Archetype: an Overview - Compilers Group 2

Abhay Shankar K: cs21btech11001 Kartheek Tammana: cs21btech11028
Prasham Walvekar: cs21btech11047 Sumedh Kashikar: es21btech11033

Introduction

Archetype is a DSL for abstract algebra. Generally, the field is under-represented among DSL's due to its complexity. Archetype aims to make the field more accessible to programmers by providing a simple and intuitive syntax coupled with a rich and versatile type system.

Abstract algebra deals with mathematical structures, which is reflected in the Archetype type system. The type system is designed to be as close to the mathematical definition of the structures as possible. This allows for a more natural expression of mathematical concepts in code.

Challenges

The main challenge in designing Archetype was to create a type system that is both expressive and intuitive. The type system should be able to express the mathematical structures in a natural way, while also being easy to understand for programmers who are not familiar with the mathematical definitions.

- Implementing Generics
- Implementing Forges, a powerful user-defined way to cast between types.
- Enum Functionality: We had originally allowed enums to do a lot more - however, C++ does not have a corresponding feature, so we had to remove it.

Providing standard library

A set of functions and types that are available for use, with pre-generated code detailing their operations and behaviour.

Semantic analysis

The type system also resulted in a relatively complex semantic analyser, with multiple symbol tables and pervasive type casting.

Pipeline and Execution

```
bash run.sh test.arc
```

The above script compiles the compiler, then compiles the archetype file, and also runs the executable generated. No further configuration is required.

File Structure

- `codes/` contains the source code for the compiler
  - `code/` contains the core language, including lex/yacc files.
  - `obj/` contains the object files generated during compilation.
  - `std/` contains the standard library.
  - `Makefile` is the makefile to compile the compiler.

  - `run.sh` is the script to run the compiler on a file.
- `testcases/` contains all the test cases
  - `Phase[2|3|4]_testcases/` contains the test cases for each phase. For 2 and 3, there are 4 each of success (first 4) and failure (last 4) test cases. For phase 4, there are 4 success cases - code generation must not fail.