Proof techniques: construction, contradiction, counterexample, case enumeration, induction, pigeonhole principle, proving cardinality, diagonalization

Languages, Strings

Alphabet: finite set of symbols

String: finite sequence of symbols from alphabet

Replication: $w^0 = \varepsilon, w^{i+1} = w^i w$

Reverse: $w^R = w = \varepsilon$ if $|w| = 0$, else $\exists a \in \Sigma$ and $\exists u \in \Sigma^*$ such that $w = ua$, then define $w^R = au^R$

$\boldsymbol{(wx)^R = x^r w^r}$: induction on $|x|$, base case $|x| = 0$ so $x = \varepsilon$, consider any string x where $|x| = n+1$, then $x = ua$ for some character a and $|u| = n$, so $(wx)^R = (w(ua))^R = (wu)a^R = a(wu)^R = a(u^R w^R) = (au^R)w^R = (ua)^R w^R = x^R w^R$

Language: set of strings (finite/infinite) from alphabet, uncountably infinite number of these (power set of Σ^*)

$\boldsymbol{\Sigma^*}$: countably infinite with non-empty alphabet, enumerate with lexicographic order

$\boldsymbol{L_1 L_2}$: $\{w \in \Sigma^* : \exists s \in L_1(\exists t \in L_2(w = st))\}$

$\boldsymbol{L^*}$: $\{\varepsilon\} \cup \{w \in \Sigma^* : \exists k \geq 1(\exists w_1, w_2, \ldots w_k \in L(w = w_1 w_2 \ldots w_k))\}$ or $L^0 \cup L^1 \cup L^2 \cup \ldots$

$\boldsymbol{L^+}$: LL^* or $L^* - \{\varepsilon\}$ iff $\varepsilon \notin L$ or $L^0 \cup L^1 \cup L^2 \cup \ldots$

$\boldsymbol{(L_1 L_2)^R = L_2^R L_1^R}$: $\forall x(\forall y((xy)^R = y^R x^R))$ from before, then $(L_1 L_2)^R = \{(xy)^R : x \in L_1 \text{ and } y \in L_2\} = \{y^R x^R : x \in L_1 \text{ and } y \in L_2\} = L_2^R L_1^R$

Decision problem: problem to which answer is yes/no or true/false

Decision procedure: answers decision problem

Machine power hierarchy: FSM (regular), PDA (context-free), TM (semi-decidable & decidable)

Rule of least power: use least powerful language suitable for expressing info, constraints or programs on WWW

$\boldsymbol{firstchars(L)}$: $\{w : \exists y \in L(y = cx \wedge c \in \Sigma_L \wedge x \in \Sigma_L^* \wedge w \in c^*)\}$, closed under FIN but not INF (since result is first character *)

$\boldsymbol{chop(L)}$: $\{w : \exists x \in L(x = x_1 c x_2, x_1 \in \Sigma_L^*, x_2 \in \Sigma_L^*, c \in \Sigma_L, |x_1| = |x_2|$, and $w = x_1 x_2)$
, language where all strings have exact middle character removed, must have had odd length to begin with, closed under FIN but not INF (can get empty set if never odd length)

Extra: to describe language with at least 2 different substrings of length 2 $L = \{w \in \{a,b\}^* : \exists x, y(x \neq y \wedge |x| = 2 \wedge |y| = 2 \wedge Substr(x, w) \wedge Substr(y, w))\}$

Finite State Machines

DFSM Quintuple: $M = (K, \Sigma, \delta, s, A)$, K = finite set of states, Σ = alphabet, δ = transition function from $(K \times \Sigma)$ to K, $s \in K$ = initial state, $A \subseteq K$ = set of accepting states

Configuration: element of $K \times \Sigma^*$, current state and remaining input

Yields relation: $|-_M$, relates 2 configurations if M can move from the first to the second in 1 step, $|-_M^*$ for 0 or more

Computation: finite sequence of configurations for some $n \geq 0$ such that C_0 is an initial configuration, C_n is of the form (q, ε) for some state $q \in K_M$ and $C_0 |-_M C_1 |-_M C_2 |-_M \ldots |-_M C_n$

DFSM will halt in $|w|$ steps: execute computation from C_0 to C_n, each step will consume one character, so $n = |w|$, C_n is either accepting or rejecting configuration, so will halt after $|w|$ steps

Parity: odd if number of 1 is odd for binary string

$\boldsymbol{MinDFSM(M : DFSM)}$:
Initialise classes with an accepting class & non-accepting class
For each class with more than 1 state
For each state and character check which class it goes to
If behaviour differs between states split them
End for
End for
Go through all the classes again until no splitting happens
Each class becomes its own state, transitions already defined above

Number of states $\geq$ equivalence classes in L: suppose it is less than equivalence classes, then by pigeonhole principle there must be at least 1 state that contains strings from 2 equivalence classes, but then future behaviour on these two strings will be identical, which is not consistent with the fact that they are in different equivalence classes

NDFSM Quintuple: replace δ with Δ, transition relation, finite subset of $(K \times (\Sigma \cup \{\varepsilon\})) \times K$

NDFSM vs DFSM: can enter configuration with input symbols left but no move available (halt without accepting), can enter configuration from which 2 or more competing transitions available (ε-transition, more than 1 transition for single input character)

$\boldsymbol{eps(q)}$: $\{p \in K : (q, w) |-_M^* (p, w)\}$, closure of $\{q\}$ under relation $\{(p, r)$: there is a transition $(p, \varepsilon, r) \in \Delta\}$, to calculate initialise $result = \{q\}$, add all transitions $(p, \varepsilon, r) \in \Delta$ where $p \in result, r \notin result$ to $result$, then return $result$

$\boldsymbol{ndfsmtodfsm(M : NDFSM)}$:
Compute $eps(q)$ for each state q, $s' = eps(s)$ (initial state)
Set $active\text{-}states = \{s'\}$ (set of set of states) and $\delta' = \varnothing$
While $\exists Q \in active\text{-}states$ for which δ' has not been computed //computing δ'
For each $c \in \Sigma_M$
Set $new\text{-}state = \varnothing$
For each state $q \in Q$
For each state $p : (q, c, p) \in \Delta$
Set $new\text{-}state = new\text{-}state \cup eps(p)$
End for
Add $(Q, c, new\text{-}state)$ to δ', if $new\text{-}state \notin active\text{-}states$ insert it
End for
End for
End while
Set $K' = active\text{-}states$ and $A' = \{A \in K' : Q \cap A \neq \varnothing\}$

Extra: when making FSM may start with complement