Prof. Dr. Christoph Bockisch
MSc Steffen Dick
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Philipps Universität Marburg

Companion exercises
Objektorientierte Programmierung: Wintersemester 2021/2022

No. 10, due until 31.01.2022

Attention: You are able to achieve more than the usual 12 points, though only 12 points count towards your debit. Any achieved points above 12 will be granted as bonus points.

Task 10.1: NieR: Automata — 16 Points

A deterministic finite automaton (DFA) can be defined with the following quintuplet: (Q, Σ, δ, q_0, F).

Q is the set of states. Every state that an automaton has is in this set. Σ defines an alphabet. Any alphabet, Σ, contains symbols, σ, that can be used as input for the automaton. δ is defined as the function for changing a state. It is defined as follows: $\delta : Q \times \Sigma \to Q$. This means that δ takes a state from Q and a symbol from Σ and returns a new state from Q. q_0 is defined as the start-state. Normally, it is called q_0 but can have a different name. F is the set of accepting states which means $F \subseteq Q$.

A DFA accepts a word (a sequence of σ) if the last achieved state is an accepting state, so $q_n \in F$. The speciality of a DFA is that δ is unambiguously defined for every $\sigma \in \Sigma$ and every $q_n \in Q$ (for every symbol and every state there exists exactly one follow-up state).

You are supposed to implement a DFA in this task.

Attention: You may ignore the empty word (ϵ) for this task.

a) Implement the following classes and make sure that their fields have the correct visibility and have Constructors: — 1.5
   - `State` that holds an ID of type `String` and a `boolean` that is set **`true`** if it is an accepting state.
   - `Transition` that holds two `String` IDs of States (one for a start and one for an end) and a `Character` for the symbol for the transition.
   - `Alphabet` that has an array of `Character` for allowed symbols. Also implement a method that checks whether a symbol is within the given alphabet.

b) Implement the following checked `exceptions`: `StateAlreadyExists`, `StateDoesNotExist`, `SymbolNotInAlphabet` and `TransitionAlreadyExists`. Make sure that every exception has a useful error message. — 1

c) Implement a abstract class `GenericAutomaton`. `GenericAutomaton` should have the following fields: `State[] states`, `Transition[] transitions`, `Alphabet alphabet` and `String start` (holds the ID of the start state). Make sure that your fields have the correct visibilities (some fields may have to be accessed by heirs). Implement a constructor that takes one instance of `Alphabet` and initializes every other required field with default values. — 1

d) Add abstract methods javaCodevoid reset() and `boolean isAccepting()` to `GenericAutomaton`. [1]

e) Implement `void addState(State state, boolean isStart)` in `GenericAutomaton` that takes a `State` and adds it to `states`. If a state with that id already exists, throw a new `StateAlreadyExistsException`. If `isStart` is set to `true`, `start` should be set to the ID of the new state. [2]

f) Implement `public State findState(String id)` that should return the `State` with the id `id`. If no such state exists, return `null`. [1]

g) Implement `protected void addTransition(Transition transition)` in `GenericAutomaton` that adds a `Transition` to `transitions`. Should either one of the two states not exist within `states`, throw a new `StateDoesNotExistException`. Should the symbol not exist within your `Alphabet`, throw a new `SymbolNotInAlphabetException`. [2]

h) Implement the class `DFA`. `DFA` should inherit from `GenericAutomaton` and have its own private field `String current` for storing the ID of the current state. Make sure that you use the inherited constructor. [0.5]

i) Implement both of the abstract methods `void reset()` and `boolean isAccepting()` in `DFA`. `void reset()` is supposed to set the current state back to the start state. `boolean isAccepting()` should return `true` if the current state is an accepting state and `false` otherwise. [1]

j) In order to add new transitions, implement `public void makeTransition(String q1, String q2, Character symbol)` that adds a new transition to your automaton. Use `addTransition` in your implementation. If a transition with the very same start-state and symbol exists already, throw a new `TransitionAlreadyExistsException`. Make sure that exceptions from `addTransition` get thrown further. [1]

k) Lastly, to actually use transitions to change states, we need another method. Write the method `public String delta(Character symbol)` in `DFA` that uses a ForEach-loop to iterate over every transition to find the new state by using the current state and the symbol. Remember to set the current state to the new state and return the id of the new state. [2]

l) Think about a scenario where a DFA has at least 2 states, 2 symbols within its alphabet and contains enough transitions. Use that scenario to test every method that is publicly visible from an instance of DFA with JUnit-tests. [2]