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

\[
\text{scanner} = \text{token to regex} + \text{regex to NFA} + \text{NFA to DFA} + \text{DFA to code}
\]
Warm-up / review: NFA $\rightarrow$ DFA

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

**regular expression** = pattern describing a language

**operands:** single characters, epsilon

**operators:**

- alternation ("or"):  a | b

- concatenation ("followed by"):  a.b  ab

- iteration ("Kleene star"):  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 //

**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

**Example:** C/C++ identifiers (with one added restriction)
- sequence of letters/digits/underscores
- cannot begin with a digit
- cannot end with an underscore
From regular expressions to NFAs

Conversion of literals and epsilon
Conversion of operators

Regex to NFA rules

Rules for operands
Regex to NFA rules

Rules for alternation  A|B

Regex to NFA rules

Rules for catenation  A.B
Regex to NFA rules

Rules for iteration $A^*$

Tree representation of a regex
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 ( ) can be expressed as a table

→ very efficient array representation

→ efficient algorithm for running (any) DFA

```java
s = start state
while (more input){
    c = read next char
    s = table[s][c]
}
if s is final, accept
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