Carmichael Numbers in Lean 4

Matthew Hough

December 8, 2023

1 Motivation

Carmichael numbers are composite numbers for which there exists no Fermat witness [1]. Given a composite number n, an integer a is a Fermat witness for n if

$$a^n \equiv a \pmod{n}.$$

To the best of my knowledge, Carmichael numbers have not been formalized in Lean. In fact, Lean's Mathlib specifically states[1] the lack of formalization of Carmichael numbers. With this in mind, I decided to learn Lean by defining Carmichael numbers in the prover and formalizing some theorems about them.

2 Project structure

The formalization of Carmichael numbers in Lean is fully typed out in the included Lean project. See the file titled `Carmichael.lean` for the working code corresponding to this section. The file structure of the project is as follows:

```
carmichael
├── Carmichael
│   └── Carmichael.lean
├── .lake
├── Carmichael.pdf
├── lakefile.lean
├── lake-manifest.json
└── lean-toolchain
```

The folder `Carmichael` is where my work lies. The file `lakefile.lean` contains the config that lake ("lean make") needs to build the Lean application, `lake-manifest.json` keeps track of the versions of the dependencies used in the project, while `lean-toolchain` contains a single line which specifies the Lean build used in the project. There is also a hidden folder `.lake` which contains the installed dependencies.

[1] `https://leanprover-community.github.io/mathlib4_docs/Mathlib/NumberTheory/FermatPsp`

3 Dependencies

My work in this project is dependent only on the formalizations defined in `Mathlib.FieldTheory.Finite.Basic`.

4 Formalization

In `Carmichael.lean`, we first define what it means to be a Carmichael number. We give the definition below and then the formalization of that definition in Lean.

Definition 1. *A Carmichael number is a composite number n for which the congruence relation*

$$a^n \equiv a \pmod{n}$$

holds for all integers a.

Formalization 1.

```
def CarmichaelNumber (n : ℕ) : Prop :=
  (∀ (a : ℤ) , a.pow n ≡ a [ZMOD n]) ∧ ¬ Nat.Prime n ∧ (n > 0)
```

Before we write out any theorems, we need to prove a lemma. The following lemma will be needed in the proof of Theorem 1.

Lemma 1. *Suppose n is a natural number and m, k are integers. If* $\gcd(m, k) = 1$*, then* $\gcd(m^n, k) = 1$*.*

Formalization 2.

```
lemma Int.Coprime.pow_left (n : ℕ) {m k : ℤ} (h : Int.gcd m k = 1) : Int.gcd
 (Int.pow m n) k = 1 :=
  Int.isCoprime_iff_gcd_eq_one.1 (Int.isCoprime_iff_gcd_eq_one.2 h).pow_left
```

Now we can prove two theorems about Carmichael numbers.

Theorem 1. *If n is a Carmichael number, for every integer a coprime to n, we have*

$$a^{n-1} \equiv 1 \pmod{n}.$$

Formalization 3.

```
theorem carmichael_sub_one {n : ℕ} (hc : CarmichaelNumber n) : ∀ (a : ℤ) (h :
 Int.gcd a n = 1),
(a.pow (n - 1) ≡ 1 [ZMOD n]) := by
  intro a h
  nth_rewrite 1 [← Nat.div_one n]
  apply Int.Coprime.pow_left n at h
  rw [Int.gcd_comm] at h
```

```
  nth_rewrite 1 [← h]
  apply @Int.ModEq.cancel_right_div_gcd n (Int.pow a (n-1)) 1 (Int.pow a n) _ _
  · have := hc.2.2.lt
    linarith [this]
  rw [one_mul,Int.ModEq,hc.1,Int.mul_emod,hc.1,← Int.mul_emod,pow_eq]
  nth_rewrite 2 [← pow_one a]
  rw [← pow_add, add_comm, add_comm, ← Nat.sub_add_comm,
 Nat.add_one_sub_one, ← Int.ModEq, ← pow_eq]
  apply hc.1
  apply hc.2.2
```

Theorem 2. *All Carmichael numbers are odd.*

Formalization 4.

```
theorem carmichael_num_is_odd {n : ℕ} (hc : CarmichaelNumber n) : Odd n := by
  by_contra! n_even
  have n_sub_pow_cong : Int.pow (n-1) n ≡ 1 [ZMOD n]
  · apply Int.ModEq.trans (@Int.ModEq.pow n (n-1) (-1) _ _)
    · rw [(neg_one_pow_eq_one_iff_even _).2 (Nat.even_iff_not_odd.2 n_even)]
      simp
    rw [Int.modEq_iff_add_fac]
    refine' ⟨-1, by simp [sub_add_eq_add_sub]⟩
  have := hc.1 (n-1)
  have := this.symm.trans n_sub_pow_cong
  obtain (a | b) := le_or_gt n 2
  · obtain (rfl | lt) := eq_or_lt_of_le a
    · apply hc.2.1
      exact Nat.prime_two
    · obtain (rfl | rfl) := Nat.le_one_iff_eq_zero_or_eq_one.1 (Nat.lt_succ.1
 lt)
      · apply hc.2.2.ne rfl
      apply n_even odd_one
  have hf := Int.ModEq.eq this
  rw [Int.emod_eq_of_lt, Int.emod_eq_of_lt] at hf
  · apply ne_of_gt b
    linarith
  · linarith
  · linarith
  · linarith
  linarith
```

References

[1] Jeffrey Hoffstein J.H. Silverman, Jill Pipher. *An Introduction to Mathematical Cryptography*. Springer New York, 2008.