MATH3349 Special Topics in Mathematics
Interactive Theorem Proving

Ainsley Rose Pahljina (u6045544)
Supervisor: Assoc Prof. Scott Morrison

November 15, 2019

Acknowledgements

I wish to thank Scott Morrison for his assistance and patience in teaching me Lean, and helping to construct this project. In addition, I acknowledge the assistance of Lean users, Mario Carneiro and Bhavik Mehta, who provided assistance with lemmas supplementary to this project.

Abstract

Motivated by an interest in number theory and cryptography, I have attempted to encapsulate the Lucas-Lehmer lemma within Lean. The Lucas-Lehmer lemma is a primality test for the Mersenne numbers. Primality testing offers a computationally efficient alternative to integer factorisation, potentially advantageous to existing tactics. This report will outline the existing mechanism for primality determination in Lean, the formalisation of the Lucas-Lehmer lemma and the associated constraints and arising challenges. The code for this project may be accessed at: `https://gitlab.cecs.anu.edu.au/u6045544/interactive-theorem-proving-week-1/blob/master/src/Project.lean`.

Introduction

Primes in Lean

Introduced by Microsoft Research in 2013, Lean supports interaction and the construction of specified axiomatic proofs, bridging the gap between automated and interactive theorem proving. [1] Fundamentally, Lean is a functional and dependently typed programming language. Primality in lean is constructed as a decidable proposition - a given natural number is determined to be prime or not prime by its integer factorisation. Whilst this works in the kernel, it is far from optimal. To determine whether a number is prime, one may use the norm_num tactic. Amongst other things, this tactic engages primality tests that exist within Lean - including Fermat's Little theorem, Euler's criterion, and Wilson's lemma.

The Mersenne numbers are not currently encapsulated in Lean's library, mathlib. The efficient implementation of the Lucas-Lehmer lemma may supplement existing primality testing. However, the utility of the Lucas-Lehmer lemma as a primality test relies on the recognition of a natural number as Mersenne, which has not been attempted in this implementation. This project has focused on the generation of Mersenne primes by proving this lemma.

Mersenne Numbers

The Mersenne numbers are of the form $2^n - 1$, for $n \in \mathbb{N}$. It was once postulated that $2^n - 1$ is prime, for any prime n. Whilst this was disproved in 1536, the determination of Mersenne primes occurred right up until 1948. [2] As of June 2019 51 Mersenne primes are known, with the largest known prime number recognised to be a Mersenne prime. [2] It is noted that exponentiation is computationally quite expensive, thus the generation of Mersenne numbers to determine primality may not be optimal.

The Mersenne numbers have been defined in Lean for this project, as shown below:

```
def M (p : ℕ) : ℕ := 2^p - 1
```

Formalisation: Lucas-Lehmer primality test

The Lemma

The Lucas-Lehmer lemma is a primality test for the Mersenne numbers. Originally developed by Édouard Lucas in 1856, it was refined by Derrick Henry Lehmer in 1930. [3]

Firstly we consider the Mersenne number, $M_p = 2^p - 1$ with odd prime p. It is noted that it is exponentially more efficient to determine prime p, than to determine M_p as a prime. The lemma defines a sequence, s_i for $i \leq 0$, as:

$$s_i = \begin{cases} 4 & i = 0 \\ s_{i-1}^2 - 2 & \text{otherwise} \end{cases}$$

The lemma states that:

$$M_p \text{ is prime} \iff s_{p-1} = 0 \pmod{M_p} \text{ with } s_i \text{ and } M_p \text{ defined as above.}$$

This project has attempted only to establish sufficiency, that $s_{p-2} = 0 \pmod{M_p}$ implies that M_p is prime. This offers greater utility in prime generation when compared to the reverse implication, that M_p is prime necessarily implies that $s_{p-2} = 0 \pmod{M_p}$.

In Lean, s_i was defined slightly differently, performing the modular operation as each iteration as to enhance computational efficiency. However in establishing that properties associated with s_i, we must consider the recurrence relation without taking modulo M_p.

The equivalence between the recurrence relations, one taken with iterative modular arithmetic and the other with a singular modular application, has been constructed but not proved below. This may be proved inductively, however would be quite algebraically involved in Lean. The Lucas Lehmer residue of p, defined as $s_{p-2} \bmod M_p$, is similarly encapsulated in Lean below.

```
 -- definition of recurrence relation
 def s : ℕ → ℕ
| 0 := 4
| (i+1) := ((s i)^2 - 2)
```

```
  -- definition of recurrence relation, taking mod M_p iteratively
def s' (p : ℕ ) : ℕ → ℕ
| 0 := 4
| (i+1) := ((s' i)^2 - 2) % (2^p - 1)

def Lucas_Lehmer_residue (p : ℕ) := s' p (p - 2) % (2^p - 1)

lemma s_eq_s' (p : ℕ) (i : ℕ) : (s i) % (2^p - 1) = s' p i := sorry
```

The Lucas-Lehmer residue, and its equality or inequality with 0, is a decidable proposition. The proposition of sufficiency draws on this decidable procedure for the Lucus Lehmer residue, and where successful for prime p, confirms that M_p is indeed prime.
This is illustrated in the below example, where M_5 is proved to be prime, employing norm_num to confirm 5 is indeed prime and greater than 2, and using our decidable proposition with the tactic dec_trivial.

```
instance : decidable_pred Lucas_Lehmer_test := λ p,
by { dsimp[Lucas_Lehmer_test], apply_instance }

theorem Lucas_Lehmer_sufficiency (p : ℕ) (pp : prime p) (h : p > 2) (w:
   Lucas_Lehmer_test p) : prime (M p) :=
begin
dsimp[Lucas_Lehmer_test] at w,
apply final p,
exact w,
end
M
example : prime (M 5) := Lucas_Lehmer_sufficiency _ (by norm_num)
(by norm_num) dec_trivial
```

With this end goal established, we may now consider how sufficiency is proved in Lean.

Proof

Closed-form solution

The proof initially requires the closed form solution for our defined recurrence relation, s_i. For notational ease, let $\omega = 2 + \sqrt{3}$ and $\bar{\omega} = 2 - \sqrt{3}$. Inductively it may be proved that

$$s_i = \omega^{2^i} + \bar{\omega}^{2^i} \text{ for all i}$$

In considering the base case:

$$s_0 = \omega^{2^0} + \bar{\omega}^{2^0} = 2 + \sqrt{3} + 2 - \sqrt{3} = 4$$

With inductive hypothesis, $s_n = \omega^{2^n} + \bar{\omega}^{2^n}$, we have that:

$$\begin{aligned} s_{n+1} = {s^2}_n - 2 & \\ &= (\omega^{2^n} + \bar{\omega}^{2^n})^2 - 2 \\ &= (\omega^{2^n)^2} + (\bar{\omega}^{2^n)^2} + 2(\bar{\omega}\omega)^{2^n} - 2 \\ &= \omega^{2^{n+1}} + \bar{\omega}^{2^{n+1}} + 2(1)^{2^n} - 2 \\ &= \omega^{2^{n+1}} + \bar{\omega}^{2^{n+1}} \end{aligned}$$

The proof for the Lucas Lehmer lemma is ultimately achieved by contradiction. To derive this contradiction, we suppose that M_p is in fact composite and consider q, defined as the minimum prime factor of M_p. q is defined in Lean as a positive natural number, using existing definitions of the minimum factor.

```
def q (p :ℕ) : ℕ+ := ⟨min_fac (M p), by exact min_fac_pos (M p)⟩
```

Returning to the our closed form solution for s_i; the desired contradiction arises by considering the following group defined as:

$$X = \{a + b\sqrt{3} \mid a,\ b \in \mathbb{Z}_q\}$$

The integers modulo q ($\mathbb{Z}_q$) are defined within Lean. Thus in Lean, this group has been defined by the following:

```
def X (q : ℕ+) := (zmod q) × (zmod q)
```

In translating the inductive proof of the closed form solution into Lean, the arguments of proposition are considered within the group, X. This is necessary as $\omega = 2 + \sqrt{3}$ and $\bar{\omega} = 2 - \sqrt{3}$, are defined within this group as shown below.

```
def ω : X q := (2, 1)
def ωb : X q := (2, -1)
```

Since our recurrence relation was initially defined in the natural numbers, this ultimately caused challenges in coercing statements between $\mathbb{N}$ and X. In addition, the inductive step of this proof relies on subtleties of index arithmetic, which proved to be intensive in Lean without other supporting lemmas. The proof below still requires the facts referenced in the comments to be completed.

```
lemma closed_form (i : ℕ) : (s i : X q) = (ω : X q)^(2^i) + (ωb : X q)^(2^i)
:= begin induction i with i ih,
{dsimp[s, ω, ωb],  /- base case -/
simp, ring, ext, simp, simp, },
{calc (s (succ i) : X q)  = (((s i)^2 -2) : ℕ) : begin dsimp[s], refl, end
            ... = ((((s i) : X q) ^2 -2)) : sorry
                /- the coercion from ↑(s i^2-2) to ↑(s i)^2-2 -/
            ... = (ω^(2^(i)) + ωb^(2^(i)))^2 - 2 : by rw ih
```

```
                ... = ω^(2^(i))^2 + ωb^(2^(i))^2
                      + 2*(ωb)^(2^(i))*(ω)^(2^(i)) - 2 : sorry
                    /- (a + b)^2 = a^2 + b^2 + 2*a*b -/
                ... = ω^(2^(i))^2 + ωb^(2^(i))^2 + 2*(ωb*ω)^(2^(i)) - 2 : sorry
                ... = ω^(2^(i))^2 + ωb^(2^(i))^2 + 2*(1)^(2^(i)) - 2
                            : by rw inverse_val
                ... = ω^(2^(i))^2 + ωb^(2^(i))^2 + 2*(1) - 2
                            : by simp only[_root_.one_pow]
                ... = ω^(2^(i))^2 + ωb^(2^(i))^2
                            : by simp only[mul_one, add_sub_cancel]
                ... = ω^(2^(i)*2) + ωb^(2^(i)*2) : sorry
                    /- (2^(2^i))^2 = (2^((2^i)+(2^i)) = 2^(2*(2^i)) -/
                ... = ω^(2^(succ i)) + ωb^(2^(succ i)) : sorry
                    /- 2^i * 2 = 2^(i+1) -/
 }
end
```

X as a ring

To verify that X is a ring in Lean, the requisite substructures were constructed. This included establishing X as an additive, commutative group with addition defined as:

$$(a + b\sqrt{3}) + (c + d\sqrt{3}) = (a + c) \bmod q + (b + d)\sqrt{3} \bmod q$$

This also included establishing X as a monoid, with multiplication defined as:

$$(a + b\sqrt{3})(c + d\sqrt{3}) = (ac + 3bd) \bmod q + (ad + bc)\sqrt{3} \bmod q$$

In Lean, an extensionality lemma was essential to prove the properties of multiplication and addition, since component-wise these properties are satisfied under the ring $z\text{mod } q$. This is depicted below:

```
 -- Defining multiplication

 instance : has_mul (X q) :=
{ mul := λ x y, (x.1*y.1+3*x.2*y.2,x.1*y.2+x.2*y.1) }

-- Defining addition

instance : has_add (X q) :=
{ add := λ x y, (x.1+y.1,x.2+y.2) }

-- Establishing X q as a additive, commutative group

instance : add_comm_group (X q) := begin
```

```
  dsimp[X],
  apply_instance,
end

-- Establishing X q as a monoid

instance : monoid (X q) :=
{ mul_assoc := λ x y z, by simp[multiplicative_assoc],
  one := ⟨1,0⟩,
  one_mul := λ x, begin ext; simp, end,
  mul_one := λ x, begin ext; simp, end,
  ..(infer_instance : has_mul (X q)) }

-- Establishing X q as a ring

instance : ring (X q) :=
{ left_distrib := left_distributive,
  right_distrib := right_distributive,
  ..(infer_instance : add_comm_group (X q)),
  ..(infer_instance : monoid (X q))}
```

Units in X

We consider ω as an invertible element of X. Although ω has already been defined, in Lean we must specify that ω belongs to the units of X. Lean requires the value, its inverse and the proof of commutative invertibility, $\omega\bar{\omega} = \bar{\omega}\omega = 1$, as shown below:

```
def omega_unit (p : ℕ) : units (X (q p)) :=
{val := ω,
inv := ωb,
val_inv := by simp[val_inverse],
inv_val := by simp[inverse_val],
}
```

Cardinality

The cardinality of X, defined as $z\text{mod } q \times z\text{mod } q$, is exactly q^2. Prior to proving this, Lean had to confirm that $X\ q$, is indeed a finite group. Lean was able to deduce this, using $z\text{mod } q$ as a finite group. The cardinality of $z\text{mod } q$ as q was similarly referenced, to attain this desired conclusion as shown below.

```
 /- X is a finite group -/
instance : fintype (X q) :=
begin
  dsimp [X],
  apply_instance,
end
```

```
 -- The cardinality of the group, z mod q, is q.
lemma fin_zmod : fintype.card (zmod q) = q := begin
exact zmod.card_zmod q,
end
```

The more challenging case however is to determine the cardinality of the units of X. For notational simplicity, let $X^\star$ denote the group of units. As described previously, this initially involves proving that the $X^\star$ is a finite group. Whilst this is a logical and somewhat obvious conclusion given that X is finite, this fact is not trivial in Lean. Using the injectivity of the coercion, $X^\star \to X$, Lean recognises there are finitely many elements in $X^\star$. Lean warns that this list of elements is non-computable. However, for the purpose of this proof we only require the fact that $X^\star$ is finite, thus we may mark the instance as non-computable.

```
noncomputable instance fintype_units : fintype (units (X q)) :=
begin
apply fintype.of_injective (coe : units (X q) → (X q)),
exact units.ext,
end
```

With this fact, we now consider that $\mid X^\star \mid$ must be strictly less than $\mid X \mid$, since 0 is one known element without an inverse (i.e. $0 \in X$, but $0 \notin X^\star$). This proof has been constructed but not implemented, as shown below:

```
lemma units_card : fintype.card (units (X q)) < q^2 := sorry
```

The important take away from this section is that $\mid X^\star \mid < q^2$.

Results derived from the Lucas Lehmer test

We recall that the Lucas Lehmer test established whether the Lucas Lehmer residue is congruent to 0 mod M_p. Upon satisfying this test - we may use the definition of s_i and modular arithmetic to conclude that:

$$\begin{aligned}
s_{p-2} \equiv 0 \bmod M_p &\implies \omega^{2^{p-2}} + \bar{\omega}^{2^{p-2}} = kM_p \text{ (for some integer k)} \\
&\implies \omega^{2^{p-2}} = kM_p - \bar{\omega}^{2^{p-2}} \ (2) \\
&\implies (\omega^{2^{p-2}})^2 = (kM_p - \bar{\omega}^{2^{p-2}}) \times \omega^{2^{p-2}} \ (3) \\
&\implies \omega^{2^{p-1}} = kM_p\omega^{2^{p-2}} - \bar{\omega}\omega^{2^{p-2}} \ (4) \\
&\implies \omega^{2^{p-1}} = kM_p\omega^{2^{p-2}} - 1 \ (5)
\end{aligned}$$

In Lean, the desired property of modular congruency was not directly accessible in mathlib, and thus Scott Morrison developed the below lemma to be used. As the proof relies on s_i, defined in $\mathbb{N}$, a coercion was again required to achieve the result in X. The initial hypothesis that the Lucas Lehmer residue is congruent to 0, was shown to be definitionally equivalent to step 2 above. However the remaining algebraic manipulation in steps (3), (4) and (5) has not being completed. The existing code below with reference to the above reasoning will hopefully assist anyone who intends to complete this proof in the future.

```
 -- Credit: Scott Morrison
lemma multiple_mod {a b : ℕ} (h : a % b = 0) : ∃ k, k * b = a :=
⟨a / b, nat.div_mul_cancel (nat.dvd_of_mod_eq_zero h)⟩

theorem multiple_k (p : ℕ) (h : Lucas_Lehmer_residue p = 0) :
∃ (k : ℕ), (ω : X (q p))^(2^(p-1)) = k*(M p)*((ω : X (q p))^(2^(p-2)))-1 :=
   begin
dsimp[Lucas_Lehmer_residue] at h,
rewrite ← s_eq_s' at h,
simp at h,
replace h := multiple_mod h, -- using the above lemma
cases h with k h,
use k,
replace h := congr_arg (λ (n : ℕ), (n : X (q p))) h, -- coercion from ℕ to X
   q
dsimp at h,
rewrite closed_form at h,
dsimp[M],
sorry
end
```

We now consider that:

$$kM_p\omega^{2^{p-2}} \equiv 0 \bmod q.$$

This arises from the very definition of q, as the minimum factor of M_p. As a multiple of q, M_p evaluates to 0 mod q.

In Lean, M_p exists within X. Thus we wish to use extensionality to consider M_p mod q, and definitionally simplify M_p as a multiple of q. Upon difficulties initially with extensionality and coercion, the latter has not being defined within this attempt, and thus this proof has been constructed without verification.

```
theorem modulo_q (p : ℕ ) : ((M p) : X (q p)) = 0 := sorry
```

In continuing from (5) and using the above fact we have:

$$\omega^{2^{p-1}} = -1$$

This is encapsulated in Lean by the following lemma. Given the simplicity of the implication, this may be implemented more succinctly in future.

```
theorem sufficiency_simp (p : ℕ) (h : Lucas_Lehmer_residue p = 0) : (ω : X (q
   p))^(2^(p-1)) = -1 :=
begin
have w := multiple_k p h,
cases w with k w,
rewrite[modulo_q] at w,
```

```
simp at w,
exact w,
end
```

So now in squaring both sides of the equation we obtain:

$$\omega^{2^p} = 1$$

This is similarly established in Lean by the below lemma. However it is noted that Lean did not recognise the simplification: $\omega^{2^{p-2}} \times \omega^{2^{p-2}} = \omega^{2^p}$. A lemma for this was not constructed, however it seems that recognised power operations would be useful additions to mathlib. The proof was constructed in a calc block, with readable progressions and reasoning below. This similarly may be achieved in a more concise manner.

```
theorem suff_squared (p : ℕ) (h : Lucas_Lehmer_residue p = 0) :
(ω: X (q p))^(2^p) = 1 := begin
calc (ω : X (q p))^2^p = (ω^(2^(p-1)))*(ω^(2^(p-1)))
         : sorry  /- required lemma -/
         ... = (-1) * (ω^(2^(p-1)))
         : begin rw sufficiency_simp _, exact h, end
         ... = (-1) * (-1)
         : begin rw sufficiency_simp _, exact h, end
         ... = -1 * -1 : by ring
         ... = 1 : by simp
end
```

Order of ω

We now observe that ω^{2^p} lies within $X^\star$ with inverse $\omega^{2^{p-1}}$ (since $\omega^{2^p}\omega^{2^{p-1}} = \omega^{2^{p-1+1}} = \omega^{2^p} = 1$). From this, we conclude that the order of ω must divide 2^p. Now since the order does not divide 2^{p-1}, we conclude that the order must be exactly 2^p. These two lemmas were written externally, with the latter relying on an inequality property of divisible prime powers depicted below. To maintain relevance to the given proof the code for these lemmas has been excluded however, they are available within the file. This lemma in turn relied on establishing the power operation as a monotonic injective function, which has been provided with the assistance of Bhavik Mehta.

```
lemma prime_pow_dvd_prime_pow {p k l : ℕ} (pp : prime p) : p^k ∣ p^l ⟺ k ≤ l
    := unspecified

lemma dvd_prime_power (a p k : ℕ) (pp : prime p) (h₁ : ¬(a ∣ p^k))
(h₂ : a ∣ p^(k+1)) : a = p^(k+1) := unspecified
```

To derive the conclusion that the order of ω must divide 2^p from $\omega^{2^p} = 1$, an additional lemma was constructed. This was similarly produced by Scott Morrison, and would be useful to add to mathlib. The lemma is constructed below, and similarly available in complete form in the file.

```
lemma order_of_dvd_iff {n : ℕ} {g : G} : order_of g ∣ n ⟺ g^n = 1 :=
    unspecified
```

Now, in establishing that the order of ω is 2^p in Lean, the proof initially engages the above lemma to generate two goals: (1) ω divides 2^p (2) ω does not divide 2^{p-1}. The first goal follows very easily from the defined order lemma above, with the direct application of $\omega^{2^p} = 1$. To address the second goal, we attempt to derive a contradiction. The contradiction arises from the previously proven fact that $\omega^{2^{p-1}} = -1$. To apply this fact, we had to coerce omega unit to ω in X. This was achieved by a supporting lemma, utilising the cong_arg tactic. In applying this fact in conjunction with our desired order property, we obtain $1 = -1$. This does not trivially generate a contradiction, since the equality is inside X. We reason that for the equality to hold, $2 = 0 \mod q$ and therefore $q = 1$ or $q = 0$. This implication itself was not easily formulated in Lean, and required repeated case analysis. When introduced into the updated Lean version, the tactic nat_cases would be able to replace these cases.

```
lemma mod_eq_one_or_two (q : ℕ+) (h : (2 : zmod q) = 0) : q = 1 ∨ q = 2 :=
begin
  have w : (2 : zmod q) = (2 : ℕ) := begin apply fin.eq_of_veq, simp, end,
  rw w at h,
  replace h := congr_arg fin.val h,
  rw zmod.val_cast_nat at h,
  simp at h,
  by_cases t : q ≤ 2,
  { -- case bashing
    rcases q with ⟨q,qp⟩,
    cases q, cases qp,
    cases q, left, refl,
    cases q, right, refl,
    cases t, dsimp at t_a, cases t_a, cases t_a_a, },
  { simp at t,
    have t' : 2 % ↑ q = 2 := mod_eq_of_lt t,
    rw t' at h,
    cases h, }
end
```

Now in establishing that $q = 1$ or $q = 0$, we generate a contradiction by realising that $q > 2$. This was attempted by considering cases for $q = 0$, $q = 1$ and $q = 2$, and deriving a contradiction for each. Without the nat_cases tactic, this proved to be quite an intensive method. Thus, this proof still remains to be completed (see minimum_q). Finally, using this inequality we conclude that $q \neq 1$ or $q \neq 2$ which would clearly contradict our existing hypothesis. It was surprisingly challenging to combine these hypothesises to assert this contradiction. The proof as described is visible below.

```
theorem order_omega (p : ℕ) (h : 0 < p) (h_2 : Lucas_Lehmer_residue p = 0):
    order_of (omega_unit p) = 2^p :=
begin
  have t : p = p - 1 + 1 := (nat.sub_eq_iff_eq_add h).mp rfl,
  conv { to_rhs, rw t },
  apply dvd_prime_power, -- generate two goals
  norm_num,
```

```
  { intro h, -- goal 1  ω divides 2^p
    have w := order_of_dvd_iff.1 h,
    have w' := coercion _ w, -- coercion from omega unit to X q
    have h' := sufficiency_simp p h_2,
    have h_2 := (w'.symm).trans h',
    have h_3 := congr_arg (prod.fst) h_2,
    have h_4 : (1 : zmod (q p)) = -1 := h_3,
    have h_5 : (2 : zmod (q p)) = 0 :=
      calc 2 = ((1 + 1) : zmod((q p)))  : rfl
         ... = ((1 + -1) : zmod((q p))) : by rw ← h_4
         ... = 0 : add_neg_self 1,
    have h_6 : q p = 1 ∨ q p = 2 := mod_eq_one_or_two _ h_5,
    have h_7 :  2 < q p := minimum_q,
    have h_8 : 2 ≠ q p := ne_of_lt h_7,
    have h_9 : 1 ≠ q p := begin by_contradiction, simp at a, sorry end,
    sorry
     },
     -- goal 2  ω does not divide 2^(p-1)
  { rw ← t, apply order_of_dvd_iff.2, apply units.ext, simp, apply (
    suff_squared _), exact h_2 }
end
```

Contradiction

Now, it is known that the order of an element is at most the size of the group. Therefore, since omega is contained in $X^\star$:

$$| \omega | = 2^p \leq | X^\star | < q^2$$

In Lean, this property was sourced from mathlib (order_of_le_card_univ) and thus this fact could be easily constructed in a calc block:

```
lemma order_ineq (p : ℕ) (h : 0 < p) (h_2 : Lucas_Lehmer_residue p = 0):
2^p < (qp : ℕ)^2 :=
calc 2^p = order_of (omega_unit p) : (order_omega _ h h_2).symm
     ... ≤ fintype.card _          : order_of_le_card_univ
     ... < (q p : ℕ)^2             : units_card
end residue_zero
```

Since q is the smallest prime factor of M_p (assumed to be composite). We also know that:

$$q^2 < M_p = 2^p - 1$$

This was generalised in Lean, so that for any minimum factor of natural number n it is known that its square is less than or equal to n. This drew upon existing prime documentation, and was similarly achieved through a calc block.

```
lemma min_fac_sq (n : ℕ) (h_1 : 0 < n) (h : ¬ prime n) : (min_fac n)^2 ≤ n :=
begin
  have t : (min_fac n) ≤ (n/min_fac n) := min_fac_le_div h_1 h,
  calc
   (min_fac n)^2 = (min_fac n) *(min_fac n) :  by exact pow_two (min_fac n)
             ... ≤ (n/min_fac n)* (min_fac n) : by exact mul_le_mul_right (
    min_fac n) t
             ... ≤ n : by exact div_mul_le_self n (min_fac n)
end
```

In combining the above inequalities we ultimately obtain:

$$2^p < 2^p - 1$$

Which generates a contradiction, and therefore we conclude the M_p must be prime.

This is summarised in Lean by the desired implication that $s_{p-2} = 0 \pmod{M_p} \implies M_p$ is prime. In discounting the possibility that $p = 0$, we attempt to derive the above contradiction. Using the inequalities defined above, we are able to derive an inequality contradiction as described.

```
theorem final (p : ℕ) : Lucas_Lehmer_residue p = 0 → prime (M p) :=
begin
by_cases w : 0 < p,
{ contrapose,  -- case p > 0
  intro,
  intro t,
  have h_1 := order_ineq p w t,
  have h_2 := min_fac_sq (M p) (M_pos w) a,
  dsimp [q] at h_1,
  have h := lt_of_lt_of_le h_1 h_2,
  dsimp [M] at h,
  exfalso,
  exact inequality_contradition h, },
{ simp at w, subst w, intro h, cases h, } -- case p = 0
end
```

This ultimately completes the proof of sufficiency, as described initially.

Conclusion

The sufficiency implication of the Lucas-Lehmer lemma has been constructed to supplement existing primality testing. The challenges and constraints faced in formalising this lemma have been discussed for each component of the proof. Overall however, it was also challenging to ensure all the requisite hypothesis were carried through the proof and supporting lemmas. In many instances, it is acknowledged that the proof remains to be completed or could be further optimised. However, the existing framework and rationale will hopefully support refined implementations of this lemma in future.

References

[1] Jeremy Avigad, Leonardo de Moura, and Soonho Kong, *Theorem Proving in Lean*, `https://leanprover.github.io/theorem_proving_in_lean/introduction.html` 2017.

[2] Chris K. Caldwell, *Mersenne Primes: History, Theorems and Lists*, `https://primes.utm.edu/mersenne/`

[3] Tony Reix, *A LLT-like test for proving the primality of Fermat numbers*, `https://arxiv.org/abs/0705.3664` 2007.