Towards Automated Differential Program Verification For Approximate Computing

Student: Shaobo He, Advisor: Zvonimir Rakamarić
{shaobo,zvonimir@cs.utah.edu}
School of Computing, University of Utah

SOFTWARE ANALYSIS RESEARCH LABORATORY

Introduction

- Approximate computing is an emerging area for trading off the accuracy of an application for improved performance, lower energy costs, and tolerance to unreliable hardware
- There is a lack of techniques for rigorous analysis of approximation acceptability criteria such as **safety**, **termination**, and **quality of results**
- Our main contribution is to leverage SymDiff[1], a **semantic** diff tool based on SMT, to **rigorously** and **automatically** verify acceptability criteria of approximate programs

Motivating Example: Swish++

```
function RelaxedEq(x:int, y:int) returns (bool) {
  (x <= 10 && x == y) || (x > 10 && y >= 10)
}
procedure swish(max_r:int, N:int)
    returns (num_r:int) {
  old_max_r := max_r; havoc max_r;
  assume RelaxedEq(old_max_r, max_r);
  num_r := 0;
  while (num_r < max_r && num_r < N)
    num_r := num_r + 1;
  return;
}
```

Generates search results[2]. The underlined statements denote the approximation that non-deterministically changes the threshold to a possibly smaller number, without suppressing the top few (10 in this case) results.

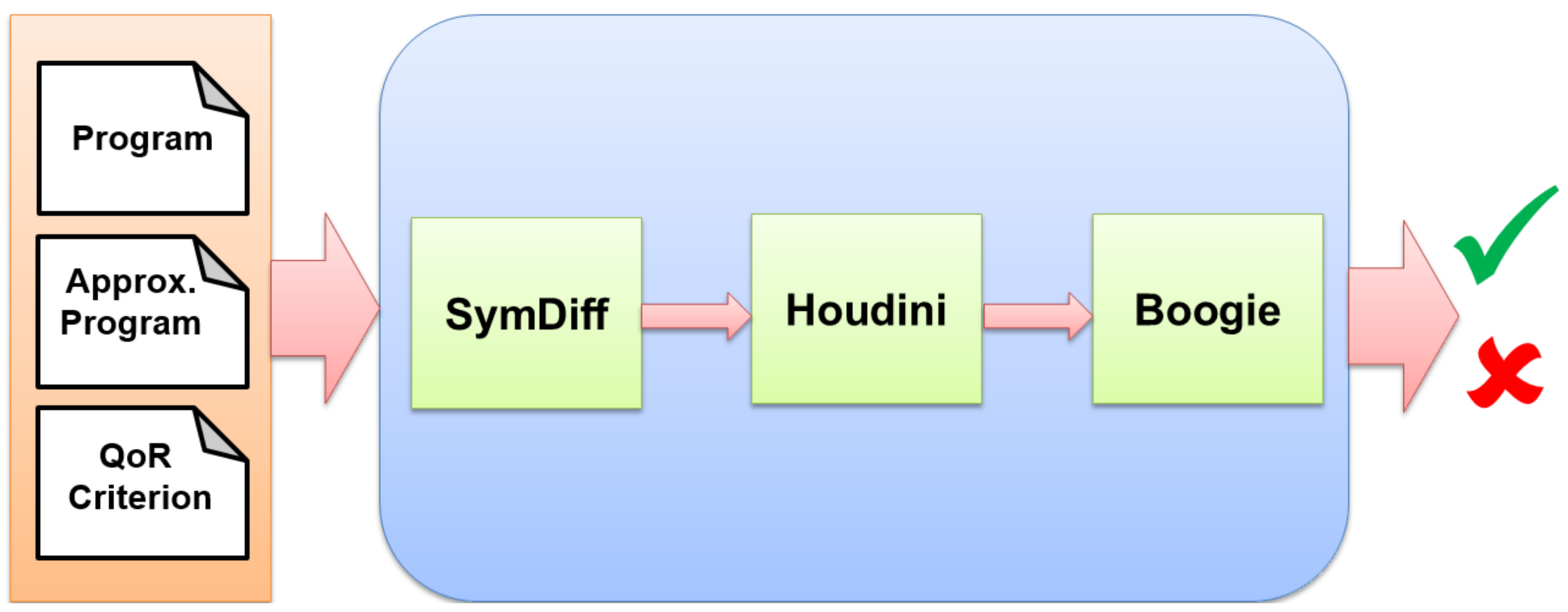
Checking QoR[3]

- Quality of results (QoR) is encoded into *mutual summaries*, a relational specification over the inputs and outputs of the original and approximate procedures
- The verification of mutual summaries over two procedures is converted into a verification problem over a single product procedure
- Arbitrary boolean combination over manually specified predicate templates is automatically computed to improve automation

```
procedure MS_v1.swish_v2.swish(v1.max_r: int, v1.N: int,
                               v2.max_r: int, v2.N: int)
returns (v1.num_r: int, v2.num_r: int);
requires abshoudini(v1.in_max_r <= 10, v2.in_max_r >= 10
v1.in_N <= v2.in_N, v2.in_N <= v1.in_N, ...)
ensures v1.max_r == v2.max_r && v1.N == v2.N
       ==> RelaxedEq(v1.num_r, v2.num_r)
{
  //inline v1.swish
  //inline v2.swish
  call MS_v1.swish_loop_v2.swish_loop(...)
}
```

Signature and skeleton of the product program for Swish++ example. Underlined ensure clause defines the mutual summary and wavy-underlined requires clause invokes full predicate abstraction over simple atomic predicates.

Tool Flow



- SymDiff takes as input two program versions and user-provided acceptability criteria
- It generates a product program from the two versions
- Invariants are inferred using the Houdini algorithm
- Boogie[4] verifier checks the correctness of the product program using Z3 theorem prover

Experimental Results

Benchmark	#Preds	#Manual	#Min-disj	Time(s)
Swish++	14	4	1	5.7
LU Decomposition	32	4	0	6.7
Water	27	0	0	6.7
ReplaceChar	10	1	0	7.2
Selection Sort	66	4	6	306.7
Bubble Sort	38	4	3	48.8
Array Operations	41	1	0	6.7

#Preds and #Manual is the number of atomic predicates automatically generated and manually provided respectively; #Min-disj is the minimum number of disjunctions required in invariants.

Future Work

- Connect our framework to an approximate compiler[5]
- Improve scalability on large programs
- Prove relative termination

References

[1] Shuvendu K. Lahiri, Chris Hawblitzel, Ming Kawaguchi, and Henrique Rebêlo. SymDiff: A language-agnostic semantic diff tool for imperative programs. In *International Conference on Computer Aided Verification (CAV)*, pages 712–717, 2012.

[2] Michael Carbin, Deokhwan Kim, Sasa Misailovic, and Martin C. Rinard. Proving acceptability properties of relaxed nondeterministic approximate programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 169–180, 2012.

[3] Chris Hawblitzel, Ming Kawaguchi, Shuvendu K. Lahiri, and Henrique Rebelo. Towards modularly comparing programs using automated theorem provers. In *International Conference on Automated Deduction (CADE '13)*. Springer, June 2013.

[4] K Rustan M Leino. This is Boogie 2. *Manuscript KRML*, 178:131, 2008.

[5] Accept: An approximate compiler. `http://accept.rocks`.