CS 3313:
Foundations of Computing

Spring 2021

Instructor: Prof. Bhagi Narahari

TAs: Siyuan (Andy) Feng and Linsheng Liu

UTA: Grant McClearn

LAs: Marshall Thompson, Oliver Broadrick

http://gw-cs3313-2021.github.io

CS 3313: What is it about ?

- Theoretical foundations of Computer Science
  - It's also a look at history of CS…Concept of computing existed before the first computer was built…
- Two aspects to CS:
  - 1. Fundamental ideas & models underlying computing
  - 2. Engineering techniques HW& SW for design of computing systems
- Theoretical CS had its beginnings in diverse fields
  - Electrical Eng.: studying switching circuits
  - Mathematics: studying foundations of logic
  - Linguists: investigating grammars for natural languages
  - Out of these studies came models that are now central to CS!!

Questions that drove the Theory of Computing field

- Answer/Ask fundamental (abstract) questions:
  - What is computation –i.e., how do you define what an algorithm is?
  - mathematical models for different types of computing machines ?
    - Why is this an interesting question ?
  - How do you formally define a language
    - Natural language or Programming language

Course Learning Objectives

- Understand and design different automata (machine) models
  - Apply mathematical reasoning to assert properties of the machines
- Formal models for defining languages – Grammars
- Relationship between languages and automata/machines
- Study fundamental limits of computing (and each machine model), develop ability to determine the computational complexity or solvability of problems
- Further develop the ability to abstract computational problems and mathematical perspective to problem solving
- Above all: *course is about problem solving using Math tools*
  - Mathematical puzzles which abstract “real” computational questions

Course Schedule - Topics

- Part 1: Regular Languages and Finite State Automata.

  (Weeks 1-5). familiar territory but now from a math perspective

  - Finite Automata ….same as Finite State Machines in Hardware!
  - Class of regular languages
  - Properties of regular languages
- Part 2: Context Free Languages and Grammars (weeks 5-10)
  - Formal Grammars – context free grammars, parsing
  - Pushdown Automata – adding simple ”memory” to finite state machines
- Part 3: Turing Machines and Computability
  - Turing machine model and Universal Turing machine
  - What is computable ? Proving a problem is not solvable
  - Computational complexity models.

Instruction team

- Bhagi Narahari
- Grad TAs: Andy Feng and Linsheng Liu

  - Undergraduate TAs and LAs:
    - Grant McClearn-UTA (Senior, BS-CS)
    - Oliver Broadrick (Junior, BS-CS)
    - Marshall Thompson (Junior, BS-CS)
  - All grading inquiries on homeworks directed to Andy or Linsheng (and then follow up with instructor) via email
    - No posting to piazza
  - All discussions on labs and lectures, post to Piazza

In-class work

- You will learn through in-class activities and exercises most classes (lecture+lab)
  - Must read the material and come to class
- If you are assigned an exercise during class (i.e., an "in class exercise"), you need to complete the exercises by the end of the class/day !
  - We may ask a group/student to present solutions to class

Course Materials – "confusion will be my epitaph"!

- Course webpage – will have links to syllabus, lecture notes, online resources (and videos when applicable)
  - http://gw-cs3313-2021.github.io

- Blackboard will be used for:
  - Synchronous online lectures & labs
  - Homeworks and solutions
  - Reporting grades
- Piazza – for discussions: you've used this before…

Piazza

- you've used this before, so you know the protocols:
  - The purpose of this:
    - to encourage you to ask and answer questions
      - Most of the time, you do better than we do!
    - *Be very careful not to border on plagiarism!*
    - *Don't post your HW solution to the world,*
  - Signup instructions posted on Blackboard
  - Do not expect instant response or substitute Piazza for office hours!
    - Piazza is not manned 24 hours/7 days a week
    - Sometimes an answer may take more than 24 hours!
  - Posting on piazza, not the same as telling instructor things
    - E.g. : I'm going to miss the exam!
  - Do **NOT** wait until the last minute to ask for clairifications…
    - The instructors & TAs do NOT plan on spending their weekend checking Piazza!

Textbooks/Software

- Textbook:
  - Introduction to Formal Languages and Automata, 6th edition by Peter Linz (earlier editions will work too), JB Learning.
  - Alternate textbook (a very good book a bit denser): Introduction to the Theory of Computation by Michael Sipser, CEngage publishers.
  - Online notes and resources
- JFLAP – simulator for automata
  - You can install it locally on your laptop
  - Check the tutorial video on the course webpage

Course Requirements: Grading

- Exams: 50%
  - 3 exams
  - Approximately weeks 5, 10, and Finals week
- Homeworks: 25%
- Quiz and inclass (lab) exercises: 25%
- Grades curved (and scaled as percentage of highest score in class)
  - Check syllabus on website for details.

Three key concepts

- Languages
  - Set of sentences (strings/words) formed from some alphabet
  - How do we specify the property?
- Grammars
  - A formalism for mathematically defining the properties of a language
  - A set of rules for generating the sentences in a formal language
- Automata
  - Mathematical model of machines (of different capabilities)
  - Formal construct that accepts input, produces output and may have temporary storage and can make decisions

Formal Languages: Basic Concepts

- *Alphabet*: set of symbols, i.e. $\Sigma = \{a, b\}$
- *String*: finite sequence of symbols from Σ, such as $v = aba$ and $w = abaaa$
  - Empty string (λ)
  - Substring, prefix, suffix
- *Operations on strings*:
  - *Concatenation*: $vw = abaabaaa$
  - *Reverse*: $w^R = aaaba$
  - *Repetition*: $v^2 = abaaba$ and $v^0 = \lambda$
- *Length of a string:* $|v| = 3$ and $|\lambda| = 0$

Formal Languages: Definitions

- Σ^* = set of all strings formed by concatenating zero or more symbols in Σ
- Σ^+ = set of all non-empty strings formed by concatenating symbols in Σ

In other words, $\Sigma^+ = \Sigma^* - \{ \lambda \}$

- A *formal language* is any subset of Σ^*

Examples: $L_1 = \{ a^n b^n : n \geq 0 \}$ and $L_2 = \{ ab, aa \}$

- A string in a language is also called a *sentence* of the language

Formal Languages: Set Operations

- A language is a set. Therefore, set operations are defined as usual.
- If $L_1 = \{ a^n b^n \mid n \geq 0 \}$ and $L_2 = \{ ab, aa \}$
  - Union: $L_1 \cup L_2 = \{ aa, \lambda, ab, aabb, aaabbb, \ldots \}$
  - Intersection: $L_1 \cap L_2 = \{ ab \}$
  - Difference: $L_1 - L_2 = \{ \lambda, aabb, aaabbb, \ldots \}$
  - Complement: $L_2 = \Sigma^* - \{ ab, aa \}$
- Question: Find $L_2 - L_1$

Formal Languages: Other Operations

- New languages can be produced by reversing all strings in a language, concatenating strings from two languages, and concatenating strings from the same language.
- If $L_1 = \{ a^n b^n \mid n \geq 0 \}$ and $L_2 = \{ ab, aa \}$
  - Reverse: $L_2^R = \{ ba, aa \}$
  - Concatenation: $L_1 L_2 = \{ ab, aa, abab, abaa, aabbab, aabbaa, \ldots \}$

    The concatenation $L_2 L_2$ or $L_2^2 = \{ abab, abaa, aaab, aaaa \}$
  - Star-Closure: $L_2^* = L_2^0 \cup L_2^1 \cup L_2^2 \cup L_2^3 \cup \ldots$
  - Positive Closure: $L_2^+ = L_2^1 \cup L_2^2 \cup L_2^3 \cup \ldots$
- Question: Find $(L_2 - L_1)^R$

A better formalism to define language ?
Some questions

- Set notation works but does not specify a way to generate the words/strings in the language
- Ex: how do you define a syntactically valid C program ?
- Ex: how do you define the construction of a sentence in the English language ?

More questions….

- Will the piece of C code compile ?
  - How to capture this property

```
/* header info.. */
int foo(int x, int y, char a){
      // body of function
      }

int main{
      int i,j,k;
      k= foo(i,j);
      ….
}
```

More questions….

- What does this (English) sentence mean:

“Grant made Oliver duck”

- What does this (English) sentence mean:

“Time flies like an arrow.”

Need for formalism and Math rigor

- We would like to capture precisely, and logically, the properties (problems) in the previous examples
- Ex1: actual and formal parameters (arguments) should match
- Ex 2,3: the language is ambiguous…which is a bad thing in a programming language
  - How do we define, using a mathematical structure, what ambiguity means (in an unambiguous manner ☺)

Grammars: Definition

- Precise mechanism to describe the strings in a language
- Definition: A grammar G consists of:

  V: a finite set of *variable* or non-terminal symbols

  T: a finite set of *terminal* symbols (the *alphabet*!)

  S: a variable called the *start* symbol

  P: a set of *productions* (also called production rules)
- Example 1:

  $V = \{ S \}$

  $T = \{ a, b \}$

  $P = \{ S \rightarrow aSb, S \rightarrow \lambda \}$

Grammars: Derivation of Strings

- Beginning with the start symbol, strings are derived by repeatedly replacing variable symbols with the expression on the right-hand side of any applicable production
- Any applicable production can be used, in arbitrary order, until the string contains no variable symbols.
- Sample derivation using grammar in Example 1:

$S \Rightarrow aSb$ (applying first production)

$\Rightarrow aaSbb$ (applying first production)

$\Rightarrow aabb$ (applying second production)

The Language Generated by a Grammar

- Definition 2: For a given grammar G, the language generated by G, L(G), is the set of all strings derived from the start symbol
- To show a language L is generated by G:
  - Show every string in L can be generated by G
  - Show every string generated by L is in G

Grammars for Programming Languages

- The syntax of constructs in a programming language is commonly described with grammars
  - Commonly referred to as Backus-Naur Form (BNF)
- Assume that in a hypothetical programming language,
  - Identifiers consist of digits and the letters a, b, or c
  - Identifiers must begin with a letter
- Productions for a sample grammar:

<id> $\rightarrow$ <letter> <rest>

<rest> $\rightarrow$ <letter> <rest> | <digit> <rest> | λ

<letter> $\rightarrow$ a | b | c

<digit> $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Grammars for English

- Possible rules for a formal English grammar:

  <sentence> → <noun phrase> <verb phrase>

  <verb phrase> → <adverb> <verb> | <verb>

  <noun phrase> → <pronoun> <noun> | <noun>

- In Ex1: <verb> → duck or <noun> → duck
- In Ex.2: <verb>. → Flies or <noun> → flies

Concept 3 – modeling computational machines: Automata

- Some “real” machine models:
  - Finite state machines – i.e. sequential circuits
  - von Neumann model of computer architectures
- So why study formal models ?
  - To investigate the computational power of these machines
    what problems can they solve ?

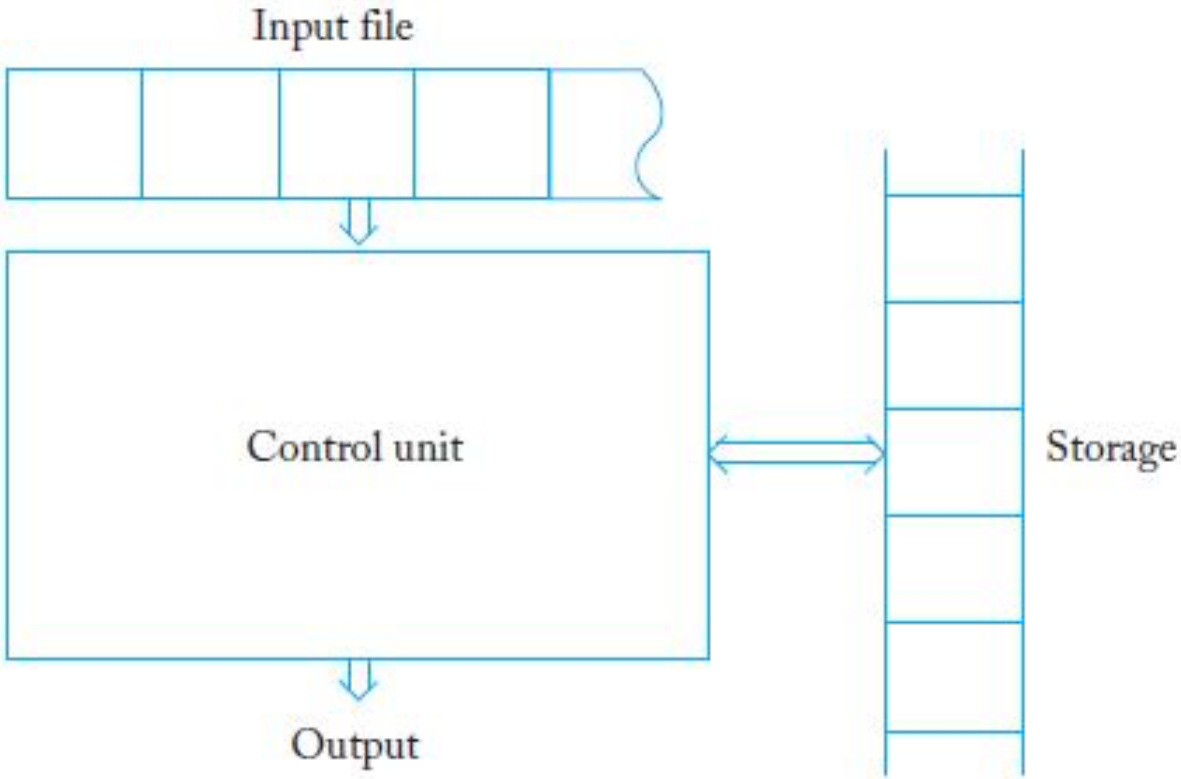
- Question: Can we build a compiler using just a finite state machine ?

  advantages: simpler design
- Bigger question: what problems are solvable by a von Neumann computer ?

Automata

- An *automaton* is an abstract model of a digital computing device
- An automaton consists of
  - An input mechanism
  - A control unit
  - Possibly, a storage mechanism
  - Possibly, an output mechanism
- Control unit can be in any number of internal *states*, as determined by a *next-state* or *transition* function.
- There are a finite number of states
  - *Why ?*

Illustration of a General Automaton



Note: this is one model of the “type” of external storage
The storage model changes based on the automata model
Question: Is there storage in a finite state machine ?

Automata we will study

- Finite Automata (Deterministic & Non-deterministic)
  - These model Finite State Machines
- Pushdown automata
  - Add the simplest form of memory to a Finite state machine
- Turing Machines
  - These model today's computers in terms of computational ability
- Questions to ask: What is the ”power” (limit) of each of these machine models ?
  - What types of problems can be solved by a finite automaton, Turing machine etc.
- Link b/w Languages and Automata: view automata as “acceptors” of input strings
  - What languages are accepted by the automata model

Limits of Computation (of automata): Questions

- Can we use a finite state machine to build a compiler ?

- Can we use a pushdown automaton to parse a programming language ?

- Can we design a compiler that will determine for any program, whether the program halts on all inputs
  - Or, will the compiler detect any bugs in the program ?
  - Or, are two programs equivalent ? (do they compute the same function)
- To answer these questions, via proofs, we need mathematical models for these types of machines!

Mathematical Rigor

- This is a theoretical course and requires formalisms and formal (math) methods……
  - Unless we ask for a JFLAP simulation, your answers need to be grounded in Maths (i.e., no multi-page discussions with no mathematical logic!)

Next…

- We start with Finite Automata
- These are the ‘same’ as Finite State machines covered in Computer Architecture
  - But we model them as acceptors of languages:

  Input is a string and the Automata determines if the string is in the language

Chapter 2 of Textbook