Master Thesis/Diplomarbeit
Lock Based Static Detection of Race Conditions in Parallel Programs

Background

The Bauhaus Project develops methods and tools to support and ease program understanding in the process of software maintenance. Since the founding in 1996, an extensive infrastructure was built that allows the execution of program analyses with low and high levels of abstraction.

The IML (InterMediateLanguage) is a detailed, programming language independent representation for programs in the form of an enhanced syntax tree. Manifold program analyses are performed on the basis of this representation, e.g. control flow and data flow analyses. The IML-library is implemented in Ada95 and provides operations for the generation and manipulation and traversal of the IML.

Task

In parallel programs tasks interact with each other to perform assigned jobs. The interaction is achieved through shared resources and communication. To ensure correct operation of these interactions, several methods for the synchronisation of threads and processes like semaphores, condition variables, critical regions and monitors were developed.

The task of this thesis is to implement an algorithm that statically detects definite and potential race conditions caused by unsynchronized access to shared data. The algorithm must be based on lock mechanisms. Therefore, it should concentrate on mutex and condition variables. The implementation must be based on IML. Further existing base analyses like control flow analysis and flow insensitive pointer analysis may be used. An API for the detection of thread creations and accesses to mutex and condition variables will be provided.

Those are the tasks that must be accomplished:

- A literature search for existing algorithms for race conditions detection must be done.
- A lock based algorithm that statically detects all potential race conditions must be implemented and analysed. The algorithm should also report the direct control dependencies for potential races. Further, the races must be categorised with respect to feasibility and actuality.
- The runtime behaviour of the algorithm must be theoretically evaluated.
• The algorithm must be implemented using the provided tools and APIs.

• Extensive tests must be performed to show the correctness of the implementation. The analysis must be applied to several large programs. The tutor will provide some of these programs, but the student is also obligated to search for his own test programs. Further, the student will receive prepared test programs where races are hidden. The analysis should be able to detect all of these race conditions. The student is also encouraged to implement his own suite of test cases for the complete test of all features of the analysis.

• The precision of the analysis must be evaluated. An important question to be answered is whether the existing base analyses (e.g. flow insensitive pointer analysis) are sufficient for the detection of race conditions in real world programs, i.e. that all potential race conditions are detected and the number of false positives is low. Further strategies to improve the results should be investigated.

The requirements for the quality of the implementation and reasonable code documentation are high. The rules of the Bauhaus Coding style for Ada programs must be observed.

Available tools and programming language

Bauhaus tool suite, APIs and libraries of Bauhaus, GNAT Ada95 compiler

Tutor

Gunther Vogel (1.205) Aoun Raza (1.240)
phone: +49-711-7816-375 phone: +49-711-7816-399
gunther.vogel@informatik.uni-stuttgart.de aoun.raza@informatik.uni-stuttgart.de