Introduction

Context

Although formal program analysis techniques (see works by Hoare [HOA 69] and Dijkstra [DIJ 75]) are quite old, the implementation of formal methods goes back to the 1980s. These techniques enable us to analyze the behavior of a software application described in programming language. Program correction (good behavior, program stop, etc.) is then demonstrated by program proof based on the calculation of the weakest precondition [DIJ 76].

It was not until the end of the 1990s that formal methods (Z [SPI 89], VDM [JON 90]) and the B method [ABR 96, ARA 97] were used in industrial applications and could be applied in an industrial context. One of the obstacles to their use was how they could be implemented in an industrial application (large application, time and cost constraints, etc.). They could only be implemented using tools that were mature enough and had sufficient performance.

It is worth noting that in the context of critical applications, at least two formal methods have a recognized and commonly used design environment that covers part of the realization of the code specification process while integrating one or several verification processes, that is to say the B method [ABR 96] and Lustre language [HAL 91, ARA 97] and its graphic version, called SCADE® [DOR 08]. The B method and SCADE® environment are associated with proven industrial tools. For example, AtelierB and Btoolkit, commercially produced by Clearsy and Bcore, respectively, are tools that completely cover the B method development cycle (specification, refinement, code generation and proof).

Introduction written by Jean-Louis BOULANGER.

Formal methods are based on different formal verification techniques, such as proof, *model checking* [BAI 08] and/or simulation.

The use of formal methods, though in full expansion, is still marginal compared to the number of code lines. Indeed, there are currently many more lines of Ada [ANS 83], C and C++ code that have been manually produced via a formal process only. For this reason other formal techniques have been implemented to verify the behavior of a software application written in a programming language such as C or Ada. The main technique, called *abstract program interpretation* [COU 00], enables us to evaluate the set of behaviors of a software application using static analysis. In the past few years, this type of technique has given rise to several tools, such as Polyspace®[1], Caveat[2], Absint[3], Frama-C[4] and/or Astrée[5].

The efficiency of these static program analysis techniques has greatly progressed with the increase in the power of office equipment. It is worth noting that these techniques generally require the integration of complementary information into the manual code, such as pre-conditions, invariants and/or post-conditions.

SPARK Ada[6] is an approach where Ada has been extended [BAR 03] in order to introduce additional elements (pre, post and invariant) and a sequence of adapted tools has been defined.

Objective

In [BOW 95] and [ARA 97], we have the first feedback from industrialists regarding formal techniques, and in particular feedback on the B method, Lustre language [HAL 91, ARA 97] and SAO+ (SCADE®'s predecessor). Other works, such as [MON 00, MON 02, HAD 06] provide an overview of formal methods from a scientific point of view.

With regards to the presentation of context and the state of the literature, our objective is to present concrete examples of the industrial uses of formal techniques. By formal techniques, we mean different approaches based on mathematics, which enable us to demonstrate that a software application respects a certain number of properties.

---

1 See www.mathworks.com/ products/polyspace/.
2 See www-list.cea.fr/labos/fr/LSL/caveat/ index.html.
3 See web www.absint.com.
4 To find out more, see web frama-c.com.
5 See www.astree.ens.fr.
6 See www.altran-praxis.com/spark.aspx contains additional information about SPARK Ada.

It is worth noting that the standard use of formal techniques consists of running specification and/or design models. Increasingly, however, formal techniques are seen as a way of carrying out verification (static code analysis, proof that the property is respected, proper management of floater calculation, etc.).

This book is part of a series that covers four different aspects:

– this first volume concerns industrial examples of the implementation of formal techniques based on static analysis, such as abstract interpretation: there are examples of the use of Astrée (Chapter 2), Caveat (Chapter 2), CodePeer (Chapter 5), Frama-C (Chapters 2 and 6) and Polsypace® (Chapters 3 and 4) tools;

– the second volume gives industrial examples of B method implementation [ABR 96];

– the third volume is dedicated to the presentation of different modeling techniques, such as SCADE® [7] [DOR 08], ControlBuild[8] and MaTeLo[9].

– the fourth volume is dedicated to the presentation of the railway sector's application of formal technics.

In conclusion to this introduction, I would like to thank all the industrialists who have given their own time to write these chapters, each one being even more interesting than the next.

Bibliography

[ABR 96] ABRIAL Jr., *The B Book – Assigning Programs to Meanings*, Cambridge University Press, Cambridge, August 1996.

[ANS 83] ANSI, ANSI/MIL-STD-1815A-1983 Standard, ADA Programming Language, ANSI, 1983.

[BAI 08] BAIER C., KATOEN J.P., *Principles of Model Checking*, MIT Press, London, 2008.

[BAR 03] BARNES J., *High Integrity Software: The SPARK Approach to Safety and Security*, Addison-Wesley, London, 2003.

[BOW 95] BOWEN J.P., HINCHEY M.G., *Applications of Formal Methods*, Prentice Hall, Upper Saddle River, 1995.

7 SCADE® is distributed by Esterel-Technologies, see www.esterel-technologies.com.

8 To find out more about the ControlBuild tool, see www.geensoft.com/en/article/controlbuild.

9 To find out more about MaTeLo, see www.all4tec.net/index.php/All4tec/matelo-product.html.

[COU 00] COUSOT P., "Interprétation abstraite ", *Technique et Science Informatique*, vol. 19, p. 155-164, no. 1-2-3, Hermès, Paris, 2000.

[DIJ 75] DIJKSTRA E.W., "Guarded commands, nondeterminacy and formal derivation of programs", *Communications of the ACM*, vol.18, no. 8, pp. 453-457, 1975.

[DIJ 76] DIJKSTRA E.W., *A Discipline of Programming*, Prentice Hall, Engelwood Cliffs, 1976.

[DOR 08] DORMOY F.X., "Scade 6 a model based solution for safety critical software development", *Embedded Real-Time Systems Conference*, Toulouse, France, 2008.

[HAD 06] HADDAD S. (ed.), KORDON F., PETRUCCI L., *Méthodes Formelles pour les Systèmes Répartis et Coopératifs*, Hermès Lavoisier, Paris, 2006.

[HAL 91] HALBWACHS N., CASPI P., RAYMOND P., PILAUD D., "The synchronous dataflow programming language Lustre", *Proceedings of the IEEE*, no. 9, vol. 79, pp. 1305-1320, 1991.

[HOA 69] HOARE CAR, "An axiomatic basis for computer programming", *Communications of the ACM*, vol. 12, no. 10, pp. 576-583, 1969.

[JON 90] JONES C.B., *Systematic Software Development using VDM*, (2nd edition), Prentice Hall, Engelwood Cliffs, 1990.

[MON 00] MONIN J.F., *Introduction aux Méthodes Formelles*, Hermès, Paris, 2000.

[MON 02] MONIN J.F., *Understanding Formal Methods*, Springer Verlag, Heidelberg, 2002.

[OFT 97] OBSERVATOIRE FRANÇAIS des TECHNIQUES AVANCEES (OFTA), *Applications des Méthodes Formelles au Logiciel*, vol. 20, Arago, Masson, Paris, June 1997.

[SPI 89] SPIVEY J.M., *The Z Notation – a Reference Manual*, Prentice Hall, Engelwood Cliffs, 1989.