CS 360
Programming Languages

Regular Expressions
Regular Languages
Finite State Automata
Regular Expressions

• We will be discussing *Basic Regular Expressions*
  – So, e.g., not PERL Regular Expressions

• The most basic regular expression is a single character, which represents itself
Building Larger Regexes

There are three basic operations, listed here in increasing order of precedence. Given regexes $R_1$ and $R_2$:

- **Choice**
  
  $R_1 \mid R_2$ is a regex

- **Concatenation**
  
  $R_1R_2$ is a regex

- **Closure**
  
  $R_1^*$ is a regex
Some Basic Conventions

• Λ is typically the alphabet
• Use parenthesis () for grouping, to force precedence
• Use epsilon, ε, for the empty string
Other Operators

You probably know some other operators, but these are syntactic sugar, and don’t bring anything new to the table

- \( R? \equiv (R \mid \epsilon) \)
- \( R+ \equiv RR* \)
- \([a-z] \equiv (a \mid b \mid ... \mid z)\)
Regular Languages

- The set of strings described by a regular expression, $R$, is a *Regular Language*
- Denoted $L(R)$
- Regular languages are a proper subset of context-free languages (coming soon)
Finite State Automata

- FSA, or FSM (finite state machines), recognise regular languages
- The set of languages generated by regular expressions is exactly the set of languages recognised by FSA (proof by construction, coming soon)
- Finite number of states, finite number of transitions among states, a single start state, and a subset of accepting states
Deterministic Finite Automata (DFA)

- For each pair of state and input symbol, there is only 1 transition
- By convention, if no transition exists, the input string fails
  - Transitions on incorrect inputs can be added that lead to a “black hole” state, which is not accepting
- So, there is always exactly 1 current state
Nondeterministic Finite Automata (NFA)

• Same state machine
• For each pair of state and input symbol there may be more than possible transition
• A transition may be labeled with $\varepsilon$ (epsilon); this transition happens (may happen) w/out consuming any input
• There is a set of possible current states
• If any current states are an accepting state at the end of input, then the machine accepts
NFA vs. DFA

• These machines are equivalent:
  – Trivially, a DFA is an NFA
  – Given any NFA, a DFA can be constructed
    • If the NFA has $N$ states, then the equivalent DFA might have $2^N$ states
• Both automata recognise only regular languages
Creating a FSA from a Regex

• We can construct larger machines from smaller ones
• Each machine will have exactly one accepting state
• Given 2 black-box machines that recognise strings in $L(R_1)$ and $L(R_2)$:
Creating a FSA from a Regex

- $R_1 \mid R_2$:

- $R_1 R_2$:

- $R_1^*$: