AIAA Infotech@Aerospace 2010
20 - 22 April 2010, Atlanta, Georgia

AIAA 2010-3385

Static Analysis and Verification of Aerospace Software by Abstract Interpretation

(Abstract)

Julien Bertrane*
École normale supérieure, Paris

Patrick Cousot*,†
Courant Institute of Mathematical Sciences, NYU, New York & École normale supérieure, Paris

Radhia Cousot*
École normale supérieure & CNRS, Paris

Jérôme Feret*
École normale supérieure & INRIA, Paris

Laurent Mauborgne*,‡
École normale supérieure, Paris & IMDEA Software, Madrid

Antoine Miné*
École normale supérieure & CNRS, Paris

Xavier Rival*
École normale supérieure & INRIA, Paris

The validation of software checks informally (e.g., by code reviews or tests) the conformance of the software executions to a specification. More rigorously, the verification of software proves formally the conformance of the software semantics (that is, the set of all possible executions in all possible environments) to a specification. It is of course difficult to design a sound semantics, to get a rigorous description of all execution environments, to derive an automatically exploitable specification from informal natural language requirements, and to completely automatize the formal conformance proof (which is undecidable). In model-based design, the software is often generated automatically from the model so that the certification of the software requires the validation or verification of the model plus that of the translation into an executable software (through compiler verification or translation validation). Moreover, the model is often considered to be the specification, so there is no specification of the specification, hence no other possible conformance check. These difficulties show that fully automatic rigorous verification of complex software is very challenging and perfection is impossible.

We present abstract interpretation[1] and show how its principles can be successfully applied to cope with the above-mentioned difficulties inherent to formal verification.

- First, semantics and execution environments can be precisely formalized at different levels of abstraction, so as to correspond to a pertinent level of description as required for the formal verification.
- Second, semantics and execution environments can be over-approximated, since it is always sound to consider, in the verification process, more executions and environments than actually occurring in real executions of the software. It is crucial for soundness, however, to never omit any of them, even rare events. For example, floating-point operations incur rounding (to nearest, towards 0, plus or minus infinity) and, in the absence of precise knowledge of the execution environment, one must consider the

*École normale supérieure, Département d'informatique, 45 rue d'Ulm, 75230 Paris cedex 05, First.Last@ens.fr.
†Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street New York, N.Y. 10012-1185, pcousot@cs.nyu.edu.
‡Fundación IMDEA Software, Facultad de Informática (UPM), Campus Montegancedo, 28660-Boadilla del Monte, Madrid, Spain.

worst case for each float operation. Another example is inputs, like voltages, that can be overestimated by the maximum capacity of the hardware register containing the value (anyway, a well-designed software should be defensive, i.e., have appropriate protections to cope with erroneous or failing sensors and be prepared to accept any value from the register).

- In the absence of an explicit formal specification or to avoid the additional cost of translating the specification into a format understandable by the verification tool, one can consider implicit specifications. For example, memory leaks, buffer overruns, undesired modulo in integer arithmetics, float overflows, data-races, deadlocks, live-locks, etc. are all frequent symptoms of software bugs, which absence can be easily incorporated as a valid but incomplete specification in a verification tool, maybe using user-defined parameters to choose among several plausible alternatives.

- Because of undecidability issues (which makes fully automatic proofs ultimately impossible on all programs) and the desire not to rely on end-user interactive help (which can be lengthy, or even intractable), abstract interpretation makes an intensive use of the idea of abstraction, either to restrict the properties to be considered (which introduces the possibility to have efficient computer representations and algorithms to manipulate them) or to approximate the solutions of the equations involved in the definition of the abstract semantics. Thus, proofs can be automated in a way that is always sound but may be imprecise, so that some questions about the program behaviors and the conformance to the specification cannot be definitely answered neither affirmatively nor negatively. So, for soundness, an alarm will be raise which may be false. Intensive research work is done to discover appropriate abstractions eliminating this uncertainty about false alarms for domain-specific applications.

We report on the successful cost-effective application of abstract interpretation to the verification of the absence of runtime errors in aerospace control software by the ASTRÉE static analyzer,[2] illustrated first by the verification of the fly-by-wire primary software of commercial airplanes[3] and then by the validation of the Monitoring and Safing Unit (MSU) of the Jules Vernes ATV docking software.[4]

We discuss on-going extensions to imperfectly synchronous software, parallel software and target code validation, and conclude with more prospective goals for rigorously verifying and validating aerospace software.

References

[1]Cousot, P. and Cousot, R., "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM Press, New York, Los Angeles, 1977, pp. 238–252.

[2]Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., and Rival, X., "Varieties of Static Analyzers: A Comparison with ASTRÉE, invited paper," *Proceedings of the First IEEE & IFIP International Symposium on Theoretical Aspects of Software Engineering, TASE'07*, edited by M. Hinchey, H. Jifeng, and J. Sanders, IEEE Computer Society Press, Los Alamitos, Shanghai, 6–8 June 2007, pp. 3–17.

[3]Delmas, D. and Souyris, J., "ASTRÉE: from Research to Industry," *Proceedings of the Fourteenth International Symposium on Static Analysis, SAS'07*, edited by G. Filé and H. Riis-Nielson, Kongens Lyngby, Lecture Notes in Computer Science 4634, Springer, Berlin, 22–24 August 2007, pp. 437–451.

[4]Bouissou, O., Conquet, E., Cousot, P., Cousot, R., Feret, J., Goubault, E., Ghorbal, K., Lesens, D., Mauborgne, L., Min, A., Putot, S., Rival, X., and Turin, M., "Space Software Validation using Abstract Interpretation," *The International Space System Engineering Conference DASIA 2009, Data Systems In Aerospace*, edited by E. publications, Istambul, Turkey, 26–29 May 2009.