HAL
open science

Formally verified certificate checkers for hardest-to-round computation

Érik Martin-Dorel, Guillaume Hanrot, Micaela Mayero, Laurent Théry

To cite this version:

Érik Martin-Dorel, Guillaume Hanrot, Micaela Mayero, Laurent Théry. Formally verified certificate checkers for hardest-to-round computation. 2013. hal-00919498v1

HAL Id: hal-00919498

https://inria.hal.science/hal-00919498v1

Preprint submitted on 16 Dec 2013 (v1), last revised 19 Dec 2014 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

JAR manuscript No.
(will be inserted by the editor)

Formally verified certificate checkers for hardest-to-round computation

Érik Martin-Dorel · Guillaume Hanrot · Micaela Mayero · Laurent Théry

the date of receipt and acceptance should be inserted later

Abstract In order to derive efficient and robust floating-point implementations of a given function f, it is crucial to compute its hardest-to-round points, i.e. the floating-point numbers x such that $f(x)$ is closest to the midpoint of two consecutive floating-point numbers. Depending on the floating-point format one is aiming at, this can be highly computationally intensive. In this paper, we show how certificates based on Hensel's lemma can be added to an algorithm using lattice basis reduction so that the result of a computation can be formally checked in the Coq proof assistant.

Keywords: formal proofs, certificate checkers, Hensel's lemma, modular arithmetic.

1 Introduction and motivations

Hensel's lemma is a very classic mathematical tool. Given a polynomial P over the integers, it starts from a solution to $P(x) \equiv 0 \pmod{p}$ and constructs a solution to

Érik Martin-Dorel
Inria Saclay - Île-de-France, LRI
Bâtiment 650, Université Paris 11
91405 Orsay Cedex, France
Phone: +33 1 74 85 42 68
E-mail: erik.martin-dorel@ens-lyon.org

Guillaume Hanrot
École Normale Supérieure de Lyon
LIP (UMR 5668 CNRS, ENSL, Inria, UCBL)
46 allée d'Italie, 69364 Lyon Cedex 07, France
E-mail: guillaume.hanrot@ens-lyon.fr

Micaela Mayero
Université Paris 13, LIPN (UMR 7030 CNRS)
99 avenue JB Clément, 93430 Villetaneuse, France
E-mail: micaela.mayero@lipn.univ-paris13.fr

Laurent Théry
Inria Sophia Antipolis
2004 route des Lucioles, BP 93
06902 Sophia Antipolis Cedex, France
E-mail: laurent.thery@inria.fr

$P(x) \equiv 0 \pmod{p^k}$ for any $k \in \mathbb{N}^*$, under mild hypotheses (for instance, that p does not divide the discriminant of P). This lemma was given its final shape by Hensel, in his study of p-adic numbers, where it plays a central role [19]. The algorithm derived from the proof of Hensel's lemma is usually called Hensel lifting.

The strengths of Hensel's lemma are its versatility (it can be stated in a very general context, being also valid, e.g., for power series rings), the fact that it combines an assertion of existence (there is a lifting) and uniqueness (this lifting is unique) under mild hypotheses, and the fact that it is effective, and provides a simple and efficient algorithm. Hensel's lemma has applications in several areas of mathematics and computer science where one has to solve equations over the integers (or over the ring of polynomials over a field, as in the last step of Guruswami-Sudan list decoding algorithm for Reed-Solomon codes [17,1]). In this work, we are interested in some of them that we present just below.

1.1 Hensel's lemma in computer algebra

Since the beginning of the development of computer algebra, the algorithmic version of Hensel's lemma (Hensel lifting) has been one of the fundamental tools for computations over the integers. The strategy is to isolate a "nice" prime p where the problem under study over the integers has a "good reduction" to the same problem modulo p^k for all $k \in \mathbb{N}^*$. One then solves the problem modulo p, lifts the solution modulo p^m for some m such that p^m is slightly larger than an *a priori* bound on the size of the solution.

Standard applications include linear system solving over the integers (Hensel's technique is very powerful in practice as it avoids manipulating huge rational numbers while keeping the size of integers involved under control), but also 0-dimensional polynomial systems solving over the integers (though the poor quality of the bounds for real solutions as the number of variables grows might limit the applicability in that case) or polynomial factoring over the integers. In all those cases, Hensel's lemma can be used either to compute the solutions or to provide a certificate for a list of solutions (using the uniqueness statement of the lemma).

In the present paper, we are mainly interested in the second aspect, in a setting where Hensel lifting is used at the end of a complex algorithmic chain where formal verification is a major issue; we now turn to the description of the problem for which this algorithmic chain was designed: the *Table Maker's Dilemma*, a problem which needs to be solved for implementing elementary functions with correct rounding.

1.2 The Table Maker's Dilemma

The newly revised IEEE 754–2008 standard for *floating-point* (FP) arithmetic recommends that some mathematical functions (exp, log, $x \mapsto 2^x$, ...) should be correctly rounded: the system must always return the FP number that is the nearest to the exact mathematical result of the operation. Having correctly rounded functions greatly improves the portability of numerical software.

If we consider a correctly-rounded elementary function f in radix 2 and precision n, most implementations use the so-called Ziv's strategy:

1. compute a small enclosing interval of width $2^{-n'}$ for $f(x)$;

2. if both endpoints of this interval round at precision n to the same FP number v, this number v is the correctly rounded value of $f(x)$;
3. otherwise, increase n', and go to step 1.

The correctness of this strategy follows from the fact that rounding is monotonic—if both endpoints of the interval round to the same value, any point of the interval also does. Note that in order to avoid an infinite loop, one should be able to deal separately with "exact cases", i.e. the FP numbers x such that (for rounding-to-nearest) $f(x)$ is the exact midpoint of two consecutive FP numbers, or (for directed rounding modes) $f(x)$ is a FP number, like $x = 0$ for the exp function.

In order to make this strategy optimal, we want to minimize the number of iterations through step 3, and set the n' to a large enough precision to guarantee that the test at step 2 is true, and yet not too large, since the cost of step 1 increases significantly with n'. Finding the optimal value (say n'_0) of this "large enough" precision is the so-called *Table Maker's Dilemma* (TMD) [27, chap. 12]. Finding an upper bound for n'_0 will be called the *Approximate TMD*. Solving the TMD, for a given function f and a rounding mode (say, rounding-to-nearest) reduces to finding the *hardest-to-round* (HR) points, that is the FP numbers x such that $f(x)$ is closest to the midpoint of two consecutive FP numbers.

In this paper, we are interested in one of the algorithms that solve this problem: the Stehlé–Lefèvre–Zimmermann (SLZ) algorithm [32, 30]. This algorithm requires complex and very long calculations (years of cumulated CPU time), and its only output as of today is a list of *hard-to-round points* (each of which gives a *lower bound* on n'_0) and a claim that the worst one is in that list. This is even worse when the SLZ algorithm is applied to the Approximate TMD, as the output is limited to a yes/no answer. This situation thus appears to be unsatisfactory: the initial motivation for the TMD problem is to provide strong guarantees for numerical computations, but the trusted computing base contains highly optimized implementations of complicated algorithms. One of the main contributions of this paper is to propose "logs" for the execution of the SLZ algorithm, allowing one to use aggressively optimized implementations of computer algebra primitives while making it easy to verify the results.

The critical step of the SLZ algorithm consists in generating two polynomials $v_1, v_2 (\in \mathbb{Z}[X, Y])$ as $\mathbb{Z}$-linear combinations of a specific polynomial family. These polynomials are hard to find—they are produced by an aggressively optimized version of the complicated LLL [23] algorithm, which, for standard parameter values is by far the most time-consuming part of the SLZ algorithm. It is then possible to treat the LLL calls as oracles and log their results in a certificate. To be more precise, the SLZ algorithm consists in the following main steps. First, it performs a domain-splitting step and calls a polynomial approximation algorithm. In doing so, the initial (Approximate)-TMD problem is rewritten into myriads of problems that involve integers only. All problems are then instances of the Integer Small Value Problem [32, Section 3.2]:

Problem 1 (ISValP) For $P \in \mathbb{Z}[X]$ and $(A, B, M) \in \mathbb{N}^3$, find *all* solutions of the system

$$\begin{cases} (x, y) \in \mathbb{Z}^2, \\ |x| \leqslant A, \\ |y| \leqslant B, \\ P(x) \equiv y \pmod{M}. \end{cases}$$

Next, each instance of ISValP is solved by successively using the modular, bivariate version of Coppersmith's technique, and the bivariate version of Hensel lifting. This is where the concept of ISValP certificate comes into play, and the core motivation for our work.

1.3 Outline of the paper

In Section 2, we first present the bivariate version of Hensel lifting and then focus on the formalization of the uniqueness statement of bivariate Hensel's lemma in the Coq formal proof assistant. In Section 3, we give the formalization of two interrelated certificate checkers, to address instances of the bivariate small integral roots problem as well as the integer small value problem (ISValP). Next, in Section 4, we present some extra formal material that is used to increase the efficiency and the modularity of these certificate checkers. We then evaluate the performances of our ISValP certificate checker on several examples. In Section 5, we discuss some salient technicalities related to our formalization. Finally we draw some conclusions in Section 6.

2 Formal study of bivariate Hensel's lemma

2.1 Preliminary definitions and notations

For any $x \in \mathbb{R}$, the largest integer $\leqslant x$ is denoted by $\lfloor x \rfloor$ and the smallest integer $\geqslant x$ by $\lceil x \rceil$, so that we have $\lceil x \rceil = -\lfloor -x \rfloor$. For any $(a, b) \in \mathbb{Z}^2$ (assuming $a \leqslant b$), the set of integers k satisfying $a \leqslant k \leqslant b$ is denoted by $[\![a, b]\!]$. Then for any $x \in \mathbb{R}$ and $q \in \,]0, +\infty[$, the "standard modulo" of x by q (also called the remainder of x modulo q) taken in $[0, q[$ is denoted by

$$x \bmod q \;:=\; x - q \left\lfloor \frac{x}{q} \right\rfloor \;\; (\geqslant 0). \tag{1}$$

As a result, we straightforwardly get $x \bmod q \in [\![0, q-1]\!]$ for any $x \in \mathbb{Z}$ and $q \in \mathbb{N}^*$. Finally for any $(a, b) \in \mathbb{R}^2$ and $q \in \,]0, +\infty[$, the usual abbreviation for modular equality is used:

$$a \equiv b \pmod{q} \iff a \bmod q = b \bmod q.$$

Definition 1 (Small integral root) For any pair $(v_1, v_2) \in \mathbb{Z}[X, Y]^2$ of bivariate polynomials over $\mathbb{Z}$ and any pair $(A, B) \in \mathbb{N}^2$, a *small integral root* of (v_1, v_2) (with respect to bounds A and B) is a pair $(x, y) \in \mathbb{Z}^2$ that is solution of the system

$$\begin{cases} |x| \leqslant A, \\ |y| \leqslant B, \\ v_1(x, y) = 0, \\ v_2(x, y) = 0. \end{cases}$$

Then the problem of finding *all* the solutions of this system is referred to as the bivariate small-integral-roots problem (SIntRootP).

Definition 2 (Modular root) For any pair $(v_1, v_2) \in \mathbb{Z}[X, Y]^2$ and any $q \in \mathbb{N}^*$, a *modulo-q root* of (v_1, v_2) is a pair $(x, y) \in \mathbb{Z}^2$ satisfying

$$\begin{cases} v_1(x, y) \equiv 0 \,(\text{mod } q), \\ v_2(x, y) \equiv 0 \,(\text{mod } q). \end{cases}$$

Note that if (x, y) is a modulo-q root of $(v_1, v_2) \in \mathbb{Z}[X, Y]^2$, then any $(z, t) \in (x + q\mathbb{Z}) \times (y + q\mathbb{Z})$ is also a modulo-q root of (v_1, v_2).

Definition 3 (Modular inverse of an integer) For any $x \in \mathbb{Z}$ coprime with $q \in \mathbb{N}^*$, the *modulo-q inverse* of x (taken in $[\![0, q-1]\!]$) is the integer $x_q^{-1} \in [\![0, q-1]\!]$ satisfying $\exists y \in \mathbb{Z},\ x x_q^{-1} + yq = 1$. This quantity can be computed by using the extended Euclidean algorithm.

Definition 4 (Modular inverse of a matrix) For any $M \in \mathcal{M}_n(\mathbb{Z})$ such that $\det M$ is coprime with $q \in \mathbb{N}^*$, the *modulo-q inverse* of M can be defined as $M_q^{-1} := (\det M)_q^{-1} \cdot (\operatorname{adj} M) \bmod q$, where the modulo operation is applied coordinatewise and where $\operatorname{adj} M$ denotes the adjugate matrix of M, that is, the transpose of the cofactor matrix of M, cf. [33]. In particular, for order-2 matrices over $\mathbb{Z}$, we get:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}_q^{-1} = (ad - bc)_q^{-1} \cdot \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \bmod q, \tag{2}$$

assuming $ad - bc$ is coprime with $q \in \mathbb{N}^*$. Note that if $q = p^{2^k}$ for a given prime p, this coprimality condition is equivalent to $ad - bc \not\equiv 0 \,(\text{mod } p)$.

A slightly unusual notion in modular arithmetic is that of centered modulo:

Definition 5 (Centered modulo) For any $x \in \mathbb{R}$ and $q \in \,]0, +\infty[$, the *centered modulo* of x by q, denoted by $x \operatorname{cmod} q$, satisfies

$$|x \operatorname{cmod} q| \leqslant \frac{q}{2} \quad \text{and} \quad \exists n \in \mathbb{Z}, \quad x \operatorname{cmod} q = x + nq. \tag{3}$$

To ensure the uniqueness of this definition, we can consider the centered modulo that has values in $\left]-\frac{q}{2}, \frac{q}{2}\right]$, which amounts to considering the following definition:

$$x \operatorname{cmod} q := x - q \left\lceil \frac{x}{q} - \frac{1}{2} \right\rceil. \tag{4}$$

Note that if x and q are integers, so is $x \operatorname{cmod} q$. As a matter of fact, in the following, we will only deal with centered moduli over the integers. Also, the centered modulo can be emulated with the standard one:

$$x \operatorname{cmod} q = \begin{cases} x \bmod q & \text{if } x \bmod q \leqslant \frac{q}{2} \\ x \bmod q \ - \ q & \text{otherwise.} \end{cases} \tag{5}$$

Furthermore, any centered modulo function trivially enjoys the following equivalence:

$$\forall (z, q, n) \in \mathbb{Z}^3, \quad |z \operatorname{cmod} q| \leqslant n \iff \exists x \in \mathbb{Z}, \quad |x| \leqslant \min\left(\frac{q}{2}, n\right) \ \wedge \ z \equiv x \pmod q, \tag{6}$$

which can be useful to "unfold" the centered modulo when it occurs in such an inequality. Note also that in practice we will always have $n < q/2$, allowing one to further simplify the right-hand side of the equivalence (6).

2.2 Finding bivariate small-integral-roots using Hensel lifting

Before focusing on *the uniqueness statement of bivariate Hensel's lemma* and in order to give more intuition, we start with the algorithmic version of this lemma, called bivariate Hensel lifting. The main step of this iterative algorithm is summarized in Algorithm 1, along with the conditions and post-conditions that actually relate to the *existence* statement of bivariate Hensel's lemma.

Algorithm 1: Bivariate Hensel lifting: sketch of the computation to be iterated

Input : $(v_1, v_2) \in \mathbb{Z}[X, Y]^2$, p prime, $k \in \mathbb{N}$, $q = p^{2^k}$,
$(a, b) \in \mathbb{Z}^2$ such that $\det [J_{v_1,v_2}(a, b)] \not\equiv 0 \pmod p$
and $v_1(a, b) \equiv 0 \equiv v_2(a, b) \pmod q$.

Output : $(a', b') \in \mathbb{Z}^2$ such that $a' \equiv a \pmod q$, $b' \equiv b \pmod q$,
and $v_1(a', b') \equiv 0 \equiv v_2(a', b') \pmod{q^2}$.

$$\begin{pmatrix} a' \\ b' \end{pmatrix} \leftarrow \begin{pmatrix} a \\ b \end{pmatrix} - \left[J_{v_1,v_2}(a, b) \right]_{q^2}^{-1} \begin{pmatrix} v_1(a, b) \\ v_2(a, b) \end{pmatrix} \operatorname{cmod} q^2$$

We can see that Algorithm 1 deals with the modular roots of a pair (v_1, v_2) of bivariate polynomials over $\mathbb{Z}$, and involves the Jacobian matrix[1] of (v_1, v_2) evaluated at $(a, b) \in \mathbb{Z}^2$. Since p is prime, the condition $\det [J_{v_1,v_2}(a, b)] \not\equiv 0 \pmod p$ implies that it is legal to consider the modular inverse $\left[J_{v_1,v_2}(a, b) \right]_{q^2}^{-1}$ (for $q^2 = p^{2^{k+1}}$). Note that this algorithm is very similar to the so-called Newton's method, so its univariate counterpart is called *p-adic Newton iteration* in the book Modern Computer Algebra [34, Algorithm 9.22, p. 264]. Note also that the univariate algorithm presented in this reference stores the modulo-(p^{2^k}) inverses in an extra variable, which is updated at each iteration using Hensel's iteration for the inverse "on the fly": this optimization could be generalized to the bivariate case too.

To find the bivariate small-integral-roots of (v_1, v_2), we just need to:

(*i*) Find a "nice" prime p so that Algorithm 1 is applicable;
(*ii*) Iterate Algorithm 1 on each modulo-p root of (v_1, v_2);
(*iii*) Stop as soon as one reaches an *a-priori* bound on the solutions.

For item (*i*), we can choose any prime p such that the following condition holds:

$$\forall (x, y) \in \mathbb{Z}^2, \quad v_1(x, y) \equiv 0 \equiv v_2(x, y) \pmod p \implies \det J_{v_1,v_2}(x, y) \not\equiv 0 \pmod p. \quad (7)$$

Note that (7) can be decided by checking p values for x and p values for y, that is, p^2 different pairs $(x, y) \in [\![0, p-1]\!]^2$.

[1] Recall that this matrix is the generalization of the derivative in higher-dimensional calculus and is defined in the bivariate case by:

$$J_{v_1,v_2}(a, b) = \begin{pmatrix} \frac{\partial v_1}{\partial X}(a, b) & \frac{\partial v_1}{\partial Y}(a, b) \\ \frac{\partial v_2}{\partial X}(a, b) & \frac{\partial v_2}{\partial Y}(a, b) \end{pmatrix}.$$

Algorithm 2 describes how to process item (*ii*) and item (*iii*), in order to solve instances of bivariate SIntRootP (according to the terminology of Definition 1).

Algorithm 2: Find the small integral roots of two bivariate polynomials

Input: $(v_1, v_2) \in \mathbb{Z}[X, Y]^2$, $(A, B) \in \mathbb{N}^2$.
Output: The small integral roots of (v_1, v_2) with respect to bounds A and B.

```
p ← a prime that satisfies (7)
S ← ∅
foreach (r, s) ∈ [[0, p − 1]]^2 such that v1(r, s) ≡ 0 ≡ v2(r, s) (mod p) do
    k ← 0
    K ← p
    a ← r cmod K
    b ← s cmod K
    // Here, K = p^(2^k), a ≡ r (mod p), b ≡ s (mod p), |a| ⩽ K/2, |b| ⩽ K/2
    while K ⩽ 2 · max(A, B) do
        k ← k + 1
        K ← K^2
        (a b)^T ← (a b)^T − [J_{v1,v2}(a, b)]^{−1}_K (v1(a, b) v2(a, b))^T cmod K
    end
    // Here, K = p^(2^k), a ≡ r (mod p), b ≡ s (mod p), |a| ⩽ K/2, |b| ⩽ K/2
    if v1(a, b) = 0 = v2(a, b) and |a| ⩽ A and |b| ⩽ B then S ← S ∪ {(a, b)}
end
return S
```

We now give a concrete example to see how Algorithm 2 works. This example comes from an actual execution of the SLZ algorithm and will thus be reused later.

Example 1 Let us consider the following bivariate polynomials:

$$\begin{cases} v_1(X,Y) = Q(X,Y)^2 \\ \qquad\qquad + (-6Y - 1593703)MQ(X,Y) \\ \qquad\qquad + (9Y^2 + 4781109Y + 634972313052)M^2, \\ v_2(X,Y) = -25MQ(X,Y) + (76Y + 19921289)M^2, \end{cases}$$

where $M = 2^{92}$ and

$$\begin{aligned} Q(X,Y) = -X + (&36506148256923413Y^3 + 28709603185988793532416Y^2 \\ &+ 15052100435175692583523319808Y \\ &+ 39458178164786967566151371477483 52). \end{aligned}$$

Suppose we want to determine all the integral roots (x, y) of (v_1, v_2) that satisfy

$$|x| \leqslant A := 2^{87} \quad \text{and} \quad |y| \leqslant B := 2^4.$$

We first compute the Jacobian matrix of (v_1, v_2),

$$J_{v_1,v_2} = \begin{pmatrix} \frac{\partial v_1}{\partial X} & \frac{\partial v_1}{\partial Y} \\ \frac{\partial v_2}{\partial X} & \frac{\partial v_2}{\partial Y} \end{pmatrix},$$

whose determinant cancels for some[2] modulo-2 roots of (v_1, v_2). In contrast, we can check that modulo 3, no modular root of (v_1, v_2) is a modular root of $\det J_{v_1,v_2}$:

$$\begin{cases} v_1(x,y) \equiv 0 \,(\mathrm{mod}\ 3), \\ \qquad v_2(x,y) \equiv 0 \,(\mathrm{mod}\ 3), \text{ and} \\ \qquad \det J_{v_1,v_2}(x,y) \not\equiv 0 \,(\mathrm{mod}\ 3) \quad \text{if } (x \bmod 3, y \bmod 3) \in \{(0,2),(2,1)\}, \\ v_1(x,y) \not\equiv 0 \,(\mathrm{mod}\ 3) \\ \qquad \text{or} \quad v_2(x,y) \not\equiv 0 \,(\mathrm{mod}\ 3) \qquad \text{otherwise.} \end{cases}$$

This means that $p := 3$ is the first prime satisfying the key hypothesis (7). Following Algorithm 2, we can iterate bivariate Hensel lifting on both modulo-3 roots of (v_1, v_2): since $3^{2^5} \leqslant 2 \times \max(A, B) < 3^{2^6}$, $k := 6$ iterations are performed and lead to the modular roots summarized in Table 1.

Iterations	Orbit of $(0,2)$	Orbit of $(2,1)$
0	$(0,-1)$	$(-1,1)$
1	$(0,2)$	$(-4,4)$
2	$(18,11)$	$(23,-14)$
3	$(2286,11)$	$(-1516,-14)$
4	$(-11125170,11)$	$(-4823851,-14)$
5	$(281986488503298,11)$	$(785654438631512,-14)$
6	$(2336732229695754577823 19,11)$	$(336478080626535692605744 40,-14)$

Table 1 Modular roots obtained by iterating Hensel lifting (with a centered modulo), for initial values $(0,2)$ and $(2,1)$ which are modulo-3 roots of (v_1, v_2).

Focusing on the values computed at the 6th iteration, we can notice that both $(x, y) = (2336732229695754577823 19, 11)$ and $(336478080626535692605744 40, -14)$ satisfy

$$v_1(x,y) = 0 = v_2(x,y), \quad |x| \leqslant A \ \text{ and } \ |y| \leqslant B,$$

which means we have found two small integral roots of (v_1, v_2). Thanks to the material presented in this paper, it will also be possible to provide a certificate that guarantees that there is no other small integral root of (v_1, v_2), that is, we have found all the solutions of the problem at stake (in particular, see Section 3 for the sequel of this example).

2.3 Bivariate Hensel's lemma: pencil and paper proof

We start this section by introducing two elementary arithmetic results that are used several times in what follows.

Lemma 1 *For all $d \in \mathbb{N}$ and $z \in \mathbb{Z}$, if $|z| < d$ and $z \equiv 0 \pmod{d}$, then $z = 0$.*

Corollary 1 *For all $(d, m, n) \in \mathbb{Z}^3$, if $|2 \cdot m| \leqslant d$, $|2 \cdot n| < d$, and $m \equiv n \pmod{d}$, then we have $m = n$.*

[2] Namely, $(0,0)$ and $(1,1)$ are modulo-2 roots of both (v_1, v_2) and $\det J_{v_1,v_2}$.

Proof (Corollary 1) Suppose $d, m, n \in \mathbb{Z}$ satisfy the hypotheses of Corollary 1. By the triangle inequality, we have $|2 \cdot (m-n)| < 2 \cdot d$, that is $|m-n| < d$. Since we also have $m - n \equiv 0 \pmod{d}$, Lemma 1 implies that $m - n = 0$, that is, $m = n$. □

We can now present the uniqueness statement of bivariate Hensel's lemma:

Lemma 2 (Hensel) *Let v_1, v_2 be two bivariate polynomials with integer coefficients, and let p be a prime that satisfies (7). For all $(x, y) \in \mathbb{Z}^2$ and $k \in \mathbb{N}$ such that*

$$v_1(x, y) \equiv 0 \equiv v_2(x, y) \pmod{p^{2^k}}, \tag{8}$$

any $(\mathbb{Z}^2)$-valued sequence $(a_n, b_n)_{0 \leqslant n \leqslant k}$ defined by the relations

$$(a_0, b_0) \equiv (x, y) \pmod{p} \tag{9}$$

and

$$\forall n \in [\![0, k-1]\!], \ \begin{pmatrix} a_{n+1} \\ b_{n+1} \end{pmatrix} \equiv \begin{pmatrix} a_n \\ b_n \end{pmatrix} - \left[J_{v_1, v_2}(a_n, b_n) \right]^{-1}_{p^{2^{n+1}}} \begin{pmatrix} v_1(a_n, b_n) \\ v_2(a_n, b_n) \end{pmatrix} \pmod{p^{2^{n+1}}} \tag{10}$$

satisfies:

$$\forall n \in [\![0, k]\!], \quad (a_n, b_n) \equiv (x, y) \pmod{p^{2^n}}. \tag{11}$$

The following corollary shows that this lemma constitutes a uniqueness result for bivariate Hensel lifting:

Corollary 2 (Uniqueness of Hensel lifting) *Assume $(v_1, v_2) \in \mathbb{Z}[X, Y]^2$ satisfies (7) for a given prime p. For any $k \in \mathbb{N}$, if $(a, b) \in \mathbb{Z}^2$ and $(c, d) \in \mathbb{Z}^2$ are modulo-(p^{2^k}) roots of (v_1, v_2), that is if*

$$\begin{cases} v_1(a, b) \equiv 0 \equiv v_2(a, b) \pmod{p^{2^k}} \\ v_1(c, d) \equiv 0 \equiv v_2(c, d) \pmod{p^{2^k}}, \end{cases}$$

then the lifting of the corresponding modulo-p roots is unique, that is

$$\begin{cases} a \equiv c \pmod{p} \\ b \equiv d \pmod{p} \end{cases} \implies \begin{cases} a \equiv c \pmod{p^{2^k}} \\ b \equiv d \pmod{p^{2^k}}. \end{cases}$$

Proof (Corollary 2) First, apply Lemma 2 with $(x, y) := (a, b)$ and fix a sequence $(a_n, b_n)_{0 \leqslant n \leqslant k}$ satisfying (9) and (10), e.g., using the standard modulo. Then take $n := k$ in (11), so that $(a_k, b_k) \equiv (a, b) \pmod{p^{2^k}}$. Second, apply Lemma 2 with $(x, y) := (c, d)$, using the same sequence $(a_n, b_n)_{0 \leqslant n \leqslant k}$ which indeed satisfies $(a_0, b_0) \equiv (a, b) \equiv (c, d) \pmod{p}$. Then take $n := k$ again, so that $(a_k, b_k) \equiv (c, d) \pmod{p^{2^k}}$, hence the result. □

Before giving a complete mathematical proof of Lemma 2, let us mention that we give in Appendix A the mathematical proof of two propositions (including a correctness proof of Algorithm 2) that we deduce from this lemma. Although these propositions are not directly related to the main topic of the paper, they can be useful to illustrate the usefulness of Lemma 2 beyond the formal verification of certificates checkers based on Hensel's lemma.

Proof (Lemma 2) Let us assume that all the hypotheses of the lemma hold for $(v_1, v_2) \in \mathbb{Z}[X,Y]^2$, p prime, $(x,y) \in \mathbb{Z}^2$, and $(a_n, b_n)_{0 \leqslant n \leqslant k}$. To shorten the formulas involved in the sequel, we denote by $(x_n, y_n)_{n \in \mathbb{N}}$ the sequence defined for all $n \in \mathbb{N}$ by $(x_n, y_n) := (x \bmod p^{2^n}, y \bmod p^{2^n})$. Proving (11) thus amounts to showing that $\forall n \in \mathbb{N},\ n \leqslant k \Rightarrow (a_n, b_n) \equiv (x_n, y_n) \pmod{p^{2^n}}$. We prove it by induction on n:

- First, we have $p = p^{2^0}$, so that (9) implies $(a_0, b_0) \equiv (x_0, y_0) \pmod{p^{2^0}}$, which proves the base case.
- For the inductive case, let us show that for any integer $n < k$ that satisfies $(a_n, b_n) \equiv (x_n, y_n) \pmod{p^{2^n}}$, we have $(a_{n+1}, b_{n+1}) \equiv (x_{n+1}, y_{n+1}) \pmod{p^{2^{n+1}}}$. We can write $x_n = x \bmod p^{2^n} = [x \bmod p^{2^{n+1}}] \bmod p^{2^n} = x_{n+1} \bmod p^{2^n}$, so that $x_{n+1} \equiv x_n \pmod{p^{2^n}}$, hence $x_{n+1} \equiv a_n \pmod{p^{2^n}}$ by using the induction hypothesis. This implies that

$$\exists \lambda \in \mathbb{Z}, \quad x_{n+1} = a_n + \lambda p^{2^n}.$$

Likewise, we deduce that

$$\exists \mu \in \mathbb{Z}, \quad y_{n+1} = b_n + \mu p^{2^n}.$$

For any bivariate polynomial P, we consider the operator

$$\Delta_{i,j}(P) := \frac{1}{i!j!}\left(\frac{\partial^{i+j}}{\partial X^i \partial Y^j} P\right) \tag{12}$$

which maps $\mathbb{Z}[X,Y]$ to $\mathbb{Z}[X,Y]$ (this is easily checked on monomials $X^k Y^\ell$). Then we apply Taylor's theorem to each bivariate polynomial v_l $(l = 1, 2)$:

$$v_l(x_{n+1}, y_{n+1}) = \sum_{i,j \in \mathbb{N}} \left(\lambda p^{2^n}\right)^i \left(\mu p^{2^n}\right)^j \Delta_{i,j}(v_l)(a_n, b_n). \tag{13}$$

The left-hand side of (13) is zero modulo $p^{2^{n+1}}$, since

$$v_l(x_{n+1}, y_{n+1}) = v_l(x \bmod p^{2^{n+1}}, y \bmod p^{2^{n+1}}) \equiv v_l(x,y) = 0 \pmod{p^{2^{n+1}}}.$$

Concerning the right-hand side, we can notice that

$$\forall i, j \in \mathbb{N}, \quad i + j \geqslant 2 \implies \left(\lambda p^{2^n}\right)^i \left(\mu p^{2^n}\right)^j = \left[\lambda^i \mu^j \left(p^{2^n}\right)^{i+j-2}\right] \cdot p^{2^{n+1}}$$

and

$$\Delta_{i,j}(v_l)(a_n, b_n) \in \mathbb{Z},$$

therefore all terms in the summation involved in (13) are zero modulo $p^{2^{n+1}}$ whenever $i + j \geqslant 2$. As a result, (13) becomes

$$0 \equiv v_l(a_n, b_n) + \lambda p^{2^n} \frac{\partial}{\partial X} v_l(a_n, b_n) + \mu p^{2^n} \frac{\partial}{\partial Y} v_l(a_n, b_n) \pmod{p^{2^{n+1}}}. \tag{14}$$

We can combine (14) for both values $l = 1, 2$ to obtain

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} v_1(a_n, b_n) \\ v_2(a_n, b_n) \end{pmatrix} + \left[J_{v_1, v_2}(a_n, b_n)\right] \begin{pmatrix} \lambda p^{2^n} \\ \mu p^{2^n} \end{pmatrix} \pmod{p^{2^{n+1}}}, \tag{15}$$

where the modulo is taken coordinatewise.

Then, we can easily combine (8) with the induction hypothesis to deduce that $v_1(a_n, b_n) \equiv 0 \equiv v_2(a_n, b_n) \pmod{p}$, hence by (7), $\det J_{v_1,v_2}(a_n, b_n) \not\equiv 0 \pmod{p}$ and thereby $\det J_{v_1,v_2}(a_n, b_n) \not\equiv 0 \pmod{p^{2^{n+1}}}$, so that (15) can be rewritten to

$$-\left[J_{v_1,v_2}(a_n, b_n)\right]^{-1}_{p^{2^{n+1}}} \begin{pmatrix} v_1(a_n, b_n) \\ v_2(a_n, b_n) \end{pmatrix} \equiv \begin{pmatrix} \lambda p^{2^n} \\ \mu p^{2^n} \end{pmatrix} \pmod{p^{2^{n+1}}}. \tag{16}$$

Then we replace λp^{2^n} with $x_{n+1} - a_n$ (resp. μp^{2^n} with $y_{n+1} - b_n$) and we obtain

$$\begin{pmatrix} a_n \\ b_n \end{pmatrix} - \left[J_{v_1,v_2}(a_n, b_n)\right]^{-1}_{p^{2^{n+1}}} \begin{pmatrix} v_1(a_n, b_n) \\ v_2(a_n, b_n) \end{pmatrix} \equiv \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} \pmod{p^{2^{n+1}}}.$$

Now we can use (10) to finally deduce that

$$\begin{pmatrix} a_{n+1} \\ b_{n+1} \end{pmatrix} \equiv \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} \pmod{p^{2^{n+1}}}. \qquad \square$$

2.4 Bivariate Hensel's lemma: formal proof in Coq

The code of our CoqHensel library, along with some documentation, is available online at `http://tamadi.gforge.inria.fr/CoqHensel/`.

Let us now focus on the formalization of bivariate Hensel's lemma in Coq [13,3]. As regards the basic types, functions and lemmas required for this formalization, we can see from the previous Section 2.3 that we need:

- (*i*) $\mathbb{N}$ with the usual operations (including exponentiation) as well as the primality and divisibility predicates;
- (*ii*) $\mathbb{Z}$ with the usual operations (including absolute value) and a comprehensive formalization of modular arithmetic;
- (*iii*) $\mathbb{Z}[X, Y]$ with polynomial evaluation, derivatives, and Taylor's theorem for bivariate polynomials;
- (*iv*) Some facilities to handle iterated operators, and in particular the summations that appear when proving (and using) Taylor's theorem;
- (*v*) ExtendedGCD-based modular inversion over $\mathcal{M}_2(\mathbb{Z})$.

A large part of these notions are covered by the libraries distributed with the SSReflect [15,16] extension of Coq. First, item (*i*) above is fully handled using libraries `ssrnat`, `prime` and `div`. For item (*ii*), we rely on the SSReflect libraries `ssrint`, `intdiv` and `ssrnum` which define integers as

```
Inductive int := Posz (n : nat) | Negz (n : nat).
```

0 is represented by `(Posz 0)`, 1 by `(Posz 1)`, -1 by `(Negz 0)`, and so on. As this definition involves Peano integers (`nat`), it is definitely not adapted to large-scale computation, but quite convenient to develop proofs and going back-and-forth between signed integers and natural numbers. For computing effectively with integers we will rely on other data structures, which will be the topic of Section 4.1.

For item (*iii*), we simply define bivariate polynomials as polynomials with univariate polynomials as coefficients. The corresponding Coq code is:

```
Notation "{ 'bipoly'  R }" := {poly {poly R}} : type_scope.
```

Note that this definition is asymmetric with respect to the indeterminates X and Y (which are respectively defined as the outermost indeterminate and the innermost one). One of the benefits of this definition is that we directly inherit all ring operations and support theorems from the SSReflect library `poly`. We just need to define the n-th partial derivatives and bivariate Horner evaluation, then prove the following version of Taylor's theorem:

Theorem 1 (Taylor) *For any unital ring R, for any polynomial $P \in R[X,Y]$ and for any $x_0, y_0, h, k \in R$ that commute pairwise*[3] *we have*

$$P(x_0 + h, y_0 + k) = \sum_i \sum_j \Delta_{i,j}(P)(x_0, y_0) \cdot h^i \cdot k^j. \tag{17}$$

The operator $\Delta_{i,j}$ above has been previously defined in Equation (12), and we recall that it maps the ring $R[X,Y]$ to $R[X,Y]$, despite the presence of the division $\frac{1}{i!j!}$. The proof of Theorem 1 essentially relies on some manipulations of summations, and uses twice the univariate version of Taylor's theorem (which has been included in the SSReflect library `poly` at the occasion of this work).

For item (iv), we benefited from the formalization of generic "big operators" provided in the SSReflect library `bigop` [4].

For item (v), we rely on the formalization of rings $\mathbb{Z}/q\mathbb{Z}$ provided in the SSReflect library `zmodp` to address Definitions 3 and 4. Note that for the sake of simplicity, we define our own type for 2-by-2 matrices—a polymorphic record with four projections—; this material will be reused to straightforwardly get a computational version of Jacobian matrices, involved in the effective certificate checkers presented in Section 4. Although not essential to derive the proof of Lemma 2, the use of matrices leads to more concise expressions and proof steps. In particular, a key ingredient about 2-by-2 matrices is given by the following version of Cramer rule with moduli:

$$\begin{aligned} &\forall p \text{ prime}, \quad \forall k \in \mathbb{N}, \quad \forall A \in \mathcal{M}_2(\mathbb{Z}), \quad \forall v \in \mathbb{Z}^2 = \mathcal{M}_{2,1}(\mathbb{Z}), \\ &\qquad \det(A) \not\equiv 0 \pmod{p} \implies A^{-1}_{p^{2^{k+1}}} \times (A \times v) \equiv v \pmod{p^{2^{k+1}}}. \end{aligned} \tag{18}$$

The formal proof of Lemma 2 closely follows the pencil-and-paper proof presented in Section 2.3. In particular, we rely on the result (18) to deduce (16) from (15).

3 Formal verification of certificates using Hensel's lemma

We now follow the certificate-based approach to address the Integer Small Value Problem (ISValP) described in Section 1.2 :

- In Section 3.1, we devise some bivariate small integral roots certificates that can be generated by using a variant of Algorithm 2. These certificates can then be verified by a checker that does not need to iterate Hensel lifting all over again, and whose correctness proof relies on Corollary 2.
- Then in Section 3.2, we devise some ISValP certificates that are built upon the previous certificates, and that can be generated using the modular, bivariate version of Coppersmith's technique, based on the LLL algorithm [23]. Again, the checker does not need any call to the LLL algorithm to verify the certificates.

[3] that is, $x_0 y_0 = y_0 x_0$, $x_0 h = h x_0$, $x_0 k = k x_0$, $y_0 h = h y_0$, $y_0 k = k y_0$ and $hk = kh$.

3.1 Certificates for the bivariate small-integral-roots problem

3.1.1 The problem at stake

The problem we tackle in this section is to formally verify that we know all the small integral roots $(x, y) \in \mathbb{Z} \times \mathbb{Z}$ of a pair of bivariate polynomials (v_1, v_2) on $\mathbb{Z}$, with respect to bounds A and B:

Problem 2 (Certifying bivariate small integral roots) Given

$$\begin{cases} v_1 \in \mathbb{Z}[X, Y], \\ v_2 \in \mathbb{Z}[X, Y], \\ A \in \mathbb{N}, \\ B \in \mathbb{N}, \\ \mathcal{S} \subset \mathbb{Z} \times \mathbb{Z}, \end{cases} \tag{19}$$

we want to formally verify that the following property holds:

$$\forall (x, y) \in \mathbb{Z}^2, \quad (x, y) \in \mathcal{S} \iff v_1(x, y) = 0 = v_2(x, y) \ \wedge \ |x| \leqslant A \ \wedge \ |y| \leqslant B. \tag{20}$$

For this purpose, we follow the following steps:

1. Define the type for bivariate small integral roots certificates as a `Record`;
2. Define the checker for these certificates as a Coq function;
3. Prove the correctness of this checker (if it returns `true`, then (20) holds), using the uniqueness result given by Corollary 2;
4. Optimize the implementation of the checker by using efficient data structures for the computation.

We detail below the main definitions or statements that are involved in the three first steps, while the fourth step will be described in Section 4.

3.1.2 The type of certificates

First, we need to define the type of certificates for the bivariate small-integral-roots problem. Typically, it should gather at the same time the "input/output" of the problem—here given in Equation (19)—, as well as some additional data that allow one to deduce that the particular instance of the problem is valid—here it amounts to showing that Equation (20) holds. We thus consider the following type:

```
Record bivCert := BivCert
{ bc_v1 : {bipoly int}
; bc_v2 : {bipoly int}
; bc_A : nat
; bc_B : nat
; bc_p : nat
; bc_k : nat
; bc_L : list (int * int * bool)
}.
```

In addition to the polynomials v_1, v_2, and the bounds A and B—which constitute the "input" of the problem—, the certificate stores the prime p and the final value of k that both occur in Algorithm 2.

We recall that p is a prime number that fulfills (7), and k is the number of iterations of Hensel lifting performed to reach the specified bounds (i.e., we have $p^{2^k} > 2 \cdot \max(A, B)$). Finally, the list L involved in the certificate is supposed to store all the modulo-(p^{2^k}) roots[4] of (v_1, v_2) modulo, along with a Boolean value that indicates whether each *modular root* is an actual *integral root* or not.

Example 2 (Bivariate small integral roots certificate) A possible certificate built on Example 1 is the following one:

$$\begin{aligned}\mathcal{C} := \Big(& v_1, v_2, \left(A := 2^{87}\right), \left(B := 2^4\right), (p := 3), (k := 6), \\ & \Big(L := \big\{(233673222969575457782319, 11, \mathsf{true}), \\ & \qquad (336478080626535692605744 40, -14, \mathsf{true})\big\}\Big)\Big).\end{aligned}$$

where v_1 and v_2 are not shown here for brevity (see Example 1 for their definition).

3.1.3 The certificate checker

Let us now present the algorithm to be implemented in the certificate checker. We say that a certificate $(v_1, v_2, A, B, p, k, L)$ is valid if the following conditions are fulfilled:

$$p \text{ is a prime number,} \tag{21a}$$

$$\text{if the list } L \text{ is not empty, the integer } k \geqslant 0 \text{ satisfies } p^{2^k} > 2 \cdot \max(A, B), \tag{21b}$$

for $L_p := \{(r \bmod p, s \bmod p) \mid \exists t \in \{\mathsf{true}, \mathsf{false}\}, \ (r, s, t) \in L\}$, we have:

$$\forall (x, y) \in [\![0, p-1]\!]^2, \quad (x, y) \in L_p \iff v_1(x, y) \equiv 0 \equiv v_2(x, y) \pmod{p}, \tag{21c}$$

$$\text{the elements of } L_p \text{ are pairwise distinct,} \tag{21d}$$

and for all $(r, s, t) \in L$, we have:

$$\det J_{v_1, v_2}(r, s) \not\equiv 0 \pmod{p}, \tag{21e}$$

$$v_1(r, s) \equiv 0 \equiv v_2(r, s) \pmod{p^{2^k}}, \tag{21f}$$

$$|2 \cdot r| \leqslant p^{2^k} \quad \text{and} \quad |2 \cdot s| \leqslant p^{2^k}, \tag{21g}$$

$$t = \mathsf{true} \iff \Big(v_1(r, s) = 0 = v_2(r, s) \quad \wedge \quad |r| \leqslant A \quad \wedge \quad |s| \leqslant B\Big). \tag{21h}$$

These conditions are implemented by a Coq function `biv_check` that takes a certificate of type `bivCert` and returns a boolean that indicates whether the certificate is valid or not.

[4] These modular roots being considered in an appropriate range, say in $\left]-\frac{p^{2^k}}{2}, \frac{p^{2^k}}{2}\right]$.

3.1.4 The formal correctness proof

The correctness proof of the bivariate checker `biv_check` consists in proving that any certificate that is accepted by the checker is valid, i.e., contains all the small integral roots of the considered pair of bivariate polynomials. We thus prove the following

Theorem 2 (Correctness of `biv_check`) *For any bivariate small-integral-roots certificate* $\mathcal{C} := (v_1, v_2, A, B, p, k, L)$*, if* (`biv_check` $\mathcal{C}$) *returns* `true`*, then for all* $(x, y) \in \mathbb{Z}^2$ *we have the equivalence*

$$\Big(v_1(x,y) = 0 = v_2(x,y) \quad \wedge \quad |x| \leqslant A \quad \wedge \quad |y| \leqslant B\Big) \iff (x, y, \mathsf{true}) \in L. \tag{22}$$

Remark 1 (Invertibility hypothesis) Equation (21e) can be viewed as a "distributed version" of the main invertibility hypothesis (7) of bivariate Hensel's lemma. Indeed, according to (21c), the values that are given to variable (r, s) in (21e) correspond to all the modulo-p roots of (v_1, v_2), so that (21c) and (21e) imply

$$\forall (x,y) \in [\![0, p-1]\!]^2\,, \\ v_1(x,y) \equiv 0 \equiv v_2(x,y) \pmod{p} \implies \det J_{v_1,v_2}(x,y) \not\equiv 0 \pmod{p}.$$

Let us now present the main steps of this correctness proof, which has been mechanized in Coq.

Proof (Theorem 2) Suppose that (`biv_check` $\mathcal{C}$) returns `true` for a given certificate $\mathcal{C} := (v_1, v_2, A, B, p, k, L)$, and let $(x, y) \in \mathbb{Z}^2$. Note that modular equalities over $\mathbb{Z}^2$ below are taken coordinatewise.

- Let us prove the "$\Longrightarrow$" part of equivalence (22), relying on Corollary 2. To begin with, the assumption $v_1(x, y) = 0 = v_2(x, y)$ implies

$$v_1(x \bmod p, y \bmod p) \equiv 0 \equiv v_2(x \bmod p, y \bmod p) \pmod{p}. \tag{23}$$

  So applying (21c) with $(x_0, y_0) := (x \bmod p, y \bmod p)$ leads to $(x_0, y_0) \in L_p$, that is by definition of L_p,

$$\exists (r, s, t) \in L, \quad (x_0, y_0) = (r \bmod p, s \bmod p),$$

  hence

$$(x, y) \equiv (r, s) \pmod{p}. \tag{24}$$

  Now let us show that $(r, s, t) = (x, y, \mathsf{true})$, which will allow one to deduce that $(x, y, \mathsf{true}) \in L$.
  - First, we want to show that $(x, y) = (r, s)$. We can use Corollary 2: the main hypothesis (7) is fulfilled thanks to (21e), as explained in Remark 1. Moreover, we immediately have

$$v_1(x, y) \equiv 0 \equiv v_2(x, y) \pmod{p^{2^k}}$$

    as well as

$$v_2(r, s) \equiv 0 \equiv v_2(r, s) \pmod{p^{2^k}}$$

by using (21f). So we can apply Corollary 2 and use (24) to deduce that

$$(x, y) \equiv (r, s) \pmod{p^{2^k}}.$$

In addition, with $K := p^{2^k}$, we have

$$\begin{cases} |x| \leqslant A < K/2 & \text{by (21b)}, \\ |r| \leqslant K/2 & \text{by (21g)}, \\ |y| \leqslant B < K/2 & \text{by (21b)}, \\ |s| \leqslant K/2 & \text{by (21g)}. \end{cases}$$

Hence by applying twice Corollary 1, we get $x = r$ and $y = s$.

– Second, we want to show that $t = \mathsf{true}$, knowing that the equivalence (21h) holds. By hypothesis, we have

$$v_1(x, y) = 0 = v_2(x, y) \quad \wedge \quad |x| \leqslant A \quad \wedge \quad |y| \leqslant B.$$

It then suffices to combine this result with the fact that $(x, y) = (r, s)$ that we have proved just now to get $v_1(r, s) = 0 = v_2(r, s) \ \wedge \ |r| \leqslant A \ \wedge \ |s| \leqslant B$, which is equivalent to the desired result $t = \mathsf{true}$.

– For the "$\Longleftarrow$" part of (22), we need to verify that the values stored in L are actual integral roots. This directly follows from (21h). □

3.2 Certificates for the Integer Small Value Problem (ISValP)

3.2.1 The problem at stake

In the previous Section 3.1, we have seen that we can take advantage of Hensel's lemma to devise some bivariate small integral roots certificates. We will now see how this kind of certificates can be reused to address the ISValP problem. For this, we introduce the list $\mathcal{S}$ and rephrase Problem 1 as:

Problem 3 (Certifying ISValP) Given

$$\begin{cases} P \in \mathbb{Z}[X], \\ A \in \mathbb{N}, \\ B \in \mathbb{N}, \\ M \in \mathbb{N}, \\ \mathcal{S} \subset \mathbb{Z} \times \mathbb{Z}, \end{cases} \tag{25}$$

we want to formally verify that the following property holds:

$$\forall (x, y) \in \mathbb{Z}^2, \quad |x| \leqslant A \ \wedge \ |y| \leqslant B \ \wedge \ P(y) \equiv x \pmod{M} \implies (x, y) \in \mathcal{S}. \tag{26}$$

Notice that the order of variables x and y in (26) is reversed with respect to Problem 1 and [32]. This different wording is due to some Coq technicalities, on which we will elaborate in Section 5.

3.2.2 Focusing on the SLZ algorithm

In order to understand which kind of certificates could be suitable for ISValP, we have to give a closer look to what the SLZ algorithm is doing. The main steps are summarized in Figure 1.

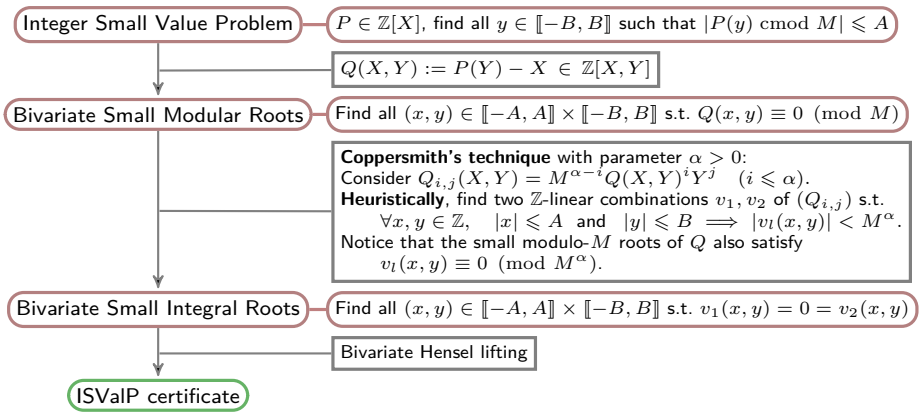

Fig. 1 Solving ISValP using the SLZ algorithm.

The algorithm starts from an ISValP instance and reduces it to a bivariate small integral root problem (to be solved by using Hensel lifting, as described in Section 3.1). Here is the proof of correctness of this reduction:

Proof (Reduction from an ISValP instance) Assume that $y \in [\![-B, B]\!]$ satisfies $|P(y) \bmod M| \leqslant A$. According to Equation (6) in Section 2.1, this amounts to saying that we have $(x, y) \in [\![-A, A]\!] \times [\![-B, B]\!]$ such that $P(y) \equiv x \pmod M$. Then, pose $Q(X, Y) := P(Y) - X \in \mathbb{Z}[X, Y]$, implying that

$$Q(x, y) \equiv 0 \pmod M. \tag{27}$$

Then, choose an integer $\alpha > 0$ (the Coppersmith parameter) and consider the family of polynomials

$$Q_{i,j}(X, Y) = Q(X, Y)^i M^{\alpha-i} Y^j \quad (i \leqslant \alpha).$$

Under heuristic assumptions, we may find two $\mathbb{Z}$-linear combinations[5] v_1, v_2 of $(Q_{i,j})$ such that

$$\forall (z, t) \in [\![-A, A]\!] \times [\![-B, B]\!], \quad |v_1(z, t)| < M^\alpha \text{ and } |v_2(z, t)| < M^\alpha. \tag{28}$$

By definition of $(Q_{i,j})$, v_1 and v_2, Equation (27) implies

$$v_1(x, y) \equiv 0 \equiv v_2(x, y) \pmod{M^\alpha}. \tag{29}$$

[5] These polynomials v_1 and v_2 can be found by means of the LLL algorithm for Euclidean lattice basis reduction, which is the most time consuming part of the overall algorithm SLZ. More details on the way the polynomials v_1 and v_2 are computed may be found in [30,32,31].

Combining (28) and (29) leads to

$$v_1(x, y) = 0 = v_2(x, y). \qquad \square$$

Note that the small integral roots of the reduced problem may happen to be a strict superset of the solutions of ISValP.

3.2.3 Designing the certificate

The certificate must obviously contain the initial values P, A, B, M. We also need to include the Coppersmith parameter α. As regards the linear combinations v_l for $l \in \{1, 2\}$, the polynomial

$$v_l = \sum_{i \leqslant \alpha} \sum_{j} u_{l;i,j} Q_{i,j}(X, Y),$$

is stored in the certificate as the polynomial

$$u_l = \sum_{i \leqslant \alpha} \sum_{j} u_{l;i,j} X^i Y^j.$$

We then need to include the values p, k, and L that correspond to the bivariate small integral root certificate. Altogether this gives the following Coq record:

```
Record cert_ISValP := Cert_ISValP
{ ic_P : {poly int}
; ic_A : nat
; ic_B : nat
; ic_M : nat
; ic_alpha : nat
; ic_u1 : {bipoly int}
; ic_u2 : {bipoly int}
; ic_p : nat
; ic_k : nat
; ic_L : list (int * int * bool)
}.
```

Let us give an example of ISValP certificate.

Example 3 (ISValP certificate) This example is based on the same numerical data as in Examples 1 and 2 :

$$\begin{aligned}\mathcal{C}'' := \Big(P, \left(A := 2^{87}\right), \left(B := 2^{4}\right), \left(M := 2^{92}\right), (\alpha := 2), u_1, u_2, (p := 3), (k := 6), \\ \Big(L := \big\{(2336732229695754577823 19, 11, \mathsf{true}), \\ (336478080626535692605744 40, -14, \mathsf{true})\big\}\Big)\Big),\end{aligned}$$

where $P(X) = 36506148256923413X^3 + 2870960318598879353241 6X^2$
$+ 1505210043517569258352331980 8X$
$+ 3945817816478696756615137147748352,$

and

$$\begin{cases} u_1(X,Y) = X^2 + (-6Y - 1593703)X + (9Y^2 + 4781109Y + 634972313052), \\ u_2(X,Y) = -25X + (76Y + 19921289). \end{cases}$$

3.2.4 The certificate checker

The algorithm of our ISValP certificate checker is described in Figure 2.

`check_ISValP` $(P, A, B, M, \alpha, u_1, u_2, p, k, L) \in \{\mathsf{true}, \mathsf{false}\}$

1. Compute $Q(X,Y) := P(Y) - X \in \mathbb{Z}[X,Y]$;
2. Compute $v_l(X,Y) = M^\alpha \cdot u_l\left(\frac{Q(X,Y)}{M}, Y\right)$ for $l \in \{1,2\}$;
3. Return true if and only if:

$$\begin{cases} \sum_{i,j} |v_{1;i,j}|\, A^i B^j < M^\alpha, \\ \sum_{i,j} |v_{2;i,j}|\, A^i B^j < M^\alpha, \\ (v_1, v_2, A, B, p, k, L) \text{ is a valid SIntRootP certificate (as specified in Section 3.1.3).} \end{cases}$$

Fig. 2 ISValP certificate checker.

In this algorithm, Line 1 computes the bivariate polynomial Q, then Line 2 performs a change of polynomial basis: the polynomials $v_1(X,Y)$ and $v_2(X,Y)$ are initially expressed in the polynomial basis $\{Q_{i,j}(X,Y)\}_{i,j}$ and we need to compute their coefficients in the usual monomial basis. In the formalization, we perform this change of polynomial basis in two steps:

1. First, we "lift" the coefficients of $u = \sum_{i \leqslant \alpha} \sum_j u_{ij} X^i Y^j$ from u_{ij} to $u_{ij} M^{\alpha - i}$;
2. Then in the polynomial $\sum_{i \leqslant \alpha} \sum_j u_{ij} M^{\alpha-i} X^i Y^j$, we replace the indeterminate X with the polynomial Q to get $v = \sum_{i \leqslant \alpha} \sum_j u_{ij} M^{\alpha-i} Q(X,Y)^i Y^j$; this may be accomplished by using a Horner-based polynomial composition.

Weighted Norm-1. Now let us explain Line 3 in Figure 2. A crucial ingredient for building our ISValP checker consists in formalizing a simple and efficient test to ensure that (28) is fulfilled. We introduce the notion of "weighted norm-1," defined for any $P \in \mathbb{Z}[X,Y]$ and $(A,B) \in \mathbb{N}^2$ by:

$$|P|(A,B) := \sum_{i,j} |P_{i,j}| A^i B^j \in \mathbb{N}.$$

This notion enjoys the following result:

Lemma 3 *For any $P \in \mathbb{Z}[X,Y]$ and any $(A,B) \in \mathbb{N}^2$, we have*

$$\forall x, y \in \mathbb{Z}, \quad |x| \leqslant A \quad \wedge \quad |y| \leqslant B \quad \Longrightarrow \quad |P(x,y)| \leqslant |P|(A,B).$$

Thanks to this result (and considering $P := v_l$), we can notice that it is *sufficient* to have $|v_l|(A,B) < M^\alpha$ to make sure that Equation (28) holds. (Note that the design of the SLZ algorithm internally relies on a similar notion, so the fact that it only provides a sufficient condition does not raise any issue.)

3.2.5 The formal correctness proof

The correctness claim for our ISValP checker is given by the following statement:

Theorem 3 (Correctness of `check_ISValP`) *For any ISValP certificate $\mathcal{C} := (P, A, B, M, \alpha, u_1, u_2, p, k, L)$ that is accepted, we have the following property:*

$$\forall (x,y) \in [\![-A, A]\!] \times [\![-B, B]\!], \quad P(y) \equiv x \pmod{M} \implies (x, y, \mathsf{true}) \in L.$$

As pointed out in Section 3.2.2, Coppersmith's technique proceeds by implication, so it finally leads to an implication in Theorem 3. In other words, all the solutions of the ISValP instance are gathered in the list L. At the very end (to compute the hardest-to-round points of the considered elementary function), it will suffice to check the solution candidates gathered in the list L.

In order to prove Theorem 3, we closely follow the reduction proof given in Section 3.2.2. We formally prove Lemma 3, then we combine it with Lemma 1 to obtain the following result:

Lemma 4 *For any $P \in \mathbb{Z}[X,Y]$ and any $(N,A,B) \in \mathbb{Z}^3$, we have:*

$$\begin{aligned} \forall x, y \in \mathbb{Z}, \quad |x| \leqslant A \quad \wedge \quad |y| \leqslant B \quad \wedge \quad & |P|\,(A,B) < N \\ \wedge \quad P(x,y) \equiv 0 \pmod{N} \implies & P(x,y) = 0. \end{aligned}$$

We use this result twice, one for each polynomial v_1, v_2. Then we apply Theorem 2 to get Theorem 3.

4 Effective certificate checkers

The certificate checkers that we have presented in Sections 3.1 and 3.2 have been developed using some data structures that are suitable for proving, but not for computing. To get effective computation, we need to re-implement our certificate checkers with effective data structures. This is a standard approach, see e.g. [14].

4.1 Focusing on modularity and efficiency

We implement the effective version of our certificate checkers in a modular way, relying on the `Module` system of Coq. These checkers are then proved correct with respect to the reference checkers presented in Section 3. A key aspect of our modular implementation is that we can easily change the arithmetic with which these checkers compute. Three instantiations are available:

- binary integers from Coq's standard library `ZArith` that represent numbers as list of booleans with linear access;
- machine-efficient integers from Coq's standard library `BigZ` that represents numbers as complete binary trees of machine integers with logarithmic access;
- *Integers Plus Positive Exponent* (IPPE) numbers that are unevaluated dyadic numbers $m \times 2^e$ with $e \geqslant 0$. We detail below the motivation and the formalization of this latter implementation.

This last instantiation has been motivated by the fact that the coefficients of the approximation polynomials that typically occur in the final verification chain for SLZ are floating-point numbers in radix-2. In order to switch to integer arithmetic, these polynomials are then scaled by a large power of the radix to generate an instance of ISValP. For instance, in some example presented in Section 4.2.1, the power of 2 to be considered is 2^{13660}, leading to a polynomial on $\mathbb{Z}$ whose degree-0 coefficient is $M \times 2^{10629}$ where M is an odd integer in $[\![2^{3032}, 2^{3033} - 1]\!]$. Factoring out the powers of 2 in all computation seems a reasonable optimization. In fact, it leads to an appreciable twofold speedup as we will see in Section 4.2.1.

Some obvious optimizations were mandatory in order to get reasonable checking time:

- First, the quantity $\det J_{v_1,v_2}(r,s)$ involved in Condition (21e) can be computed in several ways: we can either compute the determinant $\Delta(X,Y) = \frac{\partial v_1}{\partial X} \times \frac{\partial v_2}{\partial Y} - \frac{\partial v_2}{\partial X} \times \frac{\partial v_1}{\partial Y}$ and deduce the quantity $\Delta(r,s)$, or perform the Horner evaluation beforehand on each of the derivatives and compute the determinant of the resulting matrix. The latter calculation has the benefit of not requiring multiplication of polynomials, since the determinant acts on a matrix in $\mathcal{M}_2(\mathbb{Z})$ instead of $\mathcal{M}_2(\mathbb{Z}[X,Y])$. For implementing this calculation, we benefit from our formalization of 2-by-2 matrices with coefficients in an arbitrary type.

- Second, the case where L is empty is very frequent so worth optimizing. In particular, when this list is empty, some computations like that of the Jacobian determinant above are unnecessary.

- The last optimization concerns the condition (21c). This condition checks whether the list
$$L_p = \{(r,s) \bmod p \mid \exists t \in \{\mathsf{true}, \mathsf{false}\}\,,\ (r,s,t) \in L\}$$
satisfies
$$\forall (x,y) \in [\![0, p-1]\!]\,, \quad (x,y) \in L_p \iff v_1(x,y) \equiv 0 \equiv v_2(x,y) \pmod p.$$
Since the coefficients of v_1 and v_2 are huge, the polynomials $v_1(X,Y) \pmod p$ and $v_2(X,Y) \pmod p$ are computed once before performing the different Horner evaluations.

4.2 Evaluation of the performances of the ISValP checker

The current section is organized as follows. First, we evaluate the performances of the current version of our library by focusing on four ISValP instances that were

first presented in [24, Section 5.4.5] and which cover a broad spectrum of situations related to our target application (solving the TMD using SLZ). Second, we present a complete verification of an ISValP computation that addresses the full range of an exponent, in order to demonstrate the scalability of our Coq implementation.

All the Coq benchmarks have been computed on an Intel® Xeon® X5482 CPU clocked at 3.20 GHz, relying on the following two versions of Coq: the experimental native-coq[6] branch of Coq [6] and the current stable version of Coq (version 8.4pl2). Note that the native-coq version is obviously faster but it has also a larger trusted computing base since it introduces another trusted link with the entire OCaml compiler.

4.2.1 Four typical instances of ISValP

For each instance of ISValP, Table 2 gives the function at stake, the target precision n, the TMD precision n', the normalized (or scaled) approximation polynomial $P \in \mathbb{Z}[X]$, the modulo M, and the bounds A and B. These instances have been chosen so that

- #1, #2, and #3 deal with the binary64 format, while #4 deals with binary128;
- #1 contains a hardest-to-round point, unlike other instances;
- #3 and #4 deal with the Approximate TMD (cf. Section 1.2), the value of n' being much larger than $2 \times n$.

Instance of ISValP	f	n	n'	$\deg(P)$	$\max_i(\lvert P_i \rvert)$	M	A	B
#1	exp	53	100	2	$\lesssim 1.68 \times 2^{237}$	2^{185}	2^{139}	2^{12}
#2	exp	53	100	2	$\lesssim 1.22 \times 2^{237}$	2^{185}	2^{139}	2^{12}
#3	exp	53	300	12	$\lesssim 1.36 \times 2^{996}$	2^{942}	2^{696}	2^{32}
#4	exp	113	3000	90	$\lesssim 1.36 \times 2^{13661}$	2^{13547}	2^{10661}	2^{72}

Table 2 Short description of four instances of ISValP.

For a given parameter α, SLZ produces the polynomials v_1 and v_2, and runs the bivariate Hensel lifting to generate the ISValP certificates. Table 3 summarizes the data stored in the certificates, and the timings for their verification inside Coq. The last four columns give the timings using the compiled `native_compute` reduction tactic of native-coq (using IPPE numbers, then `BigZ` numbers), and the timings using the interpreted `vm_compute` reduction tactic of Coq 8.4pl2 (again using IPPE numbers, then `BigZ` numbers).

In what follows, we focus on the last two rows of Table 3 (as using native-coq yields no speedup for the small examples #1 and #2, because of the compilation overhead). These figures indicate that using IPPE integers in place of the standard `BigZ` integers yields a 1.2x to 3.9x speedup, and using the compiled `native_compute` in place of the interpreted `vm_compute` yields a 2.5x to 9.5x speedup. Also, the fastest version of the Coq checker is only 12 to 19 times slower than Maple. This is reasonable if we consider that the formal checker is executed in a trusted framework with restricted computing power, unlike Maple that notably takes advantage of the GMP library.

[6] URL: `https://github.com/maximedenes/native-coq`

Inst.	α	M^α	p	size(L)	CPU time to return true				
					Maple 15	native-coq		Coq 8.4pl2	
						IPPE	BigZ	IPPE	BigZ
#1	2	2^{370}	5	1	0.003s	0.162 s	0.140 s	0.119 s	0.103 s
#2	2	2^{370}	23	0	0.006s	0.158 s	0.179 s	0.128 s	0.149 s
#3	4	2^{3768}	5	0	0.029s	0.369 s	0.455 s	0.926 s	1.843 s
#4	6	2^{81282}	5	0	56.14s	$17'$ 21s	$30'$ 44s	1h $15'$ 30s	4h $51'$ 24s

Table 3 Short description of several ISValP certificates with their verification timing in Coq. The first column refers to Table 2.

4.2.2 Large-scale benchmarks for our formalized ISValP checker

We now focus on sets of certificates that are needed to address the full range of an exponent in binary64. We recall that the bound B involved in the ISValP certificates directly indicates the number of floating-point numbers that are taken into account. As a result, we would need $2^{53-1}/B = 2^{20}$ certificates similar to the ISValP instance #3 of Table 3 to address all the normal floating-point numbers of one exponent of binary64. We can then extrapolate the overall timing that would be required in this context: $0.369\,\text{s} \times 2^{20} \approx 108$ hours. As a matter of fact, we can do much better by taking a larger value of α for the SLZ algorithm. For the parameters given in Table 4, only $2^{53-1}/B = 2^{12}$ certificates are necessary for addressing the same range. The generation of each certificate took approximately 140 s, which leads to around 20 h of real time on a 8-core Intel® Xeon® X5550 CPU clocked at 2.67 GHz. Roughly 95 % of this time was spent in the lattice reduction calls, on a 105×3056 matrix with 700-bit integer entries.

All these certificates were verified on another 8-core machine (the same machine as in Section 4.2.1) by compiling 8 different Coq files, each containing the *evaluation* (i.e., `Eval native_compute` or `Eval vm_compute`) of the ISValP checker on 512 certificates. The average verification time (with standard deviation) for the considered population of ISValP certificate is given in Table 5. We can see that using the `native_compute` reduction tactic in place of `vm_compute` yields a speedup of 3.2x on average. Almost all certificates contain no root and are relatively easy to verify. Also, there is a ratio of 4x (= 140 s/34.9 s) between the generation using SAGE (which relies, mostly, on the MPIR, FLINT and fpLLL libraries) and the verification in Coq. This means that using certificates, the computation has been formally verified with an overhead of 25 % only. As shown by our implementation of the checker in

f	n	n'	range	B	α	total size of input files
exp	53	300	$[1, 2]$	2^{40}	13	$\lesssim 100$ MB

Table 4 Short description of the common values of 4096 ISValP certificates.

Class of certificates	Average CPU time ± std deviation to accept one certificate	
	native-coq & `native_compute`	Coq 8.4pl2 & `vm_compute`
4090 certs. s.t. size(L) = 0	34.6 s ± 6.6 s	112.5 s ± 20.3 s
6 certs. s.t. size(L) = 1	221.3 s ± 49.6 s	468.5 s ± 111.7 s
all the 4096 certificates	34.9 s ± 9.9 s	113.0 s ± 24.8 s

Table 5 Some statistics on the formal verification of 4096 ISValP certificates within Coq, using the "`bigZ` × `bigN`" implementation of IPPE integers. The biggest chosen prime is $p_{\max} := 47$.

Maple, progress could be made to further reduce this overhead. Note that for the very big certificate #4 of Table 3, the formalized ISValP checker is only one order of magnitude slower than `Maple`.

5 Discussion

In order to prove the correctness of our checkers, a variety of mathematical notions has to be formalized: Taylor's theorem for bivariate polynomials, Jacobian matrix for pairs of bivariate polynomials, Cramer's rule for order-2 matrices in modular arithmetic, and weighted norm-1 for bivariate polynomials among others.

Taylor's theorem already existed in Coq for real analysis. It has been reproved independently so as to be applicable to polynomials over an arbitrary ring (this is integrated in the standard distribution of SSReflect). This leads to the notion of normalized derivatives that uses binomial coefficients to internalize the division by $\frac{1}{n!}$ that is not available on a ring. Doing so, we also avoid a dependency with the real analysis and its axiomatic approach. A similar dependency could also occur for the IPPE integers, since they rely on the Coq.Interval library. We carefully checked that no dependency was introduced also here. So, our development is axiom-free.

A careful reader would have noticed that in Section 3.2, we have interchanged the two variables X and Y with respect to [32, Section 3.2], which was considering $Q_{i,j}(X,Y) = X^i Q^j(X,Y) M^{\alpha-j}$ $(j \leqslant \alpha)$. This technical detail comes from the way we have represented bivariate polynomials: $\mathbb{Z}[X,Y]$ is encoded as $(\mathbb{Z}[Y])[X]$. A consequence of the asymmetry between the two variables X and Y is that we get for free the substitution of a polynomial $Q(X,Y)$ for the variable X in $P(X,Y)$ with the simple univariate composition $P \circ Q$. The other substitution with the variable Y is less direct. The ISValP checker is then encoded using the first substitution.

Formalizing effective implementations of algorithms in a proof assistant is often a challenging task. A key ingredient that we have been using here is to develop modular code. The Coq proof assistant provides three mechanisms for modularity: *type classes* [29], *canonical structures* [28], and *modules* [10,9]. Modules are less generic than the other two (which are first-class citizens) but they have a better computational behavior. Indeed, module applications are performed statically, so the code that is executed is often more compact. We have been using modules to develop the generic implementation presented in Section 4.1. In particular, this has clearly simplified the formalization of IPPE integers, since the Coq.Interval library is also relying on modules.

An aspect of our work that could be largely automated is the refinement from the naive checker that works on a simple data-structure to the more efficient one. For the moment, this is done in an ad-hoc way using morphisms between data-structures and explicitly applying rewriting rules for the correctness proof. The use of a more systematic approach such as the ones proposed in [11,18,22] would be a clear improvement.

6 Conclusion and perspectives

In this work, we have built several effective verified certificate checkers upon Hensel's lemma. In particular, our ISValP certificate checker is suitable to validate that all

the solutions of given instances of the Integer Small Value Problem (ISValP) have been found. This problem consists of *finding all the small integer entries on which a univariate polynomial* $P \in \mathbb{Z}[X]$ *has small values modulo a large integer* M. It can be solved by using the SLZ algorithm, which is based on a variant of Coppersmith's technique [12]. We recall that this algorithmic chain consists of four main steps: (i) domain splitting, then on each sub-domain: (ii) polynomial approximation, (iii) the modular, bivariate version of Coppersmith's technique, and (iv) bivariate Hensel lifting. The formalization presented here constitutes a key formal component for certifying the SLZ algorithm. In this paper, we have devised some certificates that address steps (iii) and (iv) in a formal setting. To demonstrate the scalability of this component, we have evaluated its performances within the Coq proof assistant on realistic examples generated by the SLZ algorithm.

We have been using the Coq proof assistant to design certificates that make it possible to validate a long computation. We have derived some reference checkers that have been proved correct. In our case, they were efficient enough to be used for validation. It is always debatable how much trust one can have in a Coq computation. What can be said is that all the components that we have been using (starting from arbitrary precision arithmetic) have been designed with a very strong concern about correctness.

Hensel's lemma has already been formalized in an abstract setting in the Isabelle/HOL proof assistant [21]. To the best of our knowledge, our work constitutes the first formalization of bivariate Hensel's uniqueness lemma along with verified, Hensel-based certificate checkers. Further optimizations of our certified library could be performed. For example, our formalization of quadratic Hensel lifting naturally yields an arithmetic modulo p^{2^k}, while we could restrict ourselves to the largest power $p^m > 2 \cdot \max(A, B)$ in the final certificates. Even if this would have no impact on most certificates (those whose list L is empty), it would be interesting to measure the performance gain for the other few certificates. Finally we would like to improve the multiplication algorithm (and thereby, the bivariate polynomial composition one) by implementing a Karatsuba-based approach [20]. This may lead to a further speedup for verifying some "extreme" certificates such as our binary128 example in Section 4.2.1, while keeping the confidence we have in our formally verified ISValP certificate checker.

Our next priority will be to combine the CoqHensel machinery presented here with that of the CoqApprox library for certified polynomial approximation [8, 25]. We will then get a validation for the whole SLZ algorithm for solving the Table Maker's Dilemma. It will then be possible to provide certificates that formally guarantee that *we know all the bad cases* for correctly rounding a given mathematical function.

Finally the part of this work which, beyond Hensel's lemma, addresses Coppersmith's technique, might have, up to slight modifications, some use in applications of the latter, for instance the list decoding algorithm for CRT-type codes, see [7, 2], or for RS-codes already pointed out in Section 1.

A Two additional mathematical proofs

The two propositions below illustrate the usefulness of Lemma 2 beyond the formal proof of the certificate checkers presented in the paper.

Proposition 1 (Correctness of Algorithm 2) *For any* $(v_1, v_2) \in \mathbb{Z}[X, Y]^2$, $(A, B) \in \mathbb{N}^2$, *if there exists a prime* p *satisfying* (7), *then the output of Algorithm 2 satisfies*

$$S = \left\{(x, y) \in \mathbb{Z}^2 \;\middle|\; v_1(x, y) = 0 = v_2(x, y) \quad \wedge \quad |x| \leqslant A \quad \wedge \quad |y| \leqslant B\right\}.$$

Proof (Proposition 1) First, let us notate

$$R := \left\{(x, y) \in \mathbb{Z}^2 \;\middle|\; v_1(x, y) = 0 = v_2(x, y) \quad \wedge \quad |x| \leqslant A \quad \wedge \quad |y| \leqslant B\right\}.$$

In order to prove that $S = R$, we proceed by double-inclusion:

- First, we have $S \subset R$ since S is initially empty, and each element added to S by Algorithm 2 on Line 13 does belong to R;
- Next, let $(x, y) \in \mathbb{Z}^2$ be an element of R. Note that $v_1(x, y) = 0 = v_2(x, y)$ implies

$$v_1(x \bmod p, y \bmod p) \equiv 0 \equiv v_2(x \bmod p, y \bmod p) \pmod p. \tag{30}$$

  So the **foreach** loop of Algorithm 2 is executed for $(r, s) := (x \bmod p, y \bmod p)$. In particular, it iterates bivariate Hensel lifting on this modulo-p root of (v_1, v_2), and the corresponding **while** loop stops as soon as we have $K > 2 \cdot \max(A, B)$. Let us denote by ℓ the final value of k (at the end of the **while** loop). Then notice that the successive values of variables (a, b) in this loop correspond to a sequence $(a_k, b_k)_{0 \leqslant k \leqslant \ell}$ that satisfies relations (9) and (10). Given that (7) holds by hypothesis, and that (8) is trivially satisfied by the integral root (x, y), we can apply Lemma 2. To sum up, we have

$$\begin{cases} K = p^{2^\ell}, & \\ (a_\ell, b_\ell) \equiv (x, y) \pmod K & \text{by taking } n := \ell \text{ in (11)}, \\ |a_\ell| \leqslant K/2, & \\ |b_\ell| \leqslant K/2 & \text{by definition of cmod}, \\ |x| \leqslant A < K/2, & \\ |y| \leqslant B < K/2 & \text{by hypothesis}. \end{cases}$$

  Then we deduce that $x = a_\ell$ and $y = b_\ell$ by applying twice Corollary 1. Thence, the small integral root (x, y) of (v_1, v_2) has indeed been found and added to S by Algorithm 2. □

Proposition 2 (Necessary condition on prime p) *If Hypothesis* (7) *holds, then* p *has to be greater than or equal to the square root of the number of the bivariate integral roots of* (v_1, v_2).

Proof (Proposition 2) Let us denote by R_0 the set of integral roots of (v_1, v_2), i.e.,

$$R_0 = \left\{(x, y) \in \mathbb{Z}^2 \;\middle|\; v_1(x, y) = 0 = v_2(x, y)\right\},$$

and let us show that (7) implies $\# R_0 \leqslant p^2$. It suffices to exhibit an injection from R_0 into $[\![0, p-1]\!]^2$. For instance, let us show that $g : (x, y) \in R_0 \mapsto (x \bmod p, y \bmod p) \in [\![0, p-1]\!]^2$ is injective. Let (a, b) and (c, d) be two elements of R_0 satisfying $g(a, b) = g(c, d)$ and let $A := \max(|a|, |c|)$ and $B := \max(|b|, |d|)$. If we denote by k the smallest natural number satisfying $K := p^{2^k} > 2 \cdot \max(A, B)$, we have

$$\begin{cases} v_1(a, b) \equiv 0 \equiv v_2(a, b) \pmod K & \text{as } (a, b) \in R_0, \\ v_1(c, d) \equiv 0 \equiv v_2(c, d) \pmod K & \text{as } (c, d) \in R_0, \\ (a, b) \equiv (c, d) \bmod p & \text{as } g(a, b) = g(c, d), \\ (a, b) \equiv (c, d) \bmod K & \text{by Corollary 2, itself based on Lemma 2}, \\ |a| \leqslant A < K/2, & \\ |b| \leqslant B < K/2, & \\ |c| \leqslant A < K/2, & \\ |d| \leqslant B < K/2 & \text{by definition of } A, B, \text{ and } K. \end{cases}$$

We can then apply Corollary 1 to deduce that $(a, b) = (c, d)$. Thence, we have $p \geqslant \sqrt{\# R_0}$. □

Acknowledgements This research was partly funded by the TaMaDi project of the *Agence Nationale de la Recherche* (ref. ANR-2010-BLAN-0203-01). It was mainly done while the first author was PhD student in *École Normale Supérieure de Lyon*, with the LIP lab.

References

1. Daniel Augot and Lancelot Pecquet. A Hensel Lifting to Replace Factorization in List-Decoding of Algebraic-Geometric and Reed-Solomon Codes. *Information Theory, IEEE Transactions on*, 46(7):2605–2614, November 2000.
2. Daniel J. Bernstein. Simplified High-Speed High-Distance List Decoding for Alternant Codes. In Bo-Yin Yang, editor, *PQCrypto*, volume 7071 of *LNCS*, pages 200–216. Springer, 2011.
3. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.
4. Yves Bertot, Georges Gonthier, Sidi Ould Biha, and Ioana Pasca. Canonical Big Operators. In Mohamed et al. [26], pages 86–101.
5. Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors. *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, volume 7998 of *LNCS*. Springer, 2013.
6. Mathieu Boespflug, Maxime Dénès, and Benjamin Grégoire. Full Reduction at Full Throttle. In Jean-Pierre Jouannaud and Zhong Shao, editors, *CPP*, volume 7086 of *LNCS*, pages 362–377. Springer, 2011.
7. Dan Boneh. Finding Smooth Integers in Short Intervals Using CRT Decoding. *J. Comput. Syst. Sci.*, 64(4):768–784, 2002.
8. Nicolas Brisebarre, Mioara Joldeş, Érik Martin-Dorel, Micaela Mayero, Jean-Michel Muller, Ioana Paşca, Laurence Rideau, and Laurent Théry. Rigorous Polynomial Approximation Using Taylor Models in Coq. In Alwyn Goodloe and Suzette Person, editors, *NASA Formal Methods 2012*, volume 7226 of *LNCS*, pages 85–99. Springer, 2012.
9. Jacek Chrząszcz. Implementing Modules in the Coq System. In David A. Basin and Burkhart Wolff, editors, *TPHOLs*, volume 2758 of *LNCS*, pages 270–286. Springer, 2003.
10. Jacek Chrząszcz. Modules in Coq Are and Will Be Correct. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *TYPES*, volume 3085 of *LNCS*, pages 130–146. Springer, 2003.
11. Cyril Cohen, Maxime Dénès, and Anders Mörtberg. Refinements for free! Accepted for publication at CPP, 2013. Available from: `http://www.maximedenes.fr/download/refinements.pdf`.
12. Don Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.
13. The Coq Development Team. *The Coq Proof Assistant: Reference Manual: version 8.4pl2*, 2013. Available from: `http://coq.inria.fr/distrib/V8.4pl2/refman/`.
14. Maxime Dénès, Anders Mörtberg, and Vincent Siles. A Refinement-Based Approach to Computational Algebra in Coq. In Lennart Beringer and Amy P. Felty, editors, *ITP*, volume 7406 of *LNCS*, pages 83–98. Springer, 2012.
15. Georges Gonthier and Assia Mahboubi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, INRIA, 2008.
16. Georges Gonthier and Assia Mahboubi. An introduction to small scale reflection in Coq. *Journal of Formalized Reasoning*, 3(2):95–152, 2010.
17. Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.
18. Florian Haftmann, Alexander Krauss, Ondrej Kuncar, and Tobias Nipkow. Data Refinement in Isabelle/HOL. In Blazy et al. [5], pages 100–115.
19. Kurt Hensel. Neue Grundlagen der Arithmetik. *Journal für die reine und angewandte Mathematik (Crelle's Journal)*, 1904(127):51–84, 1904. 10.1515/crll.1904.127.51.
20. A. Karatsuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. *Doklady Akad. Nauk SSSR*, 145:293–294, 1962. Translation in Physics-Doklady 7, 595–596, 1963.

21. Hidetune Kobayashi, Hideo Suzuki, and Yoko Ono. Formalization of Hensel's lemma. In *Theorem Proving in Higher Order Logics: Emerging Trends Proceedings*, number PRG-RR-05-02 in Oxford University Computing Laboratory Research Reports, pages 114–127, 2005.
22. Peter Lammich. Automatic Data Refinement. In Blazy et al. [5], pages 84–99.
23. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.
24. Érik Martin-Dorel. *Contributions to the Formal Verification of Arithmetic Algorithms*. PhD thesis, École Normale Supérieure de Lyon, Lyon, France, 2012. Available from: `http://erik.martin-dorel.org/papers/MARTIN-DOREL_Erik_2012_these.pdf`.
25. Érik Martin-Dorel, Micaela Mayero, Ioana Paşca, Laurence Rideau, and Laurent Théry. Certified, Efficient and Sharp Univariate Taylor Models in COQ. In *SYNASC 2013*, Timişoara, Romania, 2013. IEEE CPS. 8 pages. To appear. Available from: `http://hal.inria.fr/hal-00845791_v2/en/`.
26. Otmane Aït Mohamed, César Muñoz, and Sofiène Tahar, editors. *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *LNCS*. Springer, 2008.
27. Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Clause-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.
28. Amokrane Saïbi. Typing Algorithm in Type Theory with Inheritance. In *POPL*, pages 292–301, 1997.
29. Matthieu Sozeau and Nicolas Oury. First-Class Type Classes. In Mohamed et al. [26], pages 278–293.
30. Damien Stehlé. *Algorithmique de la réduction des réseaux et application à la recherche de pires cas pour l'arrondi des fonctions mathématiques*. PhD thesis, Université Nancy 1 Henri Poincaré, December 2005.
31. Damien Stehlé. On the Randomness of Bits Generated by Sufficiently Smooth Functions. In Florian Hess, Sebastian Pauli, and Michael E. Pohst, editors, *Algorithmic Number Theory, 7th International Symposium, ANTS-VII, Berlin, Germany, July 23-28, 2006, Proceedings*, volume 4076 of *LNCS*, pages 257–274. Springer, 2006.
32. Damien Stehlé, Vincent Lefèvre, and Paul Zimmermann. Searching Worst Cases of a One-Variable Function Using Lattice Reduction. *IEEE Transactions on Computers*, 54(3):340–346, March 2005.
33. G.W. Stewart. On the adjugate matrix. *Linear Algebra and its Applications*, 283(1–3):151–164, 1998.
34. Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2nd edition, 2003.