EoAS Utils

Rajdeep Das (CS2315)

April 22, 2024

1 Introduction

In this document, we present Python code for verifying properties of algebraic structures such as groups and rings. We also provide explanations of the algebraic operations involved.

2 Definitions

2.1 Group

A group $(G, *)$ is a set G equipped with a binary operation $*$ that satisfies the following properties:

1. Closure: For all $a, b \in G$, $a * b \in G$.
2. Associativity: For all $a, b, c \in G$, $(a * b) * c = a * (b * c)$.
3. Identity Element: There exists an element $e \in G$ such that for all $a \in G$, $a * e = e * a = a$.
4. Inverse Element: For every $a \in G$, there exists an element $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$, where e is the identity element.

2.2 Commutativity

A binary operation $*$ on a set G is said to be commutative if for all $a, b \in G$, $a * b = b * a$.

3 Python Implementation

4 Algorithm Explanations

4.1 Associativity Check

The `Associativity_CHK` function checks if a binary operation $*$ on a set S is associative by comparing the results of operations performed in different paren-

Algorithm 1 Closure Check

```
1: function CLOSURE_CHK(S, ∗)
2:     Input: Set S, binary operation ∗
3:     Output: Boolean verdict
4:     verdict ← True
5:     for a in S do
6:         for b in S do
7:             if a ∗ b ∉ S then
8:                 verdict ← False
9:                 return verdict
10:            end if
11:        end for
12:    end for
13:    return verdict
14: end function
```

Algorithm 2 Associativity Check

```
1: function ASSOCIATIVITY_CHK(S, ∗)
2:     Input: Set S, binary operation ∗
3:     Output: Boolean verdict
4:     verdict ← True
5:     for a in S do
6:         for b in S do
7:             for c in S do
8:                 if (a ∗ b) ∗ c ≠ a ∗ (b ∗ c) then
9:                     verdict ← False
10:                    return verdict
11:                end if
12:            end for
13:        end for
14:    end for
15:    return verdict
16: end function
```

thesized orders for all triples of elements in S. If any triple violates associativity, the function returns `False`; otherwise, it returns `True`.

4.2 Identity Check

Algorithm 3 Identity Check

```
function IDENTITY_CHK(S, ∗)
    Input: Set S, binary operation ∗
    Output: Identity element or None
    arr ← []
    for a in S do
        if a ∗ S[0] = S[0] ∗ a = S[0] then
            arr.append(a)
        end if
    end for
    for el in arr do
        cnt ← 0
        for a in S do
            if a ∗ el ≠ a or el ∗ a ≠ a then
                break
            else
                cnt ← cnt + 1
            end if
        end for
        if cnt = len(S) then
            return el
        end if
    end for
    return None
end function
```

5 Algorithm Explanations

5.1 Inverse Check

The `Inverse_CHK` function determines if a set S with a binary operation $*$ has inverse elements for each element in S. It first checks for the existence of an identity element using the `Identity_CHK` function. Then, it iterates over all elements in S and finds their inverses. If all elements have inverses, it returns a list of inverse elements; otherwise, it returns `None`.

Algorithm 4 Inverse Check

```
function INVERSE_CHK(S, ∗)
    Input: Set S, binary operation ∗
    Output: List of inverse elements or None
    identity ← IDENTITY_CHK(S, ∗)
    if identity = None then
        return None
    else
        arr ← [S[i] for i in range(len(S))]
        for i in range(len(S)) do
            flag ← False
            for j in range(i, len(S)) do
                if S[i] ∗ arr[j] = identity and arr[j] ∗ S[i] = identity then
                    arr[i], arr[j] ← arr[j], arr[i]
                    flag ← True
                end if
            end for
            if flag = False then
                return None
            end if
        end for
        return arr
    end if
end function
```

Algorithm 5 Commutativity Check

```
function COMMUTIVITY_CHK(S, ∗)
    Input: Set S, binary operation ∗
    Output: Boolean verdict
    verdict ← True
    for a in S do
        for b in S do
            if a ∗ b ≠ b ∗ a then
                verdict ← False
                return verdict
            end if
        end for
    end for
    return verdict
end function
```

5.2 Commutativity Check

The `Commutivity_CHK` function checks if a binary operation $*$ on a set S is commutative by comparing the results of operations performed in different orders for all pairs of elements in S. If any pair violates commutativity, the function returns `False`; otherwise, it returns `True`.

5.3 Create Composition Table

Algorithm 6 Create Composition Table

```
1:  function CREATE_COMPOSITION_TABLE(S, *)
2:      Input: Set S, binary operation *
3:      Output: Composition table t_arr
4:      t_arr ← [[0 for_i nrange(len(S))] for_i nrange(len(S))]     for i in range(len(S)) do
6:      j in range(len(S))
7:      t_arr[i][j] ← S[i] * S[j]
8:
9:
10:     return t_arr
11:   end function
```

The `Create_Composition_Table` function creates a composition table for a set S with a binary operation $*$. It iterates over all pairs of elements in S and fills in the composition table with the results of their operations.

5.4 Find Order

Algorithm 7 Find Order

```
1:  function FIND_ORDER(S, *, el)
2:      Input: Set S, binary operation *, element el
3:      Output: Order of el
4:      if el ∉ S then
5:          return None
6:      else
7:          cnt ← 1
8:          a ← el * el
9:          while a ≠ el do
10:             a ← a * el
11:             cnt ← cnt + 1
12:         end while
13:         return cnt
14:     end if
15: end function
```

6 Algorithm Explanations

6.1 Generator

Algorithm 8 Generator

```
1: function GENERATOR(el, ∗)
2:     Input: Element el, binary operation ∗
3:     Output: Generated subgroup
4:     arr ← []
5:     arr.append(el)
6:     a ← el ∗ el
7:     while a ≠ el do
8:         arr.append(a)
9:         a ← a ∗ el
10:    end while
11:    identity ← arr.pop()
12:    arr ← [identity] + arr
13:    return arr
14: end function
```

The `Generator` function generates a subgroup of a group with the given element el and binary operation $*$. It repeatedly applies the operation $*$ to el until it returns to el, storing the generated elements in a list.

6.2 isGroup

The `isGroup` function checks if a set S with a binary operation $*$ forms a group. It checks for closure, associativity, identity element, and inverses using the corresponding functions. It returns `True` if S is a group; otherwise, it returns `False`.

6.3 isSubgroup

The `isSubgroup` function checks if a set S is a subgroup of a group G with a binary operation $*$. It verifies that S is a subset of G, and then checks if S forms a group using the `isGroup` function.

6.4 isNormalSubgroup

The `isNormalSubgroup` function checks if a subgroup S is a normal subgroup of a group G with a binary operation $*$. It first verifies if S is a subgroup of G. Then, it checks if S is closed under conjugation by all elements of G.

Algorithm 9 isGroup

```
1: function isGroup(S, ∗, skip)
2:     Input: Set S, binary operation ∗, list skip
3:     Output: Boolean verdict
4:     if not skip[0] then
5:         if not Closure_CHK(S, ∗) then
6:             return False
7:         end if
8:     end if
9:     if not skip[1] then
10:        if not Associativity_CHK(S, ∗) then
11:            return False
12:        end if
13:    end if
14:    if not skip[2] then
15:        if Identity_CHK(S, ∗) = None then
16:            return False
17:        end if
18:    end if
19:    if not skip[3] then
20:        if Inverse_CHK(S, ∗) = None then
21:            return False
22:        end if
23:    end if
24:    return True
25: end function
```

Algorithm 10 isSubgroup

```
1: function isSubgroup(S, G, ∗, skip)
2:     Input: Subgroup S, Group G, binary operation ∗, list skip
3:     Output: Boolean verdict
4:     if len(G) % len(S) ≠ 0 then
5:         return False
6:     end if
7:     for s in S do
8:         if s ∉ G then
9:             return False
10:        end if
11:    end for
12:    return isGroup(S, ∗, skip)
13: end function
```

Algorithm 11 isNormalSubgroup

```
1: function isNormalSubgroup(S, G, ∗, skip)
2:     Input: Subgroup S, Group G, binary operation ∗, list skip
3:     Output: Boolean verdict
4:     if not isSubgroup(S, G, ∗, skip) then
5:         return False
6:     end if
7:     inv_table ← Inverse_CHK(G, ∗)
8:     for i in range(len(G)) do
9:         for s in S do
10:            if G[i] ∗ (s ∗ inv_table[i]) ∉ S then
11:                return False
12:            end if
13:        end for
14:    end for
15:    return True
16: end function
```

Algorithm 12 Ret Left Coset

```
1: function Ret_Left_Coset(el, S, ∗)
2:     Input: Element el, Subgroup S, binary operation ∗
3:     Output: Left coset of S generated by el
4:     arr ← []
5:     for s in S do
6:         arr.append(el ∗ s)
7:     end for
8:     return arr
9: end function
```

6.5 Ret Left Coset

The `Ret_Left_Coset` function computes the left coset of a subgroup S in a group with respect to an element el. It applies the binary operation $*$ to el with each element of S and stores the results in a list.

7 Algorithm Explanations

7.1 Ret Right Coset

Algorithm 13 Ret Right Coset

```
1: function RET_RIGHT_COSET(el, S, *)
2:     Input: Element el, Subgroup S, binary operation *
3:     Output: Right coset of S generated by el
4:     arr ← []
5:     for s in S do
6:         arr.append(s * el)
7:     end for
8:     return arr
9: end function
```

The `Ret_Right_Coset` function computes the right coset of a subgroup S in a group with respect to an element el. It applies the binary operation $*$ to each element of S with el and stores the results in a list.

7.2 Ret Quotient Group

The `Ret_Quotient_Group` function computes the quotient group G/N, where N is a normal subgroup of G. It iterates over all elements of G and computes the left coset of N with respect to each element not already in the appended list, then adds the coset to the quotient group.

7.3 isDistributive

The `isDistributive` function checks if a ring R with addition operation add_opr and multiplication operation Mul_opr satisfies the distributive property. It iterates over all triples of elements in R and verifies if the distributive property holds for both left and right distributivity.

7.4 isRing

The `isRing` function checks if a set R with addition operation add_opr and multiplication operation Mul_opr forms a ring. It verifies if R is a group under addition, if addition is commutative, if R is a group under multiplication, and if the distributive property holds using the corresponding functions.

Algorithm 14 Ret Quotient Group

```
1: function RET_QUOTIENT_GROUP(N, G, ∗)
2:     Input: Normal subgroup N, Group G, binary operation ∗
3:     Output: Quotient group G/N
4:     arr ← []
5:     arr.append(N)
6:     appended ← [s for s in N]
7:     for g in G do
8:         if g in appended then
9:             continue
10:        end if
11:        coset ← RET_LEFT_COSET(g, N, ∗)
12:        arr.append(coset)
13:        for el in coset do
14:            appended.append(el)
15:        end for
16:    end for
17:    return arr
18: end function
```

Algorithm 15 isDistributive

```
1: function ISDISTRIBUTIVE(R, add_opr, Mul_opr)
2:     Input: Ring R, addition operation add_opr, multiplication operation
       Mul_opr
3:     Output: Boolean verdict
4:     for a in R do
5:         for b in R do
6:             for c in R do
7:                 if Mul_opr(a, add_opr(b, c)) ≠
       add_opr(Mul_opr(a, b), Mul_opr(a, c)) or Mul_opr(add_opr(b, c), a) ≠
       add_opr(Mul_opr(b, a), Mul_opr(c, a)) then
8:                     return False
9:                 end if
10:            end for
11:        end for
12:    end for
13:    return True
14: end function
```

Algorithm 16 isRing

```
1: function isRing(R, add_opr, Mul_opr, skip_add, skip_mul, skip_C_D)
2:     Input: Ring R, addition operation add_opr, multiplication operation
   Mul_opr, lists skip_add, skip_mul, skip_C_D
3:     Output: Boolean verdict
4:     if not isGroup(R, add_opr, skip_add) then
5:         return False
6:     end if
7:     if skip_C_D[0] = 0 then
8:         if not Commutivity_CHK(R, add_opr) then
9:             return False
10:        end if
11:    end if
12:    if not isGroup(R, Mul_opr, [skip_mul[0], skip_mul[1], 1, 1]) then
13:        return False
14:    end if
15:    if skip_C_D[1] = 0 then
16:        if not isDistributive(R, add_opr, Mul_opr) then
17:            return False
18:        end if
19:    end if
20:    return True
21: end function
```