A Formalization of Finite Group Theory

David M. Russinoff
david@russinoff.com

November 13, 2023

Formalizing Groups in ACL2

- Challenge: To formalize the notions of *set* and *operation*.
- Previous approaches: `defn-sk`, `encapsulate`
- Observation: Progress beyond Lagrange's Theorem requires induction on group order
- Conclusion: An ACL2 formalization should begin with `(defun groupp (g) ...)`.

A Formalization of Group Theory: Part I (ACL2 Workshop 2022)

First four books of `books/projects/groups/`:

- `lists`: dlists, sublists, disjoint lists, permutations
- `groups`: groups, subgroups
- `quotients`: cosets, normal subgroups, quotient groups
- `cauchy`: *If the order of a group G is divisible by a prime p, then G has an element of order p.*

A Formalization of Group Theory: Part II

Three more books:

- `maps`: homomorphisms, isomorphisms
- `products`: external and internal direct products
- `abelian`: Fundamental Theorem of Finite Abelian Groups: *Every finite abelian group is isomorphic to the direct product of a list of cyclic p-groups, the orders of which are unique up to permutation.*

A Formalization of Group Theory: Part III

Final four books:

- `symmetric`: symmetric and alternating groups
- `actions`: action of a group on a dlist, conjugation of subgroups
- `sylow`: Sylow theorems pertaining to p-subgroups
- `simple`: `(alt 5)` is the smallest non-cyclic simple group

What is a Finite Group?

We define a group to be an operation table, i.e., a matrix of group elements.

For example, the multiplicative group of integers modulo 7:

```
DM!> (z* 7)
((1 2 3 4 5 6)
 (2 4 6 1 3 5)
 (3 6 2 5 1 4)
 (4 1 5 2 6 3)
 (5 3 1 6 4 2)
 (6 5 4 3 2 1))
```

DEFINITIONS

List of group elements:

```
(defmacro elts (g) `(car ,g))
(defun order (g) (len (elts g)))
(defmacro in (x g) `(member ,x (elts ,g)))
```

Group operation is a table look-up:

```
(defmacro ind (x g) `(index ,x (elts ,g)))
(defun op (x y g) (nth (ind y g) (nth (ind x g) g)))
```

Existence of a left identity is built into the definitions:

```
(defun e (g) (caar g))
(defthm group-left-identity
  (implies (in x g)
           (equal (op (e g) x g) x)))
```

DEFINITIONS

Inverse operator conducts a search:

```
(defun inv-aux (x l g)
  (if (consp l)
      (if (equal (op (car l) x g) (e g))
          (car l)
        (inv-aux x (cdr l) g))
    ()))
(defun inv (x g) (inv-aux x (elts g) g))
```

Group recognizer:

```
(defun groupp (g)
  (and (matrixp g (order g) (order g))  ;square matrix
       (posp (order g))                 ;non-nil
       (dlistp (elts g))                ;distinct elts
       (closedp g)                      ;group properties
       (assocp g)
       (inversesp g)))
```

Parametrized Groups

Once we define the element list, group operation, and inverse operation and verify the group axioms, a group is automatically constructed by the `defgroup` macro:

```
(defgroup z+ (n)  ;group name and parameters
  (posp n)        ;parameter constraints
  (ninit n)       ;list of elements: (0 1 ... n-1)
  (mod (+ x y) n) ;group operation
  (mod (- x) n))  ;inverse
```

This defines the group `(z+ n)` and proves several theorems.

Other examples: `(z* n)`, `(quotient g h)`, `(sym n)`, `(direct-product l)`

Parametrized Subgroups

Once we define a sublist of `(elts g)` and prove closure under the group operation and the inverse operator, a subgroup is automatically defined by the `defsubgroup` macro:

```
(defsubgroup cyclic (a) ;subgroup name and parameters
  g                     ;parent group
  (in a g)              ;parameter constraints
  (powers a g))         ;element list
```

This calls `defgroup` to define `(cyclic a g)` and proves that it is a subgroup of `g`.

Other examples: `(center g)`,`(product-group h k g)`, `(group-power n g)`,`(conj-sub h a g)`,`(alt n)`, `(stabilizer s a g)`,`(trivial-subgroup g)`

Comparing Lists of Group Elements as Sets

Various problems arising from the absence of sets are addressed by requiring sublists of `(elts g)`. e.g., subgroups and cosets, to be ordered:

```
(defun ordp (l g)
  (if (consp l)
      (and (in (car l) g)
           (if (consp (cdr l))
               (and (< (ind (car l) g) (ind (cadr l) g))
                    (ordp (cdr l) g))
             (null (cdr l))))
    (null l)))

(defun insert (x l g)
  (if (consp l)
      (if (equal x (car l))
          l
        (if (< (ind x g) (ind (car l) g))
            (cons x l)
          (cons (car l) (insert x (cdr l) g))))
    (list x)))
```

Permutations

Permutation of an arbitrary list:

```
(defun permutationp (l m)
  (if (consp l)
      (and (member-equal (car l) m)
           (permutationp (cdr l) (remove1-equal (car l) m)))
    (endp m)))
```

Permutation of a dlist:

```
(defund permp (l m)
  (and (dlistp l) (dlistp m)
       (sublistp l m) (sublistp m l)))

(defthmd permp-permutationp
  (implies (and (dlistp l) (dlistp m))
           (iff (permutationp l m)
                (permp l m))))
```

`(perms l)` is a list of all permutations of a dlist `l`.

Symmetric Groups

Element list, group operation, and inverse operator:

```
(defund slist (n) (perms (ninit n)))

(defun comp-perm-aux (x y l)
  (if (consp l)
      (cons (nth (nth (car l) y) x)
            (comp-perm-aux x y (cdr l)))
    ()))
(defund comp-perm (x y n)
  (comp-perm-aux x y (ninit n)))

(defun inv-perm-aux (x l)
  (if (consp l)
      (cons (index (car l) x)
            (inv-perm-aux x (cdr l)))
    ()))
(defund inv-perm (x n)
  (inv-perm-aux x (ninit n)))
```

Symmetric Groups

Once we have proved the group axioms, we invoke `defgroup`:

```
(defgroup sym (n)
  (posp n)              ;parameter constraint
  (slist n)             ;element list
  (comp-perm x y n)     ;group operation
  (inv-perm x n))       ;inverse operator
```

Computations in `(sym n)`:

```
DM !>(op '(2 1 3 0) '(1 3 0 2) (sym 4))
(1 0 2 3)

DM !>(inv '(1 2 0 4 5 3) (sym 6))
(2 0 1 5 3 4)
```

ALTERNATIVE FORMULATION OF `permutationp`

Based on the number of occurrences of each member of a list:

- `(hits x l)` counts the number of occurrences of `x` in `l`.
- `(hits-diff l m)` searches `(append l m)` for `x` such that `(hits x l)` $\neq$ `(hits x m)`.

If every element has the same number of occurrences in `l` as in `m`, then `l` is a permutation of `m`:

```
(defthmd hits-diff-perm
  (iff (permutationp l m)
       (not (hits-diff l m))))
```

MAPS

A *map* is an alist representing a function:

```
(defund domain (m) (strip-cars m))
(defund mapp (m) (and (cons-listp m) (dlistp (domain m))))
(defund mapply (map x) (cdr (assoc-equal x map)))
```

Maps may be defined by the `defmap` macro:

```
(defmap compose-maps (map2 map1)  ;name, parameters
  (domain map1)                   ;domain
  (mapply map2 (mapply map1 x)))  ;value
```

This defines `(compose-maps map2 map1)` and derives its basic properties.

HOMOMORPHISMS

A homomorphism from `g` to `h` is a map `m` such that if `x` and `y` are elements of `g`, then

(1) `(in (mapply m x) h)`

(2) `(mapply m (op x y g))`
 = `(op (mapply m x) (mapply m y) h)`

(3) `(mapply m (e g))` = `(e h)`

```
(defund homomorphismp (m g h)
  (and (groupp g)
       (groupp h)
       (mapp m)
       (sublistp (elts g) (domain m))
       (not (codomain-cex m g h))       ;no counterexample of (1)
       (not (homomorphism-cex m g h))   ;no counterexample of (2)
       (equal (mapply m (e g)) (e h))))
```

Image of a Homomorphism

Given a homomorphism `map` from `g` to `h`, `(ielts map g h)` is the list of images of elements of `g`, defined (using `insert`) to be an ordered sublist of `(elts h)`.

This forms a subgroup of `h`:

```
(defsubgroup image (map g) h
  (homomorphismp map g h)
  (ielts map g h))
```

An *epimorphism* is a surjective homomorphism:

```
(defund epimorphismp (map g h)
  (and (homomorphismp map g h)
       (equal (image map g h) h)))
```

KERNEL OF A HOMOMORPHISM

Given a homomorphism `map` from `g` to `h`, `(kelts map g h)` is the ordered sublist of `(elts g)` that are mapped to `(e h)`.

This forms a subgroup of `g`:

```
(defsubgroup kernel (map h) g
  (homomorphismp map g h)
  (kelts map g h))
```

An *endomorphism* is an injective homomorphism:

```
(defund endomorphismp (map g h)
  (and (homomorphismp map g h)
       (equal (kernel map h g) (trivial-subgroup g))))
```

An *isomorphism* is a bijective homomorphism:

```
(defund isomorphismp (map g h)
  (and (epimorphismp map g h)
       (endomorphismp map g h)))
```

Direct Products

The element list of the direct product of a list of groups `l`:

```
(defun group-tuples-aux (l m)
  (if (consp l)
      (append (conses (car l) m)
              (group-tuples-aux (cdr l) m))
    ()))

(defun group-tuples (l)
  (if (consp l)
      (group-tuples-aux (elts (car l)) (group-tuples (cdr l)))
    (list ())))
```

Direct Products

The group operation:

```
(defun dp-op (x y l)
  (if (consp l)
      (cons (op (car x) (car y) (car l))
            (dp-op (cdr x) (cdr y) (cdr l)))
    ()))
```

The inverse operator:

```
(defun dp-inv (x l)
  (if (consp l)
      (cons (inv (car x) (car l))
            (dp-inv (cdr x) (cdr l)))
    ()))
```

Direct Products

Once the group axioms are proved, we invoke `defgroup`:

```
(defgroup direct-product (l)
  (and (group-list-p l)    ;parameter constraints
       (consp l))
  (group-tuples l)         ;element list
  (dp-op x y l)            ;group operation
  (dp-inv x gl))           ;inverse operator
```

Internal Direct Products

Requirements of an internal direct product:

```
(defun internal-direct-product-p (l g)
  (if (consp l)
      (and (internal-direct-product-p (cdr l) g)
           (normalp (car l) g)
           (equal (group-intersection
                     (car l)
                     (product-group-list (cdr l) g) g)
                  (trivial-subgroup g)))
      (null l)))
```

Internal Direct Products

Representation of `g` as an internal direct product:

```
(defthmd isomorphismp-dp-idp
  (implies (and (groupp g)
                (consp l)
                (internal-direct-product-p l g)
                (= (product-orders l) (order g)))
           (isomorphismp (product-list-map l g)
                         (direct-product l)
                        g)))
```

Appending internal direct products:

```
(defthmd internal-direct-product-append
  (implies (and (internal-direct-product-p l g)
                (internal-direct-product-p m g)
                (equal (group-intersection (product-group-list l g)
                                           (product-group-list m g)
                                          g)
                       (trivial-subgroup g)))
           (internal-direct-product-p (append l m) g)))
```

FACTORIZATION OF AN ABELIAN p-GROUP

Fundamental lemma:

```
(defthm factor-p-group
  (implies (and (p-groupp g p)
                (abelianp g)
                (in a g)
                (equal (ord a g) (max-ord g)))
           (let ((g1 (cyclic a g)) (g2 (g2 a p g)))
             (and (internal-direct-product-p (list g1 g2))
                  (equal (* (order g1) (order g2))
                         (order g))))))
```

Factorization of an Abelian p-Group

Every abelian p-group is an internal direct product of cyclic groups:

```
(defun cyclic-p-subgroup-list (p g)
  (if (and (p-groupp g p) (abelianp g) (> (order g) 1))
      (if (cyclicp g)
          (list g)
        (let ((a (elt-of-ord (max-ord g) g)))
          (cons (cyclic a g)
                (cyclic-p-subgroup-list p (g2 a p g)))))
    ()))

(defthmd p-group-factorization
  (implies (and (p-groupp g p) (abelianp g) (> (order g) 1))
           (let ((l (cyclic-p-subgroup-list p g)))
             (and (consp l)
                  (cyclic-p-group-list-p l)
                  (internal-direct-product-p l g)
                  (equal (order g) (product-orders l))))))
```

Factorization of an Abelian Group

The ordered list of all elements of g with order dividing m:

```
(defun elts-of-ord-dividing-aux (m l g)
  (if (consp l)
      (if (divides (ord (car l) g) m)
          (cons (car l) (elts-of-ord-dividing-aux m (cdr l) g))
        (elts-of-ord-dividing-aux m (cdr l) g))
    ()))

(defund elts-of-ord-dividing (m g)
  (elts-of-ord-dividing-aux m (elts g) g))
```

If g is abelian, then these elements form a subgroup of g:

```
(defsubgroup subgroup-ord-dividing (m) g
  (and (abelianp g) (posp m))
  (elts-of-ord-dividing m g))
```

Factorization of an Abelian Group

Fundamental lemma:

```
(defthmd rel-prime-factors-product
  (implies (and (groupp g)
                (abelianp g)
                (posp m)
                (posp n)
                (= (gcd m n) 1)
                (= (order g) (* m n)))
           (let ((h (subgroup-ord-dividing m g))
                 (k (subgroup-ord-dividing n g)))
             (and (equal (group-intersection h k g)
                         (trivial-subgroup g))
                  (equal (* (order h) (order k))
                         (order g))))))
```

FACTORIZATION OF AN ABELIAN GROUP

We define a list of subgroups of `g` recursively, using `cyclic-p-subgroup-list`:

```
(defun cyclic-subgroup-list (g)
  (if (and (groupp g)
           (abelianp g))
      (if (= (order g) 1)
          ()
        (let* ((p (least-prime-divisor (order g)))
               (m (max-power-dividing p (order g)))
               (n (/ (order g) m))
               (h (subgroup-ord-dividing m g))
               (k (subgroup-ord-dividing n g)))
          (append (cyclic-p-subgroup-list p h)
                  (cyclic-subgroup-list k))))
    ()))
```

Factorization of an Abelian Group

The following is proved by induction:

```
(defthmd idp-cyclic-subgroup-list
  (implies (and (groupp g) (abelianp g) (> (order g) 1))
           (let ((l (cyclic-subgroup-list g)))
             (and (cyclic-p-group-list-p l)
                  (internal-direct-product-p l g)
                  (equal (product-orders l) (order g))))))
```

Finally, we invoke `isomorphismp-dp-idp`:

```
(defthmd abelian-factorization
  (implies (and (groupp g) (abelianp g) (> (order g) 1))
           (let ((l (cyclic-subgroup-list g)))
             (and (cyclic-p-group-list-p l)
                  (isomorphismp (product-list-map l g)
                                (direct-product l)
                                g)))))
```

Uniqueness of the Factorization

```
(defthmd abelian-factorization-unique
  (implies (and (consp l) (cyclic-p-group-list-p l)
                (consp m) (cyclic-p-group-list-p m)
                (isomorphismp map (direct-product l)
                                  (direct-product m)))
           (permutationp (orders l) (orders m))))
```

Proof sketch:

- `p` = `(first-prime l)`
- `l'` = `(delete-trivial (group-power-list l p))`
- `m'` = `(delete-trivial (group-power-list m p))`
- `l'` and `m'` inherit the hypotheses of the theorem
- By induction, `(permutationp (orders l') (orders m'))`
- Every `x` has same hit count in `(orders l')` as in `(orders m')`
- Every `x` has same hit count in `(orders l)` as in `(orders m)`
- `(permutationp (orders l) (orders m))`

Future Work

Linear algebra:

- Fields
- Matrix algebra and systems of linear equations
- Vector spaces and linear transformations

Galois theory:

- Polynomials and factorization
- Algebraic extensions and number fields
- Galois groups