Simpler Implementation of Sketches Through Enhanced Expressiveness

Miguel Velez¹ and Armando Solar-Lezama²

1 University of St. Thomas, St. Paul, MN, 2 Massachusetts Institute of Technology, Cambridge, MA

Introduction

The Sketch [6] engine is a tool that allows programmers to write a sketch, a partial program with some missing implementation, which the synthesizer uses to discover the missing code. The programmer uses expressions and place holders to indicate what variables and functions should be used to determine the missing code. When sketches increase in size, it can be very difficult to track and specify all the variables and functions that the engine should use. We implemented some features that made the language more expressive and the engine more powerful.

Local Variables Construct

• Pick local variables within scope and of the specific type

  void swap(ref bit[w] x, ref bit[w] y){
    minrepeat{
      bit[w] = x;
    }
  }

  • Can use regular expression generators [5] to specify what variables should be considered when finding a solution

  generator Vec Gen(bit [d]oml, Mat a, Mat b, Vec w, Vec x, Vec y, Vec z, double d1, double d2){
    Vec tt = {w[x][y][z]};
    if(tt){
      tt = scale(tt, (d1,d2));
      if doml & a?{
        tt = newml(tt || b);
      }
      if (tt){
        tt = vsub([w][x][y][z][tt] || [w][x][y][z][tt]);
      }
      return tt;
    }
  }

  Results and Conclusions

  • Enhanced expressiveness and increased the domain of applications that programmers can develop with this engine
  • Increased productivity and made sketches easier to write and debug
  • Local variables construct provides a cleaner method to instruct the synthesizer what variables it should consider when finding an answer
  • Lambda expressions are a syntactically lightweight method to create a function that can be reused and passed to high-order functions
  • Automatic casting of expressions combines the power of the previous features to avoid implementing specific and complicated functions

Acknowledgements

I would want to thank Professor Armando Solar-Lezama for his valuable advice and guidance, the Computer-Aided Programming group for their support, and the MIT Summer Research Program and the MIT Office of the Dean for Graduate Education for funding this project.

References


Lambda Expressions

Grammar

  lambda_expr : ( vars ) right_expr;
  vars : comma delimited variables
  right_expr : any valid expression in the language

  • Supported in many other languages

  \((x \rightarrow x + 1)\)
  \((\lambda x. x + 2)\)

  Haskell [2]
  Python [3]

  • Syntactically lighter than declaring a function with a name, a return type, and formal parameters

  harness void main() {
    Circle c1 = new Circle (...);
    ...
    assert circ(c1) != circ(c3);
  }

  • Need to know that the function circ takes a Circle struct as a parameter and that the return type is a float

  float circ(Circle c) {
    return 2 * 3.1415 * c.radius;
  }

  • Lambda function performs the same operation, but it is simpler to write

  harness void main() {
    Circle c1 = new Circle (...);
    ...
    fun circ = (c) \rightarrow 2 * 3.1415 * c.radius;
    assert circ(c1) != circ(c3);
  }

  Casting of Expressions

  • Allows to pass expressions in high-order function calls

    int apply(fun f, int x){
      return f(x);
    }

  • Combines the previous features to increase reuse of these functions

    generator int rec(fun choices){
      if ???{
        return choices();
      } else{
        return {rec(choices({}) | - | - + rec(choices())};
      }
    }

  • To pass an arbitrary number of variables to the generator, the user needs to define a generator and use a regular expression

    harness void sketch(int x, int y, int z){
      generator int f(){
        return {x | y | z}:
      }
      assert rec(f) == (x + y) * (y - z);
    }

  • By combining the features of lambdas and local variables construct, we do not need to declare a generator with choices

    harness void sketch(int x, int y, int z){
      assert rec($\{int\}) == (x + y) * (y - z);
    }

Future Work

• Currying, which allows easier creation of anonymous functions
• Built-in functions that take lambdas expression and modify arrays
• Local variable construct automatically infer the type of variable
• Optimize and develop systems that benefit from these features
• Systems include, but are not limited to, an automatic grader for programming assignments [4], a system that automatically transforms fragments of application logic into SQL queries [1].