Cpts 317
Wed Feb 26
Mid-term review
A few notes

* Course work / Grading
  - 8 Homeworks (60%)
    - 7 best policy
  - 2 mid-term (20%)
  - 1 final term (20%)

* Homeworks are much more different from the mid-terms (or even the final)

* Mid-term & final have space & time constraint
Mid-term 1

* 6 - 8 problems

* You will have the page left open for answering your questions.

* Points will be indicated.
Problems

- Write down a regular expression for the following language

- Give a finite automaton accepting this regex expression
  (your choice of NFA or DFA)
  (State diagram)

- Convert this NFA to DFA
- Show that this language is not a regular language using the pumping lemma.

- Given a language \( L_3 \) defined by some operation \( op \) on \( L_1 \) and \( L_2 \):
  \[ L_3 = L_1 \text{ or } L_2 \]
  Show that if \( L_1 \) and \( L_2 \) are regular, so is \( L_3 \).

- State true or false for statements regarding regular expressions.

- State true or false for statements regarding NFA and DFA.
- Give English description of the language represented by the regular expression.

- Convert the DFA into a 2-state NFA.

- Give CFG generating this language.
Topics Covered

1. Regular Language
   1.1. Finite automata
       - Formal definition of FA
       - Formal definition of computation
       - Designing FA
       - Regular operators
   1.2. Non determinism
       - Formal definition of NFA
       - Equivalence of NFAs & DFAs
       - Closure under regular operators
   1.3. Regular Expression
       - Formal definition of reg. exp.
       - Equivalence with FA
   1.4. Non regular language - PL
2. Context-free language

2.1. Context-free grammars
   - formal definition of CFG
   - Examples of CFGs
   - Designing CFGs

   - Ambiguity
   - Chomsky Normal form

2.2. Push Down Automata
   - formal definition of a PDA
   - Examples
   - Equivalence with CFGs
Context-free language

- Context-free grammar
  - PDA

Regular language

- Regular expressions
  - NFA

Non-regular language
e.g. $L = \{0^n 1^n \mid n \geq 0\}$
**NFA**s and **DFA**s

* Be able to clearly state the different characteristics of an NFA and a DFA

<table>
<thead>
<tr>
<th>DFA</th>
<th>VFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Every state has exactly one exit arrow for each symbol.</td>
<td>* Not an extent</td>
</tr>
<tr>
<td>* Labels on transition arrows are symbols from the alphabet.</td>
<td>* May have arrows labelled with members of the alphabet or $E$.</td>
</tr>
<tr>
<td></td>
<td>Zero, one or more arrows may exit from each state with label $E$.</td>
</tr>
</tbody>
</table>
NFA Computation

* After reading a symbol, a machine splits into multiple copies of itself and forms all possibilities in parallel.

* Process continues until some copy, or one (or more) of the copies, reaches an accept state at the end of the input; in which case the NFA accepts the input string.
NFA to DFA Conversion

By product of the theorem

Every non-deterministic FA has an equivalent DFA.

→ Convert the NFA that recognizes the language into an equivalent DFA that simulates the NFA.

→ Key idea: If \( k \) is the number of states of the NFA, it has \( 2^k \) subsets of states. You need to keep track of these to do the conversion.
# See proof idea sketch in lecture of Jan 29.

- Case when \( N \) has no \( \varepsilon \) arrows
- Case when \( N \) has \( \varepsilon \) arrows

Example \( N_4 \)

![Diagram](image-url)
Convert regular expressions into an NFA

(lecture of Jan 31)

Example: Convert the reg. exp. 
\((ab \cup a)^*\) into an NFA.

\[ a \xrightarrow{a} 0 \]
\[ b \xrightarrow{b} 0 \]
\[ ab \xrightarrow{a} 0 \xrightarrow{e} 0 \xrightarrow{b} 0 \]
\[ ab \cup a \xrightarrow{e} 0 \xrightarrow{a} 0 \]
\[ (ab \cup a)^* \xrightarrow{\epsilon} a \xrightarrow{e} 0 \xrightarrow{e} b \xrightarrow{0} \]
Conversion of DFA into regular expression

By present of the result/Lemma

If a language is regular, then it
is described by a reg. exp.

Prog Idea

1. DFA -> Generalized NFA
2. GNFA -> reg. exp.

GNFA
- Are NFA, more transitions arrows
may have any reg. exp. as labels,
read blocks of symbols from input.
- non determinate
DFA $\rightarrow$ GNFA

* Add a new start state with an $\varepsilon$ arrow to the old start state.
* Add a new accept state with $\varepsilon$ arrows from the old accept states.

* If any arrows have multiple labels, replace each with a single arrow whose label is the union of the previous labels.
* Add arrows labeled $\emptyset$ between states that had no arrows.
To convert GNFA → reg. exp.

* Support GNFA has k states (k ≥ 2)
* If k > 2, construct an equivalent GNFA with k-1 states.
* Repeat until the new GNFA is reduced to 2-states.

Example

3-state DFA → 5-state GNFA → 4-state GNFA

Reg. Exp.
"Ripping"
(any state that is not start state or accept state will do)

before

\[ \begin{array}{cc}
\text{P1} & \rightarrow \\
\text{P2} & \rightarrow \\
\text{P3} & \rightarrow \\
\text{P4} & \rightarrow \\
\end{array} \]

\[ L, R_2^+, R_3, U R_4 \]

after

\[ \begin{array}{cc}
\text{P1} & \rightarrow \\
\end{array} \]
Example

```
Example

```

```
Example

```

```
Example

```

```
Example

```
Pumping Lemma (How to apply)

See lecture of Feb 12