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

Aufgabenblatt 2

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

10.04.2018, Graz

1 IntelliTest

1.1 IntelliFunctions (3 Punkte)

Ziel dieser Aufgabe ist es, sich mit IntelliTests in C# vertraut zu machen. Dafür sollen Sie drei Funktionen testen, die auf Arrays von Integern arbeiten. Die Funktionen sind in der Solution `IntelliFunctions` implementiert, in welcher bereits ein IntelliTest-Testprojekt enthalten ist, in das Sie Ihre parametrisierten Unittests hinzufügen sollen.

`public static int[] SortDescending(int[] array)`:
Sortiert ein Array von Integern absteigend.

`public static bool IsSubsetOf(int[] subset, int[] superset)`:
Returniert genau dann true, wenn der Parameter `subset` eine Teilmenge des Parameters `superset` ist. Beachten Sie, dass `subset` keine echte Teilmenge von `superset` sein muss. Es wird also auch true returniert, wenn beide Eingabemengen gleich sind.

`public static int[][] CombinationWithoutRepetition(int[] set, int numberOfElements)`:
Berechnet alle Kombinationen ohne Wiederholungen der Eingabemenge `set`, wobei die Anzahl der Elemente in jeder Kombination durch `numberOfElements` angegeben wird. Eine Kombination ist eine Auswahl von Objekten aus einer Menge, wobei die Reihenfolge nicht beachtet wird[1]. Stellen Sie sicher, dass `numberOfElements` nicht kleiner als eins und nicht größer als die Kardinalität von `set` ist.
Beispiel: Eine korrekte Implementierung könnte für den Aufruf mit den Parametern `{2,4,6}, 2` das Resultat `{{2,4},{2,6},{4,6}}` erzeugen oder auch `{{4,2},{2,6},{4,6}}`, da die Reihenfolge innerhalb der Kombinationen nicht berücksichtigt wird. Inkorrekt wären beispielsweise `{{2,2},{{2,4},{2,6},{4,6}}` oder `{{2,6},{4,6}}`.

Funktionen die Mengen verwenden (`IsSubsetOf` und `CombinationWithoutRepetition`), erwarten diese als Eingabeparameter. Stellen Sie in Ihren Tests also mittels `PexAssume` sicher, dass diese Funktionen nur mit Mengen als Parameter aufgerufen werden. Da die Überprüfung für große Arrays bei `CombinationWithoutRepetition` sich als schwierig erweisen kann, gilt weiters folgende Einschränkung: Die übergebene

[1] Siehehttp://de.wikipedia.org/wiki/Kombination_(Kombinatorik)

Menge kann maximal 10 Elemente beinhalten. Stellen Sie auch diese Bedingung sicher. Achten Sie darauf, dass die Resultate exakt die in der Definition angegebenen Elemente beinhalten und dass die Eingabeparameter durch die Funktionen nicht verändert werden können.

2 IntelliTest MessageBoard

Für diese Aufgabe sollen Sie IntelliTest verwenden, um einen Teil der Klasse `Dispatcher` möglichst genau zu testen. Sie sollen dafür parametrisierte Unittests schreiben, die alle Requirements testen. Im Übungsframework ist bereits ein IntelliTest-Testprojekt enthalten, in welches Sie Ihre Tests hinzufügen sollen. Für die Klasse `Dispatcher` sollen Sie Ihre Tests in die Klasse `DispatcherTest` einfügen.

Dispatcher Für diese Klasse existiert im Übungsframework bereits eine Ableitung `DispatcherPublic`, welche public Zugriffsfunktionen für alle relevanten Attribute bereitstellt. Zusätzlich dazu ist eine Factory für diese Ableitung und für die `InitCommunication`-Message im Framework enthalten. Factories werden von Pex verwendet, um die zu testende Klasse zu initialisieren und damit Testeingaben zu erzeugen. Sie können diese Factory und die Klasse `DispatcherPublic` beliebig erweitern und verändern (natürlich sollten die zu testenden `Dispatcher`-Methoden nicht überschrieben werden).
Beim Test soll darauf geachtet werden, dass der Dispatcher alle Requirements erfüllt. Die Methoden der Basisklasse `SimulatedActor`, wie beispielsweise `SimulatedActor.Tick()`, müssen nicht getestet werden. Folgende Requirements sollen abgedeckt werden:

RD1 Der Dispatcher startet im `NORMAL`-Modus.

RD2 Beim Starten (Spawnen) des Dispatchers sollen n Worker-Aktoren und ein MessageStore-Aktor gestartet werden, wobei n die Anzahl an Workern ist, die beim Dispatcher-Konstruktoraufruf angegeben wird.

RD3 Beim Starten des Dispatchers soll außerdem genau eine Instanz von `MessageStore` gestartet werden.

RD4 Im `NORMAL`-Modus soll der Dispatcher `InitCommunication`-Messages an den korrekten Worker weiterleiten. Wenn alle Worker in der Worker-Liste von 0 bis $n-1$ durchnummeriert werden, dann soll die Nachricht zufällig an einen Worker mit dem Index $x \mod n$ weitergeleitet werden. Dabei entspricht x einer Zufallszahl welche mithilfe der dem Attribut `InitCommunication.CommunicationId` als Seed erzeugt wird. $a \mod b$ entspricht dem nicht-negativen Rest der Ganzzahldivision $\frac{a}{b}$.

RD5 Im `NORMAL`-Modus soll der Dispatcher beim Erhalt einer `Stop`-Message allen Workern `Stop`-Messages schicken und in den `STOPPING`-Modus wechseln.

RD6 Im `STOPPING`-Modus soll der Dispatcher beim Erhalt einer `InitCommunication`-Message eine `OperationFailed`-Message an den sendenden Client zurücksenden, wobei `OperationFailed.CommunicationId` gleich `InitCommunication.CommunicationId` sein soll.

Hinweis: Für die ersten drei Requirements macht es Sinn, die Factory zu umgehen, indem man keinen Unittest-Parameter für den Dispatcher vorsieht und den Dispatcher selbst über den Konstruktor erzeugt.

3 Spezifikation (Code Contracts)

3.1 ExamPreparation (3 Punkte)

Ziel dieser Aufgabe ist es, sich mit den Code Contracts[2] von C# vertraut zu machen. Dazu sollen Methoden eines Interfaces möglichst genau spezifiziert werden. Die Kontrakte sind direkt im Projekt `ExamPreparation` zu erstellen. Erstellen Sie dafür Kontraktklassen für das Interface (siehe 2.8 Interface Contracts im User Manual). 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[] SortDescending(int[] array)`:
Sortiert ein Array absteigend.

`public int Min(int[] array)`:
Gibt als Ergebnis das minimale Element zurück. Die Funktion darf nur mit Arrays aufgerufen werden, die mindestens ein Element beinhalten.

`public int MaxIndex(int[] array)`:
Gibt den Index des größten Elements zurück. Die Funktion darf nur mit Arrays aufgerufen werden, die mindestens ein Element beinhalten.

`public int[] SymmetricDifference(int[] array1, int[] array2)`:
Bildet als Ergebnis die symmetrische Differenz $a1 \triangle a2$ zweier Arrays $array1$ und $array2$.

`public Tuple<int, int>[] RelationXIsBiggerThanY(int[] intarray))`:
Implementiert die Relation X ist größer als Y, es liefert also alle Tuplepaare(X,Y) für die die Relation in der Grundmenge gilt. Beispiel: Input: [1,2,3], Output: [(2,1),(3,1),(3,2)]

Bei den Methoden `SymmetricDifference` und `RelationXIsBiggerThanY` soll angenommen werden, dass es sich beim Input 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.

3.1.1 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] Beschreibung, Downloadlink und User Manual: `https://github.com/Microsoft/CodeContracts`

3.2 MessageBoard (5 Punkte)

In dieser Aufgabe sollen Teile der Klasse `ISimulatedActorSystem` möglichst genau spezifiziert werden. Die Kontrakte sind direkt im Projekt `MessageBoard` zu erstellen. Erstellen Sie dafür Kontraktklassen für die abstrakten Klasse (siehe 2.9 Contracts on Abstract Methods im User Manual). 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.

ISimulatedActorSystem (6 Punkte) Im Fokus des Tests dieser Klasse stehen die Methoden `Tick()`, `Spawn()`, `RunFor()` und `RunUntil()`. Um die Methoden `Tick()` und `Spawn()` adäquat spezifizieren zu können, wurde die Klasse `ISimulatedActor` um eine Property `TimeSinceSystemStart` erweitert, welche bei jedem Aufruf von `SimulatedActor.Tick()` inkrementiert wird.
Diese Property reflektiert wie `ISimulatedActorSystem.currentTime` die Anzahl an Zeiteinheiten seitdem das System gestartet wurde.
Folgende Requirements sollen abgedeckt werden:

ISAS1 Die aktuelle Zeit ist immer positiv.[3]

ISAS2 Es werden nur nicht-negative IDs für Aktoren vergeben. Die IDs werden beim Spawnen von Aktoren aufsteigend, in Einserschritten, beginnend bei 0 vergeben.[4]

ISAS3 Initial sind keine Aktoren im System registriert.

ISAS4 Initial ist die aktuelle Zeit gleich 0.

ISAS5 Es können nur Aktoren mittels eines Aufrufs von `Spawn` im System registriert werden, die noch nicht bereits registriert wurden.

ISAS6 Beim Spawnen von Aktoren soll `SimulatedActor.TimeSinceSystemStart` für den jeweiligen Aktor gesetzt werden.

ISAS7 Das Verstreichen einer Zeiteinheit soll allen Aktoren mitgeteilt werden, die aktuell im System registriert sind.

ISAS8 Ein Aufruf von `RunFor` soll nur möglich sein, wenn die übergebene Anzahl an Ticks größer als 0 ist.

ISAS9 Beim Aufruf von `RunFor` soll das System für die übergebene Anzahl an Ticks simuliert werden.

ISAS10 Ein Aufruf von `RunUntil` soll nur möglich sein, wenn der übergebene Endzeitpunkt größer oder gleich wie die aktuelle Zeit ist.

ISAS11 Beim Aufruf von `RunUntil` soll das System für so viele Ticks simuliert werden, bis der Endzeitpunkt erreicht ist. Zum Endzeitpunkt soll noch ein Tick simuliert werden (die aktuelle Zeit soll nach dem Aufruf also um eins größer sein als der Übergabeparameter).

[3] In `RunFor` und `RunUntil` könnten Integeroverflows entstehen. Stellen Sie mittels Kontrakten sicher, dass die Zeit nicht negativ wird.

[4] In `Spawn` können Integeroverflows entstehen, schränken Sie diese Methode mittels Preconditions ein, um sicherzustellen, dass keine negativen IDs vergeben werden.

3.2.1 Bewertungsgrundlage

Als Bewertungsgrundlage dient das Projekt `Messageboard`. 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), die gegen die angegebenen Requirements verstoßen, von Ihren Kontrakten als solche erkannt werden.

4 Punkteverteilung 16 Punkte

- **8 Punkte:** IntelliTests
  - **3 Punkte:** IntelliFunctions
  - **5 Punkte:** MessageBoard
- **8 Punkte:** Spezifikation (Code Contracts)
  - **3 Punkte:** ExamPreparation
  - **5 Punkte:** MessageBoard

5 Abgabedetails

Frist: bis Di. 15.05.2017, **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

`/task2`

- `participation.txt`
- `readme.txt`
- IntelliTests
  - IntelliFunctions
    - `IntelliFunctions`
      - `Functions.cs`
      - `IntelliFunctions.csproj`
      - `Properties/AssemblyInfo.cs`
    - `IntelliFunctions.Tests`
      - `Properties/AssemblyInfo.cs`
      - `Properties/PexAssemblyInfo.cs`
      - `IntelliFunctions.Tests.csproj`
      - `FunctionsTest.cs`
    - `IntelliFunctions.sln` Haupt-Projektdatei
  - MessageBoard
    - `MessageBoard`
      - alle Source-Files
      - `MessageBoard.csproj`
      - `Properties/AssemblyInfo.cs`
    - `MessageBoard.Tests`
      - alle Source-Files
      - `Factories/` alle Source-Files
      - `Properties/AssemblyInfo.cs`
      - `Properties/PexAssemblyInfo.cs`
      - `MessageBoard.Tests.csproj`
    - `MessageBoard.sln` Haupt-Projektdatei
- CodeContracts
  - ExamPreparation
    - `ExamPreparation`
      - alle Source-Files
      - `ExamPreparation.csproj`
      - `Properties/AssemblyInfo.cs`
    - `UnitTests`
      - alle Source-Files
      - `UnitTests.csproj`
      - `Properties/AssemblyInfo.cs`
    - `ExamPreparation.sln` Haupt-Projektdatei
  - MessageBoard
    - `MessageBoard`
      - alle Source-Files
      - `MessageBoard.csproj`

    - `Properties/AssemblyInfo.cs`
  * `MessageBoardTests`
    - alle Source-Files
    - `MessageBoardTests.csproj`
    - `Properties/AssemblyInfo.cs`
  * `MessageBoard.sln` Haupt-Projektdatei