Project Proposal

First-class functions

11 December 2012

Stanislas Signoud
Michaël Schneeberger

What is this all about ?

We would like to add **functions as a first-class citizen**, enabling programmers to store functions as values in the Tool language. With a simple syntax (`(Type1, Type2...) => ReturnType`), as well as type-checking and context awareness (*closures*), it opens new ways to write code in Tool.

Code exemples

```
class UnitTests {
  def testWithSimpleCall() : Bool = {
     var i: Int;
     var func: Int => Bool;
     i = 1;

     func = (test : Int) => { return test == 1; };

     return func(i); // should return true
  }

  def testWithImmediateInlineCall() : Bool = {
     var i: Int;
     i = 2;

     return ((test : Int) => { return test == 2; })(i); // should return true
  }

  def testWithContext() : Bool = {
     var context: Int;
     var func: Int => Bool;
     i = 1;
     context = 2;

     func = (test : Int) => { return test + 1 == context; };

     return func(i); // should return true
  }
```

```
  def testWithUnit() : Bool = {
     var i: Int;
     var j: Int;
     var func : Unit => Unit;
     i = 0;
     j = 0;

     func = () => { i++; };

     while(j < 3) {
        func();
        j++;
     }

     return (i == j); // should return true
  }
}
```

Note: Our compiler currently doesn't compile at all this code (syntax errors).

How to achieve that?

First, we will create the corresponding type family.

Then, we will implement the – quite challenging – syntax, with its specificities (arrows, parenthesis, arguments declarations like methods, statements like methods too, etc.).

We will create a new type familly, TFunction; it will takes arguments types and return type as paramaters and will enable type-checking. The TUnit type will also be created to enable non-returning functions, useful for some code patterns and to preserve syntax consistency (enabling `Unit => Unit`) to describe void methods that only affects context.

We will implement analysis, ignoring closures and trying to create a simple function. To do that, we will have to create a simple Object on the Java-side, with a reference stored in variables, and a simple method “apply” to call the actual statements.

When that will be done, we will have to understand how the *closures* aspects can be implemented in our code and what will be the consequences.

What's next?

In a addition to these improvements, we could work on anonymous inline class declarations, runtime method injecting (creating methods at runtime) or how to implement Prototype-Oriented Programming in Tool (as in ECMAScript).