Joe Emenaker
CSC 530
Spring 2021
Homework #5 (Sketch & Automated Grading)

Deliverable

My recollection from last week's class was that you wanted us to experiment with Sketch (and focus less on the transformations of Python->mPy->Sketch discussed in the Galwani paper)

Sources

I started with Solar-Lozama's "Sketching Approach to Program Synthesis" (https://people.csail.mit.edu/asolar/papers/Solar-Lezama09.pdf) to get an idea of how the tool was meant to be used, but moved to the programmer's manual (https://people.csail.mit.edu/asolar/manual.pdf) which proved to be a great resource for what features Sketch supports.

Interesting Features

Restricting the range of constants and test values

It's no surprise that Sketch provides a few ways to reduce the search space (the "second dimension" of program synthesis from the Galwani paper) for the ?? holes.

- Probably the easiest way to restrict the search space for a hole is with the bit-depth annotation, `??(bits)`. This allows us to limit the trial values for that hole to 0 – (2^{bits}-1). It's important to understand that Sketch only uses positive values for the holes. If you want to access negative values, you need to account for this manually (eg. `??(4)-8` will give you values from -8 to +7).
- Much more powerful than this is to enumerate the values to try with the regex generators:
  ```
  #define RVAL_CHOICES {| -4 | -2 | -1 | 1 | 2 | x | y |}
  ```
  so that we can then refer to this in our program…
  ```
  int t = RVAL_CHOICES
  ```
  and Sketch will try each of the substitutions. This is quite useful when we want to try a range of values which don't number as a power of 2 or where the values aren't contiguous (or are comprised of non-literal expressions, like "x+y")
- Another interesting choice for hole-filling is any local variable of a given type (see section 4.3 of the manual). For example, `$(int)` would be replaced by any local variables or formal parameters of that type in the function. It's not clear, from the documentation, whether this construct looks *into* structures, however.

With as much flexibility as Sketch gives you for the holes, it's a little surprising that I couldn't find such flexibility for the parameters to the functions. For example, with the function myfunc(int x, int y)
the x and y parameters are going to be tested for values from 0-(2^N-1) where N, by default, is 5. This value of N can be changed with the --bnd-inbits flag, but Sketch, by itself, doesn't give us the ability to represent negative values or specific selections of values. We can work around this, however, with array literals:

```
/* Run with --bnd-inbits=2 option */
int[4] x_vals = { -1, 12, 13, 13 }; /* 3 specific values for x */
int[4] y_vals = { 3, 7, 7, 7 }; /* 2 specific values for y */
int my_func(int x, int y) {
   return x_vals[x] * y_vals[y];
}
```

Because all int test values (i.e. all of those passed as parameters to our candidate function) share the same --bnd-inbits value (and because that value is used to determine the number of bits in the range), if any range of values number different from a power of 2 (like x_vals above) or number less than another range we need (like y_vals above), we need to pad the array literal with some duplicate values.

Resetting global variables

Although not really a feature that gives the *programmer* much control, Sketch supports global variables by *resetting* those variables after each call to the test function and the reference implementation to make sure that the function calls are not affected by side-effects.

Experiments

Curve Fitting (with specific input values) – (See: *nomial_solver.sk)

Question: Can Sketch be given the literal results of a function for *specific* input values (in this case, where the zeroes of a function are) and divine an arithmetic function (in this case, the coefficients of a polynomial) which gives those results for those inputs?

To investigate this, I stored the specific in/out values in inputs[] and outputs[] arrays, and used the test parameter (limited to just 0 and 1 with --bnd-inbits=1) to select which input/output pair to test. I started with a simple constant (a '*no*nomial', if you will):

```
/**
 ** Can we solve a constant? A "NOnomial"?
**/

int reference(int idx){
    return 2;
}

int polynomial(int idx) implements reference {
    int a0 = ??(2);
    return a0;
}
```

This works just fine. However, when trying to solve a *monomial* (i.e. $y = a_1x + a_0$), I get the result that this is unsatisfiable (well, first, I was getting "array out of bounds" errors until I expanded the array to be *three* elements… not sure what's going on with that):

```
/**
 ** Can we solve a MONOmial? Where y = a1*x + a0?
**/

/* MUST run sketch with --bnd-inbits=1 or we'll get an array out
of-bounds error */
/* Analytic solution is: y = 1*x + 3 */
int[3] inputs = { -3, -2, -2 };
int[3] outputs = { 0, 1, 1 };

int reference(int idx){
    /* We don't need to get the x values, just return
    the y value for this index */
    return outputs[idx];
}

int polynomial(int idx) implements reference {
    /* Set coefficients to have the range [-4,3] */
    int a1 = ??(3) - 4;
    int a0 = ??(3) - 4;
    int input = inputs[idx];
    int result = a1 * input + a0;
    return result;
}
```

Surprisingly, Sketch gives me these results:

```
UNSATISFIABLE ASSERTION The spec and sketch can not be made to be
equal
```

This doesn't make sense, since our target coefficients are well within the ranges specified for the holes. Furthermore, I even *overrode* the coefficients with the correct answers and I still got the same "unsatisfiable" result. I suspect that there's something strange going on with the arithmetic expression, but that's hard to check, as I couldn't get Sketch to support print statements. Section 2.20 of the manual makes reference to a "@NeedsInclude()" annotation to include headers (like stdio.h, in my case), but I couldn't get Sketch to not give me a parse error.

Sketch *does* offer a few different debugging/verbosity flags, however, and by turning up the verbosity, part of the output looked like this:

```
input idx_7_7_0 has value 19= (2)
 input outputs__ANONYMOUS_s1_a_a_idx_0_0 has value 1= (0)
 input outputs__ANONYMOUS_s1_a_a_idx_1_0 has value 1= (0)
 input outputs__ANONYMOUS_s1_a_a_idx_2_0 has value 1= (0)
 input inputs__ANONYMOUS_s3_9_9_idx_0_0 has value 1= (0)
 input inputs__ANONYMOUS_s3_9_9_idx_1_0 has value 1= (0)
 input inputs__ANONYMOUS_s3_9_9_idx_2_0 has value 1= (0)
 * After optims it became = 2 was 19
  UNSATISFIABLE ASSERTION The spec and sketch can not be made to be equal.
_p_out_reference_ANONYMOUS
  UNSATISFIABLE ASSERTION The spec and sketch can not be made to be equal.
_p_out_reference_ANONYMOUS
```

I'm not quite clear what to make of (what looks to be) these strange assignments to the array values.

Revisiting Loop Invariants (from the last homework) – (See: invariant.sk)

Question: Is there a way to use Sketch to solve our loop-invariant problem from the last homework? In other words, is there a way to get Sketch to find the a_n coefficients of

$$I = a_1x + a_2y + a_3 \geq 0 \vee a_4x + a_5y + a_6 \geq 0$$

To investigate this, I created an "invariant(x, y)" function which had polynomials in an inequality, and tasked Sketch with finding suitable coefficients:

```
/* We want the reference function to always return true */
bit always_true(int x, int y){
    return 1;
}

bit unknown_invariant(int x, int y) implements always_true {
    return (
        invariant(50, y)
        && ( !invariant(x,y) || !(x<0)  || invariant(x+y, y+1))
        && ( !invariant(x,y) || !(x>=0) || y>0 )
    );
}
```

```
bit invariant(int x, int y) {
    int a1 = ??(3) - 4;
    int a2 = ??(3) - 4;
    int a3 = ??(3) - 4;
    int a4 = ??(3) - 4;
    int a5 = ??(3) - 4;
    int a6 = ??(3) - 4;
    return (a1*x + a2*y + a3 >= 0) || (a4*x + a5*y + a6 >= 0);
}
```

Even though the coefficients had bounds, [-4, +3], which should have been suitable for the solution, Sketch responded with "`[1620679323.1860 – ERROR] [SKETCH] Sketch Not Resolved Error:`" This is different from the "cannot be made to be equal" from the polynomial experiment, so it seems there is some kind of reduction/resolution that Sketch is trying to do which I don't fully understand, yet.

Exploring 2nd-Order Logic – (See: second_order.sk)

I did have *one* definite success, and I actually think this is the most powerful.

Sketch allows us to pass functions as parameters (see section 2.13 of the manual). Maybe we can store a variety of functions in an array and allow Sketch to iterate through them. **Question: Can we have Sketch find us combinations of functions which give us the outputs we want?**

Yes, but figuring out how wasn't easy. First, Sketch doesn't allow us to create an array of anything except integers

We can define add, sub, mul, and div functions, and then do something like:

```
fun[4] the_funcs = { add, sub, mul, div }
```

Within a function (Sketch only allows integers to initialize global arrays), but calling the function is a problem. We can't do:

```
the_funcs[??](a,b)  /* parse error */
```

Because Sketch can't parse it. However, it *can* parse:

```
fun chosen_func;  /* global function… assigned later */
int choose_your_func(int a, int b) {
    int idx = ??;
    if (idx == 0) { chosen_func = add; }
    if (idx == 1) { chosen_func = sub; }
    if (idx == 2) { chosen_func = mul; }
    if (idx == 3) { chosen_func = div; }
    return chosen_func(a,b);
}
```

That will parse, but Sketch complains that chosen_func is ambiguous.

At this point, I was just doing a roundabout way of just executing the functions, directly, from the if statements (the reason I'm not using switch statements is because those work differently in Sketch). So, let's ditch chosen_func and just call the functions, directly:

```
int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }
int mul(int a, int b) { return a * b; }
int div(int a, int b) { return a / b; }

int example(int a, int b) {
        return a-b;
}

int whats_the_func(int a, int b) implements example {
        int choice = ??(2);
        int c = 0;
        if (choice == 0) { c = add(a,b); }
        if (choice == 1) { c = sub(a,b); }
        if (choice == 2) { c = mul(a,b); }
        if (choice == 3) { c = div(a,b); }
        return c;
}
```

That correctly figures out to use 1 for the hole (to choose subtraction). Now, can we use Sketch's *repeat* construct with it's any-local-variable-of-this-type construct to have Sketch figure out more-complicated functions?

```
int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }
int mul(int a, int b) { return a * b; }
int div(int a, int b) { return a / b; }

int example(int a, int b) {
        /* How about (a + b) * (a - c) */
        int c = a + b;
        int d = a - b;
        int e = c * d;
        return e;
}

int whats_the_func(int a, int b) implements example {
        int parm1;
        int parm2;
        int result;
        /* The repeat construct treats all holes */
         /* as distinct for each repetition */
        repeat(3) {
                 /* Here's where we decide which function */
                int choice = ??(2);
                / * select the parms, first, so that Sketch */
                 /* doesn't do it for each case */
```

```
                parm1 = $(int);
                parm2 = $(int);
                if (choice == 0) { result = add(parm1,parm2); }
                if (choice == 1) { result = sub(parm1,parm2); }
                if (choice == 2) { result = mul(parm1,parm2); }
                if (choice == 3) { result = div(parm1,parm2); }
                /* Let Sketch decide what to assign it to */
                 /* (maybe reuse a or b?) */
                $(int) = result;
        }
        return $(int);
}
```

When I feed this to Sketch, the output is a little cryptic because it's creating temporary variables as it needs them (and it prints function calls with the *return value assignment as the last parameter*, so "add(a, b, c);" means "c = add(a, b);").

```
void whats_the_func (int a_0, int b_1, ref int _out)  implements
example/*second_order.sk:14*/
{
  int result_s10 = 0;
  mul(b_1, b_1, result_s10); // result_s10 = b * b
  int result_s10_0 = 0;
  mul(a_0, a_0, result_s10_0); // result_10_0 = a * a
  int result_s8 = 0;
  sub(result_s10_0, result_s10, result_s8); //  result_s8 = a*a -
b*b
  _out = result_s8;
  return;
}
```

Initially, this looks wrong, because the example function was never multiplying any parameters with themselves, but, on closer inspection, (a – b)*(a + b) is the same as (a*a – b*b), **so it actually worked!**

Interestingly, it appears that the Sketch team has done something similar with perform-operation-selected-by-index (see generators.skh, line 151, function op()).