DFAs/NFAs continued and Parse trees / ASTs

CS230 Tutorial 09
Regexs from English

- Give a regex for:
  - All 11-digit phone numbers with a leading 1, dashes for separation
  - A UWaterloo building/room number combination
    - Assume 2-4 letter building abbreviations (we don’t care if it is a real building)
    - Examples: E5-5022   DC-3548   ZQTP-1524
  - Placements from 1st to 9th
    - Examples: 3rd   4th
Regexs from English

● Give a regex for:

○ All 11-digit phone numbers with a leading 1, dashes for separation
  ■ 1-[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9][0-9]

○ A UWaterloo building/room number combination
  ■ [A-Z][A-Z0-9][A-Z]?[A-Z]?-[0-9][0-9][0-9][0-9]

○ Placements from 1st to 9th
  ■ (1st|2nd|3rd|[4-9]th)
Deterministic Finite Automata (DFAs)

- Regex’s can be converted to DFAs with a series of rules
  - $a^*$
  - $a^+$
  - $a?$
  - $a|b = [ab]$
DFA Practice

- Convert the following regular expressions to DFAs (alphabet is: a,b,c,d)
  - $a^*(b|c)?$
  - $a+b?c^*[b-d]+$  
  - $ab[^ad]+db?$
DFA Practice: Answers

- Convert the following regular expressions to DFAs (alphabet is: a,b,c,d)
  - $a^*(b|c)?$  
    - $a+b?c^*[b-d]^+$
      - you can write this more simply as $a+[b-d]^+$
  - $ab[^ad]+db?$
    - here note: $[^ad] = [bc]$

- Notice that some of these were quite difficult? NFAs make this much easier!
NFA Practice

- Convert the following regular expressions to NFAs (alphabet is: a,b,c,d)
  - a*(b|c)?
  - a+b?c*[b-d]+
  - ab[^ad]+db?
NFA Practice: Answers

- Convert the following regular expressions to NFAs (alphabet is: a,b,c,d)
  - $a^*(b|c)?$*
  - $a+b?c^*[b-d]+$
  - $ab[^ad]+db$?
    again, remember
    $[^ad] = [bc]$

- NFAs let you just go through
  the regex in order and convert
  each step (allows epsilon transitions, and multiple transitions on the same
  terminal). But, they are ambiguous!
Tokenizing

We already talked about tokenizing during the last tutorial. But just as a reminder:

- **Tokenizing** is splitting the input stream into component parts as recognized by the language’s grammar
- These component parts are called **tokens**
- Examples include: variable name, keyword, number, etc.

The scanner breaks the input up into tokens.
Parsing

Parsing looks as the token stream (i.e. the list of tokens we got from the scanner) and checks to make sure the rules of the grammar are all being followed.

So for example:

5 + + 6

this does not follow the math grammar, since you can’t have 2 +’s in a row like this.

From the parser, we can get a parse tree or an AST.
CFG - Context Free Grammar

- the opposite of “context sensitive” grammar
- **context free**: this just means the meaning of something is independent of what's around it
- Important Note: not all CFGs are regular grammars (this is another class of grammars, with some overlap)

So, an example of a context sensitive language:

I am going to the **park**.
I am going to **park** my car.

Here, context is key! The same word means different things depending on where it is. Clearly, English is not a context-free language :P
AST/Parse Tree - example

These are both ways of visualizing a parse.

Consider the following grammar (E3 from class, just with the nonterminals renamed) (where E = Expression, T = Term, F = Factor):

\[
E \rightarrow E + T \\
\rightarrow E - T \\
\rightarrow T \\
T \rightarrow T * F \\
\rightarrow T / F \\
\rightarrow F \\
F \rightarrow ( E ) \\
\rightarrow \text{int (actual number)}
\]

Example: draw the parse tree and the AST for “(3 * (2 + 7)) / 6” in this language.
Example: draw the parse tree and the AST for “(3 * (2 + 7)) / 6” in this language.

Solution: check out the parse tree on the side.
Example: draw the parse tree and the AST for “(3 * (2 + 7)) / 6” in this language.

Solution: check out the AST on the side. Notice how it’s wayyy simpler than the parse tree, but means the exact same thing!
The general idea

You can always represent your parse with either a parse tree and/or an AST.

- **Parse tree**: has more information about which rules were followed and which nonterminals appeared along the way (encodes the entire derivation)

- **AST**: less massive and unwieldy (simpler to work with and easier on the eyes)
Assignment reminders

● Submit a .txt XOR a .pdf for each question
  ○ Do not submit both for the same question!
  ○ You may submit a .pdf for one question and a .txt for a different question

● Make sure your diagrams and tables are clear and easy to read
  ○ Make sure to leave enough space