Archtrait: Domain specific language for the representation of Abstract Algebra

Type system

We have devised a rich and flexible type system to aid in expressing complex algebraic concepts. Each type is assigned one or more of the five Archtraits, which are as follows

Group

INSERT DEFINITION OF GROUP

Members:

- 32/64 bit signed/unsigned integers: Our language does not treat integers and reals as primitive data types.
- Boolean
- Add some more. Cyclic groups, maybe enums?

Ring

INSERT DEFINITION OF RING

Members:

- BigInt
- Matrices
  - A matrix is treated as a generic over any type, but the methods it provides wil depend on the Archtrait of that type. Some types, such as files (see below) clearly cannot be added or subtracted, and so the matrix of files reduces to a buffer.
- Polynomials
  - Similar to matrices, they can be generic over any type.
- Add more.

Field

INSERT DEFINITION OF FIELD

Members:

- reals
- complex numbers
- BigRationals
- Non-Singular matrix (multiple traits)
- Polynomials over a field (multiple traits)

Space

Vector space

Members:

- string
- array
- Any more?

System

These are the data types/ objects offered by the system.

- Files, IO
- Pointers
- Do we need anything else?

Programming language constructs

Operators

- Relational: >, <, ==
- Logical: &&, ||, !
- Arithmetic: +, *, -, /
- Shifts, etc.

All the operators have the same meaning as in C, with enhanced functionality for non-C types (matrices, for example).

Conditionals

- The keywords `if` and `else` are used as in standard languages. Any members of a group, ring or field may be compared using relational operators (again, the standard >, < and ==) as part of the predicate. Booleans are already a group.
- The body of statements is enclosed in curly braces.
- The syntax:

```
if <pred> {
    <body>
```

```
}
else if <pred> {
    <body>
}
else {
    <body>
}
```

Loops

- Using the `for` and `in` keywords, we can iterate over the members of a space.

```
for <member> in <space> {
    <body>
}
```

- The `while` keyword can be used with a predicate as usual.

```
while <pred> {
    <body>
}
```

Statements

- All statements end with a semicolon.
- Statements which declare/initialise new variables must begin with a let. The type of the variable must be specified also. (Rust syntax)

```
let a: u32 = 0;
a = 1;
let m: Matrix<real> = Matrix<real>::new(<dimensions>)
```

Functions

Function prototypes begin with the `do` keyword, followed by the function name, the arguments within parentheses, and then the return type.

```
do <name>  (<args>) : <return type> {
    <body>
}
```

Function calls begin with the `call` keyword.

```
call <name>(<args>)
```

Custom types

Programmers can create their own groups, rings and fields with the `ink` keyword.

```
ink <name> : <Archtrait> {
    <Archtrait functionality>
}
```

Within the body, the programmer must specify the operations on the new type, viz. addition, multiplication.

The System Archtrait cannot be `ink`ed.