Towards Interfaces for Chapel

Chris Wailes and Jeremy G. Siek
Indiana University
{cwailes,jsiek}@indiana.edu

As software projects become larger and more complicated proposals have arisen for ways to increase programmer productivity. One such proposal is *constrained generics*, which are a method of specifying which types may be used with a generic class or function. Our project has been working to add constraints to the generic classes and functions already present in the Chapel language. It is our hope that constraints will make error messages involving generics more useful, allow generics to be type-checked independently from the code that calls it, and prevent the accidentally hijacking of function calls inside generics.

To this end we have developed syntax and semantics for the use of constrained generics in Chapel and begun modification of the compiler. Our design decisions will allow programmers to use the current unconstrained generics, the new constrained generics, or a combination of the two forms of functions. An example of the new syntax can be seen in Figure 1 where we can see the definition of an `interface` named `Monoid`, an `implements` declaration that says type, in this case `int`, meets the interface's requirements, and the use of the interface to constrain the acceptable values of the type variable for the generic `accumulate` function. The code in Figure 1 finished with the construction of a list and a call to `accumulate`, which implicitly instantiates the generic to use the implementation of `Monoid(int)`.

Chapel Implementers and Users Workshop May, Phoenix, Arizona.


```
interface Monoid(T) {
  proc binary_op(T,T) : T;
  proc identity_elt() : T;
}

proc binary_op(a: int, b :int) : int {
  return a + b;
}
proc identity_elt() : int {
  return 0;
}
implements Monoid(int);

proc accumulate(ls : list(?U))
  where { implements Monoid(U) }
{
  ...
}

var L : list(int) = makeList(1,2,3);
writeln(accumulate(L));
```

Figure 1: An example use of constrained generics.

In this talk we give an status report on our ongoing work towards implementing constrained generics and we describe some of the challenges that we have encountered in the development of this feature. Some of the challenges are inherent in the language, such as the rules for function overload resolution and some of them concern the implementation, such as understanding the complex invariants of Chapel's abstract syntax tree and which internal functions establish or break those invariants. We suggest modifications to the language and compiler that could help resolve these challenges.