### Group 1
- Question 1.1
- Question 1.2
- Question 1.3
- Question 1.4

### Group 2
- Question 2.1
- Question 2.2
- Question 2.3

### Group 3
- Question 3.1
- Question 3.2

**Instructions:**

- Provide your answer in the box after each question. If you absolutely need extra space, use the backs of the pages, but try to avoid it. Keep your answers short and to the point.

- You are not allowed to use a cheat sheet.

- Read every question carefully before answering. In particular, do not waste time on a proof if none is asked for, and do not forget to provide one if it is required.

- Do not forget to write your banner number and name on the top of this page.

- This exam has 8 pages, including this title page. Notify me immediately if your copy has fewer than 8 pages.

- The total number of marks in this exam is 95.
Regular Languages and Finite Automata

**Question 1.1 (Definition)** 5 marks

A language is a set of strings over an alphabet. What are the conditions this set has to satisfy for the language to be regular? In other words, provide the set-theoretic definition of regular languages. Do not use their relationship to regular expressions and finite automata as a definition.

A language \( \mathcal{L} \subseteq \Sigma^* \) is regular if it is of one of the following forms:

- \( \mathcal{L} = \emptyset \).
- \( \mathcal{L} = \{ x \}, x \in \Sigma \).
- \( \mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2 \), where \( \mathcal{L}_1 \) and \( \mathcal{L}_2 \) are regular languages.
- \( \mathcal{L} = \mathcal{L}_1 \cdot \mathcal{L}_2 = \{ x_1x_2 \mid x_1 \in \mathcal{L}_1 \text{ and } x_2 \in \mathcal{L}_2 \} \), where \( \mathcal{L}_1 \) and \( \mathcal{L}_2 \) are regular languages.
- \( \mathcal{L} = \mathcal{L}_1^* \), where \( \mathcal{L}_1 \) is a regular language and \( \mathcal{L}_1^* \) is the set of all strings that can be formed by concatenating any finite number of strings in \( \mathcal{L}_1 \).

**Question 1.2 (Finite automata)** 15 marks

In class we discussed that a language is regular if and only if it can be recognized by a finite automaton.

(a) Provide the formal definition of a deterministic finite automaton (DFA).

A DFA is a 5-tuple \( (Q, \Sigma, \delta, q_0, F) \), where

- \( Q \) is a set of states,
- \( \Sigma \) is an alphabet,
- \( q_0 \in Q \) is the start state,
- \( F \subseteq Q \) is a set of accepting states, and
- \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function.

(b) When does a DFA accept a string \( \sigma \)?

Given a DFA \( M = (Q, \Sigma, \delta, q_0, F) \), the string \( \sigma = x_1x_2 \ldots x_n \in \Sigma^* \) defines a sequence of states \( q_0, q_1, \ldots, q_n \), where \( q_i = \delta(q_{i-1}, x_i) \), for all \( 1 \leq i \leq n \). \( M \) accepts \( \sigma \) if and only if \( q_n \in F \).

(c) What are the two differences between a non-deterministic finite automaton (NFA) and a DFA?

- The transition function maps each state-character pair to a set of states and allows \( \epsilon \)-transitions: \( \delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q \).
- Using this modified transition function, every string \( \sigma = x_1x_2 \ldots x_n \in \Sigma^* \) defines a set of state sequences \( q_0q_1 \ldots q_m \) such that there are indices \( -1 = i_0 < i_1 < \cdots < i_n + 1 = m \) satisfying \( q_{i_j+1} \in \delta(q_{i_j}, x_{i_j}) \), for all \( 1 \leq j \leq n \), and \( q_k \in \delta(q_{k-1}, \epsilon) \), for all \( 1 \leq j \leq n + 1 \) and \( i_{j-1} < k \leq i_j \). The NFA accepts \( \sigma \) if and only if one of these sequences ends in an accepting state \( q_m \in F \).
Question 1.3 (Pumping Lemma) 5 marks

State the Pumping Lemma.

For every regular language $L$, there exists an integer $n_L$ with the following property: Every string $\sigma \in L$ of length at least $n_L$ can be divided into three substrings $\alpha$, $\beta$, and $\gamma$ such that $|\alpha\beta| \leq n_L$, $|\beta| > 0$, and, for all $k \geq 0$, the string $\alpha\beta^k\gamma$ is also in $L$.

Question 1.4 (Deciding whether a language is regular) 15 marks

For each of the following languages, state whether it is regular or not. If it is, prove your claim by providing a graphical representation of a (deterministic or non-deterministic) finite automaton that recognizes it. If it is not, prove your claim using the Pumping Lemma.

(a) The language $L$ of all binary strings with at least one 1 in each substring of length three. In other words, the string 10100111001 belongs to this language, the string 10100011 does not.

This language is regular. Here is a DFA that recognizes it:

(b) The language $L$ of all binary strings that are palindromes, that is, strings that are of the form $x_1x_2 \ldots x_nx_n \ldots x_1$ or $x_1x_2 \ldots x_{n-1}x_nx_{n-1} \ldots x_1$.

This language is not regular. Assume it is, and let $n_L$ be the integer associated with $L$ by the Pumping Lemma. Now let $\sigma = 0^n 10^n \in L$. By the Pumping Lemma, we can divide $\sigma$ into three substrings $\alpha$, $\beta$, and $\gamma$ such that $|\alpha\beta| \leq n_L$, $|\beta| > 0$, and $\alpha\beta^2\gamma \in L$. Since the first $n_L$ letters in $\sigma$ are 0s, we have $\beta = 0^k$, for some $k > 0$, and, hence, $\sigma' = \alpha\beta^2\gamma = 0^n 10^n$. This is a contradiction because $\sigma' \in L$, but it is not a palindrome.

(c) The language $L$ of all binary strings whose second and second-last letters are 1s. For example, the strings 010011 and 11 belong to this language, the strings 1011 and 0101101 do not.

This language is regular. Here is a DFA that recognizes it:
Question 2.1 (Grammars) 10 marks

(a) Define formally what a context-free grammar is.

A context-free grammar is a quadruple \( (V, \Sigma, S, P) \), where
- \( V \) is a set of non-terminals,
- \( \Sigma \) is a set of terminals (the alphabet),
- \( S \in V \) is the start symbol, and
- \( P \) is a set of productions: \( P \subseteq V \times (V \cup \Sigma)^* \). The elements \( (X, \sigma) \in P \) are usually written as \( X \rightarrow \sigma \).

(b) Define formally when a grammar is LL(\( k \)).

Informally, \( k \) letters of look-ahead suffice to decide which production to apply for the first non-terminal in the current sentential form.

Formally, every sentential form \( X\beta \), where \( X \in V \), and every string \( \sigma \in \Sigma^* \) satisfy the following two conditions: (1) If \( X\beta \Rightarrow^* \sigma \), then all leftmost derivations \( X\beta \Rightarrow^* \sigma' \), where \( \sigma \) and \( \sigma' \) share the same prefix of \( k \) letters, start with the same production for \( X \). (2) If \( X\beta \not\Rightarrow^* \sigma \), then the first \( k \) letters of \( \sigma \) suffice to prove this.
Question 2.2 (Recognizing LL(1) grammars) 15 marks

Is the following grammar LL(1)? \((T\) is the start symbol.) Prove that your answer is correct.

\[
\begin{align*}
T & \rightarrow AB \\
A & \rightarrow PQ \\
A & \rightarrow CB \\
P & \rightarrow pP \\
P & \rightarrow \epsilon \\
Q & \rightarrow qQ \\
Q & \rightarrow \epsilon \\
P & \rightarrow \epsilon \\
B & \rightarrow bB \\
B & \rightarrow e \\
C & \rightarrow cC \\
C & \rightarrow f
\end{align*}
\]

A grammar is LL(1) if all rules with the same left-hand side have disjoint PREDICT sets. For this grammar, the FIRST, FOLLOW, and PREDICT sets are as follows:

<table>
<thead>
<tr>
<th>X</th>
<th>FIRST(X)</th>
<th>X</th>
<th>FOLLOW(X)</th>
<th>R</th>
<th>PREDICT(R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{p}</td>
<td>T</td>
<td>{\epsilon}</td>
<td>T \rightarrow AB</td>
<td>{c,f,p,q,b,e}</td>
</tr>
<tr>
<td>q</td>
<td>{q}</td>
<td>A</td>
<td>{b,e}</td>
<td>A \rightarrow PQ</td>
<td>{p,q,b,e}</td>
</tr>
<tr>
<td>b</td>
<td>{b}</td>
<td>P</td>
<td>{b,e,q}</td>
<td>A \rightarrow CB</td>
<td>{c,f}</td>
</tr>
<tr>
<td>c</td>
<td>{c}</td>
<td>Q</td>
<td>{b,e}</td>
<td>P \rightarrow pP</td>
<td>{p}</td>
</tr>
<tr>
<td>e</td>
<td>{e}</td>
<td>B</td>
<td>{b,e,\epsilon}</td>
<td>P \rightarrow \epsilon</td>
<td>{b,e,\epsilon}</td>
</tr>
<tr>
<td>f</td>
<td>{f}</td>
<td>C</td>
<td>{b,e}</td>
<td>Q \rightarrow qQ</td>
<td>{q}</td>
</tr>
<tr>
<td>T</td>
<td>{c,f,p,q,b,e}</td>
<td></td>
<td></td>
<td>Q \rightarrow \epsilon</td>
<td>{b,e}</td>
</tr>
<tr>
<td>A</td>
<td>{c,f,p,q,\epsilon}</td>
<td></td>
<td></td>
<td>B \rightarrow bB</td>
<td>{b}</td>
</tr>
<tr>
<td>P</td>
<td>{p,\epsilon}</td>
<td></td>
<td></td>
<td>B \rightarrow e</td>
<td>{e}</td>
</tr>
<tr>
<td>Q</td>
<td>{q,\epsilon}</td>
<td></td>
<td></td>
<td>C \rightarrow cC</td>
<td>{c}</td>
</tr>
<tr>
<td>B</td>
<td>{b,\epsilon}</td>
<td></td>
<td></td>
<td>C \rightarrow f</td>
<td>{f}</td>
</tr>
</tbody>
</table>

Since the predict sets of all rules with the same left-hand side are disjoint, the grammar is LL(1).
Question 2.3 (Parsing LL(1) languages) 10 marks

Provide the pseudo-code of a recursive-descent parser that accepts all strings in the language defined by the grammar in Question 2.2 and rejects all strings not in this language.

```plaintext
procedure parseT
    parseA;
    parseB;
    if no input symbols left
        then accept the input;
        else reject the input;

procedure parseA
    case next input symbol of
        p, q, b, e → begin parseP; parseQ; end;
        c, f → begin parseC; parseB; end;
        otherwise → reject the input

procedure parseP
    case next input symbol of
        p → begin match(p); parseP; end;
        b, e, q → /* Do nothing */;
        otherwise → reject the input

procedure parseQ
    case next input symbol of
        q → begin match(q); parseQ; end;
        b, e → /* Do nothing */;
        otherwise → reject the input

procedure parseB
    case next input symbol of
        b → begin match(b); parseB; end;
        e → match(e);
        otherwise → reject the input

procedure parseC
    case next input symbol of
        C → begin match(c); parseC; end;
        f → match(f);
        otherwise → reject the input

procedure match(x)
    if next input symbol = x
        then consume this input symbol
        else reject the input
```
Question 3.1 (Is it lexical, syntactic or semantic?) 10 marks

For each of the following conditions on a valid C program, state whether it is a lexical, syntactic or semantic constraint.

(a) An identifier is a non-empty sequence of letters, digits and underscores not starting with a letter and not equal to one of the keywords of the language.

Lexical

(b) Statements need to be separated by semicolons.

Syntactic

(c) A variable has to be declared before it is used.

Semantic

(d) Every ‘{’ has to be matched by a ‘}’ and vice versa.

Syntactic

(e) A string is a sequence of characters starting and ending with a double quote and not containing any unescaped double quotes.

Lexical

(f) The dereferencing operator ‘*’ expects a pointer as its argument.

Semantic

(g) The number of arguments provided to a function call matches the number of formal parameters in the function definition.

Semantic
Question 3.2 (Attribute grammars) 10 marks

Provide a context-free grammar that defines the language of all valid arithmetic expressions using multiplication, division, addition, subtraction, and parenthesization. The only operands you need to recognize are numbers, and you can treat numbers as indivisible tokens that have already been recognized by a scanner. Your grammar does not necessarily have to be LL(1). (The second part of this question is substantially easier if it is not.)

Augment your grammar so that it becomes an attribute grammar that computes the value of every expression that conforms to the grammar. The calculation of these values should respect the standard precedence rules for addition, subtraction, multiplication, and division and should respect left-associativity of these operations.

\[
\begin{align*}
\text{Expr} & \rightarrow \text{Term} \\
\text{Expr} & \rightarrow \text{Expr} + \text{Term} \\
\text{Expr} & \rightarrow \text{Expr} - \text{Term} \\
\text{Term} & \rightarrow \text{Factor} \\
\text{Term} & \rightarrow \text{Term} \times \text{Factor} \\
\text{Term} & \rightarrow \text{Term} / \text{Factor} \\
\text{Factor} & \rightarrow \text{number} \\
\text{Factor} & \rightarrow ( \text{Expr} )
\end{align*}
\]

\[
\begin{align*}
\text{Expr}.\text{val} & := \text{Term}.\text{val} \\
\text{Expr}_1.\text{val} & := \text{Expr}_2.\text{val} + \text{Term}.\text{val} \\
\text{Expr}_1.\text{val} & := \text{Expr}_2.\text{val} - \text{Term}.\text{val} \\
\text{Term}.\text{val} & := \text{Factor}.\text{val} \\
\text{Term}_1.\text{val} & := \text{Term}_2.\text{val} \times \text{Factor}.\text{val} \\
\text{Term}_1.\text{val} & := \text{Term}_2.\text{val} / \text{Factor}.\text{val} \\
\text{Factor}.\text{val} & := \text{number}.\text{val} \\
\text{Factor}.\text{val} & := \text{Expr}.\text{val}
\end{align*}
\]