The B method takes up floating-point numbers

Lilian Burdy, Jean-Louis Dufour, Thierry Lecomte

The B method takes up floating-point numbers

Lilian Burdy[1], Jean-Louis Dufour[2], Thierry Lecomte[1]

1: ClearSy – Aix en Provence – France – www.clearsy.com
2: Sagem, groupe Safran, Avionics division – Eragny sur Oise – France – www.sagem-ds.com

Abstract: For a long time, formal methods have ignored floating-point computations. About ten years ago this has changed, and today specification languages and tools are in use in research and pre-industrial contexts. Better late than never: the B method, which has been the first formal method to prove real-size software, will soon be able to prove the correctness of floating-point computations. This paper gives the motivations, the philosophy and the first impacts on the AtelierB tool.

Keywords: formal methods, B method, floating-point arithmetic, DO-178

1. Motivation.

The industrial penetration of formal methods is very context-dependent: in railway, from the mid-nineties, middle-size applications (typically 20K-30K Lines of Code) are proved functionally correct with the B method [DehMej94]; in avionics, ten years later, much bigger applications are formally proved with the Astree analyzer, but only low-level correctness is addressed [DelSou07].

This seeming "conceptual regression" has two main reasons, which tend to lessen:
- Avionics is not a culturally easy target: DO-178B doesn't easily allow to replace tests by proofs (the only example is the use of CAVEAT by Airbus [SouWieDel09]). The C version, to be published in 2012, introduces significant clarifications and guidelines as regards formal methods: with qualified tools, they can now be routinely accepted as a substitute for some low-level tests, which is the economic prerequisite for an industrial use.
- Avionics is not a technically easy target: floating-point ("FP" thereafter) computations are the rule, and not the exception (the opposite of railway); and FP correction is one order of magnitude more complex than integer correction [Mon08]. But today, FP "proof technology" is rapidly maturing, thanks among others to a tool-independent specification language, ACSL [BolFil07], and a tool collaboration framework, Frama-C [AyaMar10].

So it is time to functionally prove avionics applications: an "easy" way is to transfer the FP technology to the B method. Unfortunately, the transfer is not so "easy", because compatibility must be ensured with the existing B conceptual and practical framework for integer algorithms modelling and proof. To understand the current limitation of the B language to integers, it is useful to go back to its origins [ChaGal89] [GuiHen90].

2. Integers, and the railway origins of the B method.

The "A" commuter line of the Parisian railway network opened at the end of 1977, but it soon became clear that it was a victim of its success, and it became overcrowded. The operator (French Railway and Parisian Transportation Company) decided to reduce the headway (the interval between trains), with a driving assistance system called SACEM. Software development began in 1983 in the consortium Alstom/Matra/CSEE, and as it was the first time software would ensure safety in a French railway system, with moreover some complexity (7Kloc on-track, 14Kloc on-board), it was decided to replace some of the tests by formal proofs. Retrospectively it was a brave decision, because Hoare logic was only 15 years old, and no tool existed! The other original technical solution was the safe computing platform, which was not based on redundancy, but on coding: the "coded processor". This technique can not handle FP computations, so distance, speed and energy calculations were designed with fixed-point computations.
However, in 1987, audits carried out by two independent software experts, Jean-Raymond Abrial and Stéphane Natkin, made explicit the weak point of Hoare logic, which had already been observed in-house: its inability to scale up. Proofs were effective on leaf-procedures, but on intermediate and high-level procedures the formal specification became enormous and worse not traceable to the informal specification.
J.-R. Abrial, who is one the fathers of the Z specification method (1980), had already at that time

turned to the next phase and had began to define what could be a formal development [Abr84]. Indeed, the missing link was at the very same time under development: (data and control) refinement logic [Bac81] [Mor90]. In his audit report Abrial proposed to bridge the gap between the informal specification and the previous formal activities by a "formal re-expression": it is the first application of the "B method", performed manually in 1988 (the ancestor of AtelierB, the "B-Tool", was already in development in British Petroleum Research, but it was first released in 1992). The commissioning of SACEM took place in 1989.
Since, the major B developments also occurred in railway systems, where the typical applications are either interlocking or speed/interval control: the first ones need only Booleans, the second ones can be satisfied with fixed-point numbers (especially when the "coded processor" is used). Concerning the speed/interval control applications, a key point is the relative simplicity of the algorithms: they often form the MONitoring part of an "asymmetric" fail-safe COMmand/MONitoring architecture: the COM part is complex but not safety-related (for example, speed setpoint computation, to drive engines), because the simpler MON part ensures safety (same example: speed control, to trig emergency braking).

3. The avionics context, FP numbers and their resolution.

In avionics this is the opposite: integers are in the minority, and this is particularly true in navigation systems, where 64bits FP numbers ("double" precision in [IEEE] terminology) are common. The algorithms are also more complex, let's have a look at two typical examples:

- A FADEC (Engine Controller) drives fuel injection with regulation loops working on temperatures, pressures, throughputs and rotation speeds. The relative accuracy of the outputs is around 10^{-3}, and to obtain it the relative accuracy of the basic operations must be around 10^{-6} (with a good margin: ten years ago, automotive engine controllers performed similar computations with 16bits fixed-point numbers). So a FADEC uses only 32bits FP numbers ("single" precision in [IEEE] terminology), which offer a relative resolution of $2^{-23} \approx 1.2.10^{-7}$.
- An INS (Inertial Navigation System) gives your position on earth (sometimes in space): an absolute error of 1m corresponds to a relative accuracy of 1m / 40000km = 2.5 10^{-8}, which is already better than single precision. This could be still compatible with 32bits fixed-point numbers (which were used in the early "gimballed" INS), but when you consider all the operations needed in a modern "strap-down" INS (two examples: the high frequency integration of the gyroscopic informations, the Kalman filters), the accuracy needed on the basic arithmetic operators is around 10^{-13} (hopefully compatible with the resolution of double precision: $2^{-52} \approx 2.10^{-16}$). To our knowledge, the only industrial case where double precision is not enough is the representation of UTC time in a GPS system, where the resolution is the nanosecond and the range spans decades.

And the complexity of the algorithms can not be mitigated by the architecture, as was the case with the fail-safe COM/MON architecture of railway: avionics means "fail-operational" and avionics architectures are duplication (FADEC) and triplication (INS). To ensure the absence of hazardous common modes between the redundant channels, the complete function is declared safety-critical (leading on software to "DAL A" in avionics jargon or to diversity). Moreover, for an INS, the safety criterion coincides with the function ("estimate the position with a given maximum error"), so even in an hypothetical fail-safe context, it is difficult to imagine a simpler MON function which could ensure alone the requested level of safety.

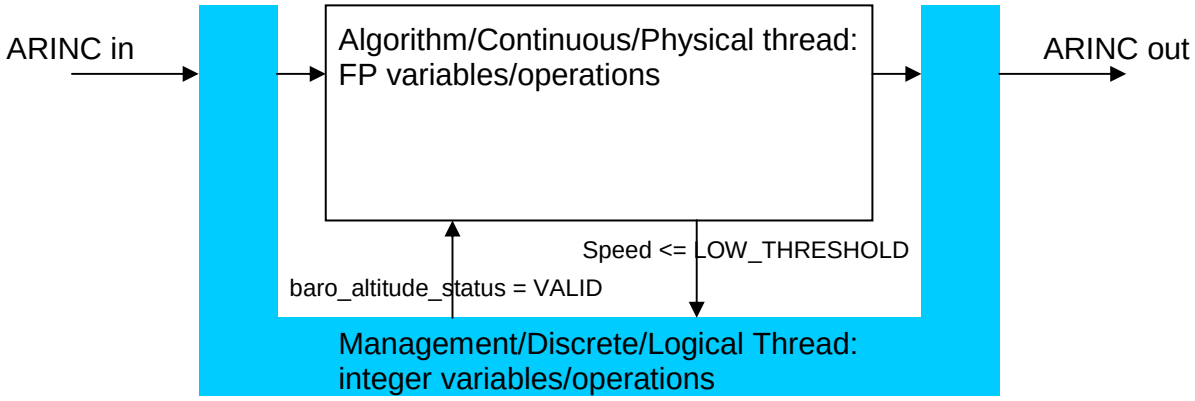
4. The conceptual segregation between FP and non-FP computations, and the associated false good idea.

From now on, we will focus on INSs. In an INS the predominance of FP numbers over the other data-types is such that it can suggest false good ideas. To illustrate it, here are three observations on a typical INS application software (lower layers are excluded):

- More than a half of the atomic variables (standard ones or members of structures or arrays) are of a FP type (exclusively "double" in recent systems),
- Almost no integer variable contains "navigation" information (position, attitude …). The only two exceptions are
    - In the ARINC 429 input/output interfaces, temporary integers are converted to/from FP numbers to analyse/create messages,

    - In tabulated functions, temporary integer indexes are created from FP numbers.
- Almost all integers are used to encode states (enumerated types). The only three exceptions are
    - Counters, which represent durations for triggering state transitions,
    - Array indexes for vector/matrices algorithms (typically between 0 and 2 in static loops),
    - Array indexes for buffer management in the input/output interfaces.

It means that conceptually, such an application can be viewed as two threads loosely coupled: the "algorithm" or "continuous" or "physical" thread which performs all FP operations and only those ones, and the "management" or "discrete" or "logical" thread which computes the operating mode of the INS. Except for input/output messages, they exchange only enumerated values, for example algorithm sending to management the Boolean "speed <= LOW_THRESHOLD", and management sending to algorithm the Boolean "baro_altitude_status = VALID".



This conceptual segregation can lead to the impression that it is possible to segregate also the design and the correctness:

- A B model will specify/design only the discrete part, and the continuous part will be seen through "basic machines", that is machines which are not refined. This way, as interfaces are of an enumerate nature, FP numbers don't have to be introduced in B.
- The continuous part is proved with a more adapted tool, like Frama-C, and an informal traçability is performed between the basic machines and the ACSL specification.

The problems are that

- this segregation is purely conceptual, and the algorithms are never designed in this way: the gap between the model and the software would be important,
- the traçability between B and ACSL is informal because the specification languages are different,
- the level of abstraction is also different, because Frama-C has today few refinement capabilities.

So, the introduction of real and FP numbers in B is a necessity.

5. IEEE-754 FP numbers and a possible first misconception

One of the interests of the IEEE standard is that FP operations are "total functions": they are defined for every input value, with the help of the three "exceptional" FP numbers +/-INFINITY and NaN ("Not a Number"). For example, FP addition has the following behaviour (with double precision numbers):

- 1E308 + 1E308 = INFINITY, because the maximum representable number is less than 2E308,
- INFINITY + x = INFINITY, for any x greater than – INFINITY,
- INFINITY + (– INFINITY) = NaN,
- NaN + x = NaN, for any x.

The equation "INFINITY – INFINITY = NaN" may seem surprising, because every finite number x is solution of "INFINITY = INFINITY + x": an appropriate name could have been "Undefined Number", but the name "NaN" was chosen with reference to sqrt(-1).

Moreover, when an operation generates one of the three exceptional values, a "sticky" bit is set in a status register ("sticky" means that it is not automatically reset by the next operation if it generates a finite result, it must be explicitly reset).

The possible misconception is a naïve one: it is true that every operation gives a sensible result, in that

- when a “finite” (means “non-exceptional”) result is obtained, it is correct,
- when INFINITY (resp. –INFINITY) is obtained, either there is no “real” result (ex: 1/0), or the “real” result is in the range]MAXDBL, +∞[(resp.]-∞, -MAXDBL[), with MAXDBL ≈ 1.8E308.

Informally, it means that we can trust (finite) results. Note that this property is wrong on integers: for example on 16-bits signed integers, 30000+30000 gives something, but certainly not 60000 (which is not representable).
It can lead to the quick conclusion, particularly on fail-safe architectures, that there is nothing to prove on FP algorithms, because exceptional results will point out problems. This is obviously wrong, for a lot of reasons:

- The property “a finite result is a correct result” is true for individual operators, true for composed expressions which contain only FP sub-expressions, but wrong in case of Boolean sub-expressions (tests):
    - 2*1E308 >= 0 evaluates to TRUE, and the intermediate INFINITY has been forgotten: this is the raison d’être of the sticky bits, which memorise all creations of exceptional values,
    - The C expression “ 2*1E308 >= 3*1E308 ? 1.0 : 2.0 ” evaluates to 1, whereas the “real” result is 2.
- The property “+INFINITY denotes a positive number” is also wrong, due to the non-associativity of FP addition:
    - “ (2*1E308 – 1.5E308) -1.5E308 “ evaluates to +INFINITY, whereas the “real” result is -1E308.
- But the good reason is that this misconception indicates confusion between low-level faults and functional faults: if an overflow occurs, for sure there is a bug, but conversely if you write “>=” instead of “<=” or “+” instead of “-“, it will not necessarily lead to an exceptional number.

6. The link between architecture and safety properties

B is a modular specification/design language, a module is called a “machine”. It’s a kind of object with
- variables (the “state”), read-only from outside, which must satisfy a property called the “invariant”,
- operations which change the state, but in such a way to preserve the invariant.

In a fail-safe COM/MON architecture, the safety properties are naturally functional invariants of MON. An invariant is a property that each “instantaneous” state of the system must satisfy, and says nothing about the evolution of this state (excepted of course that the evolution must preserve the invariant). Here are some typical safety invariants of a railway system, which are very naturally formalized in B:

- on-track, an occupied section shall be followed by an empty section,
- on-track, the free distance in front of a train shall permit to stop it with the minimum guaranteed deceleration in emergency braking,
- on-board, if emergency braking is not activated, the speed shall be lower than the current limitation,
- on-board, if the doors are unlocked, the speed shall be null.

In a fail-operational system, safety invariants are not so common, because safety is synonymous of correct operation. For example in an INS, the ultimate property concerning attitude would be:

“the attitude estimated is the integral of the elementary rotations perceived by the gyroscopes since the system has been initialised, and the initial attitude was such that the initial specific force perceived by the accelerometers was zero in the XY plane”.

Of course, this is not easily formalised, even in B, so an informal refinement leads to:

“at each step, the new attitude is the previous one combined with elementary rotations perceived by the gyroscopes”.

The two key points are:

- the words “combined” or “integral” hide a rather complex algorithm,
- the property is not an invariant of the state but an evolution of the state.

This last point is not a problem from the point of view of the B method, it means simply that safety will

be formalized more in the operations than in the state.

So, the key word here is “complexity”: to the complexity of FP proofs (greater than the complexity of integer proofs) must be added the complexity of the needed properties.
Hopefully, interesting invariants exist, like the determinant of a rotation matrix which must be equal to 1 in the reals, or near to 1 in the FP numbers. But they are not so directly linked to safety as was the case in the fail-safe MON examples.

Today, in the railway industry, the economic balance between test-based verification and formal B design is neutral: it has not been proved that B reduces cost, but clearly B reduces risk and increases confidence. Tomorrow, in the avionics industry, the introduction of formal methods has to be carefully planned with a regular progression in the size of the projects to maintain this balance.

7. Philosophy

The concrete and abstract FP numbers

The “concrete” ones are the FP numbers as described by the IEEE standard and used in a programming language, the “abstract” ones are the model used in B.

Let’s begin with the concrete numbers: the target is the IEEE-754 FP number system, with the following parameters/contraints :
- “simplicity”: the application contains only single (32 bits; for example engine control) or double (64 bits; for example inertial navigation) numbers: no mix between these two precisions.
- “state of the art”: the unique rounding mode is “to nearest, ties to even”; bearing in mind that in other contexts, other rounding mode may be mandatory (see [MulCol10] p. 95 on the accounting context).
Denormalized numbers are of course included (because they are mandated by the IEEE standard; we precise this because Intel x86 and Motorola/IBM PowerPC architectures make their exclusion possible for efficiency reasons).
The exceptional numbers +/-INFINITY and NaN are forbidden as nominal values, and are symptomatic of an abnormal behaviour: usually in safety-critical systems, their occurrence leads to task suppression and loss of equipment. Proof obligations must ensure that they will not be generated during the computation.
Concerning the operations, from the IEEE standard only the 4 basic FP operations are used, the comparisons, the “floor” function and the conversions to and from integers. The “remainder” function is not used and is replaced by a “modulo” function, useful typically to maintain angles between 0 and 2*pi. The sqrt function is not used, because it is too accurate.

This target is modelled in B by about the same set, with the following exceptions:
- +/-INFINITY and NaN are not included, because the B philosophy is “offensive programming” (the converse of “defensive programming”): we don’t have to dynamically test for the presence of problems, because we statically prove that problems are impossible. It means that FP operators, which were total functions in “real life”, become partial functions (with preconditions) in B.
- +0 and -0 are both modelled by “0”.
It means simply that the B FP numbers are mathematically a subset of R (the real numbers), which is not the case for the IEEE numbers.

Real numbers

A model of R with its 4 basic operations, comparisons, “round” and FP conversions, “floor” and integer conversions is also added, with two goals:
- specify algorithms with a high level of abstraction
- implement algorithms in an ideal world, to simplify the proofs; this second goal has only an exploratory purpose, and has no meaning from a certification point of view.

Contrary to ACSL, Integers are not and will probably never be a subset of Reals, because the two sets have very different formal properties, and in practice “communicate” very rarely. Whether FP numbers should be a subset of Reals is an open point.

8. Impacts on AtelierB

Evolution of the B language

Until now, the B language has only supported Boolean and integer basic types, as safety critical railway applications do not need other types. For this purpose coefficients are applied to each data of a B model in order to obtain the desired level of accuracy while retaining the semantics of integer arithmetic on computer (32-bit, non-overflow checking, etc.).

The introduction of support for FP numbers in the B language was accompanied by the addition of real numbers mathematics, the underlying idea being to be able to specify a treatment with real numbers and to ensure that its floating point implementation is "close" to its specification.
The first choice was to fully distinguish real and floating point numbers. Indeed, these two mathematical objects having no or very few properties in common, it seemed more appropriate to have things inconsistent in terms of the B language. The REAL base type, distinct from INTEGER, has overloaded arithmetic operators except the division operator which is specialized (the semantics of the integer division being too specific).
The FLOAT type in turn has specific arithmetic operators.

Conversion operators between integers, real and floating point numbers have also been developed to make this extension usable.

Impact on tools

Related typing and well-definedness rules (division by zero, access to array elements, etc.) were adapted to the specific language extensions. The proof obligations normalization rules (mathematical lemmas to prove in order to verify B model correctness) must also be suited. Finally the definition of the “B language for implementation” (called B0) has been extended.
The distinction between REAL and INTEGER (the property "INTEGER is included in REAL" is false) allows keeping validation rules database untouched. This rules database, and the tools making use of it, that have been validated and certified during industrial developments will not have to be completely redeveloped but simply to be extended without affecting the validity of the existing data.

Perspectives

An Atelier B CASE tool implementing these extensions has been developed and used to validate the proof of concept. It allows generating proof obligations for real and floating point models. The challenge now is to be able to prove the resulting proof obligations. Proof of real numbers can be achieved by extending the rule-based theorem prover of Atelier B. Proof of floating point numbers in turn requires the implementation of other techniques and tools that will be subject to further work.
These studies have demonstrated the feasibility of this approach for the modeling phase and the generation of proof obligations. The feasibility of the automation of mathematical proof is needed to assess on its eligibility in the aerospace and automotive worlds.

9. Conclusion

Experiments have started in Sagem with Reals on simplified algorithms of INSs. They warn us against the complexity of algorithmic specification. One of the answers is modularity and reuse. The next step, which is to experiment with FP numbers, has not yet started, but it is clear that for a basic economic reason, we can not not fall into the complexity of proofs based on an exact model of FP numbers and operations: an approximate model has to be found, in the sense that FP rounding will not be exactly described, at the cost of some loss of precision. To permit proofs, maybe it will impact the way we write applications, but if it leads only to simple design/coding rules, it will be acceptable.

10. References

[Abr84] Jean-Raymond Abrial
The mathematical construction of a program
Science of Computer Programming 4 (1984) 45-86

[AyaMar10] Ali Ayad, Claude Marché

Multi-prover verification of floating-point programs
5th intl. Joint Conference on Automated Reasoning, 2010
See also the latest ACSL description at http://frama-c.com/download.html

[Bac81] Ralph-Johan Back
On correct refinement of programs
Journal of Computer and System Sciences 23, 49-68 (1981)

[BolFil07] Sylvie Boldo, Jean-Christophe Filliâtre
Formal verification of FP programs
18th IEEE int. symp. on comp. Arithmetic, Montpellier (France), June 2007, p. 187-194

[ChaGal89] Pierre Chapront, Christian Galivel
Results of a safety software validation: SACEM
Proceedings of the IFAC CCCT'89 symp. (Control, Computers, Communication in Transportation)

[DehMej94] Babak Dehbonei, Fernando Mejia
Formal methods in the railways signalling industry
FME '94: Industrial Benefit of Formal Methods, LNCS 873, 1994, p. 26-34

[DelSou07] David Delmas, Jean Souyris.
Astrée: from Research to Industry.
14th International Static Analysis Symposium, LNCS 4634, 2007, p. 437—451

[IEEE] IEEE-754-1985
IEEE Standard for Binary Floating-Point Arithmetic

[GuiHen90] Gérard Guiho, Claude Hennebert
SACEM software validation
Proceedings of the ICSE'90 (Intl. Conf. on Software Engineering)

[Mon08] David Monniaux
The pitfalls of verifying FP computations
ACM Trans. on Programming Languages and Systems (TOPLAS) Vol. 30 Issue 3, May 2008

[Mor90] Carroll Morgan
Programming from specifications
Prentice Hall, 1990 – online: http://www.cs.ox.ac.uk/publications/books/PfS/

[MulCol10] Jean-Michel Muller, and coll.
Handbook of FP arithmetic
Birkhaüser, 2010

[SouWieDel09] Jean Souryis, Virginie Wiels, David Delmas, Hervé Delseny
Formal verification of avionics software products
Formal Methods 2009, LNCS 5850