Review for theory of computation textbook / slides

Proof dictionary

- Theorem 1.25 The class of regular languages is closed under the union operation.
- Theorem 1.39 every nondeterministic finite automaton as an equivalent deterministic finite automaton
    - Corollary 1.40 a language is regular if and only if some nondeterministic finite automaton recognizes it.
- Theorem 1.45 the class of regular languages is closes under the union operation
    - Two expressions connected before starting state by a state with two leading empty strings
- Theorem 1.47 the class of regular languages is closed under the concatenation operation
    - Two expressions connected by empty string
- Theorem 1.49 the class of regular languages is closed under the star operation
    - Expression connected to itself by empty strings from all accept states to starting state including a new accept state created before the starting state
- Theorem 1.54 a language is regular if and only if some regular expression describes it
    - Lemma 1.55 if a language is described by a regular expression, then it is regular.
        - By corollary 1.40 if an NFA recognizes A then A is regular
        - Prove regular by building up from the smallest subexpression
    - Lemma 1.60 if a language is regular then it is described by a regular expression
        - Use a generalized nondeterministic finite automaton
- Theorem 1.70 pumping lemma
    - If A is a regular language, then there is a number p(the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, s = xyz, satisfying the following conditions
        - For each i that is greater than or equal to 0, xy^iz member of A
        - $|y|$ greater than 0 and
        - $|xy|$ less than or equal to p

0.2

Jargon and notation

- Sets -- {} --
    - A group of objects represented as a unit
        - May contain any type of object including numbers symbols and even other sets
        - The objects in a set are called its elements or members
        - E shows set membership
        - ~~E~~ shows nonmembership
        - Subset operator Shows that it is a sub set with a line through it it is a proper subset
            - Proper subset means that <A> is a subset of <B> but not equal to it
        - Order describing a set is irrelevant
        - Repetition is irrelevant unless multi set
        - Set with 0 members is called empty set
        - Describing a set according to some rules
            - {<n>| rule about <n>}
- Union -- U --
    - Combining the elements of both sets into a single set called <A> U <B>
- Intersection -- intersection operator --
    - All elements in the set that are part of both sets
- Compliment -- <A>$^-$ --
    - Set of all elements not considered in set <A>
- Natural numbers -- N --

- Numbers 0-9 positive
- Integer -- Z --
  - Non fraction numbers infinite and positive and negative
- Sequence -- () --
  - A sequence of objects is a list of objects in some order designated by writing elements within parentheses
  - Order matters
  - Can be finite or infinite
  - Finite are called tuples
    - Sequence with <K> elements is a <K> tuple
    - 2 tuple sequence also called ordered pair
- Power set -- {} --
  - Set of all the sets
- Cartesian product -- X --
  - Set of all ordered pairs where the first element is a member of <A> and the second element is a member of <B>
- Function -- f(<A>) = <B> --
  - Also called mapping
  - An object that sets up an input-output relationship
    - f of <A> = <B>
  - The set of possible inputs is called the domain -- D --
  - The outputs of a function is the range -- R --
- Formal description of Finite automaton
  - 5-tuple (Q, sigma, triangle, q_0, F)
    - Q finite set called states
    - Sigma is a finite set called the alphabet
    - Triangle : Q x sigma leads to Q is the transition function
    - q_0 is a member of Q is the start state
    - F is a subset of Q is the set of accept state
- Straight lines -- l<n>l --
  - The length of string n

- Alphabet -- sigma --
  - Any non empty finite set
  - Members are the symbols
- Order
  - *
    - DFA reference to itself
  - Concatenation
    - DFA leads to another state
  - Union
    - DFA leads to another state through another state

1.3

- Formal definition of regular expression
  - R is a regular expression if R is
    - <A> for some <A> in the alphabet sigma
    - It is in the set (E)
    - An empty set
    - R1 U R2 when both are regular expressions
    - R1 o R2 when both are regular expressions
    - R1* where R1 is a regular expression

1.3

Generalized nondeterministic finite automaton

- The start state has transition arrows going to every other state but no arrows coming in from any other state
- There is only a single accept state and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state
- Except for the start and accept states, one arrow goes from every stat to every other stat and also from each state to itself

- Converting DFA into GNFA
  - Add new start state with epsilon to the old start state and a new accept state with epsilon arrow from the old accept states
  - If any arrows have multiple labels or if there are multiple arrows going between the same two states in the same direction we replace each with a single arrow whose label Is the union of the previous label
  - Add arrows labeled empt set between states that had no arrows
- Must have different accept and start state
- If k states must have greater than or equal to 2 states, will equal k-1 state if greater than 2
- Remove state called q_{rip}
- Repaired labels go from q_i to q_j

Formal description of generalized nondeterministic finite automaton

- 5 tuple
  - Q is finite set of states
  - Sigma is alphabet
  - Delta : (Q-{qaccept})x(Q-{qstart}) leads to $\underline{R}$ transition
    - $\underline{R}$ = collection of all regular expressions over sigma
  - q_{start} is the start state, and
  - q_{accept} is the accept state

CONVERT(G):

- K = number of states of G
- K=2 G must consist of a start state, an accept state and a single arrow connecting them and labeled with a regular expression R
  - Return expression R
- If K greater than 2 select any state q_{rip} membership Q different from q_{start} or q_{accept} and let G' be the GNFA (Q', sigma, delta', q_{start}, q_{accept}), where
  - Q' = Q - {q_{rip}},
- And For any q_i member of Q' - {q_{accept}} and any q_j member of Q' - {q_{start}}, let
  - Delta' (q_i,q_j) = (R1)(R2)*(R3)U(R4),
- For R1 = delta(q_i, q_{rip}), R2= delta(q_{rip}, q_{rip}), R3= delta(q_{rip}, q_j) and R4 = delta(q_i,q_j)
- Compute CONVERT(G') and return this value

1.2

- Formal description of nondeterministic finite automaton
  - 5 tuple (Q, sigma, delta, q_0, F)
    - Q is a finite set of states
    - Sigma is a finite alphabet
    - Delta: Q x $\text{sigma}_{\text{epsilon}}$ leads to P(Q) is the transition function
      - Transition function takes a state and an input symbol or the empty string and produces the set of possible next states
      - P(Q) power set of Q
        - The collection of all subsets of Q
      - $\text{Sigma}_{\text{epislon}}$ means sigma<alphabet> union{ empty string}
    - Q_0 membership Q is the start state
    - F is a subset of Q is the set of accept states

1.4

- Pumping lemma
  - Used to prove that a language is regular
  - All strings in a language can be pumped if they are at least as long as the pumping length
  - If A is a regular language, then there is a number p(the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, s = xyz, satisfying the following conditions
    - For each I is greater than or equal to 0, xy^iz member of A
    - lyl greater than 0 and
    - lxyl less than or equal to p
  - Pigeonhole principle if p pigeons are placed in fewer than p holes more than one pigeon is in each hole
  - For DFA M
    - N=length of sequence states
    - S= input ({s1, s2, s3, ... sn})
    - S is divided into three pieces x,y,z
      - X = before repeating step
      - Y = between both ends of repeat step
      - Z = after repeating step
    - Thus DFA M will accept xy^iz for i=0, xy^iz=xz condition 1 satisfied
    - lyl is greater than 0 satisfying condition 2
    - p+1 states must have repetition and lXYl is less than or equal to p

Theorem 0.20 For any two sets A and B, not(AUB) = not(A) intersect not(B)
Theorem 0.21 For every graph G, the sum of the degrees of all the nodes in G is an even number
Theorem 0.22 For each even number n greater than 2, there exist a 3-regular graph with n nodes
Theorem 1.25 The class of regular languages is closed under the union operation.
Theorem 1.26 the class of regular languages is closed under the concatenation operation
Theorem 1.39 every nondeterministic finite automaton as an equivalent deterministic finite automaton

- Corollary 1.40 a language is regular if and only if some nondeterministic finite automaton recognizes it.

Theorem 1.45 the class of regular languages is closes under the union operation

- Two expressions connected before starting state by a state with two leading empty strings

Theorem 1.47 the class of regular languages is closed under the concatenation operation

- Two expressions connected by empty string

Theorem 1.49 the class of regular languages is closed under the star operation

- Expression connected to itself by empty strings from all accept states to starting state including a new accept state created before the starting state

Theorem 1.54 a language is regular if and only if some regular expression describes it

- Lemma 1.55 if a language is described by a regular expression, then it is regular.
  - By corollary 1.40 if an NFA recognizes A then A is regular
  - Prove regular by building up from the smallest subexpression
- Lemma 1.60 if a language is regular then it is described by a regular expression
  - Use a generalized nondeterministic finite automaton

Theorem 1.70 pumping lemma

- If A is a regular language, then there is a number p(the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, s = xyz, satisfying the following conditions
  - For each i that is greater than or equal to 0, xy^iz member of A
  - lyl greater than 0 and
  - lxyl less than or equal to p

Definition 1.64

- Formal description of generalized nondeterministic finite automaton
  - 5 tuple
    - Q is finite set of states
    - Sigma is alphabet
    - Delta : (Q-{qaccept})x(Q-{qstart}) leads to <u>R</u> transition
      - <u>R</u> = collection of all regular expressions over sigma
    - q_{start} is the start state, and
    - q_{accept} is the accept state

Definition 1.16 a language is called a regular language if some finite automaton recognizes it

- Formal description of nondeterministic finite automaton
  - 5 tuple (Q, sigma, delta, q_0, F)
    - Q is a finite set of states
    - Sigma is a finite alphabet
    - Delta: Q x $sigma_{epsilon}$ leads to P(Q) is the transition function
      - Transition function takes a state and an input symbol or the empty string and produces the set of possible next states
      - P(Q) power set of Q
        - The collection of all subsets of Q
      - $Sigma_{epislon}$ means sigma<alphabet> union{ empty string}
    - Q_0 membership Q is the start state
    - F is a subset of Q is the set of accept states

Definition 1.52

R is a regular language if R is

A for some a in the alphabet sigma

Epsilon

empty string

R1 U R2 both regular expressions

R1 concate R2 both regular expressions

R1* where R1 is regular expressions

Definition 2.2

A context-free grammar is a 4 tuple (V, sigma, R, S)

V is a finite set called the variables

Sigma is a finite set, disjoint from V, called terminals

R is a finite set of rules, with each rule being a variable and a string of variables and terminals,

S is a member of V is the start variable

Definition 2.7

A string w is derived ambiguously in context free grammar G if it has two or more different leftmost derivations. Grammar G is ambiguous if it generates some string ambiguously

Theorem 4.1

A_{DFA} = { <*B*, *w*> I *B* is a DFA that accepts input string *w*}

A_{DFA} is a decidable language

M = " On input <*B*, *w*>, where *B* is a DFA and *w* is a string:

1. Simulate *B* on input *w*.
2. If the simulation ends in an accept state, *accept*. If it ends in a non accepting state, *reject*."

Theorem 4.2

A_{NFA} = { <*B*, *w*> I *B* is an NFA that accepts input strings *w*}

A_{NFA} is a decidable language

N = " On input <*B*, *w*>, where *B* is an NFA and *w* is a string:

1. Convert NFA *B* to an equivalent DFA *C*, using the procedure for this conversion given in theorem 1.39.
2. Run TM *M* from theorem 4.1 on input < *C*, w>.
3. If *M* accepts, *accept*; otherwise, *reject*."

Theorem 4.3

Let A_{REX} = { < *R*, *w*> I *R* is a regular expression that generates string *w*}

A_{REX} is a decidable language

P = " On input <*R*, *w*>, where *R* is a regular expression and *w* is a string:

1. Convert regular expression *R* to an equivalent NFA *A* by using the procedure for this conversion in theorem 1.54
2. Run TM *N* on input <*A*, *w*>.
3. If *N* accepts, *accept*; if *N* rejects, *reject*."

Theorem 4.4

E_{DFA} = { <*A*> I *A* is a DFA and L(*A*) = null set}

E_{DFA} is a decidable language

T = " On input <*A*>, where *A* is a DFA:

1. Mark the start state of *A*.
2. Repeat until no new states get marked:
    3. Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, *accept*; otherwise, *reject*.

Theorem 4.5

EQ_{DFA} = { <*A*, *B*> I *A* and *B* are DFAs and L(*A*) = L(*B*) }

EQ_{DFA} is a decidable language

F = " On input <*A*, *B*>, where *A* and *B* are DFAs:

1. Construct DFA *C* that recognizes L(*C*) = (L(*A*) intersection of the compliment of L(*B*)) union of (the compliment of L(*A*) intersection of L(*B*)).
2. Run TM *T* from theorem 4.4 on input <*C*>
3. If *T* accepts, *accept*. If *T* rejects, *reject*."

Theorem 4.7

A_{CFG} = { <*G*, *w*> I *G* is a CFG that generates string *w*}

A_{CFG} is a decidable language

S = " On input <*G*, *w*>, where *G* is a CFG and *w* is a string:

1. Convert *G* to an equivalent grammar in Chomsky normal form.
2. List all derivation with 2*n* -1 steps, where *n* is the length of *w*; Except if *n* = 0, then instead list all derivations with one step.
3. If any of these derivations generate *w*, *accept*; if not, *reject*"

Theorem 4.8

E_{CFG} = { <G> | G is a CFG and L(G) = null set}

E_{CFG} is a decidable language

R = "On input <G>, where G is a CFG:

1. Mark all terminal symbols in G.
2. Repeat until no new variables get marked:
3. Mark any variable A where G has a rule A -> $U_1U_2....U_k$ and each symbol U_1, ... , U_k has already been marked.
4. If the start variable is not marked, *accept*; otherwise *reject*."

EQ_{CFG} = { <G, H> | G and H are CFGs and L(G) = L(H)} ?

Theorem 4.9

Every context free language is decidable

M_G = " On input w:

1. Run TM S on input <G, w>.
2. If this machine accepts, *accept*; if it rejects, *reject*."

Theorem 4.11

A_{TM} = { <M, w> | M is a TM and M accepts w}

A_{TM} is undecidable

U = " On input <M, w> where M is a TM and w is a string:

1. Simulate M on input w
2. If M ever enters its accept state, *accept*; if M ever enters its reject state, *reject*"

Theorem 4.22

A language is decidable if it is turing recognizable and co turing recognizable

Show that decidable languages are closed under concatenation

Let K and L be decidable languages

Let KL = { xy | x member of k and y member of L }

Let M_K and M_L decide K and L respectively

M_{KL} = " On input <w> where w is a string

1. Non-deterministically partition w into x and y.
2. Input x to M_k and y to M_L.
3. *Accept* if both M_K and M_L accept, otherwise *reject*.

Show that decidable language are closed under star

Let L be decidable language

Let L* = { x member of L U LL U LLL ... } all strings obtained by concatenating L with L

Let M_L decide L

ML* = " On input <w> where w is a string

1. partition w non-deterministically into $w_1 w_2 .... w_N$
2. Run M_L with input w_p for p= 1, ... , N
3. If ML accept each string w_p *accept*, otherwise *reject*

Show that recognizable languages are closed under union

Let L_1 and L_2 be recognizable languages

Let L_U = L_1 U L_2

Let M_1, M_2 and M_U recognize L_1, L_2 and L_U respectively

M_U = " On input <w> where w is a string

1. Run M_1 and M_2 with input w in parallel
2. If either M_1 or M_2 accept, *accept*, otherwise *reject*.

Show that recognizable languages are closed under intersection

Let L_1 and L_2 be recognizable languages

Let $L_U = L_1$ intersect L_2

Let M_1, M_2 and M_U recognize L_1, L_2 and L_U respectively

M_U = " On input <*w*> where *w* is a string

1. Run M_1 and M_2 with input *w* in parallel
2. If both M_1 or M_2 accept, *accept*, otherwise *reject*.

Show that recognizable languages are closed under concatenation

Let L_1 and L_2 be recognizable languages

Let $L_U = L_1 \circ L_2$

Let M_1, M_2 and M_U recognize L_1, L_2 and L_U respectively

M_U = " On input <*w*> where *w* is a string

1. Non-deterministically partition *w* into *x* and *y*.
1. Run M_1 and M_2 with input *w* in parallel
2. If both M_1 or M_2 accept, *accept*, otherwise *reject*.

Show that recognizable languages are closed under star

Let L be recognizable language

Let L_U = { x member of L U LL U LLL ... } all strings obtained by concatenating L with L

Let M_1, and M_U recognize L_1 and L_U respectively

M_U = " On input <*w*> where *w* is a string

1. partition *w* non-deterministically into $w_1 w_2 .... w_N$
2. Run M_L with input w_p for p= 1, ... , N
3. If ML accept each string w_p *accept*, otherwise *reject*

CFG

Divide and conquer relax and turn into smaller logical parts, programming is your ace.