Complexity Theory Introduction

Charles E. Hughes

COT6410 – Spring 2021 Notes

Who, What, Where and When

- **Instructor: Charles Hughes;**
  HEC-247C (but will be virtual)
  charles.hughes@ucf.edu
  (e-mail is a good way to get me)
  Use Subject: COT6410
  Office Hours: T 10:45AM-11:45AM; R 12:15PM-1:30PM
- **Web Page: https://www.cs.ucf.edu/courses/cot6410/Spring2021**
- **Meetings: TR 9:00AM-10:15AM, Virtual**
  28 periods, each 75 minutes long.
  Final Exam (Thursday, April 29 from 7:00AM to 9:50AM) is separate from class meetings
- **GTA: Daniel Gibney**
  Use Subject: COT6410
  Office Hours: MW 1:30-3:00

Zoom Meeting Links

- Class (TR 9:00-10:15)
  - https://ucf.zoom.us/j/92647255891?pwd=YVdSVWY2cWVWSUhXM3ZtbXFCZE4xQT09
- My Office Hours (T 10:45-11:45; R 12:15-1:30)
  - https://ucf.zoom.us/j/94885874930?pwd=UW10elpWbzM0SzFRenQ4RW4zVVpGQT09
- GTA's Office Hours (MW 1:30-3:00)
  - https://ucf.zoom.us/j/94756865717?pwd=c0NzNGlXZU9kRGxnOTJCZmtBWkhZUT09

Text Material

References:

- **My Notes and videos**
- **Sipser, Introduction to the Theory of Computation 3rd Ed., Cengage Learning, 2013.**
- **Hopcroft, Motwani&Ullman, Intro to Automata Theory, Languages and Computation 3rd Ed., Prentice-Hall, 2006.**
- Cooper, Computability Theory 2nd Ed., Chapman-Hall/CRC Mathematics Series, 2003.
- Garey&Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., 1979.
- Davis, Sigal&Weyuker, Computability, Complexity and Languages 2nd Ed., Acad. Press (Morgan Kaufmann), 1994.
- Papadimitriou & Lewis, Elements of the Theory of Computation, Prentice-Hall, 1997.
- Bernard Moret, The Theory of Computation, Addison-Wesley, 1998.
- Oded Goldreich, Computational Complexity: A Conceptual Approach, Cambridge University Press, 2008.
- Oded Goldreich, P, NP, and NP-Completeness: The Basics of Complexity Theory, Cambridge University Press, 2010.
- Arora&Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2009.
- Various other people's notes on web

Goals of Course

- Introduce Computability and Complexity Theory, including
  - Review background on automata and formal languages
  - Basic notions in theory of computation
    - Algorithms and effective procedures
    - Decision and optimization problems
    - Decision problems have yes/no answer to each instance
  - Limits of computation
    - Turing Machines and other equivalent models
    - Determinism and non-determinism
    - Undecidable problems
    - The technique of reducibility; The ubiquity of undecidability (Rice's Theorem)
    - The notions of semi-decidable (re) and of co-re sets
  - Complexity theory
    - Order notation (quick review)
    - Polynomial reducibility
    - Time complexity, the sets P, NP, co-NP, NP-complete, NP-hard, etc., and the question does P=NP? Sets in NP and NP-Complete.
    - Gadgets and other reduction techniques

Expected Outcomes

- You will gain a solid understanding of various types of computational models and their relations to one another.
- You will have a strong sense of the limits that are imposed by the very nature of computation, and the ubiquity of unsolvable problems throughout CS.
- You will understand the notion of computational complexity and especially of the classes of problems known as P, NP, co-NP, NP-complete and NP-Hard.
- You will (hopefully) come away with stronger formal proof skills and a better appreciation of the importance of discrete mathematics to all aspects of CS.

Keeping Up

- I expect you to visit the course web site regularly (preferably daily) to see if changes have been made or material has been added.
- Attendance is preferred, although I do not take roll.
- I do, however, ask lots of questions in class and give lots of hints about the kinds of questions I will ask on exams. It would be a shame to miss the hints, or to fail to impress me with your insightful in-class answers.
- You are responsible for all material covered in class, whether in the notes or not.

Rules to Abide By

- Do Your Own Work
  - When you turn in an assignment, you are implicitly telling me that these are the fruits of your labor. Do not copy anyone else's homework or let anyone else copy yours. In contrast, working together to understand lecture material and solutions to problems not posed as assignments is encouraged.
- Late Assignments
  - I will accept no late assignments, except under very unusual conditions, and those exceptions must be arranged with me in advance unless associated with some tragic event.
- Exams
  - No communication during exams, except with me or a designated proctor, will be tolerated. A single offense will lead to termination of your participation in the class, and the assignment of a failing grade.

Grading

- Grading of Assignments and Exams
  - I will endeavor to return the midterm exam within a week of its taking place and each assignment within a week of its due date.
- Exam Weights
  - The weights of exams will be adjusted to your personal benefits, as I weigh the exam you do well in more than one in which you do less well.
- Paper, Presentation Slides, Video
  - This should be informative for you and me.

Important Dates

- Midterm – Thursday, March 11 (tentative)
- Withdraw Deadline – Friday, March 26
- Spring Break – April 11-18
- Final – Tues., April 29, 7:00AM – 9:50AM

Evaluation (tentative)

- MidTerm – 125 points ; Final – 125 points
- Assignments – 75 points
- Paper and Presentation – 125 points
- Extra – 50 points used to increase weight of exams or maybe paper/presentation, always to your benefit
- Total Available: 500 points
- Grading will be A >= 90%, B+ >= 85%,
  B >= 80%, C+ >= 75%, C >= 70%,
  D >= 50%, F < 50% (Minuses might be used)

Decision Problems

- A set of input data items ("instances" from some universe of discourse, e.g., natural numbers, strings over alphabet, graphs, …)
- Each input data item defines a question with an answer Yes/No or True/False or 1/0.
- A decision problem can be viewed as a relation between its universe of discourse and its binary range
- A decision problem can also be viewed as a partition of the universe of discourse into those that give rise to true instances and those that give rise to false instances.
- In each case, we seek an algorithmic solution (in the form of a predicate) or a proof that none exists
- When an algorithmic solution exists, we seek an efficient algorithm, or proofs of the problem's inherent complexity
- Sometimes, when the problem appears to be intractable, we seek out fast heuristic approximate solutions

Instances vs Problems

- Each instance has an *'answer*.‘
  - An instance’s answer is the solution of the instance - it is *not* the solution of the problem.
  - A solution of the problem is a computational procedure that finds the answer of any instance given to it – the procedure must halt on all instances – it must be an *'algorithm*.'

Assignment # 1 is Required for Financial Aid

Complete questionnaire (in quizzes category) on Webcourses.

Complete all questions on time for a few free points out of total points for all assignments.

Complete and submit by one minute before Midnight Friday, 1/15.

UNIVERSE OF DISCOURSE
USUALLY STRINGS OR NATURAL NUMBERS

DECISION PROBLEMS

S
Subset of interest, maybe with ordered elements

For some element, x, is x in S?

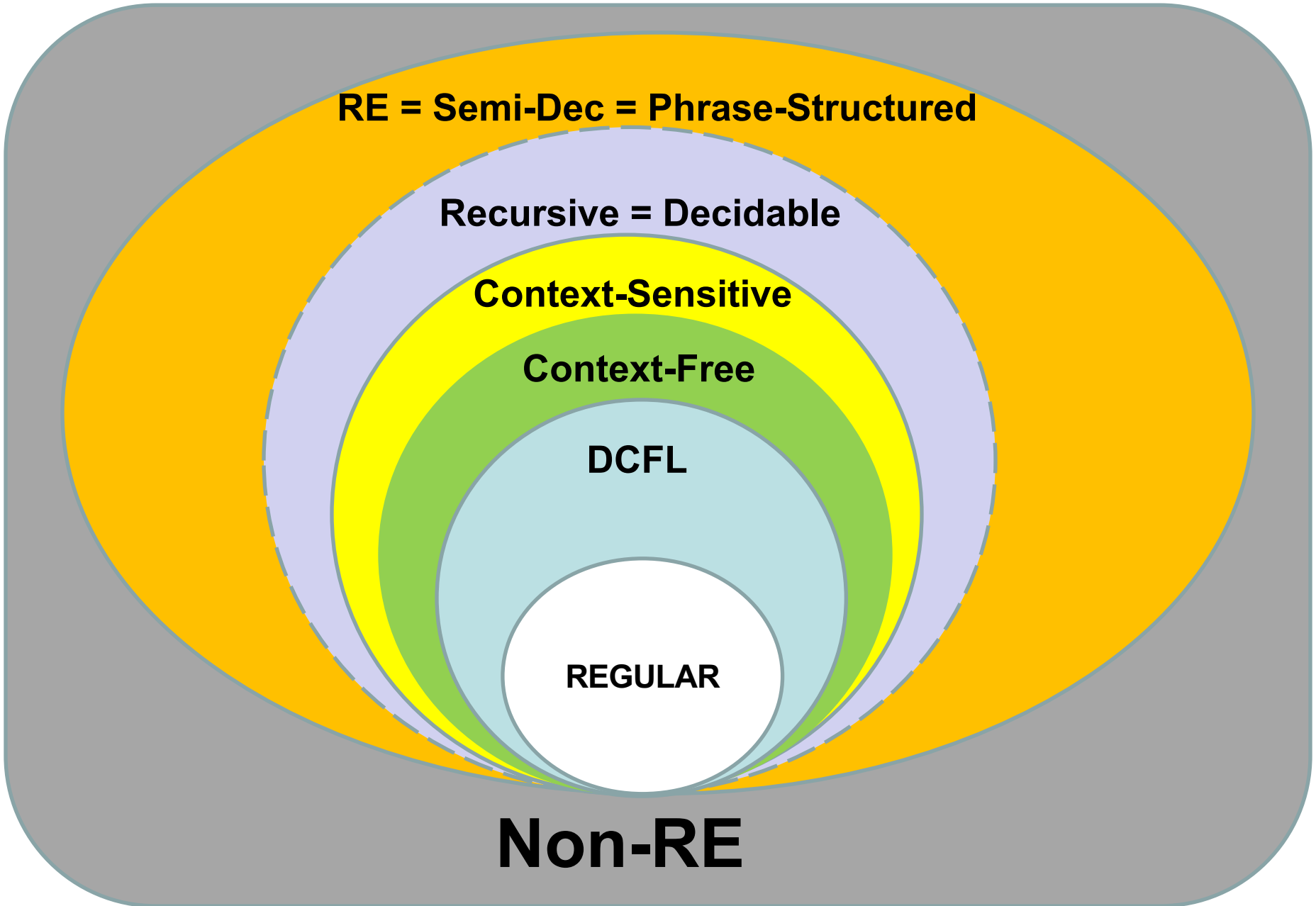
Question: How many subsets of Natural Numbers are there? How many languages are there over some finite alphabet?

Example 1: S is set of Primes and x is a natural number; is x in S (is x a prime)?
Example 2: S is an undirected graph (pairs of neighbors); is S 3-colorable?
Example 3: S is a program in C; is S syntactically correct?
Example 4: S is program in C; does S halt on all input?
Example 5: S is a set of strings; is the language S Regular, Context-Free, … ?

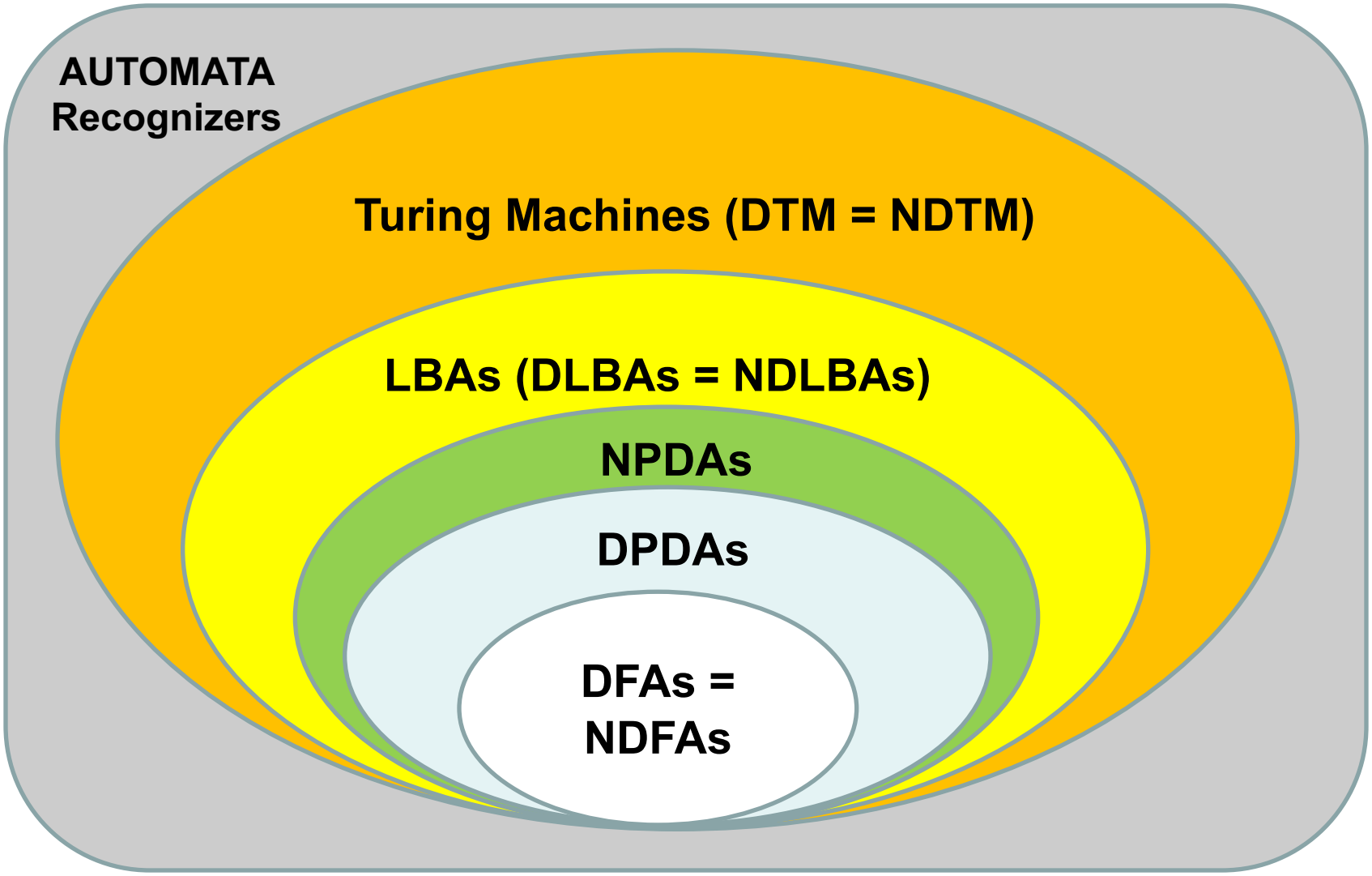
Recognizer and Generators

1. When we discuss languages and classes of languages, we discuss recognizers and generators
2. A recognizer for a specific language is a program or computational model that differentiates members from non-members of the given language
3. A portion of the job of a compiler is to check to see if an input is a legitimate member of some specific programming language – we refer to this as a syntactic recognizer
4. A generator for a specific language is a program that generates all and only members of the given language, (usually based on a grammar)
5. In general, it is not individual languages that interest us, but rather classes of languages that are definable by some specific class of recognizers or generators
6. One type of recognizer is called an automata and there are multiple classes of automata
7. One type of generator is called a grammar and there are multiple classes of grammars
8. Our first journey will be a review of automata and grammars

UNIVERSE OF LANGUAGES

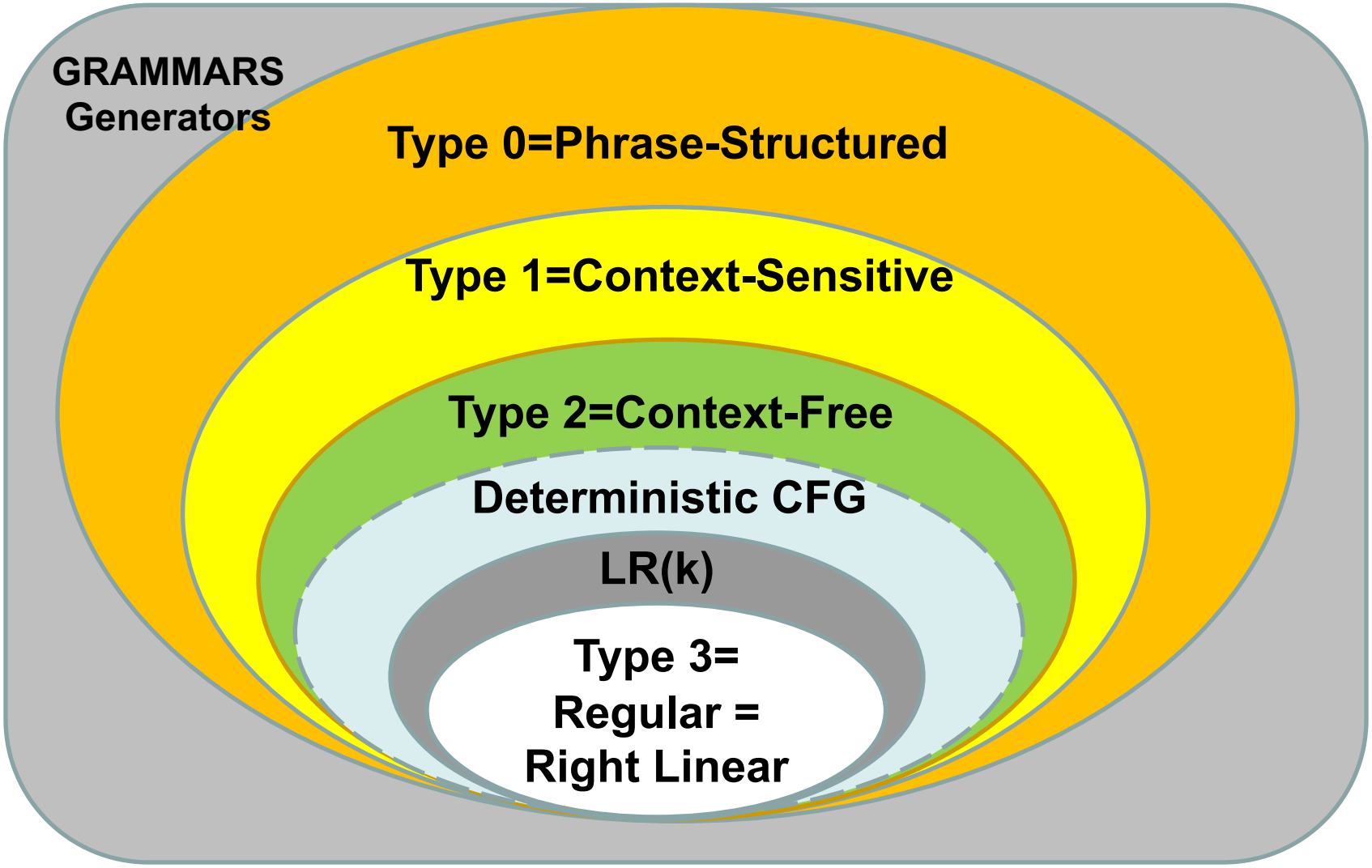


MODELS OF COMPUTATION



Of these models, only TMs can do general computation

REWRITING SYSTEMS



What We are Studying

Computability Theory

The study of what can/cannot be done via purely computational means.

Complexity Theory

The study of what can/cannot be done well via purely computational means.

Three Classes of Problems

Problems are often classified in one of three groups (classes):

Undecidable (impossible), Exponential (hard), and Polynomial (easy).

Theoretically, all problems belong to exactly one of these three classes and our job is often to find which one.

Why do we Care?

When given a new problem to solve (design an algorithm for), if it's undecidable, or even exponential, you will waste a lot of time trying to write a polynomial solution for it!!

If the problem really is polynomial, it will be worthwhile spending some time and effort to find a polynomial solution and, better yet, the lowest degree polynomial solution.

You should know something about how hard a problem is before you try to solve it.

Effective Procedure

- *A process whose execution is clearly specified to the smallest detail*
- Such procedures have, among other properties, the following:
  - Processes must be finitely describable, and the language used to describe them must be over a finite alphabet.
  - The current state of the machine model must be finitely presentable.
  - Given the current state, the choice of actions (steps) to move to the next state must be easily determinable from the procedure's description.
  - Each action (step) of the process must be capable of being carried out in a finite amount of time.
  - The semantics associated with each step must be clear and unambiguous.

Algorithm

- *An effective procedure that halts on all input*
- The key term here is "*halts on all input*"
- By contrast, an effective procedure may halt on all, none or some of its input.
- The *domain* of an algorithm is its entire universe of possible inputs
- The *domain* of a procedure is the inputs on which it converges (stops).

Sample Algorithm/Procedure

{ Example algorithm:

Linear search of a finite list for a key;

If key is found, answer “Yes”;

If key is not found, answer “No”; }

{ Example procedure:

Linear search of a finite list for a key;

If key is found, answer “Yes”;

If key is not found, try this strategy again; }

Note: Latter is not unreasonable if the list can be increased in size by some properly synchronized concurrent thread.

Procedure vs Algorithm

Looking back at our approaches to “find a key in a finite list,” we see that the algorithm always halts and always reports the correct answer. In contrast, the procedure does not halt in some cases, but never lies.

What this illustrates is the essential distinction between an algorithm and a procedure – algorithms always halt in some finite number of steps, whereas procedures may run on forever for certain inputs. A particularly silly procedure that never lies is a program that never halts for any input.

Notion of Solvable

- A problem is *solvable* if there exists an algorithm that solves it (provides the correct answer for each instance).
- The fact that a problem is solvable or, equivalently, *decidable* or, equivalently, *recursive* does not mean it is *solved*. To be solved, someone must have produced a correct algorithm.
- The distinction between solvable and solved is subtle. Solvable is an innate property – an unsolvable problem can never become solved, but a solvable one may or may not be solved in an individual's lifetime.

An Old Solvable Problem

Does there exist a set of positive whole numbers, a, b, c and an n>2 such that $a^n+b^n = c^n$?

In 1637, the French mathematician, Pierre de Fermat, claimed that the answer to this question is “No”. This was called Fermat’s Last Theorem, even though he never produced a proof of its correctness.

While this problem remained *unsolved* until Fermat’s claim was verified in 1995 by Andrew Wiles, the problem was always *solvable*, since it had just one question, so the solution was either “Yes” or “No”, and an algorithm *exists* for each of these candidate solutions.

Research Territory

Decidable – vs – Undecidable

(area of Computability Theory)

Exponential – vs – polynomial

(area of Computational Complexity)

For "easy" problems, we want to determine lower and upper bounds on complexity and develop best Algorithms

(area of Algorithm Design/Analysis)

Computability vs Complexity

Computability focuses on the distinction between solvable and unsolvable problems, providing tools that may be used to identify unsolvable problems – ones that can never be solved by mechanical (computational) means. Interestingly, unsolvable problems are everywhere as you will see.

In contrast, complexity theory focuses on how hard it is to solve problems that are known to be solvable. Hard solvable problems abound in the real world. We will address computability theory for the first part of this course, returning to complexity theory later in the semester.