Activity Report for ECOST-STSM-IC1001-190911-011708
Detection of Anomalies in TM Programs

João Lourenço

January 9th, 2012

Abstract

This document reports on my activities for the STSM—Ref. ECOST-STSM-IC1001-190911-011708—at the IBM Research Labs at Haifa, Israel, in the period September 28th to December 19th.

1 Scientific Context

Transactional Memory (TM) is a new paradigm for concurrency control that brings the concept of transactions, widely known from the Databases community, into the management of data located in main memory. TM delivers a powerful semantics for constraining concurrency and provides the means for the extensive use of the available parallel hardware. TM uses abstractions that promise to ease the development of scalable parallel applications by achieving performances close to fine-grained locking while maintaining the simplicity of coarse-grained locking.

While TM may contribute to the development of concurrent programs with fewer errors, its usage does not imply by itself the full correctness of the program. TM programs may still suffer from both low-level and high-level data races. Low-level data races, commonly called simply by data races, result from the inadequate use of a protection mechanism when accessing a shared memory location, such as forgetting to define a code block as atomic.

Data consistency is violated when two synchronized blocks have a non-synchronized (possibly empty) code block between them, and the programer intended to have those two synchronized code blocks running atomically, but mistakenly believed it was sufficient to ensure their individual atomicity. This anomaly is often referred as *atomicity violation* or *high-level data race* (HLDR).

The concept of what in a HLDR in concurrent programs is hard or even impossible to define precisely (sound and complete), I have been collaborating with the group of Dr. Eitan Farchi from IBM Research Labs at Haifa (IBM HRL)in the last 2 years investigating algorithms and techniques for static analysis that, by using a best effort strategy, are able to improve the quality of the HDLR detectors for concurrent (Java) transactional memory programs.

2 Work Plan

In August 3rd, 2011, I submitted a grant request to the IC1001 COST for a STSM at IBM Research Labs at Haifa, Israel, for the period of 3 months, from September 19th to December 19th, 2011, aiming at strengthening the ongoing collaboration with Dr. Eitan Farchi and his team. The notification of the pre-approval of the grant was received in October 5th, 2011. A working VISA request in my name was submitted to the appropriate authorities by IBM Haifa in

late July 2011. Unexpected delays with the VISA process forced me to postpone the departure date from September 19th to September 28th, reducing the stay to a total of 85 days.

The proposed work plan for the STSM included:

1. Detecting HLDR in more complex Java Bytecode programs, such as ones with dynamic binding;
2. Studying how to reduce even further the number of false positives and negatives by possibly applying techniques such as pointer analysis;
3. Dealing with the scaling of the problem size in terms of lines of code and number of concurrent threads;
4. Building and documenting a repository of buggy Transactional Memory programs, with both small synthetic examples and larger full program benchmarks.

3 Report of activities

The planned activities for the STSM period were slightly changed to take better advantage of the physical proximity between the researchers, focusing on discussing new ideas instead of simple improvements over already ongoing work. Hence, the activities at IBM Haifa covered three main topics:

Organization of HVC'11 HVC'11 (`http://www.research.ibm.com/haifa/conferences/hvc2011/`) is an International Conference in the scope of the work plan of the STSM, and took place in IBM HRL in December 6-8, 2011. Together with the other Co-Program Chair, Kerestin Eder (`http://www.cs.bris.ac.uk/~eder/`) and the Conference Chair Onn Shehory (`http://u.cs.biu.ac.il/~shechory/`), I worked in the local organization of the conference, which counted with 158 technical papers and near 70 participants. Similarly to previous editions, the conference will have post-proceedings published by Springer-Verlag in the LNCS series.

Repository of buggy TM programs During the period of the STSM, we crated a compilation of a set of micro benchmarks of well buggy programs, the majority well known from related scientific works in in atomicity violations, each accompanied with an appropriate README, describing the objectives/semantics of the benchmark and the known faults, each accompanied with a short description. The repository will be publicly available to the scientific community at `https://projects.fct.unl.pt/di-tm-repository` by February 1st, 2012.

Program Closure Development of a new *Program Closure* concept, which is not restricted to Transactional Memory programs and can be applied to concurrent programs in general. Intuitively, we say a program is *closed* if adding a new tread to the program does not increase the number of HLDRs already existing in that program. This concept can be applied to estimate the potential of future releases of a program to suffer from atomicity violations. It can also be applied to the adaptation of APIs from thread-unsafe to thread-safe, helping identifying atomicity violations resulting from the concurrent usage of API services/methods. See next section for some more details on the *Program Closure* concept.

4 Future of the Collaboration with IBM HRL

The collaboration with Dr. Eitan Farchi group started approximately three years ago, with the aim at working on Testing and Debugging of Concurrent Programs. This general theme collaboration will certainly be pursued in the future, as there is a strong willingness and commitment from both partners in this collaboration.

In the specific topic of Detection of Anomalies in Transactional Memory Programs we plan to pursue the topic of detection of HLDR in concurrent programs in general and in Transactional Memory programs, namely by developing further the concept of *Program Closure*, where we plan to submit a paper to an appropriate venue during the first semester of 2012. Additionally, we also plan to continue extending the *Repository of buggy TM programs* by incorporating the benchmarks used in future scientific works in this theme.

5 Financial Report

I applied for a grant in the total amount of 3.500 EUR, according to the following rubrics:

Travel	1.000 EUR
Subsistence (hotel/meals)	2.500 EUR
Total	**3.500 EUR**

The travel cost was 737,67 EUR for the plane plus 2×80 EUR for local transportations, in a total amount of 897,67 EUR. This amount was slightly less than the planned 1.000 EUR.

Due to the administrative difficulties with the Working VISA for Israel, the effective STSM was a bit shorter than planned. However, the lodging and living expenses were considerable higher than expected. The lodging cost was 8.970 NIS $\approx$ 1.850 EUR (see receipt attached). The amount of 752,33 EUR requested for living expenses corresponds to 8,85 EUR per day.

Travel	897,67 EUR
Lodging	1.850,00 EUR
Living	752,33 EUR
Total	**3.500,00 EUR**

For all the above, and although the period of stay in Haifa was reduced in 9 days, I kindly request to be awarded with the full amount of 3.500 EUR as initially planned in the Grant Proposal/Request.

6 On the Closure of Concurrent Programs

In this section we make a brief overview of our ongoing work, whose main concept (Program Closure) was originated during the STSM at Haifa.

NOTE: this is a draft with the main ideas/concepts of the ongoing work on Program Closure, and is by no means sound or complete.

6.1 Introduction

Concurrent programs may suffer from different concurrency specific anomalies, among which we find *high-level data races* (HLDR), also known as *atomicity violations*. HLDR result from a

misconception of the software developer of which critical sections (code blocks) must be executed under mutual exclusion with some or all of the other critical sections, defining two or more disjoint critical sections where a single one should have been declared.

Intuitively, we say a program is closed if adding a new tread to the program does not increase the number of HLDRs already existing in that program. This concept can be applied to estimate the potential of future releases of a program to suffer from atomicity violations. It can also be applied to the adaptation of APIs from thread-unsafe to thread-safe, helping identifying atomicity violations resulting from the concurrent usage of API services/methods. See next section for some more details on the *Program Closure* concept.

6.2 Program Closure

We adopt the concept of high level data races [1] as follows. Consider a program P that has m threads $\{t_1, \ldots, t_m\}$. The views associated with thread t_i $(1 \leq i \leq m)$ are denoted by $V_1^i, \ldots V_{n_i}^i$, where $V_{1 \leq n \leq n_i}^i$ is the set of variables accessed by thread t_i under synchronized block n. We say that two views associated with t_i, V_j^i and V_l^i, are depended if there is some execution of P for which V_j^i is executed and then V_l^i is executed and $V_j^i \cap V_l^i \neq \emptyset$[1]

For each maximal length path $p = (V_1^i, \ldots, V_k^i)$ in D_i that does not include repeating views, we create a new thread t_p that is associated with a single view $CS_p = V_1^i \bigcup V_2^i \bigcup \ldots \bigcup V_k^i$. CS_p is also referred to as the closure set associated with the maximal path p and the thread t_p. Intuitively, the maximal path is suspected to be a transaction due to the dependencies between each pair of consecutive views.

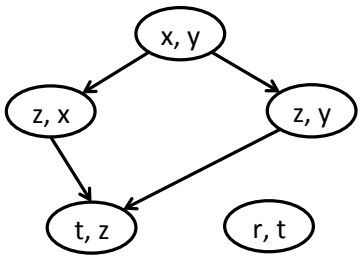
Note that each thread we are adding to the program has only one view. As a result, the interaction of any two newly added threads will never add new high level data races to the program.

We define the closure of P, $clos(P)$, to be the program obtained from P by adding such threads, t_p, for each maximal length path in D_i and for each thread in P. The set of high level data races in P is contained in the set of high level data races in $clos(P)$. If the set of high level data races in P and $clos(P)$ are equal we say that P is a closed program.

6.3 Example 1

T_0

```
atomic { x = y; }
if (cond_1)
    atomic { z = x; }
else
    atomic { z = y; }
if (cond_2)
    atomic { t = z; }
else
    atomic { r = t; }
```



$CS_1 = \{x, y, z, t\}$
$CS_2 = \{x, y, z, t\}$

Consider a program P_1 with **a single thread** executing the code T_0 above. The closure sets are $CS_1 = CS_2 = \{x, y, z, t\}$. So, to *close* the program we must add a new thread T_1 accessing the variables in CS_1. Note that the intersection of $\{r, t\}$ with $\{z, x\}$ and with $\{z, y\}$ is empty so

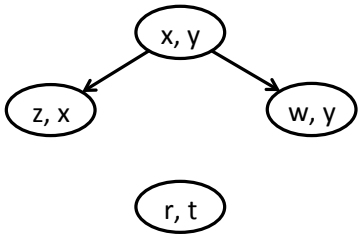
[1]This is an instance of the *suspectedAtomic*() function of **??**.

there is no dependency between $\{z, x\}$ and $\{r, t\}$ and no dependency between $\{z, y\}$ and $\{r, t\}$, although they have a control flow relation.

6.4 Example 1a

T_0

```
atomic { x = y; }
if (cond_1)
    atomic { z = x; }
else
    atomic { w = y; }
atomic { r = t; }
```



$CS_1 = \{x, y, z\}$
$CS_2 = \{x, y, w\}$

Consider a program P_{1a} with **a single thread** executing the code T_0 above. $clos(T_0)$ has two new threads T_1 and T_2, with views $V_1^1 = CS_1$ and $V_1^2 = CS_2$ respectively.

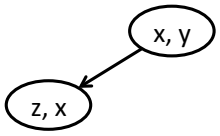
T_0	T_1	T_2
$V_1^0 = \{x, y\}$	$V_1^1 = \{x, y, z\}$	$V_1^2 = \{x, y, w\}$
$V_2^0 = \{z, x\}$		
$V_3^0 = \{w, y\}$		
$V_4^0 = \{r, t\}$		

The new program $clos(T_0)$ has now two HLDR: one from $V_1^0 + V_2^0$ with V_1^1, and another of $V_1^0 + V_3^0$ with V_1^2.

6.5 Example 1-b

T_1

```
atomic { x = y; }
if (cond_1)
    atomic { z = x; }
```



$CS_1 = \{x, y, z\}$

Consider a program P_{1b} with multiple threads executing the code above. This program has no high level data races. Each thread in P_{1b} has only one maximal path in its control flow $(\{x, y\}, \{z, x\})$ so we will add only one thread for each thread having the view $\{x, y, z\}$. The new thread has a high level data race with the original program thread as $\{x, y\}$ and $\{x, z\}$ do not form a chain.

6.6 Example 2

T_1

```
atomic { x = y; }
if (cond_1)
    atomic { z = x; }
else
    atomic { z = y; }
if cond_2)
    atomic { t = z; }
else
    atomic { r = t; }
```

T_2

```
atomic { x = y + z + t; }
```

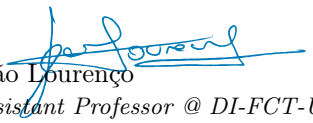
T_3

```
atomic { x = y + z + t; }
```

Consider the program P_2 above obtained from example 1 by the closure process. All the closure sets introduced by $clos(P_2)$ will be equal to an already existing view in P_2, hence no new views will be introduced. This is an example of Lemma 1 below ("$clos(clos(P))$ is always closed!")

References

[1] C. Artho, K. Havelund, and A. Biere. High-level data races, 2003.



João Lourenço
Assistant Professor @ DI-FCT-UNL