE defines some lang. to mean that the associated lang. has a simple English description. RE itself may be the simplest description.

ex. $(\Lambda + ba^*)(ab^*a + ba^*)^* b (a^* + b^*a) ba b^*$.

has no simple alternate characterization. There is no algo. to discover hidden meaning.

Foll. equivalences show that we can't treat RE's as algebraic polynomials.

$(a+b)^* = (a+b)^* + (a+b)^*$
$= (a+b)^* + a^*$
$= (a+b)^* (a+b)^*$
$= a(a+b)^* + b(a+b)^* + \Lambda$
$= (a+b)^* ab (a+b)^* + b^* a^*$

words containing substring ab. ↓

↳ words don't contain ab. all a's, all b's, Λ, b's followed by a's.

With star operator, we define an infinite lang. We represent a finite lang. by using plus sign (union) alone.

if $L = \{\Lambda \; a \; aa \; bbb\}$
then symbolic expression for L must be,
$L = \text{lang.}(\Lambda + a + aa + bbb)$