A Language Hierarchy

Chapter 3

Generator vs. Recognizer

Reminder…

Given a problem, we can develop a machine (automaton) that

- Generates solutions, or
- Recognizes a solution

Generator vs. Recognizer

Example

Given 2 integers A & B, determine the sum.

- Generator: Write a program to accept A & B as input then compute the sum A+B
- Recognizer: Write a program to accept A & B & C as input then determine if A+B = C

We usually write Generators! But when would an Recognizer be an appropriate solution?

Decision Problems

A ***decision problem*** is simply a problem for which the answer is yes or no (True or False).
A ***decision procedure*** answers a decision problem.

Example

- Given an integer *n*, does *n* have a pair of consecutive integers as factors?

The language recognition problem: Given a language *L* and a string *w*, is *w* in *L*?

↑

Our focus

Encoding

Not the same as "coding", i.e. writing a computer program!!

"Everything is a string"

Do you believe that statement?

What about computer memory?

"Most problems in computing can be converted to a string"

How? By a correct encoding.

Thus, we can develop a decision solution.

Problems that don't look like decision problems can be recast into new problems that are decision.

E.G. A+B=C

Notation for Encoding into Strings

Almost anything can be encoded as a string.

Let X & Y be some type of “object”.

What is an “object”?

<*X*> is the string encoding of *X*.

<*X*, *Y*> is the string encoding of pair *X*, *Y*.

If we can define a problem as a language (of strings), we can develop a recognizer. It becomes a decision problem.

Example of Encoding

Pattern matching on the web

Problem: Given a search string *w* and a web document *d*, do they match? In other words, should a search engine, on input *w*, consider returning *d*?

The language to be decided:

{<*w*, *d*> : *d* is a candidate match for the query *w*}

Recognizer vs. Generator?

Example of Encoding

Does a program always halt?

Problem: Given a program *p*, written in some programming language, is *p* guaranteed to halt on all inputs?

- The language to be decided:

  $HP_{ALL} = \{p : p \text{ halts on all inputs}\}$

Classic problem: The Halting Problem. More later!

Example of Encoding

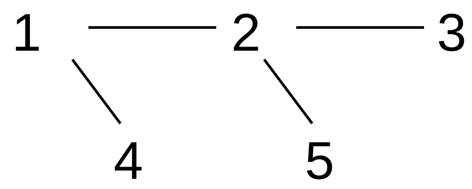
Testing for prime numbers

Problem: Given a nonnegative integer n, is it prime?

- The language
  PRIMES = $\{w : w$ is the binary encoding of a prime number$\}$.

Example of Encoding

- Problem: Given an undirected graph *G*, is it connected?
- Instance of the problem:



- Encoding of the problem: Let *V* be a set of binary numbers, one for each vertex in *G*. Then we construct ⟨*G*⟩ as follows:
  - Write |*V*| as a binary number,
  - Write a list of edges, each represented by pair of num. corresponding to vertices it connects (first # tells num. of vertices)
  - Separate all such binary numbers by "/".

101/1/10/10/11/1/100/10/101

- The language to be decided: CONNECTED = $\{w \in \{0, 1, /\}^* : w = n_1/n_2/\ldots n_i$, where each n_i is a binary string and *w* encodes a connected graph, as described above$\}$.

Turning Problems Into Decision Problems

Casting multiplication as decision:

- Problem: Given two nonnegative integers, compute the product.
- Encoding : Transform computing into verification.
- The language:

L = {w of the form:

$<integer_1>\mathtt{x}<integer_2>=<integer_3>$, where:

$<integer_n>$ is any well formed integer, and

$integer_3 = integer_1 * integer_2$}

```
12x9=108
12=12
12x8=108
```

Turning Problems Into Decision Problems

Casting sorting as decision:

- Problem: Given a list of integers, sort it.
- Encoding of the problem: Transform the sorting problem into one of examining a pair of lists.
- The language to be decided:

$L = \{w_1 \# w_2 : \exists n \geq 1$
$(w_1$ is of the form $<int_1, int_2, \ldots int_n>$,
w_2 is of the form $<int_1, int_2, \ldots int_n>$, and
w_2 contains the same objects as w_1 and
w_2 is sorted$)\}$

Could we define sorting as a different recognition problem??

Examples:

`1,5,3,9,6#1,3,5,6,9` $\in L$
`1,5,3,9,6#1,2,3,4,5,6,7` $\notin L$

The Traditional Problems & Their Language Formulations are Equivalent

Equivalent means either problem can be ***reduced to*** (***converted to***) the other.

Given a machine to solve one, a machine to solve the other can be built using the first machine & other functions that can be built using a machine of equal or lesser power.

An Example

Consider the multiplication example:

$L = \{w : \; <integer_1> \times <integer_2> = <integer_3>$, where:

$<integer_n>$ is a well-formed integer &

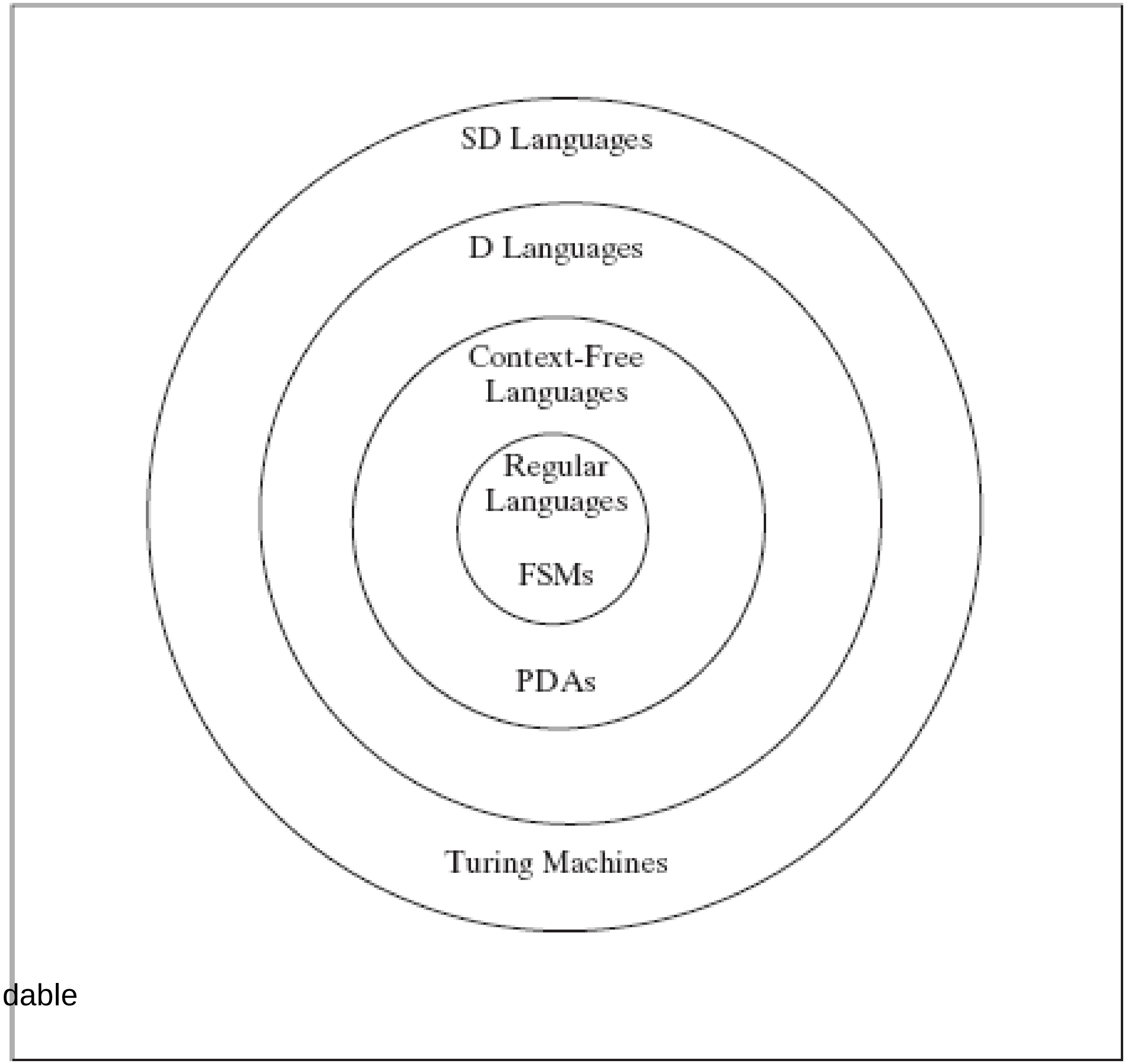
$$integer_3 = integer_1 * integer_2\}$$

Given a multiplication machine, we can build the language recognition machine.

Given the language recognition machine, we can build a multiplication machine.

This is not saying each machine is efficient!

One Hierarchy of Languages



D=decidable
SD = Semidecidable

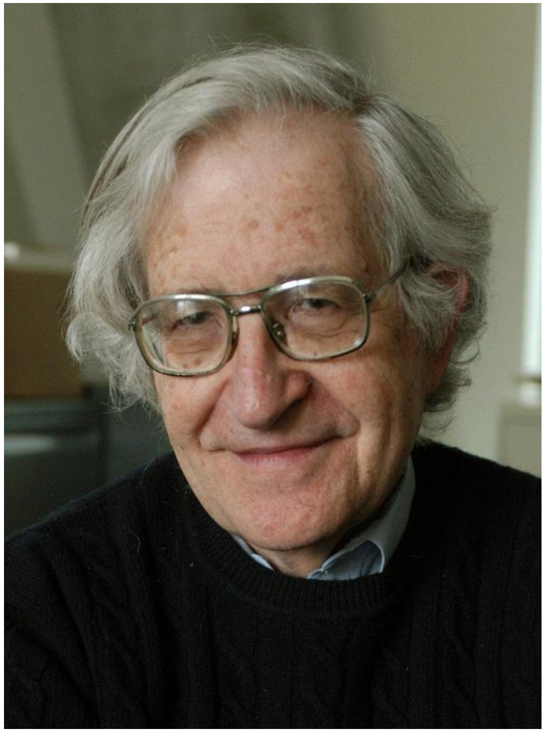
Chomsky Hierarchy of Languages

Languages from "simplest" to "complex"

Each is a subset of the ones below

- Regular
- Context Free
- Context Sensitive
- Recursively Enumerable

Can be defined by the type of Machine that will recognize it.

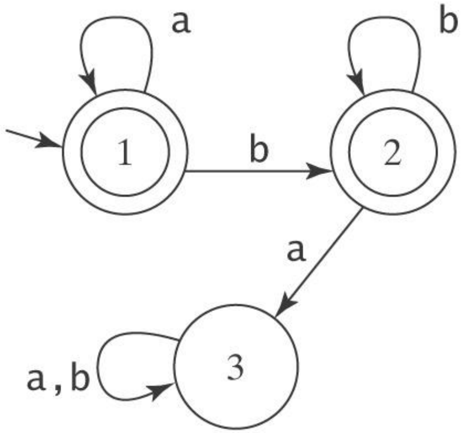


Noam Chomsky

Regular Languages

A Regular Language is one that can be recognized by a Finite State Machine.

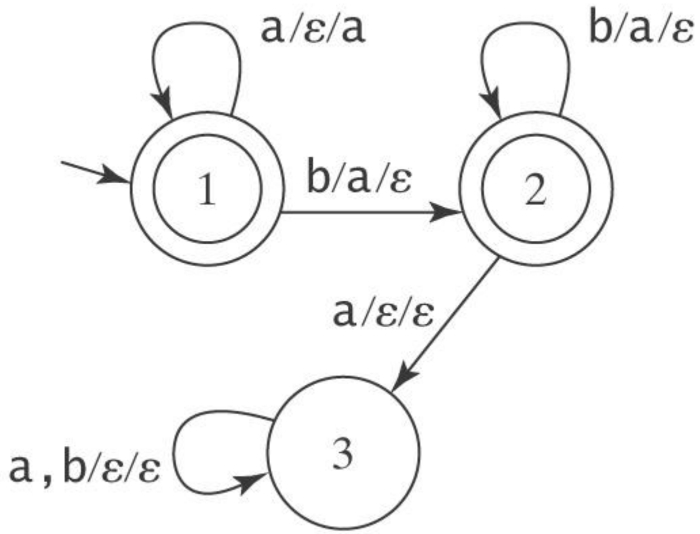
An FSM to accept a*b*:



Context Free Language

A Context Free Language is one that can be recognized by a Push Down Automata.

A PDA to accept $A^nB^n = \{a^nb^n : n \geq 0\}$

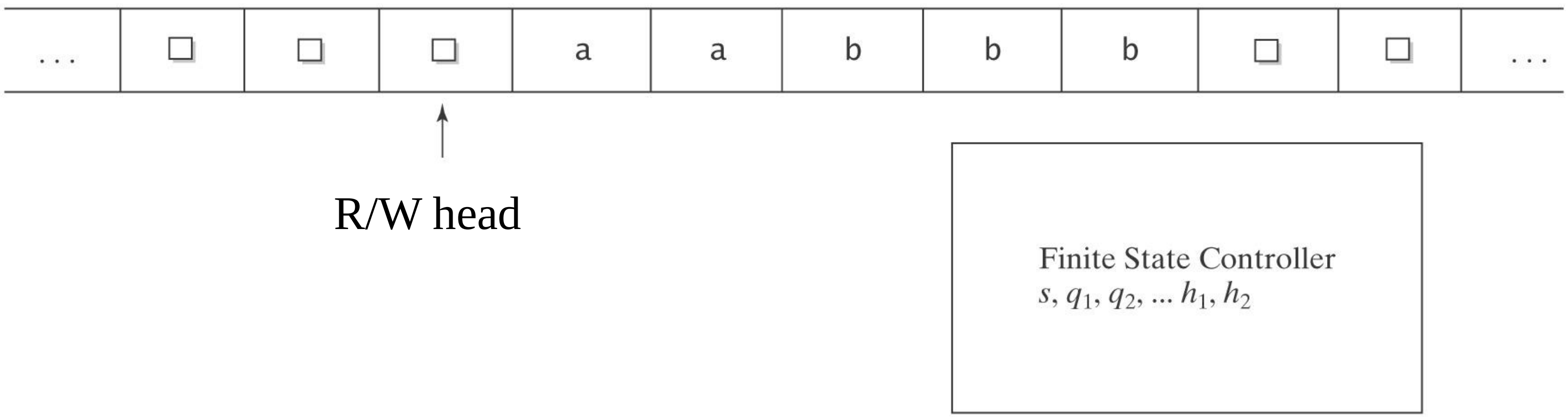


Decidable & Semidecidable Languages

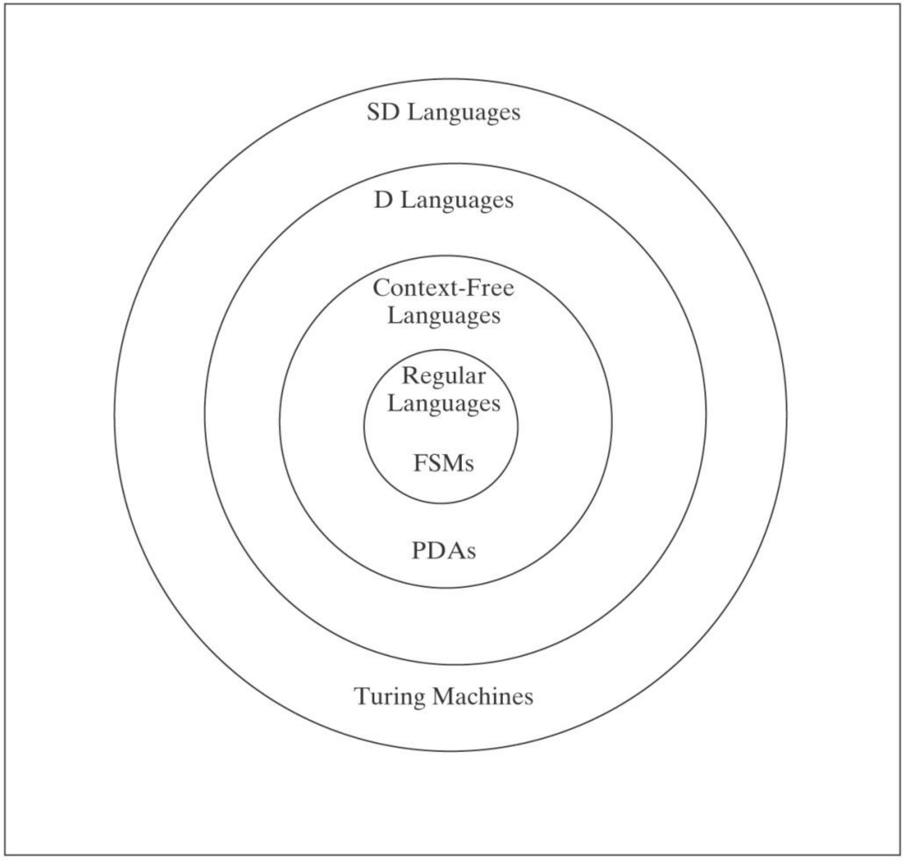
A Decidable Language is one that is recognized by a Turing Machine which halts on all input strings.

A Semidecidable Language is one that is recognized by a Turing Machine which halts on all input strings which are in the language, but may loop infinitely on some strings which are not in the language

Turing Machines



Languages and Machines



Rule of Least Power: “Use the least powerful language suitable for expressing information, constraints or programs on the World Wide Web.”