State Final Examination (Computer Science Sample Questions)

Fall 2019

1 Automata and grammars (3 points)

1. Give two definitions (or characterizations) when a language L over an alphabet Σ is *regular*.
2. Consider the following two grammars G_1 and G_2 over an alphabet (terminals) $\Sigma_1 = \{a, b, \neg, \vee, \wedge, (,)\}$ resp. $\Sigma_2 = \{a, b, \neg, \vee, \wedge\}$ with a single (starting) variable S_1 resp. S_2 and rules

$$\begin{aligned} G_1\colon &\quad S_1 \to a \mid b \mid (\neg S_1) \mid (S_1 \vee S_1) \mid (S_1 \wedge S_1), \\ G_2\colon &\quad S_2 \to a \mid b \mid \neg S_2 \mid S_2 \vee S_2 \mid S_2 \wedge S_2. \end{aligned}$$

   Determine whether G_1 resp. G_2 generates a regular language. Justify your claims.

2 Breadth First Search (3 points)

1. Describe data structure of list of neighbours for representing an undirected graph $G = (V, E)$.
2. Write a pseudocode of the breadth-first search algorithm for searching an undirected graph $G = (V, E)$ starting at a vertex $s \in V$.
3. Determine the time complexity of the breadth-first search algorithm assuming the input graph is represented by a list of neighbours.
4. Define the notion of a bipartite graph and describe how breadth-first search can be used to determine whether a given graph is bipartite.

3 ER diagrams and SQL queries (3 points)

Design a (very) simplified airport system for storing aircraft arrival data. For each airport, we want to store its name and unique code. In addition, we want to be able to record dates and times of aircraft arrivals, their manufacturers, types and registrations (unique aircraft codes). Ignore rare situations like changing aircraft registrations, changing airport codes or names or merging of manufacturers.

1. Draw an ER diagram for this case.
2. Transform the ER diagram to a relational schema (UML class diagram or SQL DDL) compliant with 3NF. Highlight keys and foreign keys.
3. Write an SQL query that returns the aircraft manufacturer ranking for a given airport and a given day, ordered by the number of arriving aircraft of that manufacturer.

4 OO design of a GUI framework (3 points)

Your task is to design a simple windowing framework in the C++, C# or Java language. The framework will be used to create applications with a GUI. Your implementation should adhere to standard principles of extensible and maintainable

object-oriented design – however, focus only on the required properties of the framework, and do not create overly complex design.

1. Your framework should have the following basic properties:
   - It should support the concept of a window. A window can contain any number of controls.
   - Every control contained in a window is located on some X and Y coordinates and has its width and height (these four values are grouped into a `Rectangle` structure that has been already implemented for you; the structure also has a `bool IsPointInside(Point p)` method that tests if a point `p` is located inside the rectangle).
   - Your implementation should include a button control (Button), and a user-editable text field (TextBox) control. Both controls should have a `Text` property representing a text string they are displaying (do not implement the actual logic of displaying the text).
2. Every control has a `void HandleMouseClick(Point p)` method that should be called if a mouse click is detected visually over the control (the method will contain the behavior logic specific to every control type; however, you are not required to implement it; for simplicity expect the controls do not overlap). To get information about a mouse click location from the operating system you should use a prepared `OS` class (the point it returns is relative to the same coordinate system that is used to define the location of controls):

   ```
   enum UserEvent { MouseClick, /* ... */ };
   static class OS {
       public static UserEvent WaitForNextEvent() { /* ... */ }
       public static Point ReadLastMouseClick() { /* ... */ }
   }
   ```

3. Your framework must support applications that react to a press of a button (which would be detected for example by its `HandleMouseClick` method). Programmers of such applications must not be required to modify any part of the framework code. The application code must be able to communicate with other controls of the same window, e.g. read text from a TextBox located in it.

   As an example of how your framework should be used, sketch the key parts of a simple application that will have a single window containing one "Erase" Button, and one TextBox. If the user enters a file name into the TextBox, and then presses the "Erase" button (clicks on it), the application should delete the given file using the `void Delete(string path)` method of a prepared class `File`.

5 Compilation (3 points)

Let us describe a hypothetical CPU architecture. It has three special 32-bit registers:
– **SP** - Stack Pointer, it points to the first free byte of the stack (the empty stack has SP=0)
– **PC** - Program Counter, it points to the next instruction to be executed
– **FP** - Frame Pointer, it holds the pointer to the stack frame of the current function (see the CALL instruction)

The memory provided by the CPU architecture is organized as a stack. Any address must be in the range [0, SP-1], otherwise it is an invalid memory access. Numbers are represented as 32-bit signed integers in two's complement.

The CPU instructions are listed in the table below. We use the following notation in the operation description:
– *imm32* - 32-bit signed integer constant
– *label* - symbolic label to the code
– *PUSH(X)* - store value X on the top of the stack
– *POP(N)* - pops a value from the top of the stack, this operation is an Nth operation
– * - dereference

instruction	description	operation
ADDI4	add two integers from the top of the stack and store the result on the top of the stack	PUSH(POP(1)+POP(2))
ADDSP imm32	add a signed integer constant to the SP register	SP=SP+imm32
SHL imm32	shift the integer on the top of the stack to the left by imm32 bits	PUSH(POP(1) SHL imm32)
LLD4 imm32	push a 32-bit value from the stack frame on the top of the stack	PUSH(*(FP+imm32))
LST4 imm32	store a 32-bit value from the top of the stack to the stack frame	*(FP+imm32)=POP(1)
JMP label	unconditional jump	PC=label
JLEI4 label	conditional jump, executed if the value on the top of the stack is less or equal than the value below the top	if POP(1)<=POP(2) then PC=label
LIC4 imm32	push a 32-bit integer constant on the top of the stack	PUSH(imm32)
RET	return from a function	FP=POP(1), PC=POP(2)
CALL label	call a function	PUSH(PC), PUSH(FP), FP=SP
XLD4	push a 32-bit value referenced by the top of the stack on the top of the stack	PUSH(*(POP(1)))
XST4	store a 32-bit value from the stack to an address referenced by the top of the stack	*(POP(1))=POP(2)

Let us further consider a Pascal compiler. The INTEGER data type corresponds to the 32-bit integer type. The VAR keyword denotes a parameter passed by reference. Passing an array by reference is implemented as passing a reference to the first element of the array. As an example, consider the following procedure:

```
PROCEDURE incvar(VAR a:INTEGER);
BEGIN
  a := a + 1;
END;
```

This can be compiled into processor instructions for example as follows:

```
LLD4    -12     ; read A reference
XLD4            ; read A
LIC4    1       ; const 1
ADDI4           ; A + 1
LLD4    -12     ; read A reference
XST4            ; store A + 1
RET
```

1. Translate the following procedure in the Pascal language to the instructions of our architecture.

   ```
   PROCEDURE addvar(a:INTEGER; b:INTEGER; VAR c:INTEGER);
   BEGIN
     c := a + b;
   END;
   ```

2. Translate the following procedure in the Pascal language to the instructions of our architecture.

   ```
   PROCEDURE incitem5(VAR a:ARRAY[0..9] OF INTEGER);
   VAR index:INTEGER;
   BEGIN
     index := 5;
     a[index] := a[index] + 1;
   END;
   ```

3. Translate the following procedure in the Pascal language to the instructions of our architecture.

   ```
   PROCEDURE incarr(VAR a:ARRAY[0..9] OF INTEGER);
   VAR i:INTEGER;
   BEGIN
     FOR i := 0 TO 9 DO
       a[i] := a[i]+1;
   END;
   ```

6 Optimization (3 points)

Note: Without any further explanation, you can exploit as a black-box the following theorem (informal): Any flow f can be decomposed into paths and cycles.

1. Let $G = (V, E)$ be an oriented graph and $s, t \in V$ be two different vertices in G. Formulate the shortest $s - t$ path problem in G as an integer linear program in such a way that for each edge $e \in E$, a binary variable x_e is used, and no other variables appear in the linear program.
2. Consider a linear relaxation of your integer linear program from question 1 and compare the optimal values of the original ILP and its relaxation. What is the relation between them? Explain.
3. Write the dual for the linear program from question 2; try to use at most $|V|$ dual variables.

7 Morphological, Syntactic and Semantic Analysis of Natural Languages (specialization question – 3 points)

1. List at least 4 basic requirements for a good spellchecker. Explain and justify those requirements.
2. Identify two most frequently used methods of spellchecking based upon the exploitation of a dictionary of the given language. Explain for which language types they are suitable and why.
3. Offering the best correction candidates is an inseparable part of the task of spellchecking. Describe at least two practically useful methods for selecting the best correction candidates.

8 Basic Formalisms for Description of a Natural Language (specialization question – 3 points)

1. Unification grammars use a special data type, the so-called Feature structures. Describe the basic properties of this data type.
2. Explain the mechanism of unification of two feature structures. Discuss the reasons why it is more convenient to use typed feature structures.
3. The most widely used type of a unification grammar has been the Head Driven Phrase Structure Grammar (HPSG). Describe the basic characteristics of this grammar.

9 Theory of Information (specialization question – 3 points)

We roll a die and we consider the outcomes from the set $\{1, 2, 3, 4, 5, 6\}$ as a value of the random variable X. We assume the uniform distribution of X. Then we consider the random variable Y having two possible values *even/odd* and the random variable Z with the values *true* (if the outcome is greater than 4) or *false* (if the outcome is not greater than 4). The ranges of X, Y and Z are summarized in the following table

random variable	**range**
X	$\{1, 2, 3, 4, 5, 6\}$
Y	$\{even, odd\}$
Z	$\{true, false\}$

1. Are X and Y statistically independent? Justify your answer.
2. Which of the random variables X, Y, Z has the highest entropy? Provide an exact justification.
3. Calculate the mutual information $\mathrm{I}(X; Y)$.