WEB3CLUBS FOUNDATION LIMITED

Course Instructor: DR. Cyprian Omukhwaya Sakwa
PHONE: +254723584205   Email: cypriansakwa@gmail.com

Foundational Mathematics for Web3 Builders

Implemented in RUST

Lecture 43

August 15, 2024

$(G, +)$

The direct product groups

Definition 16

Given a pair of abelian groups G and H, we can construct a new group from these two by getting their direct product. The direct product $G \times H$ is the set of ordered pairs (g, h) with $g \in G$ and $h \in H$ under multiplication

$$(g_1, h_1)(g_2, h_2) = (g_1 g_2, h_1 h_2).$$

This implies that group operation in a direct product group is performed componentwise. If $(x, y), (a, b) \in G \times H$ then $(x, y) \cdot (a, b) = (x \cdot a, y \cdot b)$.

The identity element in the group $G \times H$ is (e_1, e_2) where e_1 is the identity in G and e_2 is the identity in H.

Direct product group of more than two groups can also be obtained. For instance, $G_1 \times G_2 \times G_3 = \{(x, y, z) : x \in G_1,\ y \in G_2,\ z \in G_3\}$. The identity element here is (e_1, e_2, e_3) where $e_1,\ e_2$ and e_3 are the identities in $G_1,\ G_2$ and G_3 respectively.

Direct product groups are an effective technique in cryptography, laying the groundwork for the development of secure, efficient, and versatile cryptographic protocols and systems. Their ability to incorporate numerous group structures into a single framework enables the creation of cryptographic systems with more security and usefulness.

1
2
3
~~4~~
5

Example 79

Let $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$ and $\mathbb{Z}_8^* = \{1, 3, 5, 7\}$. Find:

a) $\mathbb{Z}_5^* \times \mathbb{Z}_8^*$ b) $(1, 5)(1, 7)$ c) $(3, 3)(3, 7)$

d) Inverse of $(2, 7)$ e) Inverse of $(3, 5)$

Solution

a) $\mathbb{Z}_5^* \times \mathbb{Z}_8^* = \{(1,1),\ (1,3),\ (1,5),\ (1,7),\ (2,1),\ (2,3),\ (2,5),\ (2,7),$

$(3,1),\ (3,3),\ (3,5),\ (3,7),\ (4,1),\ (4,3),\ (4,5),\ (4,7)\}$

b) $(1,5)(1,7) = (1 \bmod 5, 35 \bmod 8) = (1,3)$.

c) $(3,3)(3,7) = (9 \bmod 5, 21 \bmod 8) = (4,5)$.

d) Since $\mathbb{Z}_5^*$ and $\mathbb{Z}_8^*$ were both multiplicative groups, the identity element of $\mathbb{Z}_5^* \times \mathbb{Z}_8^*$ is $(1,1)$. To find inverse of $(2,7)$ we determine an element $(x,y) \in \mathbb{Z}_5^* \times \mathbb{Z}_8^*$ such that $(2,7)(x,y) = (1,1)$. This implies that
$2x = 1 \bmod 5$ and $7y = 1 \bmod 8$. This gives inverse as $(3,7)$.

e) $(3,5)(x,y) = (1,1)$ implies $3x = 1 \bmod 5$ and $5y = 1 \bmod 8$. Solving gives inverse as $(2,5)$.

Example 80

Find $\mathbb{Z}_4 \times \mathbb{Z}_{12}^*$ then compute;

a) $(2,7)(3,11)$

b) Inverse of $(3,7)$

c) Identity of $\mathbb{Z}_4 \times \mathbb{Z}_{12}^*$.

Solution

$$\begin{aligned}\mathbb{Z}_4 \times \mathbb{Z}_{12}^* = & \{(0,1),(0,5),(0,7),(0,11),(1,1),(1,5),(1,7),(1,11),\\ & (2,1),(2,5),(2,7),(2,11),(3,1),(3,5),(3,7),(3,11)\}\end{aligned}$$

a) $(2,7)(3,11) = (1,5)$

b) Inverse of $(3,7)$ is $(1,7)$

c) Identity of $\mathbb{Z}_4 \times \mathbb{Z}_{12}^*$ is $(0,1)$.

Example 81

Let $G = \mathbb{Z}$ and $H = \{i, -i, 1, -1\}$. Find:

a) $(G, +) \times (H, \cdot)$

b) $(8, -1)(-2, i)$

c) $(3, -i)(-7, -1)$

d) $(5, -i)(7, -i)$

e) Identity in $(G, +) \times (H, \cdot)$

f) Inverse of $(2, -i)$

g) Inverse of $(3, i)$

Solution

a) $(G, +) \times (H, \cdot) = \big\{(g, h) : g \in \mathbb{Z},\ h = \pm 1 \text{ or } h = \pm i\big\}$

b) $(8, -1)(-2, i) = (8 - 2, -1 \cdot i) = (6, -i)$

c) $(3, -i)(-7, -1) = (3 - 7, -i \cdot -1) = (-4, i)$

d) $(5, -i)(7, -i) = (5 + 7, -i \cdot -i)$

$$= (12, -1)$$

Solution (conti...)

e) Identity element is $(0, 1)$ since the first group is additive and the second multiplicative.

f) Let inverse be (x, y). Then, $(2, -i)(x, y) = (0, 1)$

$(2 + x, -iy) = (0, 1)$

$\Rightarrow \; x = -2 \; y = i$ inverse be (x, y) is$(-2, i)$.

a) Similarly, inverse of $(3, i)$ is $(-3, -i)$.

The following Rust program shows modular arithmetic operations on the direct product of two cyclic groups $\mathbb{Z}_n \times \mathbb{Z}_m$. It contains functions to generate elements of the cyclic groups $\mathbb{Z}_n$ and $\mathbb{Z}_m$, compute their direct product, perform modular addition, and compute modular inverses.

1. Generating Elements of $\mathbb{Z}_n$

```
// Function to find elements of Z_n
fn zn(n: u32) -> Vec<u32> {
        (0..n).collect()
}
```

$\mathbb{Z}_8 = \{0, 1, 2, 3, \dots, 7\}$

0 . n

```
// Function to compute the direct product of Z_n and Z_m
fn direct_product(n: u32, m: u32) -> Vec<(u32, u32)> {
        let zn_elements = zn(n);
        let zm_elements = zn(m);

        let mut product = Vec::new();
        for &a in &zn_elements {
                for &b in &zm_elements {
                        product.push((a, b));
                }
        }
        product
}

// Function to compute the sum (a, b) + (e, f) mod (n, m)
fn sum_mod((a, b): (u32, u32), (e, f): (u32, u32), n: u32, m: u32) -> (u32, u32) {
        ((a + e) % n, (b + f) % m)
}

// Function to compute the inverse of (t, s) in Z_n x Z_m
fn inverse_mod((t, s): (u32, u32), n: u32, m: u32) -> (u32, u32) {
        ((n - t) % n, (m - s) % m)
}
```

```
fn main() {
        let n = 10;
        let m = 8;

        let elements = direct_product(n, m);

        println!("Direct␣product␣of␣Z_{}␣and␣Z_{}:", n, m);
        for (a, b) in &elements {
                println!("({},␣{})", a, b);
        }

        // Example of computing (a, b) + (e, f)
        let (a, b) = elements[9]; // Example element from Z_n x Z_m
        let (e, f) = elements[11]; // Another example element from Z_n x Z_m

        let sum = sum_mod((a, b), (e, f), n, m);

println!
("\nSum␣of␣({},␣{})␣and␣({},␣{})␣mod␣({},␣{}):␣({},␣{})", a, b, e, f, n, m, sum.0, sum.1

        // Example of computing inverse of (t, s)
        let (t, s) = elements[13]; // Example element from Z_n x Z_m
        let inverse = inverse_mod((t, s), n, m);

println!
("\nInverse␣of␣({},␣{})␣in␣Z_{}␣x␣Z_{}␣is:␣({},␣{})", t, s, n, m, inverse.0, inverse.1);
}
```

Understanding the Rust code

1. Generating Elements of $\mathbb{Z}_n$

```
fn zn(n: u32) -> Vec<u32> {
        (0..n).collect()
}
```

- This function generates all the elements of the cyclic group $\mathbb{Z}_n$.
- The function returns a vector containing the integers from 0 to $n-1$.

(2) Computing the Direct Product $\mathbb{Z}_n \times \mathbb{Z}_m$

```
fn direct_product(n: u32, m: u32) -> Vec<(u32, u32)> {
        let zn_elements = zn(n);
        let zm_elements = zn(m);

        let mut product = Vec::new();
        for &a in &zn_elements {
                for &b in &zm_elements {
                        product.push((a, b));
                }
        }
        product
}
```

Understanding the Rust code (conti...)

- This function computes the direct product of the groups $\mathbb{Z}_n \times \mathbb{Z}_m$.
- It first generates the elements of $\mathbb{Z}_n$ and $\mathbb{Z}_m$ using the zn function.
- It then creates a vector of tuples representing all possible pairs (a, b) where $a \in \mathbb{Z}_n$ and $b \in \mathbb{Z}_m$.

(3) Addition in $\mathbb{Z}_n \times \mathbb{Z}_m$ Modulo n and m

```
fn sum_mod((a, b): (u32, u32), (e, f): (u32, u32), n: u32, m: u32) -> (u32, u32) {
        ((a + e) % n, (b + f) % m)
}
```

- This function computes the sum of two elements (a, b) and (e, f) in the direct product group $\mathbb{Z}_n \times \mathbb{Z}_m$, with the results taken Modulo n and m.
- The sum of the pairs (a, b) and (e, f) is calculated component-wise

Understanding the Rust code (conti...)

✓ The first component is $(a+e) \mod n$.

✓ The second component is $(b+f) \mod m$.

- The result is a tuple (x, y) representing the sum in the direct product group.

(4) Computing the Inverse in $\mathbb{Z}_n \times \mathbb{Z}_m$

```
fn inverse_mod((t, s): (u32, u32), n: u32, m: u32) -> (u32, u32) {
        ((n - t) % n, (m - s) % m)
}
```

- This function computes the inverse of an element (t, s) in the group $\mathbb{Z}_n \times \mathbb{Z}_m$.
- The inverse of t modulo n is $(n-t) \mod n$, and the inverse of s modulo m is $(m-s) \mod m$.
- The result is a tuple representing the inverse element $(-t, -s) \in \mathbb{Z}_n \times \mathbb{Z}_m$

Understanding the Rust code (conti...)

(5) Main Function: Demonstrating the Operations

```
fn main() {
        let n = 10;
        let m = 8;

        let elements = direct_product(n, m);

        println!("Direct␣product␣of␣Z_{}␣and␣Z_{}:", n, m);
        for (a, b) in &elements {
                println!("({},␣{})", a, b);
        }

        // Example of computing (a, b) + (e, f)
        let (a, b) = elements[9]; // Example element from Z_n x Z_m
        let (e, f) = elements[11]; // Another example element from Z_n x Z_m

        let sum = sum_mod((a, b), (e, f), n, m);

println!
("\nSum␣of␣({},␣{})␣and␣({},␣{})␣mod␣({},␣{}):␣({},␣{})", a, b, e, f, n, m, sum.0, sum.1

        // Example of computing inverse of (t, s)
        let (t, s) = elements[13]; // Example element from Z_n x Z_m
        let inverse = inverse_mod((t, s), n, m);

println!
("\nInverse␣of␣({},␣{})␣in␣Z_{}␣x␣Z_{}␣is:␣({},␣{})", t, s, n, m, inverse.0, inverse.1);
}
```

Understanding the Rust code (conti...)

- The main function defines specific values for n and m.
- It computes the direct product $\mathbb{Z}_n \times \mathbb{Z}_m$ nd prints the elements.
- It demonstrates how to compute the sum of two elements in the direct product group using sum_mod.
- It also shows how to compute the inverse of an element in the direct product group using inverse_mod

The following Rust code shows operations on the direct product of the cyclic group $\mathbb{Z}_n$ and the group of units $\mathbb{Z}_m^*$. It offers capability for generating elements of these groups, performing modular arithmetic on their direct product, calculating element products, and determining an element's inverse.

```
use num::Integer; // Import traits needed for modular arithmetic

fn main() {
        let n = 5;
        let m = 7;

        // Generate elements of Z_n x Z_m*
        let units_mod_m: Vec<i32> = (1..m).filter(|x| x.gcd(&m) == 1).collect();
        let elements: Vec<(i32, i32)> = (0..n)
        .flat_map(|a| units_mod_m.iter().map(move |&b| (a, b)))
        .collect();

        println!("Direct␣product␣of␣Z_{}␣and␣Z_{}^*:", n, m);
        for elem in &elements {
                println!("{:?}", elem);
        }

        // Compute product of two elements
        let (a, b) = elements[0];
        let (e, f) = elements[1];
        let product = ((a + e) % n, (b * f) % m);
println!
("Product␣of␣({},␣{})␣and␣({},␣{})␣mod␣({},␣{}):␣{:?}", a, b, e, f, n, m, product);

        // Find inverse of an element
        let (a, b) = elements[1];
        let inverse_b = units_mod_m.iter().find(|&&x| (b * x) % m == 1).unwrap();
        let inverse = ((-a).rem_euclid(n), *inverse_b);
        println!("Inverse␣of␣({},␣{})␣in␣Z_{}␣x␣Z_{}^*␣is␣{:?}", a, b, n, m, inverse);
}
```

Understanding the Rust code

1. Using num::Integer for Modular Arithmetic. The code imports the Integer trait from the num crate, which contains methods for performing operations such as greatest common divisor $\gcd$ and modular arithmetic.

(2) The Main Function defines two numbers, n and m, representing the orders of the cyclic group $\mathbb{Z}_n$ and the group of units $\mathbb{Z}_m^*$, respectively.

(3) Generating Elements of $\mathbb{Z}_m^*$.

```rust
let units_mod_m: Vec<i32> = (1..m).filter(|x| x.gcd(&m) == 1).collect();
```

- The units_mod_m vector includes all elements of $\mathbb{Z}_m^*$, which are integers from 1 to $m-1$ that are coprime with m. This is accomplished by filtering the range $1 \cdots m$ to only include numbers with a $\gcd$ of 1.

Understanding the Rust code (conti...)

(4) Generating Elements of $\mathbb{Z}_n \times \mathbb{Z}_m^*$.

```
let elements: Vec<(i32, i32)> = (0..n)
.flat_map(|a| units_mod_m.iter().map(move |&b| (a, b)))
.collect();
```

- The elements vector contains all the elements of the direct product $\mathbb{Z}_n \times \mathbb{Z}_m^*$.
- It is generated by taking each element $a \in \mathbb{Z}_n$ and pairing it with each element $b \in \mathbb{Z}_m^*$.

(5) Displaying the Elements.

```
println!("Direct␣product␣of␣Z_{}␣and␣Z_{}^*:", n, m);
for elem in &elements {
        println!("{:?}", elem);
}
```

- The code then prints out all the elements of $\mathbb{Z}_n \times \mathbb{Z}_m^*$.

Understanding the Rust code (conti...)

(6) Computing the Product of Two Elements

```
        let (a, b) = elements[0];
        let (e, f) = elements[1];
        let product = ((a + e) % n, (b * f) % m);
```

- The code shows how to compute the product of two elements from the direct product group. The first component is the sum of $\mathbb{Z}_n$ components modulo n. The second component is the product of $\mathbb{Z}_m^*$ components modulo m.
- It then prints the result of this product.

(7) Finding the Inverse of an Element

```
        let (a, b) = elements[1];
        let inverse_b = units_mod_m.iter().find(|&&x| (b * x) % m == 1).unwrap();
        let inverse = ((-a).rem_euclid(n), *inverse_b);
        println!("Inverse␣of␣({},␣{})␣in␣Z_{}␣x␣Z_{}^*␣is␣{:?}", a, b, n, m, inverse);
}
```

- The code finds the inverse of an element $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_m^*$.

Understanding the Rust code (conti...)

- The inverse of $a \in \mathbb{Z}_n$ is $-a \bmod n$, which is calculated using rem_euclid to ensure the result is non-negative.
- The inverse of $b \in \mathbb{Z}_m^*$ is the element x such that $b \times x \equiv 1 \bmod m$.
- The inverse is then printed.