Implementation of Algebraic Libraries in HOL4

Hing-Lun Chan

College of Engineering and Computer Science
Australian National University

PhD Yearly Review Talk 2013

Mechanisation of AKS Algorithm – Road Map

- Foundation Work:
  - Build Monoid theory in HOL4.
  - Build Group theory from Monoid theory.
  - Build Ring theory using Group and Monoid.
  - Build Field theory using Ring and Group.
  - Build Polynomial theory using Field and Ring.
- Apply to AKS:
  - Code in HOL4: *AKS* n that returns true or false upon input n.
  - Prove in HOL4: *AKS* n returns true iff n is prime.
  - Prove in HOL4: number of steps of *AKS* n is bound by $O(\log^k n)$.

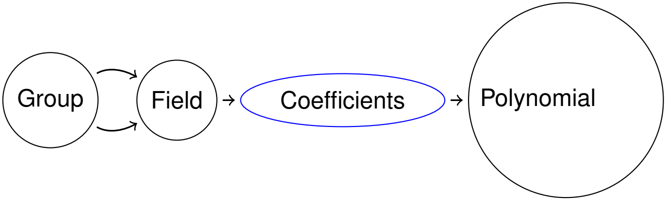
Mechanisation of AKS Algorithm – Road Map

- Foundation Work:
  - Build Monoid theory in HOL4.($\surd$)
  - Build Group theory from Monoid theory.($\surd$)
  - Build Ring theory using Group and Monoid.($\surd$)
  - Build Field theory using Ring and Group.($\surd$)
  - Build Polynomial theory using Field and Ring.($\surd$)
- Apply to AKS:
  - Code in HOL4: *AKS n* that returns true or false upon input n.
  - Prove in HOL4: *AKS n* returns true iff n is prime.
  - Prove in HOL4: number of steps of *AKS n* is bound by $O(\log^k n)$.

Outline

First Attempt

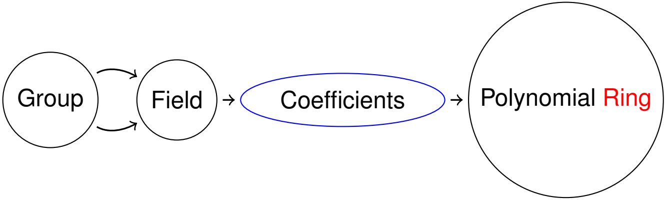
- Quick start:



- Group: $(G, \circ)$ with binary operation $\circ$ satisfying 4 axioms: (1) closure, (2) associativity, (3) identity and (4) inverse.
- Field: $(F, +, \times)$ with Group $(F, +)$, Group $(F^*, \times)$, and distributive law holds for $+$ and $\times$.
  (Here: X^* = non-zeroes in X)

First Attempt

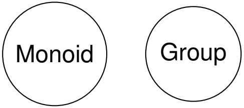
- Quick start:



- Group: $(G, \circ)$ with binary operation $\circ$ satisfying 4 axioms: (1) closure, (2) associativity, (3) identity and (4) inverse.
- Field: $(F, +, \times)$ with Group $(F, +)$, Group $(F^*, \times)$, and distributive law holds for $+$ and $\times$.
  (Here: X^* = non-zeroes in X)
- 0 - polynomial is a polynomial, 1$/$ polynomial is not a polynomial.
  Ring: $(R, +, \times)$ with Group $(R, +)$, but $(R^*, \times)$ a broken group.

Next Attempt

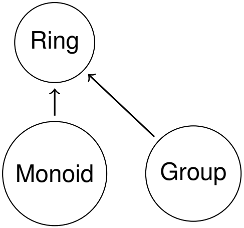
- Introduce Monoid (a broken Group):



- Monoid: $(M, \circ)$ with binary operation $\circ$ satisfying 3 axioms.
- Group: $(G, \circ)$ with binary operation $\circ$ satisfying 4 axioms.

Next Attempt

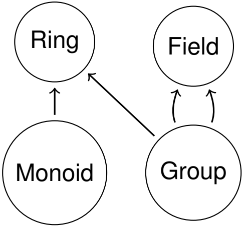
- Introduce Monoid (a broken Group):



- Monoid: $(M, \circ)$ with binary operation $\circ$ satisfying 3 axioms.
- Group: $(G, \circ)$ with binary operation $\circ$ satisfying 4 axioms.
- Ring: $(R, +, \times)$ with Group $(R, +)$, Monoid $(R, \times)$, and distributive law holds for $+$ and $\times$.

Next Attempt

- Introduce Monoid (a broken Group):



- Monoid: $(M, \circ)$ with binary operation $\circ$ satisfying 3 axioms.
- Group: $(G, \circ)$ with binary operation $\circ$ satisfying 4 axioms.
- Ring: $(R, +, \times)$ with Group $(R, +)$, Monoid $(R, \times)$, and distributive law holds for $+$ and $\times$.
- Field: $(F, +, \times)$ with Group $(F, +)$, Group $(F^*, \times)$, and distributive law holds for $+$ and $\times$.

Discussion

- Monoid and Group have many things in common – they are not distinct.
- Should it be that: Monoid = broken Group ?
- Or it should be: Group = better Monoid ?

Discussion

- Monoid and Group have many things in common – they are not distinct.
- Should it be that: Monoid = broken Group ?
- Or it should be: Group = better Monoid ?
- Conceptually, it doesn't matter.
- For implementation, it does matter.

Monoid in HOL4 – part 1

- Define the datatype:

```
Hol_datatype`
  monoid = <| carrier: 'a -> bool;
                      op: 'a -> 'a -> 'a;
                      id: 'a
              |>`;
```

- Use overloads:

```
overload_on ("*", ``g.op``);
overload_on ("#e", ``g.id``);
overload_on ("G", ``g.carrier``);
```

Monoid in HOL4 – part 2

- Definition:

```
val Monoid_def = Define`
  Monoid (g:'a monoid) =
   (* Closure *)
      (!x y. x IN G /\ y IN G ==>
          x * y IN G) /\
   (* Associativity *)
      (!x y z. x IN G /\ y IN G /\ z IN G ==>
         ((x * y) * z = x * (y * z))) /\
   (* Identity *)
      #e IN G /\ (!x. x IN G ==>
         (#e * x = x) /\ (x * #e = x))
 `;
```

Invertibles in Monoid

- The invertibles:

```
val invertibles_def = Define`
    invertibles (g:'a monoid) =
    { x | x IN G /\ (?y. y IN G /\
     (x * y = #e) /\ (y * x = #e)) }`;
overload_on ("G*", ``invertibles g``);
```

- Extract function by existence (Skolemization):

```
 |- !g x. Monoid g /\ x IN G* ==>
 monoid_inv g x IN G /\ (x * monoid_inv g x = #e)
                    /\ (monoid_inv g x * x = #e)
```

- Extend the data structure:

```
add_record_field ("inv", ``monoid_inv``);
overload_on ("|/", ``g.inv``);
```

Interesting Sub-Monoid

- The monoid of invertibles:

```
val Invertibles_def = Define`
  Invertibles (g:'a monoid) : 'a monoid =
    <| carrier := G*;
               op := g.op;
               id := g.id
     |>`;
```

- Properties of invertibles:

```
|- !g. Monoid g ==> #e IN G*
|- !g. Monoid g ==> !x y. x IN G* /\ y IN G*
                            ==> x * y IN G*
|- !g. Monoid g ==> !x. x IN G* ==> |/ x IN G*
```

- In fact, it is easy to show:

```
|- !g. Monoid g ==> Monoid (Invertibles g)
```

Group in HOL4 – part 1

- Make Group share the same datatype as Monoid:

```
type_abbrev ("group", Type `:'a monoid`);
```

- Define Group to be a better Monoid:

```
val Group_def = Define`
  Group (g:'a group) =
    Monoid g /\ (G* = G)
`;
```

- Group = Monoid with all its elements invertible.
- Stronger version of previous result:

```
|- !g. Monoid g ==> Group (Invertibles g)
```

Group in HOL4 – part 2

- Alternate definition of Group:

```
|- !g:'a group. Group g =
(* Closure *)
  (!x y. x IN G /\ y IN G ==> x * y IN G) /\
(* Associativity *)
  (!x y z. x IN G /\ y IN G /\ z IN G ==>
     ((x * y) * z = x * (y * z))) /\
(* Identity *)
  #e IN G /\ (!x. x IN G ==> (#e * x = x)) /\
(* Inverse *)
  (!x. x IN G ==> ?y. y IN G /\ (y * x = #e))
```

Discussion

Advantages of having Group = better Monoid:

- By sharing the same datatype, HOL4 can treat them as equal structures.
- Group theory can now be built upon Monoid theory – no longer separate theories.
- That is, no need to prove similar theorems for two structures.
- This facilitates export rewrites in HOL4.
- Indeed, Monoid theorems can be mechanically transformed as Group theorems – this is "theorem lifting".

Ring – part 1

- Define datatype for Ring:

```
Hol_datatype`
  ring = <| carrier: 'a -> bool;
                sum: 'a group;
               prod: 'a monoid
         |>`;
```

- Use a bunch of overloading:

```
overload_on ("+", ``r.sum.op``);
overload_on ("*", ``r.prod.op``);
overload_on ("R", ``r.carrier``);
overload_on ("#0", ``r.sum.id``);
overload_on ("#1", ``r.prod.id``);
```

Ring – part 2

- Define Ring:

```
val Ring_def = Define`
  Ring (r:'a ring) =
   (* Additive Component *)
     AbelianGroup r.sum  /\
     (r.sum.carrier = R) /\
   (* Multiplicative Component *)
     AbelianMonoid r.prod /\
     (r.prod.carrier = R) /\
   (* Distributive Law *)
     (!x y z. x IN R /\ y IN R /\ z IN R ==>
        (x * (y + z) = (x * y) + (x * z)))`;
```

Field

- Be smart:
  apply the same technique as Group = better Monoid.
- Make Field share the same datatype as Ring:

```
type_abbrev ("field", Type `:'a ring`);
```

- Define Field to be a better Ring:

```
val Field_def = Define`
  Field (r:'a field) =
    Ring r /\
    Group (r.prod excluding #0)
`;
```

- Field = Ring with non-zero elements form a multiplicative Group.

Discussion

Advantages of having Field = better Ring:

- HOL4 can treat Ring and Field as equal.
- Can lift Ring theorems as Field theorems.
- Reuse of op-iteration:
  - Repeat application of $+$ gives ring numerals:
    #1, #1 + #1 = #2, #1 + #2 = #3, . . .
  - Repeat application of $\times$ gives ring exponentials:
    $z = z^1, z \times z = z^2, z \times z^2 = z^3, \ldots$
- Build Field theory upon Ring theory:
  more theorems, less work.
- Give the HOL4 Algebraic Libraries a solid foundation.