Practical Applications of Theory of Computation

Compiled by Vicente Montoya and Jefferson Chien for CSE 105 Spring 2020

In this class we have explored some of the many computational models that exist and classified them in terms of their computational complexity, from **deterministic and nondeterministic finite automata,** the most simple models of computation, to the most powerful model, the **Turing Machine.** While they are very helpful in explaining the fundamental capabilities and limitations of computers, it seems easy to overlook the importance and uses of these computational models in practical applications and modern technologies. These are some of the many applications of Computational Theory:

Search Patterns

In practice, **regular expressions** are strings that allow you to describe **search patterns** that help match, locate, and manage text in general. They are used in many **programming languages**, such as Perl, Awk, Python and Java, as well as many **text editors** such as vi and emacs. They are also the basis of the (*really useful)* Unix command line utility program **G***lobally Search a* ***R****egular* ***E****xpression and* ***P****rint*, otherwise known as **grep**.

The practical regular expressions are often called **regex.** Regex was developed under the influence of theoretical regular expressions. However some **practical regular expressions** later introduced more complex syntax rules, and thus have **different expressive capabilities** from the **theoretical ones**. Many regex in use now can express a larger family of languages than the regular language. For example, Perl regex **can** express the languages $L_1 = \{a^n b a^n \mid n \geq 0\}$, as well as $L_2 = \{ww \mid w \in \{a, b\}^*\}$. However, Perl regex **cannot** express the language $L_3 = \{a^n b^n \mid n \geq 0\}$, all of which we know to be **context-free languages.**

Read more about the origin of Regex and small introduction to its syntax in this online tutorial Get Started with Regex: Regular Expressions Make Easy

Read more about the formal classification of Regex: A Formal Study of Practical Regular Expressions (fun fact: Prof. Minnes took a class from one of the authors of this paper in college.)

Artificial and Life Simulation

One of the main goals of the field of **artificial life** is to determine how complex systems, such as life forms can emerge in an entropic universe. Researchers in this field use mathematical models in order to try to simulate biological and chemical systems to achieve this goal.

One of the most commonly used models is **cellular automata**, a mathematical model proven to be **Turing complete,** meaning it is computationally universal, or capable of simulating a Turing machine. The popular Conway's Game of Life, is a two-dimensional cellular automaton, and it is known to be related to an **undecidable** problem. (In the news: John Conway recently passed away from complications from COVID-19.)

This blog post has more about **Cellular Automata**.

Compilers

Lexical analysis is the first phase of a compiler. It consists of taking the source code in the form of sentences and breaking them down into individual characters and then assembling them into a series of **tokens**. The set of possible character sequences of a token, is called the **lexical syntax** and it is usually a **regular language**.

Context-free grammars (CFGs) also constitute an essential role in compilers. They are mainly used in the **Parsing** phase of the compiling process. The goal of this phase is to take the tokens generated by the lexer, and analyze whether there exists certain patterns between them, to then associate those patterns with expressions like calling functions, recalling variables, etc. One of the most commonly used methods is ***top-down parsing*** that tries to find **leftmost derivations** of the stream of tokens by searching for parse trees using a top-down expansion of the given formal grammar rules.

Read this blog post for more about how compiler works: Understanding Compilers — For Humans (Version 2)

Linguistics and Natural Language Processing

Natural Language Processing is a subfield of linguistics and computer science that studies, among other things, how to program computers to process and analyze natural language data.

The main techniques of how human language is analyzed by computers, which include the process of **parsing**, are rooted in **Context-Free-Grammars (CFGs),** an abstract, mathematical theory of language developed by **Noam Chomsky.** He also proposed a general hierarchy to describe the classes of formal language, which is the basis of the modern classification of formal language.

Some of the applications of Natural Language Processing include: **machine translation, speech recognition and spell checking.**

In the case of **speech recognition**, there are models that use a **finite-state transducer**, a finite state machine that has **memory tapes** like a **Turing Machine**. (Our CSE 105 textbook introduces finite-state transducers in the exercises for Chapter 1.) These have an **input tape** and an **output tape**, which they can use to relate strings in a set. While a finite-state automaton accepts strings and defines a formal language with this set of accepting strings, a finite state transducer relates strings in different sets making it extremely useful in translation. If you put in strings formed by a set of alphabets, a finite state transducer can output a string of **morphemes**, the fundamental unit in a language in linguistics.

Here's a research paper presenting one use of transducers in a speech recognition model: Transformer Transducer: A streamable Speech Recognition model with Transformer Encoders and RNN-T LOSS

Read more about Chomsky's Theory and Hierarchy: Natural Language Processing (NLP): Chomsky's Theories of Syntax, or watch an excellent video on it.

This blog post lists more applications of Natural Language Processing: 10 Applications of Artificial Neural Networks in Natural Language Processing

Security and Intractability

One of the most widely used cryptosystems for secure data transmission across the internet is **RSA**. On very broad terms, it is based on the principle that it is "easy" to multiply large numbers, but factoring large numbers is "very difficult". In complexity theory, the class that constitutes these "easy problems" is called **P,** and the class of the "harder" problems is called **NP.**

Formally, the **P class** is constituted by problems that are **solved in a polynomial time**, such that the variable that defines the time that will take the algorithm is size of the input data. The **NP class** is composed of those problems in which the solution can be **verified in a polynomial time**.

We currently have algorithms that can factorize numbers but these take **exponential time to solve** as the input number gets larger**.** Yet **verification** can be done in a polynomial time. Finding the factors of a number is a problem in the NP class, but could it be in the P class? Or is it impossible? **We actually don't know!** This is the big question of **P vs. NP**. If they were, we would have a big problem. Broadly speaking, if this were true, world security would definitely be compromised because many encryption algorithms, including **RSA**, would be able to be solved in reasonably small times.

This blog post describes RSA and the importance of intractability in security here: RSA: How Maths Will Protect Us While P!=NP

Watch this video to get a glance at complexity theorem:
https://www.youtube.com/watch?v=YX40hbAHx3s

Hardware Controllers

One of the many derivations of finite automata are **Mealy and Moore machines**. However, rather than simply accepting or rejecting input strings, **Mealy and Moore machines generate outputs**. In a Mealy machine, outputs are determined by both the current input and the current state, whereas in a Moore machine, outputs are determined by only the current state. As a result, we can construct a Mealy machine, which is equivalent to another Moore machine, using fewer states making it respond faster to inputs. This would be important in a hardware controller as many behaviors are determined by the "clock". Mealy and Moore machines are commonly used in a controller written for hardware designs and are studied in CSE 140.

These George Mason University slides give an overview of how hardware designers use Mealy and Moore machines:
https://ece.gmu.edu/coursewebpages/ECE/ECE545/F10/viewgraphs/ECE545_lecture12_Controller_6.pdf

This blog post describes a digital clock design with Verilog(a hardware description language):
http://eceprojectsbtechstds.blogspot.com/2018/11/digital-clock-using-verilog.html

Computer Graphics: L-System Procedural Plant Generation

Context-Free Grammars take an essential role in procedural plant generation for computer-generated graphics. A way to generate random plants procedurally is using a method called **L-System**. It uses the **recursive nature in plants** to grow a single starting point into a tree, leaves, etc. under certain constraints. In fact, L-System can be used to generate a lot of recursive shapes.

Watch this video to learn more about generating plants with L-System

This research paper presents an application of 3D plant modeling with L-System: Real-time 3D Plant Structure Modeling by L-System with Actual Measurement Parameters

The Importance of the Halting Problem

The **halting problem** is the **problem** of determining whether a given program on a given input will finish running (**halt**), or run forever (**loop**). **Alan Turing** proved this problem to be unsolvable, or more accurately **undecidable.** This means that there does not exist a machine that can *always* tell you (halt), if another machine will halt or not on a given input.

This claim, although true, is very bold. Turing has shown a fundamental limitation on what computers can possibly do, ever. Be it now, or a hundred years from now, there will never be any computer program that can solve the **Halting Problem**.

Read about the halting problem and the limits of computers (*and the coolest poem about computer science*): The Questions That Computers Can Never Answer

Sources

1. Penland, Jon. "Get Started with Regex: Regular Expressions Make Easy." *WhoIsHostingThis.com*, 12 Feb. 2019, www.whoishostingthis.com/resources/regex/.
2. Campeanu, Cezar, et al. " A Formal Study of Practical Regular Expressions." *International Journal of Foundations of Computer Science*, http://137.149.157.5/Articles/index.php?aid=1.

3. Kozliner, Evan. “Algorithmic Beauty: An Introduction to Cellular Automata.” *Medium*, Towards Data Science, 7 Feb. 2020, towardsdatascience.com/algorithmic-beauty-an-introduction-to-cellular-automata-f53179b3cf8f.
4. Young, Michael J. “ Typical Uses of Cellular Automata.” *Mjyonline*, 12 Nov. 2006, www.mjyonline.com/CellularAutomataUses.htm.
5. Zhang, Qian, et al. “TRANSFORMER TRANSDUCER: A STREAMABLE SPEECH RECOGNITION MODEL WITH TRANSFORMER ENCODERS AND RNN-T LOSS.” *ArXiv*, 14 Feb. 2020, https://arxiv.org/pdf/2002.02562.pdf.
6. Ozkural, Eray. “Natural Language Processing (NLP): Chomsky's Theories of Syntax.” *Medium*, Medium, 27 Dec. 2017, medium.com/@ehfirst/natural-language-processing-nlp-chomskys-theories-of-syntax-92fb8fa3d035.
7. Davydova, Olga. “10 Applications of Artificial Neural Networks in Natural Language Processing.” *Medium*, Medium, 26 Sept. 2017, medium.com/@datamonsters/artificial-neural-networks-in-natural-language-processing-bcf62aa9151a.
8. Lopez, Juan. “RSA: How Maths Will Protect Us While P!=NP.” *Medium*, Datio, 2 Apr. 2018, medium.com/datio-big-data/rsa-how-maths-will-protect-us-while-p-np-1b29ca6bff82.
9. PROJECTS, ECE. “DIGITAL CLOCK USING VERILOG.” *DIGITAL CLOCK USING VERILOG*, Blogger, 16 Nov. 2018, eceprojectsbtechstds.blogspot.com/2018/11/digital-clock-using-verilog.html.
10. Viruchpintu, Rawin, and Noppadon Khiripet. “Real-Time 3D Plant Structure Modeling by L-System with Actual Measurement Parameters.” *Bioquest*, https://www.bioquest.org/products/files/13157_Real-time 3D Plant Structure Modeling by L-System.pdf.

11. Bhatia, Aatish. “The Questions That Computers Can Never Answer.” *Wired*, Conde Nast, 3 June 2017, www.wired.com/2014/02/halting-problem/.