CS 3313:
Foundations of Computing

Spring 2022

Instructor: Prof. Bhagi Narahari

Lead TA: Siyuan (Andy) Feng

UTAs (BS CS, Class of 2022):

Linnea Dierksheide,

Marshall Thompson,

Oliver Broadrick

LA: Kyle Vitale (Class of 2023)

http://gw-cs3313.github.io

1

CS 3313: What is it about ?

- Theoretical foundations of Computer Science
  - history of CS…Concept of computing existed before the first computer
- 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
- Study fundamental limits of computing (& machines)
- Above all: about problem solving and algorithmic thinking
  - Design algorithms to solve problems for different machine models
  - Problems = Mathematical puzzles which abstract "real" computational questions

2

Why bother ?
Course Learning Objectives

- **Purpose**: Abstraction, innovative and algorithmic thinking
  - learning foundations can help us solve novel problems efficiently

Learning Objectives:

- Understand and design different automata (machine) models
- Formal models for defining languages & compilers – Grammars
- Understand foundations behind "solvable" and "unsolvable" problems
  - How to determine if a problem is solvable on a computer ?
- Develop problem solving and algorithmic thinking skills
  - Design "algorithms" to work on different machine models

3

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!
  - Regular expressions to denote regular languages (same as Unix RegEx)
  - Properties of regular languages
- Part 2: Context Free Languages and Grammars (weeks 5-10)
  - Pushdown Automata – adding simple "memory" to finite state machines
  - Formal grammars – context free grammars and a parsing algorithm
- 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.

4

In-class work

- You will learn through in-class activities and exercises most classes (lecture+lab)
  - read the material and come to class/lab
- 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 when we return to in-person classes

5

Course Materials – "confusion will be my epitaph"!

- Course webpage – will have links to syllabus, lecture notes, online resources (and tutorial videos when applicable)
  - http://gw-cs3313.github.io
- Blackboard will be used for:
  - Synchronous online lectures & labs (including recordings)
  - Homeworks and solutions
  - Reporting grades
- Piazza – for discussions: you've used this before…

6

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!

7

Textbooks/Software

- Textbook:
  - Introduction to Formal Languages and Automata, 6^{th} 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*

8

Course Requirements: Grading

- Exams: 50-55%
  - 3 exams
  - Approximately weeks 5, 10, and Finals week
- Homeworks: 25%
- Quiz and inclass (lab) exercises: 20-25%
- Grades curved (and scaled as percentage of highest score in class)
  - Check syllabus on website for details.

9

Next….let's get started with the course

- Questions?

10

Three key concepts

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

11

1. Formal Languages: Basic Concepts

- *Alphabet* Σ : set of symbols
  - Ex: $\Sigma = \{a, b\}$ $\Sigma = \{0,1\}$
- *String*: finite sequence of symbols from Σ,
  - ex: $v = aba$ and $w = abaaa$
  - ex: $v = 001$ and $w = 11001$
  - 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 $|\lambda| = 0$

12

Formal Languages: Definitions

- Σ^* = set of all strings formed by concatenating zero or more symbols in Σ
  - Ex: if $\Sigma = \{0,1\}$ *then* $\Sigma^* = \{$*all binary strings, including empty string*$\}$
- 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
  - Ex: If $\Sigma = \{a,b,c,\ldots z\}$ *then ???*

13

Formal Languages: Operations

- A language is a set, therefore set operations on languages:
- 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: $L_2 = \Sigma^* - \{ ab, aa \}$
- 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: $L_2^+ = L_2^1 \cup L_2^2 \cup L_2^3 \cup \ldots$
- More review in labs this week….

14

A better formalism to define languages ? Some questions

- Set notation works but does not specify a way to generate the words/strings in the language such that they satisfy certain properties
  - Strings are in the language if they satisfy a set of properties
- Ex: how do you define a syntactically valid C program ?
- Ex: how do you define the construction of a sentence in the English language ?

15

Why do we need formal methods…some questions….

- Will this piece of C code compile ?

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

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

How do we specify this property ?

16

More questions….

- What does this (English) sentence mean:
  "Oliver made Linnea duck"

- What does this (English) sentence mean:
  "Time flies like an arrow."

17

So what is this telling us…..
Need for formalism and Math rigor

Grammars

- Need a formalism/mechanism to define properties of languages
- 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 model) what ambiguity means (in an unambiguous manner ☺)

18

2. Grammars: Definition

- Formal tool for describing and analyzing languages
  - Set of rules by which valid sentences/strings in a language are constructed
- 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:

$$V = \{ S \}$$
$$T = \{ a, b \}$$
$$P = \{ S \rightarrow aSb,\ S \rightarrow \lambda \}$$

19

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 byG is in L

20

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:

$$
\begin{aligned}
S &\Rightarrow aSb && \text{(applying first production)}\\
&\Rightarrow aaSbb && \text{(applying first production)}\\
&\Rightarrow aabb && \text{(applying second production)}
\end{aligned}
$$

21

Example: English Grammar

- A subset of the complete English grammar:

<sentence> → <subject> <verb phrase> <object>
<verb phrase> → <adverb> <verb> | <verb>
<object> → *the* <noun> | a <noun> | <noun>
<subject> → *This* | *Computers* | *I*
<adverb> → *never*
<verb> → *run* | *tell* | *am*
<noun> → *university* | *world* | *lies* | *cheese*

Using above rules, we can derive sentences such as:

Computers run the world

object

Verb phrase

subject

I never tell lies
Computers run cheese
This am a lies

22

Grammars for Programming Languages

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

```
<identifier> → <letter> <rest>
<rest> → <letter> <rest> | <digit> <rest> | λ
<letter> → a | b | c |…|z
<digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

23

Snippet of syntax of your favorite prog. language

```
<statement> ::= <labeled-statement> | <expression-statement> |
                <compound-statement> | <selection-statement> |
                <iteration-statement> | <jump-statement>

<labeled-statement> ::= <identifier> : <statement> | case
              <constant-expression> : <statement> | default :
                                                   <statement>

<iteration-statement> ::= while ( <expression> ) <statement> |
              do <statement> while ( <expression> ) ; |
              for ( {<expression>}? ; {<expression>}? ;
                                 {<expression>}? ) <statement>
```

Parsing a program (checking for syntax errors) = determine if the program is generated by the grammar

24

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 ?

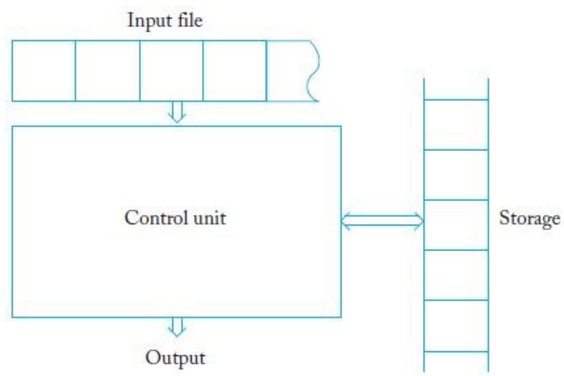
25

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 ?*

26

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 ?

27

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
- Link b/w Languages and Automata: view automata as "acceptors" of input strings
  - What languages are accepted by the automata model

28

Limits of Computation (of automata): Questions

- 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.
- Q 1: Can we use a finite state machine to build a compiler ?
- Q 2: Can we use a pushdown automaton to parse a programming language ?
- Q 3: Can we design a compiler that will determine for any program, whether the program halts on all inputs
  - Can the compiler detect all bugs in the program ?
  - 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!

29

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 logical reasoning!)

30

Proof methods….Quick Review in the Lab

- What is a proof:
  - A sequence of logical steps, each following from previous steps
  - In logic terms: a propositional (predicate) formula whose truth can be derived from a sequence of propositions (using the different rules of logical inference)
- Why: rather than depend on the gift of gab, we use proofs to assert our claims/results
- Types of proofs:
  - Direct
  - Induction
  - Contradiction
  - Contrapositive
  - Counter example
  - Constructive

31

Our Approach to studying automata models: Algorithmic Thinking

To understand how each automata (machine) model works, we take the approach of developing "algorithms" that work on that machine

32

Computational Thinking and Algorithmic Thinking – an important skill for Computer Scientists!

- "Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" – Cuny, Snyder, Wing, 2010

- Invaluable to methodically approach a complex (unknown) problem, break it down into smaller/easier problems and quickly build a robust solution
  - Not just in CS…..everyday tasks!!

33

Algorithmic Thinking

1. Understanding the problem
   - Define it "precisely"
     - What are the inputs? What are the outputs ? What constraints?
   - Can you describe the problem
2. Devising a plan
   - Identify the level of problem solving..be methodical in your steps
     - Do you apply a known solution? Do you generate a new solution?
   - Create a series of steps to solve the problem
     - Each step is precise and unambiguous
3. Program the solution…..

- In this course: You design "algorithms" to solve a problem on a specific machine model
  - Ex: machine = Finite state machine, problem = determine if input is an even integer

34

Algorithmic Thinking Learning Goals

- Use algorithmic thinking to efficiently approach and solve complex problems
  - Approach new challenges by understanding the problem and then formulating and executing a plan
  - Break down complex problems using top down approach
  - Recognize and use familiar programming patterns
  - Be efficient—work strategically, not recklessly!

35

Properties of Algorithms

Algorithm must have these properties if the "machine" is to execute it without human intervention:

- Input specified
  - Type of data expected: numbers? Strings? Letters? Alphabet?
- Output specified
  - Types of data forming the result
- Definiteness: be explicit about how to realize the computation
  - Can be achieved by giving commands that state unambiguously what to do, in sequence
    - Ex: conditional … If (input == 0) then go to step 2
- Effectiveness ensures machine can perform operation without human (or superman's !) intervention
  - Achieved by reducing task to the primitive operations of the machine
    - Ex: machine code on a computer !!
- Finiteness – must terminate

36

Next.....

- Labs this week:
  - Review of Proofs
  - Review languages/strings
  - Inclass exercises/discussions

- Introduce Finite State Automata (aka Finite State Machines)
- Your to do list:
  - Sign up for Piazza
  - Download and install JFLAP (check tutorial on course webpage)