Tapir: Language Support to Reduce the State Space in Model-Checking

Ronald Veldema Michael Philippsen

University of Erlangen-Nuremberg
Computer Science Department 2
Martensstr. 3 • 91058 Erlangen • Germany

veldema@cs.fau.de philippsen@cs.fau.de

Model-checking is a way of testing the correctness of concurrent programs. We propose a new language (Tapir) especially suited for model-checking. To model-check, a model of the program is proven to match properties and constraints specified by the programmer. The model itself is created by disregarding irrelevant program details which makes the program simpler and thereby faster and easier to analyze. There are currently two ways in which to create a model. One can either write a new program in another language or simplify the program in some way. Tapir does the latter.

To make model-checking feasable, Tapir allows two optimizations. First, the programmer can provide application specific program transformations that reduce the state space. This is possible because context-switches between threads and processes are visible at language level. Second, Tapir allows the programmer to select pieces of code to abstract away from. This is better than writing a completely new program (in another language) as that can cause the model and its implementation code to diverge. Also, if the application code is changed, the model needs to be changed as well. In the end it becomes hard to ensure that a fix in the model can also fix the application. In our approach we allow the programmer to simplify the application and allow him the choice of what program features are 'irrelevant details' by allowing him to encapsulate them into black-box code pieces. For selected classes and methods, the programmer can provide an alternative implementation: a slim black box version for use in model-checking that abstracts away many details of the full fledged implementation.

Finally, to allow model-checking of library code at all, a 'driver' must be supplied that calls the library functions. A single model-checking run then proves the library correct in respect to that driver. For good testing, we need many versions of the driver applied in different ways. We do so by a novel aspect-oriented test case generation method. Tapir's aspect-oriented test case generation combined with black-boxing and programmer supplied optimizations allows model-checking of low-level library code.

Using these techniques, a small benchmark is model-checked 3.7 times faster using 2.7 times less memory.