Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 12: Transactional Memory
Spring Semester 2017

Assigned on: **16.05.2017** Due by: **23.05.2017**

Overview

This week's assignment is about Transactional Memory. Transactional Memory tries to simplify synchronization for the programmer and place it in a hardware or software system. This assignment is intended to give you a hands-on introduction on how to use transactional memory, which you learned about during the lecture. We will be using using ScalaSTM, which is a library-based implementation of software transactional memory. (Note: despite the name you will not program in Scala, but rather still write code using the Java API.)

Exercise 1 – Circular Buffer with STM

For this exercise you are asked to implement a bounded queue, similar to `ArrayBlockingQueue` from the Java concurrency library which was used in the Bakery Simulator scenario. You will need to implement a few functions as listed below:

```java
interface CircularBuffer<E> {
    // Adds item to the queue.
    // If full, this blocks until a slot becomes free.
    public void put(E item);

    // Retrieves the oldest item from the queue.
    // If empty, this blocks until an item is present.
    public E take();

    public boolean isEmpty();
    public boolean isFull();
}
```

We now briefly explain how to use the ScalaSTM framework. The main idea is that you declare the critical sections in your code as atomic blocks:

```java
class Account {
  ...
    void withdraw(int amount) {
        STM.atomic(new Runnable() { public void run() {
            // critical section -> executed atomically
            }
        });
    }
  ...
}
```

What the STM implementation does is that it tracks all memory accesses (read/write) done inside the atomic block. Once the transaction completes, the framework checks for interference, makes sure all values are consistent and either applies them all atomically (*commit*) or rolls back the entire block and retries again later (*abort*). For this reason, all shared data objects should be either immutable or they need to be wrapped in a `Ref`. An example of how to create this in Java is:

```
Ref.View<Integer> count = STM.newRef(0);
```

Warning: it is important you do not access a `Ref` from outside a transaction (i.e. only from *within* an atomic block). In Scala, this can be enforced by the compiler but the same is not available for Java.

There are many operations you can invoke on a `Ref`. The ones you will need for the exercise are: `.get()`, `.set()` and `STM.increment`. To represent the buffer of items in the queue you can use:

```
TArray.View<E> items = STM.newTArray(capacity);

// ... and to access an element at index 'i' use:
Ref.View<E> item = items.refViews().apply(i);
```

If you need to wait, for instance because the queue is empty or full, call `STM.retry()`. This will block the calling thread until the reader/writer set has changed, at which point the transaction will be restarted automatically.

For more information have a look at the API documentation:

http://nbronson.github.io/scala-stm/api/0.7/index.html#scala.concurrent.stm.japi.STM$

There is also a section with examples and explanation in the section entitled “Start Here” on the homepage:

http://nbronson.github.io/scala-stm/

Submission

In order for us to grade your exercises and give you feedback, you need to submit your code to the Subversion repository. You will find detailed instructions on how to install and set-up Eclipse for use with Subversion in Exercise 1.

Once you have completed the skeleton, commit it to SVN in a directory named `assignment12` by following the steps described below. The questions that require written answers should all be recorded in a single file named `report.pdf` and placed in the base directory of your project (i.e., in folder `assignment12`).

- **Check-in your project for the first time**
  - Right click your created project called **assignment12**.
  - In the menu go to **Team**, then click **Share Project**.
  - In the dialog that now appears, select **SVN** as a repository type, then click **Next**.
  - In case you have submitted Exercise 1, choose **Use existing repository location** and select the pre-defined URL in the dialog that should look like this
    https://svn.inf.ethz.ch/svn/vechev/pprog17/students/NETHZ_USERNAME
    Click Finish. Otherwise follow the steps in Exercise 1 to set-up a repository location.
- **Commit changes in your project**

- Now that your project is connected with the SVN server, you need to make sure that every time you change your code or your report, at the end you submit it to the SVN server as well.
- Right click your project called **assignment12**.
- In the menu go to **Team**, then click **Commit**.
- In the Comment field, enter a comment that summarizes your changes.
- Then, click on **Ok**.