CS 536 Announcements for Thursday, February 3, 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
- non-deterministic FSMs
- equivalence of NFAs and DFAs
- regular languages
- intro regular expressions

Today
- regular expressions
- regular expressions \( \rightarrow \) DFAs
- language recognition \( \rightarrow \) tokenizers

Next Time
- scanner generators
- JLex

Recall

scanner =

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

scanner generator
Warm-up / review: NFA $\rightarrow$ DFA

Given the following NFA with $\varepsilon$-transitions, convert it to an equivalent DFA

What is $L(M)$?  $L(M) = \varepsilon a, aa, aca$

**NFA w/ $\varepsilon$ $\rightarrow$ NFA w/o $\varepsilon$**

<table>
<thead>
<tr>
<th>State</th>
<th>$\varepsilon$-closure</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>$\varepsilon P_0, P_1, P_2, P_3$</td>
</tr>
<tr>
<td>$P_1$</td>
<td>$\varepsilon P_1, P_3$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$\varepsilon P_2, P_3$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$\varepsilon P_3, P_4, P_5$</td>
</tr>
<tr>
<td>$P_4$</td>
<td>$\varepsilon P_4$</td>
</tr>
</tbody>
</table>

1) If $\varepsilon$-close(s) contains a final state in $M$, then $s$ is a final state in $M^*$
2) If $\exists$ edge $s^* \rightarrow t^*$ in $M$ where $s^* \in \varepsilon$-close(s) & $t^* = \varepsilon$-close(t), then there's edge $s \rightarrow t$ in $M^*$
3) remove $\varepsilon$ edges

NFA w/o $\varepsilon$ $\rightarrow$ DFA

**$M^{**}$$^\$

Add edge in $M^{**}$ $s \rightarrow t$ for all $s^* \varepsilon s$

Any state whose subset contains a final state in $M^*$ will be a final state in $M^{**}$
Regular Expressions

regular expression = pattern describing a language

operands: single characters, epsilon

operators:

- alternation ("or"): $a$ | $b$  
  matches $a$, matches $b$
- concatenation ("followed by"): $a.b$  
  matches $ab$
- iteration ("Kleene star"): $a^*$  
  matches $0$ or more $a$s in $E$, $a, a a, a a a, a a a a a, ...$

Conventions

- $aa$ is $a.a$
- $a+$ is $aa^*$
- letter is $a|b|c|d|...|y|z|A|B|...|Z$
- digit is $0|1|2|...|9$
- not($x$) is all characters except $x$
- parentheses for grouping and overriding precedence, e.g., (ab)*

Example: single-line comments beginning with //</br>

```
// not(\n\n)* \n```

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 (l or L) at end

```
0(x|X) hexdigit+ (E|L|L)
```

where hexdigit = digit | a | b | c | ... | f | A | B | C | ... | F

Example: C/C++ identifiers (with one added restriction)

- sequence of letters/digits/underscores
- cannot begin with a digit
- cannot end with an underscore

```
(letter | \_ ) (letter | digit | \_ )* (letter | digit) \| letter
```
From regular expressions to NFAs

Conversion of literals and epsilon
- Convert operands to NFAs
- Join NFAs

Conversion of operators

Regex to NFA rules

Rules for operands

literal 'a': $O \xrightarrow{a} O$
epsilon $\varepsilon$: $O \xrightarrow{\varepsilon} O$

Suppose $A$ is regex w/ NFA:

- Make these non-final
- Convert so only 1 final state
Rules for iteration $A^*$

Tree representation of a regex

\[(\text{letter} \mid \text{'-'} \mid \text{digit})^*\]
Bottom-up conversion
Bottom-up conversion (cont.)

Regex to DFA

We now can do:

We can add one more step: **optimize DFA**

**Theorem:** For every DFA $M$, there exists a unique equivalent smallest DFA $M^*$ that recognizes the same language as $M$.

**To optimize:**

- remove unreachable states
- remove dead states
- merge equivalent states

But what's so great about DFAs?
Table-driven DFAs

Recall: state-transition function \( \delta \) can be expressed as a table

\[ \rightarrow \text{very efficient array representation} \]

\[ \rightarrow \text{efficient algorithm for running (any) DFA} \]

\[ s = \text{start state} \]
\[ \text{while (more input)} \}
\[ \quad c = \text{read next char} \]
\[ \quad s = \text{table}[s][c] \]
\[ \} \]
\[ \quad \text{if } s \text{ is final, accept} \]
\[ \quad \text{else reject} \]

What else do we need?

**FSMs** – only check for language membership of a string

**scanner** needs to

- recognize a stream of many different tokens using the longest match
- know what was matched

**Idea**: augment states with actions that will be executed when state is reached