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

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 acceptance criteria of approximate programs

Motivating Example: Swish++

```plaintext
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

```plaintext
returns (num_r: int, v2.num_r: int);
requires absboudini(v1.in_max_r <= 10, v2.in_max_r >= 10)
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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Preds</th>
<th>#Manual</th>
<th>#Min-dis</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swish++</td>
<td>14</td>
<td>4</td>
<td>0</td>
<td>6.7</td>
</tr>
<tr>
<td>LU Decomposition</td>
<td>32</td>
<td>4</td>
<td>0</td>
<td>6.7</td>
</tr>
<tr>
<td>Water</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>7.2</td>
</tr>
<tr>
<td>ReplaceChar</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>306.7</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>66</td>
<td>4</td>
<td>3</td>
<td>48.8</td>
</tr>
<tr>
<td>Bubble Sort</td>
<td>38</td>
<td>4</td>
<td>3</td>
<td>48.8</td>
</tr>
<tr>
<td>Array Operations</td>
<td>41</td>
<td>1</td>
<td>0</td>
<td>6.7</td>
</tr>
</tbody>
</table>

#Preds and #Manual is the number of atomic predicates automatically generated and manually provided respectively; #Min-dis 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


Experimental Results

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