Discrete Mathematical Structures Two

# Definitions

Finite Automaton

* Accepts or rejects a string according to what is a valid string in a given language

Deterministic finite automaton

* Q, a set of states, one of which is the initial (or start) state, and a subset of which are final (or accepting) states.
* Σ, an alphabet of valid input symbols.
* δ : Q × Σ → Q, a function which, given a state and input symbol, determines the next state of the machine
* Must be exactly one transition out of every state for each symbol in the alphabet of the DFA

Non-deterministic finite automaton

* you cannot get to the end state with an invalid string
* there is at least one path to an acceptance state

Transition table

* The initial state appears first
* A plus sign indicates each final state
* columns are states
* rows are edges
* cells are the states that the edges lead to

Complement DFA

* simply invert the acceptability of each state in the machine
* commonly have dead states

Dead state/jail

* when there is no going back to an accepting state from the current state

Regular language

* is a formal language for which there exists a deterministic finite automaton that accepts all and only those strings in the language

Language

* Set of valid strings

Grammar

* Set of rules that you can use to generate a string for a language

Machine

* Finite Automaton
* given one string it will tell you whether it is in the language or not
  + accept and reject

Mealy machine

* rewrite system
* L-system

Parse

* Read a string and do something with it

Signature

* Name of function
* Valid input
* Valid output

Distinguishable

* exactly one of the states is an accepting state
  + different acceptability
* The states δ(q1, c) and δ(q2, c) are distinguishable for at least one symbol c ∈ Σ
  + move with the same symbol to states which have been previously found to be distinguishable

Parity bit

* is a bit that is added at the end of a bit string to indicate whether the number of 1-bits in the original string is even or odd. An even parity bit of 0 will be appended if the number of original 1-bits is even; Otherwise the even parity bit is 1.

Symbols

* δ
  + Transition function
  + Given a state and given a symbol, tell me what state to go to
    - it is like a mathematical function
    - δ((x,y),z)
      * x = start position
      * y = path to take
      * z = end position
  + represents the transition/edges in the machine
* Q {x, {y}}
  + x is the start of the machine
  + y is a set of all the final states of the machine
* \*
  + Kleene star
  + The set of all possible concatenations of elements of an arbitrary set of symbols
* ∑
  + Alphabet
* λ
  + Lambda

# Chapter One

# Chapter Two

# Chapter Three

# Chapter Four (Properties of Regular Languages)

Closure

* A term describing the property of operators that “stay within the same class of languages”
  + Union
  + Concatenation
  + Kleene star
  + Complement
  + Reversal
  + Intersection
  + Set difference
  + Set symmetric difference

Reversing a finite automaton

* Remove any jail states
* If there is more than one accepting state, create a new, unique, accepting state, reachable from what were the accepting states, by lambda transitions
* Make the new state the new start state
* Make the original start state the new, unique, accepting state
* Reverse all edges

Creating a DFA that is an intersection of two DFA’s

* Create a new starting state that is the combination of the starting states from the two tables
* Create a product table using the new starting state for the table
* Can be used to create DFAs from complicated languages that can be split into two smaller languages and then put back together
* Use logic to figure out the union, intersection, etc. It is based off of accepting states. You know this!

Determining if a DFA is infinite

* p = number of states in the DFA, counting the jail
* If a single string in the range of [p – (2p – 1)] is accepted then the DFA is infinite
  + ex. DFA has 4 states

Check all strings of length [4, 5, 6, 7]

If any of the strings are accepted then the DFA is infinite

* If the regular expression has a kleene star

Pumping Theorem

* Tests if an expression is non-regular
* Cannot test if an expression is regular
* x = all of the states before a cycle starts
* y = all of the states during the first cycle of the DFA
* z = all of the states after the first cycle is finished
* x and z can be empty
* All regular languages can be split into xyz
* if xyz is a string in the language then xyyz, xyyyz, xyyyyz … are also in the language

# Chapter Five (Pushdown Automata)

Pushdown Automaton (PDA)

* accepts context-free languages

Context-Free Languages

* Difference between deterministic and non-deterministic

Stack

* <read-char>, <pop-char> → <push-string>
* PDA only accepts if it ends in an accepting state and the stack is empty
* Lambda = nothing
* (q0, aabb, λ) ⊢ (q0, abb, X) ⊢ (q0, bb, XX) ⊢ (q1, b, X) ⊢ (q1, λ, λ)
  + is read as (state, remaining letters to read, stack) yields (state, remaining letters to read, stack)
  + shorthand, (q0, aabb, λ) ⊢ ∗ (q1, λ, λ)

# Chapter Six (Context-Free Grammar)

Formal Grammar Consists Of

* Non-terminal symbols
* Terminal symbols
* A set of rewrite rules

Non-Terminal Symbols

* AKA Variables
* One will be the start symbols

Terminal Symbols

* letters of the alphabet in use

Rewrite Rules

* Begins with the start variable
* Substitutes replacements for the variables
* Obtains a string n the language consisting only of terminal symbols
* AKA production rules or productions
  + ex. [Figure-6.1](images/Figure-6.1.jpg)

Context Free Grammars

* A set of variables (i.e., a non-terminal alphabet), V , one of which is a start variable
* An alphabet, Σ, of terminal symbols
* A set of production rules of the form v → s, where v ∈ V and s ∈ (V ∪ Σ)∗ , a string of terminals and/or variables

Senteniel Form

* Each step in a derivation

Leftmost Derivation

* It is customary to substitute for only one variable at a time
* Substituting the leftmost variable at each step

Derivation Tree

* Operations at lower levels of a derivation tree evaluate first
* Same concept as data structure tree with traversals
* When creating the leftmost child will be a copy of the parent
* E = +/-
* M = \*//
* P = ^
* R = values

Ambiguous Grammar

* A grammar that gives two distinct derivation trees (or two distinct leftmost derivations) for the same string
* Can be modified to be unambiguous by reflecting the order of operations in the grammar

Prefix notation

* Expression is formed from doing a preorder traversal of the tree (MLR)
* Evaluate expression right to left

Postfix notation

* Expression is formed from doing a post order traversal of the tree (LRM)
* Evaluate expression left to right

From CFG to PDA

* Text

  Description automatically generated with medium confidence
* Diagram

  Description automatically generated

From Single State PDA to a CFG

* A screenshot of a computer

  Description automatically generated with low confidence
* Diagram

  Description automatically generated
* A picture containing bubble chart

  Description automatically generated
* Text, letter

  Description automatically generated

From PDA to CFG (General Case)

* The symbol ⟨pV q⟩ represents the actions of a PDA that move from state p, having the symbol V at the top of the stack, to state q after one or more transitions, such that V and its aftereffects have all been popped from the stack.
* ReadCh, pop → push

# Chapter Seven

Prep For Chomsky Normal Form

* Remove Lambda from all rules
* Remove unit productions, such as x → y, while maintaining their effects

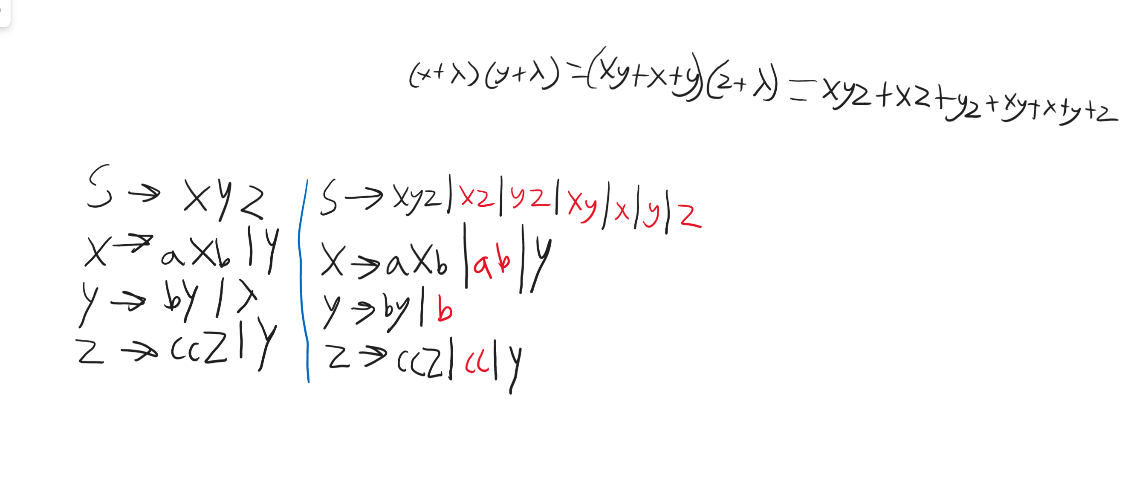
Nullable Variables

* A variable that can take a path that leads to a Lambda

Removing Lambdas

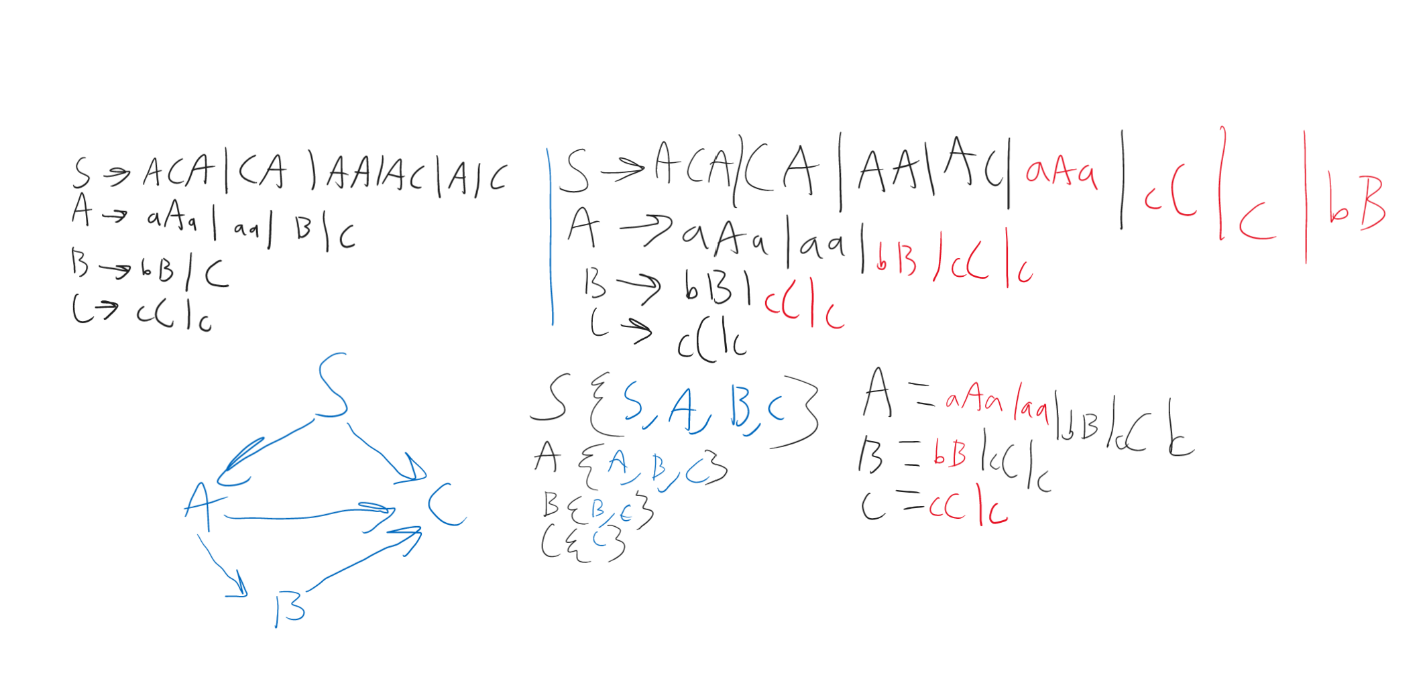
Text

Description automatically generated



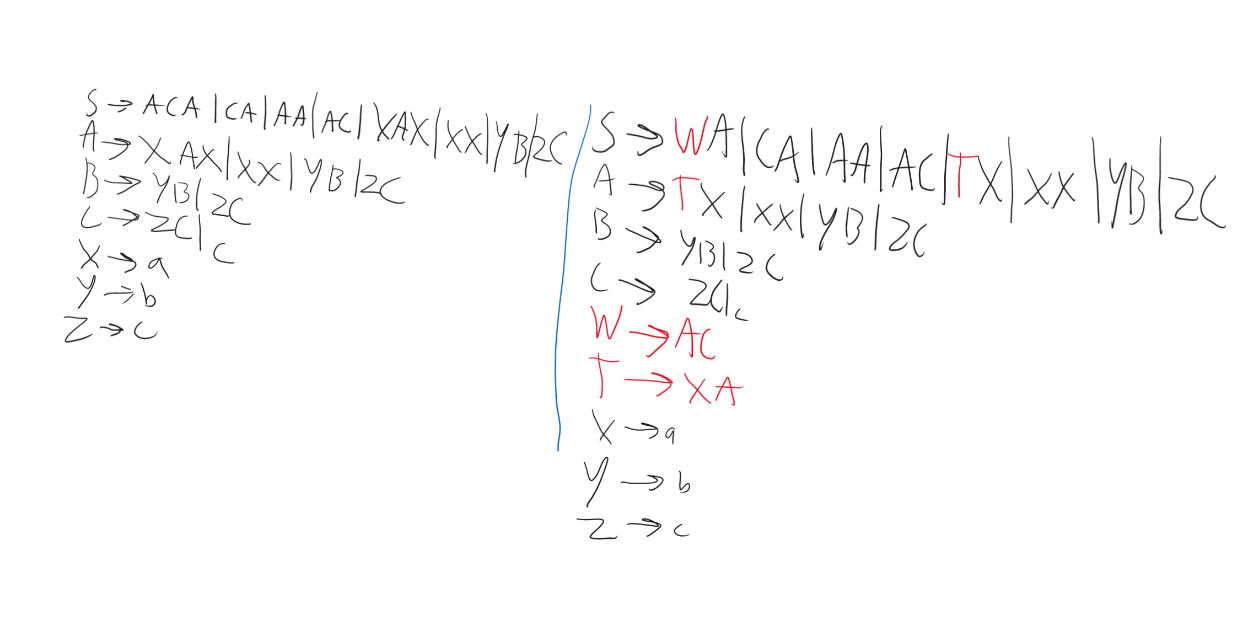
Remove Unit Productions

* Replace each individual instance of a variable with all of the non-individual values it is equal to



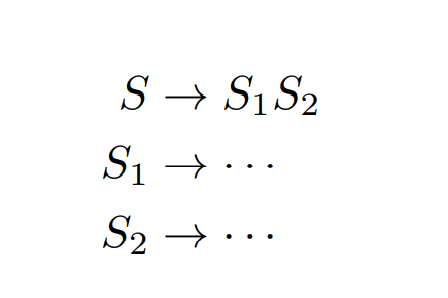
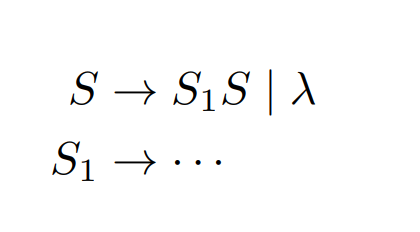
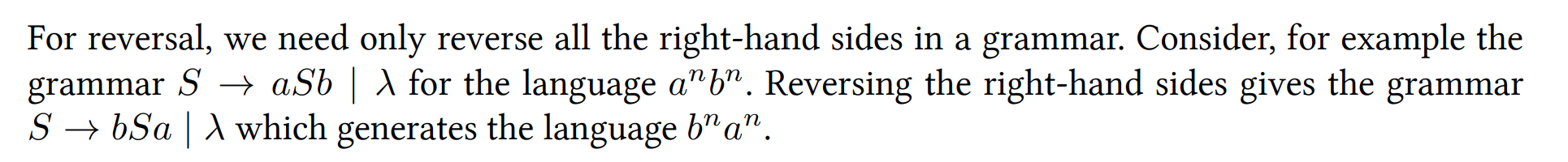
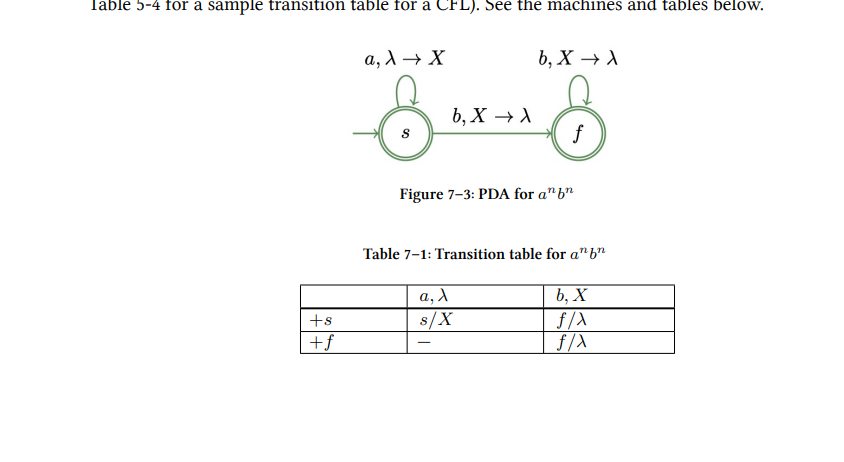
Chomsky Normal Form

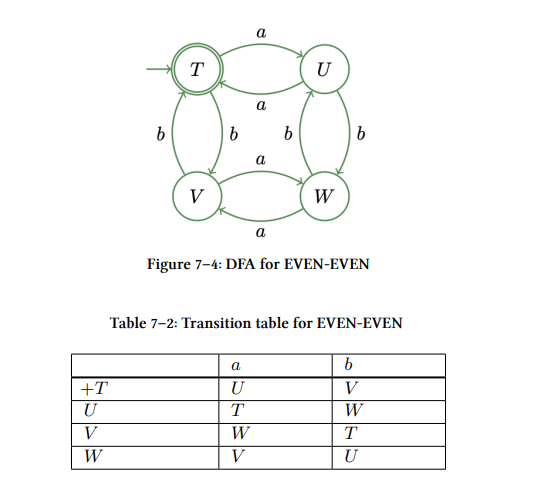
* Variables need to either equal 2 variables or 1 terminator
* Set each terminator to a Variable
* Where a variable equals more than 2 variables split that variable up



Closure Properties of CFLs

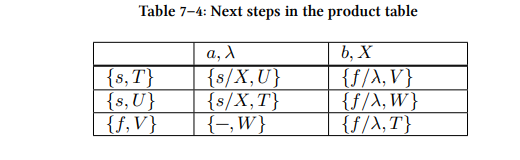
* Union
  + Chart, scatter chart

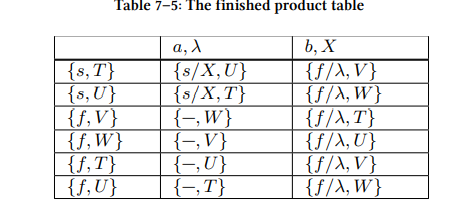
    Description automatically generated
* Concatenation
  + S → S1S2
* Kleene Star
  + 
* Reversal
  + 
* Regular Intersection, Union
  + 

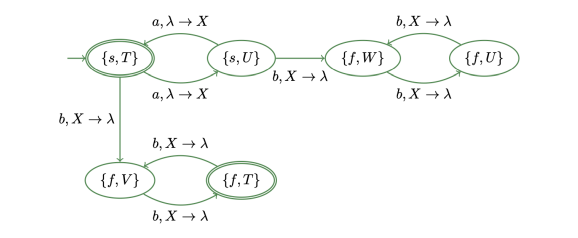


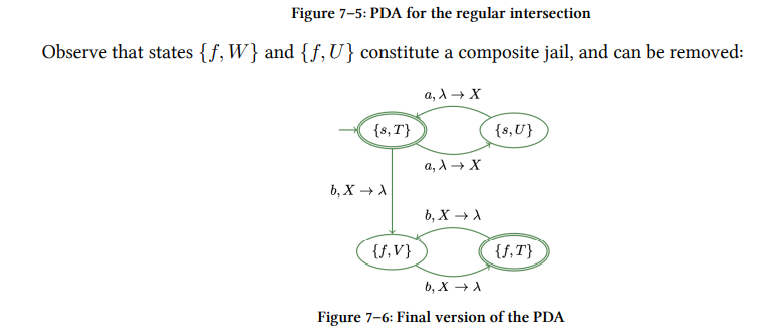
Table

Description automatically generated









Parsing Problem

# Chapter Eight

Post Machine

* Queue machine
* Is the same as using two stacks
* Is the same as using a Turing machine
* Uses an end-of-string marker
* Can cycle through the input repeatedly by popping symbols from the front and pushing them onto the back of the queue
* No separate input channel
* Queue is pre-populated with a $-delimited input string before we begin processing
* Pattern for the edges is
  + <pop-front/push-back>
* When the $ marker is reached the process restarts until the halt state begins

# Notes

Determining modulus with machines

* There are a number of states equal to the number you are modding against
  + 0 mod 3 has three states
    - states 0-2
  + 0 mod 6 has six states
    - states 0-5
* Each state is congruent to the number it is being modded against
  + state 0 is congruent to 0 mod 3
  + state 2 is congruent to 2 mod 6

Determining the redundant parts of a DFA

* If a state has the same output for the entire alphabet as another state, then that state is redundant
* make a table mark any combination of two states that have one of them as a final state as distinguishable.
* δ(q1, c) and δ(q2, c) are distinguishable for at leas one output is different from each other

Ways to describe a language

* English
* Transition table/graph
* Math
  + Alphabet: set of symbols
  + Q: set of states
  + δ: transition function
* Set of all valid strings (Roster notation)
* All strings that are accepted by a specific FA
* All strings generated by a specific grammar

Converting DFA to regular expression

Graphical user interface, text, application, email

Description automatically generated

Text, letter

Description automatically generated

# Important examples

Creating machine tips

* start by creating a ‘simple machine’ only create a single path that would lead to the correct end-state and then build off of it

Checking for distinguishability

* δ(q0, a) = q1, δ(q1, a) = q1

δ(q0, b) = q2, δ(q1, b) = q2

* if q0 and q1 have the same inputs and outputs then they are indistinguishable, and they can be combined.
* Building a table
  + mark all distinguishable pairs with a check mark
  + go to each empty cell and check if that pair is distinguishable or not
* if the pair (x, y) is indistinguishable, as well as (y, z), then the pair (x, z) is also an indistinguishable pair, and all three states can be combined
* into one state, {x, y, z}.

Minimization algorithm

* For a machine with n states, form a (n−1)×(n−1) triangular table, listing the first n−1

states down the left side as row headers, and the last n − 1 states horizontally as column

headers.

* Initialize the table by placing a check mark in those cells where exactly one of the

corresponding states in the pair is a final state.

* For each remaining empty cell: when the two corresponding states’ outputs are put into

a (x,y) pair if they lead to a marked cell, then they are distinguishable and mark that cell accordingly

* If a new check mark was entered anywhere in step 3, repeat step 3.
* Combine the states that correspond to any remaining empty cells.

Converting NFA to DFA Using a Transition Table

* Starting the table
  + column is the states
  + row is the alphabet
  + cells are the states that the transition table leads to
* Start with the starting state
* Only add a state to the column when it appears in a cell
* If a state has two paths through the same symbol create a new state that is the combination of the two states that the paths would lead to

Sequential Circuit

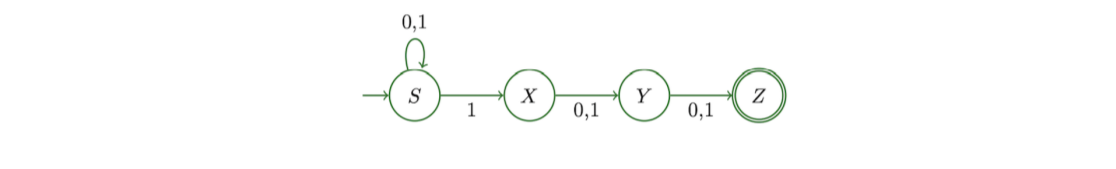
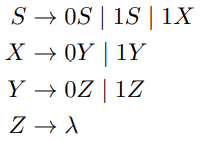
* row = inputs
* columns = value of variables
* cells = (x,y,…, n)/z
  + x = ending value of variable one
  + y = ending value of variable two
  + n = ending value of variable n
  + z = output

Computer Arithmetic in Mealy machines

* x/y
  + x = input
  + y = output

Regular grammar

* Right linear grammar
  + A formal grammar where there is at most one variable in any right-hand side rule, and if present, it is the rightmost symbol of the rule
  + Remove any jail states (they do not contribute to the language)
  + For each transition, form a grammar rule as follows:
    - Place the variable representing the from-state on the left-hand side of the rule
    - Form a string for the right-hand side of the rule by concatenating the character (or string, in the case of a GTG) on the transition followed by the variable representing the to-state
  + For each accepting state, X:
    - Add the rule X → λ

* Left linear grammar
  + If there are multiple final states, then create one final state that all the previous ones have lambda paths to
  + start at the final state and work right-to-left
  + start state becomes a final state

Converting left-linear grammar to right-linear grammar

Text

Description automatically generated

# Stuff to do for next time

* Reorganize Notes