Why Study the Theory of Computation?

Implementations come and go.

Chapter 1and 2

IBM 7090 Programming in the 1950's

```
ENTRY       SXA     4,RETURN
            LDQ     X
            FMP     A
            FAD     B
            XCA
            FMP     X
            FAD     C
            STO     RESULT
RETURN      TRA     0
A           BSS     1
B           BSS     1
C           BSS     1
X           BSS     1
TEMP        BSS     1
STORE       BSS     1
            END
```

$Ax^2 + Bx + C$

Programming in the 1970's
IBM 360 JCL (Job Control Language)

```
//MYJOB    JOB (COMPRESS),
          'VOLKER BANDKE',CLASS=P,COND=(0,NE)
//BACKUP  EXEC PGM=IEBCOPY
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=MY.IMPORTNT.PDS
//SYSUT2   DD  DISP=(,CATLG),
           DSN=MY.IMPORTNT.PDS.BACKUP,
//         UNIT=3350,VOL=SER=DISK01,
//         DCB=MY.IMPORTNT.PDS,
           SPACE=(CYL,(10,10,20))
//COMPRESS EXEC PGM=IEBCOPY
//SYSPRINT DD  SYSOUT=*
//MYPDS    DD  DISP=OLD,DSN=*.BACKUP.SYSUT1
//SYSIN    DD  *
COPY INDD=MYPDS,OUTDD=MYPDS
//DELETE2 EXEC PGM=IEFBR14
//BACKPDS  DD  DISP=(OLD,DELETE,DELETE),
          DSN=MY.IMPORTNT.PDS.BACKUP
```

Guruhood

```
(⌈/V)>(+/V)-⌈/V
```

IBM’s APL Language – Returns 1 if the largest value in a 3 element vector is greater than the sum of the other 2 and Returns 0 otherwise

APL was very powerful for processing arrays & vectors

Why study this?

Science of Computing

- Mathematical Properties (problems & algorithms) having nothing to do with current technology or languages
- E.G. Alan Turing – died 1954
- Provides Abstract Structures
- Defines Provable Limits
  - Like "Big Oh"

Goals

Principles of Problems:

- Does a solution exist?
  - If not, is there a restricted variation?
- Can solution be implemented in fixed memory?
- Is Solution efficient?
  - Growth of time & memory with problem size?
- Are there equivalent groups of problems?

Applications of the automata Theory

- Used in design of Lexical analyzer of compilers which breaks source program into tokens like identifies, Keywords etc..
- Software for designing and checking the behavior of the Digital circuits.
- FSMs (finite state machines) for vending machines, Traffic signals, communication protocols, & building security devices.
- String Matching: searching words, phrase and other pattern in large bodies of text(like web pages)
- Interactive games as nondeterministic FSMs.
- Used in Natural languages processing: for speech to text and text to speech conversions.
- Artificial Intelligence: Medical Dignosis,Factory Scheduling etc..

Languages and Strings

This is one of MOST important chapters.
It includes the TERMINOLOGY required to be successful in this course.
KNOW this chapter & ALL DEFINITIONS!!

Chapter 2

A Framework for Analyzing Problems

We need a single framework in which we can analyze a very diverse set of problems.

The framework is

Language Recognition

*A ***language*** is a (possibly *infinite*) set of *finite* length strings over a *finite* alphabet.

NOTE: Pay particular attention to use of *finite* & *infinite* in all definitions!

Alphabet - Σ

- An alphabet is a non-empty, finite set of characters/symbols
- Use Σ to denote an alphabet
- Examples

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

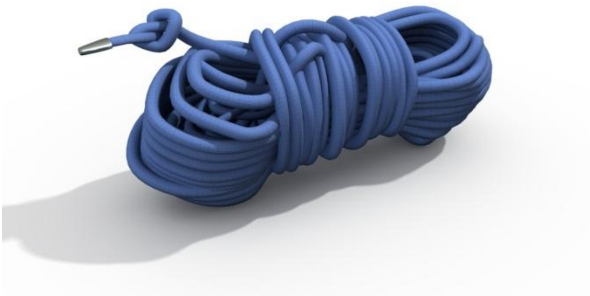
$\Sigma = \{ 0, 1, 2 \}$

$\Sigma = \{ a, b, c, \ldots z, A, B, \ldots Z \}$

$\Sigma = \{ \#, \$, *, @, \& \}$

Strings

- A ***string*** is a finite sequence, possibly empty, of characters drawn from some alphabet Σ.
- ε is the empty string
- Σ* is the set of all possible strings over an alphabet Σ.



Example Alphabets & Strings

Alphabet name	***Alphabet symbols***	***Example strings***
The lower case English alphabet	{a, b, c, …, z}	ε, aabbcg, aaaaa
The binary alphabet	{0, 1}	ε, 0, 001100, 11
A star alphabet	{★ , ✪ , ✬ , ✯, ✡, ✰}	ε, ✪✪, ✪✯✯✰★✰
A music alphabet	{𝅝, 𝅗𝅥, ♩, ♪, 𝅘𝅥𝅯, 𝅘𝅥𝅰, ●}	ε, ♩𝅝,𝅝𝅝𝅘𝅥𝅰

Functions on Strings

***Length*:**

- $|s|$ is the length of string s
- $|s|$ is the number of characters in string s.

$|\varepsilon| = 0$

$|\texttt{1001101}| = 7$

$\#_c(s)$ is defined as the number of times that c occurs in s.

$\#_{\texttt{a}}(\texttt{abbaaa}) = 4$.

More Functions on Strings

Concatenation: the ***concatenation*** of 2 strings *s* and *t* is the string formed by appending t to s; written as s||t or more commonly, *st*

Example:

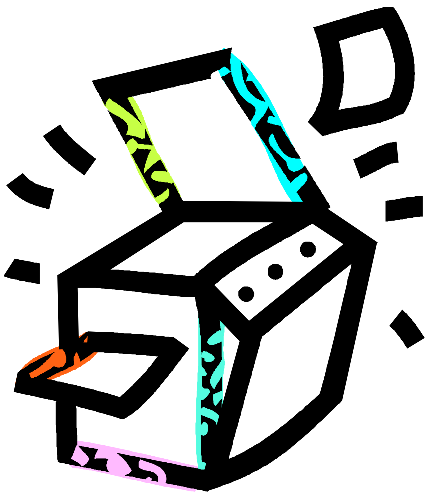
If x = `good` and y = `bye`, then xy = `goodbye` and yx = `byegood`

- Note that $|xy| = |x| + |y|$ -- Is it always??
- ε is the identity for concatenation of strings. So,
  $\forall x\ (x\,\varepsilon = \varepsilon\, x = x)$
- Concatenation is associative. So,
  $\forall s, t, w\ ((st)w = s(tw))$

More Functions on Strings

Replication: For each string *w* and each natural number *k*, the string w^k is:

$$w^0 = \varepsilon$$
$$w^{k+1} = w^k w$$



Examples:

$\mathtt{a}^3 = \mathtt{aaa}$

$(\mathtt{bye})^2 = \mathtt{byebye}$

$\mathtt{a}^0\mathtt{b}^3 = \mathtt{bbb}$

$\mathtt{b}^2\mathtt{y}^2\mathtt{e}^2 = ??$

Natural Numbers {0,1,2,…}

More Functions on Strings

Reverse: For each string w, w^R is defined as:

if $|w| = 0$ then $w^R = w = \varepsilon$

if $|w| = 1$ then $w^R = w$

if $|w| > 1$ then:

$\exists a \in \Sigma\ (\exists u \in \Sigma^*\ (w = ua))$

So define $w^R = a\ u^R$

OR

if $|w| > 1$ then:

$\exists a \in \Sigma\ \&\ \exists u \in \Sigma^*\ \ni w = ua$

So define $w^R = a\ u^R$

Proof is by simple induction

Relations on Strings - Substrings

- Substring: string s is a *substring* of string t if s occurs contiguously in t
  - Every string is a substring of itself
  - ε is a substring of every string
- Proper Substring: s is a proper substring of t iff $s \neq t$
- Suppose t = aabbcc.
  - Substrings: ε, a, aa, ab, bbcc, b, c, aabbcc
  - Proper substrings?
  - Others?

The Prefix Relations

s is a ***prefix*** of t iff $\exists x \in \Sigma^* \, (t = sx)$.

s is a ***proper prefix*** of t iff s is a prefix of t and $s \neq t$.

Examples:

The ***prefixes*** of `abba` are: ε, `a`, `ab`, `abb`, `abba`.
The ***proper prefixes*** of `abba` are: ε, `a`, `ab`, `abb`.

- Every string is a prefix of itself.
- ε is a prefix of every string.

The Suffix Relations

s is a ***suffix*** of t iff $\exists x \in \Sigma^* \, (t = xs)$.

s is a ***proper suffix*** of t iff s is a suffix of t and $s \neq t$.

Examples:

The ***suffixes*** of `abba` are: ε, `a`, `ba`, `bba`, `abba`.
The ***proper suffixes*** of `abba` are: ε, `a`, `ba`, `bba`.

- Every string is a suffix of itself.
- ε is a suffix of every string.

Defining a Language

A ***language*** is a (finite or infinite) set of strings over a (finite) alphabet Σ.

Examples: Let $\Sigma = \{a, b\}$

Some languages over Σ:

$\varnothing = \{\ \}$ // the empty language, no strings

$\{\varepsilon\}$ // language contains only the empty string

$\{a, b\}$

$\{\varepsilon, a, aa, aaa, aaaa, aaaaa\}$

Defining a Language

Two ways to define a language via a Machine = Automaton

AKA – Computer Program

- Recognizer
- Generator

Which do we want? Why?

Σ^*

- Σ^* is defined as the set of all possible strings that can be formed from the alphabet Σ^*
  - Σ^* is a language

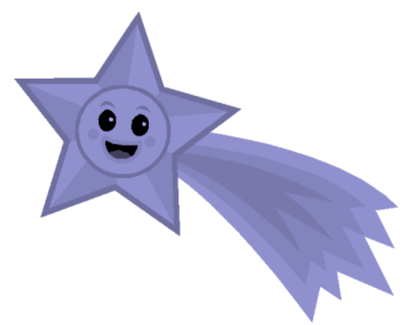
- Σ^* contains an *infinite* number of strings
  - Σ^* is *countably infinite*

Σ^* Example

Let Σ = {a, b}

Σ^* = {ε, a, b, aa, ab, ba, bb, aaa, aab, … }

Later, we will spend some more time studying Σ^*.



Defining Languages

Remember we are defining a set

Set Notation:

$$L = \{ w \in \Sigma^* \mid \text{description of } w\}$$

$$L = \{ w \in \{a,b,c\}^* \mid \text{description of } w\}$$

- "description of w" can take many forms but must be precise
- Notation can vary, but must precisely define

Example Language Definitions

$L = \{x \in \{\mathtt{a}, \mathtt{b}\}^* \mid$ all $\mathtt{a}$'s precede all $\mathtt{b}$'s$\}$

- `aab`, `aaabb`, and `aabbb` are in *L*.
- `aba`, `ba`, and `abc` are not in *L*.
- What about ε, `a`, `aa`, and `bb`?

$L = \{x : \exists y \in \{\mathtt{a}, \mathtt{b}\}^* \mid x = y\mathtt{a}\}$

- Give an English description.

Example Language Definitions

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$

- $L = \{ w \in \Sigma^* : |w| < 5\}$
- $L = \{ w \in \Sigma^* \mid$ w begins with b$\}$
- $L = \{ w \in \Sigma^* \mid \#_b(w) = 2\}$
- $L = \{ w \in \Sigma^* \mid$ each a is followed by exactly 2 b's$\}$
- $L = \{ w \in \Sigma^* \mid$ w does not begin with a$\}$

The Perils of Using English

$L = \{x\#y: x, y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*$ and, when x & y are viewed as decimal representations of natural numbers, $square(x) = y\}$.

Examples:

`3#9, 12#144`

`3#8, 12, 12#12#12`

`#`

A Halting Problem Language

L = {*w* | *w* is a C++ program that halts on all inputs}

- Well specified.
- Can we decide what strings it contains?
- Do we want a generator or recognizer?

More Examples

What strings are in the following languages?

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: no prefix of w contains $\mathtt{b}\}$

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: no prefix of w starts with $\mathtt{a}\}$

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: every prefix of w starts with $\mathtt{a}\}$

$L = \{\mathtt{a}^n : n \geq 0\}$

$L = \{\mathtt{ba}^{2n} : n \geq 0\}$

$L = \{\mathtt{b}^n\mathtt{a}^n : n \geq 0\}$

Enumeration

Enumeration: to list all strings in a language (set)

- Arbitrary order

- More useful: ***lexicographic order***
    - Shortest first
    - Within a length, dictionary order
    - Define linear order of arbitrary symbols

Lexicographic Enumeration

$\{w \in \{\mathtt{a}, \mathtt{b}\}^* : |w| \text{ is even}\}$

$\{\varepsilon$, aa, ab, bb, aaaa, aaab, …$\}$

What string is next?

How many strings of length 4?

How many strings of length 6?

Cardinality of a Language

- Cardinality of a Language: the number of strings in the language
- $| L |$
- Smallest language over any Σ is $\emptyset$, with cardinality 0.
- The largest is Σ^*.
  - Is this true?
  - How big is it?
- Can a language be uncountable?

Functions on Languages

Set (Language) functions
Have the traditional meaning

- Union
- Intersection
- Complement
- Difference

Language functions

- Concatenation
- Kleene star

Concatenation of Languages

If L_1 and L_2 are languages over Σ:

$$L_1L_2 = \{w : \exists s \in L_1 \ \& \ \exists t \in L_2 \ \ni w = st\}$$

Examples:

$L_1 = \{$`cat`, `dog`$\}$

$L_2 = \{$`apple`, `pear`$\}$

$L_1 L_2 = \{$`catapple`, `catpear`, `dogapple`, `dogpear`$\}$

$L_2 L_1 = \{$`applecat,appledog,pearcat,peardog`$\}$

Concatenation of Languages

$\{\varepsilon\}$ is the identity for concatenation:

$L\{\varepsilon\} = \{\varepsilon\}L = L$

$\varnothing$ is a zero for concatenation:

$L \varnothing = \varnothing L = \varnothing$

Concatenating Languages Defined Using Variables

The scope of any variable used in an expression that invokes replication will be taken to be the entire expression.

$L_1 = \{\mathtt{a}^n: \ n \geq 0\}$
$L_2 = \{\mathtt{b}^n : n \geq 0\}$

$L_1 \, L_2 = \{\mathtt{a}^n\mathtt{b}^m : n, m \geq 0\}$

$L_1L_2 \neq \{\mathtt{a}^n\mathtt{b}^n : n \geq 0\}$

Kleene Star *

L^* - *language consisting of 0 or more concatenations of strings from L*

$$L^* = \{\varepsilon\} \cup \{w \in \Sigma^* : w = w_1 w_2 \ldots w_{k,}\ k \geq 1 \ \&$$
$$w_1, w_2, \ \ldots w_k \in L\}$$

Examples:

$L = \{$`dog`, `cat`, `fish`$\}$

$L^* = \{\varepsilon,$ `dog, cat, fish, dogdog, dogcat, dogfish,fishcatfish,fishdogdogfishcat,` $\ldots\}$

$L_1 = \mathtt{a}^*$ $\qquad\qquad$ $L_2 = \mathtt{b}^*$

What is a? b*?*

$L_1 L_2 =$

$L_2 L_1 =$

$L_1 L_1 =$

The + Operator

L^+ = language consisting of 1 or more concatenations of strings from L

$L^+ = L\, L^*$

$L^+ = L^* - \{\varepsilon\}$ iff $\varepsilon \notin L$

Explain this definition!!

When is $\varepsilon \in L^+$?

Closure

- A set S is closed under the operation @ if for every element x & y in S, x@y is also an element of S
- A set S is closed under the operation @ if for every element $x \in S$ & $y \in S$, $x@y \in S$
- Examples

Semantics: Assigning Meaning to Strings

When is the meaning of a string important?

A semantic interpretation function assigns meanings to the strings of a language.

Can be very complex.

Example from English:

I brogelled the yourtish.
He's all thumbs.