CS 536 Announcements for Tuesday, February 1, 2022

Course websites:
pages.cs.wisc.edu/~hasti/cs536/
www.piazza.com/wisc/spring2022/compsci536

- waitlisted folks: feel free to add yourself to Piazza

Programming Assignment 1
- test code due Friday, Feb. 4 by 11:59 pm
- other files due Tuesday, Feb. 8 by 11:59 pm

Last Time
- start scanning
- finite state machines
  - formalizing finite state machines
  - coding finite state machines
  - deterministic vs non-deterministic FSMs

Today
- non-deterministic FSMs
- equivalence of NFAs and DFAs
- regular languages
- intro regular expressions

Next Time
- regular expressions
- regular expressions $\rightarrow$ DFAs

Recall
- scanner : converts a sequence of characters to a sequence of tokens
- scanner implemented using FSMs
- FSMs can be DFA or NFA

Creating a scanner

scanner $=$

```
  token to regex +
  regex to NFA +
  NFA to DFA +
  DFA to code
```

scanner generator
NFAs, formally

finite state machine $M = (Q, \Sigma, \delta, q, F)$

$L(M) = \text{the language of FSM } M = \text{set of all strings } M \text{ accepts}$

Example:

"Running" an NFA

To check if a string is in $L(M)$ of NFA $M$, simulate set of choices it could make.

The string is in $L(M)$ iff there is at least one sequence of transitions that

- consumes all input (without getting stuck)
- ends in one of the final states
NFA and DFA are equivalent

Two automata $M$ and $M^*$ are equivalent iff $L(M) = L(M^*)$

Lemmas to be proven:

Lemma 1: Given a DFA $M$, one can construct an NFA $M^*$ that recognizes the same language as $M$, i.e., $L(M^*) = L(M)$

Lemma 2: Given an NFA $M$, one can construct a DFA $M^*$ that recognizes the same language as $M$, i.e., $L(M^*) = L(M)$

Proving Lemma 2

Lemma 2: Given an NFA $M$, one can construct a DFA $M^*$ that recognizes the same language as $M$, i.e., $L(M^*) = L(M)$

Part 1: Given an NFA $M$ without $\varepsilon$-transitions, one can construct a DFA $M^*$ that recognizes the same language as $M$

Part 2: Given an NFA $M$ with $\varepsilon$-transitions, one can construct a NFA $M^*$ without $\varepsilon$-transitions that recognizes the same language as $M$
**NFA without ε-transitions to DFA**

**Observation:** We can only be in finitely many subsets of states at any one time.

**Idea:** To do NFA $M \rightarrow$ DFA $M^*$, use a single state in $M^*$ to simulate sets of states in $M$.

Suppose $M$ has $|Q|$ states. Then $M^*$ can have only up to $2^{|Q|}$ states.

Why?

### Example

\[
\begin{align*}
&\begin{array}{ccc}
A & B & C \\
0 & 0 & 0 = & 205 \\
0 & 0 & 1 = & 2C3 \\
0 & 1 & 0 = & 2B3 \\
0 & 1 & 1 = & 2B1C3 \\
1 & 0 & 0 = & 2A3 \\
1 & 0 & 1 = & 2A1C3 \\
1 & 1 & 0 = & 2A1B3 \\
1 & 1 & 1 = & 2A1B1C3
\end{array}
\end{align*}
\]

### Example

\[
\begin{array}{ccc}
A & B & C & D \\
A & 2A3 & 2B3 & 2B3 \\
B & 2C3 & 2C3 & 2C3 \\
C & 203 & 203 & 203 \\
D & 23 & 23 & 23
\end{array}
\]
NFA without $\varepsilon$-transitions to DFA

Given NFA $M$:

Build new DFA $M^*$

To build DFA: Add an edge from state $S$ on character $c$ to state $S^*$ if $S^*$ represents the set of all states that a state in $S$ could possibly transition to on input $c$.

Any state whose subset contains a final state of $M$ is final state in $M^*$.

Part 1: NFA w/o $\varepsilon$ $\rightarrow$ DFA $\checkmark$

$\varepsilon$-transitions

Example: $x^n$, where $n$ is even or divisible by 3.
Eliminating \( \epsilon \)-transitions

**Goal:** given NFA \( M \) with \( \epsilon \)-transitions, construct an \( \epsilon \)-free NFA \( M^* \) that is equivalent to \( M \)

**Definition:** epsilon closure  
\[ \text{eclose}(s) = \text{set of all states reachable from } s \text{ using } 0 \text{ or more epsilon transitions} \]

\[ \begin{array}{c|c}
\text{State} & \text{eclose} \\
\hline
P & \{P, Q, R, S\} \\
Q & \{Q\} \\
R & \{R\} \\
Q_1 & \{Q_1\} \\
R_1 & \{R_1\} \\
R_2 & \{R_2\} \\
\end{array} \]

1) Make \( S \) an accepting state of \( M^* \) iff \( \text{eclose}(S) \) contains an accepting state of \( M \)

2) Add edge from \( S \) to \( T \) labeled \( a \) iff there is an edge labeled \( a \) in \( M \) for some state in \( \text{eclose}(S) \) to \( T \)

3) Delete all edges labeled with epsilon

**Part 2:** NFA w/\( \epsilon \) \( \rightarrow \) NFA w/o \( \epsilon \)

Lemma: NFA \( \rightarrow \) DFA \( \checkmark \)
Summary of FSMs

DFAs and NFAs are equivalent
- an NFA can be converted into a DFA, which can be implemented via the table-drive approach

\( \varepsilon \)-transitions do not add expressiveness to NFAs
- algorithm to remove \( \varepsilon \)-transitions

Regular Languages and Regular Expressions

Regular language
Any language recognized by an FSM is a **regular language**

Examples:
- single-line comments beginning with `//`
- hexadecimal integer literals in Java
- C/C++ identifiers
- \{\( \varepsilon \), ab, abab, ababab, abababab, …\}

Regular expression \( \text{(regex)} \)
= a pattern that defines a regular language

**regular language**: set of (potentially infinite) strings

**regular expression**: represents a set of (potentially infinite) strings by a single pattern

Example: \{\( \varepsilon \), ab, abab, ababab, abababab, …\} \( \leftrightarrow \) (ab)*

Why do we need them?
- Each token in a programming language can be defined by a regular language
- Scanner-generator input = one regular expression for each token to be recognized by the scanner

\( \rightarrow \) **regexs are inputs to scanner generator**
Regular expressions

Formal definition

A regular expression over an alphabet \( \Sigma \) is any of the following:
- \( \emptyset \) (the empty regular expression)
- \( \varepsilon \)
- \( a \) (for any \( a \in \Sigma \))

Moreover, if \( R_1 \) and \( R_2 \) are regular expressions over \( \Sigma \), then so are: \( R_1 \mid R_2 \), \( R_1 \cdot R_2 \), \( R_1^* \)

Regular expressions as an expression language

regular expression = pattern describing a set of strings

operands: single characters, epsilon

operators:
- alternation ("or"): \( a \mid b \)
- concatenation ("followed by"): \( a.b \) \( ab \)
- iteration ("Kleene star"): \( a^* \)

Conventions

\( \begin{align*}
\text{aa} & \text{ is } a.a \\
\text{a}^+ & \text{ is } aa^* \\
\text{letter} & \text{ is } a|b|c|d|...|y|z|A|B|...|Z \\
\text{digit} & \text{ is } 0|1|2|...|9 \\
\text{not(x)} & \text{ is all characters except } x \\
\text{parentheses} & \text{ for grouping and overriding precedence, e.g., (ab)*}
\end{align*} \)

Example: single-line comments beginning with //

Example: hexadecimal integer literals in Java
- must start \( 0x \) or \( 0X \)
- followed by at least one hexadecimal digit (hexdigit)
  - hexdigit = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, A, B, C, D, E, F
- optionally can add long specifier (\( I \) or \( L \)) at end