Formalização do Algoritmo RESSOL utilizando Coq

Bruno Rafael dos Santos
Universidade do Estado de Santa Catarina
bruniculos2014@gmail.com

Orientadora: Dra Karina Girardi Roggia
Coorientador: Me Paulo Henrique Torrens

29/11/2024

Sumário

Sumário

Introdução

- A Teoria dos Números é um ramo da matemática que lida, em sua maior parte, com propriedades de números inteiros;
- É muito presente em temas relacionados a criptografia;
- Envolve definições de diversas relações em $\mathbb{Z}$, sendo duas dessas as relações de divisibilidade e congruência;

- Neste contexto que se apresenta o *símbolo de Legendre*, o qual possui relação com o algoritmo *RESSOL* e está presente na *Lei de Reciprocidade Quadrática*;
- O algoritmo *RESSOL* está relacionado a sistemas de criptografia que utilizam curvas elípticas de acordo com (SARKAR, 2024), (KUMAR, 2020) e (LI; DONG; CAO, 2014);
- A *Lei de Reciprocidade Quadrática* tem aplicações em *zero-knowledge proofs* conforme (WRIGHT, 2016) e pode-ser utilizada para tornar o algoritmo *RESSOL* mais eficiente como apresentado em (COOK, 2023);

Definição 1 (*Divisibilidade*)

$\forall d, a \in \mathbb{Z}$, *d* ***divide*** *a (ou em outras palavras: a é um múltiplo de d) se e somente se a seguinte proposição é verdadeira:*

$$\exists q \in \mathbb{Z}, a = d \cdot q$$

assim, se tal proposição é verdadeira e portanto d divide a, tem-se a seguinte notação que representa tal afirmação:

$$d \mid a$$

caso contrário, a negação de tal afirmação (d não divide a) é representada por:

$$d \nmid a$$

Definição 2 (*Congruência*)

Para todo $a, b, n \in \mathbb{Z}$, *a é congruente a b módulo n se e somente se, pela divisão euclidiana* $\frac{a}{n}$ *e* $\frac{b}{n}$ *(onde* $0 \leq r_a < |n|$ *e* $0 \leq r_b < |n|$*) tem-se*

$$a = n \cdot q_a + r_a$$

e

$$b = n \cdot q_b + r_b$$

com $r_a = r_b$, *o que também equivale a dizer que:*

$$n \mid a - b$$

tal relação entre os inteiros a, b e n é representada por:

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

Objetivo Geral

Implementar o *símbolo de Legendre* e realizar a formalização de suas propriedades (apresentadas em (BROCHERO et al., 2013)) e da corretude (da função que o implementa).

Objetivos Específicos

1. Obter conhecimentos avançados sobre o assistente de provas *Coq*.
2. Realizar o estudo sobre as principais documentações da biblioteca Mathematical Components.
3. Desenvolver a capacidade de realizar provas em *Coq* utilizando as táticas da linguagem de provas *SSReflect*.
4. Estudar conteúdos de Teoria dos Números relacionados ao símbolo de Legendre.
5. Implementar uma função que compute o valor do *símbolo de Legendre* e provar a corretude da mesma se utilizando da biblioteca Mathematical Components.
6. Provar teoremas úteis para manipulação de expressões envolvendo o símbolo de Legendre utilizando-se da biblioteca Mathematical Components.

A seguir serão apresentados os principais teoremas, lemas e definições considerados úteis para a realização do objetivo estabelecido. Esse conteúdo se baseia no livro (BROCHERO et al., 2013) e no que já havia sido implementado na biblioteca Mathematical Components até o momento de realização deste trabalho.

Propriedades de Congruência

1. (*Reflexividade*) $a \equiv a \pmod{n}$
2. (*Simetria*) $a \equiv b \pmod{n} \Longrightarrow b \equiv a \pmod{n}$
3. (*Transitividade*)

$$a \equiv b \pmod{n} \ \land \ b \equiv c \pmod{n} \Rightarrow a \equiv c \pmod{n}$$

4. (*Compatibilidade com a soma*)

$$a \equiv b \pmod{n} \land c \equiv d \pmod{n} \Rightarrow a+c \equiv b+d \pmod{n}$$

5. (*Compatibilidade com a diferença*)

$$a \equiv b \pmod{n} \land c \equiv d \pmod{n} \Rightarrow a-c \equiv b-d \pmod{n}$$

6. (*Compatibilidade com o produto*)

$$a \equiv b \pmod{n} \land c \equiv d \pmod{n} \implies a \cdot c \equiv b \cdot d \pmod{n}$$

A partir dessa propriedade, note que, para todo $k \in \mathbb{N}$:

$$a \equiv b \pmod{n} \implies a^k \equiv b^k \pmod{n}$$

7. (*Cancelamento*)

$$\operatorname{mdc}(c, n) = 1 \implies (a \cdot c \equiv b \cdot c \pmod{n} \iff a \equiv b \pmod{n})$$

Motivação sobre a resolução de congruências de grau 2:

- Sendo p um número primo maior que 2 e $a, b, c \in \mathbb{Z}$ números não divisíveis por p, como motivação suponha que se deseje resolver a seguinte equação:

$$a \cdot x^2 + b \cdot x + c \equiv 0 \pmod{p} \tag{1}$$

  Manipulando essa equação por meio das propriedades de congruência se obtém:

$$(2 \cdot a \cdot x + b)^2 \equiv b^2 - 4 \cdot a \cdot c \pmod{p} \tag{2}$$

Motivação sobre a resolução de congruências de grau 2: (continuação)

- Realizando a substituição $X = 2 \cdot a \cdot x + b$ e $d = b^2 - 4 \cdot a \cdot c$ na Equação 2, tem-se:

$$X^2 \equiv d \pmod{p} \tag{3}$$

  Portanto, resolver a Equação 1 é equivalente a resolver a Equação 3.

Sobre a Equação 3, se diz que d é um quadrado perfeito em $\mathbb{Z}/(p)$ e também que d é um *resíduo quadrático módulo p*.

Conforme (BROCHERO et al., 2013), existem $\frac{p+1}{2}$ resíduos quadráticos módulo p, que são:

$$0^2 \bmod p, 1^2 \bmod p, 2^2 \bmod p, ..., \left(\frac{p-1}{2}\right)^2 \bmod p \qquad (4)$$

pois note que, para todo $x \in \mathbb{Z}$ existe algum $i \in [0, \frac{p-1}{2}]$ tal que $x \equiv i \pmod p$ ou $x \equiv -i \pmod p$, logo $x^2 \equiv i^2 \pmod p$ (usando a propriedade do Item 6) e i^2 está na Lista 4.

Além disso, todos os os valores na Lista 4 são distintos em módulo p, pois para todo $i, j \in \left[0, \frac{p-1}{2}\right]$:

$$\begin{aligned} i^2 \equiv j^2 \pmod{p} &\iff p \mid (i^2 - j^2) && (5) \\ &\iff p \mid (i - j) \cdot (i + j) && (6) \\ &\iff p \mid (i - j) \vee p \mid (i + j) && (7) \end{aligned}$$

Com isso, dado o intervalo de i e j, então $0 \leq i + j \leq p - 1$, assim existem as seguintes possibilidades:

1. $i = j = 0$ e portanto $i \equiv j \pmod{p}$;
2. $0 < i + j \leq p - 1$, portanto $p \nmid i + j$, e então pela disjunção em 7 resta que $p \mid (i - j)$, o que equivale a $i \equiv j \pmod{p}$, ou seja, i é igual j módulo p se e somente se seus quadrados também são.

Com essas conclusões (de que a Lista 4 contém todos os resíduos quadráticos módulo p e que todos os valores dela são distintos em módulo p) pode ser provado o seguinte lema:

Lema 1

Seja $p > 2$ um número primo, existem exatamente $\frac{p+1}{2}$ resíduos quadráticos módulo p e $\frac{p-1}{2}$ resíduos não quadráticos módulo p.

Definição 3 (Símbolo de Legendre)

Seja $p > 2$ um número primo e $a \in \mathbb{Z}$, se define o símbolo de Legendre por:

$$\left(\frac{a}{p}\right) = \begin{cases} 1, \text{ se } p \nmid a \text{ e } a \text{ é um resíduo quadrático módulo } p \\ 0, \text{ se } p \mid a \\ -1, \text{ caso contrário } (a \text{ não é um resíduo quadrático}) \end{cases}$$

Pode-se dizer que a relação entre o *símbolo de Legendre* e função φ de Euler, que para um número primo p é $\varphi(p) = p - 1$, se dá pelo seguinte teorema:

Teorema 1 (*Critério de Euler*)

Para todo $a \in \mathbb{Z}$, seja $p > 2$ um número primo, então:

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$$

Para a prova do Critério de Euler, tanto na versão feita por Laurent Théry (que auxilou na realização deste trabalho) quanto na prova manual apresentada neste trabalho foram necessários os seguintes enunciados:

Lema 2

Seja $p > 2$ um número primo, para todo $a \in \mathbb{Z}$, se $\text{mdc}(a, p) = 1$ *e* $x^2 \equiv a \pmod{p}$ *não tem solução então:*

$$(p-1)! \equiv a^{\frac{p-1}{2}} \pmod{p}$$

Teorema 2 (*Teorema de Wilson*)

Seja número composto um número que pode ser escrito como a multiplicação de dois outros números menores então, dado $n > 1$:

$$(n-1)! \equiv \begin{cases} -1 \pmod{n} \text{ se } n \text{ é primo} \\ 0 \pmod{n} \text{ se } n \text{ é composto e } n \neq 4 \end{cases}$$

em que, o Lema 2 junto a uma versão do Critério de Euler para números naturais foram provados por Laurent Théry durante o período de realização deste trabalho, enquanto o Teorema de Wilson já se encontrava na biblioteca.

Os enunciados dos lemas provados por Laurent Théry em *Coq* são:

- Para o Lema 2:

```
Lemma fact_sqr_exp a p :
    prime p → ~~res_quad p a →
        (p.-1`!) = a ^ p.-1./2 %[mod p].
```

- Para a versão do Critério de Euler com números naturais (e desconsiderando o caso em que $p \mid a$):

```
Lemma euler_criterion a p :
  prime p → ~~(p %| a) →
  a ^ p.-1./2 = (if res_quad p a then 1 else p.-1) %[mod p].
```

Implementações Externas

onde `res_quad` é uma função definida como:

```
Definition res_quad p a :=
    has (fun i => i * i == a %[mod p]) (iota 0 p).
```

A prova do Lema `fact_sqr_exp` é bastante complexa, por isso existem alguns pontos a serem mencionados aqui:

- Na prova é feito o uso extenso de estruturas como anéis e corpos (*fields*), relacionadas pelo seguinte lema:

Lema 3

$\forall\, n \in \mathbb{Z}, \mathbb{Z}/(n)$ *é um corpo se e somente se* n *é primo.*

Quanto a implementação destas estruturas na biblioteca tem-se:

- O tipo `ordinal`:

```
Inductive ordinal n := Ordinal m of m < n.
Notation "'I_' n" := (ordinal n).
Coercion nat_of_ord n (i : 'I_n) :=
    let: @Ordinal _ m _ := i in m.
```

- A função `inZp`:

```
Definition inZp i :=
    @Ordinal p (i %% p) (ltn_pmod i (ltn0Sn p')).
```

- A implementação de anéis (com elementos neutros diferentes para cada operação) e corpos na biblioteca:

```
Definition Zp_trunc p := p.-2.

Notation "''Z_' p" := 'I_(Zp_trunc p).+2
  (at level 8, p at level 2, format "''Z_' p") :
      type_scope.

Notation "''F_' p" := 'Z_(pdiv p)
  (at level 8, p at level 2, format "''F_' p") :
      type_scope.
```

- A prova feita por Laurent, semelhante a prova manual, se baseia na reorganização de um produtório de tamanho arbitrário, logo há o uso extenso do operador `bigop` sobre o qual tem-se:
  - Definição do operador `bigop`:

```
Definition bigop R I idx op r (P : pred I) (F : I → R) :
    R := foldr (fun i x ⇒ if P i then op (F i) x else x)
        idx r.
```

  - Notação para o operador `bigop`:

```
Notation "\big [ op / idx ]_ ( i ← r | P ) F" :=
    (bigop idx op r (fun i ⇒ P%B) (fun i ⇒ F)) :
        big_scope.
```

- Lema `partition_big` que permite "quebrar" uma aplicação de bigop em duas aplicações aninhadas:

```
Lemma partition_big {R : Type} {idx : R}
{op : Monoid.com_law idx} {I : Type} {s : seq I}
{J : finType} {P : pred I} (p : I → J) (Q : pred J)
{F : I → R} :
(∀ i : I, P i → Q (p i)) →
    \big[op/idx]_(i ← s | P i) F i =
        \big[op/idx]_(j | Q j) \big[op/idx]_(i ← s | P i
            && (p i == j)) F i
```

- Lema `eq_bigr` útil para provar a igualdade entre duas aplicações do operador `bigop`:

```
Lemma eq_bigr r (P : pred I) F1 F2 :
    (forall i, P i → F1 i = F2 i) →
        \big[op/idx]_(i ← r | P i) F1 i =
            \big[op/idx]_(i ← r | P i) F2 i.
```

- Lema `big1` útil para provar que um resultado de uma aplicação de `bigop` é igual ao elemento `idx`:

```
Lemma big1 {R : Type} {idx : R} {op : Monoid.law idx} I r
    (P : pred I) F :
    (forall i : I, P i → F i = idx) →
        \big[op/idx]_(i ← r | P i) F i = idx.
```

Obs.: no uso deste lema com `apply` pode-se ao invés de provar que o termo geral é sempre igual a `idx`, provar que nenhum tempo do tipo `I` satisfaz o predicado `P`.

- Existem diversas notações relacionadas ao operador `bigop` para casos específicos da aplicação deste, como para produtórios, e:
  - Para cada caso específico, em geral existe mais de uma notação;
  - Algumas destas se aproveitam da existência de tipos finitos (para os quais há uma lista que contém todos os elementos), como nas seguintes expressões que aparecem durante a prova:

```
\prod_(i in 'F_p | i != 0%R) i
```

```
\prod_(j in 'F_p | j < f j) (j * f j)
```

- Durante a prova aparecem diversos conjuntos que são definidos por meio de predicados sobre tipos finitos como em:

```
set A := [predD1 'F_p & 0%R].
```

```
pose B := [pred i | (i : 'F_p) < f i].
```

assim `A` é contém os elementos de `'F_p` diferentes de `0` e `B` é contém todos os elementos de `'F_p` tal que `i < f i`.

O *símbolo de Legendre* foi implementado por meio da seguinte função:

```
Definition legendre_symb {p : int} (pL2 : (2 < p)%R)
    (pP : primez.primez p) (a : int) :=
        if (p %| a)%Z then 0%Z else if (resz_quad p a)
        then 1%Z else (-1)%Z.
```

onde `resz_quad` tem a seguinte definição (que por sua vez é baseada na definição de `res_quad`):

```
Definition resz_quad p a :=
    has (fun i => ((i * i)%:Z == a %[mod p])%Z) (iota 0 `|p|).
```

Um exemplo de uso da função `legendre_symb` é:

```
Compute (legendre_symb (_ : 2 < 7)%R
    (_ : primez.primez 7) 2).
```

Quanto a prova de corretude, esta foi implementada usando o tipo indutivo `reflect` e foi divida em duas partes por fins práticos (uso em táticas da *ssreflect*):

```
Theorem legendre_symbP {p : int} (pL2 : (2 < p)%R)
    (pP : primez.primez p) (a : int):
    reflect (exists x, x^2 = a %[mod p])
    (if (p %| a)%Z then (((legendre_symb pL2 pP a) == 0))
    else ((legendre_symb pL2 pP a) == 1)).
```

```
Theorem legendre_symbnP {p : int} (pL2 : (2 < p)%R) (pP : primez.
    primez p) (a : int):
    reflect (~ exists x, x^2 = a %[mod p])
    ((legendre_symb pL2 pP a) == -1).
```

Foram provadas todas as propriedades sobre o *símbolo de Legendre* e uma versão do Critério de Euler idêntica apresentadas em (BROCHERO et al., 2013), além de algumas propriedades auxiliares, mas por fim de breviedade serão mostradas aqui apenas os 2 primeiros grupos:

O enunciado do Critério de Euler em *Coq* é dado por:

```
Lemma eulerz_criterion {p : int} (pL2 : (2 < p)%R)
(pP : primez.primez p) (a : int):
    (a ^ ((p - 1) %/ 2)%Z =
        (legendre_symb pL2 pP a) %[mod p])%Z.
```

e considerando um número primo $p > 2$ e $a, b \in \mathbb{Z}$ tem-se as seguintes propriedades:

- $a \equiv b \pmod{p} \rightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$, cuja declaração dada em *Coq* é:

```
Lemma legendre_symbE (p a b : int) (pL2 : (2 < p)%R)
(pP : primez.primez p):
    (a == b %[mod p])%Z →
    ((legendre_symb pL2 pP a) == (legendre_symb pL2 pP b)).
```

Formalização do Símbolo de Legendre

- $p \nmid a \rightarrow \left(\frac{a^2}{p}\right) = 1$, cuja declaração dada em *Coq* é:

```
Lemma legendre_symb_Ndvd (p a b : int) (pL2 : (2 < p)%R)
(pP : primez.primez p):
~~(p %| a)%Z → (legendre_symb pL2 pP (a^2)) == 1.
```

- $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = 1 \leftrightarrow p \equiv 1 \pmod 4$, cuja declaração dada em *Coq* é:

```
Lemma legendre_symb_Neg1 (p : int) (pL2 : (2 < p)%R)
(pP : primez.primez p):
((legendre_symb pL2 pP (-1)) == 1) = (p == 1 %[mod 4])%Z.
```

- $\left(\frac{a \cdot b}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right)$, cuja declaração dada em *Coq* é:

```
Lemma legendre_symb_mul (p a b : int) (pL2 : (2 < p)%R) (pP :
     primez.primez p):
    (legendre_symb pL2 pP (a * b)%R) =
    ((legendre_symb pL2 pP a) *
        (legendre_symb pL2 pP b))%R.
```

- $\left(\frac{a \cdot b}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right)$, cuja declaração dada em *Coq* é:

```
Lemma legendre_symb_mul (p a b : int) (pL2 : (2 < p)%R) (pP :
     primez.primez p):
    (legendre_symb pL2 pP (a * b)%R) =
    ((legendre_symb pL2 pP a) *
        (legendre_symb pL2 pP b))%R.
```

Algumas informações usadas durante as provas destes enunciados são muito úteis para futuros usuários da biblioteca; se destacam os seguintes itens:

- Os números inteiros são definidos da seguinte forma na biblioteca:

```
Variant int : Set := Posz of nat | Negz of nat.
```

  e `Posz` é uma definida como coerção de `nat` para `int`, o que permite utilizar naturais em expressões com números inteiros.

Formalização do Símbolo de Legendre

- A aplicação do construtor `Posz` não é impressa, o que gera ambiguidades como entre as expressões `Posz (a - b)` e `Posz a - Posz b` que são impressas como `a - b`;
  - caso o usuário deseje imprimir as aplicações de `Posz` ele pode usar o comando:

```
Set Printing Coercions.
```

- A aplicação da função `Posz` possui certas propriedades, por exemplo:
  - É sempre o caso de que `Posz m + n = Posz m + Posz n`, assim tal reescrita pode ser feita utilizando o lema:

```
Lemma PoszD : {morph Posz : m n / (m + n)%N ↦ m + n}.
```

e para multiplicação, de forma análoga, pode ser utilizado o lema `PoszM`.

- Quando `n <= n` tem-se `Posz m - n = Posz m - Posz n`, assim tal reescrita pode ser feita utilizando o lema:

```
Lemma subzn (m n : nat) :
   (n <= m)%N → m%:Z - n%:Z = (m - n)%N.
```

- Para operação de resto da divisão tem-se o lema:

```
Lemma modz_nat (m d : nat) : (m %% d)%Z = (m %% d)%N.
```

- Há também lemas para fazer manipulações semelhantes com outras operações (mas as vezes é necessário reescrever o operador em sua versão para aneis):

```
Lemma rmorphXn n : {morph f : x / x ^+ n}.
```

- Para manipulações de expressões com `int` não se utilizam lemas específicos para este tipo (por motivo explicado na Subsessão 2.3.4);
  - para comutatividade da adição, por exemplo, se usa `addrC`, para o qual o comando `Print` traz a mensagem:

```
GRing.addrC =
fun s : nmodType =>
GRing.isNmodule.addrC
    (GRing.Nmodule.GRing_isNmodule_mixin
        (GRing.Nmodule.class s))
            : forall s : nmodType, commutative +%R

Arguments GRing.addrC {s} x y
```

BROCHERO, F. E. et al. *Teoria dos Números: um passeio com primos e outros números familiares pelo mundo inteiro*. 3. ed. Rio de Janeiro : IMPA: IMPA, 2013. (Projeto Euclides). ISBN 978-85-2444-0312-5.

COOK, J. D. *Quadratic reciprocity algorithm*. 2023. Disponível em: <https://www.johndcook.com/blog/2023/01/01/quadratic-reciprocity-algorithm/>. Acesso em: 06 de jun. de 2024.

KUMAR, R. *An algorithm for finding square root modulo p*. 2020. Disponível em: <https://doi.org/10.48550/arXiv.2008.11814>. Acesso em: 15 de maio de 2024.

Referências

LI, Z.; DONG, X.; CAO, Z. Generalized cipolla-lehmer root computation in finite fields. In: ICINS 2014 - 2014 INTERNATIONAL CONFERENCE ON INFORMATION AND NETWORK SECURITY, CP657. *ICINS 2014 - 2014 International Conference on Information and Network Security*. Pequim, China, 2014. p. 163–168.

SARKAR, P. Computing square roots faster than the tonelli-shanks/bernstein algorithm. *Advances in Mathematics of Communications*, v. 18, n. 1, p. 141–162, 2024. Disponível em: <https://www.aimsciences.org/article/id/6212ee892d80b75aa4a24c21>.

WRIGHT, S. Four interesting applications of quadratic reciprocity. In: ______. *Quadratic Residues and Non-Residues: Selected Topics*. Suíça: Springer Cham, 2016. p. 79–118. ISBN 978-3-319-45955-4. Disponível em: <https://doi.org/10.1007/978-3-319-45955-4_4>.