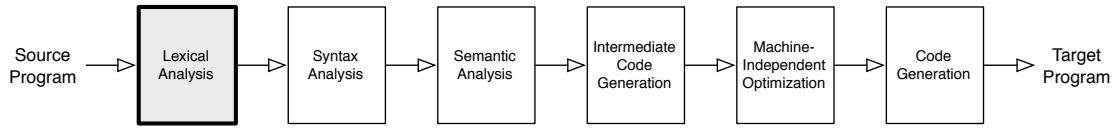
CS 301 Spring 2019
Tutorial Assignment
5 February

Regular Expressions and Finite Automata

1 Plan



Our first tutorial meeting focuses on two topics:

- **Compilation Overview.** The first reading expands on our exploration of the compilation pipeline with TINY, considers how programming language design and computer architecture affect compiler design, and surveys key programming language ideas.
- **Lexical Analysis.** The first step in compilation is *lexical analysis*, turning a string of source code characters into a stream of meaningful *lexemes* or *tokens* in the language. The reading introduces the theory of lexical analysis (regular expressions, transition diagrams, finite automata) and how this connects to implementation.

2 Readings

"Alternative" readings are an optional second perspective if the primary option is not a good fit for you.

- Dragon 1
  partial alternative: EC 1
- Dragon 3.1.2, 3.3, 3.5–3.7
  alternative: EC 2.1–2.4.3, 2.4.5
- *Regular Expression Matching Can Be Simple And Fast (but is slow in Java, Perl, PHP, Python, Ruby, ...).* Russ Cox, January 2007. `https://swtch.com/~rsc/regexp/regexp1.html`
  Read for basic ideas and commentary. The C code is less important.

3 Exercises

1. Dragon 1.1.2
2. Dragon 1.1.4
3. Dragon 1.6.1
4. Dragon 3.3.2
5. Dragon 3.3.9
6. Write a regular expression for HTTP and FTP URLs. A URL consists of four parts: the protocol (`http://` or `ftp://`), the domain name or the IP address of a host, an optional port number, and an optional pathname for a file. For simplicity, we assume that:
   - A domain name is a list of non-empty alphabetical strings separated by periods.
   - An IP address is four non-negative integers of at most three digits each, separated by periods.

- A port number is a positive integer following a colon. (*e.g.*, :8080)
- The pathname is a Unix-style absolute pathname. The allowed symbols are letters, digits, period, slash. A sequence of two consecutive slashes `//` is forbidden, *i.e.*, no empty directory name.
- A URL may end with a slash as long as it does not create the sequence `//`.

7. Write the DFAs for each of the following:
   - Binary numbers that contain the substring 011.
   - Binary numbers that are multiples of 3 and have no consecutive 1's. Your solution can accept or reject the empty string – either is fine. (You may find it easiest to create DFAs for each of the two requirements and then think about how to combine them.)

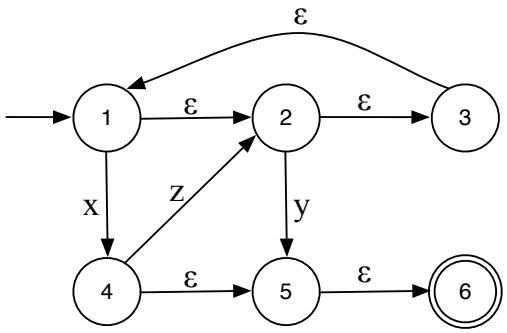
8. A comment in the C language begins with the two-character sequence `/*`, followed by the body of the comment, and then the sequence `*/`. The body of the comment may not contain the sequence `*/`, although it may contain the sequence `/*`, or the characters `*` and `/`. We use the notation $(E)^*$ for the Kleene closure of E; all other occurrences of `*` refer to the character itself, not to the Kleene operator.

   (a) Show that the following regular expression does not correctly describe C comments:

   $$\texttt{/*}\ (\texttt{/})^*\ \Big(\texttt{[}\hat{}\texttt{*/]}\ \Big|\ \texttt{[}\hat{}\texttt{*]/}\ \Big|\ \texttt{*[}\hat{}\texttt{/]}\Big)^*\ (\texttt{*})^*\ \texttt{*/}$$

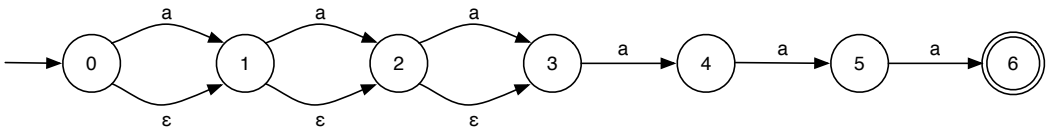
   (b) Draw the DFA that accepts C comments and then use it to write the regular expression that correctly describes C comments.

9. Convert the following NFA to a DFA. For each DFA state, indicate the set of NFA states to which it corresponds. Make sure you show the initial state and the final states in the constructed DFA.



10. Russ Cox describes how inefficient some regular expression pattern matchers can be.

    (a) To motivate the design and to see how a little bit of theory can dramatically improve software design, we start by comparing the two simulation algorithms in section "Regular Expression Search Algorithms" section of Cox's article.
    Here is the NFA corresponding to "`a?a?a?aaa`":



    Enumerate all paths taken through the NFA using the backtracking search algorithm on input `aaab`. How many are there? Given the NFA for $(\texttt{a?})^n\texttt{a}^n$, how many paths may need to be explored to test an input string of length $n+1$? Give a Big-O bound and a one or two sentence explanation.

11. Show the steps performed by Algorithm 3.22 in Dragon. (This is a more precise and succinct description of Cox's second algorithm). It suffices to show the states in S each time line (3) is executed. You will find it useful to compute the ϵ-closure for each state in the NFA.