PART 0:

Theory of Computation
Alphabets, Strings & Formal Languages
Problems as Language Recognition
Language Hierarchy: Computability & Complexity

Theory of Computation

Theory of Computation

- Theory of what can be computed and what cannot by real-world computers!
- Develop formal mathematical models of computation that reflect real-world computers.

Theory of Computation

- Central areas:
  - Formal Language Theory
  - Automata Theory
  - Computability Theory
  - Complexity Theory

Formal Language Theory

- Theory about **formal languages**.
- Formal languages?
  - A set of strings over a given alphabet.

Formal Languages

- Types of Formal languages:
  - Regular languages
  - Context-free languages
  - Context-sensitive languages
  - Recursive languages (Turing-decidable)
  - Recursively enumerable languages (semi-decidable/ Turing-recognizable)

Grammars

- Formal languages are defined by **formal grammars** as language generators.
  - A set of formation rules that describe which strings formed from the alphabet of a formal language are syntactically valid.

Grammars

- Types of Grammars:
  - Regular Grammars
  - Context-Free Grammars
  - Context-Sensitive Grammars
  - Unrestricted Grammars

Automata

- Formal language theory uses separate formalisms, **automata**, to describe their recognizers as language recognizers.
  - A typical abstract machine consists of a definition in terms of input, output, and the set of allowable operations used to turn the former into the latter.

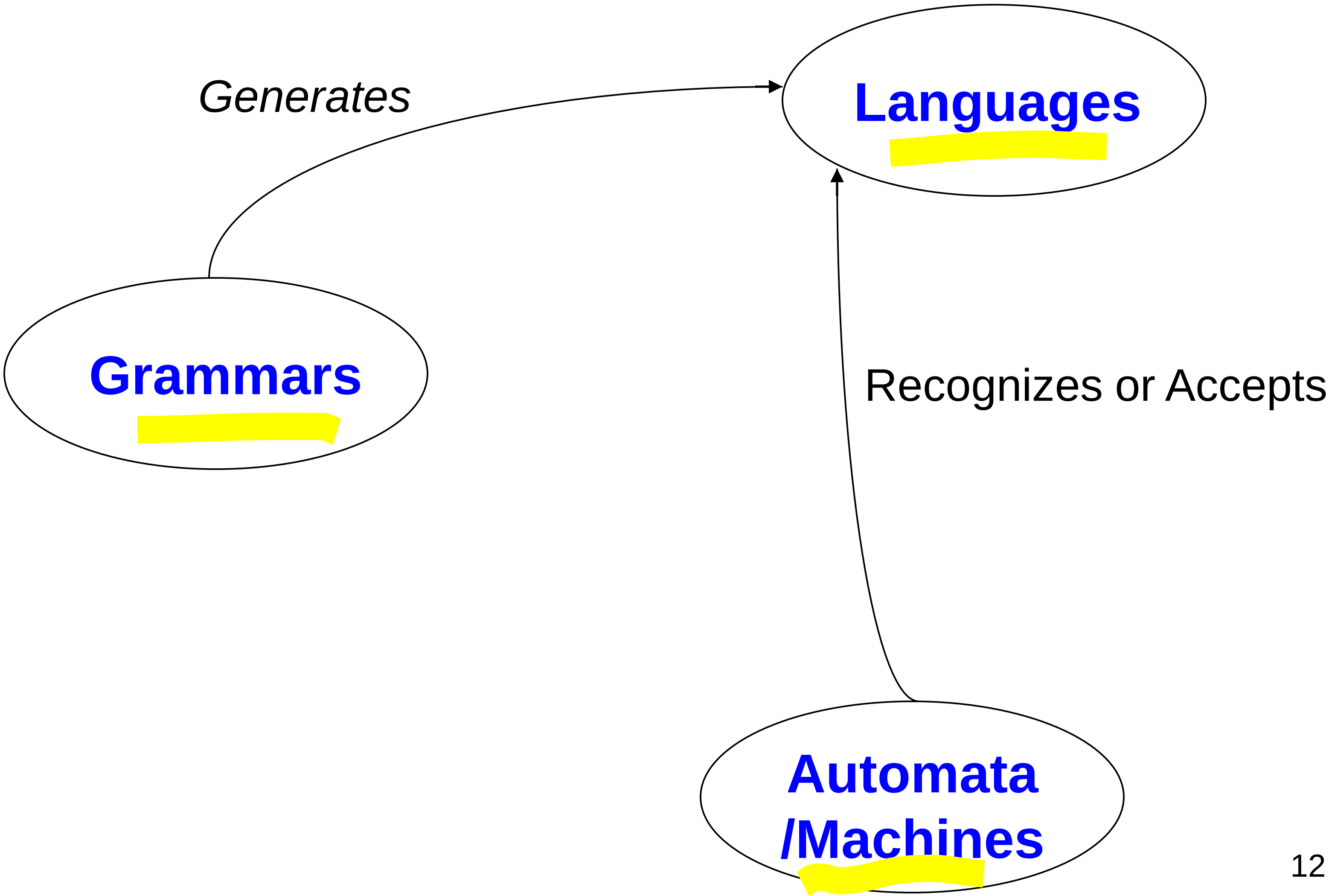
Automata

- Types of Automata:
  - FA (Finite Automata)
  - PDA (Pushdown Automata)
  - LBA (Linear Bounded Automata)
  - TM (Turing Machines)

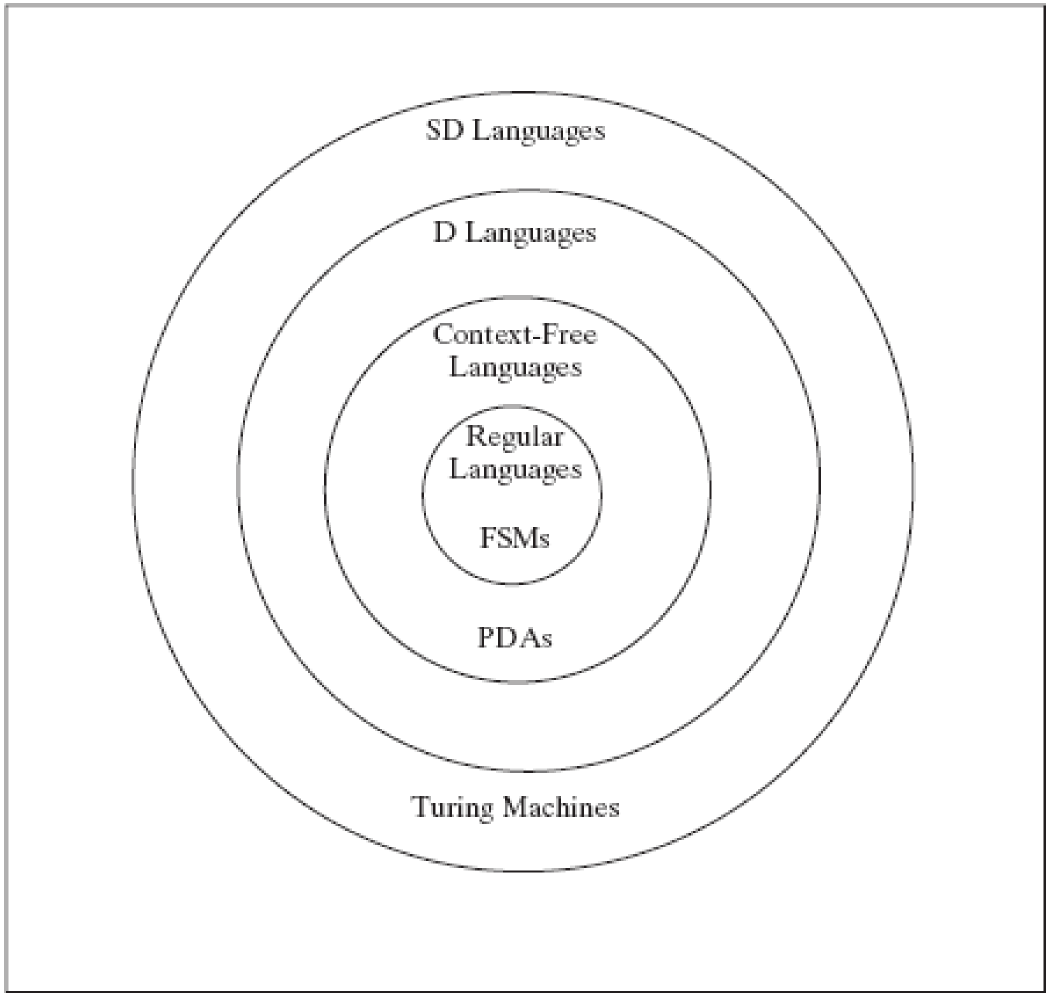
Automata Theory

- Study of abstract machines and problems they are able to solve.
  - An abstract machine, also called an abstract computer, is a theoretical model of a computer hardware or software system used in automata theory.
- Classify automata by the class of formal languages automata are able to recognize.

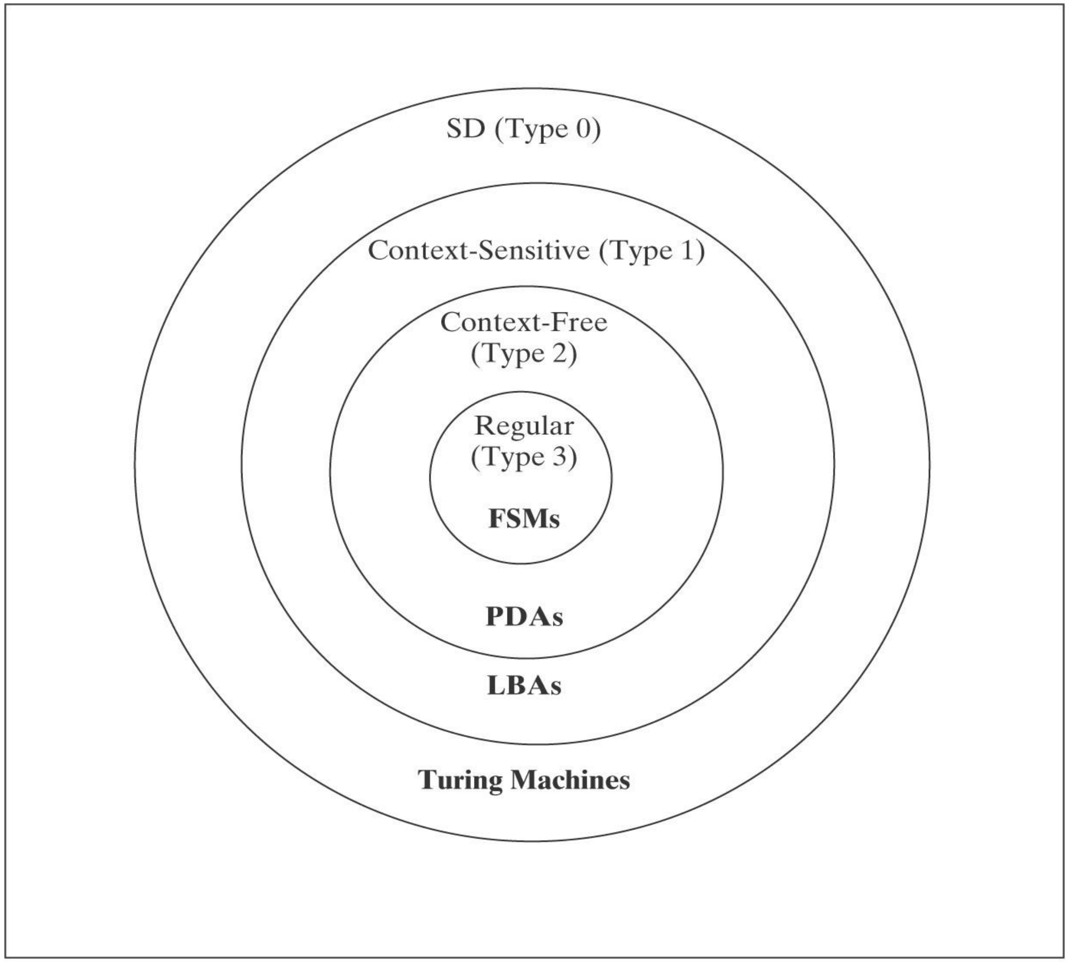
Languages, Grammars & Automata/Machines



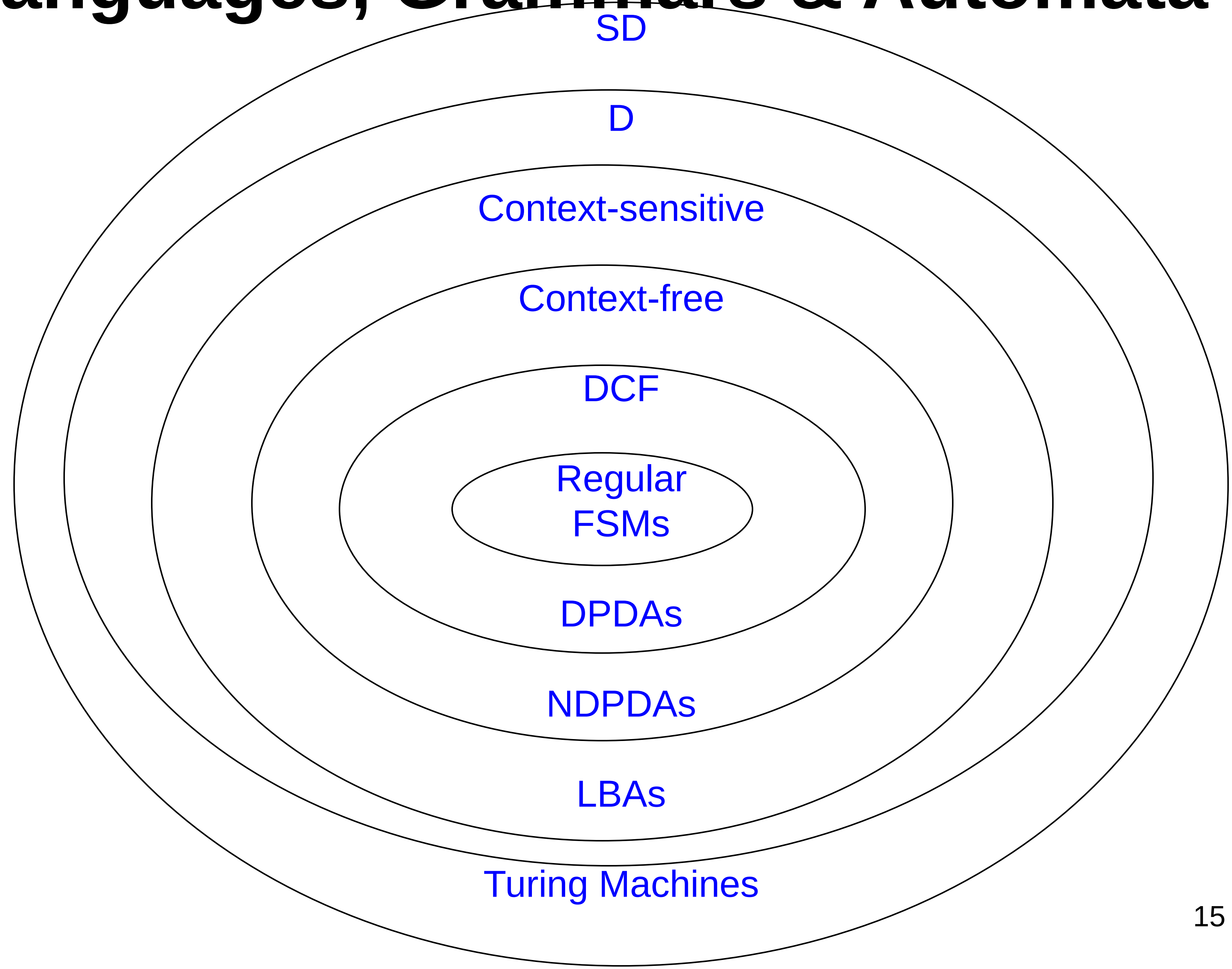
Languages, Grammars & Automata



Languages, Grammars & Automata



Languages, Grammars & Automata



Computability Theory

- **Computability**?
  - What are the fundamental capabilities and limitations of computers?
  - Classify problems as **solvable** and **unsolvable**.
  - **Unsolvability/Undecidability Theory**

Formal Models of Computation

- Both deal with formal models of computation:
  - **Turing machines**
  - Recursive functions
  - Lambda calculus
  - Production systems

Computability Hierarchy

- Decidable Languages D
  - Solvable Languages
  - Computable Languages
  - Recursive Languages
  - Turing Decidable Languages
- ¬ D Turing Undecidable Languages
- Semi-Decidable Languages SD
  - Recursively Enumerable (R.E.) Languages
  - Partially Decidable Languages
  - Turing Recognizable Languages
- ¬ SD Turing Unrecognizable Languages

Complexity Theory

- **Complexity**?
  - What makes some problems computationally hard and others easy?
    - **Time Complexity**
    - **Space Complexity**

Complexity Theory

- **Complexity**?
  - Classify solvable problems according to their degree of difficulty as **easy** ones and **hard** ones.
  - **Intractability Theory**

Complexity Hierarchy

- P
- NP
- PSPACE
- EXPTIME

Applications of Theory of Computation

- Appendices G, H, I, J, K, L, M, N O, P & Q

Reading Assignment

Chapter 1:

Sections
1.1
1.2

Alphabets, Strings & Formal Languages

Alphabets & Strings

An alphabet Σ is a finite set of symbols or characters.

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

ε is the empty string.

Example 2.1

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

Counting: $|s|$ is the number of symbols in *s*.

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

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

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

Functions on Strings

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

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

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

$\forall x \quad (x\,\varepsilon = \varepsilon\, x = x)$

Concatenation is associative:

$\forall s, t, w \quad ((st)w = s(tw))$

Functions on Strings

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

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

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

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

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

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

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 $s \neq t$.

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 $s \neq t$.

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.

Formal Languages

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

Example 2.2

Σ = {a, b}
Some languages over Σ?

- $\varnothing$ = The empty language
- {ε} = The language containing only ε
- {a, b},
- {ε, a, aa, aaa, aaaa, aaaaa}
- The language Σ^* = The set of all possible strings over an alphabet Σ.

Example 2.3

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

ε, a, aa, aabbb, and bb ?

aba, ba, and abc ?

Example 2.4

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

`a, aa, aaa, bbaa, ba` ?

ε, `bab`, `bca` ?

English description?

strings that end in a

Example 2.5

L = {*x*#*y*: *x*, *y* ∈ {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*}.

3#9, 12#144 ?

3#8, 12, 12#12#12 ?

?

Example 2.6 & 2.7

$L = \{\} = \emptyset$

$L = \{\varepsilon\}$

$L = \Sigma^*$

Example 2.9

L = {*w*: *w* is a C program that halts on all inputs}.

Example 2.10

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

$= \{ \varepsilon, \texttt{a}, \texttt{aa}, \texttt{aaa}, \texttt{aaaa}, \ldots \}$

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

$= \{w \in \{\texttt{a}, \texttt{b}\}^*$: the first char of w is $\texttt{a}\} \cup \{\varepsilon\}$

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

$= \varnothing$

Example 2.11

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

Languages Are Sets

Defining Languages?

- Language Generator (Enumerator)
- Language Recognizer

Enumeration

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

Example 2.12

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

The *lexicographic enumeration* of L?

ε, a, b, aa, ab, bb, aaa, aab, abb, bbb, aaaa, aaab, aabb, abbb, bbbb, aaaaa, …

Cardinality of Languages/Sets

- Finite
  - S has a natural number of elements.
- Infinite
  - Countably infinite
    - S has the same number of elements as there are integers.
  - Uncountably infinite
    - S has more elements than there are integers.

Finite Sets

A set A is finite and has cardinality $n \in \mathbb{N}$ iff either:

- $A = \varnothing$, or
- there is a bijection from $\{1, 2, \ldots n\}$ to A, for some n.

A set is infinite iff it is not finite.

The Cardinality of the Power Set

If S is a finite set, the cardinality of the power set of S $\mathcal{P}(S)$ is $2^{|S|}$.

The *power set* of *S* is the set of all subsets of *S*.

Example:

$S = \{1, 2, 3\}$

$\mathcal{P}(S) =$

$\{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.

Countably Infinite Sets

A is countably infinite and also has cardinality $\aleph_0$ iff there exists some bijection $f : \mathbb{N} \to A$.

A set is countable iff it is either finite or countably infinite.

To prove that a set A is countably infinite, it suffices to find a bijection from $\mathbb{N}$ to it.

Enumerations

An enumeration of a set *A* is simply a list of the elements of *A* in some order.

Each element of *A* must occur in the enumeration exactly once!

Enumerating Countably Infinite Sets

Theorem A.1 A set *A* is **countably infinite** iff there exists **an infinite enumeration** of it.

Not all infinite sets are countably infinite!

The Cardinality of the Power Set

Theorem A.4 If S is a countably infinite set, the power set of S $\mathcal{P}(S)$ is infinite, but not countably infinite. $\mathcal{P}(S)$ is called uncountably infinite!

Proof Idea:

Proof by Contradiction

The Diagonalization Method

The Diagonalization Method

	Elem 1 of S	Elem 2 of S	Elem 3 of S	Elem 4 of S	Elem 5 of S	…….
Elem 1 of $\mathcal{P}(S)$	1 (1)					…..
Elem 2 of $\mathcal{P}(S)$		1 (2)				…..
Elem 3 of $\mathcal{P}(S)$	1	1	(3)			…..
Elem 4 of $\mathcal{P}(S)$			1	(4)		…..
Elem 5 of $\mathcal{P}(S)$	1		1		(5)	…..
…						…..

A set that is not in the table:

$\neg$(1)	$\neg$(2)	$\neg$(3)	$\neg$(4)	$\neg$(5)	…..

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 2.2 If $\Sigma \neq \varnothing$ then Σ^* is countably infinite.

Proof Idea:

Proof by Construction

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 2.3 If $\Sigma \neq \emptyset$ then the set of languages over Σ is uncountably infinite.

Proof Idea:

The set of languages defined on Σ is $\mathcal{P}(\Sigma^*)$.
Σ^* is countably infinite.

If S is a countably infinite set, $\mathcal{P}(S)$ is uncountably infinite.

So $\mathcal{P}(\Sigma^*)$ is uncountably infinite.

Functions on Languages

- Set operations
  - Union
  - Intersection
  - Complement

- Language operations
  - Concatenation
  - Kleene star

Example 2.13

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

L_1 = {strings with an even number of a's}

L_2 = {strings with no b's}

- $L_1 \cup L_2 =$
- $L_1 \cap L_2 =$
- $L_2 - L_1 =$
- $\neg (L_2 - L_1) =$

Concatenation of Languages

If L_1 and L_2 are languages over Σ:

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

Example 2.14

$L_1 = \{\texttt{cat, dog, mouse, bird}\}$
$L_2 = \{\texttt{bone, food}\}$

$L_1 L_2 =$

Concatenation of Languages

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

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

$\emptyset$ is a zero for concatenation:

$$L \emptyset = \emptyset L = \emptyset$$

Concatenation of Languages

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

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

Kleene Star

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

Example 2.15

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.

Language Syntax & Semantics

Meaning = Semantics

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

Reading Assignment

Chapter 2:

Sections
2.1
2.2

Appendix A:
Sections
A.2
A.6

In-Class Exercises

Chapter 2:

1
2
3
4

Problems as Language Recognition

Language Hierarchy: Computability & Complexity

A Framework for Analyzing Problems

- A single framework in which we can analyze a very diverse set of problems.

- The framework we will use is

  Language Recognition

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.

- Must halt on all input.

Language Recognition Decision Problems

- The language recognition problem:

Given a language *L* and a string *w*, is *w* in *L*?

- The single framework into which any computational problem can be cast!

Two Ways to Describe a Problem

- As a problem
  - The problem view!
- As a language
  - The language view!

Casting Problems as Language Recognition Decision Problems

- Everything is a string.
- Problems that don't look like decision problems can be recast into new problems that do look like that.
- Define problems as languages to be decided!

Example 3.1

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:

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

Example 3.2

Problem: Given an English question q and a web document d , does d contain the answer to q?

The language to be decided:

L = {<*q*, *d*> : *d* contains the answer to q.}

Example 3.3

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:

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

Example 3.4

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

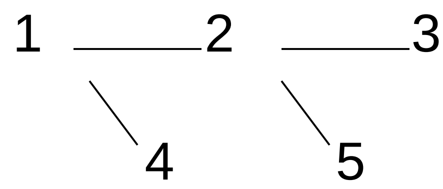
The language to be decided:

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

Example 3.6

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 $\langle G \rangle$ 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$\}$.

Example 3.8

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
```

Example 3.9

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, … int_n>,
w_2 is of the form <int_1, int_2, … 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` $\notin L$

Example 3.10

Problem: Given a database and a query, execute the query.

Encoding of the problem: Transform the query execution problem into evaluating a reply for correctness.

The language to be decided:

L = {*d* # *q* # *a*:
d is an encoding of a database,
q is a string representing a query, and
a is the correct result of applying *q* to *d*}

```
(name, age, phone), (John, 23, 567-1234)
(Mary, 24, 234-9876)#(select name age=23)#
(John)    ∈ 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.

The Reduction Method

The Reduction Method

A **reduction** is a way of converting one problem/language P to another problem/language P′ in such a way that a solution to the second problem S′ can be used to solve the first problem/language S.

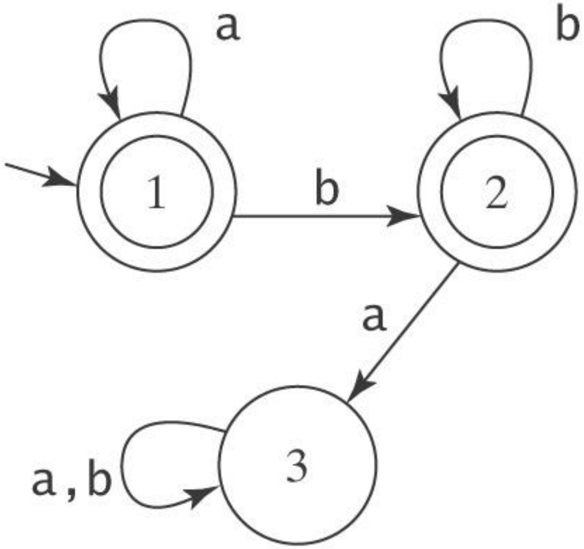
- $P \leq P'$ means that P is **reducible** to P'
- $L \leq L'$ means that L is **reducible** to L'

- Note that reduction says nothing about solving P or P′ alone, but only about the solvability of P in the presence of a solution to P′!

Computational Hierarchy of Languages

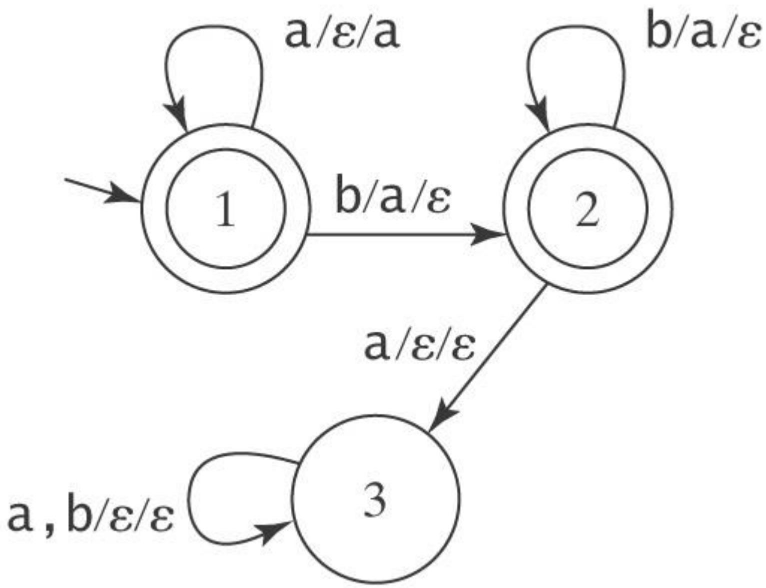
Regular Languages & Finite State Machines

An FSM to accept a*b*:



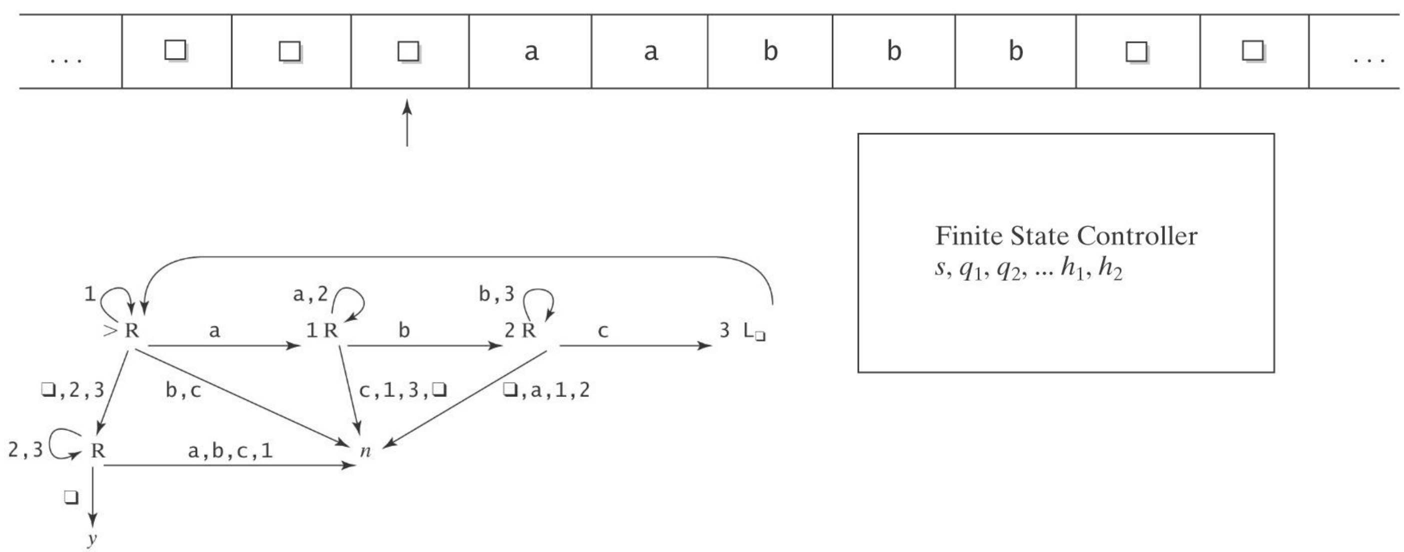
Context-Free Languages & Pushdown Automata

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



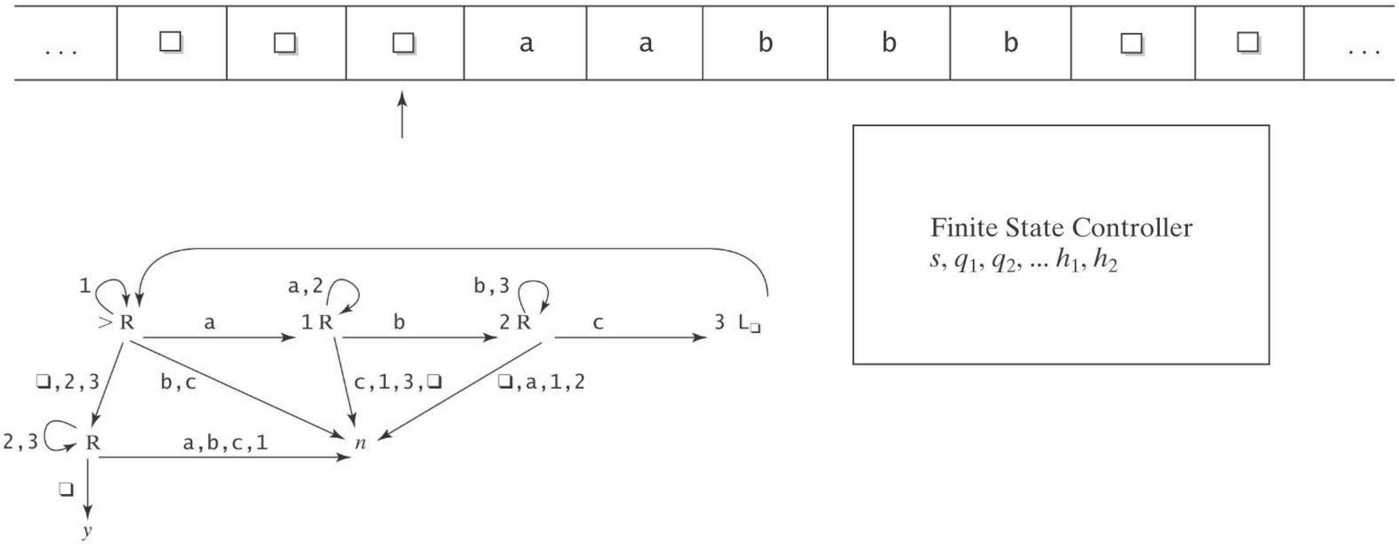
Context-Sensitive Languages & Linear Bounded Automata

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



Decidable & Semi-Decidable Languages & Turing Machines

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

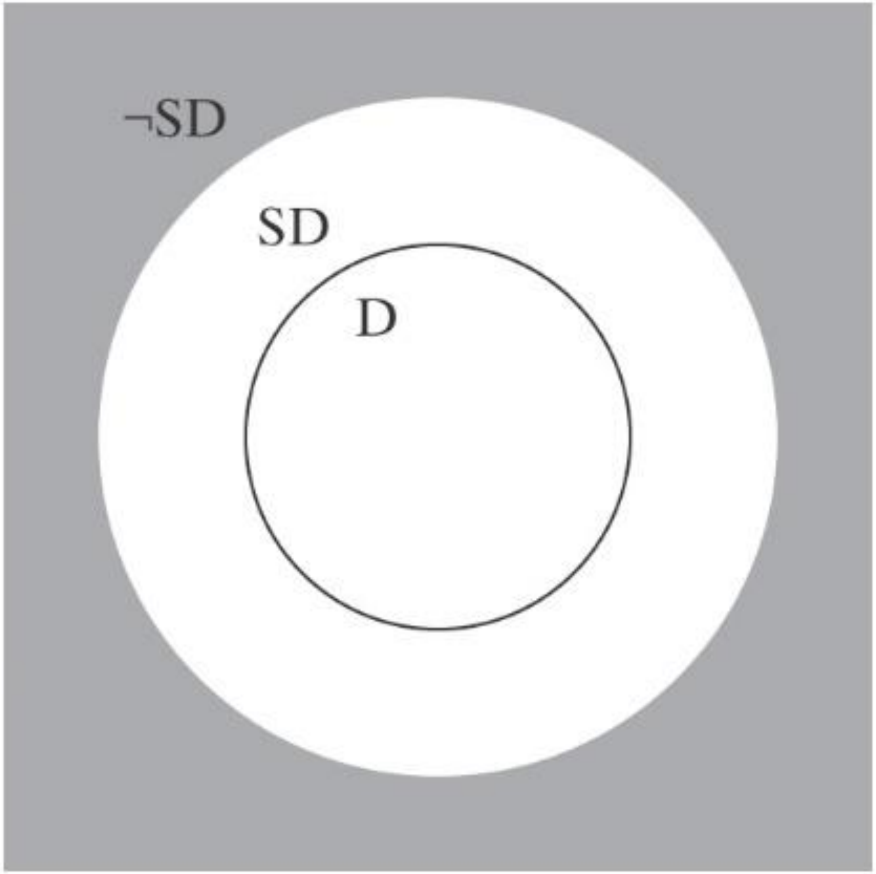


Computability Hierarchy

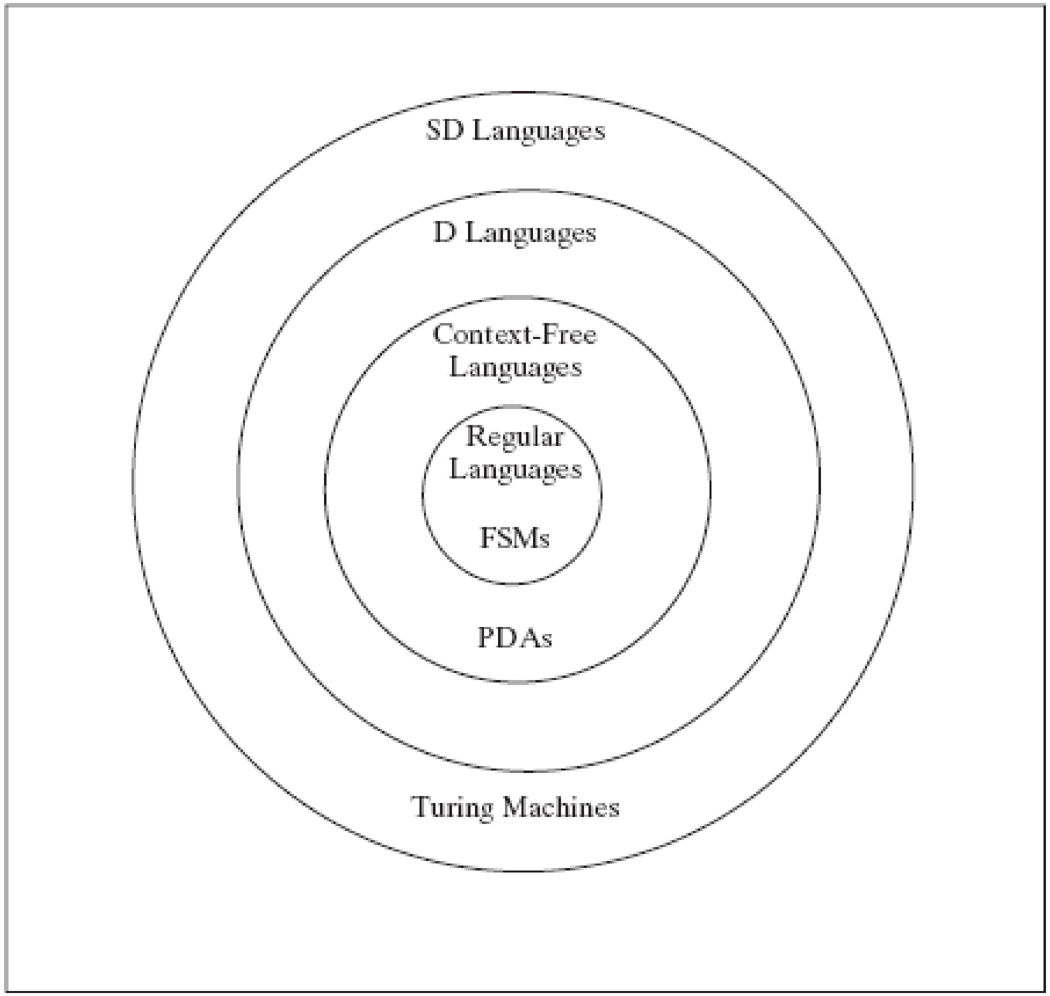
- Decidable Languages D
  - Solvable Languages
  - Computable Languages
  - Recursive Languages
  - Turing Decidable Languages
- ¬ D Turing Undecidable Languages

- Semi-Decidable Languages SD
  - Recursively Enumerable (R.E.) Languages
  - Partially Decidable Languages
  - Turing Recognizable Languages
- ¬ SD Turing Unrecognizable Languages

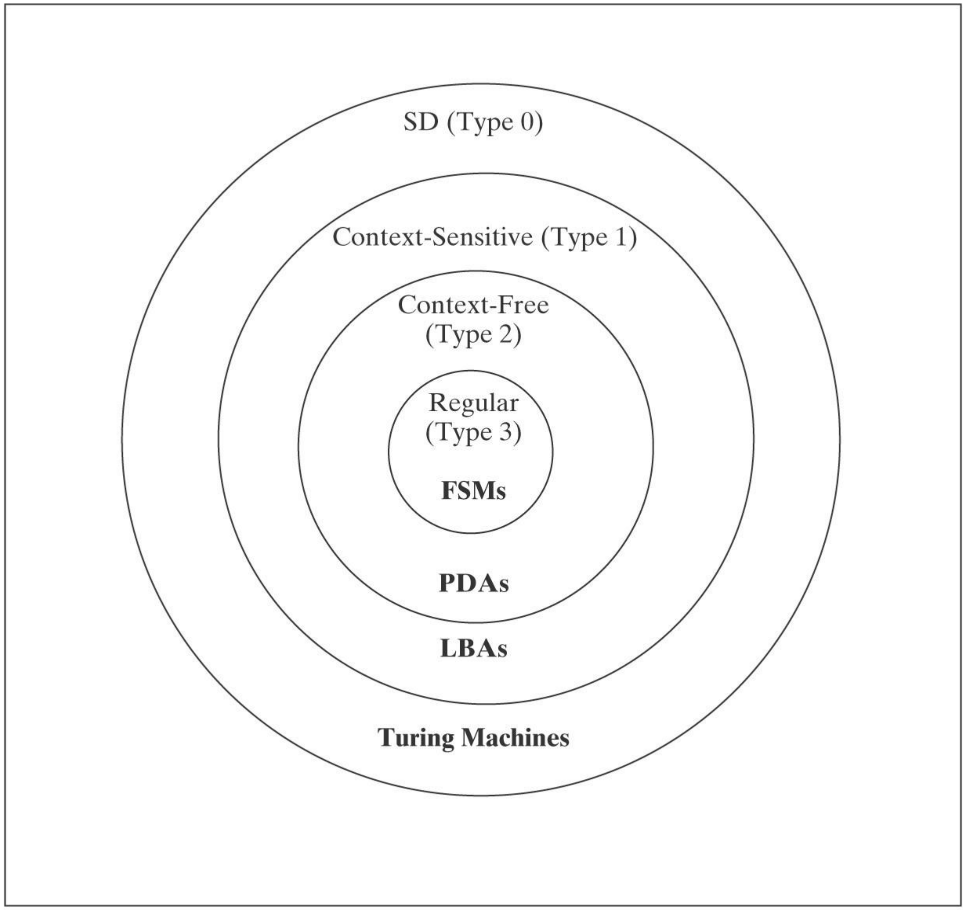
Computability Hierarchy



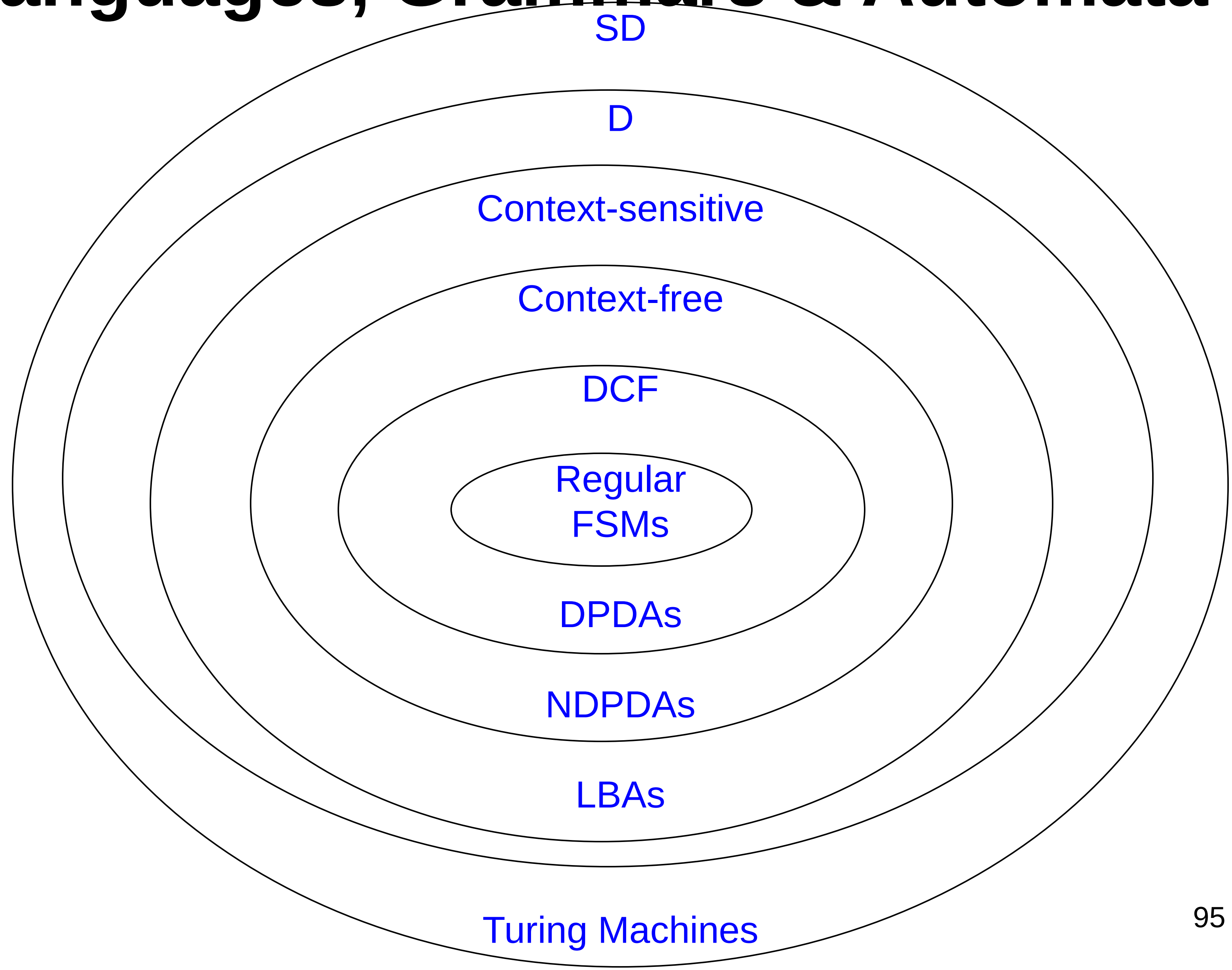
Languages, Grammars & Automata



Languages, Grammars & Automata



Languages, Grammars & Automata



Complexity Hierarchy of Decidable Languages

Complexity Hierarchy of Decidable Languages

- The class of decidable languages
- The resources (time & space) required by the best decision procedures?

Tractability Hierarchy of Decidable Languages

- P
- NP
- PSPACE
- EXPTIME

$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$

Reading Assignment

Chapter 3:

Sections
3.1
3.2
3.3
3.4

In-Class Exercises

Chapter 3:

1
2
3
4

In-Class Exercises

Chapter 4:

1
2