Formal Derivation of an Algorithm for the Stamps Problem*

Robert L. Constable   Christoph Kreitz
Department of Computer Science, Cornell-University, Ithaca, NY 14853-7501
{rc,kreitz}@cs.cornell.edu

Abstract

We show how to formally derive algorithms for a simple class of arithmetic problems that we call stamps problems. We specify them in a simple theory of numbers and prove constructively that they have solutions. From these proofs our logical programming environment constructs algorithms that are correct-by-construction.

1 Introduction

We found the first stamps problem in the book *Elements of Discrete Mathematics,* by C. L. Liu from 1985. Here is how Liu casts the problem:

> Suppose we have stamps of two different denominations, 3 cents and 5 cents. We want to show that it is possible to make up exactly any postage of 8 cents or more using stamps of these two denominations. Clearly, the approach of showing case by case how to make up postage of 8 cents, 9 cents, 10 cents, and so on, using 3-cent and 5-cent stamps will not be a fruitful one, because there is an infinite number of cases to be examined. Let us consider an alternative approach. We want to show that if it is possible to make up exactly a postage of n cents using 3-cent and 5-cent stamps, then it is also possible to make up exactly a postage of $n+1$ cents using 3-cent and 5-cent stamps.

Those who know a bit of number theory will recognize the Bezout identity that given two relatively prime numbers (also called *coprime* numbers) a, b, then for any integer z there are integers u, v such that $z = u \cdot a + v \cdot b$.

We now consider the restriction that u and v are positive and that for any $n \geq a + b$ there are natural numbers u, v such that $n = u \cdot a + v \cdot b$. For which relatively prime a, b is this equation solvable? We call these *stamps pairs*.

2 Deriving Algorithms for The Basic Stamps Problem

C. L. Liu provides a simple inductive solution for the original stamps problem, by showing how to make up a postage of n+1 cents using 3-cent and 5-cent stamps once we know how to make up a postage of n cents.

*This work was supported in part by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the Office of Naval Research (ONR) under Grant N00014-01-1-0765 (Building Interactive Digital Libraries of Formal Algorithmic Knowledge) and by NSF Grant CCR 0204193 (Proof Automation in Constructive Type Theory).

```
⊢ ∀n:ℕ. n≥8 ⇒ ∃i,j:ℕ. n = i*3 + j*5
BY  NatIndStartingAt ⌜8⌝
.....basecase.....
    ⊢ ∃i,j:ℕ. 8 = i*3 + j*5
√  BY  ExR [⌜1⌝;⌜1⌝] THEN Auto
.....upcase.....
    n:ℕ, n>8, i:ℕ,  j:ℕ,  n-1 = i*3 + j*5  ⊢  ∃i,j:ℕ. n = i*3 + j*5
    BY  Decide ⌜j=0⌝ THEN Auto
    .....Case 1.....
        n:ℕ, n>8, i:ℕ,  j:ℕ,  n-1 = i*3 + j*5, j=0  ⊢  ∃i,j:ℕ. n = i*3 + j*5
    √  BY  ExR [⌜i-3⌝;⌜2⌝] THEN Auto'
    .....Case 2.....
        n:ℕ, n>8, i:ℕ,  j:ℕ,  n-1 = i*3 + j*5, j≠0  ⊢  ∃i,j:ℕ. n = i*3 + j*5
    √  BY  ExR [⌜i+2⌝;⌜j-1⌝]  THEN Auto'
```

Figure 1: Inductive Proof of the Specification Theorem for the Basic Stamps Problem.

> We examine two cases. Suppose we make up a postage of n cents using at least one 5-cent stamp. Replacing a 5-cent stamp by two 3-cent stamps will yield a way to make up a postage of $n+1$ cents. On the other hand, suppose we make up a postage of n cents using 3-cent stamps only. Since $k \geq 8$, there must be at least three 3-cent stamps. Replacing three 3-cent stamps by two 5-cent stamps will yield a way to make up a postage of $n+1$ cents.

Figure 1 shows the trace of a formal proof in the Nuprl system that uses exactly this line of argument. The stamps problem is formalized as the theorem

`∀n:ℕ. n≥8 ⇒ ∃i,j:ℕ. n = i*3 + j*5`

and proven by induction starting at the value 8, for which we apply the library theorem

`NatIndStartingAt   ∀k:ℕ. ∀P:ℕ→ℙ. (P(k) ∧ (∀i>k ⇒ P(i-1) ⇒ P(i))) ⇒ (∀i≥k. P(i))`

The base case is solved by assigning 1 to both existentially quantified variables and using Nuprl's autotactic (trivial standard reasoning) to deal with the remaining proof obligation. In the step case from $n-1$ to n it analyzes the assignments i and j for $n-1$, introduces a case distinction on $j=0$ and then assigns either $i-3$ and 2 or $i+2$ and $j-1$, again using the autotactic to complete the proof.

The above proof implicitly contains an algorithm for computing the number of 3-cent and 5-cent stamps needed to make up a given postage n. Nuprl is capable of extracting this algorithm from the formal proof, and to execute it within Nuprl's computation environment or to export it to other programming systems.

Depending on the formalization of the existential quantifier there are two kinds of algorithms that may be extracted. If ∃ is represented as a (dependent) product type, the algorithm returns both the solution and a that verifies it. If ∃ is represented as a set type, this verification information is dropped during extraction and the algorithm – represented in Nuprl's extended lambda calculus and shown on the left – only performs the required computation. Using standard conversions, Nuprl can then transform the algorithm into any programming language that supports recursive

```
⊢ ∀n:ℕ. n≥8 ⇒ ∃i,j:ℕ. n = i*3+j*5
BY  NatIndThreeStepStartingAt ⌜8⌝
.....basecase 1.....
    ⊢ ∃i,j:ℕ. 8 = i*3+j*5
√  BY  ExR [⌜1⌝;⌜1⌝] THEN Auto
.....basecase 2.....
    ⊢ ∃i,j:ℕ. 9 = i*3+j*5
√  BY  ExR [⌜3⌝;⌜0⌝] THEN Auto
.....basecase 3.....
    ⊢ ∃i,j:ℕ. 10 = i*3+j*5
√  BY  ExR [⌜0⌝;⌜2⌝] THEN Auto
.....upcase.....
    n:ℕ, n≥8+3, i:ℕ,  j:ℕ,  n-3 = i*3+j*5  ⊢  ∃i,j:ℕ. n = i*3+j*5
√  BY  ExR [⌜i+1⌝;⌜j⌝] THEN Auto
```

Figure 2: Solution of the Basic Stamps using 3-Step Induction.

definition and export it to the corresponding programming environment. A conversion into SML, for instance, yields the program shown on the right.

```
let rec stamps_assign n
= if n=8 then <1,1>
  else let <i, j> = stamps_assign (n-1)
       in
         if j=0  then <i-3, 2>
         else <i+2, j-1>
```

```
fun stamps_assign n
= if n=8 then 1,1
  else let val i,j = stamps_assign (n-1)
       in
         if j=0  then i-3, 2
         else i+2, j-1
       end
```

Using stepwise induction is not the only way to solve the stamps problem. Instead of providing a solution for n=8 and then showing how to make up a postage of n+1 cents once we know how to do so for n cents, we could provide a solution for $n = 8$, 9, and 10, and then make up a postage of n+3 cents by adding a 3-cent stamp to the solution for n cents. In the formal proof, shown in Figure 2, we have to use 3-Step induction for this purpose, again applying a library theorem. The resulting algorithm, shown in SML notation below, has the advantage that the loop computes much faster, as it does not involve a test and reduces n by 3 instead of 1.

```
fun stamps_assign n
= if n=8  then 1,1
  if n=9  then 3,0
  if n=10 then 0,2
  else let val i,j = stamps_assign (n-3)
       in
          i+1, j
       end
```

Since Nuprl's type theory comes with built-in division and quotient remainder functions, we can provide an even faster, non-inductive solution for the stamps problem. As before, we reduce a solution for n to the cases 8, 9, and 10, but we don't reduce n recursively, but do it in one step by computing $r = 8+(n-8)\ \mathsf{rem}\ 3$. Given a solution i and j for r, the solution for n is then $i+(n-8)\div 3$

```
⊢ ∀n:ℕ. n≥8 ⇒ ∃i,j:ℕ. n = i*3+j*5
BY Assert ⌜∀n:ℕ. 11>n≥8 ⇒ ∃i,j:ℕ. n=i*3+j*5⌝ THEN Auto

   .....Assertion.....
   n:ℕ, 11>n≥8 ⊢ ∃i,j:ℕ. n = i*3+j*5
   BY Choices [⌜n=8⌝; ⌜n=9⌝;⌜n=10⌝]

      .....Case n=8.....
          ⊢ ∃i,j:ℕ. 8 = i*3+j*5
      √ BY ExR [⌜1⌝;⌜1⌝] THEN Auto
      .....Case n=9.....
          ⊢ ∃i,j:ℕ. 9 = i*3+j*5
      √ BY ExR [⌜3⌝;⌜0⌝] THEN Auto
      .....Case n=10....
          ⊢ ∃i,j:ℕ. 10 = i*3+j*5
      √ BY ExR [⌜0⌝;⌜2⌝] THEN Auto

   .....Reduction.....
   n:ℕ, n≥8, ∀n:ℕ. 11>n≥8 ⇒ ∃i,j:ℕ. n=i*3+j*5 ⊢ ∃i,j:ℕ. n = i*3+j*5
   BY allL (-1) ⌜8+(n-8) rem 3⌝ THEN Repeat (exL (-1))
      n:ℕ, n≥8, i:ℕ, j:ℕ, 8+(n-8) rem 3 = i*3+j*5 ⊢ ∃i,j:ℕ. n = i*3+j*5
   √ BY ExR [⌜i+(n-8)÷3⌝;⌜j⌝] THEN ILemma ‘div_rem_sum‘ [⌜n-8⌝; ⌜3⌝
```

Figure 3: Solution of the Basic Stamps using Direct Reduction.

and j. A formal proof of this argument is given in Figure 3. We assert that the problem has a solution over the limited range $8 \leq n < 11$, provide a solution for each of these cases, and reduce the general problem by instantiating it with $r = 8 + (n-8)\ \mathsf{rem}\ 3$ and then modify its solution by adding $(n-8) \div 3$ to i. As checking the solution involves reasoning about division and quotient remainder we supply a lemma to enable the autotactic to complete the proof. The resulting algorithm, shown in SML notation below, provides the fastest possible solution for the stamps problem.

```
fun stamps_assign n
= let q = (n-8)÷3
  and r = (n-8) rem 3 + 8
  in
     if r=8  then 1+q, 1
     if r=9  then 3+q, 0
     if r=10 then 0+q, 2
```

3 An Informal Proof for the General Stamps Problem

In the previous section we have shown how to solve the stamps problem efficiently for the pair 3 and 5. Now the question is if there are other combinations of a and b that can be proven to be stamps pairs. Obviously, $a = 1$ and any b will be stamps pairs and so will be $a = 2$ and any odd number b. But are there others?

An informal solution for this problem was first presented at the International Summer School at Marktoberdorf in July 1995. Using basic number theory it shows that there cannot be any other stamps pairs. The statement and its proof are the following.

Let $a, b \in \mathbb{N}$ and without loss of generality $a < b$. If for all $n \geq a + b$ there are $i, j \in \mathbb{N}$ such that $n = i \cdot a + j \cdot b$ then $a{=}1$ or $a{=}2$ and b is odd or $a{=}3$ and $b{=}5$.

Proof: If $a = 1$, we're done, so assume $1 < a < b$

Since $a{+}b{+}1 = i \cdot a{+}j \cdot b$ for some i, j it must be that $a \mid (b{+}1)$ or $b{=}a{+}1$ (1)

Since $a{+}b{+}2 = i \cdot a{+}j \cdot b$ for some i, j it must be that $a{=}2$ or $a \mid (b{+}2)$ or $b{=}a{+}2$ (2)

Case analysis

$a = 2$**:** by (1), b must be odd

$a > 2$**:** then $b > 3$. We use (1) to split into subcases

$a \mid (b + 1)$**:** Then, because of $a > 2$, a cannot divide $b{+}2$ as well.
By (2), we thus have $b = a{+}2$.
Now, since $a{+}b{+}3 = i \cdot a{+}j \cdot b$ for some i, j we know $a{=}3$ or $a \mid (b{+}3)$ or $b{=}a{+}3$.
$b = a{+}3$ is impossible since $b = a{+}2$.
$a \mid (b{+}3)$ is impossible since $a \mid (b + 1)$ and $a > 2$.
Thus $a = 3$ and $b = 5$.

$b \mid (a + 1)$**:** then by the same argument $b = a + 1$
But then by (2), $a \mid (a + 3)$ or $a + 1 \mid (a + 2)$, both of which are impossible.

4 A Formal Proof for the General Stamps Problem

Although the above solution for the stamps problem was generally accepted, an attempt to recast this proof in a formal setting failed, since the argument for the case $b \mid (a + 1)$ did not provide sufficient detail to complete the formal proof. In fact, being forced to take a closer look at this case revealed that the argument was wrong: the subcase $a \mid (a + 3)$ is not impossible, but leads to another stamps pair, namely $a{=}3$ and $b{=}4$. But the formal proof also showed that there were no further stamps pairs.

Figure 4 describes the main part of the formal proof. The proof proceeds by decomposing the proof goal using Nuprl's autotactic. In the case where we want to prove that there are only four combinations of stamps for which the stamps problem can be solved we consider three alternatives, among which the first ($a{=}1$) trivially leads to a solution and the other two are solved by instantiating separate lemmas with the tactic `ILemma`. In the other case, where we have to prove that the 4 combinations actually lead to a solution of the stamps problem, we do case analysis over the four possibilities, perform backward reasoning over a lemma to reduce the problem to the base case of the induction, and then provide explicit solutions for all possible values in the range $\{a{+}b ... 2 \cdot a{+}b^{-}\}$. In the case where b is odd, we make use of the fact that an odd number is equal to $2 \cdot c{+}1$ for some c.

The proofs of the main theorem and the lemmas use notation that extends the basic type theory of Nuprl to make the formal statements more comprehensible. For this purpose, the following *abstractions* were added to the library of the Nuprl system.

```
ABS int_upper     {i..}     ≡ {j:ℤ| i≤j}
ABS int_seg       {i..j⁻}   ≡ {k:ℤ| i≤k<j}
ABS divides       a | b     ≡ ∃c:ℤ. a = b*c
ABS is_odd        a is odd  ≡ 2 | a+1
ABS stampspairs   a and b are stamps pairs ≡ ∀n:{a+b...}. ∃i,j:ℕ. n = i*a+j*b
```

```
THM Stamps Theorem

∀a,b:ℕ.  (0<a ∧ a<b) ⇒
a and b are stamps pairs  ⇔  a=1 ∨ (a=2 ∧ b is odd) ∨ (a=3 ∧ b=4) ∨ (a=3 ∧ b=5))
BY Auto

   .....⇒.....
   a:ℕ, b:ℕ, 0<a, a<b, a and b are stamps pairs
   ⊢ a=1 ∨ (a=2 ∧ b is odd) ∨ (a=3 ∧ b=4) ∨ (a=3 ∧ b=5)
   BY Alternatives [⌜a=1⌝; ⌜a=2⌝; ⌜a>2⌝]

      .....Case 2.....
      a:ℕ, b:ℕ, 0<a, a<b, 2 and b are stamps pairs, a=2
      ⊢ a=1 ∨ (a=2 ∧ b is odd) ∨ (a=3 ∧ b=4) ∨ (a=3 ∧ b=5)
   √ BY ILemma ‘stampspairs_if_two‘ [⌜b⌝] THEN prover

      .....Case 3.....
      a:ℕ, b:ℕ, 0<a, a<b, a and b are stamps pairs, a>2
      ⊢ a=1 ∨ (a=2 ∧ b is odd) ∨ (a=3 ∧ b=4) ∨ (a=3 ∧ b=5)
   √ BY ILemma ‘stampspairs_if_greater_two‘ [⌜a⌝;⌜b⌝]

   .....⇐.....
   a:ℕ, b:ℕ, 0<a, a<b,  a=1 ∨ (a=2 ∧ b is odd) ∨ (a=3 ∧ b=4) ∨ (a=3 ∧ b=5)
   ⊢ a and b are stamps pairs
   BY AnalyzeCasesInHypothesis 5 THEN BackLemma ‘stampspairs_properties‘

      .....Case 1.....
      b:ℕ, 1<b,     n:{1+b..2*1+b⁻}  ⊢  ∃i,j:ℕ.  n = i*1+j*b
   √ BY ExR [⌜n⌝; ⌜0⌝]

      .....Case 2.....
      b:ℕ, b is odd, n:{2+b..2*2+b⁻}  ⊢  ∃i,j:ℕ.  n = i*2+j*b
   √ BY Choices [⌜n=2+b⌝    ; ⌜n=3+b⌝                        ]
        THENL   [ExR [⌜1⌝;⌜1⌝]; DVars [‘c’] 2 THEN ExR [⌜1 + c⌝;⌜0⌝]]

      .....Case 3.....
      n:{3+4..2*3+4⁻}                  ⊢  ∃i,j:ℕ.  n = i*3 + j*4
   √ BY Choices [⌜n=7⌝       ; ⌜n=8⌝       ;⌜n=9⌝         ]
        THENL   [ExR [⌜1⌝;⌜1⌝]; ExR [⌜0⌝;⌜2⌝]; ExR [⌜3⌝;⌜0⌝] ]

      .....Case 4.....
      n:{3+5..2*3+5⁻}                  ⊢  ∃i,j:ℕ.  n = i*3+j*5
   √ BY Choices [⌜n=8⌝       ; ⌜n=9⌝       ;⌜n=10⌝        ]
        THENL   [ExR [⌜1⌝;⌜1⌝]; ExR [⌜3⌝;⌜0⌝]; ExR [⌜0⌝;⌜2⌝] ]
```

Figure 4: Proof of the Main Theorem.

```
THM stampspairs_properties
∀a,b:ℕ. 0<a ⇒ (∀n:{a+b..2*a+b⁻}. ∃i,j:ℕ. n = i*a+j*b)  ⇒  a and b are stamps pairs
BY Unfold ‘stampspairs‘ 0 THEN Auto

   a:ℕ, b:ℕ, 0<a,  ∀n:{a+b..2*a+b⁻}.  ∃i,j:ℕ.  n = i*a+j*b,  n:{a+b...}
   ⊢ ∃i,j:ℕ.  n = i*a+j*b
   BY allL 4 ⌜a + b + (n-(a+b) rem a)⌝ THEN Repeat (exL (-1))

      a:ℕ, b:ℕ, 0<a, n:{a+b...}, i:ℕ, j:ℕ,  a + b + (n-(a+b) rem a) = i*a+j*b
      ⊢ ∃i,j:ℕ.  n = i*a+j*b
      BY ExR [⌜(n-(a+b))÷a + i⌝; ⌜j⌝]

         a:ℕ, b:ℕ, 0<a, n:{a+b...}, i:ℕ, j:ℕ,  a + b + (n-(a+b) rem a) = i*a+j*b
         ⊢ n = (((n-(a+b))÷a + i) * a) + j*b
      √ BY ILemma ‘div_rem_sum‘ [⌜n-(a+b)⌝; ⌜a⌝]
```

Figure 5: Proofs of the Reduction Theorem (`stampspairs_properties`)

Figure 5 describes the proof of the lemma `stampspairs_properties`, which is used to reduce the stamps property to a problem over the finite range $\{a{+}b...2{\cdot}a{+}b^{-}\}$. Usually, one would prove this lemma by induction over the value `n`. However, since division ($i{\div}j$) and quotient remainder (i rem j) are primitives of Nuprl's type theory, we can provide a direct solution to the general problem by instantiating the limited one with an appropriate value. This requires us to show that $(((n{-}(a{+}b)){\div}a{+}i){\cdot}a){+}j{\cdot}b$ is in fact the same as the value n. As reasoning about division and quotient remainder is more complex than the autotactic can handle, we have to supply a lemma to make it complete the proof.

Figure 6 shows the proof of lemma `stampspairs_if_two`, which is used to solve one of the cases of the main theorem. It states that a number b must be odd if 2 and b are stamps pairs. We prove it by instantiating the stamps property for the value $2{\cdot}b{+}1$ and then use arithmetical reasoning with the help of a lemma about division.

The most demanding proof in our solution is the one of lemma `stampspairs_if_greater_two`, shown in Figures 7 and 8. It shows that if $a > 2$ and $b > a$ are stamps pairs, then a must be 3 and b must be either 4 or 5. Essentially we follow the informal argument and state that a divides $b{+}1$ or $b{=}a{+}1$ and that a divides $b{+}2$ or $b{=}a{+}2$.

```
THM stampspairs_if_two
 ∀b:ℕ.  2 and b are stamps pairs  ⇒  b is odd
 BY Auto THEN StampsInstance 2 ⌜2*b+1⌝

   b:ℕ, i:ℕ, j:ℕ,  2*b+1 = i*2+j*b  ⊢  b is odd
   BY ILemma ‘odd_mul_cancel‘ [⌜j⌝;⌜b⌝]

      b:ℕ, i:ℕ, j:ℕ,  2*b+1 = i*2+j*b  ⊢  j*b is odd
   √ BY RepUnfolds ‘‘is_odd divides‘‘ 0 THEN ExR [⌜b-i + 1⌝]
```

Figure 6: Proofs of the requirements for “good” stamps.

```
THM stampspairs_if_greater_two
∀a,b:ℕ.  (2<a ∧ a<b) ⇒
a and b are stamps pairs  ⇒  (a=3 ∧ b=4) ∨ (a=3 ∧ b=5))
BY Auto THEN AssertCases ⌜(a | b+1 ∨ b=a+1) ∧ (a | b+2 ∨ b=a+2)⌝

    .....Assertion 1.....
    a:ℕ, b:ℕ, 2<a, a<b, a and b are stamps pairs ⊢  a | b+1 ∨ b=a+1
    BY StampsInstance 5 ⌜a+b+1⌝ THEN EqChoices [ ⌜j=0⌝; ⌜j=1⌝; ⌜j=2⌝; ⌜j>2⌝]

       .....Case j=0.....
       a:ℕ, b:ℕ, 2<a, a<b, i:ℕ, j:ℕ, a+b+1 = i*a + 0*b  ⊢  a | b+1 ∨ b=a+1
    √ BY orR1 THEN DividesWitness ⌜i - 1⌝
       .....Case j=1.....
       a:ℕ, b:ℕ, 2<a, a<b, i:ℕ, j:ℕ, a+b+1 = i*a + 1*b  ⊢  a | b+1 ∨ b=a+1
    √ BY Assert ⌜a | 1⌝ THENL [DividesWitness ⌜i-1⌝; ILemma ‘divisor_bound‘ [⌜a⌝;⌜1⌝]]
       .....Case j=2.....
       a:ℕ, b:ℕ, 2<a, a<b, i:ℕ, j:ℕ, a+b+1 = i*a + 2*b  ⊢  a | b+1 ∨ b=a+1
    √ BY EqChoices [⌜i=0⌝;⌜i>0⌝]  THENL [Auto’; ILemma ‘mul_bounds_1b‘ [⌜i⌝;⌜a⌝] ]
          % ------------------------------------------------------------------ +
          | The second case uses the inequality ⌜0<i*a⌝ to show a contradiction |
          + ------------------------------------------------------------------ %
       .....Case j>2.....
       a:ℕ, b:ℕ, 2<a, a<b, i:ℕ, j:ℕ, a+b+1 = i*a + j*b, j>2  ⊢  a | b+1 ∨ b=a+1
    √ BY % -------------------------------------------------------------- +
          | Show by a chain of inequalities that there is a contradiction |
          + -------------------------------------------------------------- %
               ILemma ‘mul_bounds_1a‘    [⌜i⌝;⌜a⌝]      % 0 ≤ i*a   %
          THEN ILemma ‘mul_preserves_lt‘ [⌜2⌝;⌜j⌝;⌜b⌝]  % b*2 < b*j %
```

Figure 7: Proofs of the requirements for “good” stamps.

We prove the first claim by instantiating the stamps property for the value $a+b+1$ and then analyze how often b may have been used to create this sum. If b is not used, a must divide $b+1$. If b is used twice, $b=a+1$ must be the case. All other cases are impossible. For the second claim, we use a similar argument, this time with the value $a+b+2$.

Using these two assertions gives us 4 cases, among which the case $b=a+1 \wedge b=a+2$ is impossible. In the other three cases we use the laws of divisibility to prove that $a \mid b+1 \wedge a \mid b+2$ gives us $a=1$ (a contradiction), $a \mid b+1 \wedge b=a+2$ gives us $a=3$ and $b=5$, and $b=a+1 \wedge a \mid b+2$ gives us $a=3$ and $b=4$.

The above proofs rely on a lemmas about multiplication, division, and orders, which can be found in Nuprl’s standard library. The following lemmas were used.

```
THM mul_bounds_1a                   ∀a,b:ℕ.          0 ≤ a*b
THM mul_bounds_1b                   ∀a,b:ℕ+.         0 < a*b
THM mul_preserves_lt                ∀a,b:ℤ.∀n:ℕ+.    a < b ⇒ n*a < n*b
THM mul_preserves_le                ∀a,b:ℤ.∀n:ℕ.     a ≤ b ⇒ n*a ≤ n*b
THM multiply_functionality_wrt_le   ∀i1,i2,j1,j2:ℕ.  i1≤j1 ⇒ i2≤j2 ⇒ i1*i2 ≤ j1*j2
THM div_rem_sum                     ∀a:ℤ.∀n:ℤ-0.     a = (a ÷ n)*n + a rem n
THM divisor_of_sub                  ∀a,b1,b2:ℤ.      a | b1 ⇒ a | b2 ⇒ a | (b1-b2)
THM divisor_bound                   ∀a:ℕ.∀n:ℕ+.      a | b  ⇒  a≤b
THM odd_mul_cancel                  ∀a,b:ℤ.          a*b is odd ⇒ b is odd
```

```
.....Assertion 2.....
a:ℕ, b:ℕ, 2<a, a<b, a and b are stamps pairs  ⊢  a | b+2  ∨  b=a+2
√ BY % --------------------------------------------------------------------- +
    | This is almost identical to Assertion 1, so we do everything at once |
    + --------------------------------------------------------------------- %
          StampsInstance 5 ⌜a+b+2⌝  THEN  EqChoices [ ⌜j=0⌝; ⌜j=1⌝; ⌜j=2⌝; ⌜j>2⌝]
    THENL [ orR1 THEN DividesWitness ⌜i - 1⌝
          ; Assert ⌜a | 2⌝ THENL [ DividesWitness ⌜i-1⌝
                                 ; ILemma ‘divisor_bound‘ [⌜a⌝;⌜2⌝]]
          ; EqChoices [⌜i=0⌝;⌜i>0⌝]
            THENL [ Auto’; ILemma ‘multiply_functionality_wrt_le‘ [⌜1⌝;⌜2⌝;⌜i⌝;⌜a⌝]]
                  % ---------------------------------------------------- +
                  | The second case uses 1*2<i*a to show a contradiction |
                  + ---------------------------------------------------- %
          ;      ILemma ‘mul_bounds_1a‘    [⌜i⌝;⌜a⌝]      % 0 ≤ i*a   %
            THEN ILemma ‘mul_preserves_le‘ [⌜3⌝;⌜j⌝;⌜b⌝]   % b*3 ≤ b*j %
          ]

.....Asserted Case 1.....
a:ℕ, b:ℕ, 2<a, a<b, a | b+1, a | b+2  ⊢  (a=3 ∧ b=4) ∨ (a=3 ∧ b=5)
√ BY % ---------------------------------------------------------------------- +
    | Analyzing Hyps 5 and 6 gives us a=1, which contradicts hypothesis 6   |
    + ---------------------------------------------------------------------- %
         FwdLemma ‘divisor_of_sub‘ [6;5]
    THEN Subst ⌜(b+2 - (b+1))=1⌝ (-1) THENA Auto
    THEN ILemma ‘divisor_bound‘ [⌜a⌝;⌜1⌝]

.....Asserted Case 2.....
a:ℕ, b:ℕ, 2<a, a<b, a | b+1, b=a+2  ⊢  (a=3 ∧ b=4) ∨ (a=3 ∧ b=5)
√ BY % ------------------------------------------- +
    | Analyzing Hyps 5 and 6 gives us  a=3 ∧ b=5 |
    + ------------------------------------------- %
     Assert ⌜a | 3⌝ THENL [ DVars [‘c’] 5 THEN DividesWitness ⌜c - 1⌝
                          ; ILemma ‘divisor_bound‘ [⌜a⌝;⌜3⌝] ]

.....Asserted Case 3.....
a:ℕ, b:ℕ, 2<a, a<b, b=a+1, a | b+2  ⊢  (a=3 ∧ b=4) ∨ (a=3 ∧ b=5)
√ BY % ------------------------------------------- +
    | Analyzing Hyps 5 and 6 gives us  a=3 ∧ b=4 |
    + ------------------------------------------- %
     Assert ⌜a | 3⌝ THENL [ DVars [‘c’] 6 THEN DividesWitness ⌜c - 1⌝
                          ; ILemma ‘divisor_bound‘ [⌜a⌝;⌜3⌝] ]
```

Figure 8: Proof of `stampspairs_if_greater_two` (continued)

The proofs also employ a variety of reasoning *tactics* that were written to make the formal proof comprehensible. Tactics are metalevel programs that control the application of reasoning rules of a fundamental proof calculus. The tactics used in our proofs were written to mimic specific reasoning steps that a human would use in an argument by expressing them in terms of elementary proof rules. Because we chose memnonic names (and used comments in a the proofs), most of them should be self-explanatory.