PROOF OF PROPERTIES IN AVIONICS.

Jean Souyris (jean.souyris@airbus.com)

Denis Favre-Félix (denis.favre-felix@airbus.com)
Airbus France

Abstract: This paper presents the industrial use of a program proof method based on CAVEAT (C program prover developed by the commissariat à l'énergie atomique) in the verification process of a safety critical avionics program.

Key words: Proof of properties – Avionics program – Formal method - Unit Verification – Unit Testing – DO 178B

1. INTRODUCTION

Avionics programs size and complexity ever increase. This affects the most critical ones, like flight control programs, as well. Avionics programs must conform DO178B standard, leading to spend about 60 % of the development time in verification, for the most critical of them (level A). Verification activities consist of readings, intellectual (human) analyses and test.

In the 'ever increasing' context mentioned above, the cost of all kinds of verification tends to get higher, in order to keep the dependability of the programs at the same (high) level.

The main concern is about tests. Indeed, most of the verification time is spent in testing and one can predict a dramatic augmentation of the test effort if nothing is done, specially for critical avionics programs. The main concern, here, is about the coverage of the tests: how to face the augmentation of complexity by this non exhaustive verification technique?

For at least partially compensate the above drawbacks, Airbus decided to introduce tool-aided static analysis techniques into its avionics verification workbench. The objective is the availability of static analyzers for an effective industrial usage, via the participation in R&D projects together with academic laboratories and tool makers.

Examples of static analyzers ready for an industrial usage or actually used are: AbsInt's Stackanalyzers ; AbsInt's aiT (Worst Case Execution Time) analyzers, ASTREE (proof of absence of Run Time Errors) from Ecole normale supérieure (Paris, rue d'Ulm) and CAVEAT from the Commissariat à l'énergie atomique, the latter being the topic of the rest of this chapter.

Indeed, this chapter reports the industrial usage of a program proof method based on CAVEAT, from the CEA. It is structured as follows: section 2. briefly describes CAVEAT; section 3. presents the development process before using CAVEAT and shows where CAVEAT has been introduced; section 4. describes the method of using CAVEAT; section 5. is the actual experience report; section 6. concludes.

2. CAVEAT

2.1 Features

Caveat is a static analyzer which aims at automatically deriving properties (property synthesis) from a C program and proving properties on this program. These properties are expressed in a first order predicate language.

Caveat is developed by the Commissariat à l'énergie atomique (CEA) partly with the financial support of Airbus France via R&D funding (French or EU).

Airbus France's effort on Caveat has consisted in making Caveat able to analyze the targeted avionics applications. As these applications, like most of embedded real time programs, have peculiar characteristics like hardware handling, some extensions of the subset of the C language accepted by Caveat have been developed. On another hand, Caveat's automatic proof capabilities have been extended for at least giving good "automaticity rate" on the targeted applications.

Caveat takes C source files as inputs and first automatically computes properties relative to control and data flows. In particular, caveat computes the **data dependencies** at C function level. The result of this computation is the actual interface of each C routine, which is something not obvious from the C prototype of a function. Amongst the information automatically

extracted from the sources during the data dependencies analysis one finds, for each C function encountered in the source files the list all the operands of the function: implicit operands, i.e. global variables, and explicit operands, i.e., those declared in the function prototypes; for each operand, the way it is used: In, Out or In/Out, respectively for read only, write only and read/write operands; the dependences between the operands: it is, for each Out or In/Out operand, the list of In and In/Out operand it depends on.

Proof of user-specified properties. Once the initial automatic analysis is completed (see section 2.1.1), Caveat is ready for input of properties by the user. Most of these properties are to be proved by the tool, in particular the *Post* property. *Post* stands for 'Post-condition', which means that the user wants the property of that type to hold at the end of a particular C function. Then Caveat is asked to prove the Post-condition. A success of this attempt to prove the Post-condition means that the latter holds, a failure have two possible reasons: first it might be that Caveat failed to prove a correct property; on the other hand, Caveat failed two prove a property which does not hold. In both cases, the user must analyze the first order predicate Caveat returns. Now it is up to the user to complete the proof process by using the Interactive Predicate Transformer feature of Caveat This feature allows the user to process the result predicate either to complete the proof or to find the reason why the proof is impossible to make, i.e., the property does not hold.

C language Restrictions: Caveat complains when, in the source code it analyses, it finds things like recursion, backward goto statements, pointers on functions, and other situations which are usually not allowed in critical avionics applications. It is the case of the industrial usage of Caveat reported in this paper.

Possible applications of Caveat. Basically, Caveat may be used to the following usual verification of avionics programs: Control and data flow verification, Unit Verification, Integration Verification and safety analyses.

2.2 First industrial application: Unit Proving

First target. For a first application of Caveat Airbus chose the safety test part of the program of the Flight Control and Guidance Unit of the A380. This onboard computer embeds the primary function of the Fly-by-wire system.

The interest of getting started with the industrial use of Caveat on this safety test program is twofold: firstly, this about 40,000 lines of code program is representative of a certain number of safety critical avionics

programs; secondly, the (existing) verification process of its predecessors (previous safety test programs) was best suited for the substitution of Unit Testing by Unit Proving.

First way of using Caveat. Section 2.1 listed the different kinds of verification possibly addressable by Caveat.

As the main goal of applying formal verification techniques is to reduce the cost of the verification without reducing its quality, Airbus France chose to apply Caveat to the Data flow and Unit Verification activities.

Indeed, in the traditional approach, Data flow analysis is mostly an intellectual activity, whereas Unit Testing, in spite of being partially automated, is costly in terms of time spent in identifying and programming the test cases. About the costs of Unit Testing, one must also consider the specialized hardware means (Unit test must be realistic).

In this context the choice of Unit Proving, among the different potential usages of Caveat is motivated by:

- the high degree of automation of the proofs if the analysis is performed per function;
- the fact that this replacement, i.e., Unit Testing by Unit Proving, does not affect the DO178B conforming verification process (DO178B (1992) does not really know about program proofs).

3. FROM UNIT TESTING TO UNIT PROVING

3.1 DO178B conforming development process (briefly)

The development of an avionics program must conform to DO178B. All aspects of the development are constrained by this standard.

Amongst all the activities defined by DO178B, the ones directly involved in the production of the program are: specification, design and coding. Each of them must be verified by activities also impacted by DO 178B. The kind of verification activities promoted/accepted explicitly by DO178B are: readings for checking that rules are actually applied, intellectual analyses and tests. Program proofs are only mentioned.

From the specification of the program, called High Level Requirements, the architecture (dynamic and static) and the Low Level Requirements are derived. The dynamic architecture deals with the temporal behavior of the program and static architecture consist of the decomposition of the entities defined previously into a set of modules/routines whose interaction implements the High Level Requirements. The per routine requirements (for

coding of the routine) are then produced, there are called: Low Level Requirements (LLR).

Then the code of each module/routine is written, it is the implementation of the Low Level Requirements.

A first important – sometimes the major – verification activity of safety critical avionics programs consist in checking the code of a routine against its Low Level Requirements.

3.2 Verification of the LLR by Unit Testing.

In the development process of the predecessors of the program chosen for the first application of Caveat, i.e., the safety test program of a Fly-by-Wire application) the legacy technique for verifying the Low Level Requirements is: Unit Testing. This technique consists in finding the best possible test cases (for a maximum coverage), writing test programs in the test specification language of a test automation tool, executing the tests on representative hardware and exploiting the results. The art of "finding the best possible test cases" is strongly constrained by DO178B.

3.3 Unit Proving LLR in a DO178B conforming process

The advantages of using Caveat for Unit Verification have been stated in section 2.3.2.

On the other hand, using a proof technique has some drawbacks with respect to the test-by-execution. The main concern is the fact that during a proof campaign, the actual binary code of a routine is not executed on representative - or, better, real - avionics hardware. Moreover, the proofs by Caveat are performed at source level, i.e., before compilation.

Consequently, complementary analyses and/or verifications are mandatory for demonstrating that the underlying verification objectives of the DO178B are fulfilled.

Nevertheless, the advantages of the Caveat-based proof method in terms of coverage as well as from the industrial point of view (the Low Level Requirements are directly used in the proof process whereas testing requires a costly identification of test cases and test program writing) are superior to the above mentioned drawbacks.

4. A DEDICATED METHOD

Unit Proving mainly impacts two DO178B conforming activities significantly: the production of the Low Level Requirements and the verification of these LLR.

At design time, once the static architecture has been defined, one gets a set of modules and routines whose interfaces are precisely specified. Next step is to write the LLR for each routine. These LLR mainly specify all possible behaviors of each routine. With Unit Proving, these LLR are formalized in the first order predicate language of Caveat. The properties to be produced must specify all possible input/output relations, the control flow within the routine and the interface with the called routines.

In order to write complete Low Level Requirements with respect to the High Level Requirements as well as properties which allow control flow verification, categories of properties have been defined: they are the main features of the guideline property writers must follow when they write the Low Level requirements of each routine. One of these categories is made of the so called "execution conditions" which defines relations on the inputs of a routine. These relations are the complete set of conditions which distinguish the different behaviors of the routine. A direct consequence is that the union of the subsets of all combinations of input values defined by these "execution conditions" must be the set of all possible combinations of input values.

Proof activity: Beyond the notion of exhaustivity of the verification compared to testing, a great advantage of "à la Caveat" Unit Proving is the high degree of automation of the proof activity, once the properties have been written. If algorithms have the – reduced - complexity generally admitted in safety critical avionics programs, most of the properties are proved automatically, possibly after correction of bugs in the code. Some of the properties which are not basically proved by Caveat automatically, are actually proved thanks to user-heuristics written in the script language of Caveat. The remaining properties are proved interactively under control of the Interactive Predicate Transformer of Caveat.

5. EXPERIENCE REPORT

5.1 Formalized Low Level Requirements production

The use of a formal method has great positive impact on the software design development. First of all, the property writing leads to more precise and non-ambiguous Low Level Requirements thanks to the use of a formal language.

Besides, the design guideline clearly identifies the different characteristics of a function, which should be covered by a set of properties. This categorization makes easier the way to reach the completeness of a design. In particular, some properties have to check the completeness of the execution conditions defined for a function. These properties increases in great proportion the confident we have in the completeness of the decision tree of a function.

The number of properties and the complexity of each property (in terms of writing efforts and number of operands) are a well-confident criteria in the "testability" of a function : as the properties represent the "proof plan" of the function, we can determine earlier in the software lifecycle if a function is provable or not.

Since this is the first time a formal language is used for designing functions, it was decided to keep on writing pseudo-code description of the function behaviors in order to make easier the coding phase.

Current statistics:
152 functions designed
2489 properties written
=> 16 properties per function

5.2 Proof process

One of the main objectives relative to the proof process was to automate the process as far as possible. Scripts have been developed in order to automate:

- the creation of a CAVEAT project (which include C-source file to be proved and C-source files simulating the called functions),
- the insertion of the properties,
- the addition of user-heuristics (in order to terminate automatically certain properties)
- the proof of the properties,

- the generation of a result file,
- the generation of a dataflow control file from the CAVEAT analysis of the C-source file.

User heuristics have been developed in order to automate actions under the Caveat Interactive Predicate Transformer and make easier the proof of properties. These heuristics are adapted to very specific situations, which often concern loop processing. For instance,:

- particularization of a high level quantifier into all the following low-level quantifiers
- decomposition of a quantifier in a (Min:Max) enumeration

From an industrial point of view, another advantage of the formal verification method consists in the fact that no specific hardware material is required. No test panel needs to be immobilized during the proof process.

5.3 Proofs

Current statistics :
2489 properties written
89% properties proved:

- 60 properties proved by means of heuristics or interactive manipulations (3%)

11% properties not proved :

- Syntaxes errors not yet corrected
- CAVEAT tools to be improved
- Interactive proof termination to be done
- Tools limitations

5.4 Conclusion

In comparison with a traditional software design development, the use of a formal proof method implies greater effort during the design phase. But this "waste of time" is largely compensated during the verification phase. A gain of 10 to 15 % is finally reached over the design/coding/unitary verification phases.

Furthermore, the exhaustively of the proof verification increases the confidence in the coverage of the unitary verification.

6. CONCLUSION

This paper presented how the potentialities mentioned in [Randimbivololona et al., 1999] became effective in practice.

In the future, the use of Caveat will be extended to other avionics programs as well as the way of using it will go beyond Unit Verification: towards Integration Verification.

7. REFERENCES

Randimbivololona, F., Souyris, J., Baudin, P., Pacalet, A., Raguideau, J., and Schoen, D., (1999). Applying formal proof techniques to avionics software: A pragmatic approach. In Wing, J.M, Woodcock, J., and Davies, J., editors, *Proceedings of the World Congress on Formal Methods in the Development of Computing Systems,* FM'99, volume II of *Toulouse, France, Lecture Notes in Computer Science 1709,* pages 1798-1815. Springer.