CMPSC 461: Programming Language Concepts, Fall 2025
Assignment 1

Prof. Suman Saha

Due: 11:59 PM, September 12, 2025

General Instructions:
You need to submit your homework to **Gradescope**. Do **every problem on a separate page and mark** them before submitting. **If the questions are not marked** or are submitted with incorrect page-to-question mapping, the question will be **deducted partial points**. Make sure your name and PSU ID are legible on the first page of your assignment.

You are required to submit your assignments in typed format, please follow the latex/doc template (which can be found on Canvas) for the homework submission. Furthermore, please note that no handwritten submissions (of any form on paper or digital) will be accepted.

**(Kindly refer to the syllabus for late submission and academic integration policies.)

Assignment Specific Instructions:

1. The sample examples provided in the questions below are just for your reference and do not cover every possible scenario your solution should cover. Therefore, it is advised you think through all the corner cases before finalizing your answer.
2. Students are expected to answer the questions in a way that shows their understanding of the concepts rather than just mentioning the answers. The rubric does contain partial points to encourage brief conceptual explanations.

Problem 1: Lexical Analysis [10 pts]

1. Explain token, pattern, lexeme, and symbol table in the context of lexical analysis. [4 pts]
2. Explain the purpose of "lookahead" in lexical analysis. How it resolves ambiguity in dividing strings into tokens? [2 pts]
3. What are some common methods for recovering from lexical errors, and how do they work? [4 pts]

Solution

1. (a) A token is a pair consisting of a name and an attribute (e.g., $\langle \text{Ident}, \text{ip} \rangle$, $\langle \text{Operator}, \langle \rangle \rangle$).
   (b) A pattern describes the set of character strings (lexemes) that can be associated with a token (e.g., a string of letters and digits starting with a letter).
   (c) A lexeme is the actual sequence of characters in the source program that matches the pattern for a token (e.g., "`ip`", "`<`", "`>`" in `while (ip < z)`).
   (d) The symbol table is used to store information about identifiers (and sometimes numbers) found in the source program. During lexical analysis, identifiers are often entered into the symbol table, which can later be referenced by the parser or semantic analyzer.
2. "Lookahead" in lexical analysis means checking the next character(s) before finalizing a token. It prevents ambiguity when two tokens start in the same way. For example, telling apart `=` from `==`. By using lookahead, the lexer can split the input string into tokens accurately.
3. Common methods for recovering from lexical errors include:
   (a) **Panic Mode:** Deleting successive characters until a valid token is found.
   (b) **Single Character Deletion:** Deleting one character from the remaining input.
   (c) **Single Character Insertion:** Inserting one character into the remaining input.
   (d) **Replacement/Transposition:** Replacing or transposing characters to form a valid token.

Problem 2: Finite Automata I [6 pts]

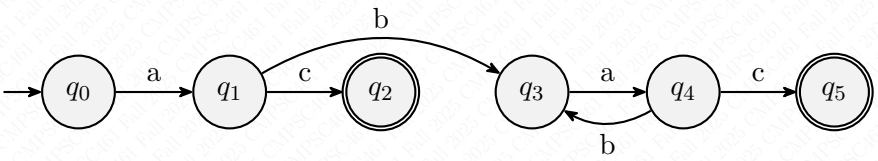
1. What is the transition table in Deterministic Finite Automata (DFA) and in Non-Deterministic Finite Automata (NFA)? What's the difference between them? [2 pts]

2. Given the regular expression "$(ab) * ac$" over the alphabet $\Sigma = \{a, b, c\}$, draw a Deterministic Finite Automata (DFA) for the same. Neatly show transitions for each state. [4 pts]

Solution

1. The transition table formally defines the behavior of the DFA and NFA. For **each state** and **each input symbol**, the table specifies what the next state is. The key difference between transition tables of a DFA and a NFA is that the next state specified by transition table of a DFA is unique (only one), while there can be multiple next states for the same current state and input symbol in the transition table of a NFA.

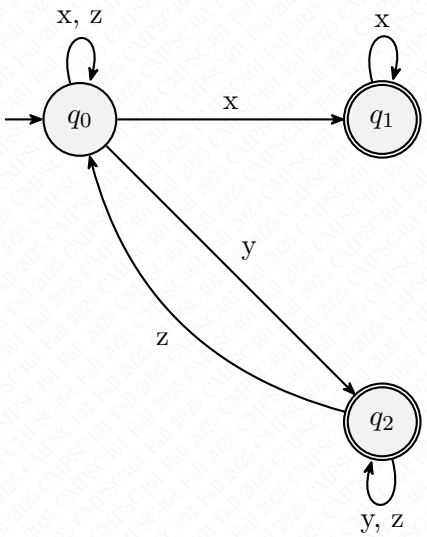
2.



Problem 3: Finite Automata II [9 pts]

For the given nondeterministic finite automata (NFA) over the alphabet $\Sigma = \{x, y, z\}$, construct a deterministic finite automata (DFA) using subset construction method. Draw 2 transition tables with each state and each symbol for (1) the given NFA and (2) the derived DFA. Also, write down what states will be the accepting/final states in the derived DFA.

NFA



Solution

NFA table			
state	**x**	**y**	**z**
q_0	q_0q_1	q_2	q_0
q_1	q_1	-	-
q_2	-	q_2	q_0q_2

DFA table			
state	**x**	**y**	**z**
q_0	q_0q_1	q_2	q_0
q_0q_1	q_0q_1	q_2	q_0
q_1	q_1	-	-
q_2	-	q_2	q_0q_2
q_0q_2	q_0q_1	q_2	q_0q_2

The accepting states are $\{q_0q_1, q_1, q_2, q_0q_2\}$.

(The state q_1 in DFA table can be omitted as it's not reachable from the start state q_0 in the derived DFA).

Problem 4: Regular Expression I [5 pts]

Write down the set of strings recognized by the following regular expressions. The alphabet is lower-case English letters (i.e. a to z). If the set is **finite**, write down all elements in the set. If the set is **infinite**, write down the first 8 shortest elements. (If there are multiple shortest elements with the same length, any of them are acceptable.)

1. `(ab|ac)*xy|x*`
2. `(ab|cd)+x`

Solution

1. {ϵ, x, xx, xy, xxx, xxxx, abxy, acxy, ...}
2. {abx, cdx, ababx, abcdx, cdabx, cdcdx, abababx, ababcdx, ...}

The solution to question 2 is not unique, and many other strings in length of 7 are also acceptable.

Problem 5: Regular Expression II [3 pts]

Construct a regular expression that matches a string adhering to the specified formatting rules:

1. The characters in the string include lowercase letters and digits.
2. It starts with at least 2 lowercase letter.
3. Each digit in the string must be followed by at least 2 lowercase letters.

NOTE: For the notation in this question, you should use `[0-9]` for representing all digits, `[a-z]` for all lowercase letters. You may also use concatenation, alternation (`|`), closures (`*`, `+`), and parentheses to construct the regular expression. Please follow the notation requirements.

Solution ***(Regex questions can have multiple solutions, and any solutions meeting the requirements are correct.)*

```
[a-z][a-z]([a-z]|[0-9][a-z][a-z])*
```

Problem 6: Regular Expression III $[3 * 4 = 12$ pts$]$

1. Create a regular expression that matches any occurrence of a date in the format MM-DD-YYYY, where the year ranges from 2025 to 2999.

   NOTE: For simplicity, assume that each month has 31 days. Specifically, months should be represented as ranging from "01" to "12", and days should range from "01" to "31".

2. Construct a regular expression that matches all email addresses adhering to the following criteria:

   (a) The email address must start with one or more alphanumeric characters.

   (b) It can include dots (.) or dashes (-) in the middle, but these characters must not be consecutive or at the beginning or end of the local part of the email.

   (c) The email must end with a single domain segment that contains 3 to 6 letters after a dot, reflecting standard email domain formats.

   (d) The '@' symbol must appear exactly once in between the local and domain parts.

   (e) The domain part between '@' and the dot can be any number of alphanumeric characters but it should appear at least once.

   Examples of valid email addresses:
   nittany.lions.club@psu.edu
   nittany@psu.edu
   beaver-stadium@penn.state
   Examples of invalid email addresses:
   nittany..lions@psu.edu (consecutive dots)
   -nittany@psu.edu (dash at the start)
   nittany.lions@psu.e (single letter in the domain)

   NOTE:

   (a) **Period (.) inside Character Classes:**
   Acts as a literal character, not a wildcard. No need to escape it.
   Example: [.] matches a literal period.

   (b) **Dash (-) inside Character Classes:**
   Indicates a range between characters unless it's at the start or end or placed such that it can't form a range.
   Example: [-.] matches a literal dash or period. No escape needed for the dash or dot here.

   (c) **Period (.) in General Context:**
   Unlike within character classes, when a period appears outside of character classes in a regular expression, it represents any single character. To use a period as a literal dot, you must escape it with a backslash. This is essential in email addresses to specify the exact placement of dot characters in domain names.
   Example: To match an actual period in an email domain, use `\.` as in `example@example\.com`

3. In a software development project involving multiple programming languages, maintaining a consistent and clear commenting style is crucial. Regular expressions are used to ensure that all comments in the codebase adhere to a specific format, facilitating easier code reviews and maintenance.

   Question: Develop a regular expression that identifies and validates the formatting of comments within the source code. The comments may appear anywhere within a line and could be embedded in larger text blocks or code snippets.

   Comment Format Rules:

   Comment Start: Each comment begins with a double slash // followed by a specific 'tag' which indicates the purpose of the comment. The tags are:

   'T' for Task;

   'B' for Bug;

   'D' for Documentation.

   Tag Structure: Immediately following the tag, there should be exactly **one colon :** and then a **space**.

   Comment Content: After the space, the comment text begins. The comment text can be any character but it should not be empty.

   Examples of Valid Comments Embedded in Code:

   //T: Implement the login function

   //B: Fix the off-by-one error in the array indexing

   //D: This section handles user input validation

   Task:

   Construct a regular expression that matches this comment structure regardless of its position within a line. Explain each part of your regex, highlighting how it captures the required format.

Solution ***(We understand that regex questions can have multiple solutions, students will be rewarded points if their solution is different from the one mentioned below but meets the requirements.)*

1. $(0[1-9] \mid 1[0-2]) - (0[1-9] \mid [1-2][0-9] \mid 3[0-1]) - (202[5-9] \mid 20[3-9][0-9] \mid 2[1-9][0-9][0-9])$
2. `[A-Za-z0-9]+([.-][A-Za-z0-9]+)*@[A-Za-z0-9]+(\.[A-Za-z]{3,6})`
3. `\/\/[TBD]: .+`
   or
   `//[TBD]: .+`
   Note: if comment content cannot be "empty" means it cannot be spaces only, this answer is also correct: `\/\/[TBD]: .*\`$S+.*$

Problem 7: Regular Expression IV [5 pts]

Problem statement: You are provided with log entries that records network traffic between servers. Each log entry contains a timestamp, server IP addresses, and packet sizes, but the format varies slightly from typical logs.

Log Entry Format:

1. Standard Format:[HH:MM:SS] Message about the packet size XXKB to server IP-ADDRESS
2. Example Log Entries:

   [14:23:01] Successfully transferred 56KB to server 192.168.0.12

   [14:23:03] Successfully transferred 4KB to server 192.168.0.1

   [14:23:04] Successfully transferred 12KB to server 192.168.0.5

   [14:23:07] Successfully transferred 0KB to server 192.168.0.237

Task: Develop a regular expression that extracts:

1. **Timestamp**: The datetime stamp formatted as HH:MM:SS, with HH ranging from 00 to 23, and MM and SS from 00 to 59.
2. **Packet Size Information**: The amount of data transferred, noted in KB. The amount can be any value(digit) in any length but not empty.
3. **IP Address**: The IPv4 address, specifically within the 192.168.0.x range.

NOTE:

1. IP Address: Should specifically match the private IP range 192.168.0.x, where x ranges from **0 to 255**, i.e. the IP address range 192.168.0.0/24. There could be other log entries out of this IP range, and your regular expression should only match log entries with such private IP range.
2. In the context of regular expressions, the dot (.) character is a wildcard that matches any single character. To match a literal dot in your data, such as in IP addresses (e.g., 192.168.0.1), you must use a backslash \ to escape the dot: `192\.168\.0\.1`
3. Similarly, you should use a backslash to escape keywords or operators in regular expression when you want to match them as literal characters in your text. Some other examples are `[, ], (, ), |, *, +`.

Objective: Construct a regular expression that accurately captures the timestamp, packet size, and IP address from each log entry. Provide a detailed explanation of how your regex is structured and functions.

Solution ***(We understand that regex questions can have multiple solutions, students will be rewarded points if their solution is different from the one mentioned below but meets the requirements.)*

```
\[(([01][0-9]|2[0-3]):([0-5][0-9]):([0-5][0-9]))\]\D*([0-9]+)KB\D
  *(192\.168\.0\.([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]))\n
```

where

1. ```
   \[(([01][0-9]|2[0-3]):([0-5][0-9]):([0-5][0-9]))\]
   ```

   matches the timestamp

2. $\backslash D*$ matches any number of non-digit characters
3. ```
   ([0-9]+)KB
   ```

   matches the packet size
4. ```
   (192\.168\.0\.([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]))\n
   ```

   matches the IP address at the end of the line.
   ```