Why Study the Theory of Computation?

Implementations come and go.

Chapter 1

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
```

Programming in the 1970's (IBM 360)

```
//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

$$(\lceil/V) > (+/V) - \lceil/V$$

Applications of the Theory

- FSMs for parity checkers, vending machines, communication protocols, and building security devices.
- Interactive games as nondeterministic FSMs.
- Programming languages, compilers, and context-free grammars.
- Natural languages are mostly context-free. Speech understanding systems use probabilistic FSMs.
- Computational biology: DNA and proteins are strings.
- The undecidability of a simple security model.
- Artificial intelligence: the undecidability of first-order logic.

Limitations of Mathematics

This sentence is false.

Limitations of Computing

Is my program correct?

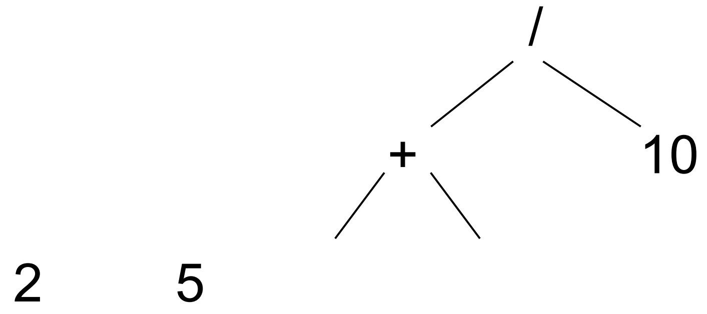
Languages and Strings

Chapter 2

Let's Look at Some Problems

```
int alpha, beta;
alpha = 3;
beta = (2 + 5) / 10;
```

(1) **Lexical analysis**: Scan the program and break it up into variable names, numbers, etc.
(2) **Parsing**: Create a tree that corresponds to the sequence of operations that should be executed, e.g.,



(3) **Optimization**: Realize that we can skip the first assignment since the value is never used and that we can precompute the arithmetic expression, since it contains only constants.
(4) **Termination**: Decide whether the program is guaranteed to halt.
(5) **Interpretation**: Figure out what (if anything) useful it does.

A Framework for Analyzing Problems

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

The framework we will use is

Language Recognition

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

Strings

A ***string*** is a finite sequence, possibly empty, of symbols drawn from some alphabet Σ.

- ε is the empty string.
- Σ* is the set of all possible strings over an alphabet Σ.

Alphabet name	***Alphabet symbols***	***Example strings***
The English alphabet	{a, b, c, …, z}	ε, aabbcg, aaaaa
The binary alphabet	{0, 1}	ε, 0, 001100
A star alphabet	{★ , ✪ , ✬ , ✯, ✡, ✰}	ε, ✪✪, ✪✯✯✰★✰
A music alphabet	{♩, ♪, ♫, ♬, ♭, ♮, ♯, |}	ε, ♪, ♪|♬♬♬

Functions on Strings

Length: $|s|$ is the number of symbols (characters, letters) in s.

$|\varepsilon| = 0$
$|\texttt{1001101}| = 7$

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

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

More Functions on Strings

Concatenation: *st* is the ***concatenation*** of *s* and *t*.

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

Note that $|xy| = |x| + |y|$.

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

Repetition (or power): For each string w and each natural number i, the string w^i is:

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

Examples:

$\mathtt{a}^3 = \mathtt{aaa}$
$(\mathtt{bye})^2 = \mathtt{byebye}$
$\mathtt{a}^0\mathtt{b}^3 = \mathtt{bbb}$

More Functions on Strings

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

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

if $|w| \geq 1$ then:

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

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

Concatenation and Reverse of Strings

Theorem: If w and x are strings, then $(w\,x)^R = x^R\,w^R$.

Example:

$(\texttt{nametag})^R = (\texttt{tag})^R\,(\texttt{name})^R = \texttt{gateman}$

Concatenation and Reverse of Strings

Proof: By induction on $|x|$:

$|x| = 0$: Then $x = \varepsilon$, and $(wx)^R = (w\,\varepsilon)^R = (w)^R = \varepsilon\, w^R = \varepsilon^R\, w^R = x^R\, w^R$.

$\forall n \geq 0\ (((|x| = n) \rightarrow ((w\,x)^R = x^R\,w^R)) \rightarrow$
$((|x| = n + 1) \rightarrow ((w\,x)^R = x^R\,w^R)))$:

Consider any string x, where $|x| = n + 1$. Then $x = u\,a$ for some character a and $|u| = n$. So:

$(w\,x)^R = (w\,(u\,a))^R$	rewrite x as ua
$= ((w\,u)\,a)^R$	associativity of concatenation
$= a\,(w\,u)^R$	definition of reversal
$= a\,(u^R\,w^R)$	induction hypothesis
$= (a\,u^R)\,w^R$	associativity of concatenation
$= (ua)^R\,w^R$	definition of reversal
$= x^R\,w^R$	rewrite ua as x

Relations on Strings

`aaa`	is a ***substring*** of	`aaabbbaaa`
`aaaaaa`	is not a substring of	`aaabbbaaa`
`aaa`	is a ***proper substring*** of	`aaabbbaaa`

Every string is a substring of itself.

ε is a substring of every string.

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 Σ = {a, b}

Some languages over Σ:

$\varnothing$,

{ε},

{a, b},

{ε, a, aa, aaa, aaaa, aaaaa}

The language Σ^* contains an infinite number of strings, including: ε, a, b, ab, ababaa.

Example Language Definitions

$L = \{x \in \{\texttt{a}, \texttt{b}\}^* :$ all `a`'s precede all `b`'s$\}$

`ab`, `aab`, and `aabbb` are in *L*.

`aba`, `ba`, and `abc` are not in *L*.

What about: ε, `a`, `aa`, and `bb`?

Example Language Definitions

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

Simple English description:

The Perils of Using English

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

Examples:

```
3#9, 12#144

3#8, 12, 12#12#12

#
```

More Example Language Definitions

$L = \{\} = \emptyset$

$L = \{\varepsilon\}$

English

L = {*w*: *w* is a sentence in English}.

Examples:

Kerry hit the ball.

Colorless green ideas sleep furiously.

The window needs fixed.

Ball the Stacy hit blue.

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?

Prefixes

What are the following languages:

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

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

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

Using Repetition in a Language Definition

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

Languages Are Sets

Computational definition:

- Generator (enumerator)
- Recognizer

Enumeration

Enumeration:

- Arbitrary order

- More useful: ***lexicographic order***
    - Shortest first
    - Within a length, dictionary order

The lexicographic enumeration of:

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

How Large is a Language?

The smallest language over any Σ is $\emptyset$, with cardinality 0.

The largest is Σ^*. How big is it?

How Large is a Language?

Theorem: If $\Sigma \neq \emptyset$ then Σ^* is countably infinite.

Proof: The elements of Σ^* can be lexicographically enumerated by the following procedure:

- Enumerate all strings of length 0, then length 1, then length 2, and so forth.
- Within the strings of a given length, enumerate them in dictionary order.

This enumeration is infinite since there is no longest string in Σ^*. Since there exists an infinite enumeration of Σ^*, it is countably infinite.

How Large is a Language?

So the smallest language has cardinality 0.

The largest is countably infinite.

So every language is either finite or countably infinite.

How Many Languages Are There?

Theorem: If $\Sigma \neq \varnothing$ then the set of languages over Σ is uncountably infinite.

Proof: The set of languages defined on Σ is $P(\Sigma^*)$. Σ^* is countably infinite. If S is a countably infinite set, $P(S)$ is uncountably infinite. So $P(\Sigma^*)$ is uncountably infinite.

Diagonalization

- Integers – countable
- Rational numbers – countable
- Irrational numbers – uncountable
  - Proof idea:
    - Assume they are countable: n_1, n_2, n_3, …
    - Construct N as follows:
      - First decimal of N ≠ first decimal of n_1
      - Second decimal of N ≠ second decimal of n_2
      - and so on
    - $N \neq n_i$ for any $i \geq 1$

Functions on Languages

- Set operations
  - Union
  - Intersection
  - Complement

- Language operations
  - Concatenation
  - Kleene star

Concatenation of Languages

If L_1 and L_2 are languages over Σ:

$$L_1L_2 = \{w \in \Sigma^* : \exists s \in L_1 (\exists t \in L_2 (w = st))\}$$

Examples:

L_1 = {cat, dog}
L_2 = {apple, pear}
L_1 L_2 ={catapple, catpear, dogapple, dogpear}

L_1 = a* L_2 = b*
L_1 L_2 =

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

$L^* = \{\varepsilon\} \cup$
$\quad \{w \in \Sigma^* : \exists k \geq 1$
$\quad\quad (\exists w_1, w_2, \ldots w_k \in L \ (w = w_1 w_2 \ldots w_k))\}$

Example:

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

L^* = {ε, `dog`, `cat`, `fish`, `dogdog`, `dogcat`, `fishcatfish`, `fishdogdogfishcat`, …}

The $^+$ Operator

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

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

L^+ is the closure of L under concatenation.

Concatenation and Reverse of Languages

Theorem: $(L_1 L_2)^R = L_2^R L_1^R$.

Proof:

$\forall x\ (\forall y\ ((xy)^R = y^R x^R))$

$(L_1 L_2)^R = \{(xy)^R : x \in L_1, y \in L_2\}$ (Def. of concat. and reverse)

$= \{y^R x^R : x \in L_1, y \in L_2\}$ (Theorem 2.1)

$= L_2^R L_1^R$ (Def. of concat. and reverse)

What About Meaning?

$A^nB^n = \{a^n b^n : n \geq 0\}$.

Do these strings mean anything?

Syntax = form

Semantics = meaning

Semantic Interpretation Functions

For “natural” languages:

- English
- DNA

For formal languages:

- Programming languages
- Network protocol languages
- Database query languages
- HTML
- BNF

The Big Picture

Chapter 3

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.

Examples:

- 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*?

$\uparrow$

Our focus

The Power of Encoding

Everything is a string.

Problems that don’t look like decision problems can be recast into new problems that do look like that.

The Power of Encoding

Pattern matching:

- **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*}

The Power of Encoding

Does a program always halt?

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

- The language to be decided:

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

What If We're Not Working with Strings?

Anything can be encoded as a string.

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

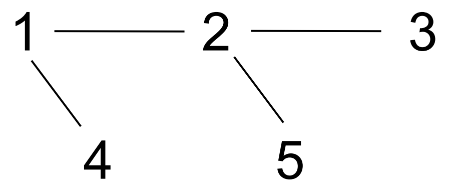
Primality Testing

- **Problem**: Given a nonnegative integer *n*, is it prime?
- An instance of the problem: Is 9 prime?
- To encode the problem we need a way to encode each instance: We encode each nonnegative integer as a binary string.
- The language to be decided:

PRIMES = {*w* : *w* is the binary encoding of a prime number}.

The Power 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,
  - 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}.

The Power of Encoding

- Protein sequence alignment:

- **Problem**: Given a protein fragment *f* and a complete protein molecule *p*, could *f* be a fragment from *p*?

- Encoding of the problem: Represent each protein molecule or fragment as a sequence of amino acid residues. Assign a letter to each of the 20 possible amino acids. So a protein fragment might be represented as `AGHTYWDNR`.

- The language to be decided: {<*f*, *p*> : *f* could be a fragment from *p*}.

Turning Problems Into Decision Problems

Casting multiplication as decision:

- **Problem**: Given two nonnegative integers, compute their product.
- Encoding of the problem:
  - ***Transform computing into verification.***
- The language to be decided:

L = {w of the form:

$<integer_1>$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$)\}$

`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 and their Language Formulations are Equivalent

By ***equivalent*** we mean that either problem can be ***reduced to*** the other.

If we have a machine to solve one, we can use it to build a machine to do the other using just the starting machine and other functions that can be built using a machine of equal or lesser power.

An Example

Consider the multiplication example:

$L = \{w$ of the form:

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

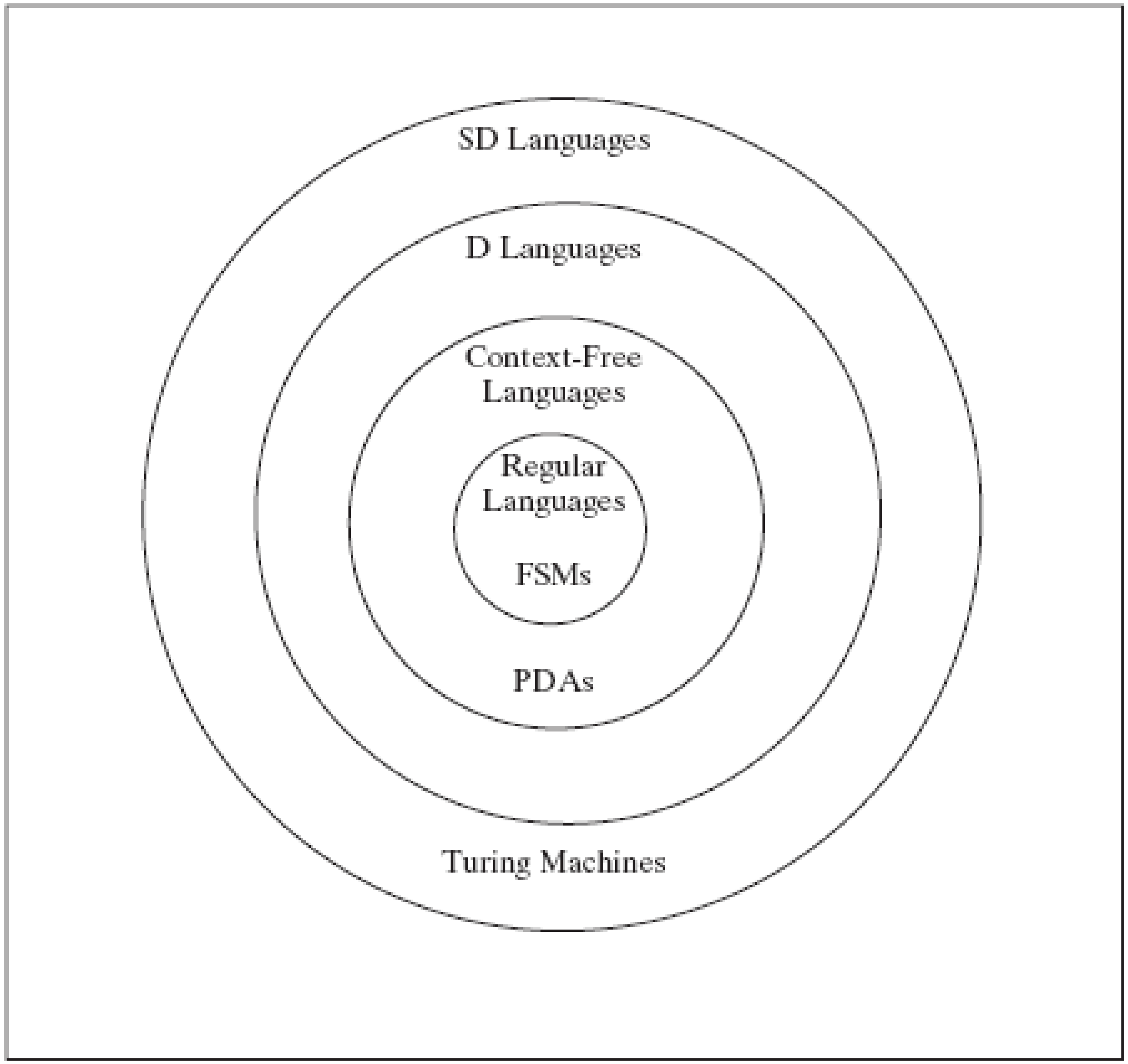
$<integer_n>$ is any well formed integer, and

$integer_3 = integer_1 * integer_2\}$

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

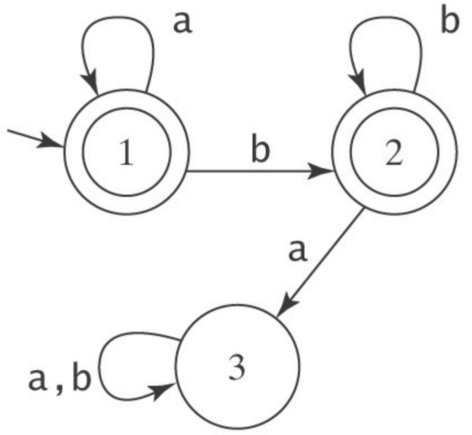
Given the language recognition machine, we can build a multiplication machine:

Languages and Machines



Finite State Machines

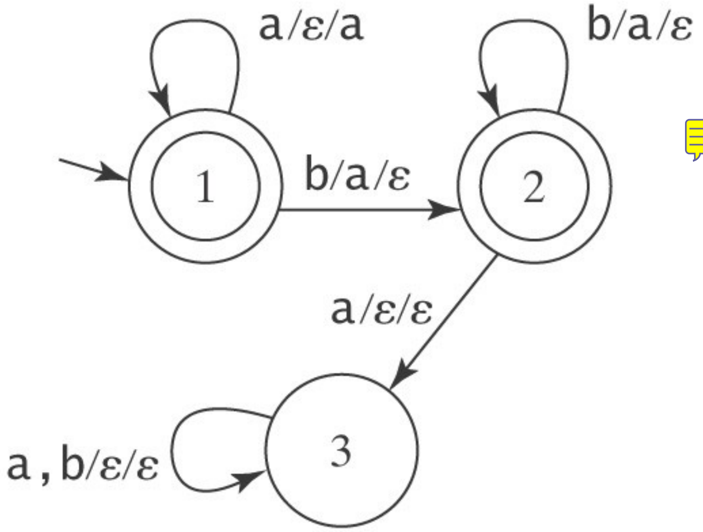
An FSM to accept a*b*:



An FSM to accept $A^nB^n = \{a^n b^n : n \geq 0\}$

Pushdown Automata

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



Example: aaabb

Stack:

Another Example

Bal, the language of balanced parentheses

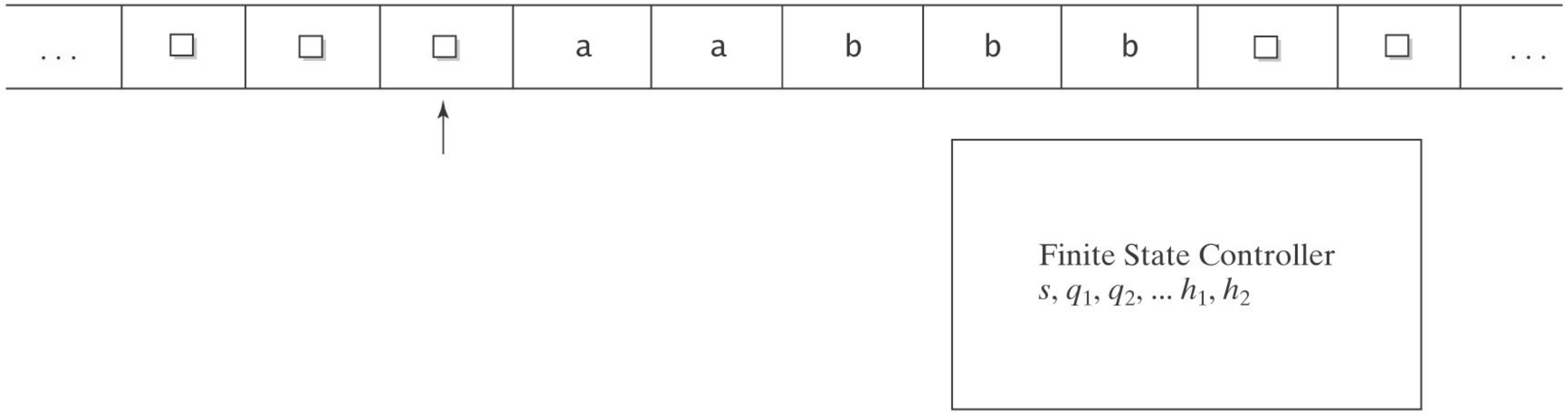
Trying Another PDA

A PDA to accept strings of the form:

$A^nB^nC^n = \{a^n b^n c^n : n \geq 0\}$

Turing Machines

A Turing Machine to accept $A^nB^nC^n$:

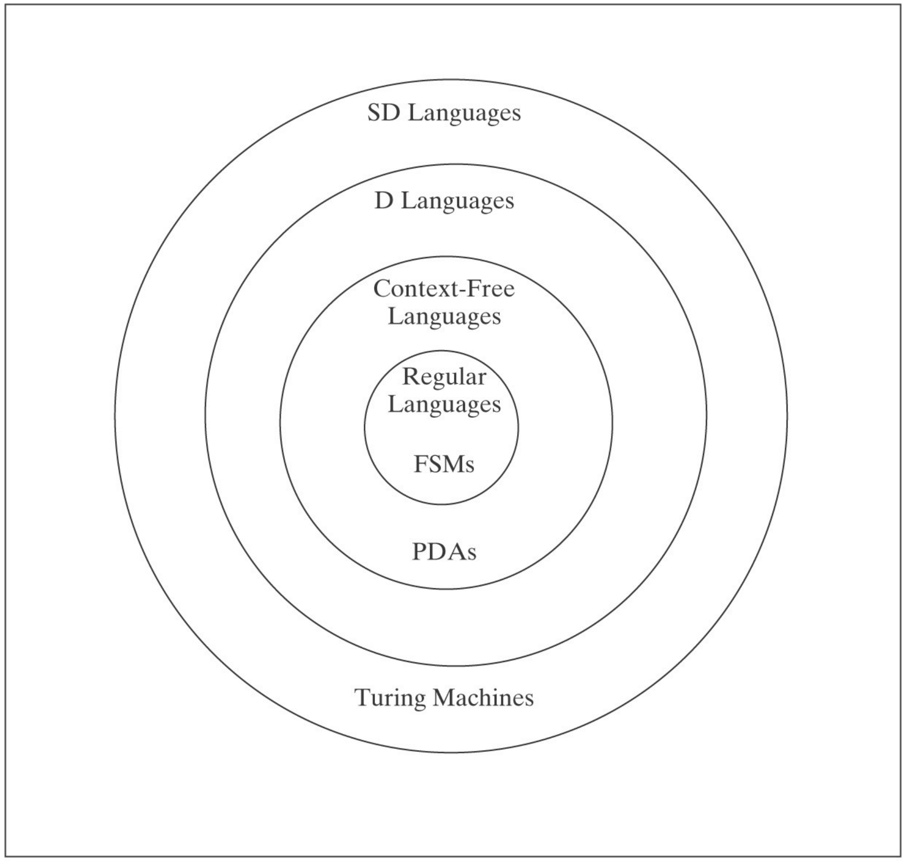


Turing Machines

A Turing machine to accept the language:

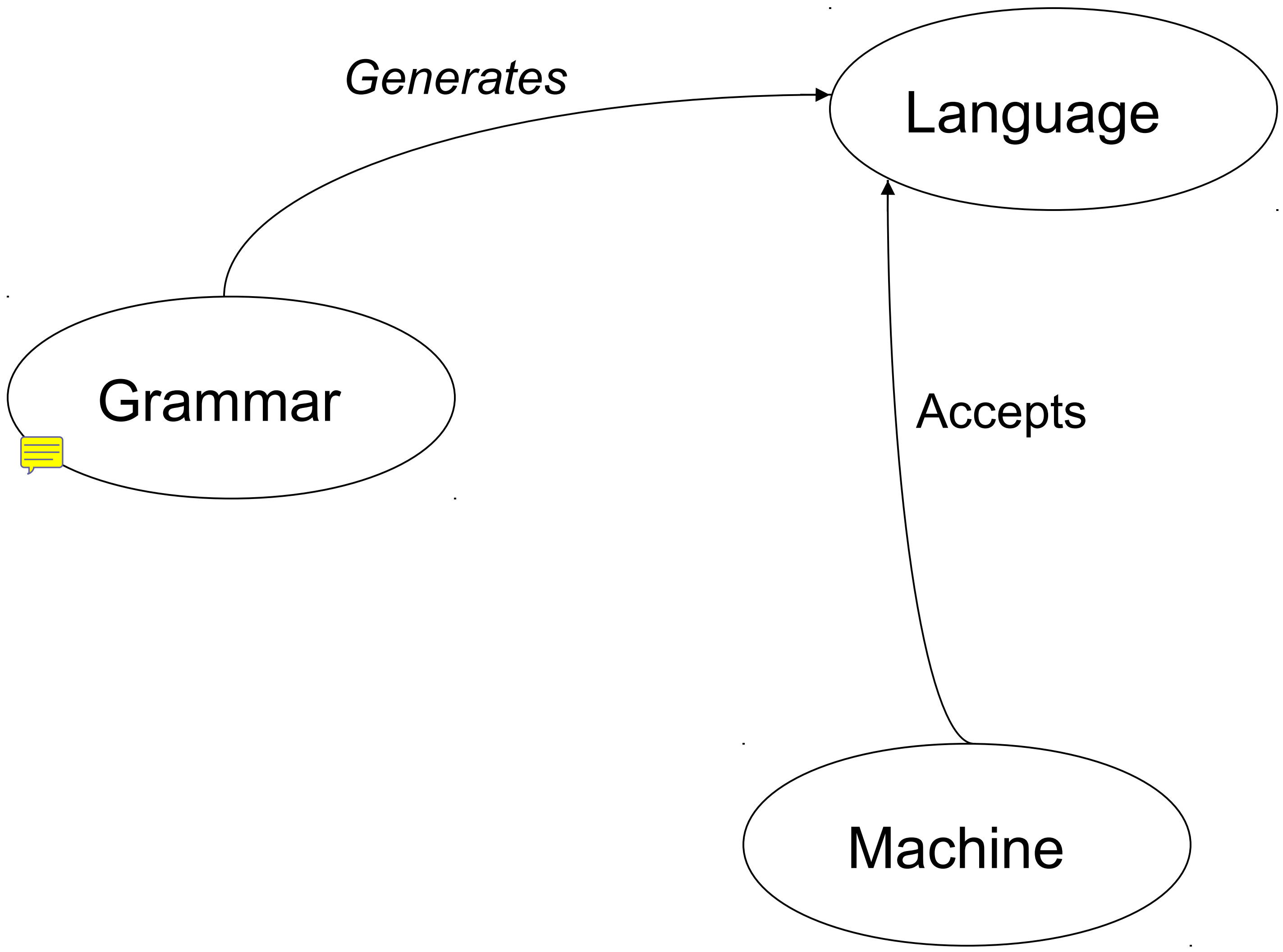
{*p*: *p* is a Java program that halts on input 0}

Languages and Machines



Rule of Least Power: "***Use the least powerful language suitable for the given problem.***"

Grammars, Languages, and Machines



Three Computational Issues

Formal Modeling of Computation:

- Problems: languages to be decided
- Programs: state machines that accept languages

Central Concerns:

1. Decision procedures
2. Nondeterminism
3. Functions on languages