CAVEAT : a Tool for Software Validation

P.Baudin - A.Pacalet - J.Raguideau - D.Schoen - N.Williams
CEA Saclay, DTSI-SLA, 91191 Gif sur Yvette Cedex, France
dominique.schoen@cea.fr

Abstract

Caveat is a static analysis tool designed to help verify safety critical software. It operates on ANSI C programs. It was developed by CEA, the French nuclear agency and is used as an operational tool by Airbus-France and EdF, the French electricity company. It is mainly based on Hoare Logic and rewriting of first order logic predicates. The main features of Caveat are property synthesis, navigation facilities, and proof of properties.

1 Features

The property synthesis is the first phase performed by the tool and is fully automatic. It computes the call graph of functions, extracts the operands for each function, generates possible threats of execution crash. The computed operands are not only explicit operands but also the implicit ones, i.e. global variables used locally and by the called functions. The dependencies between inputs and outputs are computed as well as, if possible, functional expressions linking outputs to inputs. The threat generation consists of the detection of certain unsafe operations such as division by zero, dereferencing of null pointers or out of range array access. The pre-conditions for these threats to be avoided are computed locally and then propagated through the call graph. If a pre-condition is proved to be satisfied at some level, the associated threat will not occur. Navigation facilities, such as browsing facilities or visualisation of graphs, are provided. The call graph displays the structure of the application and is colored to show the results of threat generation.

Property proof combines interactive and automatic operations: the user adds properties such as pre or post-conditions and asks for their proof. If the result is not true, meaning that the property was not automatically proven, the cause can be either a weakness of the theorem prover or missing information; in both cases the user is provided with the remaining condition; in the first case, under the control of the tool, he can interactively choose which transformations to apply to the condition until the proof can be performed automatically. In the second case, the remainder contains the missing information, so the user can identify it and adds it to go on with the verification process.

The implementation of Hoare Logic in Caveat gives good results in the verification of critical software for two main reasons: the first one is that critical applications generally impose restrictions on the use of the C language: no recursion, no alias, no dynamic use of memory. Furthermore, programming must respect rules of good practice which results in a modular architecture and small modules. The second one is that the aim is to ensure that functional, safety or robustness properties given by the user are verified and not to prove a complete functional specification. Caveat is based on the principle that it is not necessary to know everything about all functions to achieve a proof. Verification work with Caveat is an iterative process: at each step, the tool indicates just what is needed to go further. In spite of its theoretical limits, this technique gives very good results when used with a pragmatic approach.

2 Industrial use and future work

Features designed for industrial use have been added to allow re-use of previous work, writing proof plans, re-running of previous sessions. EdF uses Caveat to evaluate the dependability of applications coming from outside sources while Airbus decided, after an experimental phase, to include Caveat in the operational software development process to validate pieces of code as early as possible.

We are now working on features to extend the area of available applications, increase the automation of the proofs, and reduce the complexity of the computation. Aliases are not treated in the current version; it is necessary to address this feature to cope with a wider class of applications. Loops are a key point in a tool based on Hoare Logic. The current method used for loops is too constraining to obtain good results in the automatic proof. We are now adding abstract interpretation and making the two techniques cooperate. We plan to add slicing and propagation of hypotheses to reduce the code to consider during the computation.