Abstract: Tool Artifact for "Program Sketching using Lifted Analysis for Numerical Program Families"

Aleksandar S. Dimovski[1], Sven Apel[2], and Axel Legay[3]

[1] Mother Teresa University, 12 Udarna Brigada 2a, 1000 Skopje, N. Macedonia
[2] Saarland University, Campus E1.1, 66123 Saarbrücken, Germany
[3] Université catholique de Louvain, 1348 Ottignies-Louvain-la-Neuve, Belgium

This document summarises the evaluation results reported by the paper "Program Sketching using Lifted Analysis for Numerical Program Families". We compare three tools:

- FamilySketcher [1] that uses the decision tree-based lifted analysis and the polyhedra domain from the APRON library [2].
- Program sketching tool `Sketch` version 1.7.6 that uses SAT-based inductive synthesis [3]
- The `Brute-Force` enumeration approach that analyzes all variants, one by one, using a single-program analysis based on the polyhedra domain from the APRON library [2].

1 Performance results

All experiments are executed on a 64-bit Intel®Core^{TM} i7-8700 CPU@3.20GHz $\times$ 12, Ubuntu 18.04.5 LTS, with 8 GB memory, and we use a timeout value of 200 sec. All times are reported as average over five independent executions. For FamilySketcher and `Brute-Force` enumeration approach, we report times measured via Sys.time function of OCaml needed only for the actual static analysis task to be performed. The implementation, benchmarks, and all obtained results are also available from: `https://github.com/aleksdimovski/Family_sketcher`.

Table 1 shows the results of synthesizing our benchmarks.

2 FamilySketcher

FamilySketcher is a research prototype program sketcher designed for resolving numerical sketches using lifted static analysis based on abstract interpretation.

2.1 Installation

The tool requires the following applications and libraries:

Table 1. Performance results of FamilySketcher vs. `Sketch` vs. `Brute-Force`. All times in sec.

Bench.	5 bits			8 bits			16 bits		
	Family Sketcher	`Sketch`	`Brute Force`	Family Sketcher	`Sketch`	`Brute Force`	Family Sketcher	`Sketch`	`Brute Force`
Loop	0.007	0.215	0.628	0.007	0.218	67.79	0.007	33.74	timeout
Loop'	0.007	0.205	0.627	0.007	0.206	60.59	0.007	2.292	timeout
LinExp	0.165	0.222	0.479	26.99	0.238	36.80	timeout	timeout	timeout
Conditional	0.002	0.210	0.019	0.002	0.210	0.155	0.004	3.856	54.68
LoopCond	0.011	0.225	0.065	0.013	0.262	0.404	0.013	timeout	191.43
LoopCond'	0.022	0.221	1.615	0.022	0.267	199.95	0.023	timeout	timeout
NestedLoop	0.053	timeout	4.186	0.054	timeout	timeout	0.054	timeout	timeout

* OCaml
```
(sudo) apt-get install ocaml-interp
```
* Findlib
```
(sudo) apt-get install ocaml-findlib
```
* Menhir: LR(1) parser generator
```
(sudo) apt-get install menhir
```
* Opam: https://opam.ocaml.org/doc/Install.html
```
(sudo) apt-get install opam
```
* Initialize OPAM state
```
opam init
eval $(opam env)
```
* OUnit
```
opam install ounit
```
* APRON: numerical abstract domain library
```
opam install depext
opam depext apron
opam install apron
```
* Set the Library Path variable in /.bashrc (or /.profile) that can be opened with "gedit /.bashrc". First find the folder where the libraries of apron are installed, e.g. "/home/.opam/system/share/apron/lib" You can navigate to the folder where libraries can be found and check using command "pwd" for the absolute path of that folder. Then, set the Library Path by appending:
```
LD_LIBRARY_PATH=/home/.opam/system/share/apron/lib
export LD_LIBRARY_PATH
```
Log out of the current session, then log in and check:
```
echo $LD_LIBRARY_PATH
```
* Zarith: arbitrary-precision integer operations
```
opam install zarith
```

2.2 Compiling FamilySketcher

Once all required libraries are installed, 'ocamlbuild' can be used to build program sketcher with the following two commands:

```
   cd family_sketcher
   eval $(opam config env)
   ocamlbuild Main.native -use-ocamlfind -use-menhir -pkgs 'apron,gmp,oUnit,zarith'
 -I utils -I domains -I frontend -I main -libs boxMPQ,octD,polkaMPQ,str,zarith
```

2.3 Usage

The program sketcher performs a forward reachability analysis of program families and resolves the holes (features) so that the final assertions are valid.

The following general command-line options are recognized (showing their default value):

-tree set to perform decision tree-based lifted analysis
-single set to perform brute force enumeration approach using single analysis for each variant
-main main set the analyzer entry point (defaults to main)
-domain boxes|octagons|polyhedra set the abstract domain (defaults to boxes)
-joinfwd 2 set the widening delay in forward analysis

2.4 Examples

All benchmarks from the paper are in "tests" folder. Enter the folder that contains the tool, and write the following commands. Note that after commands in square brackets we write the reported times in seconds on our machine.

1. HelloWorld.
`$ ./Main.native -tree -domain polyhedra tests/helloworld.c`
2. Loop and Loop'.
Loop example with 5, 8, and 16 bits sizes of holes is given as loop1-5.c, loop1-8.c, and loop1-16.c
`$ ./Main.native -tree -domain polyhedra tests/loop1-5.c` [0.007]
`$ ./Main.native -tree -domain polyhedra tests/loop1-8.c` [0.007]
`$ ./Main.native -tree -domain polyhedra tests/loop1-16.c` [0.007]
Similarly, Loop' example with 5, 8, and 16 bits sizes of holes is given as loop2-5.c, loop2-8.c, and loop2-16.c
`$ ./Main.native -tree -domain polyhedra tests/loop2-5.c` [0.007]
`$ ./Main.native -tree -domain polyhedra tests/loop2-8.c` [0.007]
`$ ./Main.native -tree -domain polyhedra tests/loop2-16.c` [0.007]
3. LinExp.
LinExp example with 5 and 8 bits sizes of holes is given as linexp-5.c, and linexp-8.c (linexp-16 does not terminate)
`$ ./Main.native -tree -domain polyhedra tests/linexp-5.c` [0.165]
`$ ./Main.native -tree -domain polyhedra tests/linexp-8.c` [26.99]

```
$ ./Main.native -tree -domain polyhedra tests/linexp-16.c [FAILS]
```
4. Conditional.

Conditional example with 5, 8, and 16 bits sizes of holes is given as conditional-5.c, conditional-8.c, and conditional-16.c
```
$ ./Main.native -tree -domain polyhedra tests/conditional-5.c [0.0025]
$ ./Main.native -tree -domain polyhedra tests/conditional-8.c [0.0027]
$ ./Main.native -tree -domain polyhedra tests/conditional-16.c [0.004]
```
5. LoopCond and LoopCond'.

LoopCond example with 5, 8, and 16 bits sizes of holes is given as loopcond1-5.c, loopcond1-8.c, and loopcond1-16.c
```
$ ./Main.native -tree -domain polyhedra tests/loopcond1-5.c [0.011]
$ ./Main.native -tree -domain polyhedra tests/loopcond1-8.c [0.013]
$ ./Main.native -tree -domain polyhedra tests/loopcond1-16.c [0.013]
```
Similarly, LoopCond' example with 5, 8, and 16 bits sizes of holes is given as loopcond2-5.c, loopcond2-8.c, and loopcond2-16.c
```
$ ./Main.native -tree -domain polyhedra tests/loopcond2-5.c [0.022]
$ ./Main.native -tree -domain polyhedra tests/loopcond2-8.c [0.022]
$ ./Main.native -tree -domain polyhedra tests/loopcond2-16.c [0.022]
```
6. NestedLoop.

NestedLoop example with 5, 8, and 16 bits sizes of holes is given as nestedloop-5.c, nestedloop-8.c, and nestedloop-16.c
```
$ ./Main.native -tree -domain polyhedra tests/nestedloop-5.c [0.053]
$ ./Main.native -tree -domain polyhedra tests/nestedloop-8.c [0.054]
$ ./Main.native -tree -domain polyhedra tests/nestedloop-16.c [0.054]
```

3 Brute-Force

We stay in the same folder as for FamilySketcher, we run the same example files, but now we use command-line option "-single" instead of "-tree".

3.1 Examples

All benchmarks from the paper are in "tests" folder. Enter the folder that contains the tool, and write the following commands. Note that after commands in square brackets we write the reported times in seconds on our machine.

1. HelloWorld.
```
$ ./Main.native -single -domain polyhedra tests/helloworld.c
```
2. Loop and Loop'.

Loop example with 5, 8, and 16 bits sizes of holes is given as loop1-5.c, loop1-8.c, and loop1-16.c
```
$ ./Main.native -single -domain polyhedra tests/loop1-5.c [0.628]
$ ./Main.native -single -domain polyhedra tests/loop1-8.c [67.79]
$ ./Main.native -single -domain polyhedra tests/loop1-16.c [FAILS]
```
Similarly, Loop' example with 5, 8, and 16 bits sizes of holes is given as loop2-5.c, loop2-8.c, and loop2-16.c

```
$ ./Main.native -single -domain polyhedra tests/loop2-5.c [0.627]
$ ./Main.native -single -domain polyhedra tests/loop2-8.c [60.59]
$ ./Main.native -single -domain polyhedra tests/loop2-16.c [FAILS]
```

3. LinExp.

LinExp example with 5 and 8 bits sizes of holes is given as linexp-5.c, and linexp-8.c (linexp-16 does not terminate)

```
$ ./Main.native -single -domain polyhedra tests/linexp-5.c [0.479]
$ ./Main.native -single -domain polyhedra tests/linexp-8.c [36.80]
$ ./Main.native -single -domain polyhedra tests/linexp-16.c [FAILS]
```

4. Conditional.

Conditional example with 5, 8, and 16 bits sizes of holes is given as conditional-5.c, conditional-8.c, and conditional-16.c

```
$ ./Main.native -single -domain polyhedra tests/conditional-5.c [0.019]
$ ./Main.native -single -domain polyhedra tests/conditional-8.c [0.155]
$ ./Main.native -single -domain polyhedra tests/conditional-16.c [54.68]
```

5. LoopCond and LoopCond'.

LoopCond example with 5, 8, and 16 bits sizes of holes is given as loopcond1-5.c, loopcond1-8.c, and loopcond1-16.c

```
$ ./Main.native -single -domain polyhedra tests/loopcond1-5.c [0.065]
$ ./Main.native -single -domain polyhedra tests/loopcond1-8.c [0.404]
$ ./Main.native -single -domain polyhedra tests/loopcond1-16.c [191.43]
```

Similarly, LoopCond' example with 5, 8, and 16 bits sizes of holes is given as loopcond2-5.c, loopcond2-8.c, and loopcond2-16.c

```
$ ./Main.native -single -domain polyhedra tests/loopcond2-5.c [1.615]
$ ./Main.native -single -domain polyhedra tests/loopcond2-8.c [199.95]
$ ./Main.native -single -domain polyhedra tests/loopcond2-16.c [FAILS]
```

6. NestedLoop.

NestedLoop example with 5, 8, and 16 bits sizes of holes is given as nestedloop-5.c, nestedloop-8.c, and nestedloop-16.c

```
$ ./Main.native -single -domain polyhedra tests/nestedloop-5.c [4.186]
$ ./Main.native -single -domain polyhedra tests/nestedloop-8.c [FAILS]
$ ./Main.native -single -domain polyhedra tests/nestedloop-16.c [FAILS]
```

4 Sketch ver. 1.7.6

Download and unzip the tool from https://people.csail.mit.edu/asolar/

The unzipped tool from https://people.csail.mit.edu/asolar/ is in "`sketch-1.7.6`" folder. Enter the folder, and follow the instructions either in README.txt of sketch-1.7.6 folder or below.

4.1 Installation

The following tools need to be installed: bash, g++, bison, flex.

```
(sudo) apt-get install bison
(sudo) apt-get install flex
```

Under the sketch-1.7.6 directory, execute:

```
cd sketch-backend
chmod +x ./configure
./configure
make
cd ..
```

Testing the sketch

```
cd sketch-frontend
chmod +x ./sketch
./sketch test/sk/seq/miniTest1.sk
```

4.2 Examples

All benchmarks from the paper are in "tests" folder. Enter the folder "sketch-frontend", and write the following commands. Note that after commands in square brackets we write the reported times in seconds on our machine. Note that we use the following ./sketch options:

–**bnd-cbits** determines the size in bits of control holes [default is 5]
–**bnd-inbits** determines the size in bits of inputs [default is 5]
–**bnd-unroll-amnt** determines the unroll amount for loops [default is 8]

1. HelloWorld.

```
$ ./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 8 tests/helloworld.sk
```

2. Loop and Loop'.
Loop example is given as loop1.sk

```
$ ./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 8 tests/loop1.sk
```

[0.215]

```
$./sketch --bnd-cbits 8 --bnd-inbits 8 --bnd-unroll-amnt 8 tests/loop1.sk
```

[0.218]

```
$./sketch --bnd-cbits 16 --bnd-inbits 16 --bnd-unroll-amnt 8 tests/loop1.sk
```

[33.74]
Similarly, Loop' example is given as loop2.sk

```
$ ./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 8 tests/loop2.sk
```

[FAILS]

```
$ ./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 9 tests/loop2.sk
```

[0.205]

```
$./sketch --bnd-cbits 8 --bnd-inbits 8 --bnd-unroll-amnt 9 tests/loop2.sk
```

[0.206]

```
$./sketch --bnd-cbits 16 --bnd-inbits 16 --bnd-unroll-amnt 9 tests/loop2.sk
```

[2.292]
3. LinExp.
LinExp example is given as linexp.sk

```
$./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 8 tests/linexp.sk
```

[0.222]

```
$./sketch --bnd-cbits 8 --bnd-inbits 8 --bnd-unroll-amnt 8 tests/linexp.sk
```

[0.238]

```
$./sketch --bnd-cbits 16 --bnd-inbits 16 --bnd-unroll-amnt 8 tests/linexp.sk
```
[FAILS]
4. Conditional.
Conditional example is given as conditional.sk
```
$./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 8 tests/conditional.sk
```
[0.210]
```
$./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 8 tests/conditional.sk
```
[0.210]
```
$./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 8 tests/conditional.sk
```
[3.856]
5. LoopCond and LoopCond'.
LoopCond example is given as loopcond1.sk
```
$./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 32 tests/loopcond1.sk
```
[0.225]
```
$./sketch --bnd-cbits 8 --bnd-inbits 8 --bnd-unroll-amnt 256 tests/loopcond1.sk
```
[0.262]
```
$./sketch --bnd-cbits 16 --bnd-inbits 16 --bnd-unroll-amnt 256 tests/loopcond1.sk
```
[FAILS]
Similarly, LoopCond' example is given as loopcond2.sk
```
$./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 32 tests/loopcond2.sk
```
[0.221]
```
$./sketch --bnd-cbits 8 --bnd-inbits 8 --bnd-unroll-amnt 256 tests/loopcond2.sk
```
[0.267]
```
$./sketch --bnd-cbits 16 --bnd-inbits 16 --bnd-unroll-amnt 256 tests/loopcond1.sk
```
[FAILS]
6. NestedLoop.
NestedLoop example is given as nestedloop.sk
```
$./sketch --bnd-cbits 5 --bnd-inbits 5 --bnd-unroll-amnt 32 tests/nestedloop.sk
```
[FAILS]
```
$./sketch --bnd-cbits 8 --bnd-inbits 8 --bnd-unroll-amnt 32 tests/nestedloop.sk
```
[FAILS]
```
$./sketch --bnd-cbits 16 --bnd-inbits 16 --bnd-unroll-amnt 32 tests/nestedloop.sk
```
[FAILS]

References

1. Aleksandar S. Dimovski, Sven Apel, and Axel Legay. Program sketching using lifted analysis for numerical program families. In *Submitted to VMCAI 2021*.
2. Bertrand Jeannet and Antoine Miné. Apron: A library of numerical abstract domains for static analysis. In *Computer Aided Verification, 21st Inter. Conference, CAV 2009. Proceedings*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009.
3. Armando Solar-Lezama. Program sketching. *STTT*, 15(5-6):475–495, 2013.