HybridFluctuat: A Static Analyzer of Numerical Programs within a Continuous Environment

Olivier Bouissou[1], Eric Goubault[1], Sylvie Putot[1], Karim Tekkal[2], and Franck Vedrine[1]

[1] CEA, LIST, Modelisation and Analysis of Systems in Interaction, Boîte 65, Gif-sur-Yvette, F-91191 France
[2] FCS Digiteo, Route de l'Orme des Merisiers, Saint-Aubin, F-91190 France
Firstname.Lastname@cea.fr

Abstract. A new static analyzer is described, based on the analyzer Fluctuat. Its goal is to synthetize invariants for hybrid systems, encompassing a continuous environment described by a system of possibly switched ODEs, and an ANSI C program, in interaction with it. The evolution of the continuous environment is over-approximated using a guaranteed integrator that we developped, and special assertions are added to the program that simulate the action of sensors and actuators, making the continuous environment and the program communicate. We demonstrate our approach on an industrial case study[1], a part of the flight control software of ASTRIUM's Automated Transfer Vehicle (ATV).

1 Introduction

An emerging trend in the software verification community is to extend the analysis of programs to take into account their interaction with the external world. In the case of embedded programs, one of the most important interaction to consider is the one between the program and a physical environment on which it acts [4,5,9]. Generally, static analyzers abstract these interactions in a simple way: inputs and outputs are abstracted by intervals. If this is obviously sound, it leads to an important overestimation as it assumes that a continuously evolving variable can instantaneously jump from its minimum to its maximum value.

In this paper we present a new static analyzer, named HybridFluctuat, that makes it possible to analyze the interactions of an embedded program with its environment. Thanks to a language of assertions, it takes into account sensors (from which the program reads the value of a physical variable) and actuators (with which the program acts on the system behavior). The continuous environment is modeled as a set of switched ODEs and its evolution is abstracted using the guaranteed integration solver GRKLib, while the analysis of the program itself relies on the analyzer Fluctuat.

Running example. To illustrate this, let us look at a typical example of an embedded, control command program. Listing 1 shows a simplified version of

[1] This work was partially funded by the ESA project ITI 19783 "Space Software Validation using Abstract Interpretation". Thaks are also due to ASTRIUM SAS.

A. Bouajjani and O. Maler (Eds.): CAV 2009, LNCS 5643, pp. 620–626, 2009.

```
1  Initialize();
2  for (i=1;;i++) {
3    // Get new value for qnav and wnav
4    // Make two steps of RK4
5    a = ac[i-1];    (q1,w1) = RK4(a,qnav,wnav,h1);
6    a = ac[i];      (q,w) = RK4(a,q1,w1,h2);
7    (qest,west) = Fk(q,w,qnav,wnav);  // Kalman filter
8    ac[i+1] = Fa(qest,west,q,w);    // Next command
9    // Send ac to the actuator
10 }
```

Listing 1. Simplified algorithmic view of the ATV program

the MSU main control loop, part of the Automated Transfer Vehicle (ATV) control software. Its behavior is typical of embedded programs: at each cycle, its configuration (the position, speed, etc.) is read from sensors, then the program computes the command sent to actuators to achieve the desired thrust with the engines. In this application, the command is computed using a Kalman filter, where the prediction step is done using two Runge-Kutta integrations of order 4. The external environment (relating the position of the ATV to accelerations due to thrusters) is modeled by an ODE of dimension 7 that links the position of the ATV (recorded as `qnav` in the program) with its angular velocity (`wnav`). It also has three parameters that are linked with the vector `ac` of the program.

At line 3 of the program, the value of `qnav` and `wnav` is modified by the sensors: it takes its value from the solution of the system of ODEs. Then, the program computes the value of the command `ac`. This value is sent to the actuators that modify the thrust of the engines on line 9, and thus changes the parameters of the ODE and the evolution of the continuous system.

Contributions. The main contribution of HybridFluctuat is that it extends the static analysis of embedded programs by considering the physical environment in which they are executed. The analyzer considers programs written in C-ANSI and ordinary differential equations presented as a C++ function (see Listing 1.2 later). The tool then automatically derives invariants on the whole system.

Related work. Static analyzers for hybrid systems [2,7] mainly focus on high level models like hybrid automata and are generally used to prove the reachability of some state. Our approach differs from these as we consider the program itself and not a derived model of it, which allows us to analyze the impact of the implementation choices (e.g. the use of floating point numbers) on the behavior of the whole system (this feature is not yet implemented, but is a straightforward consequence of our approach).

2 HybridFluctuat: Description and User Point of View

In this section, we briefly describe the principles and use of HybridFluctuat. HybridFluctuat builds on two previously existing tools, Fluctuat and GRKLib,

```
void on(double* res,double* y,double* param)
   { res[0] = param[0]-y[0]/3;}
void off(double* res,double* y,double* param)
   { res[0] = -y[0]/3;}
```

Listing 1.2. Environment of the heater problem written as C++ functions

more details on which can be found in [3,8,11]. In practice, Fluctuat will be used to analyze a C program with special assertions specifying its interaction with the environment, and will call GRKlib whenever it encounters such assertions.

Fluctuat [8,11] is a static analyzer by abstract interpretation [6] that interprets a program written in ANSI-C with idealized semantics (real and integer numbers) and finite precision semantics (floating-point and machine integers). It gives bounds for variables with these two semantics, and bounds the error due to the use of finite precision numbers instead of real numbers. It decomposes this error on its provenance in the analyzed program, thus indicating which part of the program is responsible for the main imprecision.

GRKLib [3] is a C++ library that, given a system of ordinary differential equations (ODE) and an interval initial value, computes an interval overapproximation of its solution, either at a specific time stamp or over a whole time interval. To do so, the algorithm turns a numerical, non guaranteed Runge-Kutta method into a guaranteed integration method.

The input of HybridFluctuat is a system of ODEs given by the user as C++ functions, and a C program, in which the user added special assertions specifying the interaction between the program and the environment, i.e. between variables of the programs and the solutions or parameters of the ODEs.

There may be several systems of ODEs that model different parts of the environment, and interact with different parts of the program. We associate a different name `system_i` to each system. The program may modify a system `system_i` in two ways. Either the program makes a discrete change of mode, thus changing completely the evolution of the environment, or the program modifies a parameter in the ODE. To model these two kinds of interactions, we associate to each `system_i` several ODEs (named `mode_j` in the following), each representing one mode. Each mode `mode_j` also contains formal parameters that will be modified by the program. This models the action of actuators. For example, the ODEs of Listing 1.2 model the evolution of the temperature of a room with a heater that may be on (ODE `on`) or off (ODE `off`). The parameter `param[0]` represents the power of the heater (that may be changed by the controller).

In return, the environment influences the program: the program questions the values of physical variables (solutions of the ODEs), and uses them in the program. We suppose that the continuous time is computed by the program: when the program interacts with its environment, it must specify the time at which this interaction takes place. We now specify the assertions that represent these interactions in the C program.

Initializing the environment. First, the initial mode of each system `system_i` must be specified using the assertion `HYBRID_INIT_MODE(system_i, mode_j)`. Then, the initial value `param` of the j^{th} parameter of the system `system_i` must be given using the assertion `HYBRID_PARAM_DINIT(system_i, j, param)`.

Finally, for each system `system_i`, the value of each component of the initial state must be set by `HYBRID_DINIT(system_i, j, value)`, where `j` means that we set the value of the j^{th} component. When the analyzer encounters these assertions, it sends the information to the guaranteed integrator via XML files.

Getting values from the environment. As we said, the program can read values from the environment by calling `HYBRID_DVALUE(system_i, component_j, t)`. It reads the j^{th} component of the state of `system_i` at global time `t`. When the analyzer encounters this assertion, it performs the guaranteed integration of the system `system_i` up to time `t`: the result of the assertion is the solution of this integration, transmitted again through an XML file.

Modifying the environment. Finally, the program can change the mode of a given system `system_i` at time `t` to the new mode `mod_j`. It uses the assertion: `HYBRID_MODE(system_i, mode_j, t)`. The assertion for changing the value of the j^{th} parameters of the system `system_i` to the new value `value` at time `t` is: `HYBRID_PARAM(system_i, j, value, t)`. In practice, this changes the system of ODEs that will be used in order to compute the next values that will be read. Before that, the existing system is integrated until time t.

3 Experiments

The two tanks system [10]. This system is composed of two water tanks linked by a tube, and a controller that must keep the water levels in both tanks between safe bounds. We encoded the system as a C program [2] plus a set of ODEs and analyzed it with HybridFluctuat. We set the initial values of the levels to the range $[3, 8] \times [4, 7]$. As a result, HybridFluctuat proved that the water levels in both tanks remain in the range $[2.4, 8.5] \times [3.5, 7.8]$. We are thus able to automatically prove the correctness of the control command program on the whole state space with non-linear dynamics, which, to the best of our knowledge, cannot be done by existing verification tools. As the initial ranges of values are large, we used in the analysis regular subdivisions of width 0.1 on these initial ranges (this was done automatically by our tool). The analysis took one hour.

*The heating system [1].*This system is composed of n adjacent rooms, each with a heater, so that only m of them can be switched on at the same time. A controller must maintain a certain temperature in all the rooms. We used our tool to compute a range on the rooms temperature: this time, we set the initial condition to be a point (a temperature of 20 in all rooms) and introduced an uncertainty of 10% on all the parameters of the system (power of the heater, outside temperature...).

[2] The programs and ODEs mentioned in this article can be found at http://www.lix.polytechnique.fr/~bouissou/progs/hybridfluctuat/

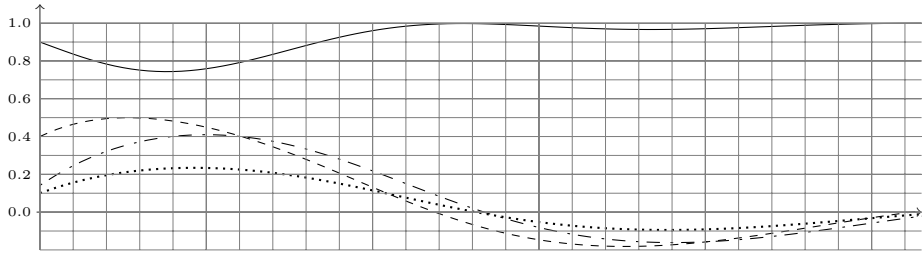

Fig. 1. Trajectory of the ATV over time in the quaternion coordinates. The black line is the first dimension of the quaternion, the dashed ones are the 3 others.

HybridFluctuat proved, in 28 seconds, that the temperature in all rooms remains above 11.5, which is similar to results obtained using PHAVer [7].

Industrial case study. We used HybridFluctuat to analyze the behavior of a simplified version of the MSU part of the safety ATV control program described in Section 1. The C program is about 400 lines of code including many array manipulations and non linear floating point computations. The ODE is of dimension 7 with 3 parameters. The open-loop system is particularly unstable, the continuous environment only converges under the action of the control program. This made this whole system hard to analyze. We first computed an overapproximation of the continuous trajectories on the 50 first seconds, with the initial condition being a point. We obtained Figure 1 that shows that the measured values converge towards $(1, 0, 0, 0)$, which is the safe state that the ATV is supposed to reach in this escape mode. This analysis, although performed on a finite time range, gives a good indication on the correctness of the control command program. Ideally, a fixpoint computation proving the convergence for unbounded time would be needed to prove the correctness. This is not implemented in HybridFluctuat yet.

We also measured the estimation error, i.e. the difference between the predicted position of the ATV and the position given by the sensors at the next cycle. The predictions were proved to be accurate: their error is around 5%.

4 Conclusion and Future Work

In this article, we presented the new static analyzer HybridFluctuat that extends the static analysis of numerical properties of embedded programs performed by Fluctuat, by accurately modeling their interaction with their physical environment. The evolution of the environment is overapproximated by GRKLib, a guaranteed integrator of ODEs. Although it is a preliminary work, we obtained very promising results and the case study we performed shows that it can be used for industrial applications. For the time being, we used HybridFluctuat to compute invariants on values of the variables of the program, and overapproximations of the continuous trajectories. We are thus able to prove the numerical stability of the implementation of a control-command algorithm. We next intend to extend this analysis in two natural directions described hereafter.

One specificity of Fluctuat is to model the propagation of initial uncertainties and rounding errors through numerical computations, pointing out the sources of the main errors on the outputs. In HybridFluctuat, we consider for now that the values sent to the actuators are error free, which is obviously not the case. We thus want to investigate the influence of the difference between the commands actually sent to the environment, and the ones that would be sent if the program used real numbers: this difference induces two distinct evolutions of the environment. We want to consider the difference between both evolutions to be the propagation of the computation errors. Secondly, we want to remove the current limitation of HybridFluctuat that imposes to analyze the system on a finite time range. To do so, we need to compute fixpoints of the continuous variables, i.e. fixpoints of the solutions of the ODEs: extrapolation algorithms may be of great help in this perspective. Finally, a current use of Fluctuat aims at bounding, when possible, not only the imprecision due to the implementation, but also the imprecision due to the method. In some applications (like the ATV case study), approximate ODE solvers are part of the program implemented. HybridFluctuat, with its guaranteed ODE solver, can give an estimation of the idealized result, and the difference with the result of the implementation, which can be seen and propagated as a method error.

References

1. Fehnker, A., Ivancic, F.: Benchmarks for hybrid systems verification. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 326–341. Springer, Heidelberg (2004)
2. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138(1), 3–34 (1995)
3. Bouissou, O., Martel, M.: GRKLib: a guaranteed runge-kutta library. In: Follow-up of International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics. IEEE Press, Los Alamitos (2007)
4. Bouissou, O., Martel, M.: Abstract interpretation of the physical inputs of embedded programs. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) VMCAI 2008. LNCS, vol. 4905, pp. 1–3. Springer, Heidelberg (2008)
5. Cousot, P.: Integrating physical systems in the static analysis of embedded control software. In: Yi, K. (ed.) APLAS 2005. LNCS, vol. 3780, pp. 135–138. Springer, Heidelberg (2005)
6. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixed points. Principles of Programming Languages 4, 238–252 (1977)
7. Frehse, G.: Phaver: Algorithmic verification of hybrid systems past hytech. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 258–273. Springer, Heidelberg (2005)
8. Goubault, E., Martel, M., Putot, S.: Asserting the precision of floating-point computations: A simple abstract interpreter. In: Le Métayer, D. (ed.) ESOP 2002. LNCS, vol. 2305, pp. 209–212. Springer, Heidelberg (2002)
9. Goubault, E., Martel, M., Putot, S.: Some future challenges in the validation of control systems. In: ERTS, SEE (2006)

10. Kowalewski, S., Stursberg, O., Fritz, M., Graf, H., Preuß, I.H.J., et al.: A case study in tool-aided analysis of discretely controlled continuous systems: the two tanks problem. In: Antsaklis, P.J., Kohn, W., Lemmon, M.D., Nerode, A., Sastry, S.S. (eds.) HS 1997. LNCS, vol. 1567, p. 163. Springer, Heidelberg (1999)
11. Putot, S., Goubault, E., Martel, M.: Static analysis-based validation of floating-point computations. In: Alt, R., Frommer, A., Kearfott, R.B., Luther, W. (eds.) Dagstuhl Seminar 2003. LNCS, vol. 2991, pp. 306–313. Springer, Heidelberg (2004)