Designing an Algorithmic Proof of the Two-Squares Theorem

João F. Ferreira*

School of Computer Science
University of Nottingham
Nottingham NG8 1BB, England
joao@joaoff.com

Abstract. We show a new and constructive proof of the two-squares theorem, based on a somewhat unusual, but very effective, way of rewriting the so-called extended Euclid's algorithm. Rather than simply verifying the result — as it is usually done in the mathematical community — we use Euclid's algorithm as an interface to *investigate* which numbers can be written as sums of two positive squares. The precise formulation of the problem as an algorithmic problem is the key, since it allows us to use algorithmic techniques and to avoid guessing. The notion of invariance, in particular, plays a central role in our development: it is used initially to observe that Euclid's algorithm can actually be used to represent a given number as a sum of two positive squares, and then it is used throughout the argument to prove other relevant properties. We also show how the use of program inversion techniques can make mathematical arguments more precise.

Keywords: algorithm derivation, sum of two squares, Euclid's algorithm, invariant, program inversion

1 Introduction

Which numbers can be written as sums of two squares? According to Dickson [1, p. 225], this classic question in number theory was first discussed by Diophantus, but it is usually associated with Fermat, who stated in 1659 that he possessed an irrefutable proof that every prime of the form $4k + 1$ can be written as the sum of two squares. (He first communicated the result to Mersenne, in a letter dated December 25, 1640; for this reason, this result is sometimes called *Fermat's Christmas Theorem*. Incidentally, Dickson names this result after Albert Girard, who, in 1632, was the first to state it. We follow Dickson's convention and we also refer to the two-squares theorem as Girard's result.) However, as with many other of his results, Fermat did not record his proof. The first recorded proof of Girard's result is due to Euler who proved it in 1749, "after he had struggled, off and on, for *seven years* to find a proof" [2, p. 69]. Euler communicated his

* Funded by Fundação para a Ciência e a Tecnologia (Portugal) under grant SFRH/BD/24269/2005

five-step argument in a letter to Goldbach dated 6 May 1747, but the fifth step was only made precise in a second letter written in 1749. In 1801, Gauss proved for the first time that such prime numbers are *uniquely* represented as the sum of two positive integers [3, Art. 182].

This classic theorem attracted the attention of many mathematicians. Since Euler's proof by the method of infinite descent, Lagrange proved it using quadratic forms (subsequently, Gauss simplified Lagrange's proof in [3, Art. 182]); Dedekind used Gaussian integers; Serret and Hermite used continued fractions [4, 5]; Brillhart improved Hermite's argument using Euclid's algorithm [6]; Smith used continuants [7]; more recently, Zagier [8] published a one-sentence proof based on an involution of a particular finite set (see also [9, chapter 4] and [10] for a detailed explanation of the proof); and Wagon [11] gave a self-contained proof based on Euclid's algorithm and on [6].

Like Brillhart and Wagon, we present a proof that is also based on Euclid's algorithm, but, rather than simply verifying Girard's result, we use the algorithm as an interface to *investigate* which numbers can be written as sums of two positive squares[1]. The precise formulation of the problem as an algorithmic problem is the key, since it allows us to use algorithmic techniques and to avoid guessing. The notion of invariance, in particular, plays a central role in our development: it is used initially to observe that Euclid's algorithm can actually be used to represent a given number as a sum of two positive squares, and then it is used throughout the argument to prove other relevant properties. We also show how the use of program inversion techniques can make mathematical arguments more precise. As we will see, the end result is also more general than the one conjectured by Girard.

In the next section we show how we can use our formulation of Euclid's algorithm to prove the theorem. At the end of the section, we describe how the argument and the paper are organised.

2 Euclid's Algorithm

We start with a somewhat unusual, but very effective, way of rewriting Euclid's algorithm when the goal is to establish the theorem that the greatest common divisor of two numbers is a linear combination of the numbers. (This is sometimes called the extended Euclid's algorithm. See [12] for a derivation of the algorithm and [13] for another problem whose solution is based on the algorithm. Also, we use "$\triangledown$" to denote "greatest common divisor". We prefer to use an infix notation whenever — as in this case — the operator is symmetric and associative.)

The algorithm is expressed in matrix terms. The input to the algorithm is a vector $(m\ n)$ of strictly positive integers. The vector $(x\ y)$ is initialised to $(m\ n)$ and, on termination, its value is the vector $(m \triangledown n\ \ m \triangledown n)$. (This is a

[1] Every square number m^2 can be written as m^2+0^2. However, this type of solution is not considered in this paper, since our formulation of Euclid's algorithm deals only with positive numbers. Therefore, our construction aims to express a number as the sum of two *positive* squares.

consequence of the invariant $(x\ y) = (m \triangledown n\ \ m \triangledown n)$. We omit this invariant in the comments below to simplify the presentation.) In addition to computing the greatest common divisor, it also computes a matrix $\mathbf{C}$. An invariant of the algorithm is that the vector $(x\ y)$ equals $(m\ n) \times \mathbf{C}$. In words, $(x\ y)$ is a "linear combination" of $(m\ n)$. Specifically, $\mathbf{I}$, $\mathbf{A}$, and $\mathbf{B}$ are 2×2 matrices; $\mathbf{I}$ is the identity matrix $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, $\mathbf{A}$ is the matrix $\left(\begin{smallmatrix} 1 & 0 \\ -1 & 1 \end{smallmatrix}\right)$, and $\mathbf{B}$ is the matrix $\left(\begin{smallmatrix} 1 & -1 \\ 0 & 1 \end{smallmatrix}\right)$. The assignment $(x\ y) := (x\ y) \times \mathbf{A}$ is equivalent to $x, y := x - y, y$, as can be easily checked.

$$
\begin{array}{l}
\{\ \ 0 < m\ \ \wedge\ \ 0 < n\ \ \} \\
(x\ y), \mathbf{C} := (m\ n), \mathbf{I}\ ; \\
\{\ \ \textbf{Invariant:}\quad (x\ y) = (m\ n) \times \mathbf{C}\ \ \} \\
\mathsf{do}\ \ y < x \quad \rightarrow \quad (x\ y), \mathbf{C} := (x\ y) \times \mathbf{A}\ ,\ \mathbf{C} \times \mathbf{A} \\
\Box\ \ x < y \quad \rightarrow \quad (x\ y), \mathbf{C} := (x\ y) \times \mathbf{B}\ ,\ \mathbf{C} \times \mathbf{B} \\
\mathsf{od} \\
\{\ \ (x\ y) = (m \triangledown n\ \ m \triangledown n) = (m\ n) \times \mathbf{C}\ \ \}
\end{array}
$$

The verification of the supplied invariant is a simple consequence of the associativity of matrix multiplication. Also, note that the algorithm constructs two linear combinations of m and n equal to their greatest common divisor.

A key insight in our development is that matrices $\mathbf{A}$ and $\mathbf{B}$ are invertible, which allows us to rewrite the invariant as $(x\ y) \times \mathbf{C}^{-1} = (m\ n)$, where the matrix $\mathbf{C}^{-1}$ is a finite product of the matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$, which are, respectively, $\left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$. In fact, we can change the above algorithm to compute the matrix $\mathbf{C}^{-1}$ instead; renaming $\mathbf{C}^{-1}$ to $\mathbf{D}$, $\mathbf{A}^{-1}$ to $\mathbf{L}$, and $\mathbf{B}^{-1}$ to $\mathbf{R}$, we rewrite it as follows:

$$
\begin{array}{l}
\{\ \ 0 < m\ \ \wedge\ \ 0 < n\ \ \} \\
(x\ y), \mathbf{D} := (m\ n), \mathbf{I}\ ; \\
\{\ \ \textbf{Invariant:}\quad (x\ y) \times \mathbf{D} = (m\ n)\ \ \} \\
\mathsf{do}\ \ y < x \quad \rightarrow \quad (x\ y), \mathbf{D} := (x\ y) \times \mathbf{L}^{-1}\ ,\ \mathbf{L} \times \mathbf{D} \\
\Box\ \ x < y \quad \rightarrow \quad (x\ y), \mathbf{D} := (x\ y) \times \mathbf{R}^{-1}\ ,\ \mathbf{R} \times \mathbf{D} \\
\mathsf{od} \\
\{\ \ (x\ y) = (m \triangledown n\ \ m \triangledown n) \qquad \wedge \qquad (m \triangledown n\ \ m \triangledown n) \times \mathbf{D} = (m\ n)\ \ \}
\end{array}
$$

It is this form of the algorithm that is the starting point for our investigation. Note that if $\mathbf{D} = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, the invariant is equivalent to

$$(m\ n) \quad = \quad (x\ y) \times \mathbf{D} \quad = \quad (x \times a + y \times c\ \ \ x \times b + y \times d) \quad ,$$

which means that if, at any point in the execution of the algorithm, $(x\ y)$ equals $(a\ c)$, we can conclude that m is a sum of two positive squares, that is:

$$(m\ n) \quad = \quad (a\ c) \times \mathbf{D} \quad = \quad (a{\times}a + c{\times}c \quad a{\times}b + c{\times}d) \quad .$$

Symmetrically, if, at any point in the execution of the algorithm, $(x\ y)$ equals $(b\ d)$, we can conclude that n is a sum of two positive squares.

It may help to visualise an execution trace of the algorithm. Table 1 depicts the execution trace when the arguments are $m = 17$ and $n = 4$. Each row of the table shows the state-space and the value of the invariant after each iteration of the algorithm. The first two columns show the values of the variables $(x\ y)$ and $\mathbf{D}$, respectively. The third column shows how the invariant is satisfied, according to the values of the first two columns. The first row corresponds to the initial state and the last row corresponds to the final state.

$(x\ y)$	$\mathbf{D}$, the same as $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$	**Invariant:** $(m\ n) = (x{\times}a + y{\times}c \quad x{\times}b + y{\times}d)$
$(17\ 4)$	$\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) = \mathbf{I}$	$(17\ 4) = (17{\times}1 + 4{\times}0 \quad 17{\times}0 + 4{\times}1)$
$(13\ 4)$	$\left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right) = \mathbf{L}$	$(17\ 4) = (13{\times}1 + 4{\times}1 \quad 13{\times}0 + 4{\times}1)$
$(9\ 4)$	$\left(\begin{smallmatrix} 1 & 0 \\ 2 & 1 \end{smallmatrix}\right) = \mathbf{LL}$	$(17\ 4) = (9{\times}1 + 4{\times}2 \quad 9{\times}0 + 4{\times}1)$
$(5\ 4)$	$\left(\begin{smallmatrix} 1 & 0 \\ 3 & 1 \end{smallmatrix}\right) = \mathbf{LLL}$	$(17\ 4) = (5{\times}1 + 4{\times}3 \quad 5{\times}0 + 4{\times}1)$
$(\mathbf{1}\ \mathbf{4})$	$\left(\begin{smallmatrix} \mathbf{1} & 0 \\ \mathbf{4} & 1 \end{smallmatrix}\right) = \mathbf{LLLL}$	$(17\ 4) = (\mathbf{1}{\times}\mathbf{1} + \mathbf{4}{\times}\mathbf{4} \quad 1{\times}0 + 4{\times}1)$
$(1\ 3)$	$\left(\begin{smallmatrix} 5 & 1 \\ 4 & 1 \end{smallmatrix}\right) = \mathbf{RLLLL}$	$(17\ 4) = (1{\times}5 + 3{\times}4 \quad 1{\times}1 + 3{\times}1)$
$(1\ 2)$	$\left(\begin{smallmatrix} 9 & 2 \\ 4 & 1 \end{smallmatrix}\right) = \mathbf{RRLLLL}$	$(17\ 4) = (1{\times}9 + 2{\times}4 \quad 1{\times}2 + 2{\times}1)$
$(1\ 1)$	$\left(\begin{smallmatrix} 13 & 3 \\ 4 & 1 \end{smallmatrix}\right) = \mathbf{RRRLLLL}$	$(17\ 4) = (1{\times}13 + 1{\times}4 \quad 1{\times}3 + 1{\times}1)$

Table 1. Execution trace of Euclid's algorithm for arguments $m = 17$ and $n = 4$

As we can see in table 1, there is a point at which $x = a = 1$ and $y = c = 4$; it follows directly from the invariant that 17 can be expressed as the sum of two positive squares ($17 = 1^2 + 4^2$).

One question that now arises is what is so special about the numbers 17 and 4 that made the vectors $(x\ y)$ and $(a\ c)$ to be equal. (Had we used as arguments the numbers 17 and 5, for example, x would never equal a.) Put more generally, how can we characterise the arguments that make the vectors $(x\ y)$ and $(a\ c)$ to be equal at some point in the execution of the algorithm?

A closer inspection of the values shown in table 1 can help us answering the general question. If we ignore the first row, we see that the sequence of successive values of the vector $(x\ y)$ is the reverse of the sequence of successive values of $(a\ c)$. Also, because the length of these sequences is the same and odd, there is a middle point at which $(x\ y)$=$(a\ c)$. So, one way of proving that at some point in the execution of the algorithm the vectors $(x\ y)$ and $(a\ c)$ are equal is

to prove that the sequences of successive values of the vectors $(x\ y)$ and $(a\ c)$, with the exception of the initial values, are reverses of each other and that both sequences have odd length. (In the example above, the length is 7.)

Taking this analysis into account, the question can be reformulated as: for which arguments m and n does Euclid's algorithm produce odd-length sequences of successive values of the vectors $(x\ y)$ and $(a\ c)$ that are reverse of each other?

Our answer to this question is divided in three parts. First, in section 3, we invert Euclid's algorithm to prove that the operations performed on the vector $(x\ y)$ are the same as those performed on the vector $(a\ c)$ when running the algorithm backwards. Second, in section 4, we determine necessary and sufficient conditions on the arguments m and n to make the initial value of the vector $(x\ y)$ equal the final value of the vector $(a\ c)$. These two parts together characterise the arguments for which the sequences of vectors are each other's reverses. Finally, in section 5, we show that if the sequences are the reverses of each other, they must have odd length.

Note that our investigation aims at expressing the argument m as a sum of two positive squares — that is why we focus on vectors $(x\ y)$ and $(a\ c)$. This means that, given a value m, we want to characterise which values n can be chosen to be passed along with m as arguments of the algorithm (we perform this characterisation in section 4).

For brevity, and whenever the context is unambiguous, we shall refer to "the sequences" to mean "the sequences of successive values of the vectors $(x\ y)$ and $(a\ c)$" and to "the sequences are reversed" to mean "the sequences are the reverses of each other". Also, we assume throughout that $\mathbf{D}{=}\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$.

3 Inverting Euclid's Algorithm

Inverting an algorithm S consists in finding another algorithm, usually denoted by S^{-1}, that when composed with S leaves the program state unchanged. In other words, executing S^{-1} after S amounts to doing nothing, that is, if we provide to S^{-1} some output of S, it will compute a corresponding input to S.

Some statements are easy to invert. The inverse of skip, for example, is skip itself. Also, the inverse of $x := x{-}y$ is $x := x{+}y$. However, other statements are difficult or impossible to invert. For example, we cannot invert $x := 1$ without knowing the value of x before the assignment; we can only invert it if we know the precondition. The inverse of

$$\{\ x{=}0\ \}\ \ x\ :=\ 1$$

is

$$\{\ x{=}1\ \}\ \ x\ :=\ 0\ \ .$$

Note that the assertion becomes an assignment and the assignment becomes an assertion. This simple example shows that we may be able to compute inverses only when the precondition is given. Therefore, we define the inverse of a statement with respect to a precondition. That is, S^{-1} is the (right) inverse of S with respect to R, if for every Q

$$\{\ R \wedge Q\ \}\quad S\,;S^{-1}\ \{\ Q\ \}\quad .$$

An important aspect of the above characterisation is that it distributes through program constructs. This allows us to reduce the inversion of a program into the inversion of its components. For example, the inverse of a sequence of commands is the reverse of the sequence of inverses of the individual commands:

$$(S_0;S_1;\cdots;S_n)^{-1}\ =\ S_n^{-1};\cdots;S_1^{-1};S_0^{-1}\quad .$$

Also, if c_0 and c_1 are constants, the inverse of

(1) $\quad v\ :=\ c_0\ \ ;\ \ S\ \ \{\ v = c_1\ \}$

is

$$v\ :=\ c_1\ \ ;\ \ S^{-1}\ \ \{\ v = c_0\ \}\quad .$$

In (1), variable v is initialised to a value c_0, S is executed, and upon termination v has the final value c_1. The inverse assigns c_1 to v, undoes what S did, and terminates with $v = c_0$. Note, again, how the assignment and the assertion switch places. Since Euclid's algorithm is an instance of (1) — instantiate v with the variables $(x\ y)$ and $\mathbf{D}$, and consider S to be the loop — its inverse is:

$$(x\ y)\ :=\ (m\triangledown n\ \ m\triangledown n)\ ;$$

initialise $\mathbf{D}$ such that $(m\triangledown n\ \ m\triangledown n)\times\mathbf{D}\ =\ (m\ n)\,;$

$$S^{-1}$$

$$\{\ (x\ y)\ =\ (m\ n)\qquad\wedge\qquad\mathbf{D}\ =\ \mathbf{I}\ \}\quad .$$

That is, provided that we initialise $(x\ y)$ to $(m\triangledown n\ \ m\triangledown n)$ and the matrix $\mathbf{D}$ in a way that satisfies $(m\triangledown n\ \ m\triangledown n)\times\mathbf{D}\ =\ (m\ n)$, undoing S terminates in a state where $(x\ y)$ and $\mathbf{D}$ equal their initial values in Euclid's algorithm. But we still have to guarantee that there is only one way of initialising $\mathbf{D}$. This is indeed the case, since

$$(m\triangledown n\ \ m\triangledown n)\times\mathbf{D}\ =\ (m\ n)$$

$=$

$$(1\ 1)\times\mathbf{D}\ =\ (m/(m\triangledown n)\ \ n/(m\triangledown n)),$$

where $(m/(m\triangledown n)\ \ n/(m\triangledown n))$ can be seen as a positive rational number in so-called lowest-form representation. We know from [12] (see also [13]) that there is a bijection between finite products of the matrices $\mathbf{L}$ and $\mathbf{R}$ and the positive rationals. Therefore, $\mathbf{D}$ (which is a finite product of $\mathbf{L}$s and $\mathbf{R}$s) is uniquely defined (more specifically, it represents the path from the origin to the rational $\frac{n/(m\triangledown n)}{m/(m\triangledown n)}$ in the Stern-Eisenstein tree of rationals).

Now, since the alternative statement in the loop of Euclid's algorithm is deterministic ($y < x$ and $x < y$ are mutually exclusive), we can use the inversion rule for deterministic alternative statements together with the inversion rule for iterative statements. Suppose we have the following loop

$\{\ G_0 \vee G_1\ \}$

do $G_0\ \rightarrow\ S_0\ \{\ C_0\ \}$

$\square\ \ G_1\ \rightarrow\ S_1\ \{\ C_1\ \}$

od

$\{\ C_0 \vee C_1\ \}$.

Execution of the loop must begin with one of the guards true, so the disjunction of the guards has been placed before the statement. Execution terminates with either C_0 or C_1 true, depending on which command is executed, so $C_0 \vee C_1$ is the postcondition. Also, to invert this loop we must know whether to perform the inverse of S_0 or to perform the inverse of S_1. Therefore, C_0 and C_1 cannot be true at the same time (i.e., $\neg(C_0 \wedge C_1)$). For symmetry, we also require $\neg(G_0 \wedge G_1)$.

Because the loop ends in a state satisfying $C_0 \vee C_1$, its inverse must begin in a state satisfying $C_0 \vee C_1$. Also, execution of $G_1 \rightarrow S_1\ \{\ C_1\ \}$ means that beginning with G_1 true, S_1 is executed, and C_1 is established. The inverse must express that beginning with C_1 true, S_1 is undone, and G_1 is established:

$C_1 \rightarrow S_1^{-1}\ \{\ G_1\ \}$.

Note how, when inverting a guarded command with a postcondition, the guard and postcondition switch places. Continuing to read backwards yields the inverse of the loop:

$\{\ C_0 \vee C_1\ \}$

do $C_1\ \rightarrow\ S_1^{-1}\ \{\ G_1\ \}$

$\square\ \ C_0\ \rightarrow\ S_0^{-1}\ \{\ G_0\ \}$

od

$\{\ G_0 \vee G_1\ \}$.

We now have to insert appropriate assertions in Euclid's algorithm so that the rules presented above can be used. Recall that, as explained in section 2, we want to ignore the initial values (in effect, this corresponds to ignoring the first row of table 1). This motivates moving the first step out of the loop body. Assuming that $n < m$, we can rewrite the algorithm as follows (note the new annotations and recall that $\mathbf{D} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$):

$\{\ 0 < n\ <\ m\ \}$

$(x\ y)\,,\mathbf{D}\ :=\ (m{-}n\ \ n)\,,\mathbf{L}$;

$\{$ **Invariant**: $\quad (x\ y) \times \mathbf{D}\ =\ (m\ n)\ \}$

$\{\ y < x\ \vee\ x < y\ \}$

do $y < x\ \ \rightarrow\ \ (x\ y)\,,\mathbf{D}\ :=\ (x\ y) \times \mathbf{L}^{-1}\ ,\ \mathbf{L}{\times}\mathbf{D}\ \ \{\ a < c\ \}$

$\square\ \ x < y\ \ \rightarrow\ \ (x\ y)\,,\mathbf{D}\ :=\ (x\ y) \times \mathbf{R}^{-1}\ ,\ \mathbf{R}{\times}\mathbf{D}\ \ \{\ c < a\ \}$

$$\begin{array}{l}
\mathsf{od} \\
\{\ a<c \ \vee\ c<a\ \} \\
\{\ (x\ y)\ =\ (m\triangledown n\ \ m\triangledown n) \quad \wedge \quad (m\triangledown n\ \ m\triangledown n)\times\mathbf{D}\ =\ (m\ n)\ \}
\end{array}$$

From this point on, whenever we refer to Euclid's algorithm, the intended reference is to this algorithm. The removal of the first step out of the loop body forces $n<m$ and $1<m$, but it allows us to include assertions after each assignment, making the inversion of the loop body a straightforward application of the rules mentioned above. (The new assertions after the assignments follow from the facts that premultiplying a matrix by $\mathbf{L}$ corresponds to adding the first row to the second, and premultiplying a matrix by $\mathbf{R}$ corresponds to adding the second row to the first.) Because the assignments in the loop body are easily inverted, the inverse of Euclid's algorithm becomes:

$$\begin{array}{l}
\{\ 0<n\ <\ m\ \} \\
(x\ y)\ :=\ (m\triangledown n\ \ m\triangledown n)\,; \\
\text{initialise } \mathbf{D} \text{ such that } (m\triangledown n\ \ m\triangledown n)\times\mathbf{D}\ =\ (m\ n)\,; \\
\{\ \textbf{Invariant}\colon\quad (x\ y)\times\mathbf{D}\ \ =\ \ (m\ n)\ \} \\
\{\ a<c\ \vee\ c<a\ \} \\
\mathsf{do}\ \ a<c\ \ \rightarrow\ \ (x\ y)\,,\mathbf{D}\ :=\ (x\ y)\times\mathbf{L}\ ,\ \mathbf{L}^{-1}\times\mathbf{D}\ \ \{\ y<x\ \} \\
\Box\ \ c<a\ \ \rightarrow\ \ (x\ y)\,,\mathbf{D}\ :=\ (x\ y)\times\mathbf{R}\ ,\ \mathbf{R}^{-1}\times\mathbf{D}\ \ \{\ x<y\ \} \\
\mathsf{od} \\
\{\ y<x\ \vee\ x<y\ \} \\
\{\ (x\ y)\ =\ (m{-}n\ \ n)\wedge\mathbf{D}=\mathbf{L}\ \}
\end{array}$$

Comparing the two algorithms, we see that the assignments to $(x\ y)$ and to $(a\ c)$ are interchanged: in the original algorithm we have

$$\begin{array}{ll}
 & y<x\ \ \rightarrow\ \ (x\ y)\ :=\ (x{-}y\ \ y) \\
\Box & x<y\ \ \rightarrow\ \ (x\ y)\ :=\ (x\ \ y{-}x)\ \ ,
\end{array}$$

and in the inverted algorithm we have

$$\begin{array}{ll}
 & a<c\ \ \rightarrow\ \ (a\ c)\ :=\ (a\ \ c{-}a) \\
\Box & c<a\ \ \rightarrow\ \ (a\ c)\ :=\ (a{-}c\ \ c)\ \ .
\end{array}$$

(We leave the reader to check the matrix arithmetic.) In other words, the inverse of Euclid's algorithm is Euclid's algorithm itself, but on different variables: the inverted version computes the greatest common divisor using the variables a and c. This means that to make the sequences of successive values of the vectors $(x\ y)$ and $(a\ c)$ the reverse of each other, we only need to guarantee that the initial value of $(x\ y)$ in the non-inverted algorithm is the same as the initial value of

$(a\ c)$ in the inverted one. In other words, we need to guarantee that in Euclid's algorithm, the initial value of $(x\ y)$ is the same as the final value of $(a\ c)$.

The initial assignments of the inverted algorithm may seem strange at first sight, but the important fact to retain is that if we compose both algorithms, the program state remains unchanged. The inversion of the algorithm serves only as a formal proof that the process applied to $(x\ y)$ in one direction is the same as the one applied to $(a\ c)$ in the opposite direction. In the remainder of our investigation, we base our discussion on Euclid's algorithm, i.e., on the non-inverted version.

For more details on the inversion rules shown in this section, we recommend the expositions in [14, chapter 21] and [15, chapter 11]. As far as we know, the technique of program inversion first appeared in [16, pp. 351–354] and, since then, it has been mentioned and used in many places (see, for example, [17–20]).

4 Reversed Sequences of Vectors

Given the result of the previous section, saying that the sequences of vectors $(x\ y)$ and $(a\ c)$ are reversed is equivalent to saying that the initial value of $(a\ c)$ is equal to the final value of $(x\ y)$ and the initial value of $(x\ y)$ is equal to the final value of $(a\ c)$.

Looking at the algorithm, we see that the initial value of $(a\ c)$ is $(1\ 1)$ and the final value of $(x\ y)$ is $(m \bigtriangledown n\ \ m \bigtriangledown n)$. So, for the sequences to be reversed, $m \bigtriangledown n$ has to be 1, i.e., m and n have to be co-prime. We thus assume henceforth that $m \bigtriangledown n = 1$.

Also, the initial value of $(x\ y)$ is $(m{-}n\ \ n)$. So, because $m \bigtriangledown n = 1$, we have the following equality:

"The sequences are reversed"

=

"The final value of $(a\ c)$ is $(m{-}n\ \ n)$" .

We can rewrite this equality in terms of matrix $\mathbf{D}$:

"The sequences are reversed"

=

"The final value of $\mathbf{D}$ is $\begin{pmatrix} m-n & b \\ n & d \end{pmatrix}$ for some b and d" .

Now, because $\mathbf{D}$ is the product of matrices whose determinant equals 1, its determinant also equals 1; this allows us to calculate b and d:

$$
\begin{array}{ll}
 & det.\mathbf{D} = 1 \\
= & \quad \{ \quad \mathbf{D} \text{ has the shape } \begin{pmatrix} m-n & b \\ n & d \end{pmatrix} \quad \} \\
 & (m{-}n){\times}d - n{\times}b \ = \ 1
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \quad \text{arithmetic} \quad \} \\
& m\times d - n\times(d+b) \;=\; 1 \\
= & \{ \quad \text{we have assumed that } m\triangledown n = 1, \text{ so, on termination,} \\
& \quad \text{the invariant states that } (1\ 1)\times \mathbf{D} \;=\; (m\ n); \\
& \quad \text{this means that } n = b+d \quad \} \\
& m\times d = n^2+1 \\
= & \{ \quad 0<m \quad \} \\
& d = \dfrac{n^2+1}{m} \quad .
\end{array}
$$

The value of b is simply $n-d$, since on termination we have $n = b+d$ (it follows from the invariant). Because $\mathbf{D}$ is a matrix of integer values, d has to be an integer, and so, a necessary condition is that $m\backslash(n^2+1)$, that is, $n^2 \cong -1 \pmod m$. (We write $m\backslash n$ to denote that m is a divisor of n. Although the notation $m|n$ is more common, we prefer to use an asymmetric symbol such as the backward slash to denote an asymmetric relation. Moreover, as the authors of [21, p. 102] point out, vertical bars are overused and $m\backslash n$ gives an impression that m is the denominator of an implied ratio. Also, $a \cong b \pmod m$ means that $m\backslash(a-b)$ and we read it as "a and b are congruent modulo m".) We can thus conclude that

$$n^2 \;\cong\; -1 \pmod m \;\Leftarrow\; \text{“The sequences are reversed”} \quad .$$

A question that now arises is whether $n^2 \cong -1 \pmod m$ is a sufficient condition for the sequences to be reversed. That is, can we prove

$$(2) \quad \text{“The final value of } \mathbf{D} \text{ is } \begin{pmatrix} m-n & n-{}^{(n^2+1)}\!/_m \\ n & {}^{(n^2+1)}\!/_m \end{pmatrix}\text{”} \;\Leftarrow\; n^2 \cong -1 \pmod m \,?$$

Using the assumption that $\mathbf{D} = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, we can simplify (2) to:

$$(3) \quad \text{“The final value of } c \text{ is } n\text{”} \quad \Leftarrow \quad n^2 \;\cong\; -1 \pmod m \quad ,$$

since c uniquely determines all the other entries (recall that $m = a+c$, $n = b+d$ and $det.\mathbf{D} = 1$). To prove (3), we first show that $n \cong c \pmod m$ follows from $n^2 \cong -1 \pmod m$ and then we use the range of n and c to conclude that $n = c$. The following lemma is used to prove that $n \cong c \pmod m$.

Lemma 1. For all integers m, n, and c, the following holds:

$$n \cong c \pmod m \;\Leftarrow\; n^2 \;\cong\; -1 \pmod m \;\wedge\; n\times c \;\cong\; -1 \pmod m \quad .$$

Proof Using the fact that, for all integers a, b, and c, the following law on congruences holds

$$(4) \quad a-c \;\cong\; b-d \pmod m \Leftarrow a \cong b \pmod m \wedge c \cong d \pmod m \quad ,$$

we can prove the lemma as follows:

$$
\begin{array}{ll}
 & n \cong c \ (\mathrm{mod}\ m) \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & n - c \cong 0 \ \ (\mathrm{mod}\ m) \\
\Leftarrow & \quad \{ \quad m \bigtriangledown n = 1 \text{ and Euclid's lemma; see below for details} \quad \} \\
 & n \times (n - c) \cong 0 \ \ (\mathrm{mod}\ m) \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & n^2 - n \times c \cong 0 \ \ (\mathrm{mod}\ m) \\
= & \quad \{ \quad n^2 \cong -1 \ \ (\mathrm{mod}\ m) \text{ and } n \times c \cong -1 \ \ (\mathrm{mod}\ m) \text{ and } (4) \quad \} \\
 & \mathsf{true} \ .
\end{array}
$$

In the second step we can safely assume that $m \bigtriangledown n = 1$, since it follows from the congruence $n^2 \cong -1 \ \ (\mathrm{mod}\ m)$. A short proof of this fact is:

$$
\begin{array}{ll}
 & n^2 \cong -1 \ \ (\mathrm{mod}\ m) \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \langle \exists q ::\ n^2 + 1 = q \times m \rangle \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & \langle \exists q ::\ 1 \ = \ q \times m - n \times n \rangle \\
\Rightarrow & \quad \{ \quad (m \bigtriangledown n) \backslash (q \times m - n \times n), \text{ so } (m \bigtriangledown n) \backslash 1; \\
 & \qquad \text{division is anti-symmetric} \quad \} \\
 & m \bigtriangledown n = 1 \ \ .
\end{array}
$$

Also, Euclid's lemma states that for all integers a, b, and c:

$$a \backslash c \quad \Leftarrow \quad a \setminus b \times c \ \wedge \ a \bigtriangledown b = 1 \quad .$$

□

Now, if, on termination, we have that $n \times c \cong -1 \ \ (\mathrm{mod}\ m)$, we can use lemma 1 to conclude that, on termination, we also have that $n \cong c \ (\mathrm{mod}\ m)$ follows from $n^2 \cong -1 \ \ (\mathrm{mod}\ m)$. Recall that an invariant of the algorithm is

$$(x \ y) \times \mathbf{D} \quad = \quad (m \ n) \quad = \quad (x \times a + y \times c \quad x \times b + y \times d) \quad .$$

Because the determinant of $\mathbf{D}$ equals 1, the inverse of $\mathbf{D}$ is $\left(\begin{smallmatrix} d & -b \\ -c & a \end{smallmatrix} \right)$, making the following property also invariant:

$$(5) \quad (x \ y) \quad = \quad (m \ n) \times \mathbf{D}^{-1} \quad = \quad (m \times d - n \times c \quad a \times n - b \times m) \quad .$$

It follows that on termination, when $(x \ y) = (1 \ 1)$, we have that $n \times c \cong -1 \ \ (\mathrm{mod}\ m)$, as the following calculation shows:

$$
\begin{array}{ll}
 & n{\times}c \cong -1 \pmod{m} \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & m \backslash (n{\times}c + 1) \\
\Leftarrow & \quad \{ \quad \text{division properties} \quad \} \\
 & m{\times}d = n{\times}c + 1 \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & m{\times}d - n{\times}c = 1 \\
= & \quad \{ \quad \text{invariant (5), } (x \ y) = (1 \ 1) \text{ on termination} \quad \} \\
 & \mathsf{true} \ .
\end{array}
$$

By lemma 1, we deduce that on termination $n \cong c \pmod{m}$ follows from $n^2 \cong -1 \pmod{m}$. Finally, because $0 < a$ and $m = a + c$ we have that $0 < c < m$; this allows us to conclude that $n = c$:

$$
\begin{array}{ll}
 & n \cong c \pmod{m} \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & m \backslash (n - c) \\
= & \quad \{ \quad 0 < n < m \text{ and } 0 < c < m \text{ imply that } -m < n - c < m; \\
 & \qquad \text{the only multiple of } m \text{ in that range is } 0 \quad \} \\
 & n - c = 0 \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & n = c \ .
\end{array}
$$

The conclusion is that $n^2 \cong -1 \pmod{m}$ is also a sufficient condition for the sequences to be reversed, leading to the equality:

$$
\begin{array}{ll}
 & \text{“The sequences are reversed”} \\
= & \\
 & n^2 \cong -1 \pmod{m} \ .
\end{array}
$$

To summarise, in the following algorithm

$$
\begin{array}{l}
\{ \ 0 < n \ < \ m \ \} \\
(x \ y) , \mathbf{D} \ := \ (m{-}n \ \ n) , \mathbf{L} \ ; \\
\{ \ \textbf{Invariant}: \quad (m \ n) = (x \ y) \times \mathbf{D} = (x{\times}a + y{\times}c \ \ \ x{\times}b + y{\times}d) \\
\quad \wedge \qquad\qquad\quad (m \ n) \times \mathbf{D}^{-1} = (x \ y) = (m{\times}d - n{\times}c \ \ \ a{\times}n - b{\times}m) \ \} \\
\mathsf{do} \ \ y < x \ \ \rightarrow \ \ (x \ y) , \mathbf{D} \ := \ (x \ y) \times \mathbf{L}^{-1} \ , \ \mathbf{L}{\times}\mathbf{D} \ \ \{ \ a < c \ \} \\
\Box \ \ x < y \ \ \rightarrow \ \ (x \ y) , \mathbf{D} \ := \ (x \ y) \times \mathbf{R}^{-1} \ , \ \mathbf{R}{\times}\mathbf{D} \ \ \{ \ c < a \ \}
\end{array}
$$

od

$\{\ (x\ y)\ =\ (1\ 1)\quad \wedge\quad (\ m\ n\ )\ =\ (1\ 1)\times \mathbf{D}\ \}$,

the sequences of vectors $(x\ y)$ and $(a\ c)$ are reverses of each other exactly when $n^2 \cong -1 \pmod m$.

5 Length of the Sequence of Vectors

We now have to prove that the final value of matrix $\mathbf{D}$ is decomposed into an odd-length product of the matrices $\mathbf{L}$ and $\mathbf{R}$. However, because $\mathbf{D}$ is initially $\mathbf{L}$ and because it is iteratively premultiplied, $\mathbf{D} = \mathbf{M}\times\mathbf{L}$ for some $\mathbf{M}$. So we can alternatively prove that $\mathbf{M}$ is decomposed into an even-length product of the matrices $\mathbf{L}$ and $\mathbf{R}$. Observing that

$$\mathbf{M} = \mathbf{D}\times\mathbf{L}^{-1} = \begin{pmatrix} m-(2\times n - {}^{(n^2+1)}\!/_m) & n-{}^{(n^2+1)}\!/_m \\ n-{}^{(n^2+1)}\!/_m & {}^{(n^2+1)}\!/_m \end{pmatrix} \quad ,$$

we see that $\mathbf{M}$ has the top-right and bottom-left corners equal, which means that $\mathbf{M} = \mathbf{M}^T$ ($\mathbf{M}$ equals the transpose of $\mathbf{M}$). We also know that $\mathbf{R} = \mathbf{L}^T$ and $\mathbf{L} = \mathbf{R}^T$.

There are also two functions from finite products of $\mathbf{L}$ and $\mathbf{R}$ to naturals, $\#\mathbf{L}$ and $\#\mathbf{R}$, that give, respectively, the number of $\mathbf{L}$s and the number of $\mathbf{R}$s in the decomposition of their argument[2]. Now, a fundamental property is that $\#\mathbf{L}.\mathbf{M} = \#\mathbf{R}.\mathbf{M}^T$, whenever $\mathbf{M}$ is a product of $\mathbf{L}$s and $\mathbf{R}$s. This fundamental property means that the number of $\mathbf{L}$s in the decomposition of $\mathbf{M}$ equals the numbers of $\mathbf{R}$s in the decomposition of $\mathbf{M}^T$, which is easy to see because $\mathbf{R} = \mathbf{L}^T$ and $\mathbf{L} = \mathbf{R}^T$. Using these observations, a simple calculation showing that the length of $\mathbf{M}$ is an even number is:

length.$\mathbf{M}$
$=$ { $\mathbf{M}$ is a product of $\mathbf{L}$s and $\mathbf{R}$s }
$\#\mathbf{L}.\mathbf{M} + \#\mathbf{R}.\mathbf{M}$
$=$ { $\#\mathbf{L}.\mathbf{M} = \#\mathbf{R}.\mathbf{M}^T$ }
$\#\mathbf{R}.\mathbf{M}^T + \#\mathbf{R}.\mathbf{M}$
$=$ { $\mathbf{M}^T = \mathbf{M}$ }
$2 \times \#\mathbf{R}.\mathbf{M}$.

Hence, the length of $\mathbf{M}$ is an even number. Subsequently, the length of the final value of $\mathbf{D}$ is odd.

[2] Note that, given that we can easily provide algorithms that compute them, functions length, $\#\mathbf{L}$, and $\#\mathbf{R}$ are well-defined. As proved in [13] and [12], there is a bijection between finite products of matrices $\mathbf{L}$ and $\mathbf{R}$, and binary strings made of the symbols L and R; defining these functions in the realm of strings is easy.

6 Sum of two positive squares

In the above sections we have proved the following theorem:

Theorem 1. A number m at least 2 can be written as the sum of two positive squares if there is a number n such that $0 < n < m$ and $n^2 \cong -1 \pmod{m}$.
□

The argument we provide is constructive because we show how to use Euclid's algorithm to represent a number as the sum of two positive squares. Indeed we can extend Euclid's algorithm so that it expresses a given number m as the sum of two positive squares:

$$\{\ 1 < m\ \wedge\ \langle \exists n\ :\ 0 < n\ <\ m\ :\ n^2 \cong -1\ \pmod{m} \rangle\ \}$$

• Find a number n such that $0 < n\ <\ m$ and $n^2 \cong -1\ \pmod{m}$;

$$\{\ 0 < n\ <\ m\ \wedge\ n^2 \cong -1\ \pmod{m}\ \}$$

$$(x\ y)\,,\mathbf{D}\ :=\ (m{-}n\ \ n)\,,\mathbf{L}\ ;$$

$$\{\ \textbf{Invariant}\colon\quad (x\ y) \times \mathbf{D} = (m\ n) = (x{\times}a + y{\times}c\quad x{\times}b + y{\times}d)\ \}$$

do $(x\ y) \neq (a\ c)\ \rightarrow$

$$\begin{array}{rcl} y < x & \rightarrow & (x\ y)\,,\mathbf{D}\ :=\ (x\ y) \times \mathbf{L}^{-1}\ ,\ \mathbf{L}{\times}\mathbf{D} \\ \square\ x < y & \rightarrow & (x\ y)\,,\mathbf{D}\ :=\ (x\ y) \times \mathbf{R}^{-1}\ ,\ \mathbf{R}{\times}\mathbf{D} \end{array}$$

od

$$\{\ (x\ y) = (a\ c)\ \wedge\ m = x^2{+}y^2 = a^2{+}c^2\ \ \}$$

Theorem 1 is more general than Girard's result: while Girard's theorem is only on odd prime numbers, theorem 1 concerns all positive integers at least 2. As an example, we can say that the number 10 is expressible as the sum of two positive squares, since $3^2 \cong -1 \pmod{10}$ (and, in fact, we have that $10 = 3^2{+}1^2$). Moreover, given the following lemma (see [9, p. 17, Lemma 1]), Girard's result is an immediate corollary of theorem 1.

Lemma 2. For primes $p = 4k + 1$ the equation $s^2 \cong -1 \pmod{p}$ has two solutions $s \in \{1\,..\,p{-}1\}$, for $p = 2$ there is only one such solution, while for primes of the form $p = 4k + 3$ there is no solution.
□

Although we believe that theorem 1 may be known by some number-theorists, we have not found it in the literature.

Please note that developing an algorithm to find a number n such that $0 < n\ <\ m$ and $n^2 \cong -1\ \pmod{m}$ is beyond the scope of this paper. For more details on this topic, we recommend [11] and [22], where the authors discuss different algorithms that can be used to find such a number n.

Finally, the algorithm shown above can be generalised. In a recent private communication, Wagon told us that the method of using Euclid's algorithm to write a number as a sum of two squares (or, more generally, as $a^2 + g{\times}c^2$) is

known as the Smith-Cornacchia algorithm (he referred us to [22] and [23]). Also, in [24], Hardy, Muskat, and Williams show a more general algorithm for solving $m = f \times a^2 + g \times c^2$ in coprime integers a and c. The algorithm presented in this paper treats the case $f = g = 1$. At the moment, we do not know how to adapt it to solve the more general problem. Recall that we have started our argument by observing that if, at any point in the execution of the algorithm, $(x\ y)$ equals $(a\ c)$, it follows from the invariant

$$(m\ n) \quad = \quad (x\ y) \times \mathbf{D} \quad = \quad (x \times a + y \times c \quad x \times b + y \times d)$$

that m can be written as a sum of two positive squares. To solve the general problem, we have to investigate when it is possible to have, at any point in the execution of the algorithm, $a \backslash x$ and $c \backslash y$. If this happens, that is, if there are two integers f and g such that $x = f \times a$ and $y = g \times c$, it follows from the invariant that $m = f \times a^2 + g \times c^2$:

$$(m\ n) \quad = \quad (f \times a \quad g \times c) \times \mathbf{D} \quad = \quad (f \times a^2 + g \times c^2 \quad f \times a \times b + g \times c \times d) \quad .$$

7 Discussion

This paper shows a new and constructive proof of the two-squares theorem based on a somewhat unusual, but very effective, way of rewriting the so-called extended Euclid's algorithm. As mentioned in the introduction, the use of Euclid's algorithm to prove the theorem is not new: Brillhart [6] and Wagon [11] have used it to *verify* the theorem. Effectively, given the close relationship between Euclid's algorithm and continued fractions, we can say that Serret [5] and Hermite [4] were the first to provide the germ of the essential idea presented here (in fact, Brillhart's note is described as an improvement on Hermite's method: in using Euclid's algorithm, Brillhart avoids the calculation of the convergents arising in the continued fractions).

The novel contribution of this paper is the use of the algorithm to *investigate* which numbers can be written as the sum of two positive squares. The precise formulation of the problem as an algorithmic problem is the key, since it allows us to use algorithmic techniques and to avoid guessing. The notion of invariance, in particular, plays a central role in our development: it is used initially to observe that Euclid's algorithm can actually be used to represent a given number as a sum of two positive squares, and then it is used throughout the argument to prove relevant properties. Also, section 3 is an example of how the use of program inversion can make our arguments more precise.

We conclude by mentioning that this paper is part of a larger endeavour which aims at reinvigorating mathematics education by exploiting mathematics' algorithmic nature [13, 12, 25]. In our view, the combination of practicality with mathematical elegance that arises from an adequate focus on the algorithmic content of mathematics can enrich and improve, not only mathematics education, but also the process of constructing computer programs. Moreover, the emphasis on investigation and construction rather than verification brings tremendous benefits. As Leibniz once put it:

Nothing is more important than to see the sources of invention which are, in my opinion, more interesting than the inventions themselves.

Acknowledgements I would like to thank Roland Backhouse for helping me with the initial analysis of the problem, for suggesting the use of program inversion techniques, and for his comments on earlier drafts of this paper. Also, thanks to Alexandra Mendes for helping me simplifying section 5 and to Wei Chen and Shin-Cheng Mu for reading and checking earlier drafts of the paper. I would also like to thank Stan Wagon for sending me references on the Smith-Cornacchia algorithm and the anonymous referees, whose comments and corrections have led to significant improvements.

This work was developed in the context of the MATHIS project, which is supported by Fundação para a Ciência e a Tecnologia (Portugal) under contract PTDC/EIA/73252/2006.

References

1. Dickson, L.E.: History of the Theory of Numbers: Diophantine Analysis: Diophantine Analysis Vol 2 (AMS/Chelsea Publication). American Mathematical Society (August 1999)
2. Bell, E.T.: Men of Mathematics - The Lives and Achievements of the Great Mathematicians from Zeno to Poincaré. Touchstone (July 2008)
3. Gauss, C.F.: Disquisitiones arithmeticae. (1801)
4. Hermite: Note au sujet de l'article précédent. Journal de Mathématiques Pures et Appliquées **13** (1848) 15
5. Serret, J.A.: Sur un théorème relatif aux nombres entiers. Journal de Mathématiques Pures et Appliquées **13** (1848) 12–14
6. Brillhart, J.: Note on representing a prime as a sum of two squares. Mathematics of Computation **26**(120) (1972) 1011–1013
7. Clarke, F.W., Everitt, W.N., Littlejohn, L.L., Vorster, S.J.R.: H. J. S. Smith and the Fermat two squares theorem. The American Mathematical Monthly **106**(7) (1999) 652–665
8. Zagier, D.: A one-sentence proof that every prime $p \equiv 1 (mod\ 4)$ is a sum of two squares. The American Mathematical Monthly **97**(2) (1990) 144+
9. Aigner, M., Ziegler, G.: Proofs From The Book, 3rd Edition. Springer-Verlag (2004)
10. Dijkstra, E.W.: A derivation of a proof by D. Zagier. (August 1993)
11. Wagon, S.: Editor's corner: The Euclidean algorithm strikes again. The American Mathematical Monthly **97**(2) (1990) 125–129
12. Backhouse, R., Ferreira, J.F.: On Euclid's algorithm and elementary number theory. To appear in the journal *Science of Computer Programming*. Available from http://joaoff.com/publications/2009/euclid-alg/ (2010)
13. Backhouse, R., Ferreira, J.F.: Recounting the rationals: Twice! In: Mathematics of Program Construction. Volume 5133 of LNCS., Springer-Verlag (2008) 79–91
14. Gries, D.: The Science of Programming. Springer-Verlag (1981)
15. van de Snepscheut, J.L.: What Computing Is All About. Springer-Verlag New York Inc. (1993)

16. Dijkstra, E.W.: Program inversion. In: Selected Writings on Computing: A Personal Perspective. Springer-Verlag (1982) 351–354
17. van de Snepscheut, J.L.A.: Inversion of a recursive tree traversal. Inf. Process. Lett. **39**(5) (1991) 265–267
18. Chen, W.: A formal approach to program inversion. In: CSC '90: Proceedings of the 1990 ACM annual conference on Cooperation, ACM Press (1990) 398–403
19. von Wright, J.: Program inversion in the refinement calculus. Inf. Process. Lett. **37**(2) (1991) 95–100
20. Mu, S.C., Bird, R.: Rebuilding a tree from its traversals: A case study of program inversion. In: Programming Languages and Systems. Volume 2895 of LNCS. (2003) 265–282
21. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics : a Foundation for Computer Science. second edn. Addison-Wesley Publishing Company (1994)
22. Buhler, J., Wagon, S.: Basic algorithms in number theory. In Buhler, J.P., Stevenhagen, P., eds.: Algorithmic Number Theory. Lattices, Number Fields, Curves and Cryptography. Cambridge University Press (2008) 25–68
23. Cornacchia, G.: Su di un metodo per la risoluzione in numeri interi dell'equazione $\sum_{h=0}^{n} C_h x^{n-h} y^h = P$. Giornale di Matematiche di Battaglini **46** (1908) 33–90
24. Hardy, K., Muskat, J.B., Williams, K.S.: A deterministic algorithm for solving $n = fu^2 + gv^2$ in coprime integers u and v. **55**(191) (July 1990) 327–343
25. Ferreira, J.F., Mendes, A., Backhouse, R., Barbosa, L.S.: Which mathematics for the information society? In: Teaching Formal Methods. Volume 5846 of LNCS. Springer-Verlag (2009) 39–56