SPACE SOFTWARE VALIDATION USING ABSTRACT INTERPRETATION

Olivier Bouissou[(1)], Eric Conquet[(2)], Patrick Cousot[(3)], Radhia Cousot[(3)(6)], Jérôme Feret[(3)(7)], Khalil Ghorbal[(1)], Eric Goubault[(1)], David Lesens[(4)], Laurent Mauborgne[(3)], Antoine Miné[(3)(6)], Sylvie Putot[(1)], Xavier Rival[(3)(7)], Michel Turin[(5)]

[(1)] CEA LIST, MeASI, Point Courrier 94, Gif-sur-Yvette, F-91191 France, Email: FirstName.LastName@cea.fr
[(2)] ESA - ESTEC, Noordwijk - The Netherlands, Email: eric.conquet@esa.int
[(3)] ENS, DI, ENS, 45, rue d'Ulm, 75230 Paris Cedex 5, France, Email: FirstName.LastName @ens.fr
[(4)] Astrium ST, 66, route de Verneuil, 78133 Les Mureaux, France, Email: david.lesens@astrium.eads.net
[(5)] GTI6, 33, avenue Kennedy, 91300 Massy, France, Email: michelturin@GTI6.com
[(6)] CNRS, France
[(7)]INRIA Paris-Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay, France

ABSTRACT

This paper reports the results of an ESA funded project on the use of abstract interpretation to validate critical real-time embedded space software. Abstract interpretation is industrially used since several years, especially for the validation of the Ariane 5 launcher. However, the limitations of the tools used so far prevented a wider deployment. Astrium Space Transportation, CEA, and ENS have analyzed the performances of two recent tools on a case study extracted from the safety software of the ATV:

- ASTRÉE, developed by ENS and CNRS, to check for run-time errors,
- FLUCTUAT, developed by CEA, to analyse the accuracy of numerical computations.

The conclusion of the study is that the performance of this new generation of tools has dramatically increased (no false alarms and fine analysis of numerical precision).

1. INTRODUCTION

As recent NASA mission failures illustrate, any single error in critical software can have catastrophic consequences. More than half of all satellite failures from 2000 to 2003 involved software. Even though failures are usually not advertised, some software bugs have become famous, such as the error in the MIM-104 Patriot.

One use of abstract interpretation techniques is to improve the confidence and reduce the cost of software validation. Software validation is a difficult and costly activity representing more than half of the total development cost. Software validation is the last development step, but, unfortunately, testing and code review, the most widely deployed verification methods, suffer from severe shortcomings. Both methods are very time consuming and labour intensive processes. For most critical systems, 50% of the overall development costs are allocated to testing. In fact, it is not practically feasible to hunt down to the last bug. In short, as E. W. Dijkstra puts it: Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.

One of the most promising technical axes practised since several years by ASTRIUM is the use of static analysis [3]. INRIA has developed around 1993 a tool, IABC, based on academic studies to detect run-time errors (e.g., arrays out of bound, overflow, zero divide, etc.); this tool has then been scaled for Ariane 5 ADA products, and industrialized by Polyspace Technologies during the following years (now The MathWorks). This tool has provided some good help at that time and is still in use for all critical software developed by ASTRIUM-ST.

But with this first generation of abstract interpretation-based static analysis tools, it remains difficult (indeed impossible) to avoid false alarms with floating-point operations and iterative algorithms. So, the use of these techniques which was planned for the development of the ATV safety software (MSU software), has been finally abandoned due to the high number of false alarms raised by the tool on floating-point operations.

"Space Software Validation using Abstract Interpretation" (SSVAI) is an ESA project which had the objective to investigate the use of abstract interpretation-based static analysis techniques to improve the validation of space critical embedded software applied to numerical algorithms for which other tools have not provided satisfactory results.

Two tools have been studied:

- ASTRÉE: Analyse Statique de logiciels Temps-RÉel Embarqués (Static Analysis of Real-Time Embedded Software) [1,4]. This tool, developed by the École Normale Supérieure and the CNRS, aims at automatically proving the absence of run-time errors, such as division by zero, out of range array indexes, arithmetic overflows, etc.
- FLUCTUAT: This tool, developed by the CEA, aims at analysing the numerical precision and stability of complex algorithms [5].

2. WHAT IS ABSTRACT INTERPRETATION?

The formal verification of a program (and more generally a computer system) consists in proving that its semantics (describing "what the program executions actually do") satisfies its specification (describing "what the program executions are supposed to do").

Abstract Interpretation [3] formalizes the idea that this formal proof can be done at some level of abstraction where irrelevant details about the semantics and the specification are ignored. This amounts to proving that an abstract semantics satisfies an abstract specification.

Abstractions shall be sound (no conclusion derived from the abstract semantics is wrong relative to the program concrete semantics and specification). Abstractions should also preferably be complete (no aspect of the semantics relevant to the specification is left out). In the considered applications, which tackle undecidable program properties, completeness is impossible. Hence, the objective is to minimize false alarms on a specific family of programs while keeping a reasonable analysis cost.

Abstract interpretation can be applied to the systematic construction of methods and effective algorithms to approximate undecidable or very complex problems in computer science such that the semantics, the proof, the static analysis, the verification, the safety and the security of software or hardware computer systems. In particular, static analysis by abstract interpretation, which automatically infers dynamic properties of computer systems, has been very successful these last years to automatically verify complex properties of real-time, safety-critical embedded systems.

Verifying the software specification of numerical algorithms including iterative loops is considered a difficult problem. Formal specifications usually do not exist but implicit specifications can be used, such as the absence of run-time errors (overflows, etc.) or the stability of numerical computations. Due to undecidability issues, complete tools are generally impossible to design and tools may fail to prove (part of) the specification. Soundness dictates that the tools raise alarms to signal *all* potential violations of the specification. A spurious alarm reported by the tool when the specification is not actually violated is called a false alarm.

However, static analysis tools producing very few or no false alarms have been designed and used in industrial contexts for specific families of properties and programs. In all cases, abstract interpretation provides a systematic construction method based on the effective approximation of the concrete semantics, which can be (partly) automated and/or formally verified.

When dealing with undecidable questions on program execution, the verification problem must reconcile:

- correctness (which excludes non exhaustive methods such as simulation, test, bounded model checking, or syntactic pattern-matching),
- automation (which excludes model checking with manual production of a program model and deductive methods where provers must be manually assisted),
- precision (which excludes general analysers which would produce too many false alarms, i.e., spurious warnings about potential errors),
- scalability (for software of a few hundred thousand lines),
- efficiency (with minimal space and time requirements allowing for rapid verification during the software production process which excludes a costly iterative refinement process).

3. RESULTS OF THE PROJECT

3.1. Scope

The programs studied in this project have been automatically generated using proprietary tools familiar to control engineers (such as MATLAB/SIMULINK or SCADE) from high-level specifications (such as systems of differential equations or synchronous operator building blocks, which is equivalent to the use of synchronous languages like Lustre).

Such synchronous data flow specifications are quite common in real-time safety-critical control systems developed for on-board flight software. Periodic synchronous programming perfectly matches the need for the real-time integration of differential equations by forward fixed step numerical methods. The verification tools shall cope with this family of programs and the current status is the following:

- The Polyspace Verifier tool (The MathWorks) is currently used at Astrium SAS BU Space Transportation and other space industry but has shown some strong limitations. In fact, in order to limit the number of false alarms raised, the analysis of floating-point number overflows had to be disabled, which makes the approach much less interesting.
- The ASTRÉE tool [1,4] (studied in this project) is a static program analyzer aiming at proving the absence of run-time errors (RTE) in programs written in the C programming language. ASTRÉE uses generalist abstractions (intervals, octagons, etc.) and specific abstractions which have been designed for the application domain (to handle filters, integrators, etc.). The proof that the software satisfies the implicit specification (absence of RTE) is mathematically valid since it is made for a superset of all possible program behaviours and all possible execution

environments. However some executions in the over-approximations can lead to false alarms that do not correspond to an actual concrete execution. The whole difficulty of the undecidable problem of software verification is to chose sound over-approximations without false alarms (by soundness, no true error can be forgotten). ASTRÉE has been used successfully on the flight control software of the AIRBUS A340 and A380 [2] where it raised no false alarms, even for complex computations involving floating-point numbers. In the case of ASTRÉE, the programs to be analysed are real-time synchronous control-command applications.

- The FLUCTUAT tool [5] (also studied in this project) is an abstract interpretation tool for studying numerical programs coded in C, and in particular the propagation of uncertainties in floating-point computations. Its aim is to detect automatically a possible catastrophic loss of precision and its source, or else prove its absence. It relies on abstract domains for the estimation of values and errors, based on interval and affine arithmetic (with zonotopic concretization).
    - A language of assertions helps specifying the range of inputs and initial uncertainties. The tool delivers, for each scalar variable of the analyzed program, ranges for the value that variables would take if computed with an idealized semantics in real numbers, ranges for the machine values (floating-point or integer), and ranges for the error between the idealized and the machine semantics, decomposed by contribution from each line of code. The tool produces a graphical representation of the source of each numerical precision loss. It allows the user to know quickly the lines in the C source code causing the biggest losses of numerical precision.
    - For loops, the tool also allows to produce graphics representing the evolution of bounds for the values and errors of variables during the computation. This is an important feature for real-time systems, as it allows understanding the evolution of the numerical precision during the duration of the software execution.
    - Finally, it can also deliver information about the sensitivity of a code to initial errors.

A problem of numerical instability of the same kind as those that can be detected by FLUCTUAT had been discovered in the navigation algorithm of the MSU Software. But, due to a difference of precision between the host machine used for the algorithm validation and the target machine, this bug was detected very late, in the last stage of target validation.

3.2. Case study

A representative piece of space software has been provided by ASTRIUM ST in order to be used for the assessment of the ASTRÉE and FLUCTUAT tools. This case study is based on the on-board software of the Monitoring and Safing Unit (MSU) of the ATV space vehicle. The following criteria have been used to select the case study:

- The case study is representative of the complexity of software developed by ASTRIUM ST. The MSU SW comprises a simple GNC (Guidance, Navigation, Control), but fully representative of the numerical algorithms developed at ASTRIUM ST. The mission management part of the MSU SW is less representative of ASTRIUM ST Software development, but the study focuses on numerical algorithms.
- The case study is small enough to be manageable during a R&T study.
- The case study is available in C (even though the operational version has been developed in ADA, several C versions of the MSU SW exist).

The MSU SW contains mainly:

- Navigation and Control algorithms (i.e., GNC without Guidance),
- a simplified mission management (composed of one state automaton and of a plan sequence).

The Technical Specification of the MSU SW is based on a SCADE model. This model covers the high level software architecture, the mission management, and the architecture of the control and of the navigation. The flight version of the MSU SW has then been designed and developed in ADA. For the needs of several R&T projects, two versions of the MSU Software have been developed, respectively in SCADE V5 and in SIMULINK, which can generate C code. The code of this case study has then not been developed with the usual level of quality of operational projects of ASTRIUM-ST.

The analysis of the SIMULINK models has failed due to the optimization of the C code generated by RTW-EC. Thus, this version of the case study was abandoned and the work has focused on the SCADE models. These SCADE models have been updated to SCADE V6 (last version of the editor and code generator available since October 2007). Several C code generation options of SCADE 6 have been tested (not expanded or expanded with several optimization levels) in order to complete the assessment of ASTRÉE and FLUCTUAT.

3.3. Usability of abstract interpretation-based static analysis tools in space domain

3.3.1. Exhaustive detection of run-time errors with ASTRÉE

Software verification consists in proving that all executions of a program satisfy a specification. In the case of the ASTRÉE [1,4], the specification is implicit: no execution should lead to a run-time error or undesirable behaviour (out of range index, division by zero, dangling pointer, numeric overflow, etc.). ASTRÉE can also check some user-defined assertions (such as variables staying within user-specified ranges).

This study has shown that ASTRÉE is well adapted to the C code generated from SCADE V6 and to manual C code, but it is less efficient on C code generated from SIMULINK. It cannot analyse C++ nor ADA code.

On the proposed case study, the tool has allowed to detect and correct several bugs: incorrect access to an array, incorrect numerical protection, and incorrect use of memory copies (due to bugs in the experimental SCADE KCG tool when generating non optimized code).

To obtain the first analysis results, the following activities were performed:

- Definition of a specific library which can be parameterized for either run-time error analysis or embedded code generation. This library defines trigonometric, vector, matrix, and quaternion computations, as well as the square root and memory copy operations. This library has been developed in the scope of this project and can be reused in any analysis of similar space software. It may however have to be completed to more specific needs.
- Experimentation with the various SCADE KCG code generator options to discover which ones provide optimal analysis speed and precision (e.g., the analysis precision depends on how Boolean operations are compiled; it is faster when optimized code is generated, etc.). These settings are generic and can be reused on any analysis of similar space software.
- Definition and formalization of properties on the execution environment (e.g., acceptable values of input or maximum run time) to ensure the exhaustive data coverage of the analysis. This activity has to be performed for each project.

Then, in order to ensure a minimal number of alarms, the following actions were performed:

- Addition of a minimal number of numerical protections. The soundness of the tool ensures that no protection has been forgotten. In many cases, the tool can prove that no protection is necessary, thus greatly reducing the number of useless protections to insert (and these should be avoided as they have a negative impact on the efficiency of the code).
- Addition of known facts, that is, user-defined predicates that the tool assumes correct. The correctness of these known facts comes from a manual analysis or an analysis by the FLUCTUAT tool. The use of known facts could be avoided by extending ASTRÉE with domain-specific abstractions (such work was performed for avionics software, hence ASTRÉE's ability to reach zero false alarms on AIRBUS code without the need for any known fact; moreover, after the end of the SSVAI project, ASTRÉE has been extended with abstractions specific to quaternion computations, which reduces the need for known facts on the case study considered here).
- After these additions, ASTRÉE outputs 0 false alarms. Moreover, the analysis is extremely efficient: the 8 KLOC (lines before pre-processing of C code generated by SCADE KCG V6 in non-expanded and O3 optimized mode) control part of the case study is analyzed in 2mn30s on a 64-bit laptop PC while the 6 KLOC navigation part is analyzed in 1mn40s.

Even if additional work would be useful in order to improve the precision of ASTRÉE analysis on some specific features such as the handling of Kalman filters (e.g., to reduce the need for known facts), the ASTRÉE tool can clearly be used on this and any similar critical real-time embedded space software.

3.3.2. Analysis of numerical stability of algorithms with FLUCTUAT

This study has shown that FLUCTUAT [5] is compatible with C code generated from SCADE V5 (V6 has not been tested) and with manual C code but it is less efficient on C code generated from SIMULINK and is not compatible with C++ and ADA.

The tool has shown the following on the MSU code:

- The full code (38246 LoC, expanded inlined SCADE V5 generated C code) has been analyzed, under some restrictive hypotheses, and proved to behave well numerically.
- The ranges of the output of some critical functions of the MSU which were studied in internal specification documents of Astrium have been confirmed automatically by FLUCTUAT. FLUCTUAT also gave bounds on the imprecision errors for those functions that could not be computed by hand, hence were not detailed in these internal documents.
- The stability of a 8-th order filter (used to filter accelerations in the main control mechanism of the MSU software) has been proved automatically. The ranges of the output ($[-14.07, 14.07]$), both on real numbers and floating point numbers, found by

the tool, with full loop unfolding, correspond to the expected theoretical ranges, as specified in internal Astrium documents (inputs within [-10,10] and a gain equal to 1.4). Imprecision errors were shown to be negligible (the global error lies in [−5.45 10^{-5}, 5.45 10^{-5}] for simple precision floating point numbers).

The 8th order filter is made with 4 connected cells of order 2, Fluctuat shows that the biggest error comes from the third cell of order 2 (Fig. 1).

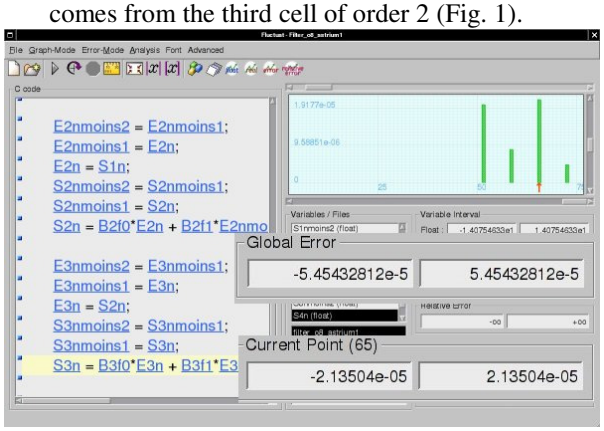

Figure 1. Biggest contribution to the global error comes from the transfer function of the third cell

Test cases were automatically produced by the tool to derive "worst-case" scenarios for values and errors. The two sequences are different (the maximal error is not related to the maximal value) and hard to derive manually (Fig. 2).

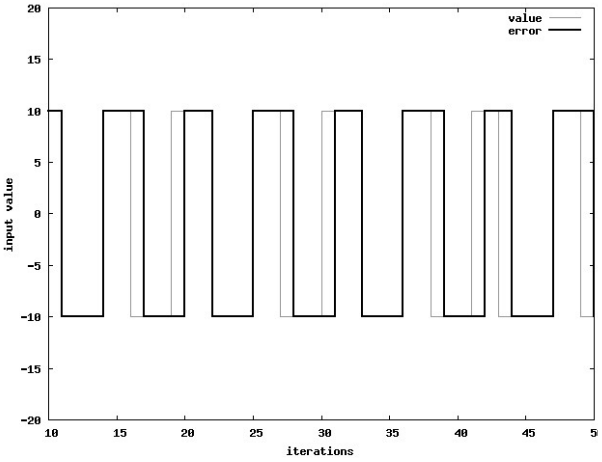

Figure 2. Sequences that derive the maximal value (gray line) and the maximal global error (black line)

For simple precision floating point numbers, the output of the filter was proved to be within [−14.0754471, 14.0754471], and the global error within [−5.454328 10^{-5}, 5.454328 10^{-5}]. The maximal value reached was 14.0754108; its related global error was bounded by 1.15896 10^{-6}. However, the maximal error reached was 7.22078 10^{-6} (Fig. 3) and its related value was −1.17364860.

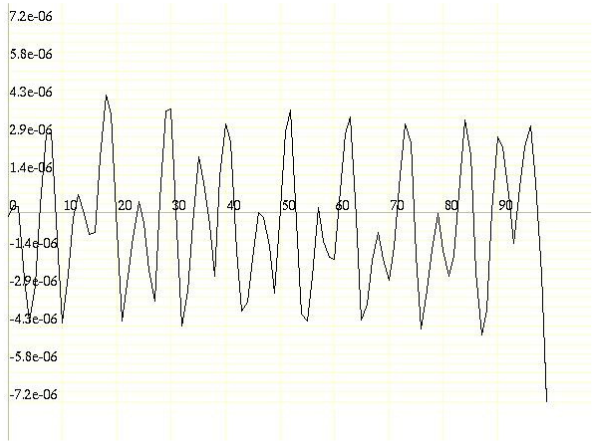

Figure 3. Evolution of the error function of the sequence leading to the maximal error

The current version of Fluctuat embeds improvements of the abstract domain [6] made after the ATV case study. The tool is able to derive a tight invariant of the former filter ([-15.97, 15.97]).

- The prediction part of the Kalman filter, heart of the control mechanism, relies on two 4-th order Runge-Kutta (RK4) integrations. .

  Here, two different kinds of errors are of interest: the imprecision error due to the use of finite precision numbers and “functional” errors related to the integration scheme used to solve ordinary differential equations (ODE).

  For the first kind, that is, the imprecision error, Fluctuat shows that the main contribution to the global error on the acceleration command ac[] comes from the representation in floating point numbers of integration steps (0.075 and 0.925 in real numbers). For example, in Fig. 4, one can see that the representation of 0.925 introduces an error of 1.06 10^{-10} on variable ac[1] which value is 2.60 10^{-6} . This corresponds to a non negligible relative error of 4 10^{-5}.

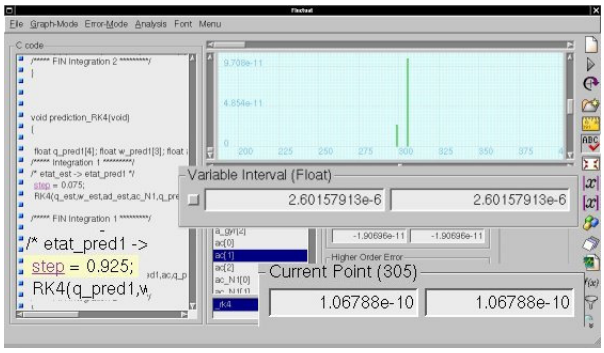

Figure 4. Contribution of the integration step to the global error

For the functional error, we have used an external guaranteed integrator, GRKLib [7], to estimate the difference between the values given by a guaranteed integration of ODEs modelling the physical environment and the values given by Fluctuat on the implementation of the prediction part of the MSU Software. We have found that the relative discrepancy is around 18%, which means that the integration scheme used is rather imprecise. However, comparative analyses showed that the choice of an RK4 integration algorithm (with these large integration steps) was better than simpler Euler-like integration algorithms (with smaller integration steps – for instance 0.1 and 0.01).

After the end of the SSVAI project, the whole Kalman filter (and not only its prediction part) has been analysed with HybridFluctuat [8] (Fluctuat tool enhanced with a guaranteed integrator), to analyse the difference between the real position of the ATV, given by sensors (over approximated by the guaranteed integrator), and the estimation of this position given by the software (implementation of the Kalman filter): the discrepancy was found to be around 5% [8].

All these features made the tool very practical and very efficient to better understand the numerical behaviour of the system under analysis.

This study has also shown that space software is much more difficult to analyze than aeronautic software due to the important number of non linear computations, especially with quaternions (most software from the aeronautics industry that have been analyzed by FLUCTUAT were using linear computations, except for some specific and isolated functions).

4. CONCLUSION

The ASTRÉE [1,4] and FLUCTUAT [5] tools are applicable to any embedded space software developed manually in C. They can also analyze C code generated from SCADE models but they are less efficient on C code generated from SIMULINK models and cannot analyze other programming languages such as C++, ADA, or Java, which is an important restriction.

This study has shown that embedded space software are difficult to analyze due to non linearity (mainly in quaternion computations) and complex control command algorithms involved (e.g., Kalman filter). ASTRÉE can be extended to handle this by designing new specific abstract domains. Some progress was made after the end of the project through the addition of a new domain specialized in the analysis of quaternion computations. The architecture (concerning the activation conditions) of the software has also an important impact on the efficiency of the analyses. But it should be noticed that the software architecture which suits static analysis by abstract interpretation best is also the more readable one and maintainable one. This technique can thus be a metrics of good architectures.

In spite of these difficulties, abstract interpretation techniques can greatly improve the quality of space embedded software:

- ASTRÉE has allowed correcting bugs in the case study,
- the number of remaining false alarms is equal to zero (compared to several hundred of remaining false alarms for an analysis with Polyspace Verifier)
- FLUCTUAT has confirmed some manual analyses performed on the MSU software,
- FLUCTUAT has delivered results on global errors which were not manually achievable,
- the tools are complementary: they prove different properties and may be used together (e.g., the ranges found by a global analysis by ASTRÉE can be used as input by FLUCTUAT to study the relative precision of a given numeric computation; on the other hand, properties proved with the help of FLUCTUAT can be inserted as known facts in code analyzed by ASTRÉE),
- a process of use has been defined for both tools.

Thanks to a case study representative of the software developed at ASTRIUM ST, the results of this study will be applicable to any type of embedded critical real-time space software (launchers, satellites, spacecrafts, and space probes) developed in C. They will improve the quality of software (fewer residual bugs) and will at the same time dramatically decrease the costs of robustness testing.

The study has also hinted towards some directions of improvement for the tools.

As a conclusion, the Technology Readiness Level (TRL) for ASTRÉE and FLUCTUAT on space software is evaluated between 4 (component and/or breadboard validation in laboratory environment) and 5 (component and/or breadboard validation in relevant environment).

Acknowledgement.

This study was partially funded by ESA contract ITI19783.

5. BIBLIOGRAPHY

1. The ASTRÉE static analyzer, available from AbsInt Angewandte Informatik, http://www.absint.com/
2. D. Delmas, J. Souyris. ASTRÉE: from research to industry. In Proc. of the 14th Int. Static Analysis Symposium (SAS'07), Kongens Lyngby, Denmark, pages 437-451 of LNCS volume 4634. Springer, 2007.

3. P. Cousot, R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proc. of the 4th ACM Symposium on Principles of Programming Languages (POPL'77), Los Angeles, pages 238-252. ACM Press, 1977.
4. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE analyzer. In Proc. of the 14th European Symposium on Programming (ESOP'05), Edinburgh, Scotland, pages 21-30 of LNCS volume 3444. Springer, 2005.
5. E. Goubault, M. Martel, and S. Putot. Asserting the precision of floating-point computations: a simple abstract interpreter. In Proc. of the 11th European Symposium on Programming (ESOP'02), Grenoble, France, pages 287-306 of LNCS volume 2305. Springer, 2002.
6. K. Ghorbal, E. Goubault, and S. Putot. The Zonotope Abstract Domain Taylor1+. In Proc. of the 21st Computer Aided Verification (CAV'09), Grenoble, France, pages 627-633 of LNCS volume 5643. Springer 2009.
7. O. Bouissou, and M. Martel. Abstract Interpretation of The Physical Inputs of Embedded Programs. In Proc. of the 9th Verification, Model Checking, and Abstract Interpretation (VMCAI'08), LNCS volume 4905. Springer, 2008.
8. O. Bouissou, E. Goubault, S. Putot, K. Tekkal, and F. Vedrine. HybridFluctuat: a static analyzer of numerical programs within a continious environment. In Proc. of the 21st Computer Aided Verification (CAV'09), Grenoble, France, pages 620-626 of LNCS volume 5643. Springer 2009.