CSC 240

COMPLEXITY 1

Are there natural laws which govern what we can do with computer science?

Which types of problems can be solved by a computer?

Computability Theory

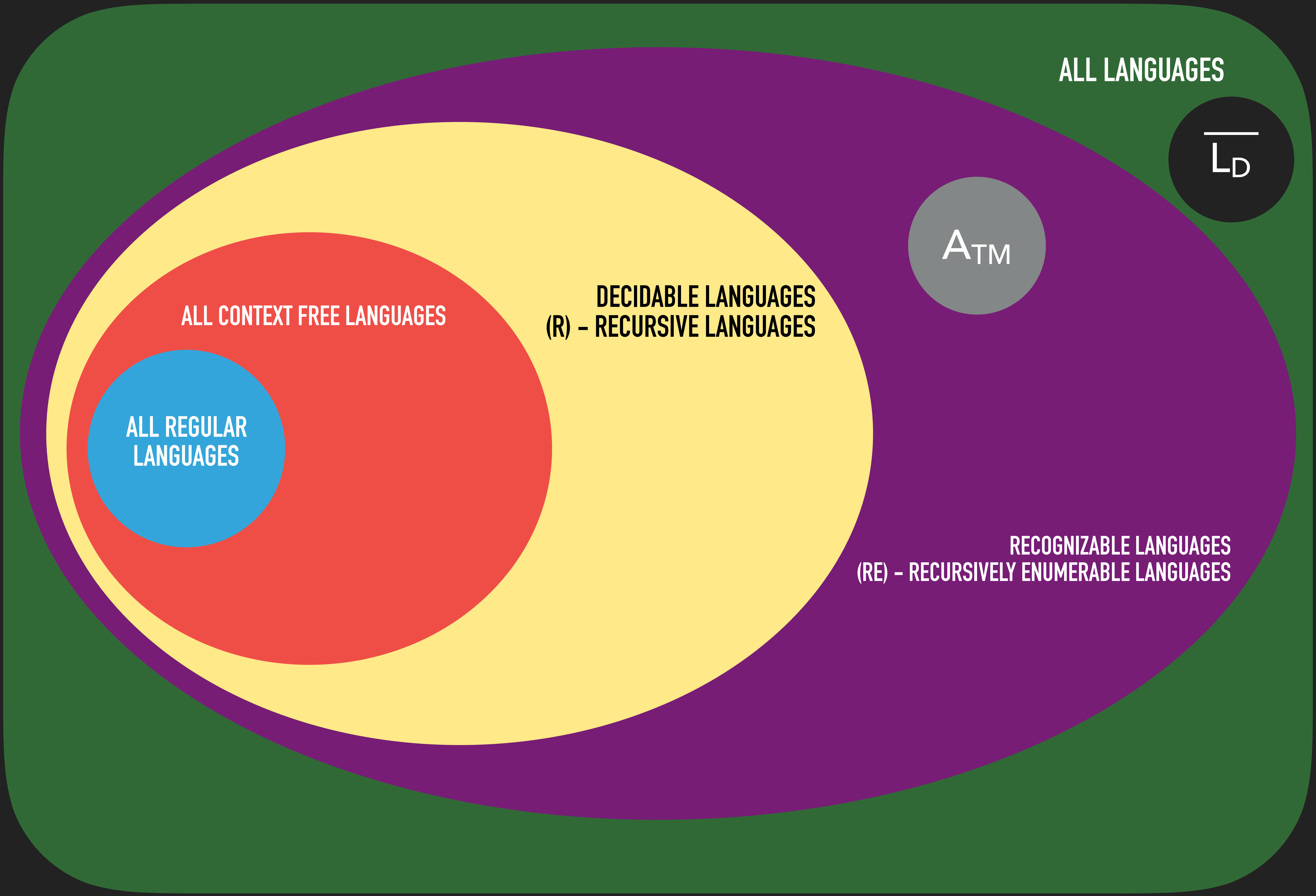
How can we quantify the difficulty of a problem?

Complexity Theory

How can we prove that our answers to these questions are correct?

Discrete Math

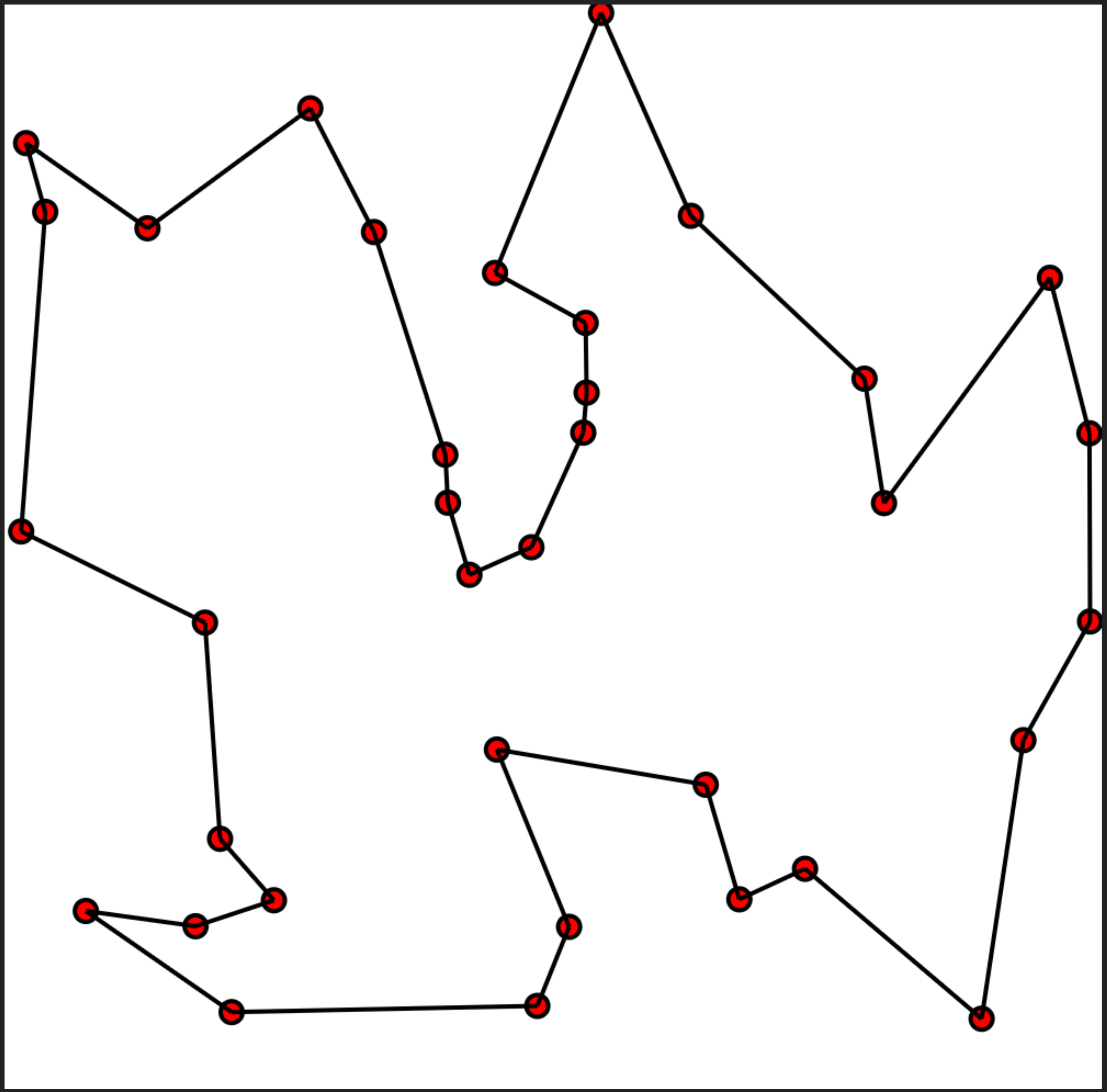
LANGUAGES


ALL LANGUAGES
$\overline{L_D}$
A_{TM}
DECIDABLE LANGUAGES
(R) – RECURSIVE LANGUAGES
ALL CONTEXT FREE LANGUAGES
ALL REGULAR LANGUAGES
RECOGNIZABLE LANGUAGES
(RE) – RECURSIVELY ENUMERABLE LANGUAGES

KEY FACTS

- We now know ways to determine if a problem is solvable.
- Solvable problems can be solved in a finite amount of time.
- However, just because a problem is solvable in a finite amount of time, doesn’t mean we can wait that long.

COMPLEX PROBLEMS



COMPLEX PROBLEMS

Subsequence Problem: Find the longest *increasing* subsequence (Ex: 1, 4, 5, 11)

1	4	9	2	5	11	3	8	12	7

Naive Approach: Try every possible subsequence until you find the longest increasing one.

For a sequence of length n, there are 2^n possibilities.

This means, a sequence of 60 numbers would take 2^{60} attempts.

The universe is less than 2^{59} seconds old.

COMPLEX PROBLEMS

Subsequence Problem: Find the longest *increasing* subsequence (Ex: 1, 4, 5, 11)

1	4	9	2	5	11	3	8	12	7

Another Approach: Patience Sorting Algorithm

1	4	9	11	12
	2	5	8	
		3	7	

O(n log n)

- Searching for some specific type of thing (biggest, smallest, longest, etc…) object in a group usually results in exponentially many options.
- This means that brute-force approaches tend to run in exponential time - $O(2^n)$.
- “Good” algorithms tend to run in polynomial time $O(n)$, $O(n^2)$, $O(n^3)$, etc…
- $O(n^k)$ algorithms tend to scale well as the size of the data (n) increases, while $O(2^n)$ algorithms scale poorly.

"Computational problems can be feasibly computed on some computational device only if they can be computed in polynomial time."

"Languages can be decided efficiently on some Turing machine only if they can be computed in $O(n^k)$."

According to Cobham-Edmonds:

Efficient Algorithms	Inefficient Algorithms
$O(n \log n)$	$O(2^n)$
$O(n^2)$	$O(n!)$
$O(n^3)$	$O(k^n)$
$O(n^{500})$	$O(1.000001^n)$

Not everyone agrees with this thesis!

PROPERTIES OF POLYNOMIALS

Combining polynomial time algorithms in different ways result in polynomial time algorithms.

$$x^a \cdot x^b = x^{(a+b)}$$

$$x^{(n+1)} = x \cdot x^n$$

$$(x^a)^b = x^{(a \cdot b)}$$

$$(x^a)^b = x^{(a \cdot b)} = (x^b)^a$$

$$x^{\frac{1}{2}} = \sqrt{x}$$

$$x^{\frac{1}{a}} = \sqrt[a]{x}$$

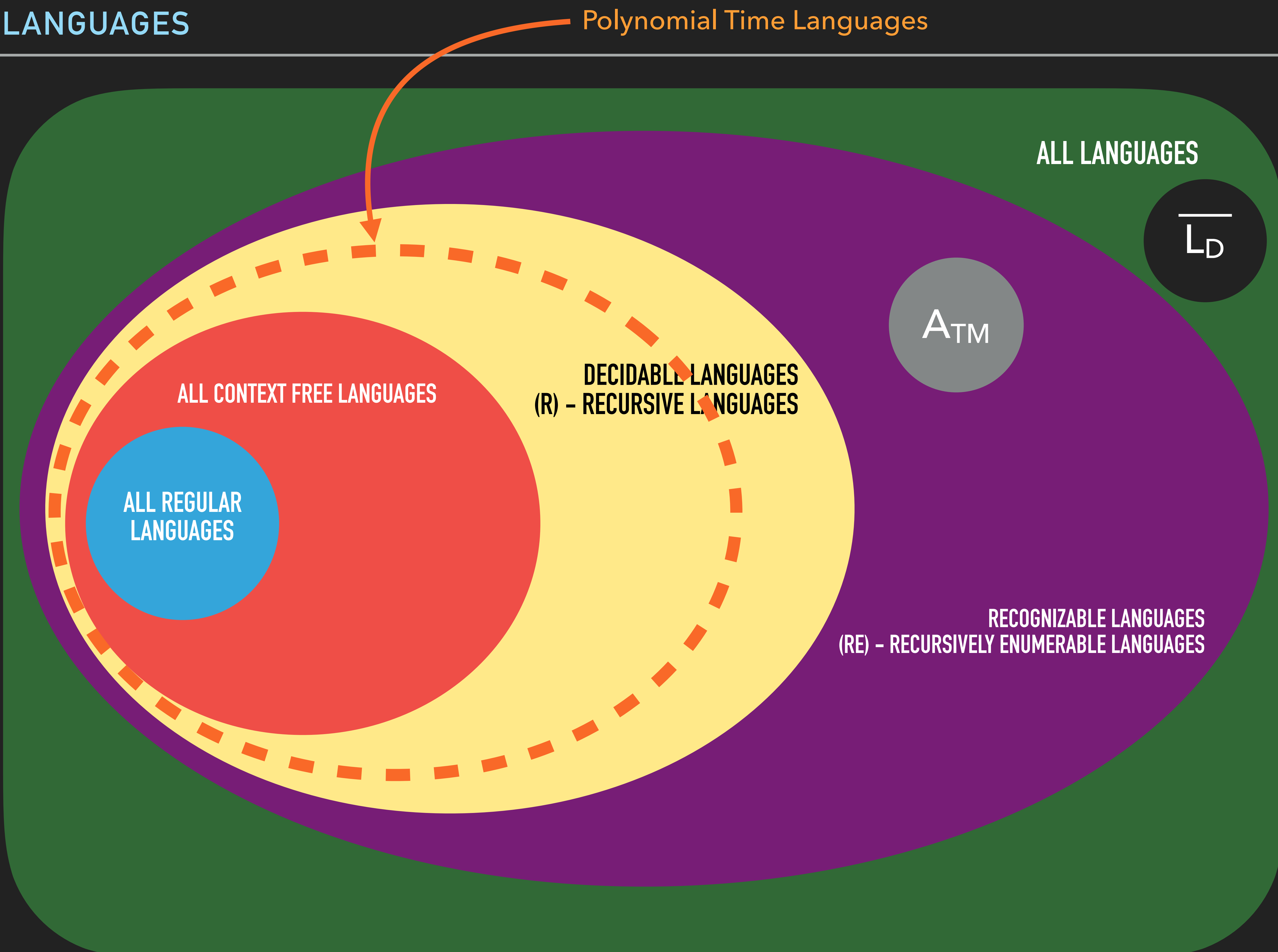
COBHAM-EDMONDS THESIS

"Computational problems can be feasibly computed on some computational device only if they can be computed in polynomial time."

"Languages can be decided efficiently on some Turing machine only if they can be computed in $O(n^k)$."

P = { L | There is a polynomial time decider for L }

LANGUAGES


Polynomial Time Languages
ALL LANGUAGES
$\overline{L_D}$
A_{TM}
DECIDABLE LANGUAGES
(R) – RECURSIVE LANGUAGES
ALL CONTEXT FREE LANGUAGES
ALL REGULAR LANGUAGES
RECOGNIZABLE LANGUAGES
(RE) – RECURSIVELY ENUMERABLE LANGUAGES