Formal Verification and Code-Generation of Mersenne-Twister Algorithm

1st Takafumi Saikawa
Graduate School of Mathematics
Nagoya University
Nagoya, Japan
tscompor@gmail.com

2nd Kazunari Tanaka
Graduate School of Mathematics
Nagoya University
Nagoya, Japan
tzskp1@gmail.com

3rd Kensaku Tanaka
IT Department
RAKUDO Inc.
Nagoya, Japan
kensaku_tanaka@rakudo.io

***Abstract*—We formalize the pseudocode and linear-algebraic presentations of Mersenne-Twister, and formally establish their equivalence. Based on this formalization, we investigate the long-period property of Mersenne-Twister, formally proving that the property is reduced to the primitivity of the characteristic polynomial of the matrix representation.**

The formalization is done in Coq proof assistant. This enables us to generate a C program code from the verified pseudocode written in Coq.

I. Introduction

Pseudorandom number generators (PRNGs) are present everywhere in modern computing systems. Among many algorithms for PRNGs, *Mersenne-Twister* by Matsumoto and Nishimura [1] features a very long-period ($2^{19937} - 1$) of its internal states and stochastic properties such as the equidistribution of generated points in a fairly high-dimensional vector space. These properties make Mersenne-Twister a suitable choice for various applications including simulation uses such as Monte Carlo methods.

Mersenne-Twister is presented in two ways in the original paper [1]: one as a piece of pseudocode based on binary arithmetic (i.e., operations on bit sequences), and another as a series of linear transformations represented as matrix multiplications. The pseudocode can be easily translated to practical programming languages and used in applications. On the other hand, the linear-algebraic presentation provides a foundation for proving the properties of Mersenne-Twister.

These two presentations are implicitly identified in the literature and the results proved for the linear-algebraic version are transported to the pseudocode version through this identification. Since the two presentations are apparently quite different, we want to be sure of the correctness of the identification through a careful formalization, which is however not yet done to the best of our knowledge.

In this paper, we fill this gap by formally proving the equivalence of the two presentations, using the proof-assistant Coq and the library MathComp. Both presentations are realized as Coq functions, each based on a different data structure. We state and prove the equivalence through a pair of embedding and projection functions between these data structures.

Thanks to this equivalence, we can attempt to formally verify the properties of Mersenne-Twister by working on the linear-algebraic presentation. As a work-in-progress result, we show that the long-period property is reduced to the primitivity of the characteristic polynomial of a matrix. We also formalize an algorithm called *inversive-decimation* that is used for proving the primitivity in the original paper [1].

Our formalization can also be combined with a code generation plugin [2] to translate the Coq functions for algorithms (Mersenne-Twister and inversive-decimation) into C program code. We can use it to reduce the possibility of bugs to be introduced at a translation (e.g. a case in the standard library for PHP [3]) by confining the risk into a dictionary for translating the primitive operations and the plugin itself.

All of our Coq code is stored in a `github` repository [4]. In order to reproduce our environment and trace the proofs, please follow the instructions provided in `README.md` and `Dockerfile` in the repository.

II. Formalization of Mersenne-Twister algorithm

In this section, we formalize the two presentations of Mersenne-Twister algorithm. The first one is based on binary arithmetic operations on an array of integers and is close to implementations in practical programming languages such as C. The second one is based on linear algebra and follows the mathematical presentation in the original paper [1] After defining these presentations, we conclude this section by explaining the formal proof of the equivalence between them.

A. Algotirhm based on binary arithmetic

The first presentation is essentially the same as the original pseudocode [1, Section 2.1]. The core of the algorithm is a function `next_random_state` that computes the next state from a given previous state. The state is represented by a record `random_state`, which is just a list of natural numbers with an index. Binary arithmetic operations that appear in the pseudocode are realized as following Coq functions that come from `BinNat` module in Coq's standard library: `N.lor`, `N.land`, `N.lxor`, `N.shiftr`, `N.shiftl`, and `N.testbit`.

Definition 1 (`next_random_state` in [4, `mt.v`])**.**

This is a preprint of a paper to be presented at ISITA 2020 http://www.isita.ieice.org/2020/home.html

```
Variables len m r a w : N.
Variables u s t l b c : N.
Hypothesis rw : (r <= w)%nat.

Definition upper_mask :=
(N_of_word (make_upper_mask rw)).
Definition lower_mask :=
(N_of_word (make_lower_mask rw)).
Record random_state :=
{index : N; state_vector : seq N}.

Definition next_random_state
(rand : random_state) : (N * random_state) :=
let state_vec := state_vector rand in
let ind := index rand in
let current := nth 0 state_vec ind in
let next_ind := N.modulo (N.succ ind) len in
let next := nth 0 state_vec next_ind in
let far_ind := N.modulo (ind + m) len in
let far := nth 0 state_vec far_ind in
let z := N.lor (N.land current upper_mask)
               (N.land next lower_mask) in
let xi := N.lxor (N.lxor far (N.shiftr z 1))
          (if N.testbit z 0 then a else 0) in
let next_rand := {|
  index := next_ind;
  state_vector := set_nth 0 state_vec ind xi;
|} in
(xi, next_rand).

Definition tempering xi :=
let y1 := N.lxor xi (N.shiftr xi u) in
let y2 := N.lxor y1
 (N.land (N.shiftl y1 s) b) in
let y3 := N.lxor y2
 (N.land (N.shiftl y2 t) c) in
let y4 := N.lxor y3 (N.shiftr y3 l) in
y4.
```

We explain some of the lines above by their correspondence to the pseudocode. In the following, the references to equations and pages point to those in the original description [1, Section 2.1]. The values `far` and `z` correspond to $\mathbf{x}_{k+m}$ and $(\mathbf{x}_k^u | \mathbf{x}_{k+1}^l)$ respectively, both apeearing in Equation (2.1). The value `xi` is the composition of Equation (2.1) and the equation for $\mathbf{x}A$ appearing at the end of p.8. The return value `(xi, next_rand)` is a pair of the tempered random value and the next random state. The auxiliary computation of values `y1`, `y2`, `y3`, and `y4` are called *tempering*, and correspond to Equations (2.2), (2.3), (2.4), and (2.5), respectively.

B. Algorithm based on linear algebra

The second presentation is based on linear algebra. The states are represented by elements in $\mathbb{F}_2^{19937}$ that is seen as a vector space over $\mathbb{F}_2$, and the next random state is computed from a given state by multiplying a matrix.

In the following, we abstract the magic number 19937 to generalize the construction of the matrix.

Definition 2 (`A`, `S`, and `B` in [4, `cycle.v`]). *Let n, w, r be integers such that $2^{nw-r} - 1$ is a prime number, and $\{a_i\}_{i=0\ldots w-1}$ a sequence in $\mathbb{F}_2$. We define as follows the $(nw-r)$-dimensional square matrix B which embodies the linear-algebraic representation of Mersenne-Twister:*

$$A = \left(\begin{array}{c|cccc} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ \hline a_0 & a_1 & a_2 & \cdots & a_{w-1} \end{array}\right)$$

$$S = \left(\begin{array}{c|c} & 1_r \\ \hline 1_{w-r} & \end{array}\right) A$$

$$B = \left(\begin{array}{c|cccc} & 1_w & & & \\ & & 1_w & & \\ & & & \ddots & \\ 1_w & & & & 1_w \\ & & & & & \ddots \\ & & & & & & 1_{w-r} \\ \hline S & & & & & & \end{array}\right)$$

C. Equivalence

We can now state the equivalence of these two algorithms, namely, the function `next_random_state` (Definition 1) and the linear transformation defined by multiplying the matrix B (Definition 2). Here we just show the statements of a central lemma and the equivalence theorem, and provide an explanation about the difficulty in the formalized proof

The first step is to rewrite the multiplication by B into a computation similar to Equation (2.1) in [1, Section 2.1].

Lemma 1 (`computeB` and `mulBE` in [4, `cycle.v`]). *For any bit sequence $(\mathbf{x}_{n-1}, \ldots, \mathbf{x}_1, \mathbf{x}_0\restriction_{w-r}) \in \mathbb{F}_2^{nw-r}$,*

$$(\mathbf{x}_{n-1}, \ldots, \mathbf{x}_1, \mathbf{x}_0\restriction_{w-r})B = (\mathbf{x}_n, \ldots, \mathbf{x}_2, \mathbf{x}_1\restriction_{w-r})$$

where

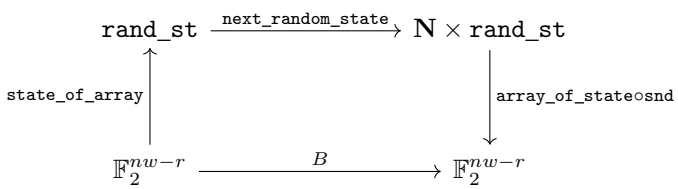
$$\mathbf{x}_n = \mathbf{x}_m + \left(\begin{array}{c|c} & \\ \hline & 1_r \end{array}\right) \mathbf{x}_1 A + \left(\begin{array}{c|c} 1_{w-r} & \\ \hline & \end{array}\right) \mathbf{x}_0 A$$

and $\mathbf{x}_k\restriction_{w-r}$ is the first $w-r$ bits of $\mathbf{x}_k$.

By this lemma, the multiplication by B is decomposed into several primitive operations such as rotation, masking, and addition of bits, still expressed in the language of linear algebra.

In order to go over to the language of binary arithmetic, we need to translate the data structures used for internal states. This translation is given by an injective function `state_of_array` that maps a vector to a list of integers with index 0, and a surjective function `array_of_state` that rotates the list of integers to normalize the index to 0 and extracts the bits as a vector. With these functions, the equivalence theorem between the two presentations is stated as follows:

Theorem 1 (`next_random_stateE` in [4, `cycle.v`])**.** *The following diagram*



commutes.

III. Long-period property of Mersenne-Twister

We can now investigate algebraic properties of Mersenne-Twister using the algebraic definition presented in the previous section. Here we describe the most peculiar property: the period of cycle in Mersenne-Twister is very long. The matrix representation of the algorithm plays a crucial role.

We consider the extension field $\mathbb{F}_{2^{nw-r}}$ over the binary field $\mathbb{F}_2$ such that the degree $[\mathbb{F}_{2^{nw-r}} : \mathbb{F}_2]$ is $nw - r$. Let $\varphi(X)$ be the characteristic polynominal $\varphi(X)$ of B.

The Galois group $\mathrm{Gal}(\mathbb{F}_{2^{nw-r}}/\mathbb{F}_2)$ is generated by the frobenius map $\sigma(= x \mapsto x^2) \in \mathrm{Aut}(\mathbb{F}_{2^{nw-r}})$. Because the extension field $\mathbb{F}_{2^{nw-r}}$ is a group ring $\mathbb{F}_2[\mathrm{Gal}(\mathbb{F}_{2^{nw-r}}/\mathbb{F}_2)]$, we observe $\sigma^{nw-r} = 1$ which is equivalent to $X^2 \not\equiv_{\varphi(X)} X$ and $X^{2^{nw-r}} \equiv_{\varphi(X)} X$. In addition, this condition will be equivalent to primitivity of the characteristic polynominal $\varphi(X)$.

Then according to Cayley-Hamilton theorem, we can conclude that a period of the matrix B is $2^{nw-r} - 1$.

We wish to formalize above situations in Coq. For a better readability of the discussion, we will explain the theorems and proofs in natural language rather than Coq expressions.

In the following lemma, note that the primitivity and irreducibility of a polynomial are equivalent notions.

Lemma 2. *[`irreducibleP` (1 and 4), `irreducibleP2` (1 and 3), `expandF` (2 and 4) in [4, `irreducible.v`]] Let $x \in \mathbb{F}_2[X]/\varphi(X)$. If we assume $x^2 \neq x$, the following are equivalent.*

1) *$\varphi(X)$ is irreducible.*
2) *$\sigma^{nw-r} = 1$.*
3) *$\sigma^{nw-r} x = x$.*
4) *$X^2 \not\equiv_{\varphi(X)} X$ and $X^{2^{nw-r}} \equiv_{\varphi(X)} X$.*

Remark 1. *Because we think that the frobenius map σ is just a map, these statement make sense.*

Proof. We will show that (3) implies (1). Assume that there exist $q_1, q_2 \in \mathbb{F}_2[X]$ such that $\varphi(X) = q_1 q_2$ and $\deg q_2 > 1$. Then we wish to show $q_2 = \varphi(X)$ i.e. $q_1 = 1$. So we assume $q_1 \neq 1$. Since if $q_1 = 0$ then it is just a trivial case, we can assume $\deg q_1 > 1$.

According to (3), a set $\mathbb{F}_2[X]/\varphi(X) \setminus \{0\}$ is of the form $\{x^{i+1} \mid i = 1, \ldots, 2^{nw-r} - 1\}$. We will excuse an above statement. Because $|\mathbb{F}_2[X]/\varphi(X) \setminus \{0\}| \leq 2^{nw-r} - 1$, it is suffice to show $|\{x^{i+1} \mid i = 1, \ldots, 2^{nw-r} - 1\}| = 2^{nw-r} - 1$ i.e. for l which satisfies $0 < l < 2^{nw-r} - 1$, $x^{l+1} \neq x$. So we assume that there is l such that $0 < l < 2^{nw-r} - 1$ and $x^{l+1} = x$. Because there is the minimum element m of a set $\{l \mid 0 < l \text{ and } x^{l+1} = x\}$, l and $2^{nw-r} - 1$ are divided by m. An assumption $x^2 \neq x$ means $m \neq 1$. So we can conclude $m = 2^{nw-r} - 1$. But it contradicts to $l < 2^{nw-r} - 1$.

Let $\pi \colon \mathbb{F}_2[X] \to \mathbb{F}_2[X]/\varphi(X)$ be a canonical surjection. We have $0 = \pi\varphi(X) = \pi q_1 \pi q_2$, $\pi q_1 \neq 0$ and $\pi q_2 \neq 0$. So there is an integer i such that $0 = x^{i+1} \in \mathbb{F}_2[X]/\varphi(X) \setminus \{0\}$. It is obviously a contradiction.

Implication of (4) to (1) is a special case of implication of (3) to (1).

We will show that (1) implies (4). Since it trivially holds $X^2 \not\equiv_{\varphi(X)} X$, we will focus on $X^{2^{nw-r}} \equiv_{\varphi(X)} X$ i.e. an order o of πX is $2^{nw-r} - 1$. Note that $|(\mathbb{F}_2[X]/\varphi(X))^\times| = 2^{nw-r} - 1$ is divided by o (by Lagrange's theorem) and that $2^{nw-r} - 1$ is prime, we can conclude $o = 2^{nw-r} - 1$.

We will show that (4) implies (2). Note that $\mathbb{F}_2[X]/\varphi(X)$ is a linear space which has a specific basis $\{\pi X^i \mid i = 0, \ldots, nw - r\}$, it is suffice to show that $(\pi X^i)^{2^{nw-r}} = \pi X^i$. But it trivially holds. □

In terms of the multiplication of B, the long-period property is stated as follows:

Lemma 3 (`cycleB_dvdP` in [4, `cycle.v`])**.** *Assume that the characteristic polynomial $\varphi(X)$ of B is irreducible. Then for any $q \in \mathbb{N}_{>0}$, the following are equivalent.*

1) *$B^q = B$.*
2) *$q - 1$ is divided by $2^{nw-r} - 1$.*

Proof. We first show that (1) implies (2). We begin by noting that B^q and B are the evalutations by B of polynomials X^q and X respectively:

$$X^q[X := B] = X[X := B].$$

We can calculate the inverses of these evaluations (`mx_inv_horner` in [5, `mxpoly.v`]), which are polynomials modulo $\varphi(X)$.

$$X^q[X := B][X := B]^{-1} \equiv_{\varphi(X)} X[X := B][X := B]^{-1}.$$

The evaluations and their inverses cancel out, and we obtain that

$$X^q \equiv_{\varphi(X)} X.$$

By Lemma 2, we also have $X^{2^{nw-r}} \equiv_{\varphi(X)} X$. Since $2^{nw-r} - 1$ is prime, we can conclude by Lagrange's theorem that $q - 1$ is divided by $2^{nw-r} - 1$.

We next show that (2) implies (1). By Euclidean algorithm, there are $p(X), r(X) \in \mathbb{F}_2[X]$ such that $X^q = p(X)\varphi(X) + r(X)$ and $\deg r(X) < \deg \varphi(X)$. Since Cayley-Hamilton theorem tells that $\varphi(B) = 0$, $B^q = r(B)$. For the desired equality, it suffices to show that $r(X) = X$. This is obtained again by Lemma 2 with noticing that $X^q = X^{q-1+1} = X^{(2^{nw-r}-1)i+1} \equiv_{\varphi(X)} X$ for the quotient $i = (q-1)/(2^{nw-r} - 1)$.

□

IV. Algorithm of the inversive-decimation method

In the previous section, we proved that the long-period property of Mersenne-Twister is reduced to the irreducibility of the characteristic polynomial of B. The remaining part, the irreducibility is rather a difficult task. In the original paper [1], its proof is achieved by the reflection on the inversive-decimation algorithm. We attempted to formalize this strategy, and currently formalized the algorithm itself as a Coq function:

```
Definition generate a (state : state_vector) :=
  let rand := make_mtRand state in
  let start_word_seq :=
   word_seq_of_state_vector state in
  generate_aux a start_word_seq rand p2n.

Fixpoint decimate_aux
(words : word_seq) (acc : word_seq) times :=
  match times with
  | 0%nat => acc
  | S times' =>
    let j := plus (minus p times) 1%nat in
    let k := minus (2%nat * j) 1%nat in
    decimate_aux words
    (set_nth_word_seq acc j
     (nth_word_seq words k)) times'
  end.

Definition decimate words :=
  decimate_aux words words p.

Fixpoint process_aux a words times :=
  match times with
  | 0%nat => words
  | S times' =>
    let k := plus times (minus n 1%nat) in
    let xk := nth_word_seq words k in
    let kn := minus k n in
    let xkn := nth_word_seq words kn in
    let knm := plus kn m in
    let xknm := nth_word_seq words knm in
    let kn1 := plus kn 1%nat in
    let xkn1 := nth_word_seq words kn1 in
    let y := N.lxor (N.lxor xk xknm)
      (if N.eqb (N.land xkn1 1) 0
      then 0 else a) in
    let y1 := N.shiftl y 1 in
    let y2 :=
        if N.eqb (N.land xkn1 1) 0
        then N.land y1 bottom_zero_mask
        else N.lor y1 bottom_one_mask in
    let newxkn1 :=
        N.lor (N.land upper_mask xkn1)
              (N.land lower_mask y2) in
    let newxkn :=
        N.lor (N.land upper_mask y2)
              (N.land lower_mask xkn) in
    let words1 :=
        set_nth_word_seq words kn1 newxkn1 in
    let words2 :=
        set_nth_word_seq words1 kn newxkn in
    process_aux a words2 times'
  end.

Definition process a (state : state_vector) :=
  let expandedWords := generate a state in
  let decimatedWords :=
      decimate expandedWords in
  let pn1 := minus p (minus n 1%nat) in
  state_vector_of_word_seq
   (process_aux a decimatedWords pn1).
```

We have so far succeeded in defining the above algorithm and type-check it.

V. Code generation

We can utilize the code generation plugin for Coq [2] to generate a working C program code for the algorithms formalized in this paper, namely Mersenne-Twister and inversive-decimation.

As an example, we show the piece of generated C code for `next_random_state`:

```
static prodNrnd
next_random_state(rand_state v1_rand)
{
  list_N v2_state_vec, v35_l;
  N v3_ind, v4_n, v6_current, v7_n;
  N v8_n, v9_next_ind, v10_n;
  N v12_next, v13_n, v14_n, v15_n;
  N v16_far_ind, v17_n;
  N v19_far, v20_n, v21_n, v22_n;
  N v23_n, v24_z;
  N v26_n, v27_n, v28_n, v29_n;
  N v31_n, v32_xi, v33_n;
  nat v5_n, v11_n, v18_n, v34_n;
  positive v25_p;
  bool v30_b;
  rand_state v36_next_rand;
  v2_state_vec = STATE_VECTOR(v1_rand);
  v3_ind = INDEX(v1_rand);
  v4_n = N0();
  v5_n = nat_of_bin(v3_ind);
  v6_current = nth(v4_n, v2_state_vec, v5_n);
  v7_n = succ(v3_ind);
  v8_n = len();
  v9_next_ind = modulo(v7_n, v8_n);
  v10_n = N0();
  v11_n = nat_of_bin(v9_next_ind);
  v12_next = nth(v10_n, v2_state_vec, v11_n);
  v13_n = m();
  v14_n = add(v3_ind, v13_n);
  v15_n = len();
  v16_far_ind = modulo(v14_n, v15_n);
  v17_n = N0();
  v18_n = nat_of_bin(v16_far_ind);
  v19_far = nth(v17_n, v2_state_vec, v18_n);
  v20_n = upper_mask();
  v21_n = land(v6_current, v20_n);
  v22_n = lower_mask();
  v23_n = land(v12_next, v22_n);
  v24_z = lor(v21_n, v23_n);
  v25_p = xH();
  v26_n = Npos(v25_p);
  v27_n = shiftr(v24_z, v26_n);
  v28_n = lxor(v19_far, v27_n);
  v29_n = N0();
  v30_b = testbit(v24_z, v29_n);
  switch (v30_b)
  {
    default:
      v31_n = a();
      break;
```

```
    case 0:
      v31_n = N0();
      break;
  }
  v32_xi = lxor(v28_n, v31_n);
  v33_n = N0();
  v34_n = nat_of_bin(v3_ind);
  v35_l =
   set_nth(v33_n, v2_state_vec, v34_n, v32_xi);
  v36_next_rand =
   Build_random_state(v9_next_ind, v35_l);
  return make_prodNrnd(v32_xi, v36_next_rand);
}
```

VI. Conclusion and Future Work

We have formalized two presentations of Mersenne-Twister. One is the pseudocode, and the other is linear-algebraic. We have formally proved the equivalence of these two presentations. Based on these formalization, we formally investigated the long-period property of Mersenne-Twister, proving that the property is reduced to the irreducibility of the characteristic polynomial of a matrix. We started to prove the primitivity by formalizing the inversive-decimation algorithm. We have shown that the Coq definitions of algorithms using binary arithmetic can be fed to the code generation method to achieve a runnable C code.

In the future work, we plan to complete the proof of the primitivity using the inversive-decimation algorithm. We also plan to formalize other properties of Mersenne-Twister such as the distribution of generated points in a high-dimension space. Apart from Mersenne-Twister, we have as a long-term plan to generalize this work to other pseudorandom number generators, providing a framework to formalize broader examples.

References

[1] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998. [Online]. Available: http://doi.acm.org/10.1145/272991.272995

[2] A. Tanaka, R. Affeldt, and J. Garrigue, "Safe low-level code generation in Coq using monomorphization and monadification," *Journal of Information Processing*, vol. 26, pp. 54–72, 2018.

[3] "Fix #71152: mt_rand() returns the different values from original mt19937ar.c #1681," https://github.com/php/php-src/pull/1681/files, 2016.

[4] K. Tanaka, K. Tanaka, and T. Saikawa, "Formalization of mersenne-twister," https://github.com/tzskp1/codegen-examples/releases/tag/ISITA2020_paper, 2020.

[5] Mathematical Components Team, "Mathematical Components library," https://github.com/math-comp/math-comp, 2007, development version. Last stable version 1.10 (2019) available on the same website.