CS-277 : Notes

Charlie Stuart : src322

Fall 2020

Note: Section numbers correspond only to course and reading order

Contents

1 Proofs

From pages 17-23 in *Introduction to the Theory of Computation* by Michael Sipser

Definition : Describes and object or notation
Proof : Convincing logical argument that a statement is true
Theorem : A mathematical statement proved true
Lemma : A theorem that assists in the proof of another theorem
Corollaries : The parts of a theorem where we can conclude that other related statements are true

1.1 Proof By Construction

To prove something exists, we prove we can construct the object.

1.2 Proof By Contradiction

In order to prove a theorem true, we can say it's false, then show that that leads to an even more false statement.

1.3 Proof By Induction

In induction, we need an ordered set of variables with consistent variance. We create a base case, prove it is true, then for the inductive case, we prove that x+1 is also true.

2 Set Theory Intro

From pages 3-7 in *Introduction to the Theory of Computation* by Michael Sipser

Set : A group of objects represented as a unit
Element : An object in a set
Member : An object in a set
Multi Set : An set containing an element that occurs multiple times
Subset : A set that consists of elements that exist in a different set
Proper Subset : A set that is a subset of another set, but not equal
Infinite Set : A set of infinitely many elements
Empty Set : A set of no elements
Singleton Set : A set of one elements
Unordered Pair : A set of two elements
Sequence : A set in a specific order
Tuple : A finite set
k-Tuple : A tuple of k elements
Ordered Pair : A 2-tuple
Power Set : All the subsets of A
$\in$: Is a member of
$\notin$: Is not a member of
$\subset$: Is a proper subset of

$\not\subset$: Is not a proper subset of
$\subseteq$: Is a subset of
$\not\subseteq$: Is not a subset of
$\cup$: Union of two sets
$\cap$: Intersection of two sets
$\times$: Cross product of two sets
$\mathbb{N}$: Set of natural numbers
$\mathbb{Z}$: Set of integers
$\mathbb{Q}$: Set of rational numbers
$\mathbb{A}$: Set of algebraic numbers
$\mathbb{R}$: Set of real numbers

A set is defined in a few ways

$S = \{7, 21, 57\}$	Finite Set
$S = \{1, 2, 3...\}$	Infinite Set of all natural numbers $\mathbb{N}$
$S = \{7, 7, 21, 57\}$	Multi Set
$S = \varnothing$	Empty Set
$S = \{5\}$	Singleton Set
$S = \{5, 3\}$	Unordered pair
$S = \{n \mid n = m^2 \text{ for some } m \in \mathbb{N}\}$	Set of perfect squares

The union of two sets is the same as an OR operator in boolean algebra. It's all the elements in both sets.

$$\begin{aligned} A &= \{1, 2, 3\} \\ B &= \{3, 4, 5\} \\ A \cup B &= \{1, 2, 3, 4, 5\} \end{aligned}$$

The intersection of two sets is the same as an AND operator in boolean algebra. It's all the elements that appear only in both sets.

$$\begin{aligned} A &= \{1, 2, 3\} \\ B &= \{3, 4, 5\} \\ A \cap B &= \{3\} \end{aligned}$$

The Cartesian product, or cross product, of two sets is the set of all ordered pairs where the first element is a member of the first set and the second element is a member of the second set for every combination.

$$\begin{aligned} A &= \{1, 2\} \\ B &= \{x, y, z\} \\ A \times B &= \{(1, x), (2, x), (1, y), (2, y), (1, z), (2, z)\} \end{aligned}$$

3 Functions

From pages 7-8 in *Introduction to the Theory of Computation* by Michael Sipser

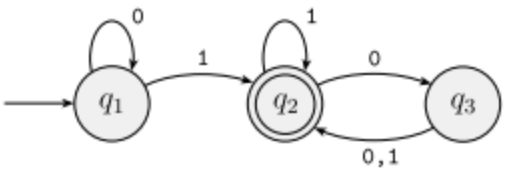
Function : An objects that sets up an input-output relationship
Domain : The set of possible inputs to a function
Range : The set of possible outputs to a function

$f : D \to R$ Function f has domain D and range R

4 Finite Automata

From pages 31-43 in *Introduction to the Theory of Computation* by Michael Sipser

A finite automaton, or finite state machine, is the simplest way to describe a computational models behavior.



In the above state machine, there are three states denoted by circles, q_1, q_2, and q_3. The transitions are denoted by arrows. The start state is denoted by the arrow coming from no where. The accept state is denoted by the bubble with a circle inside it. Upon receiving a string, it processes it then produces an "accept" or "reject" output. Only if we end in an accept state is the output "accept".

While the simple description of a finite automaton is simple enough for me to understand, there are five things a finite automaton must include. This can be described using the 5-tuple, $(Q, \Sigma, \delta, q_0, F)$.

1. **States :** A set of states (Q)
2. **Alphabet :** The allowed input symbols (Σ)
3. **Transition Function :** The rules for moving from one state to another ($\delta : Q \times \Sigma \to Q$)
4. **Start State :** Self Explanatory ($q_0 \in Q$)
5. **Set of Accept States :** Self explanatory ($F \subseteq Q$)

Looking back at the above state machine, we can describe it as the following.

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. δ can be described in the following table with states on the left, input on the top, and the state transitioned to being the intersection

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. $q_0 = q_1$
5. $F = \{q_2\}$ (in this case, a singleton set)

4.1 Language

From pages 44-45 in *Introduction to the Theory of Computation* by Michael Sipser

Unary Operator : An operation on one element

Binary Operator : An operation on two elements

Regular Language : A language that can be expressed with a regular expression

A is the set of all strings that the state machine accepts. A is the language of the machine where $L(M) = A$. M recognizes A. M accepts A.

So we have three main operations that can act on languages.

Union : $A \cup B = \{x | x \in A \text{ or } x \in B\}$

Concatenation : $A \circ B = \{xy | x \in A \text{ and } y \in B\}$

Concatenation is like a cross product but concatenating the tuples.

Star : $A^* = \{x_1 x_2 ... x_k | k \geq 0 \text{ and } x_i \in A\}$

In star, ε is always a member of the language. This one is a little wierd so see the example below

Given alphabet Σ where it's the standard English alphabet, a through z. If $A = \{\text{good}, \text{bad}\}$ and $B = \{\text{girl}, \text{boy}\}$, then

$$A \cup B = \{\text{good}, \text{bad}, \text{boy}, \text{girl}\}$$
$$A \circ B = \{\text{goodgirl}, \text{goodboy}, \text{badgirl}, \text{badboy}\}$$
$$A^* = \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badgood}, \text{badbad}, \text{goodgoodgood...}\}$$

Theorem

1.25 in *Introduction to the Theory of Computation* by Michael Sipser

The class of regular languages is closed under the union operation. In dumb bitch english, if A_1 is regular and A_2 is regular, then $A_1 \cup A_2$ is regular.

To prove this we could make two machines M_1 and M_2 that both separately accept A_1 and A_2 respectively. We then could combine them into a machine M, however the machine would not know the difference between something M_1 should accept vs M_2 should accept and could confuse inputs. Now to learn about non-determinism.

A quick note on regular expressions! This is like regex in UNIX but a lot simpler. Below is the formal definition of the operations

1. a for some a in the alphabet Σ
2. ε
3. $\varnothing$
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions
6. (R_1^*), where R_1 is a regular expression

For example, $(0 \cup 1)^*$ is a sequence of as many 0s and 1s

4.2 Non-deterministic Finite Automata

From pages 47-50 and 53 in *Introduction to the Theory of Computation* by Michael Sipser

Deterministic : Given a state and an input symbol, the next state is determined by a function and we know what it will be.

Non-Deterministic : There are many options in a given state.

Non Deterministic Finite Automata (NFA)

- Can have zero, one, or many transitions for each alphabet member
- Can include ε in addition to the alphabet
- Traverses in a tree

Deterministic Finite Automata (DFA)

- Each state has one transition for each alphabet member
- Only includes alphabet members
- Traverses sequentially
- All DFAs are NFAs

An NFA is determined with a reject/accept state in a different way than a DFA. A DFA traverses sequentially, one state after the next, an NFA traverses and creates a tree of all the possible outcomes. Some rules when traversing:

- If there is not a transition for an input, the branch stops
- If there are multiple transitions for an input, it branches for as many times with than input.
- Upon reaching ε

We also have a more formal definition of an NFA:

1. **States :** A finite set of states (Q)
2. **Alphabet :** The finite allowed input symbols (Σ)
3. **Transition Function :** The rules for moving from one state to another ($\delta : Q \times \Sigma_\varepsilon \to P(Q)$) $P(Q)$ is the power set of Q
4. **Start State :** Self Explanatory ($q_0 \in Q$)
5. **Set of Accept States :** Self explanatory ($F \subseteq Q$)

4.3 Non Regular Languages

A language that isn't regular. It cannot be constructed with a DFA or NFA. An example of one is a language with an equal number of 0's and 1's. Since there can be an infinite number of 0s and 1s, there would have to be an infinite number of states to account for infinite possibilities where there are more 0s than 1s and so on and so forth.

4.3.1 Pumping Lemma

All strings in a regular language have a special property, if the language does not have this property, it is not regular. The property is:

For a regular language A, there is a number p (the pumping length) where if s is any string in A of at least length p, then s can be divided into at least 3 pieces, where $s = xyz$, where the following conditions are met.

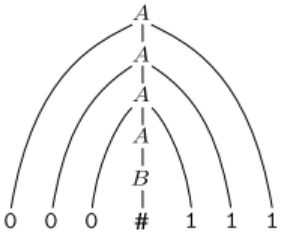
1. for each $i \geq 0, xy^iz \in A$
2. $|y| > 0$, and
3. $|xy| \leq p$

5 Context Free Grammars

A powerful way of describing languages. Some have a recursive structure which is sexy in computing. Regular languages are contained within context free grammars, but CFGs can describe languages that are not regular.

Productions : The set of substitution rules
Variable : The symbol on the left of a production
Terminal : The product of a production. A cobination of symbols and products
Start Variable : The start of a CFG

$A \to 0A1$
$A \to B$
$B \to \#$



The resulting set of strings accepted by the grammar is the language of the grammar. A language generated by a CFG is a context-free language.

The formal definition of a CFG is a 4 tuple (V, Σ, R, S) where:

1. V is a finite set of variables
2. Σ is a finite set, disjoint from V of terminals
3. R is a finite set of rules, with each rule being a variable and a string of variables and terminals
4. $S \in V$ is the start variable

5.0.1 Pumping lemma

Just as before with regular languages, there is a pumping lemma for context free languages.

If A is a context free language, then there is a number p (the pumping length) where if s is any string in A of at least length p, then s can be divided into the following 5 part structure $s = uvxyz$ where

- for each $i \geq 0, uv^ixy^iz \in A$
- $|vy| > 0$
- $|vxy| \leq p$

6 Push Down Automata

Like an NFA but contain a stack which provides additional memory. Can recognize some nonregular languages. PDA can recognize context free grammars and prove a language is context free.

PDA can write to the stack, and read them back later. Just like the stack in any other CS context, you can push and pop symbols and its a last-in, first-out structure.

Remember that problem with the equal 0s and 1s and how thats non-regular, well a pushdown automata can recognize it by reading symbols from input. Push each 0 onto the stack. For a 1, pop a 0 off the stack. If a 1 tries to pop an empty stack, a 0 comes after a 1, or the stack is not empty at the end of input, reject.

Now a formal definition where a pushdown automatais a 6 tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

1. Q is a finite set of states
2. Σ is a finite input alphabet
3. Γ is a finite stack alphabet
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ is the transistion function
5. $q_0 \in Q$ is the start state
6. $F \subseteq Q$ is is the finite set of accept states

And the formal definition of computation is as follows where w is the input string, r is the sequence of states, and s is the stack.

1. $r_0 = q_0$ and $s_0 = \varepsilon$. This means M starts properly on the right start state with an empty stack
2. For $i = 0, ..., m-1$ we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$ This is fancy math words for making sure M moves to the next state properly according to the state, stack, and next input symbol.
3. $r_m \in F$ The accept state is the last state.

A \$ on the stack represents an empty stack. $a, b \to c$ means that on, input a a transition is made, and if b is on the stack, pop b and push c onto the stack. All values can be ε

7 Turing Machines

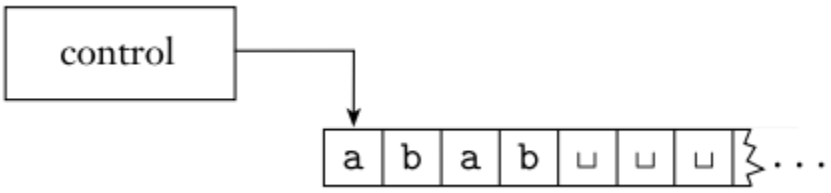
My homie Alan Turing thought of these bad boys.

7.1 Structure

It uses an infinite tape as memory. This tape starts with the input string, and is infinitely empty afterwards.

The head can move left and right and read or write symbols.

Reject and accept states happen immidiately.



7.2 Computing

Turing-Recognizable : A language is Turing-Recognizable if some Turing machine recognizes it
Decider : A Turing machine without a loop
Turing-Decidable : A decider recognizes the language

A Turing Machine has three possible outputs. It can accept input, reject input, or not halt and loop infinitely.

8 Algorithms

From CLRS pp5-7

Algorithm : Any well define computational procedure that takes input and produces output
Correct : When an algorithm where, for every input instance, it halts with the correct output

8.1 Runtime

8.1.1 Table of Formal Definitions

Name	Symbol	Informal Definition	Formal Definition
Little Omega	ω	Lower bound	$g(n) \in \omega(f(n)) \iff g(n) > cf(n) \forall n > n_0$
Big Omega	Ω	Tight Lower bound	$g(n) \in \Omega(f(n)) \iff g(n) \geq cf(n) \forall n > n_0$
Big Theta	Θ	Both an upper and lower bound	$g(n) \in \Theta(f(n)) \iff c_1 \leq g(n) \leq c_2 f(n) \forall n > n_0$
Big Oh	O	Tight Upper bound	$g(n) \in O(f(n)) \iff g(n) \leq cf(n) \forall n > n_0$
Little Oh	o	Upper bound	$g(n) \in o(f(n)) \iff g(n) < cf(n) \forall n > n_0$

8.1.2 Table of Limit Definitions

Name	Symbol	Proving with Limits
Little Omega	ω	$\lim_{n\to\infty} f(n)/g(n) = \infty$
Big Omega	Ω	$\lim_{n\to\infty} f(n)/g(n) \neq 0$
Big Theta	Θ	$\lim_{n\to\infty} f(n)/g(n) \neq 0, \infty$
Big Oh	O	$\lim_{n\to\infty} f(n)/g(n) \neq \infty$
Little Oh	o	$\lim_{n\to\infty} f(n)/g(n) = 0$