CMPSC 461, Fall 2025
Programming Language Concepts
Instructor: Dr. Suman Saha

Conflict Midterm-I Solutions

October 01, 2025 || 7:00 PM - 9:00 PM

P#	Possible	Score
1	20	
2	15	
3	10	
4	10	
5	10	
6	15	
Bonus	20	
Total	100	

This exam is a closed book and closed notes. You may not use any additional materials during the exam. All electronic devices must be put away. You may have nothing on your desk except this exam, and writing instruments. Make sure you use a pencil/pen of high contrast and your name, user ID, and student ID are clearly visible and legible in the space provided. If you fail to do so Gradescope might not process it correctly. Also please don't forget to mention your user ID on each page (in the space provided). **If you are using the extra sheet or paper to answer your question make sure you mention a note about it under that particular question.**
This exam consists of 6 questions + 1 bonus question (in total 7 questions). The bonus question is 20 marks making exam total of 100 points (80 points + 20 points for bonus questions).
How bonus question works: The maximum points you can score in this whole exam is 80 points. If you get all the first 6 six questions correct & you will score 80 points out of 100 points, you will be getting 80 marks (100 %). If you do all 7 questions correctly and score 100 out of 100 marks, you will still be getting 80 marks (100 %). If the total of your attempt in the whole exam is less than 80 points, for example: say 60 pts out of 100 pts, your final score is 60 out of 80 (75%).

Read each question carefully and use your time judiciously. The problems are not ordered by their difficulty. Also, the examples provided with the questions (if any) are not exhaustive.

Please **refer to the last page (appendix)** to refer to the rules on how to approach and solve problems related to first and follow.

User ID (Eg: abc2345): _____

Problem 1: Regular Expressions [20 pts]

(a) Construct a regular expression to match a U.S. phone number: [7 pts]

- It should start with optional country code, Accept either `+1` or nothing
- Additionally, the area code (first 3 digits) cannot begin with 0 or 1
- Phone number is of the format 3 digits - 3 digits - 4 digits
- Separators are hyphens only

Valid:
`+1-234-567-8901`
`234-567-8901`

Invalid:
`+1-034-567-8901` – area code begins with 0
`123-4567-8901` – incorrect grouping of digits

(b) Construct a regular expression over $\Sigma = \{0, 1\}$ that generates all binary strings that do not contain two consecutive `1`. [7 pts]

Valid:
`0`
`1010`
`01010`

Invalid:
`11` – contains consecutive 1s
`1100` – starts with consecutive 1s
`1011` – ends with consecutive 1s

(c) Construct a regular expression for floating-point constants of the form: [6 pts]

- Optional sign (+ or -),
- One or more digits,
- Decimal point `.`,
- One or more digits.

Valid:
`3.14`
`-0.99`
`+123.456`

Invalid:
`.5` – missing integer part before decimal
`123.` – missing digits after decimal

Solution

a. `(\+1-)?[2-9][0-9]{2}-[0-9]{3}-[0-9]{4}`

b. `(0|10)*1?`

c. `(\+|-)?[0-9]+\.[0-9]+`

Problem 2: Finite Automata [15 pts]

1. Design a DFA recognizing the string over the alphabet $\{1, 2\}$ where the string has odd number of 2s and all 1s (if any) in the string are followed by 2s. [5 pts]

   For example, accepted strings: 2, 222, 1222, 112; rejected strings: ϵ, 22, 2222, 21, 21122.

   Draw the graph clearly with states and transitions. Mark the start state and all accepting state(s). No explanation required. No transition table required.

2. Design a NFA recognizing the string over the alphabet $\{a, b, c\}$ where all bs are in substrings abc. [5 pts]

   For example, accepted strings: ϵ, abc, $aabc$, $abcacc$, $abcabc$; rejected strings: $abbbc$, $abcb$, cba.

   Draw the graph clearly with states and transitions. Mark the start state and all accepting state(s). No explanation required. No transition table required.

3. Convert the following NFA transition table into equivalent DFA transition table using subset construction.

   [5 pts]

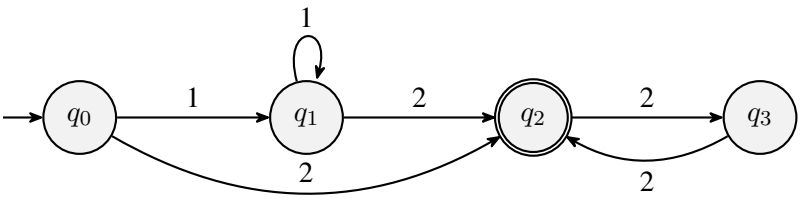
   Only DFA transition table is required. No Automata graph required. No explanation required. Also, mention the start state and all accepting state(s) in the derived DFA. You can omit the row if no transitions going out from the state or it cannot be reached from the start state.

NFA State	x	y	z
q_0	$\{q_2\}$	$\{q_1, q_2\}$	ϕ
q_1	$\{q_0\}$	$\{q_2\}$	ϕ
q_2	ϕ	$\{q_1\}$	$\{q_0, q_2\}$

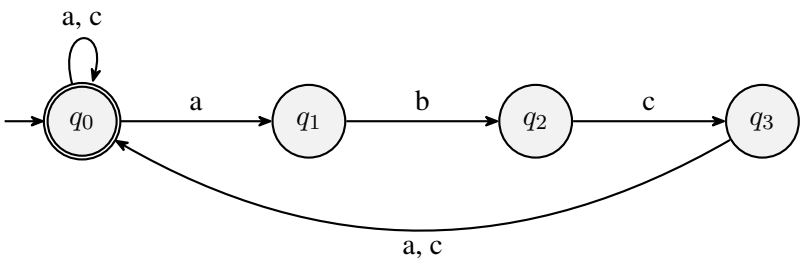
   The start state is q_0, and the accepting states are q_0 and q_1.

Solution

1. One correct DFA is shown below:



2. One correct NFA is shown below:



3. The transition table is shown below:

DFA State	x	y	z
q_0	q_2	q_1q_2	ϵ
q_1	q_0	q_2	ϵ
q_2	ϵ	q_1	q_0q_2
q_0q_2	q_2	q_1q_2	q_0q_2
q_1q_2	q_0	q_1q_2	q_0q_2

The start state is q_0, and the accepting states are q_0, q_1, q_0q_2, and q_1q_2.

——— *Extra page for Q2* ———

Problem 3: Grammar I [10 pts]
Consider the grammar:

1. $L = \{w \in \{x, y\}^* \mid w$ has at least two xs$\}$. Some valid strings are xx, xyx, yxx, xxyy, etc.
   (a) Provide CFG for the language mentioned above. [4 pts]
   (b) Using grammar generated in part a), provide the left-most derivation for string yxxy. [3 pts]
2. $L = \{(ab)^n c^n \mid n \geq 1\}$. Some valid strings in the language are abc, ababcc, abababccc, etc.
   (a) Provide a CFG for the language. [3 pts]

Solution

1. (a)

$$\begin{aligned} S &\rightarrow AxAxA \\ A &\rightarrow xA \mid yA \mid \epsilon \end{aligned}$$

   (b) $S \Rightarrow AxAxA \Rightarrow yAxAxA \Rightarrow y\epsilon xAxA \Rightarrow yxAxA \Rightarrow yx\epsilon xA \Rightarrow yxxA \Rightarrow yxxyA \Rightarrow yxxy\epsilon = yxxy$

2.

$$S \rightarrow abSc \mid abc$$

Problem 4: Grammar II [10 pts]

Convert each of the following regular expressions into an equivalent CFG using BNF notation. Your grammar should be unambiguous and use a minimal number of production rules. To abbreviate ranges of consecutive characters, you can use the syntax $a \mid \cdots \mid z$.

1. `[2-9][0-9][0-9]-([2-9][0-9][0-9]-)?[0-9]{4}` [5 pts]

   Matches strings like `200-1234` and `800-555-0000` and `999-999-9999` but not `199-9999` or `123-456-789-0000` or `200-100-1234`

2. `(0|[1-9][0-9]*)(\.[0-9]*[1-9])?e(0|[1-9][0-9]*)` [5 pts]

   Matches strings like `0e0` and `14.5e-5` and `1.234e10` but not `01e0` or `1e01` or `1.2000e3`

Solution

1.
$$
\begin{aligned}
S &\to D_1\ D_0\ D_0 - N_0\ D_0\ D_0\ D_0\ D_0 \\
N_0 &\to D_1\ D_0\ D_0 - \mid \epsilon \\
D_0 &\to 0 \mid \cdots \mid 9 \\
D_1 &\to 2 \mid \cdots \mid 9
\end{aligned}
$$

2.
$$
\begin{aligned}
S &\to N_0\ N_1\ \texttt{e}\ N_0 \\
N_0 &\to \texttt{0} \mid D_1\ D_n \\
N_1 &\to \texttt{.}\ D_n\ D_1 \mid \epsilon \\
D_n &\to D_0\ D_n \mid \epsilon \\
D_0 &\to \texttt{0} \mid \cdots \mid \texttt{9} \\
D_1 &\to \texttt{1} \mid \cdots \mid \texttt{9}
\end{aligned}
$$

——— *Extra page for Q4* ———

Problem 5: Name, Scope, and Binding I [10 pts]

Consider the following pseudo-code:

```
int z = 99;

void reporter() {
    print(z);
}

void recur(int n) {
    if (n > 0) {
        int z = n * 5;
        recur(n - 1);
    }

    if (n == 2) {
        reporter();
    }
}

void main() {
    recur(3);
    print(z);
}
```

(a) Draw a diagram of the **runtime stack** at the exact moment the `reporter()` function is called. For each activation frame, clearly label its **static link** and **dynamic link**. [4 pts]

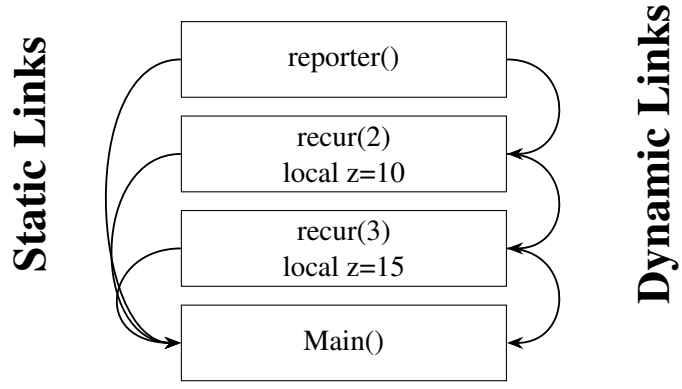
(b) What is the complete output of this program, assuming the language uses: [6 pts]

i. Static scoping?

ii. Dynamic scoping?

Solution

(a) **Runtime Stack at `reporter()` call**



(b) **Program Outputs**

i. Static scoping: `99  99`

ii. Dynamic scoping: `10  99`

——— *Extra page for Q5* ———

Problem 6: Name, Scope and Binding II [15 pts]

Consider the following pseudo-code, assuming dynamic scoping rules:

```
1  typedef void (*FuncPtr)();
2
3  void action() {
4      print(y);
5  }
6
7  FuncPtr maker(int n) {
8      int y = n * 5;
9      return action;
10 }
11
12 void driver(FuncPtr F) {
13     int y = 99;
14     F();
15 }
16
17 void main() {
18     FuncPtr f = maker(7);
19     driver(f);
20 }
```

(A) Draw the **symbol tables** for the scopes `global`, `action`, `maker`, and `driver`. [5 pts]

(B) What is the output with **shallow binding**? Justify via the call chain. [5 pts]

(C) What is the output with **deep binding**? Explain how a closure preserves the captured environment. [5 pts]

Solution

(A) Symbol Tables

global

name	kind
action	func
maker	func
driver	func
main	func

action

name	kind

maker

name	kind
n	para
y	id

driver

name	kind
F	para (func)
y	id

(B) Shallow binding output: `99`.

Explanation: Shallow binding resolves variables from the **call site**. The function ‘action‘ is called from within ‘driver‘. The call stack is ‘main - > driver -> action‘. The search for ‘y‘ checks ‘action‘ (none), and then its caller, ‘driver‘, where it finds the local ‘y = 99‘.

(C) Deep binding output: `35`.

Explanation: Deep binding uses a **closure** to capture the environment where the function reference was **created**. The reference to ‘action‘ is created when ‘maker(7)‘ is called. At this point, a closure is formed, bundling ‘action‘ with the environment of ‘maker‘, where ‘y = 7 * 5 = 35‘. Even though ‘action‘ is called later from inside ‘driver‘, it uses this preserved environment from its closure to find ‘y‘, printing ‘35‘.

——— *Extra page for Q6* ———

Problem 7: Bonus [20 pts]

1. Consider the grammar:

$$\begin{aligned} S &\rightarrow A\,C \\ A &\rightarrow a \mid \epsilon \\ C &\rightarrow c\,C \mid d \end{aligned}$$

   (a) Compute the **FIRST** and **FOLLOW** sets for S, A, C. [5 pts]

   (b) Construct the **LL(1) parsing table** and state whether the grammar is LL(1). [7 pts]

2. In Python, functions are declared using the `def` keyword, a function name, and a pair of parentheses containing a *parameter list*. This list contains zero or more variable names followed by zero or more variables with default values. This parameter list can be expressed using the following CFG: [8 pts]

$$\begin{aligned} S &\rightarrow X, Y \mid X \mid Y \mid \epsilon \\ X &\rightarrow X, X \mid \langle var \rangle \\ Y &\rightarrow Y, Y \mid \langle var \rangle = \langle expr \rangle \end{aligned}$$

   This grammar matches inputs like $var_1, var_2 = expr_2$ but rejects inputs like $var_1 = expr_1, var_2$.

   Show that the grammar is ambiguous using **right-most derivation** and the smallest possible example. Make sure to clearly indicate which rule is being applied at each step.

Solution

1. (a) FIRST sets: FIRST$(A) = \{a, \epsilon\}$, FIRST$(C) = \{c, d\}$, FIRST$(S) = \{a, c, d\}$.
   FOLLOW sets: FOLLOW$(S) = \{\$\}$, FOLLOW$(A) = \{c, d\}$, FOLLOW$(C) = \{\$\}$.

   (b) Parsing table:

Non-terminal	a	c	d	$
S	$S \rightarrow AC$	$S \rightarrow AC$	$S \rightarrow AC$	
A	$A \rightarrow a$	$A \rightarrow \epsilon$	$A \rightarrow \epsilon$	
C		$C \rightarrow cC$	$C \rightarrow d$	

   No conflicts $\Rightarrow$ Grammar is LL(1).

2. The grammar is ambiguous for three or more parameters of a single kind. $\langle var \rangle, \langle var \rangle, \langle var \rangle$ and $\langle var \rangle = \langle expr \rangle, \langle var \rangle = \langle expr \rangle, \langle var \rangle = \langle expr \rangle$ are functionally equivalent, so only the former is shown below:

$\underline{S}$	$S \to X$
$\underline{X}$	$X \to X, X$
$\underline{X}, X$	$X \to X, X$
$\underline{X}, X, X$	$X \to \langle var \rangle$
$\langle var \rangle, \underline{X}, X$	$X \to \langle var \rangle$
$\langle var \rangle, \langle var \rangle, \underline{X}$	$X \to \langle var \rangle$

(a) First Derivation

$\underline{S}$	$S \to X$
$\underline{X}$	$X \to X, X$
$\underline{X}, X$	$X \to \langle var \rangle$
$\langle var \rangle, \underline{X}$	$X \to X, X$
$\langle var \rangle, \underline{X}, X$	$X \to \langle var \rangle$
$\langle var \rangle, \langle var \rangle, \underline{X}$	$X \to \langle var \rangle$

(b) Second Derivation

——— *Extra page for Q7* ———

——— *Extra page if needed.* ———

——— *Extra page if needed.* ———

Appendix

First(C)

FIRST(C) for a grammar symbol C is the set of terminals that begin the strings derivable from C. Formally, it is defined as:

$$\text{First}(C) = \{t \mid C \to^* t\alpha\} \cup \{\varepsilon \mid C \to^* \varepsilon\}$$

Follow(C)

Follow(C) is defined to be the set of terminals that can appear immediately to the right of Non-Terminal C in some sentential form.

Rules:

- If C is the start symbol, then \$ $\in$ Follow(C).
- If $C \to AB$, then First$(B) \subseteq$ Follow(A) and Follow$(C) \subseteq$ Follow(B).
- Also, if $B \to^* \varepsilon$, then Follow$(C) \subseteq$ Follow(A).

Constructing LL(1) Parsing Table

Construct a parsing table T for CFG G using the following procedure:

For each production $A \to \alpha$ in G do:

1. For each terminal $t \in$ First(α) do
   - $T[A, t] = \alpha$
2. If $\varepsilon \in$ First(α), for each $t \in$ Follow(A) do
   - $T[A, t] = \alpha$
3. If $\varepsilon \in$ First(α) and \$ $\in$ Follow(A) do
   - $T[A, \$] = \alpha$