VU Qualitätssicherung in der Softwareentwicklung
LV-Nr. 716.066, SS 2018

Aufgabenblatt 3

ao. Univ.-Prof. Dr. Bernhard Aichernig
Christoph Rehbichler, crehbichler@student.tugraz.at
Stefan Kuhs, kuhs@student.tugraz.at

15.05.2018, Graz

1 Code Contracts - ExamPreparation (2 Punkte)

Ziel dieser Aufgabe ist es, sich zur Klausurvorbereitung nochmals mit den Code Contracts von C# zu beschäftigen. Dazu sollen, wie in Aufgabenblatt 2, Methoden eines Interfaces möglichst genau spezifiziert werden. Die Kontrakte sind direkt im Projekt `ExamPreparation` zu erstellen. Erstellen Sie dafür Kontraktklassen für das Interface. Eine korrekte Implementierung der Methoden ist bereits gegeben und diese darf nicht verändert werden. Für das enthaltene Test-Projekt bestehen keine formalen Anforderungen. Sie können dieses nutzen, um Ihre Kontrakte mit selbst erstellten Mutanten zu überprüfen.

Spezifizieren Sie die folgenden Methoden:

`public int[] Intersect(int[] array1, int[] array2)`:
Berechnet die Schnittmenge zweier Arrays. Die Reihenfolge der Elemente ist nicht zu beachten. `Beispiel` Input: [1, 2, 3], [2, 3, 4] , Output: [2, 3]

`public bool IsProperSubsetOf(int[] subset, int[] superset)`:
Diese Funktion gibt genau dann true zurück, wenn `subset` eine echte Teilmenge von `superset` ist. `Beispiel` Input: [13], [13, 37], Output: true

Es soll angenommen werden, dass es sich bei den Inputs um **Sets** handelt. Achten Sie außerdem darauf, dass die Resultate exakt die in der Definition angegebenen Elemente beinhalten. Zusätzlich zu diesen Einschränkungen darf kein Eingabeparameter `null` sein.

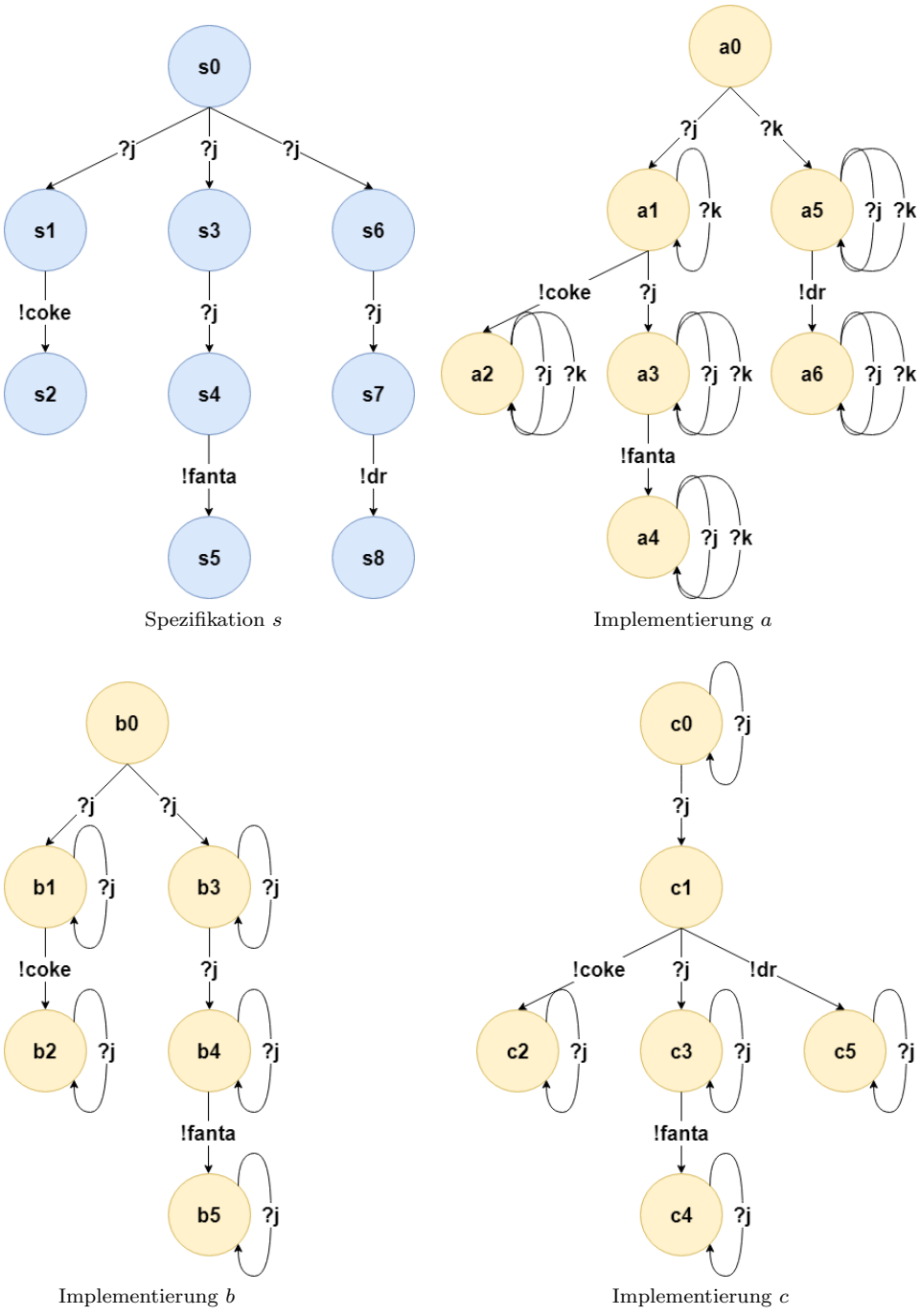
1.1 Bewertungsgrundlage

Als Bewertungsgrundlage dient das Projekt `ExamPreparation`. Das enhaltene Test-Projekt wird nicht überprüft. Die volle Punktezahl erhält eine Abgabe genau dann, wenn

- alle nicht erlaubten Eingabeparameter von Ihren Kontrakten erkannt werden,
- alle fehlerhaften Implementierungen (Mutanten) von Ihren Kontrakten als solche erkannt werden.

2 IOCO (4 Punkte)

Es sind die folgenden LTS-Modelle von Getränkeautomaten gegeben, wobei das erste Modell von links die Spezifikation angibt und die restlichen 3 Modelle Implementierungen darstellen:



Spezifikation s | Implementierung a

Implementierung b | Implementierung c

Abbildung 1: Die LTS-Modelle von Getränkeautomaten

Die Input-Actions $?j$ und $?k$ entsprechen jeweils einem bestimmten Tastendruck und die Output-Actions entsprechen der Ausgabe von Getränken, wobei $!coke$ für Coca Cola, $!fanta$ für Fanta und $!dr$ für Dr. Pepper steht.

2.1 Aufgabenstellung

Folgende Aufgaben sollen durchgeführt werden:

- Die LTS-Modelle in Abbildung 1 beinhalten keine Transitions, welche mit δ-Labels für die Quiescence-Observation markiert sind. Geben Sie für jedes Modell an, welche States quiescent sind.
- Bestimmen Sie die IO-Konformanz der Implementierungen zur Spezifikation s unter der Berücksichtigung von Quiescence:
  - a **ioco** s ?
  - b **ioco** s ?
  - c **ioco** s ?

  Begründen Sie Ihre Entscheidungen. Eine Begründung für eine nicht-konforme Implementierung zeigt den Output nach einem Trace der Implementation Under Test (IUT) und, dass dieser nicht in der Spezifikation möglich ist.

  Beispiel: out(x0 after ?j) = {!fanta} $\not\subseteq$ out(s0 after ?j) = {!coke}

  Für konforme Implementierungen sollen alle Outputmengen der Traces der Spezifikation angeführt werden. Wiederholungen, welche zu unendlich vielen Traces führen, müssen nur einmal wiederholt werden. Für alle Traces soll gezeigt werden, dass die Outputmengen der IUT in der Spezifikation möglich sind.

  Beispiel: out(y0 after ?j) = {!coke} $\subseteq$ out(s0 after ?j) = {!coke, !dr}

  Wichtig: Für die konforme Implementierung werden alle Traces benötigt. Nicht nur einer, wie in dem angeführten Beispiel.
  Die gegebenen Beispiele decken sich nicht zwingend mit der für dieses Beispiel definierten Spezifikation s.

Erstellen Sie eine Datei `solution_ioco.pdf`, welche Ihre Lösungen der Aufgaben beinhaltet.

3 Model-Based Testing - FsCheck (12 Punkte)

Im Rahmen dieser Aufgabe sollen Sie sich mit einer Black-Box-Testmethode beschäftigen. Ihre Aufgabe ist es, mittels FsCheck[1] einen Teil des Messageboards modellbasiert zu testen. Dazu müssen Sie das Nuget Package FsCheck[2] herunterladen und installieren. Ein Beispiel zu FsCheck ist auf der Vorlesungsseite zu finden. Desweiteren können Sie im GitHub-Repository von FsCheck Tutorials finden. Es gibt zwei unterschiedliche Methoden für das Testen von Modellen. Die in dieser Aufgabe verwendete Methode ist in der FsCheck Dokumentation unter *Model-based testing* beschrieben.

3.1 Aufgabenstellung - Modellierung

Das FsCheck Model soll in dem Projekt `MessageBoardTest` implementiert werden. Einige vorimplementierte Klassen sind bereits in diesem Projekt zu finden, um den Einstieg zu erleichtern. Stellen Sie sicher, dass sich Ihre Abgabe mit den originalen Klassen des Implementierungsprojekts kompilieren lässt. Für die Bewertung Ihrer Abgabe werden die Änderungen an der Implementierung verworfen und mit mutierten Implementierungen getestet.

Es sollen drei Commands für das Messageboard implementiert werden.

- `Publish(string author, string message)`:
  Dieses Command soll eine Nachricht mit dem gegebenen Autor und Nachrichteninhalt veröffentlichen. Das System Under Test (SUT) soll genau dann erfolgreich sein, wenn das System mit einer `OperationAck`-Nachricht auf die `Publish`-Nachricht antwortet. Wenn das System mit `OperationFailed` antwortet, war das SUT nicht erfolgreich. Überprüfen Sie in der Post-Condition ob das Model ebenfalls erfolgreich beziehungsweise nicht erfolgreich ist. Verhält sich das SUT anders als das Model, soll die Post-Conditon fehlschlagen. Verhält es sich gleich, ist die Post-Condition nicht verletzt.

- `FindMessages(string messageAuthor)`:
  Dieses Command soll die String-Repräsentationen aller Nachrichten des gegebenen Autors als sortierte Sequenz zurückgeben. Die String-Repräsentation entspricht dem Return-Wert von `UserMessage.ToString()`. Überprüfen Sie in der Post-Condition ob sich SUT und Model gleich verhalten.

- `LikeMessage(string author, string message, string likeName)`:
  Dieses Command soll die Nachricht mit dem Inhalt `message` vom Autor `author` unter dem Namen `likeName` liken. Die SUT schlägt genau dann fehl, wenn die Kombination aus Autor und Nachricht nicht existiert, oder wenn das System mit einer `OperationFailed`-Nachricht antwortet. Wenn das System mit `OperationAck` antwortet, ist das SUT erfolgreich. Überprüfen Sie in der Post-Condition ob sich SUT und Model gleich verhalten.

3.2 Aufgabenstellung - Requirements

In dieser Aufgabe soll das MessageBoard vereinfacht als Datenbank betrachtet werden, die Nachrichten beinhaltet. Abstrakt kann eine solche Datenbank als Menge von Nachrichten mit Zugriffsfunktionen (RunActual der Commands) modelliert werden.

[1] `https://github.com/fscheck/FsCheck`
[2] `https://www.nuget.org/packages/FsCheck/`

Im Fokus der Modellierung stehen funktionale Requirements, d.h. es ist nicht nötig, Details wie die Zeit, die das System benötigt um auf eine Anfrage zu antworten, zu modellieren. Die vollständige Liste der Requirements ist:

R1 Eine Nachricht darf nur gespeichert werden, wenn ihr Text weniger als oder exakt 10 Zeichen beinhaltet. Diese Überprüfung wird in der Klasse `Worker` vorgenommen. Die Nachrichtenlänge wurde im Vergleich zu den vorhergehenden Aufgabenblättern von 140 auf 10 Zeichen reduziert, um die Lesbarkeit der Command-Sequenzen zu erleichtern.

R2 Eine Nachricht darf nur gespeichert werden, wenn noch keine identische Nachricht gespeichert wurde. Zwei Nachrichten sind identisch, wenn sowohl Autor als auch Text beider Nachrichten gleich sind.

R3 Eine Nachricht darf nur gelikt werden, wenn sie existiert.

R4 Eine Nachricht darf nur durch Personen gelikt werden, die die entsprechende Nachricht zuvor noch nicht gelikt haben.

R5 Es soll möglich sein, eine sortierte Liste der String-Repräsentationen aller Nachrichten eines Autors abzurufen. Die String-Repräsentationen von Nachrichten ist im Modell durch `AbstractUserMessage.ToString()` und in der Implementierung, wie oben angemerkt, durch `UserMessage.ToString()` gegeben.

R6 Erfolgreiche Anfragen, Like- bzw. Update-Anfragen, sollen durch das Senden von `OperationAck` bestätigt werden. Anfragen gelten als erfolgreich, wenn eine Nachricht gespeichert wurde, oder wenn ein *Like* zu einer Nachricht hinzugefügt wurde.

R7 Anfragen, die nicht erfolgreich sind, sollen durch das Senden von `OperationFailed` bestätigt werden.

Sie können davon ausgehen, dass das System immer antwortet. Wenn Sie also Nachrichten vom Typ `ClientMessage` an das System schicken, werden Nachrichten vom Typ `Reply` nach einer endlichen Anzahl von Zeiteinheiten zurückgeschickt.

Durch den vorgegebenen Fokus ergibt sich eine weitere Vereinfachung. Es ist nicht nötig zu modellieren bzw. zu testen, wie sich das System bei mehreren gleichzeitig gestellten Anfragen verhält. Dadurch ergibt sich folgende vorgeschlagene Struktur der Adapter-Methoden:

1. Initialisieren der Kommunikation zwischen System und einem Test-Client (Objekt der Klasse `TestClient`)
2. Kommunikation, um die geforderte Funktionalität der Commands zu implementieren
3. Auswerten der Antworten des Systems
4. Beenden der Kommunikation

3.3 Bewertung - Implementierungen testen (4 Punkte)

Im Framework sind 4 Implementierungen (als dll-Dateien) enthalten, von denen Sie mittels FsCheck überprüfen sollen, ob sie korrekt oder fehlerhaft sind. Führen Sie dazu Testruns des Models durch.

Beschreiben Sie in `solution_implementations.pdf` für jede fehlerhafte Implementierung, gegen welchen Teil der Spezifikation sie verstößt. Fügen Sie außerdem

eine, vom Testoutput erzeugte, Command-Sequenz hinzu, die den Fehler entdeckt. Achten Sie darauf, dass die Commands sauber repräsentiert werden. Das heißt, der Name des Commands und alle verwendeten Parameter sollen klar ersichtlich sein. Sie können dazu die ToString() Methode überschreiben. Fügen Sie die Command-Sequenzen in dem Verzeichnis `sequences` ins SVN hinzu (das Namensschema lautet: `seq_<nummer>.txt`, wobei `nummer` die Nummer der Implementierung ist). Geben Sie für jede Implementierung, die Sie als korrekt identifiziert haben, an, wie Sie sichergestellt haben, dass sie korrekt ist.

Hinweis: Sie können die Implementierungen testen, indem Sie die Referenz auf das originale MessageBoard-Projekt aus den Referenzen des MessageBoardTest-Projekts löschen und stattdessen eine der dll-Dateien referenzieren. Danach müssen Sie noch mittels `Build` → `Rebuild Solution` das Projekt neu kompilieren, danach können Sie die Tests erneut durchführen.

3.4 Bewertung - Weitere Mutanten (8 Punkte)

Zur weiteren Bewertung werden von uns, wie in den vorherigen Beispielen, Mutanten verwendet, die Sie nicht von uns erhalten. Stellen Sie also sicher, dass alle Requirements erfüllt werden und nicht nur die von uns zur Verfügung gestellten Implementationen korrekt erkannt werden. Achten Sie darauf, dass die mit `[Testmethod]` annotierte Funktion fehlschlagen soll, falls FsCheck ein Counterexample findet.

4 Punkteverteilung (18 Punkte)

- **2 Punkte:** Code Contracts - ExamPreparation
- **4 Punkte:** IOCO
- **12 Punkte:** Model-Based Testing - FsCheck
  - **4 Punkte:** Implementierungen testen
  - **8 Punkte:** Weitere Mutanten

5 Abgabedetails

Frist: bis Di. 12.06.2018, **10:30 MEZ** (somit kurz vor Vorlesungsbeginn)
Art der Abgabe: Per SVN, welches vom OnlineSystem erzeugt wird und die folgende Struktur haben soll. Es sollen keine weiteren Files, insbesondere Binär-Files und Test-Resultate, eingecheckt werden. Korrigiert wird die letzte innerhalb der Frist eingecheckte Version.
Anmerkungen: In der Datei `readme.txt` können Sie für die Korrektur relevante Informationen angeben. Im Normalfall kann diese Datei leer bleiben. In der Datei `participation.txt` soll die prozentuelle Arbeitsaufteilung auf die Gruppenmitglieder angegeben werden. Signifikante Unterschiede zwischen Mitgliedern führen zu Punkteabzügen für die Gruppenmitglieder, die weniger zur Abgabe beigetragen haben. Die Höhe der Abzüge richtet sich nach der Differenz der Arbeitsleistung zur Durchschnittsarbeitsleistung.
Wichtig: Die Projekte **müssen kompilieren**. Die korrekte Implementation darf nicht als fehlerhaft oder inconclusive erkannt werden, ansonsten wird die betroffene Methode bzw. das gesamte MessageBoard mit 0 Punkten bewertet.

5.1 SVN Ordnerstruktur

- `task3`
  - `participation.txt`
  - `readme.txt`
  - `solution_implementations.pdf`
  - `solution_ioco.pdf`
  - `sequences`
    - Command-Sequenzen (Namensschema: `seq_<nummer>.txt`)
  - MessageBoard
    - `MessageBoard`
      - alle Source-Files
      - `MessageBoard.csproj`
      - `Properties/AssemblyInfo.cs`
    - `MessageBoardTest`
      - alle Source-Files
      - `MessageBoardTest.csproj`
      - `Properties/AssemblyInfo.cs`
      - `bin/Debug/MessageBoardTest.dll` Kompiliertes Unittestprojekt
    - `MessageBoard.sln` Haupt-Projektdatei
  - ExamPreparation
    - `ExamPreparation`
      - alle Source-Files
      - `ExamPreparation.csproj`
      - `Properties/AssemblyInfo.cs`
    - `UnitTests`
      - alle Source-Files
      - `UnitTests.csproj`
      - `Properties/AssemblyInfo.cs`
    - `ExamPreparation.sln` Haupt-Projektdatei

Literatur

[1] Jan Tretmans. 2008. Model based testing with labelled transition systems. In Formal methods and testing, Robert M. Hierons, Jonathan P. Bowen, and Mark Harman (Eds.). Lecture Notes In Computer Science, Vol. 4949. Springer-Verlag, Berlin, Heidelberg 1-38.