Euclid Writes an Algorithm: A Fairytale

Translated from an anonymous Greek manuscript by
Leslie Lamport, Microsoft Research

Illustrated by Steve D. K. Richards
http://web.me.com/stevedeuce

6 June 2011

Prepared for a Festschrift celebrating the 60th birthday of Manfred Broy.

Abstract

How Euclid might have written and checked the correctness of his famous algorithm had he been a little further ahead of his time.

Euclid of Alexandria was thinking. A traveler had that morning told him of a marvelous thing he had learned many years earlier from Eudoxus of Cnidus: a marvelous way to find the greatest common divisor of two numbers. But Eudoxus had not revealed the proof, and Euclid was wondering if it really was correct. He welcomed the distraction. Writing his *Elements* had become wearisome, and he was not sure he would ever finish.

As Euclid idly scribbled in the sand, there suddenly appeared before him a beautiful woman wearing a diaphanous gown.

"Who are you that disturbs my thoughts?" he asked.

"I am a Fairy Godmother, but my friends call me FG."

"Wherefore have you appeared so abruptly?"

"I am from 2300 years in the future, and I have come to take you there so you can write this algorithm that so intrigues you as it will then be written."

"What is an algorithm?"

"There will be time to explain such things during our voyage. Come."

FG led Euclid into the temporal wormhole from which she had emerged, and they began their journey. As they traveled, she explained to Euclid what someone in the twenty-first century A.D. needed to know, including computers, programs, and mortgage-backed securities (which, in truth, Euclid only pretended to understand).

After what seemed like a few weeks, FG said, "We have arrived in the year people now call 2010."

"Why chose you this year?"

"It is the sixtieth anniversary of the birth of the computer scientist Professor Manfred Broy. In celebration, people are bringing him gifts. I have taken you here to write this algorithm so we can present it to him."

"He must be a wise and beloved man."

"Indeed he is. But there is no time now to speak of him. We must be off."

FG led him to a shopping mall where she bought a brand-new computer. "Let us now take it to my place and set it up," she said.

FG spent two hours setting up the computer, occasionally uttering expletives that were not Greek to Euclid.

Euclid was puzzled. Finally, he said: "You have such wonderful devices. Why are they so hard to use?"

FG replied, "We have made our computers so complicated that we can no longer understand them. We now suffer the consequences."

Euclid found this strange. But he was eager to begin writing his algorithm, so he let it pass. "How do I begin?" he asked.

There suddenly appeared before him a beautiful woman wearing a diaphanous gown.

```
MODULE TheAlgorithm
EXTENDS Integers

CONSTANTS M, N

(**********
  --algorithm ComputeGCD
  { variables x = M,  y = N;
    { while (x ≠ y) { if (x < y) { y := y − x }
                      else       { x := x − y }  } } }
**********)
(* BEGIN TRANSLATION *)
(* END TRANSLATION   *)
```

Figure 1: Euclid's first attempt.

"You must first choose a programming language in which to write the algorithm."

"But I am writing an algorithm, not a program."

FG didn't know how to respond to that. So, they searched the Web. (Euclid read only the Greek of his day, and FG had to translate everything they found.) They discovered something named PlusCal that was said to be an algorithm language. From the documentation[1], it looked like a programming language. Since it was much simpler than any other (executable) programming language FG had seen, Euclid decided to give it a try.

To use PlusCal, the documentation said they should download the TLA$^+$ Toolbox, which is an IDE for TLA$^+$. Euclid didn't understand why people used such incomprehensible strings of letters as IDE, TLA, and CIA.[2] He decided it was because of the strange custom, mentioned in passing by FG, of feeding their children soup made from the alphabet.

FG and Euclid downloaded the Toolbox. With the aid of the documentation, some trial and error, and much help from FG, Euclid succeeded in writing the algorithm shown in Figure 1.[3] (FG explained to him modern mathematical notation, like the = symbol, and many modern concepts that the documentation assumed.)

The algorithm Euclid wrote is meant to compute the greatest common divisor (gcd) of two numbers M and N. When it terminates, x and y should

[1] Small superscript numbers, like the one above that led you to read this footnote, refer to translator notes at the end.

With much help from FG, Euclid succeeded in writing the algorithm.

both equal that gcd. The EXTENDS statement imports the standard TLA$^+$ module *Integers* that defines the usual operations of arithmetic.

Following the directions, Euclid then had the Toolbox translate the algorithm. This inserted many non-Greek symbols that he didn't understand between the `BEGIN TRANSLATION` and `END TRANSLATION` comments. He ignored them, being in a hurry to try out the algorithm.

FG had explained model checking to him, so Euclid clicked on the Toolbox command to create a new model to check. To create the model, he had to choose specific numbers to substitute for M and N. Euclid chose 72 and 48. He then ran the TLC model checker on the model, which quickly informed him that no errors were detected. That wasn't surprising, since it wasn't checking anything.

Euclid wanted to check that the algorithm really computed the gcd of M and N. To do that, he first had to define the gcd. The gcd of two numbers is the largest number that divides both of them. FG said that he needed to

write his definition in terms of simple logic and set theory.

Euclid was familiar with logic from studying Aristotle, and he found the concept of a set to be obvious when FG explained it to him. He decided to define:

$p \mid q$ to be true if and only if the number p divides the number q.

$Divisors(q)$ to be the set of divisors of the number q

$Maximum(S)$ to be the maximum of the set of numbers S.

$GCD(p, q)$ to be the gcd of the numbers p and q.

Euclid wrote his definitions as follows, in what he learned was a language called TLA$^+$, putting them immediately after the CONSTANTS declaration. (By *number*, Euclid meant *positive integer*, and he had found that $..$ is defined in the *Integers* module so $1 \,..\, q$ is the set of numbers from 1 through q.)[4]

$$
\begin{array}{ll}
p \mid q & \triangleq \ \exists\, d \in 1 \,..\, q \ : \ q = p * d \\
Divisors(q) & \triangleq \ \{d \in 1 \,..\, q \ : \ d \mid q\} \\
Maximum(S) & \triangleq \ \text{CHOOSE } x \in S \ : \ \forall\, y \in S \ : \ x \geq y \\
GCD(p, q) & \triangleq \ Maximum(Divisors(p) \cap Divisors(q))
\end{array}
$$

Having defined GCD, Euclid modified his PlusCal code by adding the following statement after the **while** loop:

assert $x = GCD(M, N) \ \wedge \ y = GCD(M, N)$

This would cause the TLC model checker to report an error if, after exiting from the **while** loop, x and y were not both equal to $GCD(M, N)$. TLC did not report any error.

Euclid was now convinced that the algorithm worked for $M = 72$ and $N = 48$. He could check it for other values of M and N by editing the model, but that would be pretty tedious. Surely there must be a better way.

Reading more about PlusCal, Euclid found that, for any numbers M and N, he could easily check that the algorithm computed the correct gcd of all numbers x from 1 through M and y from 1 through N. To do this, he first changed the declarations of the variables so x initially equals any number from 1 through M and y initially equals any number from 1 through N. He next added two variables $x0$ and $y0$ to "remember" the initial values of x and y. This produced the following variable-declaration statement:

variables $x \in 1 \,..\, M, \ y \in 1 \,..\, N, \ x0 = x, \ y0 = y;$

He then changed the **assert** statement after the **while** loop to:

assert $x = GCD(x0, y0) \ \wedge \ y = GCD(x0, y0)$

After translating this new algorithm, running TLC on the same model ($M = 72$ and $N = 48$) took a little over a minute to check all executions with those initial values of x and y. It found no errors.

Euclid was now pretty confident that the algorithm would not compute an incorrect gcd. However, what if it failed to compute the gcd of some pair of numbers because it never finished, staying forever in the **while** loop? The **assert** statement could not catch the error because that statement would never be reached in such an execution.

In the Toolbox, Euclid saw a property labeled *Termination* that he could select for checking. He was sure this would show whether the algorithm finished in all those executions. But FG said that his time in the future was limited, so he had best stick to checking only that the algorithm produced the correct answer if it stopped.

Euclid reluctantly agreed. He would verify only this property of the algorithm. He was quite sure that the algorithm did satisfy it, since TLC had checked it for those 3456 pairs of initial values of x and y. But Euclid was a mathematician. It didn't matter how many particular values he checked in this way; he wanted a proof.

How could he write a proof? Proof means mathematics, but the algorithm wasn't mathematics. It was written in the strange PlusCal language that he didn't completely understand. Exactly what did **while** and **if** and that funny symbol $:=$ mean?

Euclid realized that the answer lay in those symbols that the translator had inserted between the `BEGIN TRANSLATION` and `END TRANSLATION` comments. So, he sat down and began studying them. And studying them. He spent many hours examining the translations of the two versions of the algorithm he had written.

Finally, he shouted "Eureka!" (This was his favorite exclamation; it was copied by his admirer Archimedes of Syracuse.) What Euclid had found was that an execution of an algorithm is a sequence of states, where a state is an assignment of values to variables. The algorithm's executions are described by two formulas:

- An *initial predicate* that specifies possible initial states. It contains the algorithm's variables. A possible initial state is an assignment of values to variables that satisfies this formula.

- A *next-state relation* that specifies all possible steps from one state to the next. It contains the algorithm's variables, some occurrences of which are primed. A step from a state s to a state t is possible if the formula is satisfied when the unprimed variables are assigned their values in s and the primed variables are assigned their values in t.

There were some other formulas in the PlusCal translation that Euclid didn't understand. They contained the symbol $\Box$ and some subscripts. These formulas didn't seem important, so he ignored them.

The idea of describing the algorithm by an initial predicate and a next-state action was wonderfully simple. They were mathematical formulas, and he could prove things about mathematical formulas. But he found the formulas produced by the translator to be more complicated than they had to be. They contained an extra variable *pc* that had the value "Lbl_1" while the algorithm was executing the **while** loop and the value "Done" after it had finished. Moreover, in a state with *pc* equal to "Done", the next-state relation allowed steps in which no variables changed.

Euclid could describe this algorithm with simpler formulas. There was no need for PlusCal and its extra variable *pc*. He could write his own initial predicate and next-state relation. His next-state relation would not be satisfied by any next state if the values of x and y were equal in the current state. The algorithm would therefore stop in such a state. He defined his initial predicate and next-state relation as

$$\begin{aligned} \mathit{Init} &\triangleq x = M \ \land \ y = N \\ \mathit{Next} &\triangleq (x < y \ \land \ y' = y - x \ \land \ x' = x) \lor \\ &\qquad (y < x \ \land \ x' = x - y \ \land \ y' = y) \end{aligned}$$

When he showed these to FG, she reminded him that the definition of *Next* would be easier to read if he used the TLA$^+$ convention of writing disjunctions and conjunctions as lists of formulas bulleted by $\lor$ and $\land$. Changing this definition, he arrived at the TLA$^+$ description of the algorithm in Figure 2.

Euclid looked with satisfaction at formulas *Init* and *Next*. "What a simple and natural way to write the algorithm," he said. "Why do people use languages like PlusCal that make it more complicated? Why don't they just use mathematics, which is so simple?"

FG answered: "The people who design computers and write programs think that mathematics is too difficult. They find **while** loops and **if** statements easier to understand than sets and logic."

MODULE *TheAlgorithmInTLA*

EXTENDS *Integers*
CONSTANTS M, N
VARIABLES x, y

$$
\begin{array}{lll}
\mathit{Init} & \triangleq & (x = M) \land (y = N) \\
\mathit{Next} & \triangleq & \lor \land x < y \\
 & & \quad \land y' = y - x \\
 & & \quad \land x' = x \\
 & & \lor \land y < x \\
 & & \quad \land x' = x - y \\
 & & \quad \land y' = y
\end{array}
$$

Figure 2: Euclid's description of the algorithm in TLA$^+$.

"No wonder computers are so hard to use!" exclaimed Euclid. "I must finish my *Elements* so they will see how simple and powerful mathematics really is."

FG did not have the heart to tell him that he *did* finish his *Elements*, but programmers still think programming languages are simple and mathematics is complicated.

Euclid had no time to dwell on these thoughts. He had expressed the algorithm with two simple formulas, but he still had to prove that it produces the correct answer if and when it stops.

Euclid couldn't think of any way to show that a formula is true when an algorithm stops. It seemed better to try to prove that a formula is true throughout the entire execution of the algorithm. Since the algorithm is stopped when x equals y, he just needed to prove that the every state of every execution of the algorithm satisfied[5]

$$(x = y) \Rightarrow (x = \mathit{GCD}(M, N) \land y = \mathit{GCD}(M, N))$$

He named this formula *ResultCorrect*.

At this point, Euclid decided it would be wise to use the TLC model checker to be sure he hadn't made a mistake. He learned that a formula that is true in every state of every execution is called an *invariant*, and that TLC can check if a formula is an invariant. He created a model just like before (substituting 72 for M and 48 for N) and ran TLC to check that *ResultCorrect* really is an invariant of the algorithm.

TLC reported the error *deadlock reached*. Euclid examined the error trace displayed by the Toolbox, but it showed a perfectly fine execution. He asked FG to explain what *deadlock* meant, and she did.

"*Deadlock* is just a synonym for *termination*," said Euclid. "Both mean that the algorithm has reached a state from which it cannot take a step." Then he remembered those steps leaving all variables unchanged that are allowed by the PlusCal translator's next-state relation. He saw that they were added just to keep the model checker from thinking that the algorithm has stopped when it has stopped.

Fortunately, it was easy to tell TLC not to report deadlock as an error, after which it found no error. Euclid then returned to the problem of proving that *ResultCorrect* is an invariant.

Euclid knew he could use mathematical induction to prove that a formula F is true in all states of any execution. He just had to prove two things:

(1) F is true in any possible initial state.

(2) For any states s and t, if F is true in state s and a step can go from state s to state t, then F is true in state t.

For this algorithm, a possible initial state is one satisfying formula *Init*. Euclid could see right away that this meant that (1) is expressed by the mathematical formula

(1) $Init \Rightarrow F$

How could he write the second assertion as a formula? This stumped him for a while. However, looking up TLA$^+$ on the Web, Euclid (with FG's help) discovered that it was possible to prime a formula, not just a variable. Priming a formula means priming all the variables in that formula. Euclid remembered that a step from state s to state t is possible if and only if the next-state relation *Next* is true when you replace the unprimed variables by their values in state s and the primed variables by their values in state t. He could therefore see that (2) is expressed by

(2) $F \wedge Next \Rightarrow F'$

Proving (1) and (2) therefore proves that F is true in every state of every execution of the algorithm—in other words, that F is an invariant.

Euclid was about to shout *Eureka!* when he realized that he could not use this method to prove that *ResultCorrect* is an invariant. Formula *ResultCorrect* is true in any state in which x and y are not equal, so (2) could not be true when *ResultCorrect* is substituted for F.

He decided that he had to find an F satisfying (1) and (2) that implied *ResultCorrect*—that is, an F satisfying

(3) $F \Rightarrow ResultCorrect$

Proving (1) and (2) proves that F is an invariant, and (3) then implies that *ResultCorrect* is also an invariant. Euclid found that a suitable choice for F would be the formula

$$\begin{aligned} InductiveInvariant \triangleq\ & \land\, x \in Number \\ & \land\, y \in Number \\ & \land\, GCD(x,\, y) = GCD(M,\, N) \end{aligned}$$

He then set about trying to prove formulas (1)–(3) with *InductiveInvariant* substituted for F—formulas that he named *InitProperty*, *NextProperty*, and *WeakensProperty*.

Euclid realized that he could not prove these properties without some facts. The most obvious fact he needed was the assumption that M and N are numbers. To write it, he had to know how to write the set of numbers in TLA$^+$. FG told him that what he called a number was now usually called a non-zero natural number. The *Integers* module defines *Nat* to be the set of natural numbers, so Euclid wrote the definition

$$Number \triangleq Nat \setminus \{0\}$$

He asserted the assumption that M and N are numbers, and gave that assumption the name *NumberAssumption*, by writing

$$\text{ASSUME } NumberAssumption \triangleq M \in Number \land N \in Number$$

Euclid knew that the algorithm's correctness also depends on some simple facts about the gcd. He was interested in proving the correctness of the algorithm, not in proving such facts. So he decided to make them assumptions:

$$\begin{aligned} &\text{ASSUME } GCDProperty1 \triangleq \forall\, p \in Number : GCD(p, p) = p \\ &\text{ASSUME } GCDProperty2 \triangleq \forall\, p, q \in Number : GCD(p, q) = GCD(q, p) \\ &\text{ASSUME } GCDProperty3 \triangleq \forall\, p, q \in Number : \\ &\qquad\qquad (p < q) \Rightarrow (GCD(p, q) = GCD(p,\, q-p)) \end{aligned}$$

Euclid did not want to bother proving these facts now. But what if he had made a mistake and they weren't really true?

FG suggested that they let TLC check them. The documentation told them that it checks ASSUME formulas, so they tried it. TLC reported the error *Nat is not enumerable*. "Of course," said Euclid. "The model checker works by enumerating all cases. It cannot check if something is true for all numbers because there are infinitely many of them."

Looking through the Toolbox's help pages, Euclid and FG discovered that they could tell TLC to pretend that a definition had been replaced by a new one, without having to change their formulas. They had TLC pretend that *Nat* is the set $0 \,..\, 50$. After Euclid changed the values of M and N in the model so TLC would find *NumberAssumption* to be true with this pretend definition, it took about 10 seconds to determine that all the assumptions were then true. Euclid felt confident that if the three gcd properties held for all numbers up to 50, then he hadn't made a mistake, so he could wait to prove them later. He was now ready to write a proof of the algorithm.

Euclid was about to ask FG for a sandbox to write in when they noticed that the Toolbox can also be used to run a proof checker. "Let us use that," he said. "Later," said FG. "I have to do the grocery shopping now." And she left. While she was gone, Euclid wrote the proof in Figure 3 and had the computer check it.

When FG returned, she found Euclid's proof on the screen, all highlighted in green to indicate that it had been checked. "How did you do that?" she asked. Euclid replied:

"I found some examples of proofs written in TLA$^+$. The basic hierarchical structure of a proof like that of *NextProperty* was rather obvious once I deduced that QED stands for *what we have to prove* (the current goal). It was not hard to see that the SUFFICES statement asserts that, to prove the current goal, it suffices to assume the facts in the ASSUME clause and prove the PROVE clause. However, it took me a while to realize that it allows us to assume those facts in the rest of the current proof[6], and that it made the PROVE clause the current goal.

"It was a little hard getting used to explicitly stating almost everything required by a proof—even which definitions must be used. (That's what all those DEF clauses do.) And I had a devil of a time figuring out that to use the CASE assumption in the proof of $\langle 1 \rangle 1$, I had to refer to it as $\langle 1 \rangle 1$. (That makes the proof look circular, as if a statement is being used to prove itself.)

"Fortunately, when the proof checker cannot verify something, the Toolbox displays exactly what it is trying to verify. I could usually see that I had forgotten to give it some fact or to tell it to use some definition. And I must admit that writing exactly what's used in each step makes it easier for a human to understand the proof. Actually, a USE statement allows you to

```
THEOREM InitProperty ≜ Init ⇒ InductiveInvariant
PROOF BY NumberAssumption DEF Init, InductiveInvariant

THEOREM NextProperty ≜ InductiveInvariant ∧ Next ⇒ InductiveInvariant′
⟨1⟩ SUFFICES ASSUME InductiveInvariant, Next
                PROVE  InductiveInvariant′
  OBVIOUS

⟨1⟩ USE DEF InductiveInvariant, Next

⟨1⟩1. CASE x < y
   ⟨2⟩1. y − x ∈ Number  ∧  ¬(y < x)
      BY ⟨1⟩1, SimpleArithmetic DEF Number
   ⟨2⟩2. QED
      BY ⟨1⟩1, ⟨2⟩1, GCDProperty3

⟨1⟩2. CASE y < x
   ⟨2⟩1. x − y ∈ Number  ∧  ¬(x < y)
      BY ⟨1⟩2, SimpleArithmetic DEF Number
   ⟨2⟩2. GCD(y′, x′) = GCD(y, x)
      BY ⟨1⟩2, ⟨2⟩1, GCDProperty3
   ⟨2⟩3. QED
      BY ⟨1⟩2, ⟨2⟩1, ⟨2⟩2, GCDProperty2

⟨1⟩3. QED
   ⟨2⟩1. (x < y)  ∨  (y < x)
      BY SimpleArithmetic DEF Number
   ⟨2⟩2. QED
      BY ⟨1⟩1, ⟨1⟩2, ⟨2⟩1

THEOREM WeakensProperty ≜ InductiveInvariant ⇒ ResultCorrect
PROOF BY GCDProperty1 DEF InductiveInvariant, ResultCorrect
```

Figure 3: Euclid's machine-checked proof

say that facts and definitions are to be assumed for the rest of the current proof. But I used it sparingly so the proof would be easier to read.

"I was getting nowhere trying to check proofs that depended on very simple facts about numbers until I discovered that I had to EXTEND the standard *TLAPS* module and use a magical fact from it called *SimpleArithmetic*.[7]

"I could check the proofs of individual steps by themselves as I wrote the proof. The checker remembered what it had already verified and never proved the same thing twice. But that wasn't important, since it was able to check my entire proof in seconds. These days, people must write much longer proofs than I do."

Although he now had a machine-checked proof, Euclid was not completely satisfied. His proof consisted of three separate theorems. Wouldn't it be nice if the correctness of the algorithm could be asserted by a single formula?

Euclid suspected that this was possible. He felt that doing it required understanding those formulas with $\Box$ and subscripts produced by the PlusCal translator. He was able to infer that he could describe the algorithm with the single formula *Spec* defined by

$$Spec \triangleq Init \wedge \Box[Next]_{\langle x, y\rangle}$$

He also discovered that $[Next]_{\langle x,y\rangle}$ is an abbreviation for

$$Next \vee \text{UNCHANGED } \langle x, y\rangle$$

and that UNCHANGED $\langle x, y\rangle$ simply means $\langle x, y\rangle' = \langle x, y\rangle$, which implies $x' = x$ and $y' = y$. He still didn't know what the $\Box$ meant, but he knew it was crucial.

In the *TLAPS* module, Euclid found

$$\begin{array}{l} \text{THEOREM } Inv1 \triangleq \text{ASSUME STATE } I,\ \text{STATE } f,\ \text{ACTION } N, \\ \qquad\qquad\qquad\qquad\qquad\quad I \wedge [N]_f \Rightarrow I' \\ \qquad\qquad\qquad\quad \text{PROVE} \quad I \wedge \Box[N]_f \Rightarrow \Box I \\ \text{PROOF OMITTED} \end{array}$$

As he was pondering this, FG said: "The hour has come. I must now take you back to your time."

"But how will we give the algorithm to Professor Broy?"

"I will handle that," FG said as she led Euclid into a passing wormhole.

On the trip back, Euclid asked what Professor Broy had done that there should be a special celebration of his birthday. FG replied: "I can spend

She led Euclid into a passing wormhole.

THEOREM $Correctness \triangleq Spec \Rightarrow \Box ResultCorrect$

$\langle 1 \rangle 1$. $Spec \Rightarrow \Box InductiveInvariant$

$\langle 2 \rangle 1$. $InductiveInvariant \wedge$ UNCHANGED $\langle x, y \rangle \Rightarrow InductiveInvariant'$
BY DEF $InductiveInvariant$

$\langle 2 \rangle 2$. $InductiveInvariant \wedge [Next]_{\langle x, y \rangle} \Rightarrow InductiveInvariant'$
BY $\langle 2 \rangle 1$, $NextProperty$

$\langle 2 \rangle 3$. $InductiveInvariant \wedge \Box[Next]_{\langle x, y \rangle} \Rightarrow \Box InductiveInvariant$
BY $\langle 2 \rangle 2$, $Inv1$

$\langle 2 \rangle 4$. QED
BY $InitProperty$, $\langle 2 \rangle 3$ DEF $Spec$

$\langle 1 \rangle 2$. $\Box InductiveInvariant \Rightarrow \Box ResultCorrect$
BY $WeakensProperty$, $PropositionalTemporalLogic$

$\langle 1 \rangle 3$. QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$

Figure 4: The mysterious proof.

many hours detailing his achievements, including his more than 200 publications and the many students he has inspired. But describing his contributions would require explaining things you have never heard of: semantics, algebraic specification, abstract data types, dataflow, software engineering, message sequence charts, system modeling, embedded systems, automobiles, and more. It is enough to say that he has been a leader in the struggle to make our computer systems easier to understand."

With a puzzled look, Euclid said: "What are computer systems? These words have a familiar echo, but I do not understand them." He was forgetting all that he had experienced in the future, as the logic of time travel implied that he must.

"It does not matter," said FG. "Tell me more about your *Elements*."

Upon her return to 2010, FG glanced at the computer screen. She was surprised to see the proof shown in Figure 4.[8] "Where did that come from?" she wondered.

Translator's Notes

1. It is not clear what documentation Euclid and FG found. They could have learned about PlusCal from the information on

`http://research.microsoft.com/en-us/um/people/lamport/tla/pluscal.html`

That site can also be reached by searching the Web for the 25-letter string obtained by removing the hyphens from `uid-lamport-pluscal-homepage`. They probably also went to `http://tlaplus.net`, the TLA$^+$ user community site. They may also have used

`http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html`

a site that can be found by searching the Web for the 21-letter string obtained by removing the hyphens from `uid-lamport-tla-homepage`.

2. The reader undoubtedly recognizes IDE as short for *Integrated Development Environment*. The TLA in TLA$^+$ stands for *Temporal Logic of Actions*. The significance of the + is not clear, but it was often found at the end of programming-language names.

3. Throughout this translation, I show not what Euclid saw while writing his algorithm, but what the Toolbox displayed as the pretty-printed version. In 2010, the pretty-printer still did not properly display PlusCal code, so I have shown the algorithm itself as it should have been displayed.

Although forced to use the Roman alphabet and some English terms like *Integers* and **while**, Euclid chose as identifiers transliterations of Greek terms and phrases. I have taken the liberty of replacing them with their modern English equivalents. For example, I changed *UpologizoMKD*, Euclid's transliteration of $U\pi o\lambda o\gamma\iota\zeta\omega MK\Delta$ (the $MK\Delta$ of course standing for $\mu\acute{\varepsilon}\gamma\iota\sigma\tau o\varsigma\ \kappa o\iota\nu\acute{o}\varsigma\ \delta\iota\alpha\iota\rho\acute{\varepsilon}\tau\eta\varsigma$), to *ComputeGCD*.

4. The observant reader will note that the definitions do not say that p and q are numbers or that S is a set of numbers. These definition have the expected meanings only if their arguments have the expected "types". If we let q be the string "abc", then the definition of *Divisors* implies that *Divisors*("abc") equals

$\{d \in 1\,..\,\text{“abc”} : d \mid \text{“abc”}\}$

but we don't know what this expression means. We couldn't prove anything interesting about it, and TLC would report an error if it had to evaluate it.

5. Euclid certainly knew that this formula is equivalent to the shorter one

$$(x = y) \;\Rightarrow\; (x = GCD(M, N))$$

because $x = y$ and $x = GCD(M, N)$ imply $y = GCD(M, N)$. He preferred his formula because symmetry was central to the Greek concept of beauty.

6. Euclid may not have realized that, if the SUFFICES step had a complete step name like $\langle 1 \rangle 7$ or $\langle 1 \rangle \mathit{suf}$, then those ASSUME facts would have been used only when referred to by that name. The need to use a step explicitly can be avoided by not giving the step a complete name.

7. Using *SimpleArithmetic* as a fact in a proof instructs the proof checker to apply Cooper's algorithm, which can check any formula in a certain decidable subset of arithmetic.

8. Had FG directed the prover to check this proof, she would have found that it verified only steps $\langle 2 \rangle 1$ and $\langle 2 \rangle 2$. Checking the other steps would have generated error messages because, in 2010, the proof checker could not yet perform temporal logic reasoning. Its author must have guessed at some details of the proof, such as the use of *PropositionalTemporalLogic* to invoke a decision procedure.

Acknowledgment

I wish to thank my long-time collaborator Leo Guibas, an expert on dialects of the Greek isles, for his help with the translation. Readers can thank Kaustuv Chaudhuri for correcting an historical error. Euclid would have wanted to thank Stephan Merz for simplifying the proof in Figure 3.