Integrating Physical Systems in the Static Analysis of Embedded Control Software*

Patrick Cousot

École Normale Supérieure, Paris, France
Patrick.Cousot@ens.fr
www.di.ens.fr/~cousot

Abstract Interpretation

Abstract interpretation is a theory of effective abstraction and/or approximation of discrete mathematical structures as found in the semantics of programming languages, modelling program executions, hence program properties, at various levels of abstraction [3,7,8,10,12].

Static Analysis by Abstract Interpretation

The prominent practical application of abstract interpretation has been to static program analysis, that is the automatic (without any human intervention), static (at compile time) determination of dynamic program properties (that always hold at runtime) involving complex abstractions of the infinite state operational semantics (e.g. [4,5,9,11]). Abstract interpretation fights undecidability and complexity by approximation of the program execution model which may lead to false alarms in correctness proofs. This happens whenever the combination of the abstract domains involved in the analyzer is not precise enough to express any inductive argument necessary in the correctness proof. Hence, among other possible alternatives, the idea to specialize static analyzers to well-defined families of programs and properties for which abstract domains can be designed to express all information necessary to perform inductive proofs [6].

Static Analysis of Embedded Control Software

This approach was successfully illustrated by the ASTRÉE static analyzer which is specialized for proving the absence of run-time errors in synchronous, time-triggered, real-time, safety critical, embedded software written or automatically generated in the C programming language [1,2,13]. It was able to prove the absence of run-time errors in large industrial avionic control-command programs [14]. It is a remarkable well-design criterion that the absence of runtime errors can be proved in such control/command software without any hypotheses on the controlled systems (but, maybe, for ranges of variation of very few volatile input variables). This means that the software will go on functioning without any

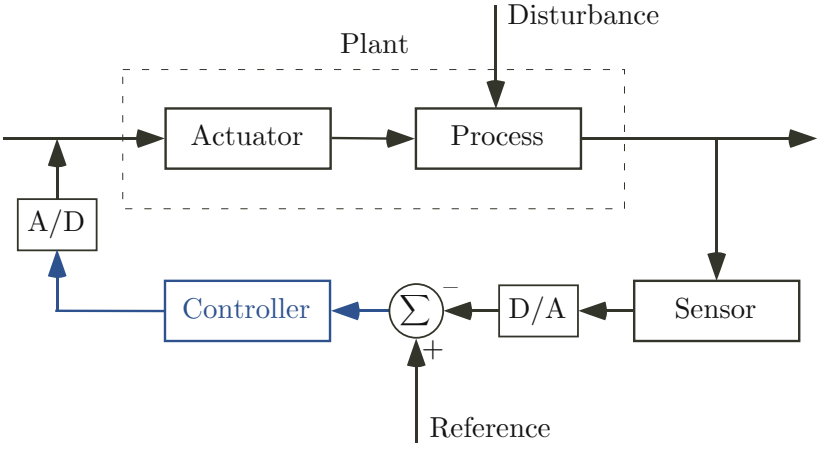
* This work was supported in part by the Jerome Clarke Hunsaker Visiting Professorship of the MIT Aeronautics and Astronautics Department in 2005. I thank John Deyst and Éric Féron for stimulating discussions.

K. Yi (Ed.): APLAS 2005, LNCS 3780, pp. 135–138, 2005.

runtime error whichever the behavior of the controlled system can be, as long as the processor on which the program is running does not fail (a situation which can be handled by fault-tolerance techniques [15]). Obviously not all desirable properties of the controlled physical system can be proved in this way by a very coarse abstraction of the properties of this physical system.

Integrating Physical Systems in the Static Analysis of Embedded Control Software

To go beyond, e.g. to prove robustness or stability, is is necessary to take into account the full feedback control system that is the controller (from which the control/command program was generated) but also the mathematical model of the physical system (either in the form of differential equations, difference equations or of a numerical model as given e.g. in Simulink^{TM}):



We advocate an approach in which code is generated by discretization both for the continuous dynamic nonlinear model of the controlled system (e.g. from the block diagram description of the plant, actuators and sensors) and for the digital implementation of the controller (as given by the control/command program to be verified).

This code can be that of a specification language when reasoning at the model level or that of a programming language when reasoning at the implementation model.

A static analysis of this code can provide information (like reachability sets) which can hardly be discovered by traditional simulation or test techniques. Such numerical simulations also involve discretization techniques and floating-point computations which might not be the same as those involved in the generated control/command program. Such intricate differences would disappear in an integrated approach.

Taking the environment of execution of the control/command program into account allows for more refined properties of this control/command program to be proved such as reachability in the actual context of use, reactivity, stability,

uncertainty and robustness, performance validation, etc of feedback control. Such properties are not always easily expressible as traditional temporal properties commonly used in computer science correctness proofs.

By static analysis, such refined properties can be verified from the more or less idealized and precise model of the controller and plant down to the actual embedded control program. Information can be translated between levels of refinement to ease static analysis or checking at lower levels and to ensure coherence and soundness of the inferred information at all levels of refinement. The verification is thus performed from the model to the derived program with respect to the full specification of the execution environment. A central advantage of this integrated approach is the potential for early discovery of design errors much before the costly experimentations on an actual physical implementation.

Convex Abstractions

We present new abstract interpretations and abstract domains issued from modern control theory and convex optimization as a first step towards reaching these ambitious objectives of integrating physical systems in the static analysis of embedded control software.

References

1. The ASTRÉE Static Analyzer. `www.astree.ens.fr`.
2. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A Static Analyzer for Large Safety-Critical Software. *Proc. ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation* (PLDI'03), San Diego, California , USA, June 7–14, 2003, pp. 196–207. ACM Press, 2003.
3. P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 21 Mar. 1978.
4. P. Cousot. Types as abstract interpretations, invited paper. In *24th POPL*, pages 316–331, Paris, Jan. 1997. ACM Press.
5. P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*, volume 173, pages 421–505. NATO Science Series, Series F: Computer and Systems Sciences. IOS Press, 1999.
6. P. Cousot. Partial completeness of abstract fixpoint checking, invited paper. In B. Choueiry and T. Walsh, editors, *Proc. 4th Int. Symp. SARA '2000*, Horseshoe Bay, LNAI 1864, pages 1–25. Springer, 26–29 Jul. 2000.
7. P. Cousot. Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. In *Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, pages 1–24, Paris, France, January 17-19, 2005. Lecture Notes in Computer Science, volume 3385, Springer, Berlin.
8. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, 1977. ACM Press.

9. P. Cousot and R. Cousot. Static determination of dynamic properties of generalized type unions. In *ACM Symposium on Language Design for Reliable Software*, Raleigh, ACM SIGPLAN Not. 12(3):77–94, 1977.
10. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *POPL*, pages 269–282, San Antonio, 1979. ACM Press.
11. P. Cousot and R. Cousot. Invariance proof methods and analysis techniques for parallel programs. In A. Biermann, G. Guiho, and Y. Kodratoff, editors, *Automatic Program Construction Techniques*, chapter 12, pages 243–271. Macmillan, 1984.
12. P. Cousot and R. Cousot. Basic concepts of abstract interpretation. In P. Jacquart, editor, *Building the Information Society*, chapter 4, pages 359–366. Kluwer Acad. Pub., 2004.
13. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The astrée analyser. In M. Sagiv, editor, *Proc. 14^{th} ESOP '2005, Edinburg, UK*, volume 3444 of *LNCS*, pages 21–30. Springer, Apr. 2-10, 2005.
14. J. Souyris. Industrial experience of abstract interpretation-based static analyzers. In P. Jacquart, editor, *Building the Information Society*, chapter 4, pages 393–400. Kluwer Acad. Pub., 2004.
15. P. Traverse, I. Lacaze, and J. Souyris. Airbus ly-by-wire — a total approach to dependability. In P. Jacquart, editor, *Building the Information Society*, chapter 3, pages 191–212. Kluwer Acad. Pub., 2004.