Archetype: Domain specific language for the representation of Abstract Algebra

INTRODUCTION

We seek to make a language which can be used to represent and manipulate algebraic structures, which we call Archetypes. The language is designed to be used by mathematicians, and so the syntax is designed to be similar to mathematical notation without being dense or hard to learn.

Syntax

Statements

- The language is case sensitive.
- 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;
```

Comments

- Single line comments begin with `//`.
- Multi-line comments begin with `/*` and end with `*/`.
- Comments cannot be nested.

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

- The `for` keyword is used to iterate over a range of values. The syntax is similar to C.

```
for (declaration; predicate; operation) {
    .
    .
    .
}
```

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

```
for (member in list) {
    .
    .
    .
}
```

Where list is a vector over type `T`, and thus the type of `member` is also `T`.

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

```
while (predicate) {
    .
    .
    .
}
```

Functions

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

```
do <name>  (arg: type, ...) : <return type> {
    <body>
}
```

Functions calls are identical to C: `name(args)`.

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}(change to four?) Archetypes, which are as follows

Structs

```
struct name {
    members
}
```

Enums

```
enum name {
    members
}
```

Archetypes

Group

INSERT DEFINITION OF GROUP

Members:

Ring

INSERT DEFINITION OF RING

Members:

- Unsigned integers `u8`, `u16`, `u32`, and `u64` : Our language does not treat integers and reals as primitive data types.
- `BigInt`: Unbounded integer type, similar to integers in Python.
- `Matrix`
  - A matrix is treated as a generic over any type, but the methods it provides wil depend on the Archetype of that type. For example, a

    matrix of integers will not have the inverse operation, because the inverse of a matrix of integers may contain reals.
- `Polynomial`
    - Similar to matrices, they can be generic over any type.

Field

INSERT DEFINITION OF FIELD

Members:

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

Vector

- Similar to vectors in C++ and Java.
- They are generic over any type, and the operations they provide will depend on the Archetype of that type.

System type

- These are the data types/objects offered by the system, and while they may be represented using algebraic constructs (aka Archetypes), those structures are relatively more complex and esoteric. Naturally, the programmer may use these types to build more complex structures.
- Wrapping a `System` type within a `struct` allows the programmer to claim an Archetype for these types, and thus use them in algebraic operations.
- Pointers
- Boolean
- Strings
- {Tuples}(? Or is the cartesian product of two Archetypes also an Archetype?)

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

The claim keyword

To create a new instance of an Archetype, the programmer may use the claim keyword. The syntax is as follows:

```
claim (name is Archetype) {
    implement operations
};
```

And one can check whether a type has claimed an Archetype using

```
if (name is Archetype) {
    .
    .
    .
}
```

where name is the name of a type that has already been declared (struct). The new type may directly implement the operations, or define mappings (morphisms) to some other type which implements the operations. {The new type may also implement the operations by inheriting them from some other type which implements them}(Line by copilot, must confer).

Tokens

Reserved words

- `let`
- `if`
- `else`
- `for`
- `while`
- `do`
- `claim`
- `is`
- `struct`
- `enum`
- `true`
- `false`

Data types

- `BigInt`
- `Matrix`
- `Polynomial`
- `u8`
- `u16`
- `u32`
- `u64`
- `real`
-