Automated Software Engineering Exam 3 (January 23, 2025)

Task 1: Short questions — 15 points

1.1 During performance measurement, we use VisualVM (a popular performance measurement tool for Java). If we run the measurement in both Sampling and Profiling mode, the Profiling mode shows that one part of the code is much slower than the other. What can we say about that part of the code?

1.2 We want to use complex state chart models in an embedded environment with limited computational capacity. Should we use code generators or interpreters? Please explain!

1.3 Please give us a simple example of a pattern-based static analysis rule!

Task 2: Graph modeling — 25 points

a) Write a **Refinery metamodel** based on the following specification:

We want to model a system for managing quantum computing experiments and resources. A quantum computing laboratory contains computers, which can be quantum computers or classical computers. Each quantum computer consists of 2 to 16 quantum bits (qubits). Qubits can be either superconducting or trapped-ion based. An experiment requires exactly one quantum computer and at least one classical computer for control. Each experiment runs one or more quantum circuits, which consist of 1 to 100 quantum gates. Quantum gates can be either single-qubit gates (with a single input qubit) or two-qubit gates (with a left and a right input qubit). Each quantum circuit may produce some measurement results. Result datasets can be either raw measurement data or processed data. Each processed dataset is derived from exactly one raw dataset using a specific noise reduction algorithm.

Only provide Refinery code and do NOT write Java code or draw an UML class diagram.

Use the following concept names: `appliedAlgorithm`, `circuits`, `ClassicalComputer`, `Computer`, `computers`, `controlledBy`, `Dataset`, `derivedFrom`, `Experiment`, `gates`, `input`, `Laboratory`, `leftInput`, `NoiseReductionAlgorithm`, `ProcessedData`, `QuantumCircuit`, `QuantumComputer`, `QuantumGate`, `Qubit`, `qubits`, `RawData`, `results`, `rightInput`, `runsOn`, `SingleQubitGate`, `SuperconductingQubit`, `TrappedIonQubit`, `TwoQubitGate`

b) Draw a **graph model** based on the following data:

The Stanford Quantum Lab has two computers: an IBM-Q65 quantum computer with three superconducting qubits (alpha, beta, and gamma) and a Dell PowerEdge R740 configured for control tasks. A quantum entanglement experiment is running on IBM-Q65 and is controlled by the PowerEdge R740. The experiment includes a quantum circuit for producing the Bell state that contains two quantum gates: a Hadamard gate operating on qubit alpha, and a CNOT gate with qubits alpha and beta as its left and right inputs, respectively. The circuit produces two datasets: a raw dataset, and a processed dataset derived from the raw dataset using the Richardson extrapolation algorithm.

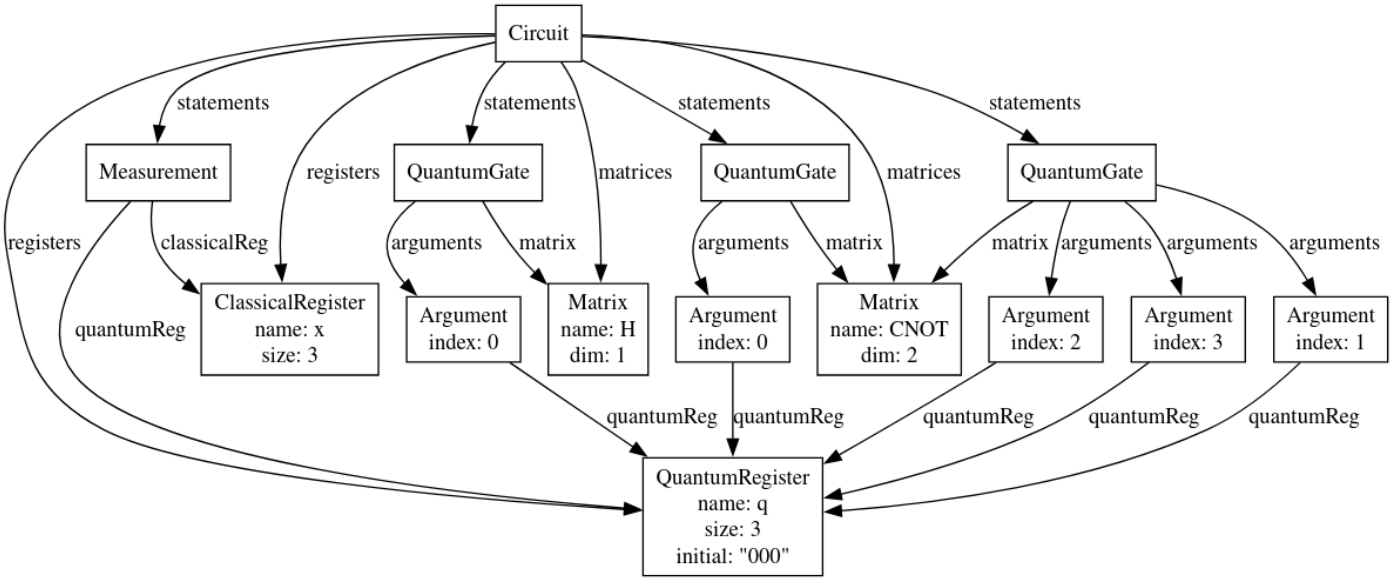
Only provide a graph model and do NOT write Java or Refinery code.

c) According to a domain expert, the following **constraint** holds: *qubits in a quantum circuit must belong to the quantum computer the circuit runs on.* However, the expert suspects that this cannot be enforced by the current version of the metamodel. To confirm, draw a **graph model** that conforms to the metamodel but violates the constraint.

d) The domain expert also noted that some quantum gates operate on 3 qubits at the same time. **Recommend** a way to update the metamodel for representing such gates.

Task 3: Textual modeling — 30 points

We would like to create a **textual domain-specific modeling language** to describe general quantum circuits. The example below shows the desired **concrete syntax** (textual description) **and abstract syntax** (instance graph model) for the language:




```
quantum reg q[3] = "000";

matrix H dim = 1;
H(q[0]);

matrix CNOT dim = 2;
CNOT(q[0]);
CNOT(q[1], q[2], q[3]);

reg x[3];
x = measure(q);
```

a) Create a **Langium grammar** to parse this language! These declarations are available to you:

```
grammar QuantumProgramming

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
terminal INT: /\d+/;
terminal STRING: /"[^"]*"/;
```

Provide the rest of the grammar.

b) Create a **Jinja2 template** to generate a quality report for the quantum circuit. The input of the template is the `Circuit` object parsed by the grammar you created in part a). An example quality report with all the issue types you need to support is shown below:

```
Too few arguments (got 1, expected 2) for matrix CNOT.
Too many arguments (got 3, expected 2) for matrix CNOT.
Index 3 of register q is out of range.
```

To help you in reading the example, we emphasized text coming directly from instance model in **bold**. Make sure to output an issue for every gate with too many or too few arguments, and for every argument with an index out of range for the corresponding quantum register. You do not have to validate register initializers and measurements.

In the instance model, the `type` attribute contains the type of each object (e.g., use `y.type == "Measurement"` to check if `y` is of type `Measurement`). Cross-references are encoded as strings equal to the `name` of the referenced object. To determine the length on a Python list named, e.g., `someList`, use the Jinja2 notation `someList|length`.