



Midterm I review

Reading: Chapters 1-4



Test Details

- In class, Wednesday, Feb. 25, 2015
3:10pm-4pm
- Comprehensive
- Closed book, closed notes



Syllabus

- Formal proofs
- Finite Automata
 - NFA, DFA, ϵ -NFA
- Regular expressions
- Regular language properties
 - Pumping lemma for regular languages
 - Note: closure properties and minimization of DFAs – not included



Finite Automata

- **Deterministic Finite Automata (DFA)**
 - The machine can exist in only one state at any given time
- **Non-deterministic Finite Automata (NFA)**
 - The machine can exist in multiple states at the same time
- ϵ -NFA is an NFA that allows ϵ -transitions
- What are their differences?



Deterministic Finite Automata

- A DFA is defined by the 5-tuple:
 - $\{Q, \Sigma, q_0, F, \delta\}$
- Two ways to define:
 - State-diagram (preferred)
 - State-transition table
- DFA construction checklist:
 - Associate states with their meanings
 - Capture all possible combinations/input scenarios
 - break into cases & subcases wherever possible
 - Are outgoing transitions defined for every symbol from every state?
 - Are final/accepting states marked?
 - Possibly, dead/error-states will have to be included depending on the design.



Non-deterministic Finite Automata

- A NFA is defined by the 5-tuple:
 - $\{Q, \Sigma, q_0, F, \delta\}$
- Two ways to represent:
 - State-diagram (preferred)
 - State-transition table
- NFA construction checklist:
 - Has *at least* one nondeterministic transition
 - Capture only valid input transitions
 - Can ignore invalid input symbol transitions (paths will die implicitly)
 - Outgoing transitions defined only for valid symbols from every state
 - Are final/accepting states marked?



NFA to DFA conversion

- Checklist for NFA to DFA conversion
 - Two approaches:
 - Enumerate all possible subsets, or
 - Use **lazy construction** strategy (to save time)
 - Introduce subset states only as needed
 - In your solutions, use the lazy construction procedure by default unless specified otherwise.
 - Any subset containing an accepting state is also accepting in the DFA
 - Have you made a special entry for Φ , the empty subset?
 - This will correspond to the dead/error state



ϵ -NFA to DFA conversion

- Checklist for ϵ -NFA to DFA conversion
 - First take ECLOSE(start state)
 - New start state = ECLOSE(start state)
 - Remember: ECLOSE(q) include q
- Then convert to DFA:
 - Use *lazy construction* strategy for introducing subset states only as needed (same as NFA to DFA), but ...
 - Only difference : take ECLOSE after transitions and also include those states in the subset corresponding to your destination state.
 - E.g., if q_i goes to $\{q_j, q_k\}$, then your subset must be: ECLOSE(q_j) \cup ECLOSE(q_k)
- Again, check for a special entry for Φ if needed



Regular Expressions

- A way to express accepting patterns
- Operators for Reg. Exp.
 - (E) , $L(E+F)$, $L(EF)$, $L(E^*)$..
- Reg. Language \rightarrow Reg. Exp. (checklist):
 - Capture all cases of valid input strings
 - Express each case by a reg. exp.
 - Combine all of them using the $+$ operator
 - Pay attention to operator precedence
 - Try to reuse previously built regular expressions wherever possible



Regular Expressions...

- DFA to Regular expression
 - Enumerate all paths from start to every final state
 - Generate regular expression for each segment, and concatenate
 - Combine the reg. exp. for all each path using the + operator
- Reg. Expression to ϵ -NFA conversion
 - Inside-to-outside construction
 - Start making states for every atomic unit of RE
 - Combine using: concatenation, + and * operators as appropriate
 - For connecting adjacent parts, use ϵ -transitions
 - Remember to note down final states



Regular Expressions...

- Algebraic laws
 - Commutative
 - Associative
 - Distributive
 - Identity
 - Annihilator
 - Idempotent
 - Involving Kleene closures (* operator)



English description of lang.

- Finite automata → english description
- Regular expression → english description

“***English description***” should be similar to how we have been describing languages in class

- E.g., languages of strings over $\{a,b\}$ that end in b ; or
- Languages of binary strings that have 0 in its even position, etc.

Thumbrule: the simpler the description is, the better.
However, make sure that the description should accurately capture the language.



Pumping Lemma

- Purpose: Regular or not? Verification technique
- Steps/Checklist for Pumping Lemma (in order):
 - 1) Let $N \leftarrow$ pumping lemma constant
 - 2) Choose a template string w in L , such that $|w| \geq N$.
(Note: the string you choose should depend on N . And the choice of your w will affect the rest of the proof. So select w judiciously. Generally, a simple choice of w would be a good starting point. But if that doesn't work, then go for others.)
 - 3) Now w should satisfy P/L, and therefore, all three conditions of the lemma. Specifically, using conditions $|xy| \leq N$ and $y \neq \epsilon$, try to conclude something about the property of the xy part and y part separately.
 - 4) Next, use one of these two below strategies to arrive at the conclusion of $xy^kz \notin L$ (for some value of k):
 - Pump down ($k=0$)
 - Pump up ($k \geq 2$)Note: arriving at a contradiction using either pumping up OR down is sufficient. No need to show both.

Working out pumping lemma based proofs as a 2-player game:

- Steps (think of this 2-party game):

Good guy (us)

Builds w using N
(without assuming
any particular value of N)

Tries to break the third condition
of P/L without assuming any
particular $\{x,y,z\}$ split

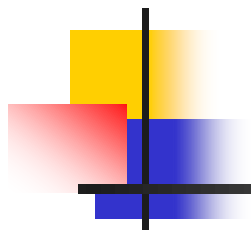
- this is done by first pumping down
($k=0$)
- if that does not work, then try
pumping up ($k \geq 2$)

Bad guy (someone else)

Claims L is regular

\Rightarrow Knows N and has the freedom
to choose any value of $N \geq 1$

Comes up with $\{x,y,z\}$ combination,
s.t. $w=xyz$
(again, has the freedom to choose
any xyz split, but meeting
the two conditions of P/L:
i.e., $|xy| \leq N$ and $y \neq \epsilon$)



GOOD LUCK!