Generics Programming in SLAC

Compiler Construction 2016 Final Report

Michael Meyer Trace Powers

KTH Royal Institute of Technology

{meye}@kth.se

1. Introduction

In the required parts of the compiler project, our task was to implement the entire pipeline of lexing, parsing, name analysis, type checking, and code generation.

Each of these specific phases transforms the current representation of the

2. Examples

Give code examples where your extension is useful, and describe how they work with it. Make sure you include examples where the most intricate features of your extension are used, so that we have an immediate understanding of what the challenges are.

You can pretty-print code like this:

With generics we can get started into basic data structures. First, we can implement Linked Lists.

```
class LinkedList[T1] {
  var value : T1;
  var link : LinkedList[T1];
  method get() : T1 = {
    value
  }
  method set(v : T1) = {
    value = v
  }
  method next() : LinkedList[T1] = {
    link
  }
}
```

This allows us to further define things such as Stacks and Queues...

```
class Stack[T1] {
  var top : LinkedList[T1]

}
```

By using generics, we can provide a richer SLAC standard library, making the language much more powerful.

3. Implementation

This is a very important section, you explain to us how you made it work.

Describe all non-obvious tricks you used. Tell us what you thought was hard and why. If it took you time to figure out the solution to a problem, it probably means it wasn't easy and you should definitely describe the solution in details here. If you used what you think is a cool algorithm for some problem, tell us. Do not however spend time describing trivial things (we know what a tree traversal is, for instance). Cite any reference work you used like this [Appel 2002].

One issue that came up when testing was that multiple casts of the generic caused the compiler to fail. Everything worked fine in a single cast case, but the second cast caused the type checking on the first one to fail. After debugging this for a few hours, we found that the issue was hidden not in the type checking or name analysis, but in the Generic phase of the compiler. This was because the tree cloning process was sharing some Identifier tree nodes. This caused the setSymbol command to overwrite each other.

4. Possible Extensions

This system only supports one type generic classes. This makes it more difficult to implement data structures such as maps, although there probably is a special way to implement them using structures such as tuples.

However, this is still an inconvience for the programmer and an arbitrary amount of generic types should be supported in the future. Since generics are just one solution for the general issue of code reusability, certain

features can be tacked on to a basic generics program. In certain cases, most code of a class can be represented with generics, except for a single function, such as a library call. It would still be useful to have generics for this situation, but it can't be done using the type replacement mechansism. By implementing first class functions and then passing a function in to the type generator as well, this issue could be alieviated. Of course, a wrapper class could be written instead but this is what the generic is trying to replace.

References

A. W. Appel. *Modern Compiler Implementation in Java*. Cambridge University Press, 2nd edition, 2002.