PART 3:

Automata:

Turing Machines

Formal Languages & Computability Theory:

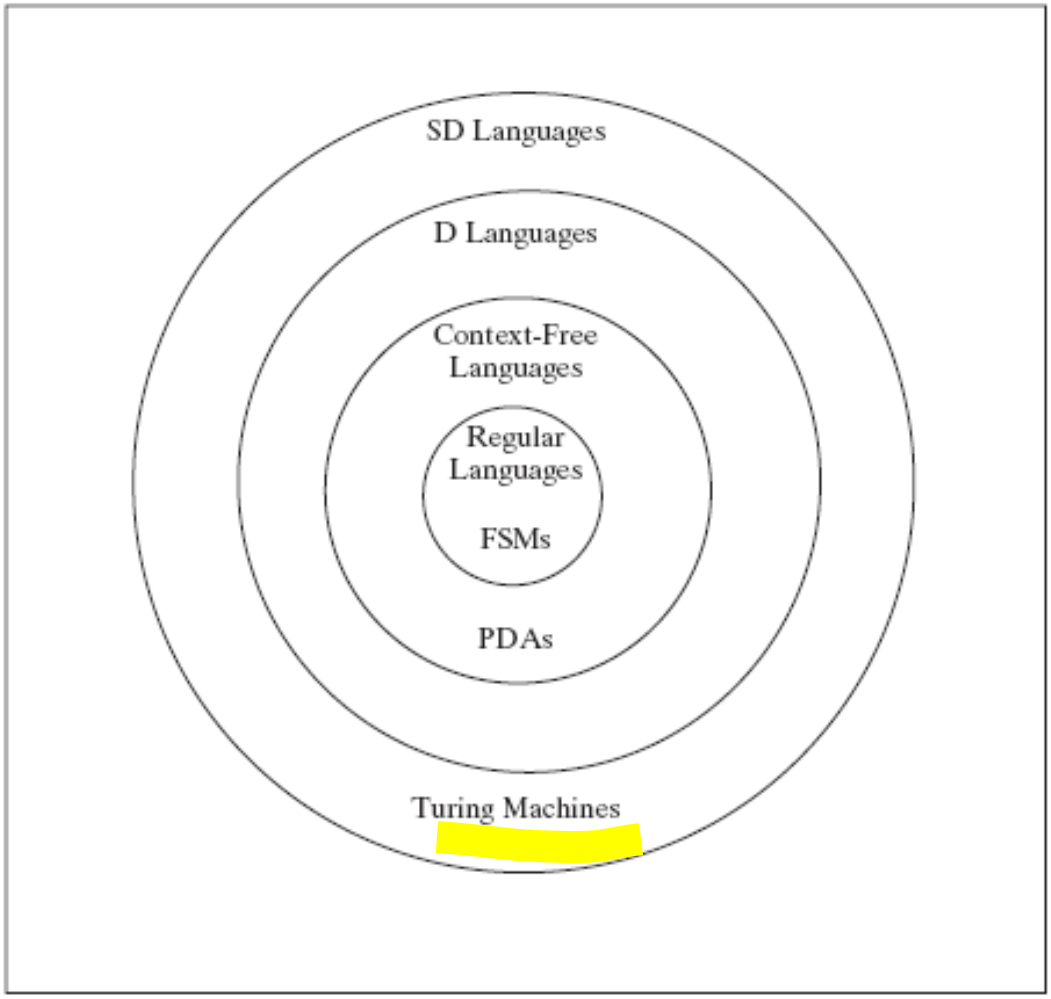
Church-Turing Thesis
Unsolvability/Undecidability of the Halting Problem
Decidable & Non-Decidable Languages
Semi-Decidable & Non-Semi-Decidable Languages

Grammar:

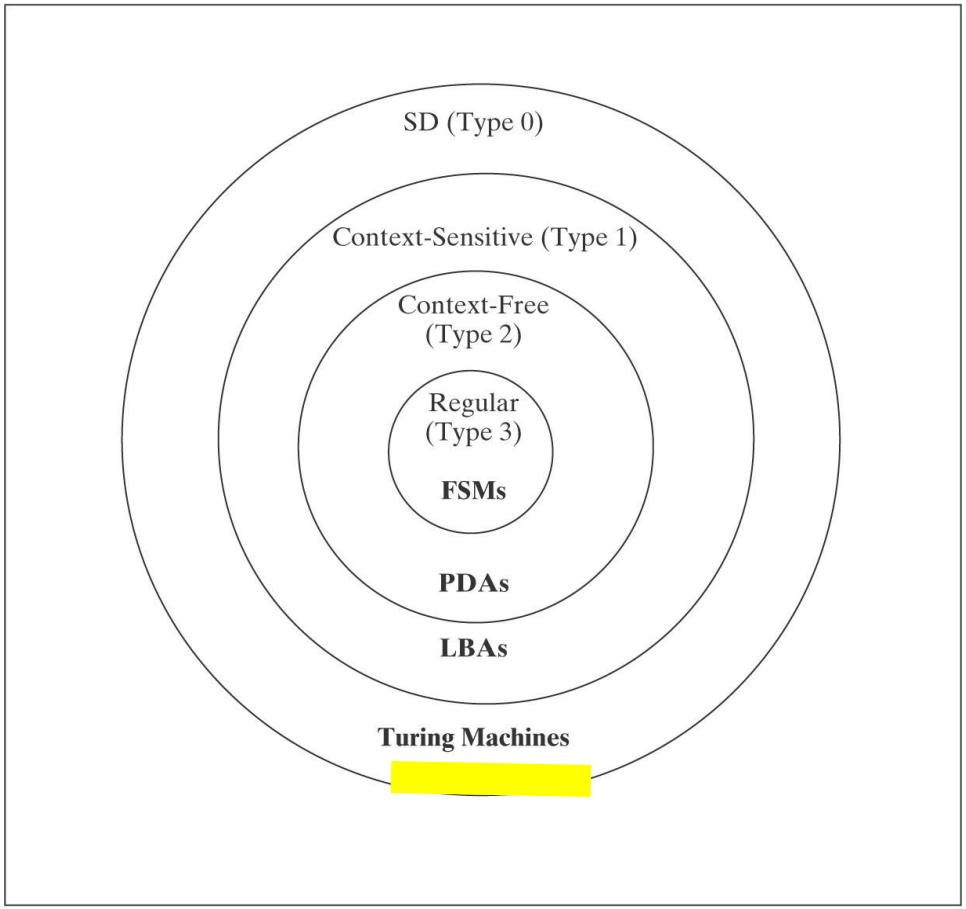
Unrestricted Grammars

Turing Machines

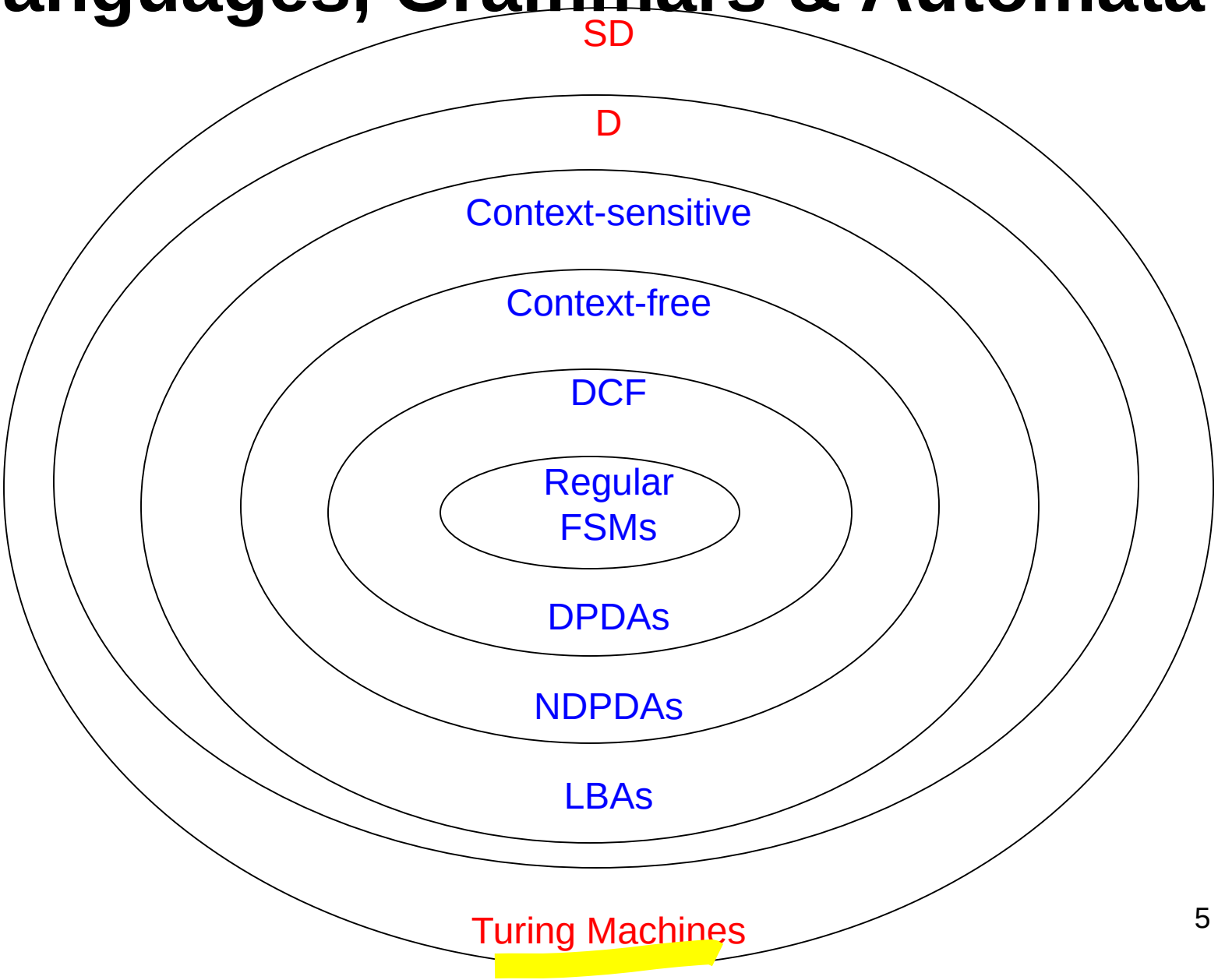
Languages, Grammars & Automata



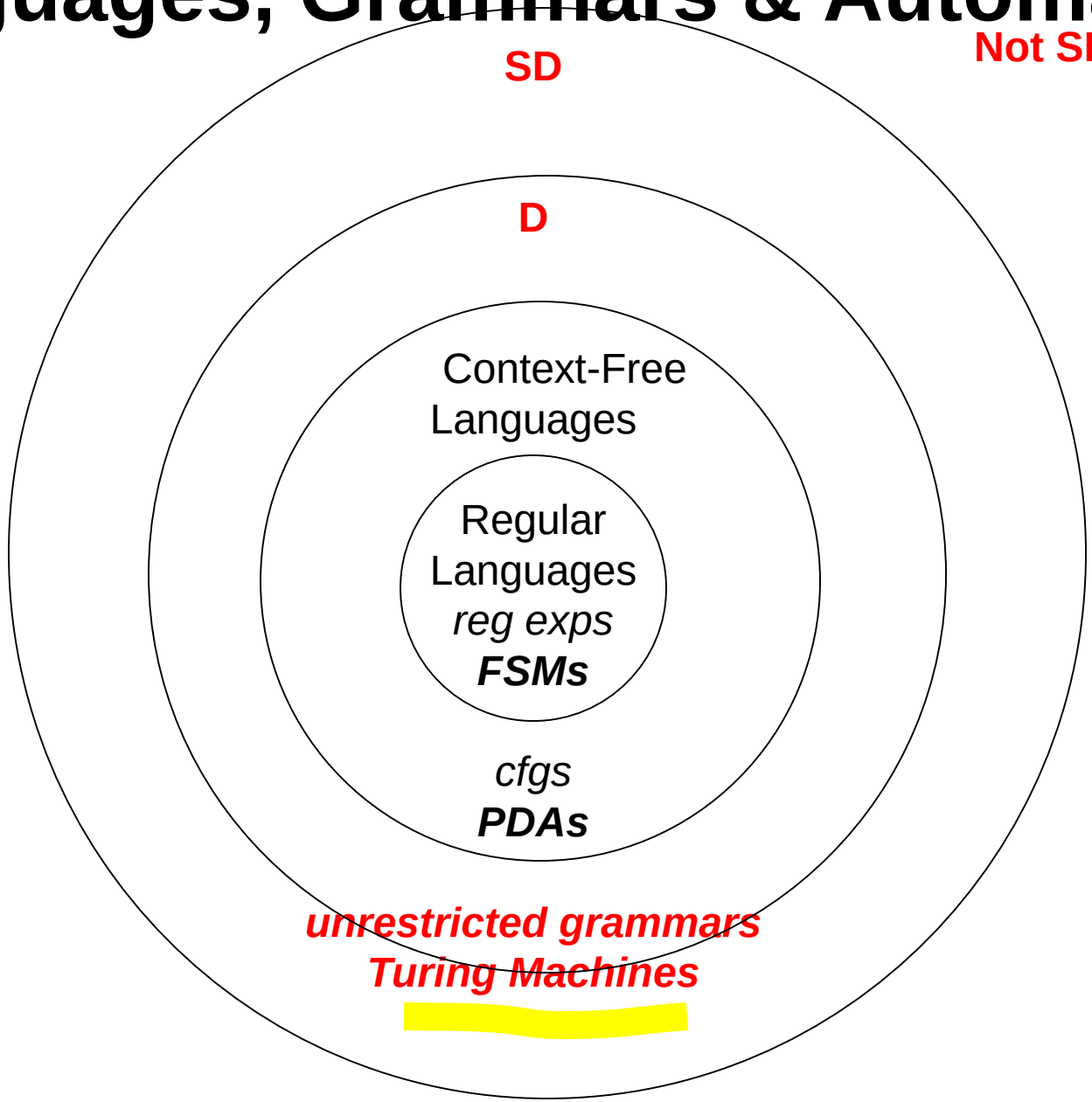
Languages, Grammars & Automata



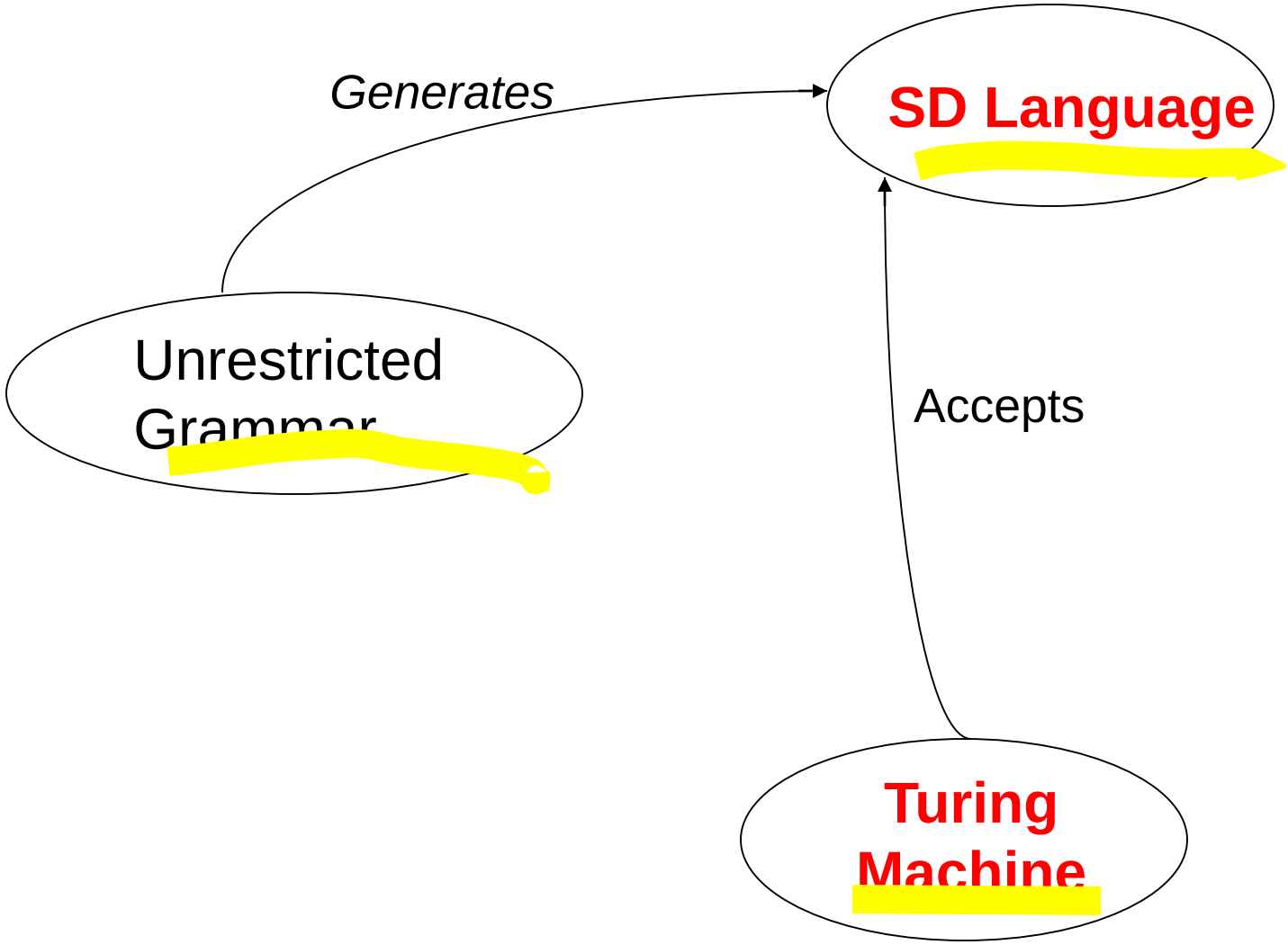
Languages, Grammars & Automata



Languages, Grammars & Automata



Grammars, SD Languages, and TMs



A Model of General Purpose Computers

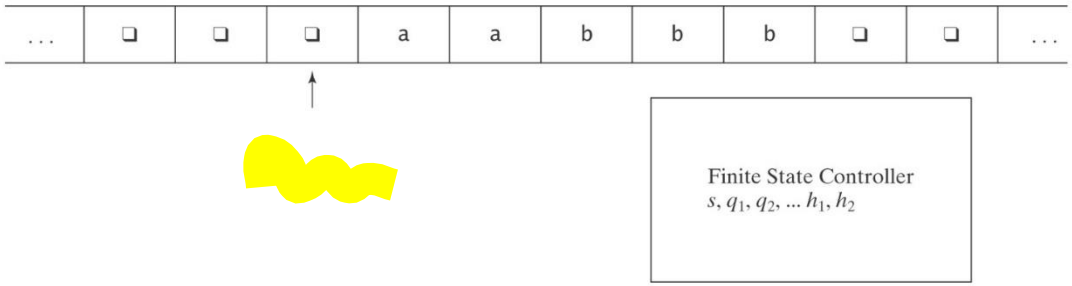
A new kind of automaton/ abstract computing machine that has two properties:

- *powerful* enough to describe all computable things (unlike FSMs and PDAs, like real computers).
- *simple* enough that we can reason formally about it (like FSMs and PDAs, unlike real computers).

Turing Machines

- FSM + an unlimited and unrestricted memory (writable tape)!
- A TM is an accurate model of a general purpose computer!
- A TM can do everything that a real computer can do!
- By Alan Turing in 1936!

Turing Machines



- **An infinite tape** (as an unlimited memory)
- A RW tape head

At each step, the machine must:

- read the current symbol
- **write on the current square**
- **move left or right**
- choose its next state

Definition of Deterministic Turing Machine

A DTM M is a sixtuple $(K, \Sigma, \Gamma, \delta, s, H)$

- K is a finite set of states;
- Σ is the input alphabet, which does not contain ❑ (blank);
- Γ is the tape alphabet, which must contain ❑ and have Σ as a subset.
- $s \in K$ is the initial state;
- $H \subseteq K$ is the set of halting states;
- δ is the transition **function**:

Definition of Deterministic Turing Machine

δ is the **transition function**:

$(K - H) \times \Gamma$ to $K \times \Gamma \times \{\rightarrow, \leftarrow\}$

non-halting state × tape char to state × tape char × action (R or L)

Deterministic Turing Machine

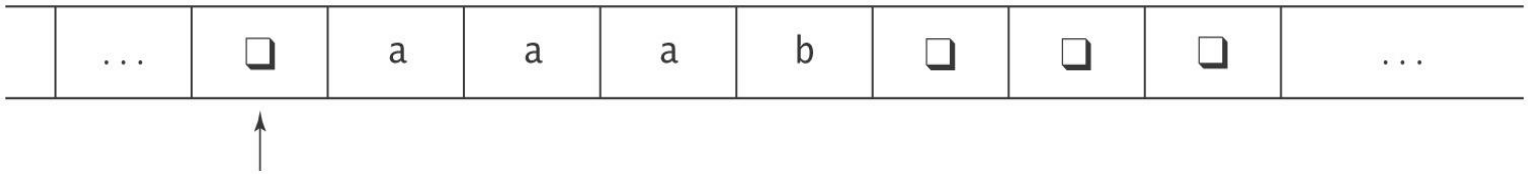
- The input tape is infinite in both directions.
- δ is a function, not a relation. So this is a definition for deterministic Turing machines.
- δ must be defined for all state, input pairs unless the state is a halting state.
- Turing machines do not necessarily halt. To halt, they must enter a halting state. Otherwise they loop.
- Turing machines generate output (*the contents of its tape when halts*) so they can compute functions.

Example 17.1

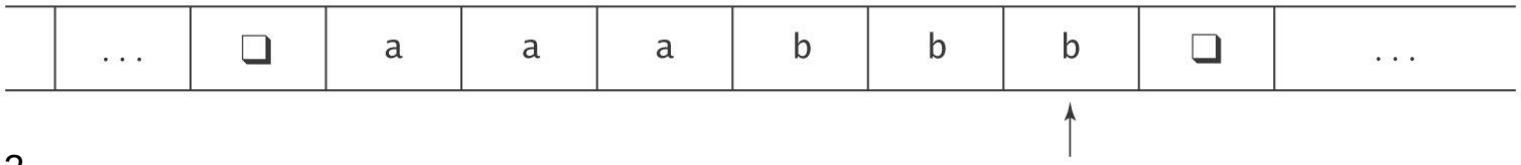
TM *M* takes as input a string in the language $\{a^i b^j, 0 \le j \le i\}$, and adds b's as required to make the number of b's equal the number of a's.

❑ aaab ❑ ❑ ❑

The input:



The output should be:



Example 17.1

TM M: An informal description!

❑ aaab ❑ ❑ ❑

$K = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, ❑, \$, \#\}$,
$s = 1$, $H = \{6\}$, $\delta =$

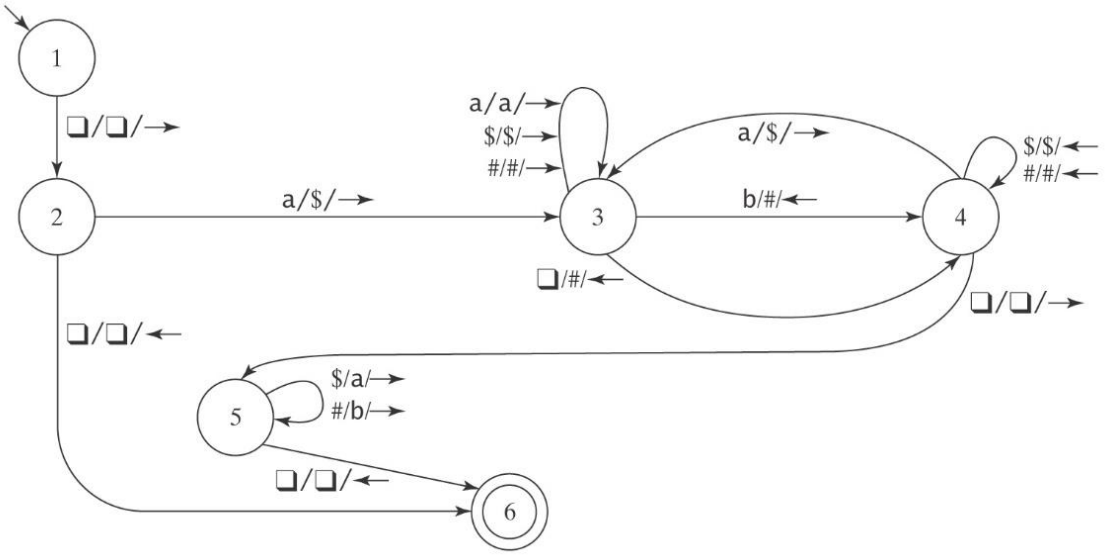
1.
2. loop
3.
4.

Example 17.2

TM M: A graphical notation (Transition Diagram)!

❑ aaab ❑ ❑ ❑

$K = \{1, 2, 3, 4, 5, 6\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, ❑, \$, \#\}$,
$s = 1$, $H = \{6\}$, $\delta =$



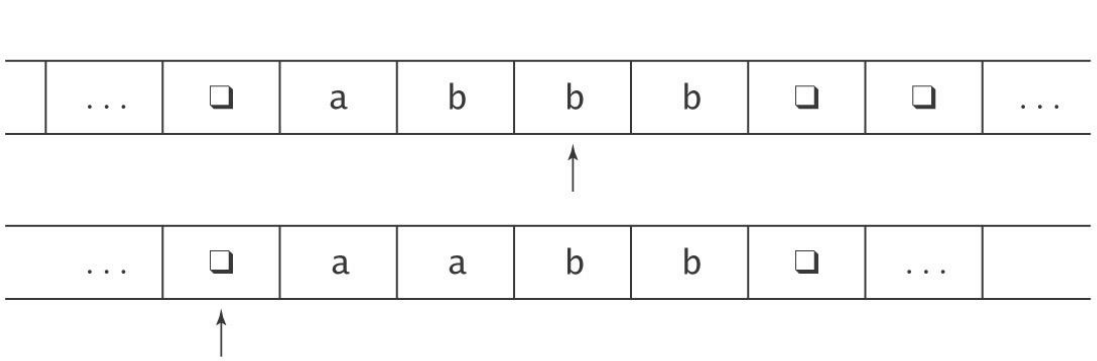
Configurations

A *configuration* of a Turing machine $M = (K, \Sigma, \Gamma, s, H)$ is an element of:

$$K \times ((\Gamma - \{\square\})\ \Gamma^*) \cup \{\varepsilon\} \times \Gamma \times (\Gamma^*\ (\Gamma - \{\square\})) \cup \{\varepsilon\}$$

state	up to scanned square	scanned square	after scanned square

Example 17.3



Initial configuration is $(s, \sqcup w)$.

Yields

$(q_1, w_1) \vdash_M (q_2, w_2)$ iff (q_2, w_2) is derivable, via δ, in one step.

For any TM M, let $\vdash_M^*$ be the *reflexive, transitive closure* of $\vdash_M$.

Configuration C_1 *yields* configuration C_2 if:

$C_1 \vdash_M^* C_2$.

Computations

A *path* through M is a sequence of configurations $C_0, C_1, \ldots, C_n$ for some $n \geq 0$ such that C_0 is the initial configuration and:

$$C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n.$$

A *computation* by M is a path that halts.

If a computation is of *length n* or has n steps, we write: $C_0 \vdash_M^n C_n$

Halting

- A DFSM *M*, on input *w*, is guaranteed to halt in |*w*| steps.
- A PDA *M*, on input *w*, is not guaranteed to halt. But there exists an algorithm to construct an equivalent PDA *M′* that is guaranteed to halt.
- A TM *M*, on input *w*, is not guaranteed to halt. And there exists no algorithm to construct an equivalent TM that is guaranteed to halt.

TM Computation and Halting

- Halt and Accept
- Halt and Reject
- Not Halt and Loop

TMs as Language Recognizers

TMs as Language Recognizers

The input string w on the tape:

❑*w*❑, *w* contains no ❑s

The initial configuration of *M*:

(*s*, ❑*w*)

Let $M = (K, \Sigma, \Gamma, \delta, s, \{y, n\})$.

- *M accepts* a string *w* iff $(s, \text{❑}w) \vdash_M^* (y, w')$ for some string *w'*.
- *M rejects* a string *w* iff $(s, \text{❑}w) \vdash_M^* (n, w')$ for some string *w'*.

Deciding a Language

TM M ***decides*** a language $L \subseteq \Sigma^*$ iff for any string $w \in \Sigma^*$:

if $w \in L$ then TM *M* accepts *w*, and
if $w \notin L$ then TM *M* rejects *w*.

TM M will always halt on all inputs!

Decidable Languages D

A language *L* is ***decidable*** *or* ***Turing-decidable*** *or* ***recursive*** iff there is a Turing Machine *M* that decides it.

We say that *L* is in ***D*** ***(****or* ***R)*** the set of all decidable languages.

Decidable Languages D

- Decidable Languages
- Solvable Languages
- Computable Languages
- Recursive Languages
- Turing-Decidable Languages

Example 17.8

L= $A^nB^nC^n$ = {$a^n b^n c^n$: $n \geq 0$} is **decidable**?

TM decides L?

TM M: An informal description!

❑ ❑ ❑ ❑
❑abc ❑ ❑ ❑
❑aabbcc ❑ ❑ ❑
❑abcc ❑ ❑ ❑

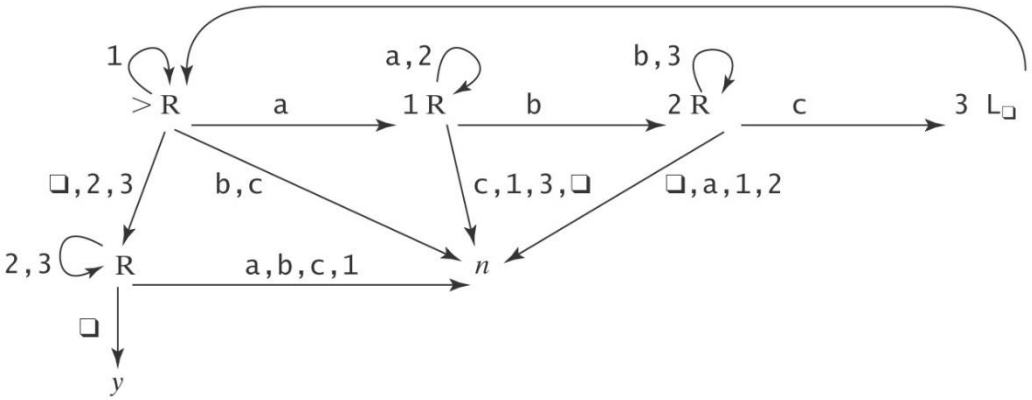
Example 17.8

L= $A^nB^nC^n = \{a^n b^n c^n : n \geq 0\}$ is **decidable**?

TM decides L?

TM M: A graphical notation!

❑ ❑ ❑ ❑
❑abc ❑ ❑ ❑
❑aabbcc ❑ ❑ ❑
❑abcc ❑ ❑ ❑



Semideciding a Language

TM M ***semidecides*** *(or* ***recognizes)*** $L \subseteq \Sigma_M^*$ iff for any string $w \in \Sigma_M^*$:

if $w \in L \rightarrow$ TM M accepts w

if $w \notin L \rightarrow$ TM M does not accept w.

TM M may either reject or fail to halt (loop)!

Semi-Decidable Languages SD

A language *L* is ***semidecidable*** *or* ***Turing-recognizable*** *or* ***recursively-enumerable*** iff there is a Turing Machine that semidecides it.

We say that ***SD (or RE) -*** the set of all semidecidable languages.

Semi-Decidable Languages SD

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

Example 17.10

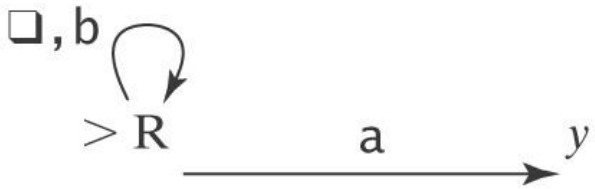
L = b*a(a ∪ b)* is **semidecidable**?

TM M semidecides (recognizes) L?

Loop:

1.1 Move one square to the right.

1.2 If the character under the read/write head is an a, halt and accept.



Example 17.10

L = b*a(a ∪ b)*

TM M decides L?

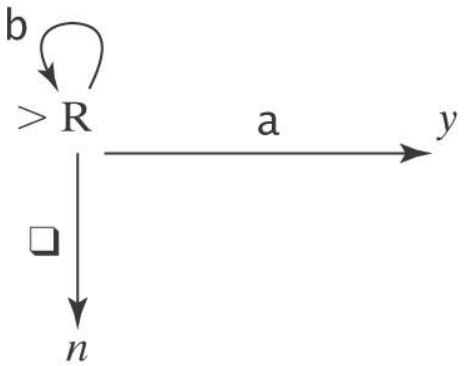
Loop:

1.1 Move one square to the right.

1.2 If the character under the read/write head is

an a. halt and accept.

t and reject.



TMs Compute Functions

TMs Compute Functions

TM $M = (K, \Sigma, \Gamma, \delta, s, \{h\})$.

Its initial configuration is $(s, \sqcup w)$.

Define $M(w) = z$ iff $(s, \sqcup w) \vdash_M^* (h, \sqcup z)$.

$\Sigma' \subseteq \Sigma$ = M's output alphabet.

f = any function from Σ^* to Σ'^*.

TMs Compute Functions

TM *M* ***computes*** *f* iff for all $w \in \Sigma^*$:

- If *w* is an input on which *f* is defined: $M(w) = f(w)$.
- Otherwise $M(w)$ does not halt.

A function *f* is ***recursive*** or ***computable*** iff there is a Turing Machine *M* that computes it and that **always halts**.

- Recursive Functions
- Computable Functions

Example 17.12

succ(n) = n + 1

Represent *n* in binary, i.e., $n \in 0 \cup 1\{0, 1\}^*$

Input: ❑*n*❑ Output: ❑*n*+1❑

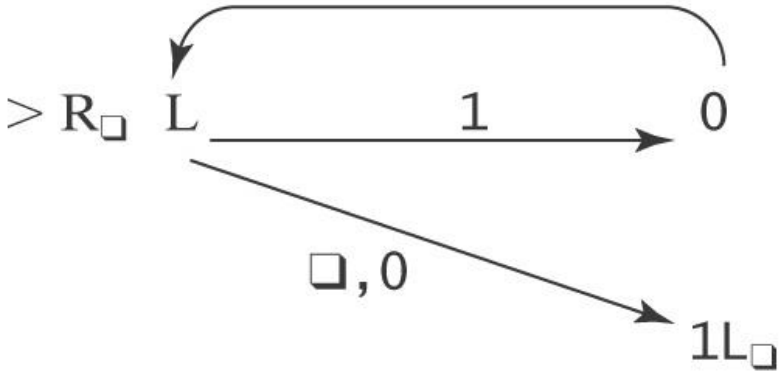
❑1111❑ Output: ❑10000❑

TM computes succ?

TM M:

1.

2.



Variants of TMs - Extensions

Variants of TMs - Extensions

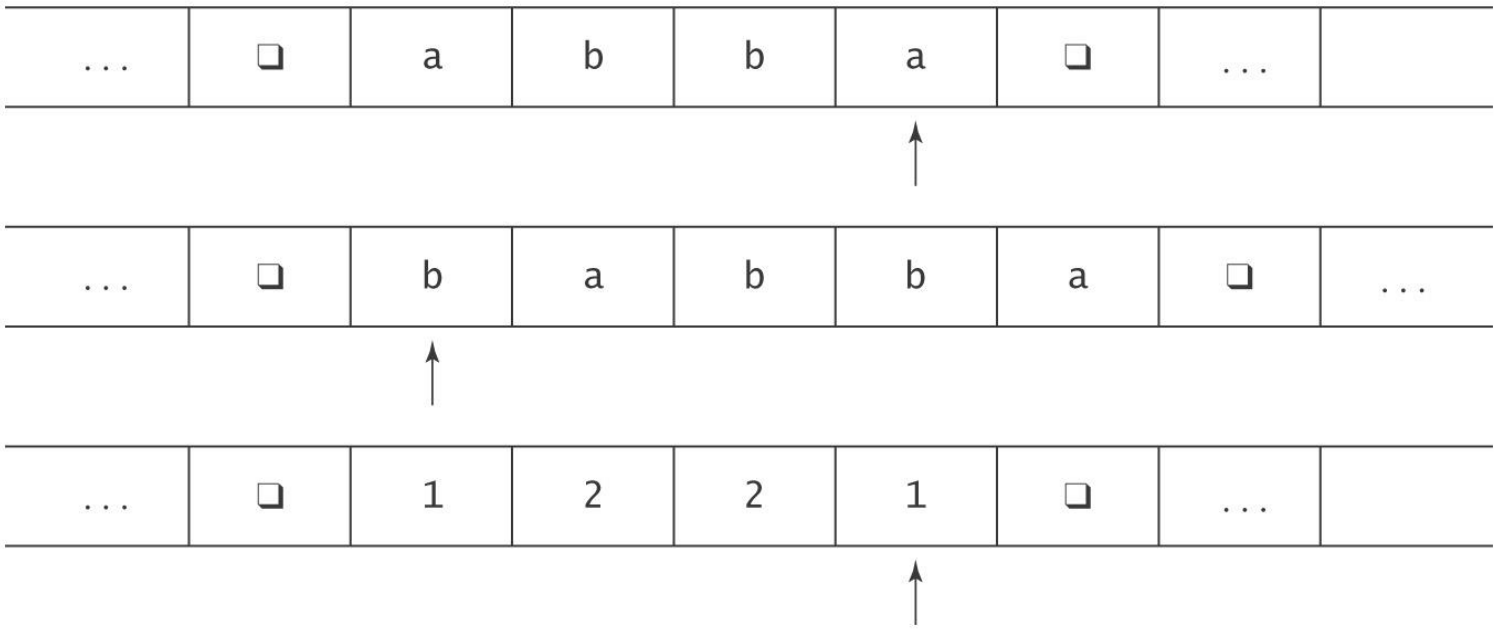
There are many extensions we might like to make to our basic Turing machine model.

Some possible extensions:

- **Multiple-tape** TMs
- **Nondeterministic** TMs

✓ Every extended Turing machine has an equivalent basic Turing machine!

Multi-tape Turing Machines



Multiple Tapes

The transition function for a *k*-tape Turing machine:

$$
\begin{array}{lll}
((K\text{-}H) , \Gamma_1 & \text{to} & (K , \Gamma_{1'}, \{\leftarrow, \rightarrow, \uparrow\} \\
, \Gamma_2 & & , \Gamma_{2'}, \{\leftarrow, \rightarrow, \uparrow\} \\
, \ . & & , \ . \\
, \ . & & , \ . \\
, \Gamma_k) & & , \Gamma_{k'}, \{\leftarrow, \rightarrow, \uparrow\})
\end{array}
$$

Input: as before on tape 1, others blank.
Output: as before on tape 1, others ignored.

Note: tape head is allowed to stay put.

Equivalence of One-tape DTM and Multi-tape DTM

One-tape DTM = Multi-tape DTM

Adding Tapes Adds No Power

Theorem 17.1 Let *M* be *a k-tape Turing machine* for some $k \geq 1$. Then there is *a standard TM M'* where $\Sigma \subseteq \Sigma'$, and:

- On input *x*, *M* halts with output *z* on the first tape iff *M'* halts in the same state with *z* on its tape.
- On input *x*, if *M* halts in ***n* steps**, *M'* halts in $\mathcal{O}(n^2)$ steps.

Proof Idea:
Proof by Construction.

Nondeterministic Turing Machines

A Nondeterministic TM is a sixtuple (K, Σ, Γ, Δ, s, H) where

Δ the **transition relation** is a *subset* of:

$$((K - H) \times \Gamma) \times (K \times \Gamma \times \{\leftarrow, \rightarrow\})$$

Nondeterministic Deciding

TM $M = (K, \Sigma, \Gamma, \Delta, s, \{y, n\})$ be a nondeterministic TM. Let w be an element of Σ^*.

- M *accepts* w iff *at least one* of its computations accepts.
- M *rejects* w iff all of its computations reject.

M ***decides*** a language $L \subseteq \Sigma^*$ iff, $\forall w$:

- There is a finite number of paths that M can follow on input w,
- All of those paths halt, and
- $w \in L$ iff M accepts w.

Nondeterministic Semideciding

TM $M = (K, \Sigma, \Gamma, \Delta, s, H)$ be a nondeterministic TM.

M ***semidecides*** a language $L \subseteq \Sigma^*$ iff for all $w \in \Sigma^*$:

- $w \in L$ iff $(s, \sqcup w)$ yields at least one accepting configuration.

Nondeterministic Function Computation

TM M ***computes*** a function *f* iff, $\forall w \in \Sigma^*$:

- All of *M*'s computations halt, and
- All of *M*'s computations result in $f(w)$.

Equivalence of DTMs and NDTMs

DTM = NDTM

Adding Nondeterminism Adds No Power

Theorem 17.2 If *a nondeterministic TM M* decides or semidecides a language, or computes a function, then there is *a standard TM M'* semideciding or deciding the same language or computing the same function.

Proof Idea:

Proof by construction.

Constructions for deciding/semideciding and for function computation.

Turing Machines and Computers

Simulating a Real Computer by a TM

- An unbounded number of memory cells addressed by the integers starting at 0.
- An instruction set composed of basic operations including load, store, add, subtract, jump, conditional jump, and halt. Here's a simple example program:

```
R       10
MIR     10
CJUMP   1001
A       10111
ST      10111
```

- A program counter.
- An address register.
- An accumulator.
- A small fixed number of special purpose registers.
- An input file.
- An output file.

Simulating a Real Computer by a TM

Theorem 17.4 *A random-access, stored program computer* can be simulated by *a Turing Machine*. If the computer requires *n* steps to perform some operation, the Turing Machine simulation will require $O(n^6)$ steps.

Proof Idea: Proof by construction.

simcomputer will use 7 tapes:

- Tape 1: the computer's memory.
- Tape 2: the program counter.
- Tape 3: the address register.
- Tape 4: the accumulator.
- Tape 5: the op code of the current instruction.
- Tape 6: the input file.
- Tape 7: the output file, initially blank.

The Universal Turing Machine

Encoding TMs as Strings

We need to describe TM $M = (K, \Sigma, \Gamma, \delta, s, H)$ as a string $\langle M \rangle$:

- The states
- The tape alphabet
- The transitions

Example 17.20

Consider M = ({s, q, h}, {a, b, c}, {❑, a, b, c}, δ, s, {h}):

state	symbol	δ
s	❑	(q,❑, →)
s	a	(s,b,→)
s	b	(q,a, ←)
s	c	(q,b, ←)
q	❑	(s,a, →)
q	a	(q,b,→)
q	b	(q,b, ←)
q	c	(h,a, ←)

state/symbol	representation
s	q00
q	q01
h	h10
❑	a00
a	a01
b	a10
c	a11

<M> = (q00,a00,q01,a00,→), (q00,a01,q00,a10,→),
(q00,a10,q01,a01, ←), (q00,a11,q01,a10,←),
(q01,a00,q00,a01,→), (q01,a01,q01,a10,→),
(q01,a10,q01,a11,←), (q01,a11,h11,a01,←)

Enumerating Turing Machines

Theorem 17.7 There exists *an infinite lexicographic enumeration* of:

- All syntactically valid TMs.
- All syntactically valid TMs with specific input alphabet Σ.
- All syntactically valid TMs with specific input alphabet Σ and specific tape alphabet Γ.

The Universal Turing Machine

Problem: All our machines so far are hardwired!

Question: Can we build a programmable TM that accepts as input: *<an arbitrary TM M, an input string w> and simulate the operation of M on w?*

- ✓ Yes, it's called the *Universal Turing Machine*!

Specification of the Universal TM

On input <*M*, *w*>, the Universal Turing Machine *U* must:

- Halt iff *M* halts on *w*.
- If *M* is a deciding or semideciding machine, then: If *M* accepts, accept. If *M* rejects, reject.
- If *M* computes a function, then *U*(<*M*, *w*>) must equal *M*(*w*).

How The Universal TM U Works

The UTM *U* will use 3 tapes:

- Tape 1: *M*'s tape.
- Tape 2: <*M*>, the "program" that *U* is running.
- Tape 3: *M*'s state.

Reading Assignment

Chapter 17:

Sections
17.1
17.2
17.3
17.6
17.7

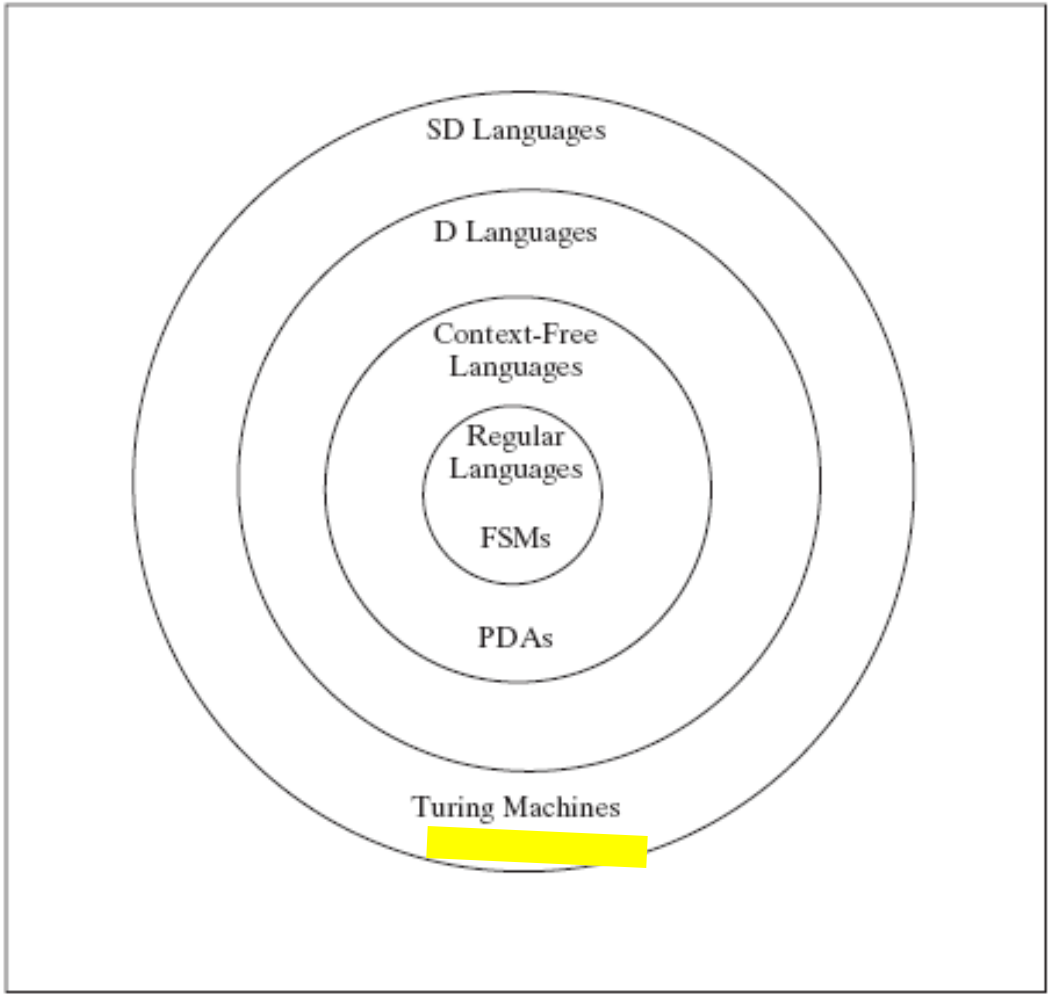
In-Class Exercises

Chapter 17:

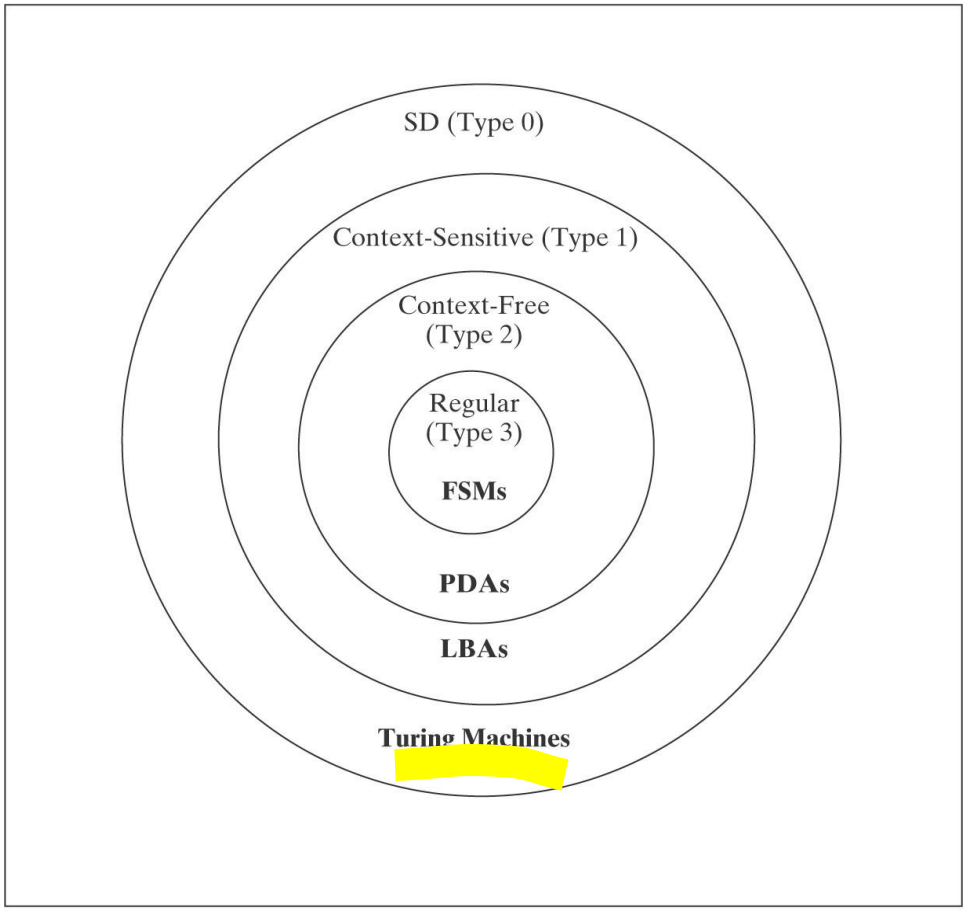
1
8
13
14

Unrestricted Grammars

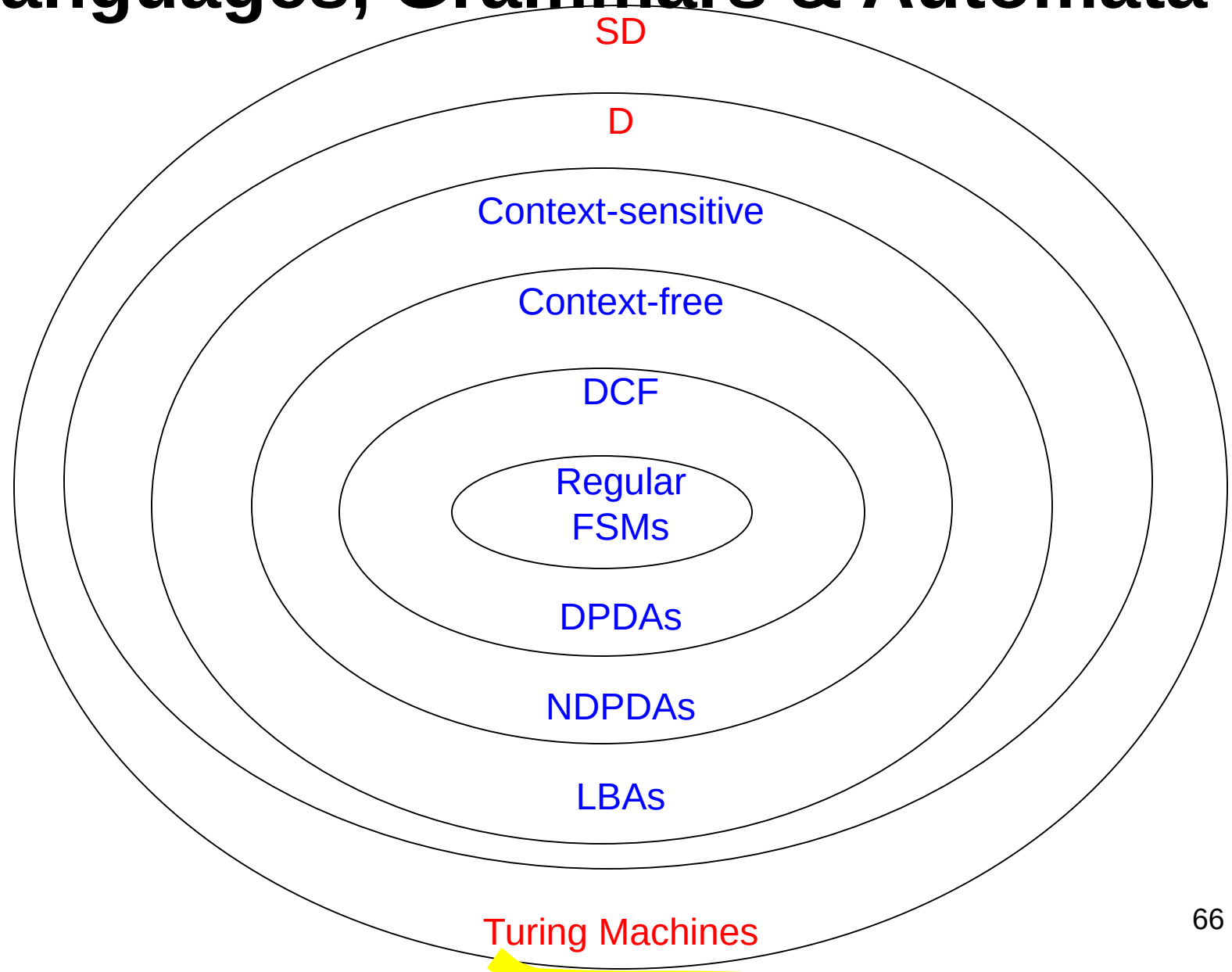
Languages, Grammars & Automata



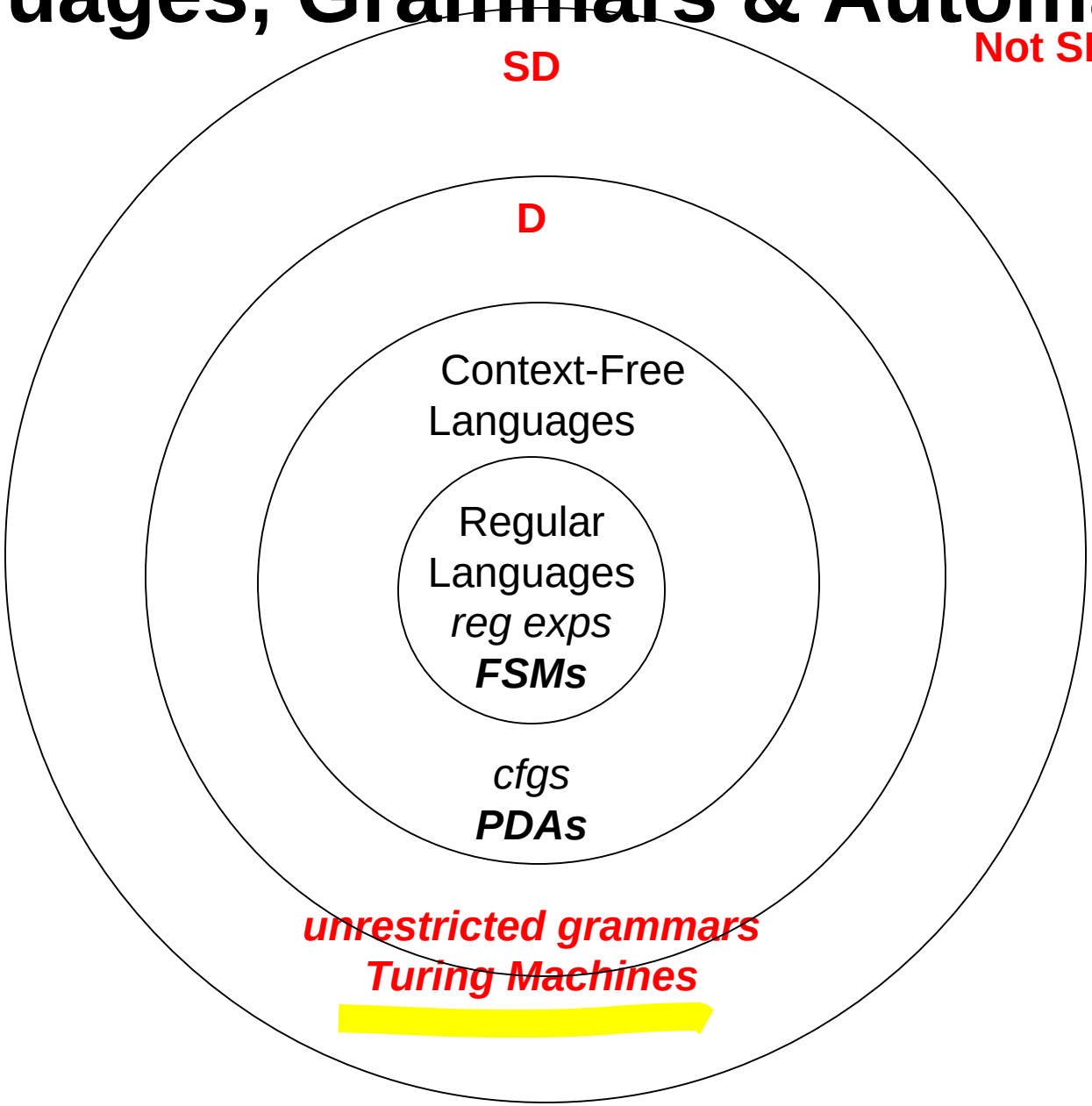
Languages, Grammars & Automata



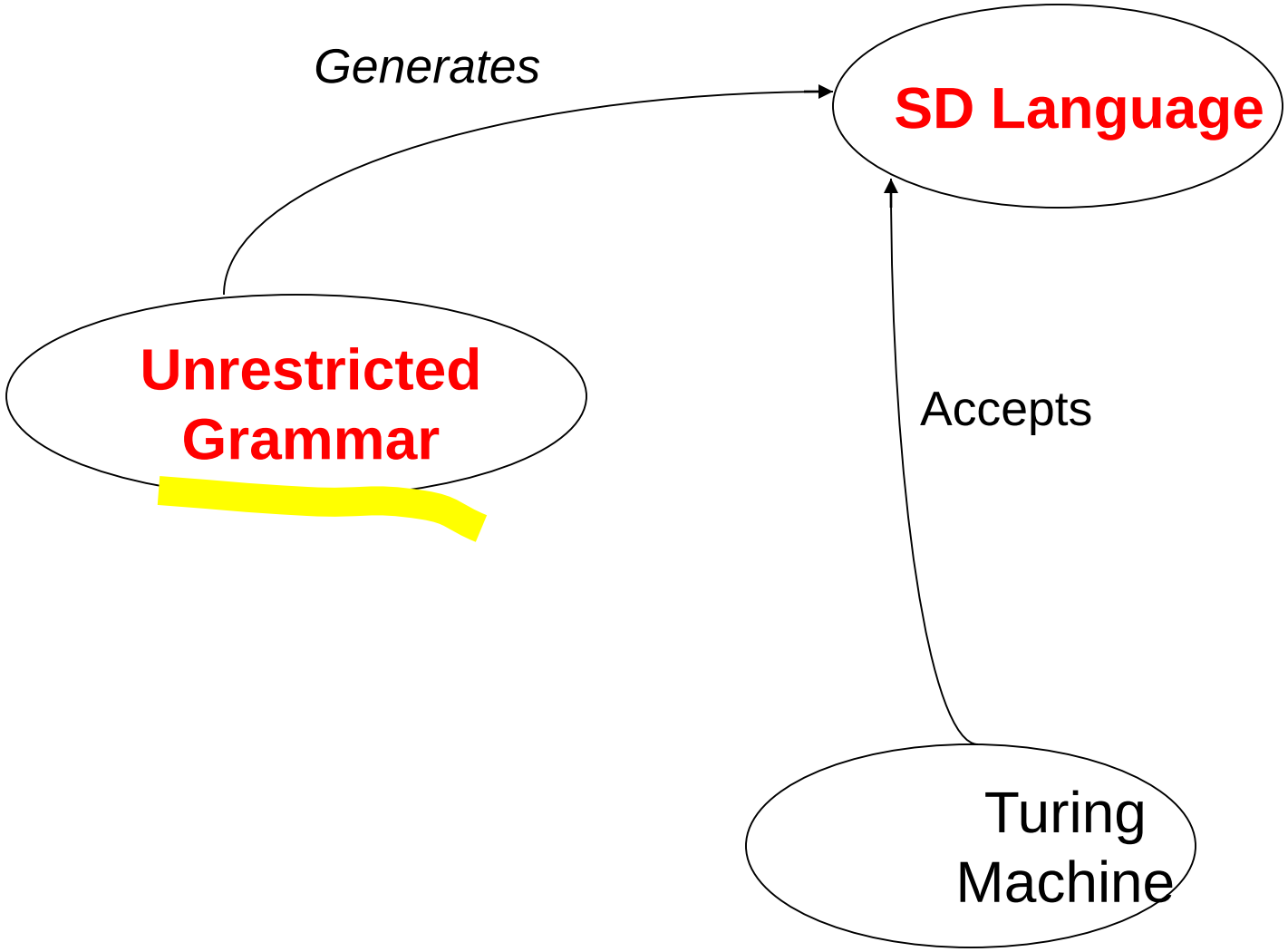
Languages, Grammars & Automata



Languages, Grammars & Automata



Grammars, SD Languages, and TMs



Unrestricted Grammars

An ***unrestricted*** or ***type 0*** or ***phrase structure grammar*** G is a quadruple (V, Σ, R, S) where:

- V is an alphabet,
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite subset of $(V^+ \times V^*)$,
- S (the start symbol) is an element of $V - \Sigma$.

The language generated by G is: $\{w \in \Sigma^* : S \Rightarrow_G^* w\}$.

Example 23.1

$L = A^nB^nC^n = \{a^nb^nc^n, n \geq 0\}$.

UG?

$S \rightarrow aBSc$
$S \rightarrow \varepsilon$
$Ba \rightarrow aB$
$Bc \rightarrow bc$
$Bb \rightarrow bb$

abc
aaabbbccc

Example 23.2

$L = \{w \in \{a, b, c\}^* : \#_a(w) = \#_b(w) = \#_c(w)\}$

UG?

$S \rightarrow ABCS$
$S \rightarrow \varepsilon$
$AB \rightarrow BA$
$BC \rightarrow CB$
$AC \rightarrow CA$
$BA \rightarrow AB$
$CA \rightarrow AC$
$CB \rightarrow BC$
$A \rightarrow a$
$B \rightarrow b$
$C \rightarrow c$

Equivalence of Unrestricted Grammars and Turing Machines

UG = TM = SD

Equivalence of Unrestricted Grammars and Turing Machines

Theorem 23.1 A language is generated by an ***unrestricted grammar*** *if and only if it is semidecided by some* ***Turing Machine*** *M, i.e.,* it is in ***SD***.

Proof Idea:

Proof by Construction

Only if (grammar → TM): by construction of an NDTM.

If (TM → grammar): by construction of a grammar that mimics the behavior of a semideciding TM.

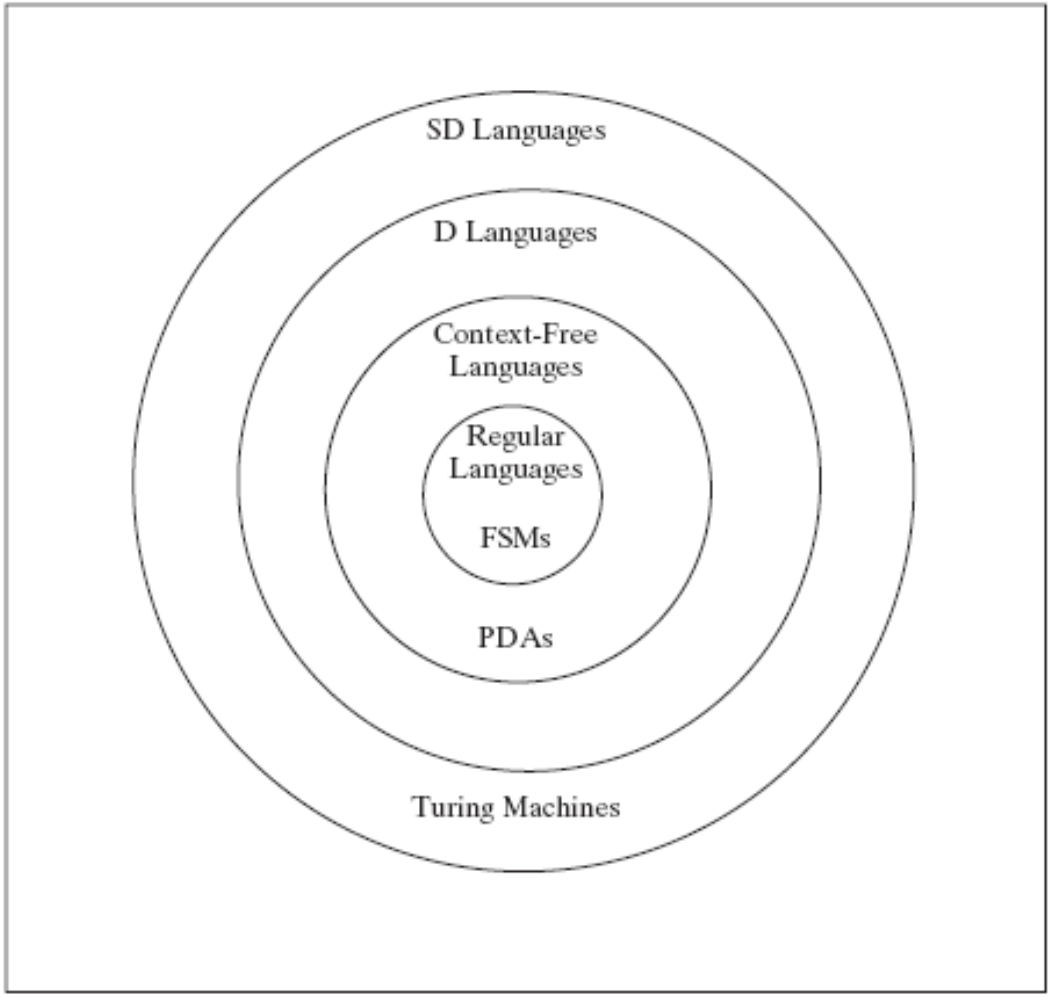
Reading Assignment

Chapter 23:

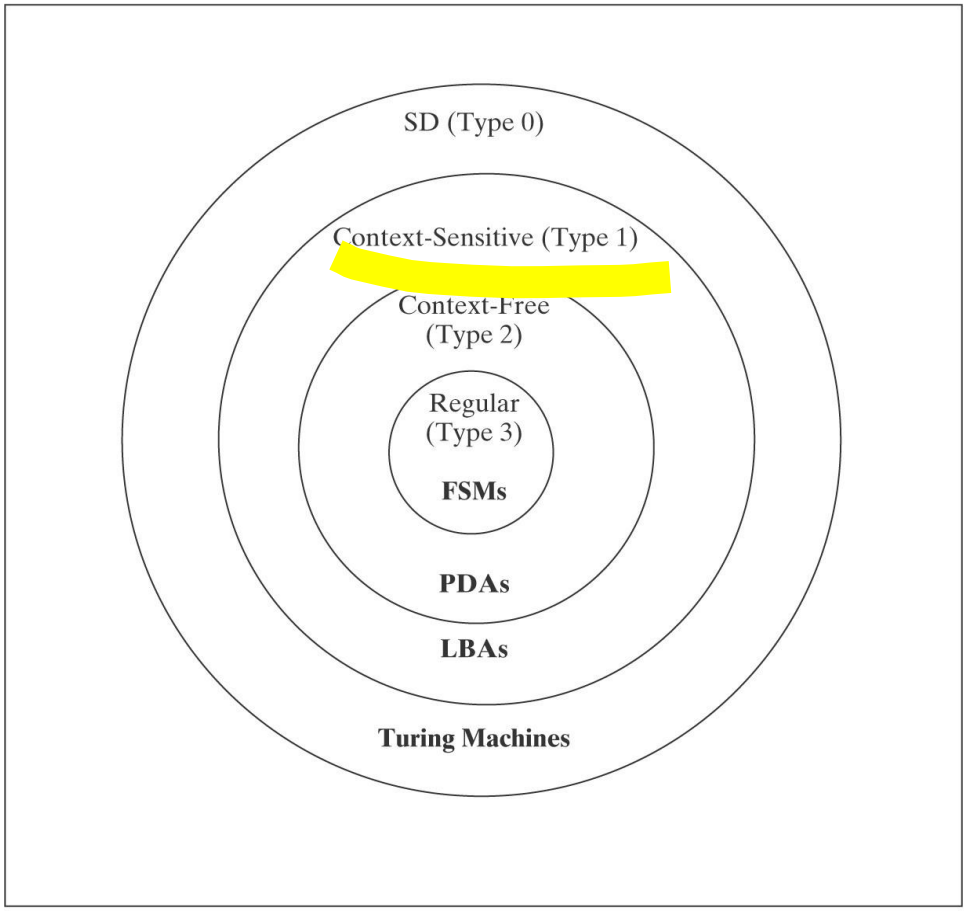
Sections
23.1
23.2

Context-Sensitive Languages, Context-Sensitive Grammars and Linear Bounded Automata (LBA)

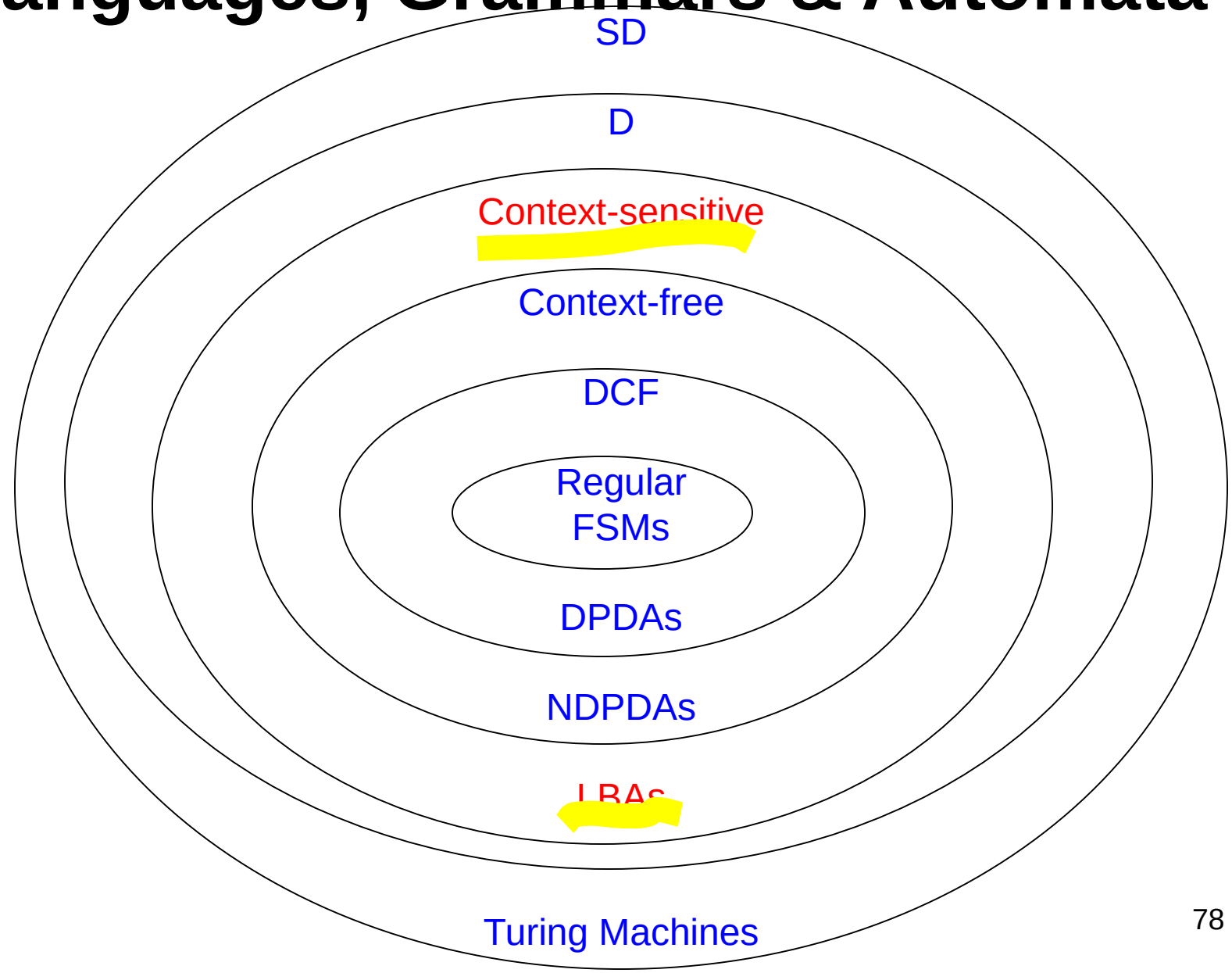
Languages, Grammars & Automata



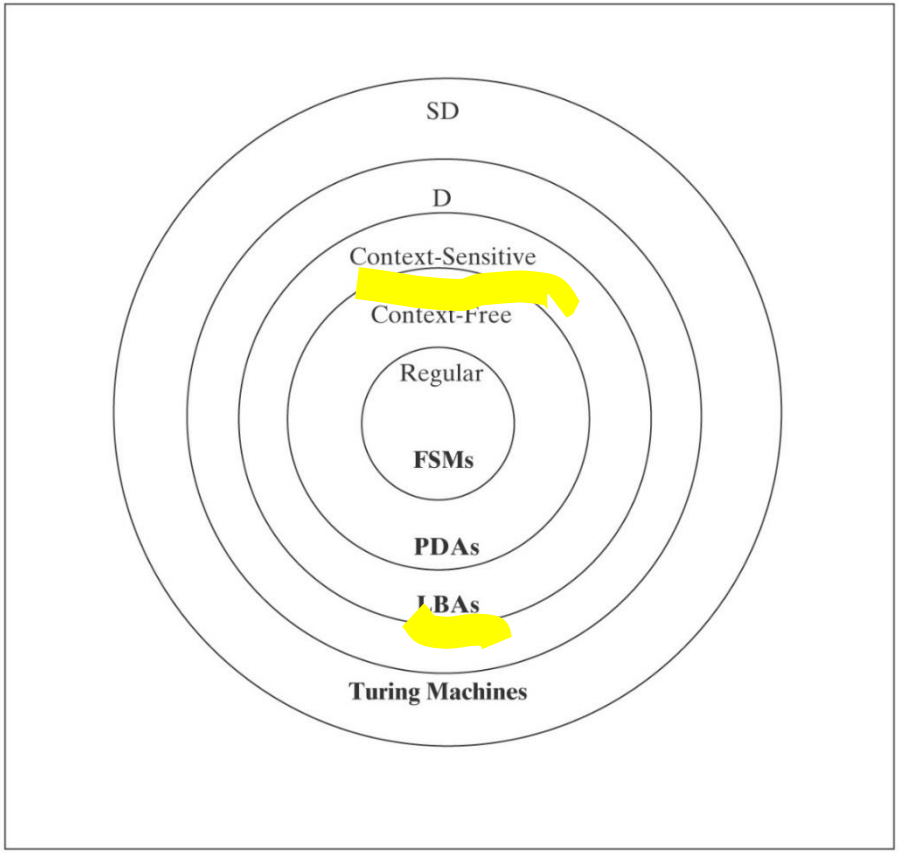
Languages, Grammars & Automata



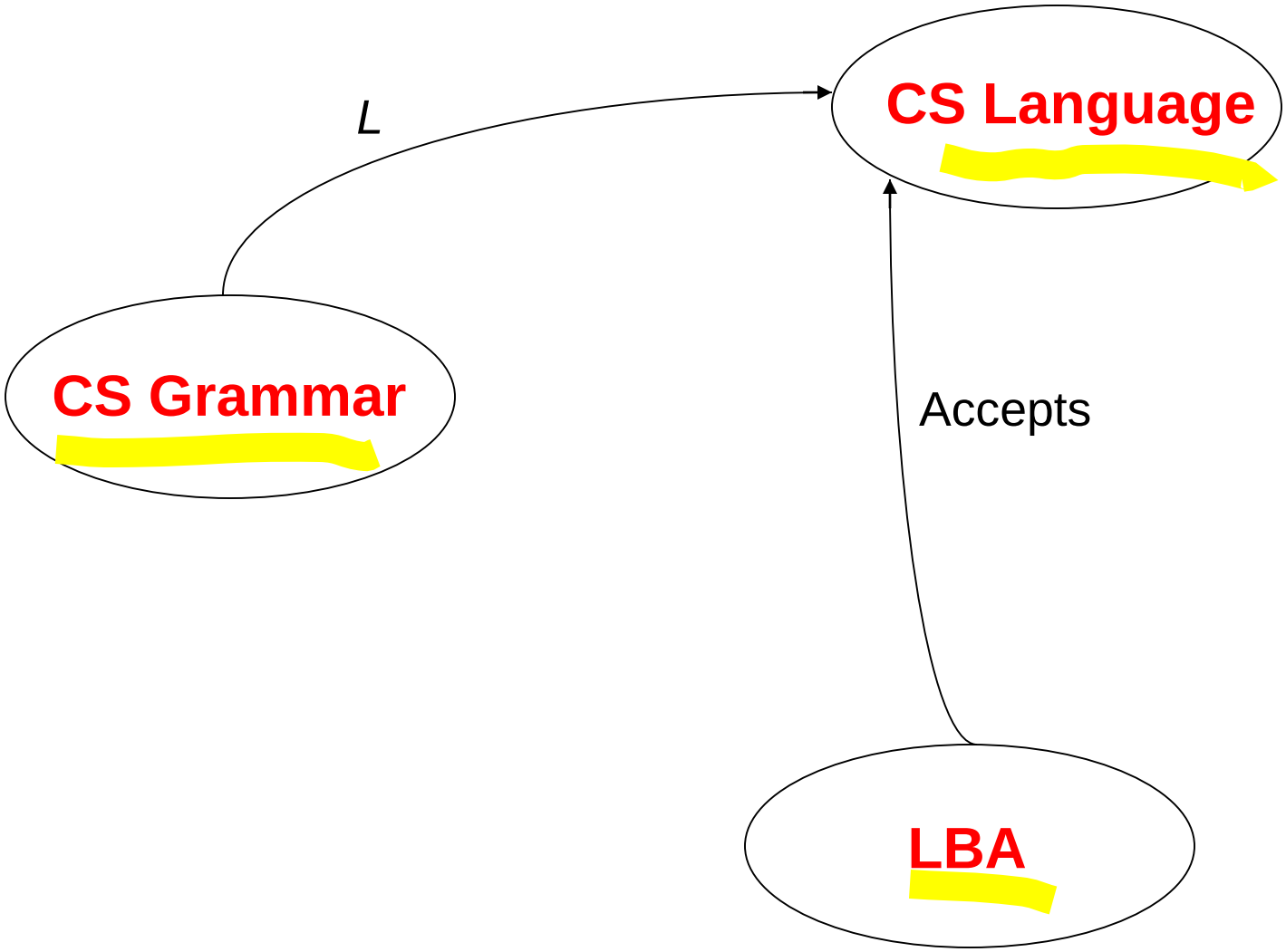
Languages, Grammars & Automata



Is There Anything In Between PDAs and Turing Machines?



Context-Sensitive Grammars, Context-Sensitive Languages, and LBAs

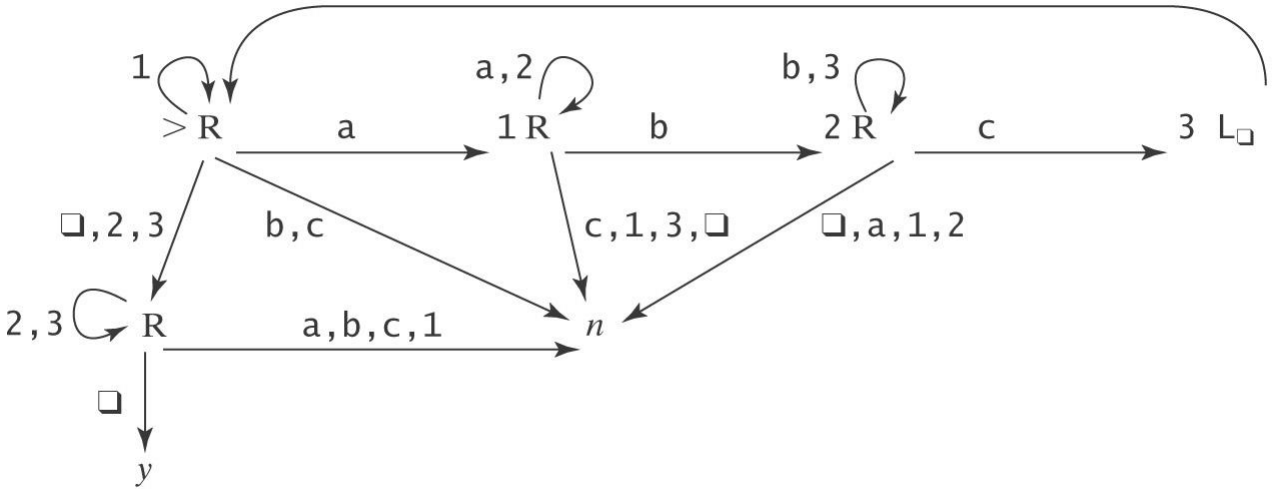


Linear Bounded Automata

A ***linear bounded automaton*** is an NDTM the length of whose tape is equal to $|w| + 2$.

Example: $A^nB^nC^n = \{a^nb^nc^n : n \geq 0\}$

❑aabbcc❑❑❑❑❑❑❑❑❑



Context-Sensitive Languages

A language is ***context sensitive*** iff there exists an **LBA** that accepts it.

CSL = LBA

Note:

It is not known whether, for every nondeterministic LBA there exists an equivalent deterministic one.

Context-Sensitive Grammars

A ***context-sensitive grammar*** $G = (V, \Sigma, R, S)$ is an unrestricted grammar in which R satisfies the following constraints:

- The left-hand side of every rule contains at least one nonterminal symbol.
- No length-reducing rules.

➢ With one exception: R may contain the rule $S \rightarrow \varepsilon$.
If it does, then S does not occur on the right hand side of any rule.

Context-Sensitive Grammars

$L = A^nB^n = \{a^nb^n, n \geq 0\}$.

- A grammar that is not context-sensitive:

  $S \rightarrow aSb$

  $S \rightarrow \varepsilon$

- An equivalent, context-sensitive grammar:

  $S \rightarrow \varepsilon$

  $S \rightarrow T$

  $T \rightarrow aTb$

  $T \rightarrow ab$

Equivalence of Context-Sensitive Languages and Linear Bounded Automata

CSL = LBA = CSG

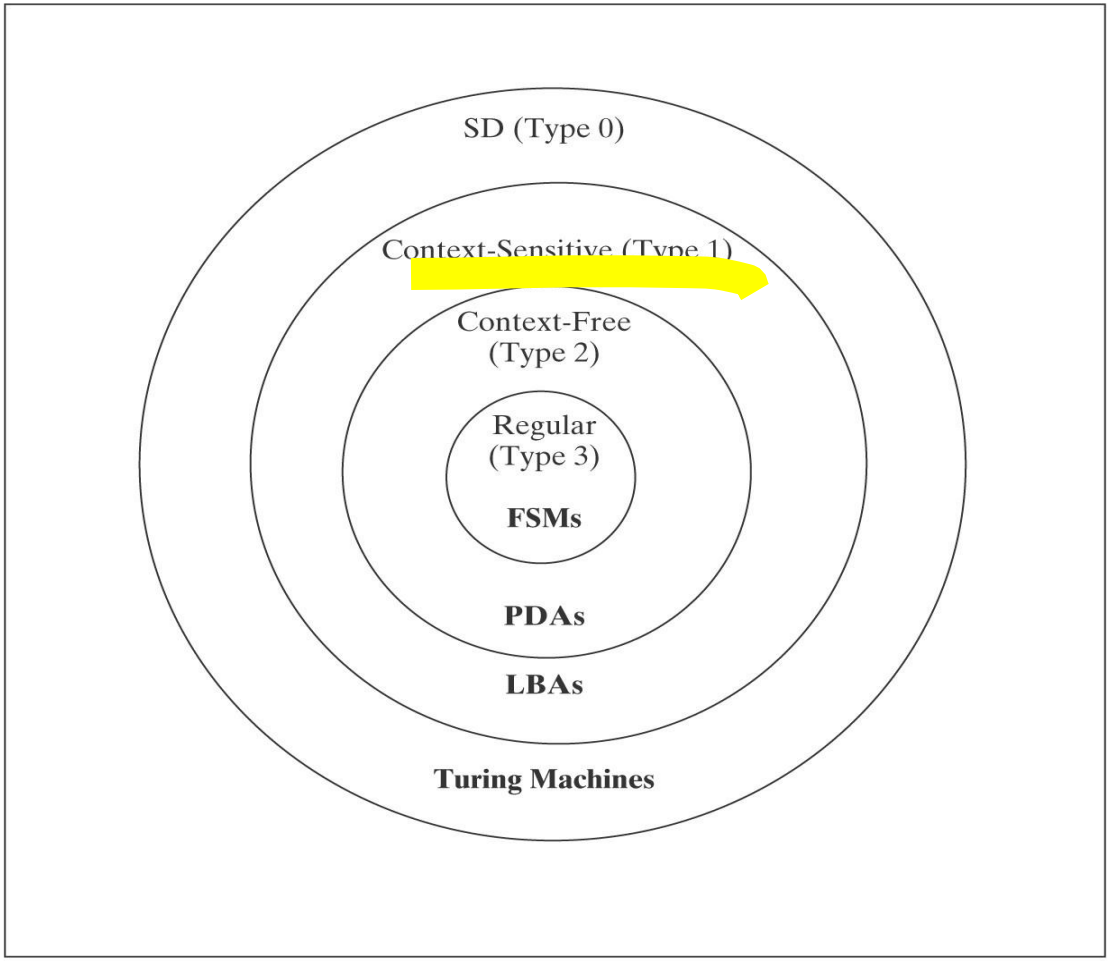
Equivalence of CSG and LBA

Theorem 24.3 The set of languages that can be generated by a ***context-sensitive grammar*** is identical to the class that can be accepted by an ***LBA***.

Context-Sensitive Languages and D

Theorem 24.4 The ***context-sensitive languages*** are a proper subset of ***D***.

The Chomsky Hierarchy



Reading Assignment

Chapter 24:

Sections
24.1