Automatically Fixing Data Races in Parallel Java Programs using Privatization

Caius Brindescu, “Politehnica” University of Timișoara

ADVISOR: Danny Dig

Background
The multicore revolution is forcing software to change.
Manually parallelizing existing software is time-consuming and error-prone.
There is a need for software tools that can make that transformation semi-automatically.

Goals
Create a refactoring tool that can privatize certain fields in a Java program.
Compare the results of that refactoring with solutions applied by programmers on real-world programs.

Fundamental Questions/Challenges
How to privatize fields such that:
✓ Races are eliminated
✓ The overhead introduced is minimal
What is the optimal privatization technique?

Research Plan
This is the workflow of a tool that determines the optimal way to privatize a given field. We focused on the areas in blue.

Research Results
Currently we have implemented a refactoring tool that privatizes a field using Java’s built-in ThreadLocal.
We also analyzed three Java programs and looked at the techniques used to make them thread-safe.

Related Work/Interaction with Other Projects
The refactoring was created using the Eclipse Refactoring Framework.
The race detection is part of the ReLooper tool, developed by Cosmin Rădoi and Danny Dig [1].


Project Name | Privatization | Synchronization | Java concurrent collections
--- | --- | --- | ---
AOI | 10 | 1 | 0
VASSAL | 3 | 0 | 4
JMol | 4 | 0 | 0

A possible solution by privatization via Thread Local:

```
ParallelArray particles;
int x; // there is a race on x
Particles.apply(new Procedure() {
    public void apply(Particle p) {
        x = p.getWeight();
        p.process(x);
    }
});
```

A possible solution by privatization via Thread Local:

```
ParallelArray particles;
ThreadLocal<Integer> x; // the race was fixed
Particles.apply(new Procedure() {
    public void apply(Particle p) {
        x.set(p.getWeight());
        p.process(x.get());
    }
});
```