Properties of the subtraction valid for any floating point system

Sylvie Boldo, Marc Daumas

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Properties of the subtraction valid for any floating point system

Sylvie Boldo, LIP
Marc Daumas, CNRS

No 4473

Juin 2002

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4473--FR+ENG

INRIA
RHÔNE-ALPES

Properties of the subtraction valid for any floating point system

Sylvie Boldo, LIP
Marc Daumas, CNRS

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arénaire

Rapport de recherche n ° 4473 — Juin 2002 — 19 pages

Abstract: We start in this text with a very generic definition of floating point systems. We show that just a few very natural necessary conditions are sufficient to focus down to two classes of implemented floating point arithmetic. Later, we prove that, for all the existing implementations, high level properties such as Sterbenz's theorem are satisfied. We finish this text by focusing on the differences between an IEEE-754 compatible unit and Texas Instrument TMS/SMJ 320C3x digital signal processing circuit that is recommended for avionics and military applications. The results presented in this text have been validated by the Coq automatic proof checker to build confidence for later implementations in critical systems such as an aircraft flight control primary or secondary computer.

Key-words: Digital signal processing, Floating point, Avionics, Formal proof.

(Résumé : tsvp)

Authors e-mail addresses: Sylvie.Boldo@ENS-Lyon.Fr, Marc.Daumas@ENS-Lyon.Fr. This text is also available as a research report of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP`.

Propriétés de la soustraction valides pour tous les systèmes à virgule flottante

Résumé : Nous commençons ce texte par une définition très générale des systèmes de représentation des nombres à virgule flottante. Nous montrons que quelques conditions naturelles permettent de se restreindre à deux grandes classes de systèmes implantés de notation à virgule flottante. Par la suite, nous prouvons la validité de propriétés de haut niveau telles que le théorème de Sterbenz pour tous les systèmes implantés. Nous finissons ce texte par une comparaison entre une unité compatible IEEE et le circuit de traitement numérique du signal TMS/SMJ 320C3x recommandé pour l'avionique et les applications militaires. Les résultats présentés dans ce texte ont été validés par l'outil de vérification de preuves Coq pour une sûreté accrue comme cela s'avère nécessaire pour des applications telles que l'ordinateur primaire ou secondaire de commandes de vol d'un avion.

Mots-clé : Traitement numérique du signal, Virgule flottante, Avionique, Preuve formelle.

1 Introduction

In 1974, Sterbenz [20] presented a theorem about the exact subtraction of two floating point numbers x and y when they are very close one from another, that is

$$\frac{y}{2} \leq x \leq 2y.$$

The theorem stating that $x - y$ is exact under the preceding condition was presented for any radix provided the hardware was accurate enough. More recently, other authors [8, 10] presented similar results with an emphasis on didactic aspects.

We have recognized in [6] that Sterbenz's theorem is not a property of the computing hardware but rather a property of the floating point number representation. Given x and y, the question is to know whether or not $x - y$ can be represented in the working floating point system. This is clearly the key necessary condition for the implemented floating point subtraction $x \ominus y$ to return the exact result $x - y$.

With IEEE-like behavior, any floating point operation is cut down to two steps. An intermediate result is first computed to sufficient accuracy and then rounded. The designer must guarantee that the system always returns the result as if the infinitely precise mathematical operation were rounded. For example the subtraction is implemented as the composition of two mathematical functions, namely, the subtraction $(-)$ and the user specified rounding function $(\circ)$

$$x \ominus y = \circ(x - y).$$

The details of the implementation are not relevant to the user since knowing the rounding function is sufficient to deduce the value returned by any operation. Users usually expect the rounding function to be a monotonous (non decreasing) projection of the real numbers over the set of the machine floating point numbers. The later property implies that for any floating point number v,

$$\circ(v) = v.$$

Establishing Sterbenz's equality does not require any additional knowledge on the rounding function provided it is a projection.

In Section 2, we describe more precisely our formalization of the floating-point system. In Section 3 we discuss key properties of this system. The proposed representation is very redundant but we will see that any machine number has one single canonical representation that can be used in hardware. We will quickly present that a floating point system must handle denormal numbers in order to verify Sterbenz's theorem on very small numbers. Finally as the existence of a negation will become key to Sterbenz's theorem, we will present a strong necessary and sufficient condition for a generic number system to be stable by negation.

Section 4 presents our results about Sterbenz's theorem and relates them to real hardware implementations. We have focused on IEEE 754 standard implementations and on Texas Instrument TMS 320C3x series described in Section 5 with its military grade processes. The SMJ 320C3x circuits can be used in avionics and military applications such as the flight control primary or secondary computer (FCPC / FCSC) [14]. Past studies have proved that an automatic proof checker must be used for such critical systems [16]. This work ends with concluding remarks and perspectives for further developments.

2 Our formalization of a floating point system

All the results have been developed and validated using Coq [11]. It is a theorem checking system based on the Curry-Howard isomorphism. Systems like Coq allow the user to define new objects and to derive consequences of these definitions formally while checking every detail. The Coq tool is based on higher-order logic. With such an expressive logic, it is possible to state properties in their most general form. For example, universal quantification has been used to state properties

that are true for an arbitrary rounding mode. Theorem provers have already been successfully used to mechanically check the correctness of floating-point algorithms [17, 9], and with a strong emphasis for avionics [2].

We will present in this text the behavior of a generic floating point system in regard to Sterbenz's theorem. We define a generic floating point system from a mapping of $\mathbb{Z}^2$ onto $\mathbb{R}$

$$(n, e) \hookrightarrow n\beta^e$$

where β is a constant integer strictly greater than one called the radix of the floating point system. Later n will be called the mantissa and e the amplitude.

In Coq, the set is defined by the `float` type defined below in ASCII

```
Record float : Set := Float {
  Fnum: Z;
  Fexp: Z }.
```

and its value is obtained by using the `FtoR` function

```
Definition FtoR := [x : float]
  (Rmult (Fnum x) (powerRZ (IZR radix) (Fexp x))).
```

Two pairs are **equivalent** if they are mapped to the same real value. This equality will be noted as $=_{\mathbb{R}}$. Coq files are hardly understandable for a non-Coq user, theorems and definitions can be presented using a integrated pretty printer as shwon in the appendix of this research report. For example:

Definition 1 *FtoR* $:= x :$ *float* $\longmapsto$ *Fnum*$(x) \times \beta^{Fexp(x)}$

All the quantities treated by a computer system must fit into a finite field, we focus our interest on pairs (n, e) such that n and e are bounded. For practical reasons, we do not use an upper bound on the amplitude and a **bounded floating point pair** is such that

$$n \in \{-N_i, \cdots, N_s\} \qquad \text{and} \qquad e \geq -E_i.$$

That is to say in Coq:

Definition 2 *FboundedI* $:= b :$ *FboundI, d :* *float* $\longmapsto$
$(-vNumInf(b) \leq Fnum(d)) \bigwedge (Fnum(d) \leq vNumSup(b))$
$\bigwedge (-dExp(b) \leq Fexp(d))$

A sectioning mechanism with implicit parameter management transforms Sterbenz's theorem with our floating point library so that it states that

> "*for any radix greater than one, for any floating-point system, for all floats x and y, if x and y are bounded, and if $\frac{y}{2} \leq x$ and $x \leq 2 \times y$ then there exists a bounded float z such that $z =_{\mathbb{R}} x - y$*".

Unfortunately, this assertion is false. For example, let the radix be two and the format such that the mantissa is between -1100_2 and 1111_2. Let x be $(1111_2, 0)$ and y be $(1110_2, 1)$. Both x and y are bounded, they are such that $\frac{y}{2} \leq x \leq 2 \times y$ but $x - y$ is -1101_2 and this value cannot be represented exactly in this floating point system. We will later give a list of necessary conditions for the assertion to be true.

Proofs are built interactively using high-level tactics that may solve some of the "easy" subgoals. We used pcoq [1]: a working environment for the Coq theorem prover with a nice graphical interface and the pretty printer.

At the end of each proof, Coq records a proof object that contains all the details of the derivation and ensures that the theorem is valid. The object can be double checked for life critical applications by a tool such as BindLib, a program independent of the Coq development.

The proofs for this work can be downloaded through the Internet at the address

`http://www.ens-lyon.fr/~sboldo/coq`.

They inlcude the current development of the floating point library available at

`http://www-sop.inria.fr/lemme/AOC/`.

All the following theorems have been proved using this very general formalization. Unless explicitly specified the properties hold for any radix greater than one and any bound on the mantissa.

3 Basic properties of the set of bounded floating point numbers

3.1 Multiple representations

Contrary to IEEE-like behavior, the proposed library defines possibly many bounded floating point pairs with the same value. For example, the three radix two floating point pairs $(1100_2, 4)_2$, $(110_2, 5)_2$ and $(11_2, 6)_2$ share the same real value $3 \times 2^6 = 192$. This fact can be disturbing as one real value can be associated to many different bounded floating point pairs that do not have the same properties.

In order to retain common floating point behavior, we define a canonical pair for each bounded pair. This pair is meant to represent the actual fields stored in a computer that are associated to the number. A pair is **normal** if it is bounded and its amplitude cannot be reduced by multiplying the mantissa by the radix, that is

$$n \times \beta \notin \{-N_i, \cdots, N_s\}.$$

A pair is **denormal** if it is bounded and the amplitude reduction is blocked by the fact that it uses already the minimal accepted amplitude despite the mantissa being small enough to be multiplied by the radix. That is

$$n \times \beta \in \{-N_i, \cdots, N_s\} \quad \text{and} \quad e = -E_i.$$

Any bounded pair is equivalent to one unique pair either normal or denormal. The later pair is called the **canonical** representation. This fact is proved by several theorems. The first one, `FcanonicIUnique` states that if p and q are two canonical floating-point numbers such that $p =_{\mathbb{R}} q$ then p and q are syntactically equal (Leibniz's equality).

Other theorems prove the correctness of the `FnormalizeI` function defined below to construct the canonical representation from any bounded representation:

```
Fixpoint FNIAux [v, N, q : nat] : nat :=  Cases q of
 0 => 0
|(S q') => Cases
  (Zcompare (Zmult (Zpower_nat radix q') v) (Zmult radix N)) of
     INFERIEUR => q' | EGAL => q' | _ => (FNIAux v N q') end
 end.

Definition FNI := [q, N : nat]  (pred (FNIAux q N (S (S N)))).

Definition FnormalizeI :=
   [b : FboundI] [p : float]
    Cases (Zcompare ZERO (Fnum p)) of
      EGAL => (Float ZERO (Zopp (dExp b)))
     | INFERIEUR => (Fshift radix (min
           (FNI (absolu (Fnum p)) (vNumSup b))
           (absolu (Zplus (Fexp p) (dExp b)))) p)
```

```
  | SUPERIEUR => (Fshift radix (min
        (FNI (absolu (Fnum p)) (vNumInf b))
        (absolu (Zplus (Fexp p) (dExp b)))) p)
end.
```

Expressing that the function is correct means that (i) if p is a bounded float, then the result FnormalizeI(p) is a bounded float (`FnormalizeIBounded`). It also means that (ii) the result is canonical (`FnormalizeIFcanonicI`) and such that (iii) the input pair p and the result pair are mapped to the same real value that is to say $p =_{\mathbb{R}}$ FnormalizeI(p) (`FnormalizeICorrect`). We omit these proofs as they are quite cumbersome but not difficult.

3.2 Negating a number

On IEEE-like number systems, the mantissa is stored with separate sign and magnitude, therefore $N_i = N_s$. This fact is not true on all floating point systems. Some hardware designers decided to use two's complement to store the mantissa as this is the case for Texas Instrument TMS 320C3x [21].

A bounded floating point number p can be negated if there exists another bounded floating point number q such that $q =_{\mathbb{R}} -p$. As we will see, almost any number can be negated even on systems based on the TMS 320C3x digital signal processors. The only two cases where a number cannot be negated cause either an overflow as the opposite of the least represented number is larger than the biggest number allowed in the number system or an underflow as the opposite of the least represented positive normal number is larger than the biggest negative number allowed in the number system. The second case would not occur on systems that handle denormal numbers.

The following theorem checked with Coq (`FoppBounded` and `FoppBoundedInv`) answers any question about negating a number. The cases study for a system that does not handle denormal numbers and for the upper bound on the amplitude are treated separately (`FoppBoundedExp`).

Theorem 1 *On a floating point system bounded by N_i, N_s and E_i with $N_i \neq N_s$, any bounded pair can be negated to a bounded float if and only if*

$$|N_i - N_s| = 1 \qquad \textit{and} \qquad \beta \mid \max(N_i, N_s).$$

Without loss of generality, we assume that $N_s > N_i$. As a consequence, any pair (n, e) with $n \in \{-N_i, \cdots, N_i\}$ can be easily negated by negating its mantissa. The pairs (n, e) with $n \in \{N_i + 1, \cdots, N_s\}$ can only be negated by manipulating the amplitude. Therefore, β should divide all the $n \in \{N_i + 1, \cdots, N_s\}$. That is possible only for $N_i + 1 = N_s$ if β divides N_s. On the contrary, if N_s is a multiple of β and $N_i = N_s - 1$, any bounded pair can be negated to find another bounded pair.

We have also proved that the negation is the only opposite on a system that handles denormal numbers: if $x \oplus y = 0$, then y is the negation of x. Rephrasing [13] we first prove that the distance between two floating point numbers is at least β^{-E_i} then we conclude in the `OppositeIUnique`:

Theorem 2 *On a floating point system bounded by N_i, N_s and E_i, let P be any rounding mode and x and y be two bounded floats. If $y \neq_{\mathbb{R}} -x$ and z is a rounded result of $x + y$ then $|z| \geq \beta^{-E_i}$.*

3.3 Usual definitions of radix complement

Radix complement may or may not be used depending on the convention for negative numbers defined by the author for radix $\beta > 2$. The three common definitions are equivalent when $\beta = 2$.

The interpretation where the sign digit is -1 when $\beta - 1$ is stored in the most significant digit leads to the bounds

$$N_i = \beta^{p-1} \qquad \text{and} \qquad N_s = (\beta - 1) \cdot \beta^{p-1} - 1$$

with p bits of mantissa ($p > 1$). If $\beta > 2$, $N_s - N_i > 1$ and some bounded pairs cannot be negated without rounding.

Some authors use the previous convention but restrict the leading digit to 0 or $\beta - 1$. In this case,

$$N_i = \beta^{p-1} \qquad \text{and} \qquad N_s = \beta^{p-1} - 1,$$

so any pair can be negated.

When the interpretation is read modulo β^p and the digits are balanced evenly with possibly an additional digit to the negative set, the bound are

$$N_i = \left\lfloor \frac{\beta^p}{2} \right\rfloor \qquad \text{and} \qquad N_s = \left\lceil \frac{\beta^p}{2} \right\rceil - 1.$$

If β is odd, the set is evenly balanced and $N_i = N_s$. If β is even, $N_i = N_s + 1$ and β divided N_i since $p > 1$.

As a conclusion, it seems natural to prefer a sign-magnitude or a two's complement notation for the mantissa. We will see in Section 5 that all the existing implementations use one of these two classes.

3.4 Denormal numbers

The number system that we have just defined handles denormal pairs (gradual underflow) as this helps write more robust codes [7]. Sterbenz's theorem cannot be true if denormal numbers are not allowed. Let λ be the lowest positive normal number. Its value is

$$\lambda = \left(\left\lfloor \frac{N_s}{\beta} \right\rfloor + 1 \right) \times \beta^{-E_i}$$

and the following floating point number is

$$\lambda^+ = \left(\left\lfloor \frac{N_s}{\beta} \right\rfloor + 2 \right) \times \beta^{-E_i}.$$

The quantities λ and λ^+ verify $\lambda^+/2 \leq \lambda \leq 2\lambda^+$ and

$$\lambda^+ - \lambda = \beta^{-E_i}$$

that is a denormal number. This example shows that without allowing denormal numbers, the subtraction of x and y under the conditions of Sterbenz's theorem may not be represented.

3.5 Lexicographic order

Many authors, including [3], have recognized that it is a nice feature for lexicographic order of the floating point pairs to coincide with the order of the represented real values. As this fact is not necessary trivial in a generic floating point system, we establish the two following `LexicoPosCanI` and `LexicoCanI` theorems.

Theorem 3 *On a floating point system bounded by N_i, N_s and E_i, for any canonical pair (n_x, e_x) representing x and any bounded pair (n_y, e_y) representing y*

$$0 \leq x \leq y \qquad \textit{implies} \qquad e_x \leq e_y.$$

Theorem 4 *On a floating point system bounded by N_i, N_s and E_i with $|N_i - N_s| \leq 1$, for any canonical pair (n_x, e_x) representing x and any bounded pair (n_y, e_y) representing y*

$$|x| < |y| \qquad \textit{implies} \qquad e_x \leq e_y.$$

The difference between the preceding theorems and the usual IEEE like situation arises from the fact that the magnitude of a floating point pair may not be represented or may use another amplitude. We establish the following corollary.

Corollary 1 *On a floating point system bounded by N_i, N_s and E_i with $|N_i - N_s| \leq 1$, for any canonical pair (n_x, e_x) representing x and any bounded pair (n_y, e_y) representing y*

$$e_x < e_y \qquad \text{implies} \qquad |x| \leq |y|.$$

This means that when $|N_i - N_s| \leq 1$, our floating point system behaves like a IEEE compliant implementation as far as lexicographical order is concerned.

When $|N_i - N_s| > 1$, we have a very different behavior. Here is an example that also shows that the bound on the difference $N_i - N_s$ is tight. We define a binary notation with the mantissa between -1001_2 and 111_2. The pairs $(100_2, 1)_2$ and $(-1001_2, 0)_2$ are canonical yet their magnitudes and their amplitudes are not in the same order. This cannot happen in IEEE compliant systems or on the TMS 320C3x.

4 Sterbenz's theorem

4.1 A first very general theorem

It is amazing to realize that the following theorem is true whatever the radix and the bounds N_i and N_s. Moreover, the proof has been upgraded automatically by the Coq proof checker from the previous proof `SterbenzAux` presented in [6] that was supposed to work only when $N_i = N_s$.

Theorem 5 *On a floating point system bounded by N_i, N_s and E_i with no assumption of a relation between N_i and N_s, for any bounded pairs (n_x, e_x) and (n_y, e_y) representing x and y such that*

$$y \leq x \leq 2y,$$

the difference $x - y$ can be represented by a bounded pair (n, e). Furthermore, the bounded mantissa n and the bounded amplitude e can be defined as

$$\begin{aligned} n &= n_x \beta^{e_x - \min(e_x, e_y)} - n_y \beta^{e_y - \min(e_x, e_y)} \\ e &= \min(e_x, e_y). \end{aligned}$$

On a floating point system where any bounded pair can be negated without rounding such as presented section 3.2, Sterbenz theorem `SterbenzOppI` stated below is proved by applying twice Theorem 5.

Theorem 6 *On a floating point system bounded by N_i, N_s and E_i where any bounded pair can be negated to another bounded pair, for any bounded pairs x and y such that*

$$\frac{y}{2} \leq x \leq 2y,$$

the difference $x - y$ can be represented by a bounded pair.

We prove the theorem correct when $y/2 \leq x \leq y$ by applying Theorem 5 to $X = y$ and $Y = x$ so that $X - Y = -(y - x)$ can be represented by a bounded pair.

4.2 Other systems

Although we presented in section 3 that it is most desirable to use a number system with a few natural properties including the fact that every bounded pair can be negated without rounding, we present now the `SterbenzI` very generic theorem. The details of the proof are available on the Internet.

Theorem 7 *On a floating point system bounded by N_i, N_s and E_i where $|N_i - N_s| \leq \delta$, for any canonical pair (n_x, e_x) representing x and any bounded pair (n_y, e_y) representing y such that*

$$\frac{y + \delta \beta^{\min(e_x, e_y)}}{2} \leq x \leq 2y,$$

the difference $x - y$ can be represented by a bounded pair.

5 Concluding remarks

5.1 Overview of existing hardware implementing sign magnitude

Most general purpose widely available processors use a sign magnitude representation. Some books [10, 15] even present the sign magnitude notation as the natural floating point notation. This notation is in use in the well-studied IEEE-754 compliant hardware. Some IBM systems use radix 10 [4] and a few of them retain a radix 16 compatibility mode [19]. Yet most systems use radix 2. For all these systems, Sterbenz's equality holds with all the natural properties presented in this work.

The properties presented here were scattered in the litterature as most of them have been published over the time. Yet the main motivations of this work was to compare the most common general purpose IEEE 754 based behavior to other implementations such as IEEE 854 compatible circuits, almost IEEE 854 behavior and non IEEE behavior as we see in the following.

5.2 Texas Instrument's two's complement notation

Texas Instrument uses in its TMS 320C3x the two's complement notation for the mantissa. This notation was also in use in Honeywell 6080N computer [18]. A different notation with the same mantissa range is studied in a exercise of [12]. This number system is well suited as all the natural properties (stability through negation, existence and unicity of an opposite and lexicographic order of the pairs) are still true and Sterbenz's equality holds. We do not have any knowledge of a working floating point unit that uses neither sign-magnitude nor the two's complement notation for the mantissa encoding.

The following theorem (`ReductRange` and `ReductRangeInv`) can be used to deduce that the set of the represented numbers is almost identical with an IEEE-compatible unit and the TMS 320C3x. Should Texas Instrument decide to implement denormal pairs and precise rounding the unit could be functionnaly IEEE-compliant.

Theorem 8 *The set of the real numbers represented on a floating point system bounded by* $N_i > 1$, $N_s > 1$ *and* E_i *is identical to the set of the numbers represented on a system bounded by* N_i, $N_s - 1$ *and* E_i *(respectively* $N_i - 1$, N_s *and* E_i*) if and only if*

$$\beta \mid N_s \qquad \textit{(respectively} \quad \beta \mid N_i\textit{)}.$$

All the theoretical results presented in this text prove that under a few assumptions there exists a bounded float that is the exact result of the subtraction. We have to look at the way the addition/subtraction is performed by the TMS 320C3x to be sure that this exact result is really returned by the unit.

Figure 1 presents a simplified version of the flowchart of the addition. This operation is first performed on extended precision and then rounded. The mantissa of one of the inputs is possibly shifted depending on the actual value of the difference of the amplitudes. The result of the addition of the mantissas lies in 30 bits. That means that 8 additional bits are used for the intermediate result. Adding a new case to the result of [10], we see that one guard bit is sufficient for Sterbenz's theorem to hold even using a different notation than the IEEE-like sign-magnitude for the mantissa. On the contrary, the Sterbenz's theorem does not hold if the user manipulates extended numbers rather than single precision numbers. In this case, the operation is performed without any guard bit and the result is not necessarily found by the floating point unit.

If no exception is triggered, the mantissa is accurate enough to hold the exact result and the result before rounding is the expected exact result. As this result was proved to be bounded, the rounding does not change it and the final result is exact.

We deduce immediately from $y/2 \leq x \leq 2 \times y$ that $x - y \in [-y/2; y]$ so no overflow can occur. If the exact result is a denormal number, the TMS 320C3x returns 0 as this processor does not handles such numbers.

In this text, we have shown that the floating point number system used for the TMS 320C3x defines almost the same real values as the system of an IEEE-compliant processor with a very

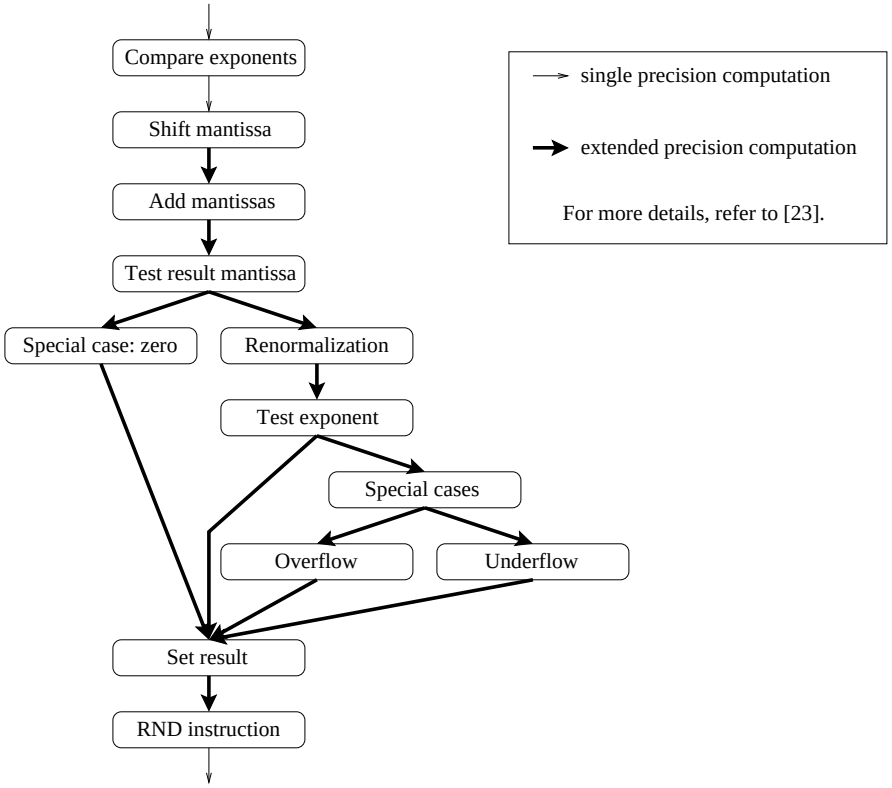

Figure 1: Flowchart for floating point addition followed by a RND instruction

different interpretation for the mantissa field. We have also shown that gradual underflow and correct rounding would be very sensible in such a system although neither was implemented. Finally, we have proved some very useful result about the TMS 320C3x such as Sterbenz's theorem provided no underflow occurred.

5.3 On automatic proof checking

Without a strong incentive on formal analysis of the TMS 320C3x, such work would probably not have been carried out. It has been made possible by the very formal and generic development of the proofs used in Coq. Odds are that such conclusions would scarcely be trusted if they were not checked by an automatic proof checker since the proofs are very technical and prone to many small mistakes that would not have been ruled out by experimental knowledge.

We will continue to investigate natural properties of floating point number systems as they lead us to necessary conditions on the number systems. In the case of this work, Sterbenz's equality and the possibility to negate a number are also key to analyze numerical software behavior such as [5].

6 Acknowledgments

We wish to thank Laurence Rideau and Laurent Théry for they work on the initial development of the Coq theory library.

References

[1] Amerkad, A., Y. Bertot, L. Rideau and L. Pottier, *Mathematics and proof presentation in Pcoq*, in: *Proceedings of Proof Transformation and Presentation and Proof Complexities*, Siena, Italy, 2001.
URL `http://www-sop.inria.fr/lemme/Laurence.Rideau/proof-pcoq.ps.gz`

[2] Carreño, V. A. and P. S. Miner, *Specification of the IEEE-854 floating-point standard in HOL and PVS*, in: *1995 International Workshop on Higher Order Logic Theorem Proving and its Applications*, Aspen Grove, Utah, 1995, supplemental Proceedings.
URL `http://shemesh.larc.nasa.gov/fm/ftp/larc/vac/hug95.ps`

[3] Coonen, J. T., *Specification for a proposed standard for floating point arithmetic*, Memorandum ERL M78/72, University of California, Berkeley (1978).

[4] Cowlishaw, M. F., E. M. Schwarz, R. M. Smith and C. F. Webb, *A decimal floating point specification*, in: N. Burgess and L. Ciminiera, editors, *Proceedings of the 15th Symposium on Computer Arithmetic*, Vail, Colorado, 2001, pp. 147–154.
URL `http://computer.org/proceedings/arith/`

[5] Daumas, M. and P. Langlois, *Additive symmetric: the non-negative case*, Theoretical Computer Science (2002).

[6] Daumas, M., L. Rideau and L. Théry, *A generic library of floating-point numbers and its application to exact computing*, in: *14th International Conference on Theorem Proving in Higher Order Logics*, Edinburgh, Scotland, 2001, pp. 169–184.
URL `http://link.springer.de/link/service/series/0558/bibs/2152/21520169.htm`

[7] Demmel, J., *Underflow and the reliability of numerical software*, SIAM Journal on Scientific and Statistical Computing **5** (1984), pp. 887–919.

[8] Goldberg, D., *What every computer scientist should know about floating point arithmetic*, ACM Computing Surveys **23** (1991), pp. 5–47.
URL `http://www.acm.org/pubs/articles/journals/surveys/1991-23-1/p5-goldberg/p5-goldberg.pdf`

[9] Harrison, J., *Floating point verification in HOL light: the exponential function*, Technical Report 428, University of Cambridge Computer Laboratory (1997).
URL `http://www.cl.cam.ac.uk/users/jrh/papers/tang.ps.gz`

[10] Higham, N. J., "Accuracy and stability of numerical algorithms," SIAM, 1996.
URL `http://www.ma.man.ac.uk/~higham/asna.html`

[11] Huet, G., G. Kahn and C. Paulin-Mohring, *The Coq proof assistant: a tutorial: version 6.1*, Technical Report 204, Institut National de Recherche en Informatique et en Automatique, Le Chesnay, France (1997).
URL `ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RT/RT-0204.pdf`

[12] Knuth, D. E., "The Art of Computer Programming: Seminumerical Algorithms," Addison-Wesley, 1997, third edition.

[13] Kulisch, U., *Rounding near zero*, in: *4th Real Numbers and Computers Conference*, Dagstuhl, Germany, 2000, pp. 23–29.

[14] Laurent, O., P. Michel and V. Wiels, *Using formal verification techniques to reduce simulation and test effort*, in: *International Symposium of Formal Methods Europe*, Berlin, Germany, 2001, pp. 465–477.
URL `http://link.springer.de/link/service/series/0558/papers/2021/20210465.pdf`

[15] Overton, M. J., "Numerical Computing with IEEE Floating Point Arithmetic," SIAM, 2001.
URL `http://www.siam.org/catalog/mcc07/ot76.htm`

[16] Rushby, J. and F. von Henke, *Formal verification of algorithms for critical systems*, in: *Proceedings of the Conference on Software for Critical Systems*, New Orleans, Louisiana, 1991, pp. 1–15.
URL `http://www.acm.org/pubs/articles/proceedings/soft/125083/p1-rushby/p1-rushby.pdf`

[17] Russinoff, D. M., *A mechanically checked proof of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7 processor*, LMS Journal of Computation and Mathematics **1** (1998), pp. 148–200.
URL `http://www.onr.com/user/russ/david/k7-div-sqrt.ps`

[18] Schryer, N. L., *A test of computer's floating-point arithmetic unit*, Technical report 89, AT&T Bell Laboratories (1981).
URL `http://cm.bell-labs.com/cm/cs/cstr/89.ps.gz`

[19] Schwarz, E. M., R. M. Smith and C. A. Krygowski, *The S/390 G5 floating point unit supporting hex and binary architectures*, in: I. Koren and P. Kornerup, editors, *Proceedings of the 14th Symposium on Computer Arithmetic*, Adelaide, Australia, 1999, pp. 258–265.
URL `http://computer.org/proceedings/arith/0116/0116toc.htm`

[20] Sterbenz, P. H., "Floating point computation," Prentice Hall, 1974.

[21] Texas Instruments, "TMS320C3x — User's guide," (1997).
URL `http://www-s.ti.com/sc/psheets/spru031e/spru031e.pdf`

Skeleton of the FnormI.v file

```
Require Import Float.
Require Import Fcomp.
Require Import Fop.
Require Import Paux.
Section FboundedI_Def.
Variable radix : Z.
Hypothesis radixMoreThanOne : (Zlt (POS xH) radix).

Local FtoRradix := (FtoR radix).
Coercion FtoRradix : float >-> R.

Record FboundI : Set := Bound {
    vNumInf: nat;
    vNumSup: nat;
    dExp: nat }.

Definition FboundedI :=
    [b : FboundI]
    [d : float]
     ((Zle (Zopp (vNumInf b)) (Fnum d)) ∧ (Zle (Fnum d) (vNumSup b))) ∧
     (Zle (Zopp (dExp b)) (Fexp d)).

Theorem isBoundedI:
 (b : FboundI) (p : float) {(FboundedI b p)}+{¬ (FboundedI b p)}.

Theorem FboundedFzero: (b : FboundI) (FboundedI b (Fzero (Zopp (dExp b)))).

Definition FnormalI :=
    [b : FboundI]
    [p : float]
     (FboundedI b p) ∧
     ((Rlt (vNumSup b) (Rmult radix (Fnum p))) ∨
      (Rlt (Rmult radix (Fnum p)) (Zopp (vNumInf b)))).

Theorem FnormalINotZero:
 (b : FboundI) (p : float) (FnormalI b p) → ¬ (is_Fzero p).

Theorem FnormalIUnique_aux:
 (b : FboundI)
 (p, q : float)
 (FnormalI b p) →
 (FnormalI b q) → <R> p == q → (Zlt (Fexp p) (Fexp q)) → <float> p = q.

Theorem FnormalIUnique:
 (b : FboundI)
 (p, q : float) (FnormalI b p) → (FnormalI b q) → <R> p == q → <float> p = q.

Definition FsubnormalI :=
    [b : FboundI]
```

```
  [p : float]
   (FboundedI b p) ∧
   ((Fexp p) = (Zopp (dExp b)) ∧
    ((Rle (Zopp (vNumInf b)) (Rmult radix (Fnum p))) ∧
     (Rle (Rmult radix (Fnum p)) (vNumSup b)))).

Theorem FsubnormalIUnique:
 (b : FboundI)
 (p, q : float)
 (FsubnormalI b p) → (FsubnormalI b q) → <R> p == q → <float> p = q.

Theorem NormalIandSubnormalINotEq:
 (b : FboundI)
 (p, q : float) (FnormalI b p) → (FsubnormalI b q) → ¬ <R> p == q.

Definition FcanonicI :=
  [b : FboundI] [a : float] (FnormalI b a) ∨ (FsubnormalI b a).

Theorem FcanonicIUnique:
 (b : FboundI)
 (p, q : float)
 (FcanonicI b p) → (FcanonicI b q) → <R> p == q → <float> p = q.

Theorem Zpower_nat_S:
 (n : nat) (Zpower_nat radix (S n)) = (Zmult radix (Zpower_nat radix n)).

Fixpoint FNIAux [v, N, q : nat] : nat :=
 Cases q of
   O ⇒ O
  | (S q') ⇒
      Cases (Zcompare (Zmult (Zpower_nat radix q') v) (Zmult radix N)) of
      INFERIEUR ⇒ q' | EGAL ⇒ q' | _ ⇒ (FNIAux v N q') end
 end.

Definition FNI := [q, N : nat] (pred (FNIAux q N (S (S N)))).

Theorem FNIAuxLess:
 (v, N, q : nat)
 (lt O v) →
 (le v N) → (Zle (Zmult (Zpower_nat radix (FNIAux v N q)) v) (Zmult radix N)).

Theorem FNILess:
 (q, N : nat)
 (lt O q) → (le q N) → (Zle (Zmult (Zpower_nat radix (FNI q N)) q) N).

Theorem FNIAuxMore:
 (v, N, q : nat)
 (lt (FNIAux v N q) (pred q)) →
  (Zlt N (Zmult (Zpower_nat radix (FNIAux v N q)) v)).

Theorem Zlt_Zpower_nat: (n : nat) (Zlt n (Zpower_nat radix n)).

Theorem FNIMore:
 (N, q : nat)
```

```
(lt O q) →
(le q N) → (Zlt N (Zmult radix (Zmult (Zpower_nat radix (FNI q N)) q))).

Definition FnormalizeI :=
  [b : FboundI]
  [p : float]
   Cases (Zcompare ZERO (Fnum p)) of
     EGAL ⇒ (Float ZERO (Zopp (dExp b)))
   | INFERIEUR ⇒
       (Fshift
        radix
        (min
         (FNI (absolu (Fnum p)) (vNumSup b))
         (absolu (Zplus (Fexp p) (dExp b)))) p)
   | SUPERIEUR ⇒
       (Fshift
        radix
        (min
         (FNI (absolu (Fnum p)) (vNumInf b))
         (absolu (Zplus (Fexp p) (dExp b)))) p)
  end.

Theorem FnormalizeICorrect:
 (b : FboundI) (p : float) <R> (FnormalizeI b p) == p.

Theorem FnormalizeBounded:
 (b : FboundI) (p : float) (FboundedI b p) → (FboundedI b (FnormalizeI b p)).

Theorem FcanonicIBoundedI:
 (b : FboundI) (p : float) (FcanonicI b p) → (FboundedI b p).

Theorem FnormalizeIFcanonicI:
 (b : FboundI) (p : float) (FboundedI b p) → (FcanonicI b (FnormalizeI b p)).

Theorem LexicoPosCanI:
 (b : FboundI)
 (x, y : float)
 (FcanonicI b x) →
 (FboundedI b y) → (Rle R0 x) → (Rle x y) → (Zle (Fexp x) (Fexp y)).

Theorem LexicoCanI:
 (b : FboundI)
 (x, y : float)
 (le (absolu (Zminus (vNumInf b) (vNumSup b))) (S O)) →
 (FcanonicI b x) →
 (FboundedI b y) → (Rlt (Rabsolu x) (Rabsolu y)) → (Zle (Fexp x) (Fexp y)).

Theorem ReductRange:
 (b, b' : FboundI)
 (p : nat)
 (vNumInf b) = (vNumInf b') →
 (dExp b) = (dExp b') →
 <Z> (vNumSup b) = (Zplus (vNumSup b') (POS xH)) →
 <Z> (vNumSup b) = (Zmult radix p) →
```

```
 (x : float) (FboundedI b x) → (Ex [y : float] (FboundedI b' y) ∧ <R> x == y).

Theorem ReductRangeInv:
 (b, b' : FboundI)
 (vNumInf b) = (vNumInf b') →
 (dExp b) = (dExp b') →
 <Z> (vNumSup b) = (Zplus (vNumSup b') (POS xH)) →
 ((x : float)
  (FboundedI b x) → (Ex [y : float] (FboundedI b' y) ∧ <R> x == y)) →
  (Ex [p : Z] <Z> (vNumSup b) = (Zmult radix p)).
End FboundedI_Def.
Section FroundI.
Variable radix : Z.
Hypothesis radixMoreThanOne : (Zlt (POS xH) radix).

Local FtoRradix := (FtoR radix).
Coercion FtoRradix : float >→ R.
Variable b : FboundI.
Hypothesis vNumSupGreaterThanOne : (Zle (POS xH) (vNumSup b)).
Hypothesis vNumInfGreaterThanOne : (Zle (POS xH) (vNumInf b)).

Definition isMin :=
    [r : R]
    [min : float]
     (FboundedI b min) ∧
     ((Rle min r) ∧ ((f : float) (FboundedI b f) → (Rle f r) → (Rle f min))).

Definition isMax :=
    [r : R]
    [max : float]
     (FboundedI b max) ∧
     ((Rle r max) ∧ ((f : float) (FboundedI b f) → (Rle r f) → (Rle max f))).

Definition ProjectorP :=
    [P : R → float → Prop]
    (p, q : float) (FboundedI b p) → (P p q) → <R> p == q.

Definition MonotoneP :=
    [P : R → float → Prop]
    (p, q : R) (p', q' : float) (Rlt p q) → (P p p') → (P q q') → (Rle p' q').

Definition TotalP :=
    [P : R → float → Prop] (r : R) (Ex [p : float] (P r p)).

Definition CompatibleP :=
    [P : R → float → Prop]
    (r1, r2 : R)
    (p, q : float)
    (P r1 p) → r1 == r2 → <R> p == q → (FboundedI b q) → (P r2 q).

Definition MinOrMaxP :=
    [P : R → float → Prop]
    (r : R) (p : float) (P r p) → (isMin r p) ∨ (isMax r p).
```

```
Definition RoundedModeP :=
  [P : R → float → Prop]
   (TotalP P) ∧ ((CompatibleP P) ∧ ((MinOrMaxP P) ∧ (MonotoneP P))).

Theorem ProjectMin: (ProjectorP isMin).

Theorem ProjectMax: (ProjectorP isMax).

Theorem RoundedProjector: (P : ?) (RoundedModeP P) → (ProjectorP P).

Theorem RoundedModeProjectorIdem:
 (P : ?) (p : float) (RoundedModeP P) → (FboundedI b p) → (P p p).

Theorem OppositeIUnique_1:
 (x, y, z : float)
 (P : ?)
 (RoundedModeP P) →
 (FboundedI b x) →
 (FboundedI b y) →
 (Rlt (Ropp x) y) →
 (P (Rplus x y) z) → (Rle (powerRZ radix (Zopp (dExp b))) z).

Theorem OppositeIUnique_2:
 (x, y, z : float)
 (P : ?)
 (RoundedModeP P) →
 (FboundedI b x) →
 (FboundedI b y) →
 (Rlt y (Ropp x)) →
 (P (Rplus x y) z) → (Rle (powerRZ radix (Zopp (dExp b))) (Ropp z)).

Theorem OppositeIUnique:
 (x, y, z : float)
 (P : ?)
 (RoundedModeP P) →
 (FboundedI b x) →
 (FboundedI b y) →
 ¬ (Ropp x) == y →
 (P (Rplus x y) z) → (Rle (powerRZ radix (Zopp (dExp b))) (Rabsolu z)).
End FroundI.
Section FpropI.
Variable radix : Z.
Hypothesis radixMoreThanOne : (Zlt (POS xH) radix).

Local FtoRradix := (FtoR radix).
Coercion FtoRradix : float >→ R.
Variable b : FboundI.

Theorem SterbenzIAux1:
 (x, y : float)
 (FboundedI b x) →
 (FboundedI b y) →
 (Rle y x) → (Rle x (Rmult (S (S O)) y)) → (FboundedI b (Fminus radix x y)).
```

Theorem *SterbenzOppI*:
(*x*, *y* : *float*)
((*u* : *float*)
(*FboundedI b u*) → (*Ex* [*v* : *float*] <*R*> *v* == (*Ropp u*) ∧ (*FboundedI b v*))) →
(*FboundedI b x*) →
(*FboundedI b y*) →
(*Rle* (*Rmult* (*Rinv* (*S* (*S O*))) *y*) *x*) →
(*Rle x* (*Rmult* (*S* (*S O*)) *y*)) →
(*Ex* [*z* : *float*] <*R*> *z* == (*Rminus x y*) ∧ (*FboundedI b z*)).

Theorem *FoppBounded*:
(*x* : *float*)
(*p* : *nat*)
<*Z*> (*vNumSup b*) = (*Zminus* (*Zmult radix p*) (*S O*)) →
<*Z*> (*vNumInf b*) = (*Zmult radix p*) →
(*FboundedI b x*) → (*Ex* [*y* : *float*] <*R*> *y* == (*Ropp x*) ∧ (*FboundedI b y*)).

Theorem *FoppBoundedInv_aux*:
(*n* : *Z*)
(*Zle* (*Zplus* (*vNumSup b*) (*S O*)) (*Zopp n*)) →
(*Zle* (*Zopp n*) (*vNumInf b*)) →
((*x* : *float*)
(*FboundedI b x*) → (*Ex* [*y* : *float*] <*R*> *y* == (*Ropp x*) ∧ (*FboundedI b y*))) →
(*Ex* [*p* : *Z*] <*Z*> *n* = (*Zmult radix p*)).

Theorem *FoppBoundedExp*:
(*x* : *float*)
(*p* : *nat*)
<*Z*> (*vNumSup b*) = (*Zminus* (*Zmult radix p*) (*S O*)) →
<*Z*> (*vNumInf b*) = (*Zmult radix p*) →
(*FboundedI b x*) →
(*Ex* [*y* : *float*]
<*R*> *y* == (*Ropp x*) ∧
((*FboundedI b y*) ∧ ((*Fexp y*) = (*Fexp x*) ∨ (*Fexp y*) = (*Zs* (*Fexp x*))))).

Theorem *nat_div_one*: (*n*, *m* : *nat*) (*mult m n*) = (*S O*) → *m* = (*S O*).

Theorem *Z_div_one*:
(*n* : *nat*) (*z* : *Z*) (*Zmult z* (*inject_nat n*)) = (*inject_nat* (*S O*)) → *n* = (*S O*).

Theorem *FoppBoundedInv*:
(*Zlt* (*vNumSup b*) (*vNumInf b*)) →
((*x* : *float*)
(*FboundedI b x*) → (*Ex* [*y* : *float*] <*R*> *y* == (*Ropp x*) ∧ (*FboundedI b y*))) →
(*Ex* [*p* : *Z*] <*Z*> (*vNumInf b*) = (*Zmult radix p*)) ∧
<*Z*> (*vNumInf b*) = (*Zplus* (*vNumSup b*) (*S O*)).

Theorem *SterbenzIAux2A*:
(*x*, *y* : *float*)
(*n* : *nat*)
(*le* (*absolu* (*Zminus* (*vNumInf b*) (*vNumSup b*))) *n*) →
(*FboundedI b x*) →
(*FboundedI b y*) →
(*Zle* (*Fexp y*) (*Fexp x*)) →

```
(Rle y x) →
(Rle x (Rminus (Rmult (S (S O)) y) (Rmult n (powerRZ radix (Fexp y))))) →
(Rle (Fminus radix y x) R0) →
(Zle (Zopp (vNumInf b)) (Zplus (Zopp (vNumSup b)) n)) →
 (FboundedI b (Fminus radix y x)).
```

Theorem *SterbenzIAux2B*:

```
(x, y : float)
(n : nat)
(le (absolu (Zminus (vNumInf b) (vNumSup b))) n) →
(FboundedI b x) →
(FboundedI b y) →
(Zle (Fexp x) (Fexp y)) →
(Rle y R0) →
(Rle y x) →
(Rle x (Rminus (Rmult (S (S O)) y) (Rmult n (powerRZ radix (Fexp x))))) →
(Rle (Fminus radix y x) R0) →
(Zle (Zopp (vNumInf b)) (Zplus (Zopp (vNumSup b)) n)) →
 (FboundedI b (Fminus radix y x)).
```

Theorem *SterbenzIAux2*:

```
(x, y : float)
(n : nat)
(le (absolu (Zminus (vNumInf b) (vNumSup b))) n) →
(FboundedI b x) →
(FcanonicI radix b y) →
(FboundedI b y) →
(Rle y x) →
(Rle
 x
 (Rminus
  (Rmult (S (S O)) y) (Rmult n (powerRZ radix (Zmin (Fexp y) (Fexp x))))) →
 (FboundedI b (Fminus radix y x)).
```

Theorem *SterbenzI*:

```
(x, y : float)
(n : nat)
(le (absolu (Zminus (vNumInf b) (vNumSup b))) n) →
(FboundedI b x) →
(FcanonicI radix b x) →
(FboundedI b y) →
(Rle
 (Rplus
  (Rmult (Rinv (S (S O))) y)
  (Rmult (Rinv (S (S O))) (Rmult n (powerRZ radix (Zmin (Fexp x) (Fexp y))))))
 x) → (Rle x (Rmult (S (S O)) y)) → (FboundedI b (Fminus radix x y)).
```

End *FpropI*.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399