The subsequent text illustrates the solution of the problem obtained with the VeriFun system. File “Infinitude of Primes (Pronic)[sf].vf” contains all definitions and proofs and can be downloaded from http://www.verifun.de/. The system is needed for inspecting the file. The system’s website provides installers for Windows, Mac and Unix/Linux for download.

Note:

- The proofs are obtained using 6 procedures (viz. >, +, −, * and mod) and 36 lemmas from the Arithmetic Library which are not displayed here.
- Termination of all procedures in this case study had been proved automatically.
- For assessing the amount of user interaction when proving a lemma, lines starting with ”%” display the proof rules which had been used interactively to complete the proof of the lemma preceding the %-line.
- Lines starting with “//” are comments belonging to the subsequent definition.
- The strings between “lemma” and “<=” are just identifiers assigning a name to a lemma for reference and must not be confused with the statement of a lemma given as a boolean term in the lemma body.
- ?0(y) stands for y = 0.
- $^{+}$(y) denotes the successor of y.
- $^{-}$(y) denotes the predecessor of y.

```
// Smallest factor: sf(x, 0) is undefined, and sf(x, 1) = x. If y ≥ 2, then
//
// sf(x, y) = x, if no n ∈ {2, ... , y} divides x, and otherwise
// sf(x, y) = n, if n is the smallest number in {2, ... , y} dividing x.
//
// Therefore sf(x, y) computes the smallest factor f ≥ 2 of x, if x ≥ 2
// and y ≥ x - 1.

function sf(x : ℕ, y : ℕ) : ℕ <=
if ¬ ?0(y)
  then if ?0(⁻(y))
         then x
         else if ?0((x mod y))
                then if sf(x, ⁻(y)) = x
                       then if x > y then y else x end_if
                       else sf(x, ⁻(y))
                      end_if
                else sf(x, ⁻(y))
              end_if
       end_if
end_if
```

```
lemma y ≠ 0 → (x = 0 ↔ sf(x, y) = 0) <= ∀ x, y : ℕ
  if{?0(y), true, if{?0(x), ?0(sf(x, y)), ¬ ?0(sf(x, y))}}
% —

lemma y ≠ 0 → (x = 1 ↔ sf(x, y) = 1) <= ∀ x, y : ℕ
  if{?0(y), true, if{?0(x), true, if{?0(⁻(x)), ?0(⁻(sf(x, y))), ¬ ?0(⁻(sf(x, y)))}}}
% —

lemma y ≠ 0 ∧ n ≠ 0 ∧ sf(n * x, y) = n * x → sf(x, y) = x <= ∀ x, y, n : ℕ
  if{?0(y), true, if{?0(n), true, if{sf(n * x, y) = n * x, sf(x, y) = x, true}}}
% 1 x Induction, 1 x Unfold Procedure

lemma y ≠ 0 → sf(x, y) = x ∨ sf(sf(x, y), ⁻(sf(x, y))) = sf(x, y) <= ∀ x, y : ℕ
  if{?0(y), true, if{sf(x, y) = x, true, sf(sf(x, y), ⁻(sf(x, y))) = sf(x, y)}}
% 1 x Apply Equation

lemma x ≠ 0 ∧ y ≠ 0 → sf(x, y) | x <= ∀ x, y : ℕ
  if{?0(y), true, if{?0(x), true, ?0((x mod sf(x, y)))}}
% 1 x Induction


// P(x, 0) is undefined, and P(x, 1) = true. Otherwise P(x, y) = true
// iff x ≠ 0, y ≥ 2 and no n ∈ {2, ... , y} divides x.
// Hence in particular x is prime iff x ≥ 2 and P(x, x−1) = true.

function P(x : ℕ, y : ℕ) : bool <=
if ¬ ?0(y)
  then if ?0(⁻(y))
          then true
          else if ?0((x mod y)) then false else P(x, ⁻(y))  end_if
         end_if
end_if

lemma x > y ≥ 1 → (sf(x, y) = x ↔ P(x, y)) <= ∀ x, y : ℕ
  if{x > y, if{?0(y), true, if{sf(x, y) = x, P(x, y), ¬ P(x, y)}}, true}
% —

lemma x ≥ 2 → P(sf(x, ⁻(x)), ⁻(sf(x, ⁻(x)))) <= ∀ x : ℕ
  if{?0(x), true, if{?0(⁻(x)), true, P(sf(x, ⁻(x)), ⁻(sf(x, ⁻(x))))}}
% 1 x Use Lemma


// ℙ(x) decides whether x is prime. Note:
//     ∀ n, m : ℕ n ≠ 0 ∧ ℙ(m) ∧ n | m → n = 1 ∨ n = m
```

```
// is a lemma in the library (not being used here).

function ℙ(x : ℕ) : bool <=
if ?0(x)
  then false
  else if ?0(⁻(x)) then false else P(x, ⁻(x))  end_if
end_if



// pf(x) is undefined, if x ≤ 1, and otherwise
//
// pf(x) = x, if no n ∈ {2, ... , x − 1} divides x, and otherwise
// pf(x) = n, if n is the smallest number in {2, ... , x − 1} dividing x.
//
// Therefore pf(x) computes the smallest factor f ≥ 2 of x, if x ≥ 2.

function pf(x : ℕ) : ℕ <=
if ¬ ?0(x)
  then if ¬ ?0(⁻(x)) then sf(x, ⁻(x)) end_if
end_if



//pf(x) is a prime number, if x ≥ 2.

lemma x ≥ 2 → ℙ(pf(x)) <= ∀ x : ℕ
  if{?0(x), true, if{?0(⁻(x)), true, ℙ(pf(x))}}
% 1 x Unfold Procedure



// Starting with the first pronic number 2, subsequent pronic numbers
// are computed by procedure S with the previous pronic number.

function S(n : ℕ) : ℕ <=
if ?0(n)
  then 2
  else let S_n−1 := S(⁻(n)) in S_n−1 * S_n−1 + S_n−1 end_let
end_if

lemma S(n) ≠ 0 <= ∀ n : ℕ
  ¬ ?0(S(n))
% —

lemma (Thm 1) ℙ(pf(S(n)+1)) <= ∀ n : ℕ
  ℙ(pf(⁺(S(n))))
```

```
% 1 x Use Lemma

lemma m ≥ 2 ∧ i ≠ 0 ∧ m | S(n)+1 → m | S(n + i) <= ∀ i, n, m : ℕ
  if{?0(m),
    true,
    if{?0(⁻(m)),
      true,
      if{?0(i), true, if{?0((⁺(S(n)) mod m)), ?0((S(n + i) mod m)), true}}}}
% 1 x Induction, 2 x Case Analysis, 3 x Apply Equation

lemma n > m → pf(S(n)+1) ≠ pf(S(m)+1) <= ∀ n, m : ℕ
  if{n > m, ¬ pf(⁺(S(m))) = pf(⁺(S(n))), true}
% 1 x Case Analysis, 1 x Use Lemma, 2 x Apply Equation

lemma (Thm 2) pf(S(n)+1) = pf(S(m)+1) → n = m <= ∀ n, m : ℕ
  if{pf(⁺(S(n))) = pf(⁺(S(m))), n = m, true}
% 1 x Use Lemma
```