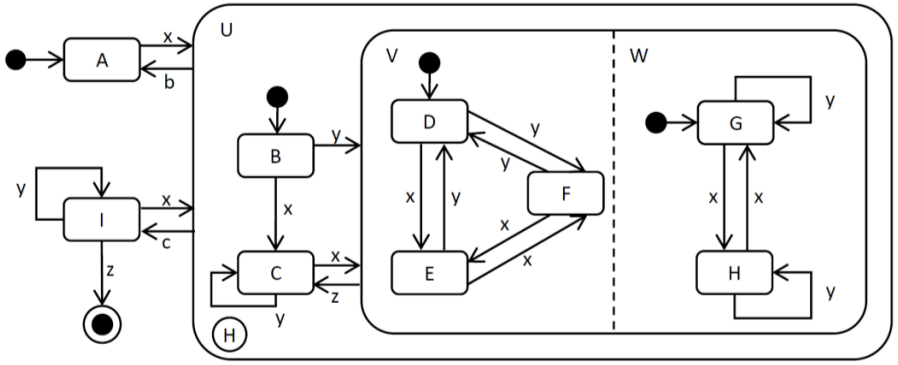
Introduction to Software Engineering

Assignment 5

Dimitri Tabatadze, 2024-11-11

1

Given the following state chart

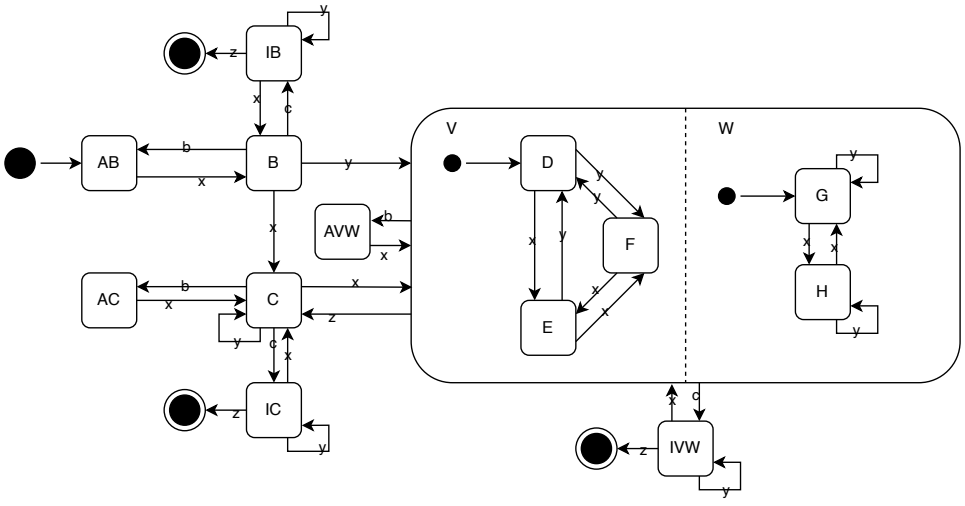


a) List the states visited for inputs `x, y, x, y, y, z, b, x, y, c, y, x`.
b) List the states visited for inputs `x, x, x, x, x, z, x, y, x, x`.
c) transform the automaton into an equivalent, parallel one without history attribute.

Solution

a) `A` → `B` → `(D G)` → `(E H)` → `(D H)` → `(F H)` → `C` → `A` → `C` → `C` → `I` → `I` → `C`.
b) `A` → `B` → `C` → `(D G)` → `(E H)` → `(F G)` → `C` → `(D G)` → `(F G)` → `(E H)` → `(F G)`

c)



2

For the following classes and interfaces, indicate briefly which secret is being hidden (or will be hidden in case of an interface), and how.

a) `java.io.Closeable`
b) `java.lang.Comparable<T>`
c) `java.util.List`
d) `java.util.Locale`

Solution

a) Behind the interface `java.io.Closeable` is hidden the type of object that the user of `java.io.Closeable` may be working with. For example, both `java.util.Scanner` and `java.io.BufferedReader` implement `java.io.Closeable`.

b) Behind the interface `java.lang.Comparable<T>` we can hide the implementation of comparison. For example, we can implement comparison of two strings in two ways:

- comparing the lexicographical order,
- comparing the length.

c) The `java.util.List` interface hides the datastructure behind the list. One can implement `java.util.List` with a linked list and one can implement it with a dynamic array.

d) `java.util.Locale`, as far as any user is concerned, can be using a single integer to store the whole data or can use a string to achieve the same thing.

3

Your colleagues at Pear Corp have run into a strange situation. Since you are coming from KIU and are reputed of solving the toughest problems, they have come to you during coffee break. They have already printed out the relevant code fragments

- class `AppPrototype`:

```java
public static void main(String[] args) {
    List<String> el = new EmployeeManagementPrototype().generateInitialEmployeeList();
    el.add("Martina Mustermann");
}
```

- class `IEmployeeManagement`:

```java
public interface IEmployeeManagement {
    /**
     * Generates the initial list of employees. Each employee is represented as a String
     * (Given Name(s), Last Name). New employees (Strings) may be added or removed from
     * this List.
     * @return the initial list of employees
     */
    List<String> generateInitialEmployeeList();
}
```

- class `EmployeeMa nagementPrototyle`:

```java
public class EmployeeManagementPrototype implements IEmployeeManagement {
    @Override
    public List<String> generateInitialEmployeeList() {
        return List.of("Martin S. Pinnér", "Marvin Bertsch", "Max Power");
    }
}
```

The above code violates Liskov's substitution principle. Describe where and why the violation occurs. Note: it is not enough to look at parameter types.

Solution

Appending to an immutable (created by `List.of`) list is not allowed.