Qualifying examination in the software area

This is a sample examination involving the following 4 areas: Database systems, compiler design, programming languages, and software engineering. Each area is worth 60 points. You must answer questions from exactly 3 areas of your choosing.

Database Systems


2. [20 points] The following tables are part of a relational database.
   
   FLIGHTS (Number, DepartureAirportID, ArrivalAirportID, Cost)
   AIRPORT (AirportID, ZIP)
   CUSTOMER (CustomerName, ZIP)
   RETAILER (RetailerName, ZIP)
   ITEM (ItemID, Model, Brand)
   ITEM-RETAILER (ItemID, RetailerName, Price, Availability)
   
   where primary keys are indicated in bold. Write the following queries using the languages that are indicated.

   I. [10 points] Find the retailers that have available all the models of brand “Western Digital” (Relational Algebra). Note: if useful, you can use temporary relational variables.

   II. [10 points] Find the average price for each retailer of their products of brand “Western Digital” products. However, we are only interested in retailers that carry at least 10 different models of that brand. (SQL)

3. [15 points]

   a) [5 points] List all non-trivial functional dependencies that are satisfied by the following relation:

<table>
<thead>
<tr>
<th>S</th>
<th>T</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>t1</td>
<td>u1</td>
</tr>
<tr>
<td>s1</td>
<td>t1</td>
<td>u2</td>
</tr>
<tr>
<td>s2</td>
<td>t1</td>
<td>u1</td>
</tr>
<tr>
<td>s2</td>
<td>t1</td>
<td>u3</td>
</tr>
</tbody>
</table>
b) [10 points] Use Armstrong’s rules to prove the soundness of the union rule:

\[ \alpha \rightarrow \beta \text{ holds and } \alpha \rightarrow \delta \text{ holds, then } \alpha \rightarrow \beta \cup \delta \text{ holds} \]

4. [5 points] Give an example schedule of transactions which is serializable but not serial.

5. [10 points] Supply is a many-many relationship set between entity set Supplier and entity set Part. Supplier has attributes s# and city and Part has attributes p# and color. How are the two entity sets and the relationship set stored in relational tables?

6. [5 points] Consider a transaction where there is a transfer of funds from one account to another. Identify the property of transactions which is required if during the execution of the transaction, there is a failure.
**Compiler Design**

1. **[10 points]** The grammar G1:

   \[ S \rightarrow SS \mid a \mid b \]

   is ambiguous. That means *some* (at least one) of the strings in its language have more than one leftmost derivation. However, it may be that some strings in the language have only one derivation.

   a. Give two examples of strings in G1 that have exactly ONE derivation (i.e., no more than one way to derive the string.)

   b. Give three strings in G1 that have exactly TWO leftmost derivations in G.

2. **[16 points]** Programming languages are often described using an extended form of a context-free grammar, where curly brackets are used to denote a construct that can repeat 0, 1, 2, or any number of times. For example:

   \[ A \rightarrow B \{C\} D \]

   says that an A can be replaced by a B and a D, with any number of \{C\}'s (including 0) between them. This notation does not allow us to describe anything but context-free languages, since an extended production can always be replaced by several conventional productions.

   Suppose a grammar has the extended production:

   \[ A \rightarrow 1\{2\}3 \]

   Convert this extended production to conventional productions.

3. **[34 points]** Consider the given grammar G3. (\(\epsilon\) denotes the empty string).

   \[
   \begin{align*}
   S & \rightarrow ABCDE \\
   A & \rightarrow a \\
   B & \rightarrow b \mid \epsilon \\
   C & \rightarrow c \mid \epsilon \\
   D & \rightarrow d \\
   E & \rightarrow e \mid \epsilon
   \end{align*}
   \]

   a. Compute the FIRST of every non-terminal symbol in the grammar.

   b. Compute the FOLLOW of every non-terminal symbol in the grammar.

   c. Compute the LL(1) parsing table for G3.
Programming Languages

1. [20 points]
   a. Suppose that function \( f(p,q,r) = \) if \( p < 2 \) then \( q \) else \( r \), and that function \( g(z) = g(z) + 1 \). Give the result of the function call \( f(0, 4, g(6)) \) under call-by-value and call-by-name, respectively.

   b. Give one disadvantage and advantage of call-by-name over call-by-value?

2. [20 points]
   a. Consider the following piece of code, where \( fn \ x \Rightarrow (x + a) \) is a function that takes argument \( x \) and returns the value of \( x + a \). What is the result of \( b(6) \) in the code under static scoping and dynamic scoping, respectively?

   ```
   let a = 1 in
   let b = fn x => (x + a) in
   let a = 2 in
   b(6)
   ```

   b. What is the disadvantage of dynamic scoping over static scoping?

3. [20 points] Discuss the concept of automatic garbage collection in programming languages and describe how an algorithm for garbage collection may typically work.

   Name at least two programming languages whose implementations (e.g., compilers) always include automatic garbage collection.

   Name and discuss at least 2 mistakes that programmers can make when they write code in a language that does not support automatic garbage collection.
Software Engineering

1. **[15 points]**
   
   (a) Explain the concept of coupling and cohesion.
   
   (b) Give examples to contrast tight vs. loose coupling.
   
   (c) Give examples to contrast weak vs. strong cohesion.

2. **[15 points]** Explain the differences between Data Flow-oriented and Object-oriented Design Methods.

3. **[30 points]** Mutation testing is an important program analysis technique that is used to evaluate the thoroughness of a test set T supplied to a program P. The basic idea of mutation testing is to generate a set of programs that are similar to P by introducing arbitrary errors in P. The programs obtained in this way are called mutants of P. Test set T is deemed adequate if it distinguishes all mutants of P from P. Test set T distinguishes a mutant M from P if there is at least one test case in T that causes M and P to produce different outputs.

   Now consider the program P below. (Assume that x and y are real variables.)

   ```
   read(x);
   read(y);
   if (x - y ≥ 3) or (y - x ≥ 4)
       then if x + y ≤ 10
           then write(1)
           else write(2)
       else write(3);
   ```

   Note that program P contains two ≥ predicates and one ≤ predicate. Suppose that mutants of program P are generated by changing the ≥ predicates appearing in P into ≤ predicates and vice versa. Note that three mutants are obtained by changing a single predicate. Three additional mutants are obtained by changing two predicates simultaneously. A final mutant is obtained by changing all 3 predicates appearing in P. Thus, we have a combined total of seven mutants of P.

   Given program P and its seven mutants above, generate a test set T that distinguishes all seven mutants from P. Your test set should not contain more than four test cases.
Textbooks:


Concepts of Programming Languages. Robert Sebesta. Addison-Wesley. 8/e.