Tehran University
College of Engineering
Faculty of ECE

Software Synthesis course

Report of final project

Sketching

Fateme Marzani

Melika Dastranj

July 2020

Contents

Summery

Since the Sketch synthesis lets programmers to write a code and leave some parts of it unspecified, we ourselves tested this type of synthesis and evaluated the strengths and weaknesses of it. So we tried coding with this tool and examine how this tool works for real codes. Finally we reach to these conclusions that they are as follows:

- Sketch like many other synthesizers is incomplete.
- We figure out Sketch is not capable to find an integer among both float and integers answers!
- If the result of a function is float, then function parameters must be float too.
- Sketching might be time consuming!
- Sketch as an inductive synthesis certainly get into correct answer but the answer might be the weakest one.
- It was better if the Sketch had a time threshold, we mean it could close the program after passing that much of time. Or at least it could inform us whether it is feasible or not.
- Sketch has well operation on small integer numbers but its operation on the bigger and negative numbers is a bit little different.
- Sketch support a very limited range of integer 0 to 31. It decreases the accuracy of answers and leads to reaching weak answers.
- Shadowing is supported by some languages like Java, but does not support by Sketch.
- Unrolling is not helpful and it's a very weak method to verifying loops and in the most of the time can't find the solution.
- In many cases, the expressions that one wants to synthesis are not simple integer constant, but complex expressions. The Sketch language provides a simple notation to describe these holes using regular expression syntax.
- It is hard to deal with string types in Sketch.

Preface

Sketching is a new approach to synthesis. The idea of synthesis goes back to many years but for a long time the traditional point of view to synthesis was that synthesis meant to start from a complete formal specification and you have to give that to your computer and the outcome would be correct code implementing exactly that specification. Unfortunately you may face with many issues to this style of synthesis. Not only the fact that it is very challenging to come up with the algorithm completely from scratch but also it is hard to write a complete formal specification.

There is a different way of looking at the synthesis problems. Rather than starting from a complete formal specification that describes everything in the algorithm, you can start from a space of possible programs that you might consider for solving your task. The way you describe such spaces program is to use parameters. Depending on how you pick values for those parameters you get different results. Also you can apply constraints on the space of programs. Actually with every constraint you apply you limit the program. For instance these constraints can be input/output example or test harness. Test harness is a function that whenever it called, the assertion should pass (I mean assertion can't be violated). By assertions you can tell to your program about properties and constraints. Eventually you get a program that matches with the constraints that you have been searching for and your program must work for all the input/output examples. These constraints can help you to tell the program what is your intention exactly and you expect what to see as a result of your selected inputs.

Sketch is a C-like programming language that has been developed by Solar Lezama and his group at MIT. In this programming language you can leave some pieces of code as a hole (??) which called Sketch and a program with this format is called a partial program. Synthesizer then replaces (??) with a constant. By assertions you can tell to your program about properties and constraints. In fact assertions force synthesizer to replace hole with a value that you expect. The efficient way is to write assertions inside the harness. As an example suppose that you want to write a code to swap two words without an extra temporary, but you don't exactly remember all the details of it. You don't remember how many times it takes to shift to the left or to the right. In other words you try to express your intention code in a high-level format and leave low-level parts unspecified. This is the time

that Sketch comes in and lets you to use its capabilities. You can leave the details alone and Sketch makes a suitable decision for you and fill the (??) with the proper options.

Some applications for Sketch is automated tutoring system for people who are learning a programming language and help them with giving them a feedback about errors they are making.

How does it work?

SAT solvers generally can't easily and efficiently solve the Sketch problems (Because of a phenomenon named 'quantify alternation' that exist in Sketch problems). Hence it gets help from an algorithm named CEGIS (Counter Example Guided Inductive Synthesis). In the picture below you will see the way this algorithm works. Every Sketching problem changes to a formula and all constraints are expressed as assertions. The search strategy for constant holes is the CEGIS. It starts with a random number and give those numbers to a checker to checks where or not they are true and satisfy the formula correctly. It means rather than checks to find a solution for all possible inputs, it tries to select a set of numbers with this idea that it will work for all possible inputs. If it didn't work, it tells to a checker for finding a counter example. In each iteration CEGIS tries to find a counter example (if it exists) that violates the formula and constraints. If it successfully could find the example which satisfies the formula, it again gives the counter example to synthesizer. It repeats these steps up to a level which it can't find any other counter example and all examples satisfies the formula and it works for all inputs. At the end the checker successfully gives the result. If the selected inputs didn't work for all possible inputs, CEGIS fails.

CEGIS is not complete.

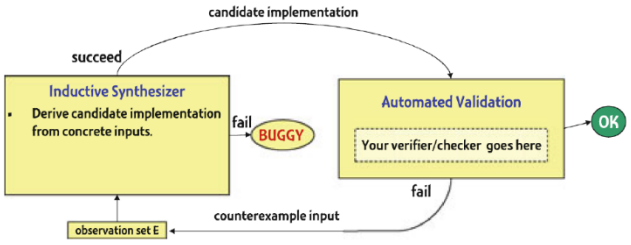

Figure 1 CEGIS procedure

How to install?

There are some preliminaries for installing Sketch that they are as follows:

Bash

- ✓ sudo apt update
- ✓ sudo apt install bash-completion

G++

- ✓ sudo apt install g++

Flex

- ✓ sudo apt-get update –y
- ✓ sudo apt-get install -y flex

Bison

- ✓ sudo apt-get update
- ✓ sudo apt-get install -y bison

Also there is an easy way as well:

Contains pre-built JAR file for the frontend

Source distribution for the backend

Versions available with pre-built backends as well

Build the backend if necessary and Run!

(But we followed the first option ☺)

Set PATH if you want to be able to run from other directories

Set SKETCH_HOME to run compiled code

- ✓ From sketch-frontend run export
  SKETCH_HOME="`pwd`/runtime"

How to run?

First and foremost place your code in a file (something.sk). Then run the Sketch synthesizer with the following command line:

>sketch something.sk

Flags

Before getting into the codes, It is a good idea to review some useful flags which will be explained in details .You can search for more flags in.

How to use flags:

If you want to use flags, write this command above your codes.

pragma options "flag's name";

Also it is possible to use flags by passing them in the command line.

sketch --bnd-inbits SketchFileName.sk

Some useful flags

--bnd-inbits

The bounded space of inputs ranging is zero to $2^{\text{bnd-inbits}}$ which is set 5 as a default number and numbers bigger than 5 is not preferred. Our experiment with " --bnd-inbits 6 " failed.

-fe-output-code

It is a good feature of Sketch helping to generate a C++ automatically from the .sk file. Without it, the synthesizer simply outputs the code to the console.

-fe-output-test

This flag causes the synthesizer to generate a test harness to run the C++ code with a set of random inputs.

-bnd-arr-size

The maximum size of arrays whit dynamic size can be defined with this flag. It means the system considers arrays size up to -bnd-arr-size. The default value of this flag is 5.

-bnd-unroll-amnt

This flag determines how many times a loop or a repeat can be unrolled. The default value of this flag is 8.

--debug-cex

This flag is used to debug the Sketch code by showing you the counter examples that it generates as a procedure to find a solution to your program.

Primitive Types

Only five primitive types such as bit, int, char, double and float are defined by Sketch.

The subtyping relation between three of them is: **bit**v**char**v**int**, so a bit variable can be used instead of a character or an integer. Also it is possible to recharge float and double but there is not any subtyping relationship between them.

For representing Boolean type Sketch uses bit type means bit 0 determines false and bit 1 determines true.

Structs

Sketch gives us a chance to define a new construct by using defined types.

```
struct name{
   type1 field1;
   ...
   typek fieldk;
}
```

Arrays

Array can be defined in Sketch like

```
Type[size] name;
```

The size of array can be fixed or dynamic.

String

Unfortunately Sketch does not support string as a specific type. It is defined as an array of characters. As same as Java or C, string appears in double quotes.

We find difficult to work with string in Sketch.

Padding

The padding can be thought of as implicit type cast from small array to bigger array to illustrate more

Let see an example:

Pad(type)=value;

Pad(int)=0;

Consider we defined an array of size 5 and just 3 parameters were inserted Int[5] a={45,87,63}. Padding can cast it to an array by assigning zero to another undefined parameters. After assignment the array a is equal to {45,87,63,0,0}.

Padding can be used for nested arrays.

Let's Sketch

We have tried to implement some sketching examples to find out how it works and is it practicable in real projects or not? Let's review some of them.

Armstrong Number

An Armstrong number of three digits is an integer, where the sum of the cubes of its digits is equal to the number itself. It is an Armstrong code written in sketch. We put ?? instead of 10.

```
include "sketchlib/math.skh";
generator boolean Armstrong(int n) {
   int a; int arm=0;
   int temp=n;
   int hol=??;
    while(n!=0)
    {
        a=n%hol;
        arm=arm+(a*a*a);
```

```
        n=n/hol;
    }
    if(arm==temp){return true;}
    else{return false;}
}
harness void main() {
    assert Armstrong(153) == true;
    assert Armstrong(20) == false;
    assert Armstrong(370) == true;
    assert Armstrong(371) == true;
    assert Armstrong(372) == false;
    assert Armstrong(1) == true;
    }
```

Sketch was not able to solve this problem. We added more assertion but it still rejected.

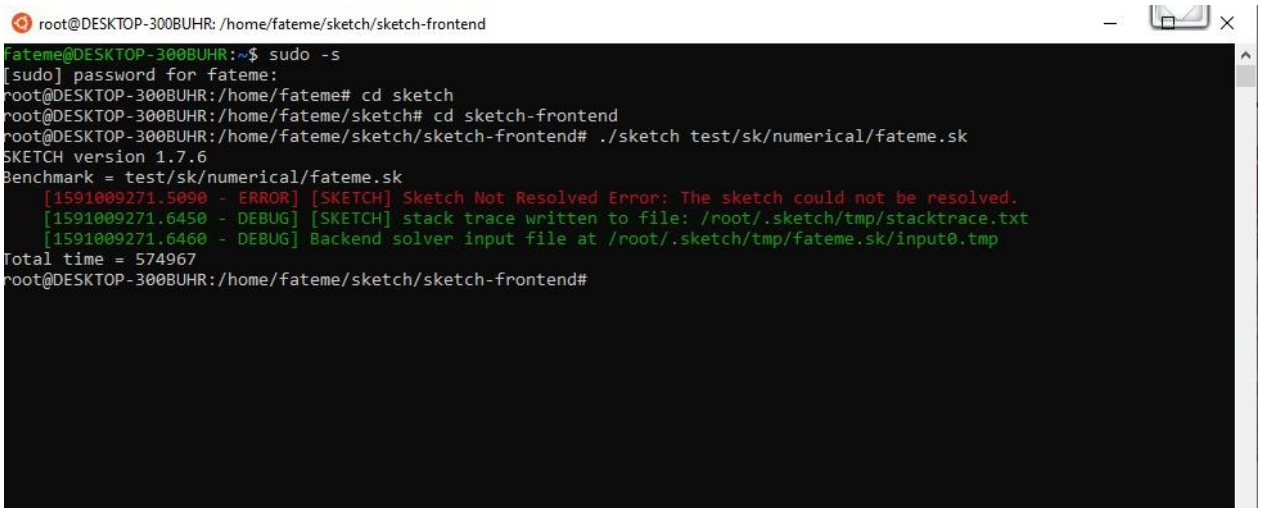

Figure 2 Armstrong Number's result

- ✓ **Sketch as same as many synthesizer is incomplete.**

Cubic

We implemented cubic equation and expected to get its answers(5,-2,1).

```
int Cubic(int a,int b,int c,int d){
    int x=??;
    return a*x*x*x+b*x*x+c*x+d;

}
```

```
harness void main() {
      assert Cubic(1,-4,-7,10) == 0;
}
```

According to figure 3 Sketch successfully solved it and returned number 5 as one of the answers.

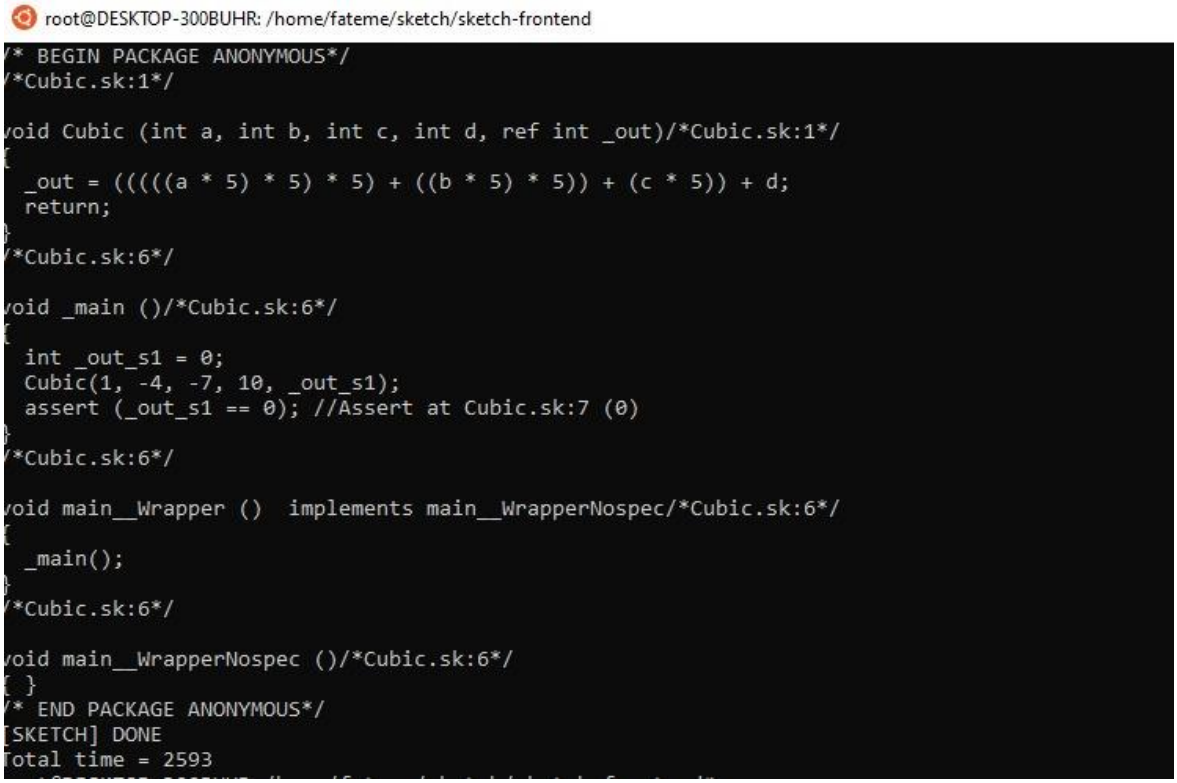

Figure 3 Cubic's result

Float cubic quadratic

We test cubic function again with parameters which leads to (x1 = 0 & x2= -2.70711 & x3 = -1.29289) answers and expected Sketch to find the answer which is integer or is it able to cast float to integer automatically? But test faced to error and failed.

```
int FloatQuadratic(int a,int b,int c){
    int x=??;
    return a*x*x+b*x+c;
}
harness void main() {
      assert FloatQuadratic(2,8,7) == 0;
}
```

- ✓ **We figure out Sketch is not capable to find an integer among both float and integers answers!**

Finally we encounter with an example [miniTestb04.sk] wich solving the float typs problem by using a flag["--be:usesnopt --be:numericalsolver]. We implement our code with this flag again and faced to another weaknesses.

```
include "sketchlib/math.skh";
pragma options "--be:usesnopt --be:numericalsolver";
generator float FloatQuadratic(int a,int b,int c){
    float x=??;
    return a*x*x+b*x+c;
}
harness void main() {
      assert FloatQuadratic(2,8,7) == 0.0;
}
```

After running the code. It failed by incompatibility of integer and floats. It means:

- ✓ **If the result of a function is float, then function parameters must be float too.**

So we refined the code as below and after passing a very long time we did not reach to answer and closed the running program.

```
include "sketchlib/math.skh";
pragma options "--be:usesnopt --be:numericalsolver";
generator float FloatQuadratic(float a,float b,float c){
    float x=??;
    return a*x*x+b*x+c;
}
harness void main() {
      assert FloatQuadratic(2.0,8.0,7.0) == 0.0;
}
```

- ✓ **Sketching might be time consuming!**

Leap year

We implemented the leap year program and write (??) Instead of number 100 to find out Sketch can reach to the correct answer by inductive synthesis or not?

```
boolean Leapyear(int y){
    int x=??;
if(y!= 0)
    {
       if(y%400==0){
           return true;}
        else if(y%x ==0){
            return false;}
        else if(y%4==0){
            return true;}
        else {
            return false;}
                              }
    else{
        return false;}
}
harness void main() {
    assert Leapyear(2000) == true;
    assert Leapyear(1712) == true;
    assert Leapyear(1912) == true;
    assert Leapyear(1876) == true;
    assert Leapyear(2007) == false;
```

```
    assert Leapyear(1998) == false;
    assert Leapyear(1900) == false;
```

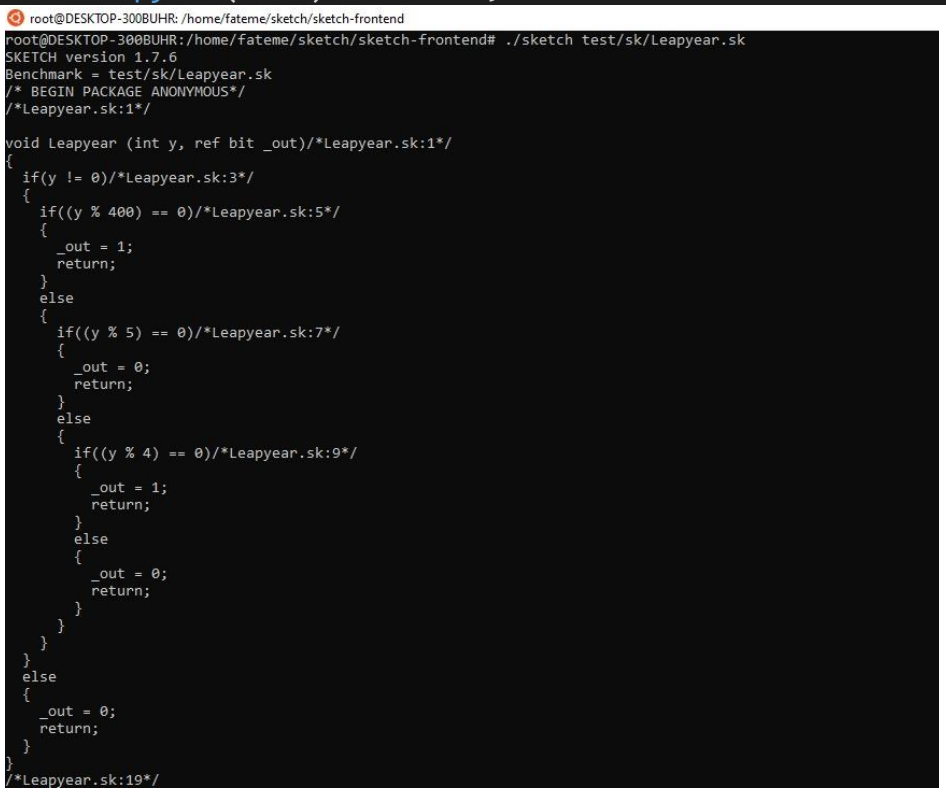

Figure 4 Leap year 's run

According to figure 4 faced to number 5 as a result which is one of divisor of 100. It was not a wrong answer but also not specific as we had expected.

We increased the number of assertion but still got 5 as a result so more specification has been needed so we wrote Leap year version 2 which dividing the hole to its divisors and we faced to 19 as the result but it still was not our expectation.

We add (assert (z!=1 && w!=1);) to get a better result and faced to 10 as x result.

It got better but still needed more specification.

```
boolean Leapyear2(int y){
    int w=??;
    int z=??;
    int x=w*z;
if(y!= 0)
    {
       if(y%400==0){
           return true;}
        else if(y%x ==0){
```

```
            return false;}
        else if(y%4==0){
            return true;}
        else {
            return false;}
                                }
    else{
        return false;}

}
harness void main() {
    assert Leapyear2(2000) == true;
    assert Leapyear2(300) == false;
    assert Leapyear2(1912) == true;
    assert Leapyear2(1876) == true;
    assert Leapyear2(2007) == false;
    assert Leapyear2(1998) == false;
    assert Leapyear2(1900) == false;
    }
```

Finally, we implemented version 3 of leap year code witch dividing the hole to its prime divisors and get to strongest answer for the hole.

```
boolean Leapyear3(int y){
    int w=??;
    int z=??;
    int x=w*w*z*z;

if(y!= 0)
    {
       if(y%400==0){
           return true;}
        else if(y%x ==0){
            return false;}
        else if(y%4==0){
            return true;}
        else {
            return false;}
                              }
```

```
    else{
        return false;}

}
harness void main() {
    assert Leapyear3(2000) == true;
    assert Leapyear3(1712) == true;
    assert Leapyear3(1912) == true;
    assert Leapyear3(1876) == true;
    assert Leapyear3(2007) == false;
    assert Leapyear3(1998) == false;
    assert Leapyear3(1900) == false;
    }
```

```
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend# ./sketch test/sk/mine/LeapYear3.sk
SKETCH version 1.7.6
Benchmark = test/sk/mine/LeapYear3.sk
/* BEGIN PACKAGE ANONYMOUS*/
/*LeapYear3.sk:1*/

void Leapyear3 (int y, ref bit _out)/*LeapYear3.sk:1*/
{
  if(y != 0)/*LeapYear3.sk:6*/
  {
    if((y % 400) == 0)/*LeapYear3.sk:8*/
    {
      _out = 1;
      return;
    }
    else
    {
      if((y % 100) == 0)/*LeapYear3.sk:10*/
      {
        _out = 0;
        return;
      }
      else
      {
        if((y % 4) == 0)/*LeapYear3.sk:12*/
        {
          _out = 1;
          return;
        }
        else
        {
          _out = 0;
          return;
        }
      }
    }
  }
  else
  {
    _out = 0;
    return;
  }
}
```

Figure 5 LeapYearV3's run

- ✓ **Sketch as inductive synthesis certainly get into correct answer but the answer might be the weakest one.**

Finding invariant

We wanted to test the Sketch to find an invariant of a division function with verification condition code(as we tried the example of the lecture 15 that Sketch had calculated the invariant correctly and successfully). But here in the following example it ran for a very long time and didn't give any result at all.

✓ **It's better if Sketch had a time threshold, could close the program after passing that much of time.**

```
include "sketchlib/generators.skh";
bit inv(int x,int y,int res,int i,int xx,int yy ){
return exprBool ({x,xx,res,i,y,yy} ,{TIMES,PLUS});
}
harness void main(int x,int y,int res,int i,int xx,int yy,int tmp_res, int tmp_i ){
if(x==xx && y==yy && res==0 && i==xx){
    assert inv(x,y,res,i,xx,yy );
    if( inv(x,y,tmp_res,tmp_i,xx,yy) && tmp_i > yy){
        assert inv(x,y,tmp_res + 1,tmp_i - yy,xx,yy);
  }
    if( inv(x,y,tmp_res,tmp_i,xx,yy) && tmp_i<=yy){
        assert(i<y);
    }
}
}
```

Figure 6 Division VC code

IsSquare

IsSquare is a function which evaluates whether an integer number is square or not. A number is a square if it is a sum of successive odd numbers (25 = 1+3+5+7+9).

```
sk > seq > isSquare.sk
generator boolean IsSquare(int n) {
    int i=1;
    int sum=0;
    int dis=??;
    while(sum<n){
        sum=sum+i;
        i=i+dis;
    }
   if(sum==n)return true;
   else return false;
    }

harness void main() {
    assert IsSquare(25) == true;
    assert IsSquare(49) == true;
    }
```

Figure 7 IsSquare function code

We wanted to synthesize the small integers in holes, so it worked successfully and returned the result correctly.

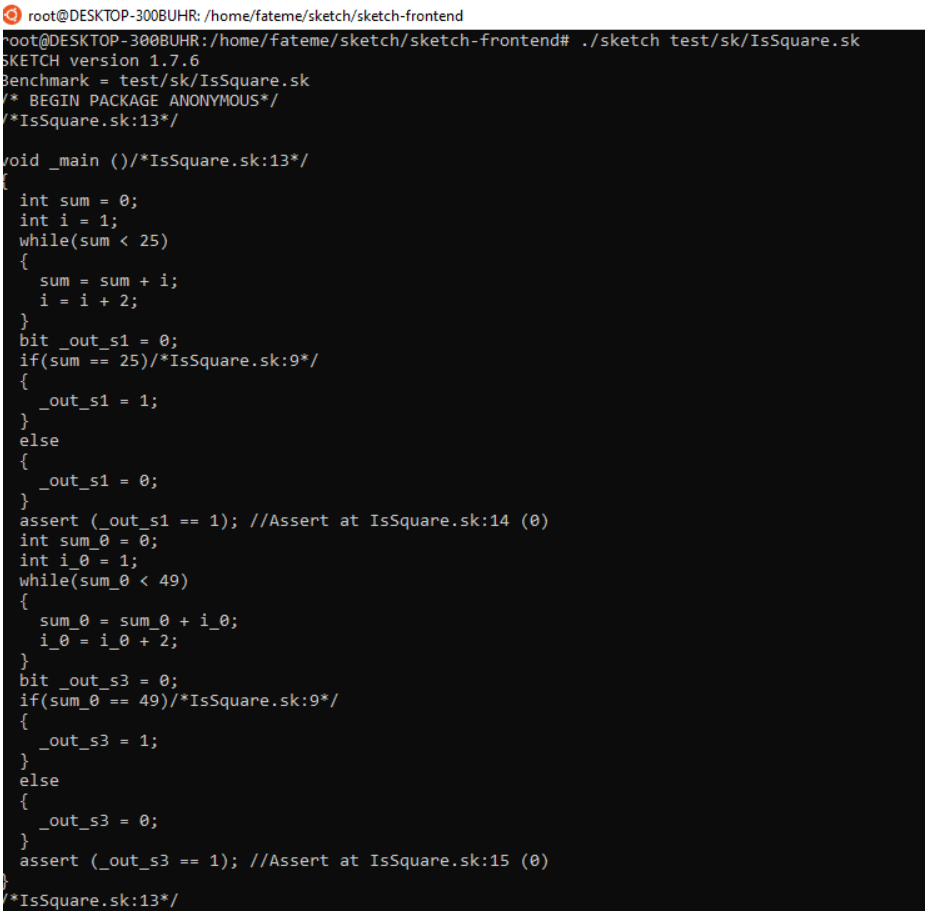

Figure 8 IsSquare run

IsTriangular

IsTriangular is a function which evaluates whether an integer number is triangular or not. An integer number is triangular if it is a sum of the successive numbers (15 = 1+2+3+4+5).

isTriangular.sk

```
generator boolean IsTriangular(int n) {
    int i=1;
    int sum=0;
    int dis=??;
    while(sum<n){
        sum=sum+i;
        i=i+dis;
    }
   if(sum==n)return true;
   else return false;
    }

harness void main() {
    assert IsTriangular(10) == true;
    assert IsTriangular(6) == true;
    }

```

Figure 9 IsTriangular function code

root@DESKTOP-300BUHR: /home/fateme/sketch/sketch-frontend

```
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend# ./sketch test/sk/isTriangular.sk
SKETCH version 1.7.6
Benchmark = test/sk/isTriangular.sk
/* BEGIN PACKAGE ANONYMOUS*/
/*isTriangular.sk:13*/

void _main ()/*isTriangular.sk:13*/
{
  int sum = 0;
  int i = 1;
  while(sum < 10)
  {
    sum = sum + i;
    i = i + 1;
  }
  bit _out_s1 = 0;
  if(sum == 10)/*isTriangular.sk:9*/
  {
    _out_s1 = 1;
  }
  else
  {
    _out_s1 = 0;
  }
  assert (_out_s1 == 1); //Assert at isTriangular.sk:14 (0)
  int sum_0 = 0;
  int i_0 = 1;
  while(sum_0 < 6)
  {
    sum_0 = sum_0 + i_0;
    i_0 = i_0 + 1;
  }
  bit _out_s3 = 0;
  if(sum_0 == 6)/*isTriangular.sk:9*/
  {
    _out_s3 = 1;
  }
  else
  {
    _out_s3 = 0;
  }
  assert (_out_s3 == 1); //Assert at isTriangular.sk:15 (0)
}
/*isTriangular.sk:13*/
```

Figure 10 IsTriangular run

For this program it works and returns the correct result.

- ✓ **We figure out, Sketch has well operation on small integer numbers but how does it work on bigger ones or negative?**

As what is written in Sketch's manuals," In practice, the solver only searches a bounded space of input starting from zero to 2^bnd-inbits−1.The default fort this flag is 5; attempting number much bigger than this is not recommended." So we decided to set flag –b-inbits 6

to figure out how it works. Although it is written flags bigger than 5 in not recommended it actually means: It is impossible to set the flag to bigger than 5.

Bound

Despite setting the --bnd-inbits flag to 6, in the following example with giving the number of 32 as a holl and we got an error.

```
pragma options "--bnd-inbits 6";
int Bound(int n){
    int t;
    int y=??;
    t= y* n;
    return t;
}
harness void main() {
    assert Bound(1) == 32;
    assert Bound(2) == 64;
    }
```

Figure 11 Bound function

```
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend# ./sketch test/sk/mine/Bound1.sk
SKETCH version 1.7.6
Benchmark = test/sk/mine/Bound1.sk
  UNSATISFIABLE ASSERTION Assert at Bound1.sk:9 (0)

*** Rejected
    [1596093053.6630 - ERROR] [SKETCH] Sketch Not Resolved Error:   UNSATISFIABLE ASSERTION Assert at Bound1.sk:9 (0)

*** Rejected
The sketch could not be resolved.
    [1596093053.6760 - DEBUG] [SKETCH] stack trace written to file: /root/.sketch/tmp/stacktrace.txt
    [1596093053.6780 - DEBUG] Backend solver input file at /root/.sketch/tmp/Bound1.sk/input0.tmp
Total time = 1560
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend#
```

Figure 12 Bound function run

Because Sketch only searches a bounded space of inputs ranging 0 to $2^5 - 1$.

So that's why we again tried this program with the correct ranging bound and it successfully get the result.

```
pragma options "--fe-output-code --fe-output-test ";
int Bound(int n){
    int t;
    int y=??;
    t= y* n;
    return t;
}
harness void main() {
    assert Bound(1) == 31;
    assert Bound(2) == 62;
    }
```

Figure 13 correct ranging Bound function

```
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend# ./sketch test/sk/mine/Bound2.sk
SKETCH version 1.7.6
Benchmark = test/sk/mine/Bound2.sk
    [1595917308.7510 - NOTE] Wrote test harness to ./Bound2_test.cpp
[SKETCH] DONE
Total time = 905
```

Figure 14 Correct Ranging Bound's run

Negative

For showing the bound range is an integer number (0 to $2^5 - 1$), we write this program to exactly understand that this number starts from 0 and doesn't support negative numbers.

C: > Users > melika > Desktop > sketch > negative > negative.sk

```
1
2   int negative(int n){
3       int t;
4       int y=??;
5       t= y* n;
6       return t;
7   }
8   harness void main() {
9       assert negative(1) == -1;
10      assert negative(2) == -2;
11      }
12
```

Figure 15 Negative Bound function

```
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend# ./sketch test/sk/mine/Bound2.sk
SKETCH version 1.7.6
Benchmark = test/sk/mine/Bound2.sk
Assert at Bound2.sk:10 (0)

*** Rejected
    [1596015521.4000 - ERROR] [SKETCH] Sketch Not Resolved Error: Assert at Bound2.sk:10 (0)

*** Rejected
The sketch could not be resolved.
    [1596015521.4160 - DEBUG] [SKETCH] stack trace written to file: /root/.sketch/tmp/stacktrace.txt
    [1596015521.4161 - DEBUG] Backend solver input file at /root/.sketch/tmp/Bound2.sk/input0.tmp
Total time = 1579
```

Figure 16 Bound negative's run

- ✓ **In conclusion, Sketch support a very limited rang of integer 0 to 31. It decreases the accuracy of answers and leads to reaching weak answers.**

Shadowing

Shadowing occurs when you use two variables with the same name within scopes that overlap. When it happens, the variable in the higher scope is hidden and shadowed.

We wanted Sketch to try to resolve the shadowing, but unfortunately it was not successful. You can see the result in figure 17.

```
int a;
generator int Shadowing(int b){
    a=5;
    int a=??;

    return (a*b);
}
harness void main() {
      assert Shadowing(1)==1;
      assert Shadowing(6)==6;
      assert Shadowing(100)==100;
}
```

Figure 17 Shadowing Function

```
SKETCH version 1.7.6
Benchmark = test/sk/mine/Shadowing.sk
    [1596193025.9610 - ERROR] [SKETCH] Error at node: Shadowing of variables is not allowe
d (at Shadowing.sk:5)
    [1596193025.9800 - ERROR] [SKETCH] Error at node: Shadowing of variables is not allowe
d (a previously declared in Shadowing.sk:2). (at Shadowing.sk:5)
    [1596193025.9830 - DEBUG] [SKETCH] stack trace written to file: /root/.sketch/tmp/stac
ktrace.txt
Total time = 336
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend#
```

Figure 18 Shadowing's run

- ✓ **Shadowing I s supported by some languages like Java, but does not support by Sketch.**

Swap-Inv

In lecture 14 of the course we saw an example that for verification of While loops. We tried to unroll the loop. We test this program with the default number of 8 for the flag of unrolling (--bnd-unroll-amnt 8). One more time we set this flag to 20, but again didn't solve (--bnd-unroll-amnt 20).

C: > Users > melika > Desktop > sketch > swap-inv > sawap.sk

```
1  pragma options"--bnd-unroll-amnt 20";
2  harness void swap( int xin , int yin ){
3  int t , x = xin , y = yin ;
4  if ( x > y ) {
5      t=x-y;
6      while ( t > 0) {
7          x = x - ??;
8          y = y + ??;
9          t = t - ??;
10         }
11     assert ( x == yin && y== xin );
12
13     }
14
15 }
```

Figure 19 Swap_unrolling function

```
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend# ./sketch test/sk/mine/swap.sk
SKETCH version 1.7.6
Benchmark = test/sk/mine/swap.sk
  UNSATISFIABLE ASSERTION swap.sk:6: This loop was unrolled 20 times, but apparently that was not
enough. Use the --bnd-unroll-amnt flag for better results.

*** Rejected
    [1596094482.0630 - ERROR] [SKETCH] Sketch Not Resolved Error:   UNSATISFIABLE ASSERTION swap.s
k:6: This loop was unrolled 20 times, but apparently that was not enough. Use the --bnd-unroll-amn
t flag for better results.

*** Rejected
The sketch could not be resolved.
    [1596094482.0810 - DEBUG] [SKETCH] stack trace written to file: /root/.sketch/tmp/stacktrace.t
xt
    [1596094482.0811 - DEBUG] Backend solver input file at /root/.sketch/tmp/swap.sk/input0.tmp
```

Figure 20 unroling with flag 20's run

- ✓ **Unrolling is not helpful and it's a very weak method to verifying loops and most of the time can't find the solution.**
- ✓ **How to deal? Use invariants.**

For writing invariants, we use verification conditions.

C: > Users > melika > Desktop > sketch > swap-inv > swap-inv.sk

```
include "sketchlib/generators.skh";
bit inv(int y,int yy,int x,int xx,int t ){
return exprBool ({y,yy,x,xx,t} ,{PLUS});
}
harness void main(int y,int yy, int x, int xx,int t,int tp,int tx,int ty ){
  if(y==yy && x==xx && x>y){
    assert inv(y , yy , x , xx , t );
    if( inv(ty,yy,tx,xx,tp) && tp>0){
        assert inv(ty-1,yy,tx-1,xx,tp-1);
    }
    if( inv(ty,yy,tx,xx,tp) && tp<=0){
        assert (ty==xx && tx==yy );
    }
  }
}
```

Figure 21 Swap_Inv function

```
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend# ./sketch test/sk/mine/swap-inv.sk
SKETCH version 1.7.6
Benchmark = test/sk/mine/swap-inv.sk


*** Rejected
    [1595784372.5410 - ERROR] [SKETCH] Sketch Not Resolved Error:

*** Rejected
The sketch could not be resolved.
    [1595784372.5720 - DEBUG] [SKETCH] stack trace written to file: /root/.sketch/tmp/stacktrace.txt
    [1595784372.5721 - DEBUG] Backend solver input file at /root/.sketch/tmp/â□□â□□bndâ□□inlineâ□□amnt/input0.tmp
Total time = 435472
```

Figure 22 swap-inv's run

As you see in the figure 22 Sketch couldn't solve this!

Check1ToZeroSwitch

Sketch support regular expression and we implemented Check1ToZeroSwitch which change the bit of 0 to 1 as an output, when the input change 1 to 0 and asked Sketch to find its implementation and successfully it did!

```
int W = 32;
bit[W] Check1ToZeroSwitch (bit[W] x) {      // W: word size
    bit isOneFound=0;
    bit[W] result = 0;
    for (int i = 0; i < W; i++)
        if (x[i] == 1) {
            isOneFound = 1;
```

```
        } else if (isOneFound) {
            if(x[i] == 0) {
                isOneFound = 0;
                result[i] = 1;
            }
        }
    return 0;
}
bit[W] Check1ToZeroSwitchsk (bit[W] x)  implements Check1ToZeroSwi
tch {
    bit[W] tmp= {| (x (+|&) x) & (!)?(x)|};
    return tmp;
}
```

```
void Check1ToZeroSwitchsk (bit[32] x, ref bit[32] _out)  implements Check1ToZeroSwitch
{
  _out = (x & x) & (!(x));
  return;
}
/* END PACKAGE ANONYMOUS*/
[SKETCH] DONE
Total time = 846
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend#
```

Figure 23 Check1ToZeroSwitch run

- ✓ **In many cases, the expressions that one wants to synthesize are not simple integer constants, but complex expressions. The sketch language provides a simple notation to describe these holes using regular expression syntax.**

StingStruct

As mentioned before String is not supported by Sketch, so tried to make an struct to deal with string as below.

```
struct String{
    int Length;
    char[Length] contant;
    }
harness void main() {
    String c =new String(Length=5,contant ="test");
    assert c.Length==5;
```

```
}
```

```
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend# ./sketch test/sk/mine/test.sk
SKETCH version 1.7.6
Benchmark = test/sk/mine/test.sk
/* BEGIN PACKAGE ANONYMOUS*/
struct String {
    int Length;
    char[Length] contant;
}
/*test.sk:5*/

void _main ()/*test.sk:5*/
{
  String@ANONYMOUS c;
  c = new String(Length=5, contant={'t','e','s','t','\0'});
  assert ((c.Length) == 5); //Assert at test.sk:8 (2)
}
/*test.sk:5*/

void main__Wrapper ()  implements main__WrapperNospec/*test.sk:5*/
{
  _main();
}
/*test.sk:5*/

void main__WrapperNospec ()/*test.sk:5*/
{ }
/* END PACKAGE ANONYMOUS*/
[SKETCH] DONE
Total time = 605
root@DESKTOP-300BUHR:/home/fateme/sketch/sketch-frontend#
```

Figure 24 StingStruct run

- ✓ **It is hard to deal with the string type in Sketch.**

Usage

JSketch

JSketch compiles a Java program with unspecified parts (a program with holes (??) in) in to the Sketch language and then maps the result of Sketch synthesis back to Java. The translation to Sketch is challenging because Sketch does not support object oriented.

We tried to install all requirements of Jsketch to use this tool, but unfortunately the available version of Jsketch was not updated and there was some inconsistencies among dependencies, that's why sadly we were not able to use this tool. ☹

References

[1] A. Solar-Lezama, "The Sketching Approach to Program Synthesis," Programming Languages and Systems. APLAS 2009, 2009.

[2] A. Solar-Lezama, "Program Sketching," International Journal on Software Tools for Technology Transfer, 2012.

[3] Solar Lezama (2012) The Sketching Approach to Program Synthesis, Available at : http://people.csail.mit.edu/asolar/sketch2012/

[4] J. Jeon, X. Qiu, J. S. Foster and A. Solar-Lezama, "JSKETCH: Sketching for Java," 2015.