PLT Fall 2018

Shoo
A Project Proposal

Claire Adams (cba2126)
Samurdha Jayasinghe (sj2564)
Rebekah Kim (rmk2160)
Cindy Le (xl2738)
Crystal Ren (cr2833)

September 18, 2018

Contents

1 Language Introduction

Shoo is a less functionally robust version of Go with slightly more C-like syntax. It will emulate Go in the way it spins off new threads via Shoo-routines, how pipes, which are similar to message queues, are used to communicate between threads, and the inclusion of first class functions.

2 Basic Syntax

We feel that it's easiest to learn by example. Here's a run-through of the basic syntax:

```
/* (This is a multi-line comment.) The primitive types are int, float, char,
    string, and bool. Advanced types include arrays, structs, functions (func), and
    pipes. All variable names must follow [a-zA-Z][a-zA-Z0-9]* and can't be a
    reserved word. */
array<int>[5] oneD = make(array<int>[5]); // declare 1d array
array<array<int>[2]>[10] twoD = make(array<array<int>[2]>[10]); // 2d array

/* Function format: func <return type> <function name>(<0 or more arguments>)
    {<function body>}
Function return type can include void or an "any" type which is used by functions
    similarly to Java wildcards or C++ templates. */
func int myMethod() {
   return 5;
}

/* A struct is a list of grouped variables, whose types can be any primitive or
    advanced types. */
struct SalesData {
   string bookNo;
   int unitsSold = 0;
   float revenue = 0.0;
}

/* Our language can do arithmetic operations using the following arithmetic
    operators: +, -, /, *, %. We allow type promotion. The addition operator can
    also concatenate strings. Order of evaluation follows PEMDAS. Operators that
    evaluate to booleans: == , !=, <, >, >=, <=. This does structural comparison
    for primitive types and physical comparison for advanced types. == and !=
    operate on booleans and evaluate to booleans. Logical Operators: !, &&, ||. */
int a = 5; // declare an int variable
float ab = a + 6.5; // ab is 11.5

/* Pipes will allow typed communication between threads in a similar manner to Go's
    channels. */
pipe<string> messages = make(pipe<string>);
```

```
/* The arrow operator pushes data into the pipe. The data needs to be of the same
   type as the pipe. */
messages <- "Hello, other end of the pipe.";
```

```
/* Shoo routines execute in a separate thread, allowing for parallel programming. */

/* Executing an unnamed shoo-routine. Return type is not required for an unnamed
   shoo-routine as the return value will be ignored. Shoo-routines do not have
   regular closures. They will still have access to variables defined in the outer
   scope, but any such referenced variables will be cloned (except for pipes) to
   prevent data-races. */
shoo function () {
}()

/* The shoo keyword may also be used to run a named function in a separate thread.
   The function will have access to variables in the parent scope where it's
   defined. Any additional arguments may be passed in as parameters. */
int x = 5;
function void someFunc(int y) {
   return y + x;
}

/* Regular function call where the x variable referenced inside of someFunc is the
   same x variable in this outer scope. */
someFunc(5);

/* Executing as a shoo-routine will cause the referenced variables to be cloned, so
   this shoo-routine is not able to modify the x in the outer scope. */
shoo someFunc(10);
```

```
/* The for loop works like C's. There is no while loop in Shoo. */
for (int i = 0; i < 10; i = i + 1) {
}

/* C-like if/else blocks. */
if (x > 0) {
   println("x is positive"); // println is a built-in function
} else if (x == 0) {
   println("x is zero");
} else {
   println("x is negative");
}
```

3 Sample Code

A sample program that demos the use of the "any" keyword, higher-order functions, and pipes in Shoo.

```
/* This program parallelizes the task of summing over each array in the 2d array. */
function void sampleProgram1() {
  pipe<int> messages = make(pipe<int>);
  array< array<int>[10]>[2] tasks = [
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
  ];

  function int sum(int x, int y) {
    return x + y;
  }

  /* The 'any' type can be used as an escape-valve to bypass the limitations of the
     type system. */
  function any foldl(func f, any acc, array<any> items) {
    if (length(items) == 0) { //length is a built-in function
      return acc;
    } else {
      return foldl(f, f(acc, first(items)), rest(items));
    }
  }

  for (int i; i < length(tasks); i++) { // length(tasks) = 2
    // a shoo-routine to sum the arrays in parallel
    shoo function(array<int> task) {
      messages <- foldl(sum, 0, task);
    }(tasks[i]);
  }

  int final = 0;

  for (int i; i < length(tasks); i++) {
    int result <- messages;
    final += result;
  }

  println(final);
}

sampleProgram1();
```