Short description

The task is to implement a program analysis tool for concurrency-related errors like deadlocks and races.

Background and motivation

Writing bug-free, safe, software in a modern language like Java can be quite a challenge: apart from getting the business logic of what is to be implemented right, null-pointer accesses and other bugs get in our way. For concurrent/multi-threaded software, we face yet another challenge: firstly, we need to figure out where to use concurrency primitives like locking to protect shared data. Design decisions have a big influence on performance here — of course we could just use one "giant" lock to protect resources, but then we'd almost end up with a sequential application again. More critically, we may accidentally introduce deadlocks or data-races, which will make our application get stuck or produce garbled results.

At PMA, we have developed a basic theory about how to analyse such software (we're certainly not the first; there's lots of existing theory and even commercial tools). Now, the challenge is to turn this into a usable tool for Java (analysing synchronized blocks, or locking using java.util.concurrent) or for C-programs using the pthreads-API. Another alternative would be the Go language.

Problem setting

A prospective student would try to use an existing program analysis framework like Soot to

- implement the analysis,
- collect performance results on scalability,
- run experiments on existing open-source software projects,
- contribute to publications/conference submissions.

It may be possible to visit TU Darmstadt (Germany) during the thesis, within the GoRETech-project.

Keywords: program analysis, concurrency, verification

References
