8-10; Finals Review

Monday, August 10, 2020 9:29 AM

Turing machines

Review
- language is closed?? properties of reg languages
- guide to lava diagram
- self-referencing or self-defeating TM/ object
- pigeonhole principle ("use it when ... always happens")

Finite Automata

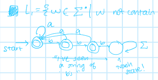
try working outside → in

CFGs
- Designing CFGs
  1. write out examples & look for patterns
     - internal repetition? → recursively repeat
     - *. "I need x for every y" → x & y needs to be added at same time!
     - independent sections? → seperate non-terminals
  2. Break up the string into repeating & independent sub-sections

$L = \{w \in \Sigma^* \mid w \text{ has no a's or has no b's}\}$
$S \to A \mid B$
$A \to aA \mid \varepsilon$
$B \to bB \mid \varepsilon$

DFAs
(DFAs technically type of NFA)
- exactly one transition per character for every state
- "transitioning between equivalence classes"
- $D = \{Q, \Sigma, \delta, q_0, F\}$
- Designing DFAs –
  - useful to use trash states
  - states keep track of info seen so far ($\delta^*(s)$)
  - Transitions = how does what I know change w/ each char

$L_1 = \{w \in \Sigma^* \mid w \text{ not contain 'bbb' as substring}\}$



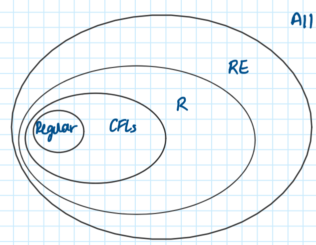
NFAs
- no limits on # transitions
- can use ε transitions
- → computational paths can "die"
  - don't need as many states tracking info as w/ DFAs
- You can chain together two NFAs, N_1 & N_2 w/ an ε, the new NFA has a language equal to $L(N_1) \cdot L(N_2)$
- Designing NFAs – (nondeterminism!)
  - remember, multiple paths available
  - trash states not actually necessary
  - guess-and-check method

Reg Exs
- Designing Regexs
  1. write out examples looking for patterns & required chars
     - repetition? → kleene stars
     - subgroups? → concatenate
     - choices? → unions

$L_2 = \{w \in \Sigma^* \mid w \text{ has odd \# a's}\}$
a, aaa, abaa ...
↳ $(b^*ab^*ab^*)^* a (b^*ab^*ab^*)^*$ — one required a; a's are paired!

*. You can convert btwn DFAs, NFAs & RegExs, b/c they all represent Regular languages

The Lava Diagram



- Regular languages
  - = can be expressed in a DFA = has a finite # of equivalence classes over indistinguishability (= for every string $w_0 \in [w]_{\equiv_L}$, $\delta^*(w_0) = \delta^*(w)$)
  - only need finite memory to express
- context-free languages = can be expressed in a CFG (aw not any more than that)
- R (decidable) = there exists a decider Turing machine that returns accept / reject, doesn't loop
- RE (recognizable) = has a (verifier) TM that can loop, for when you can't prove is terminating
  - ↳ mainly for reasoning w/ TMs
- "All" / Not in RE = cannot be recognized by any TM (diagonalizational problem)

Induction

(look for "all n∈N")
- = use when proving a statement for any/all natural numbers, when proving a recursive definition (look for recursion)
- "if it starts true & stays true, it is always true."
- basic induction: prove a base case, assume P(k) & prove P(k+1)
- complete induction: prove a base case, assume P(0) ... P(k), & prove P(k+1) (when step size is unclear, or when building up)
- If P(n) is existential → build up = start by growing 1 add on to base case
- If P(n) is universal → build down = start high level w/ an arbitrary object, break it down & prove that it matches P(k+1)
  - usually done w/ complete induction

Functions

Binary Relations
- some properties:
  - Reflexive: $\forall a \in A.\ aRa$
  - Symmetric: $\forall a \in A\ \forall b \in A.\ aRb \to bRa$
  - Transitive: $\forall a \in A\ \forall b \in A\ \forall c \in A.\ aRb \wedge bRc \to aRc$
  - Irreflexive: $\forall a \in A.\ a \not R a$
  - Asymmetric: $\forall a \in A\ \forall b \in A.\ aRb \to b \not R a$

Proving any of these:
1. Pick arbitrary elements
2. Demonstrate (if antecedent true) the consequent property is true using definition of R

- Equivalence relations: Reflexive, Symmetric, Transitive
  - ↳ Equivalence classes = all groups relating to each other ($\forall a_0 \in [a]_R,\ a_0 R a$)
    - ↳ system of Relations = set of one elem from each equiv. class
  - ↳ Partitions naturally create equivalence classes of an arbitrary equiv. relation
- Strict orders: irreflexive, asymmetric & transitive (really only need one of these)

Graphs
- Graph theory usually relates to functions – how does each node relate to nodes it's connected to?