SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

SOFTWARE ANALYSIS RESEARCH LABORATORY

TOWARDS AUTOMATED DIFFERENTIAL PROGRAM VERIFICATION FOR APPROXIMATE COMPUTING

Shuvendu Lahiri, Microsoft Research
Zvonimir Rakamarić, University of Utah

Collaborators: Ganesh Gopalakrishnan, Arvind Haran, Shaobo He

INTRODUCTION

- **Goal:** Enable rigorous exploration of approximate computing trade-offs
- **Approach:** Develop formal and automated techniques for reasoning about approximations

- Current techniques often lack in
  - rigor (e.g., dynamic analysis),
  - precision (e.g., type systems), or
  - automation (e.g., interactive theorem provers)

PROPOSED APPROACH

- Apply automated *differential program verification* for reasoning about approximations
  - Compare original and approximate program
  - Encode relaxed specifications as differential assertions
- Achieve precision and automation using SMT-based checking and invariant inference
- Ongoing work – under submission

EXAMPLE

- Taken from
  Carbin, Kim, Misailovic, Rinard, "Proving Acceptability Properties of Relaxed Nondeterministic Approximate Programs", PLDI 2012
- Inspired by an open-source search engine

```
procedure swish(maxR:int,N:int) returns (numR:int)
{
  numR := 0;
  while (numR < maxR && numR < N)
    numR := numR + 1;
  return;
}
```

EXAMPLE: APPROXIMATION

```
procedure swish(maxR:int,N:int) returns (numR:int) {
  old_maxR := maxR;
  havoc maxR;
  assume RelaxedEq(old_maxR, maxR);
  numR := 0;
  while (numR < maxR && numR < N)
    numR := numR + 1;
  return;
}

function RelaxedEq(x:int,y:int) returns (bool) {
  (x <= 10 && x == y) || (x > 10 && y >= 10)
}
```

EXAMPLE: VERIFICATION

- Relaxed specification (acceptability property)
  - Relates original and approximate versions of swish (prefixed with `v1.` and `v2.` respectively)

```
v1.maxR=v2.maxR && v1.N=v2.N ⇒
                         RelaxedEq(v1.numR,v2.numR)
```

- Verification effort
  - Carbin et al.
    - Coq proof comprised of 330 lines of proof script
  - Zvonimir et al.
    - Manually provided 4 simple predicates

DIFFERENTIAL VERIFICATION

- *Mutual summary* relates pre- and post-states of two procedure versions

  $$\textbf{old}(v1.g = v2.g) \Rightarrow v1.g < v2.g$$

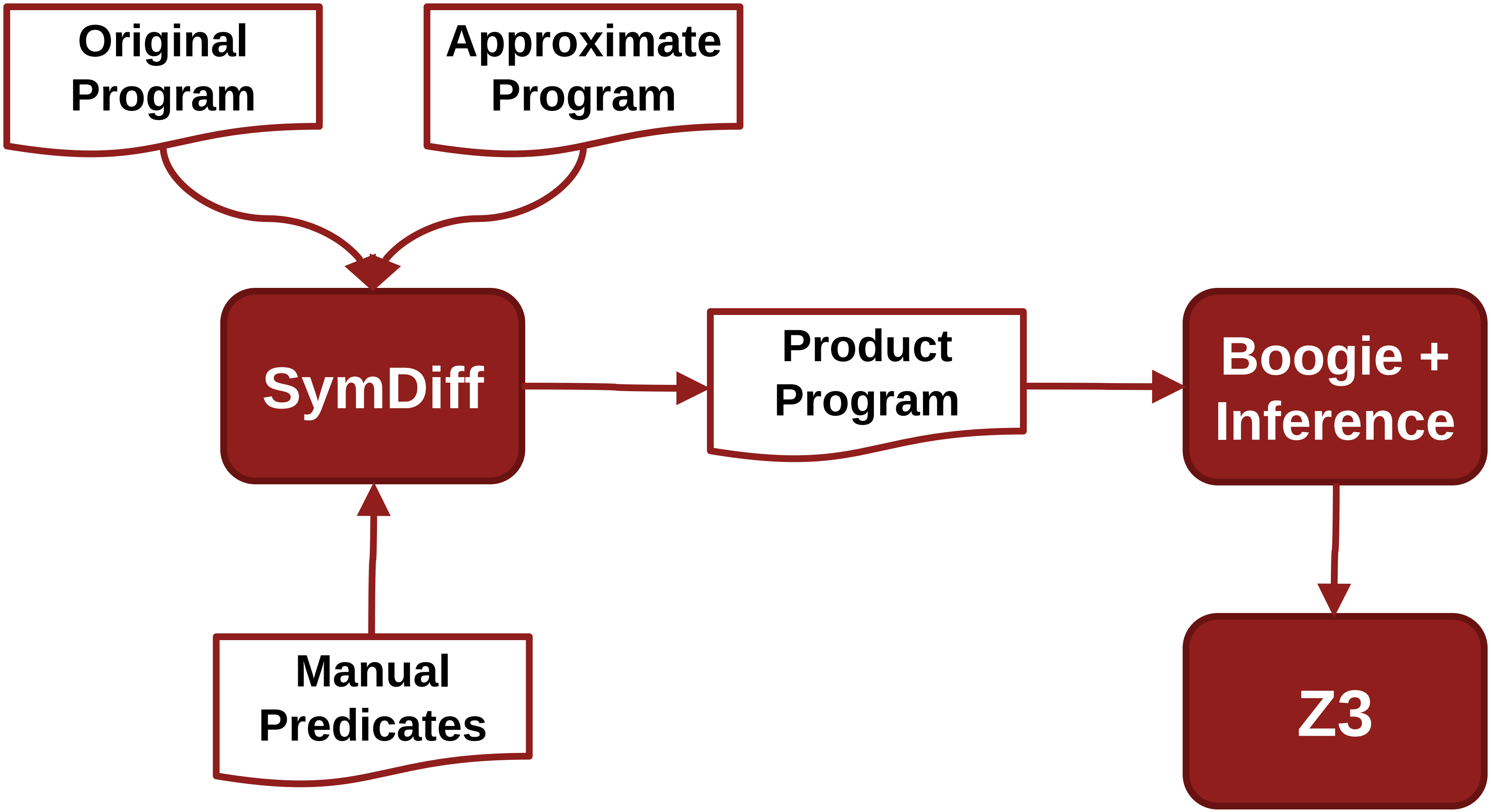
- Mutual summaries are checked modularly by constructing a *product program*
  - Implemented in SymDiff [Lahiri et al. CAV'12]
  - Handles procedure calls [Lahiri et al. FSE'13]
  - Use off-the-shelf program verifier and inference
- Allows for automatic inference of specifications
  - Leverages Houdini inference technique
  - Based on simple candidate templates

IMPLEMENTATION

- SymDiff differential program verifier
  - Implements product program generation
  - Boogie performs verification condition generation
  - Z3 solves generated verification conditions

- Extended automated inference of invariants
  - Users can specify additional predicates
  - Arbitrary Boolean combination over predicates
    - Previously just conjunction

TOOL FLOW


Original Program
Approximate Program
SymDiff
Product Program
Boogie + Inference
Manual Predicates
Z3

EVALUATION

- Acceptability of approximate programs
  - Taken from Carbin et al.
  - Swish++, LU Decomposition, Water

- Control flow equivalence
  - ReplaceChar, Selection Sort, Bubble Sort, Array Operations
  - Introduced encoding that tracks a sequence of visited basic blocks using uninterpreted functions
  - Precisely capturing array fragments

EXPERIMENTS

Benchmark	#Predicates	#Manual Preds.	Time(s)
Swish++	14	4	6
LU Decomposition	32	4	7
Water	27	0	7
ReplaceChar	10	1	7
Selection Sort	66	4	307
Bubble Sort	38	4	49
Array Operations	41	1	7

FUTURE WORK

- Automate predicate generation further
  - Interpolants
  - Indexed predicate abstraction
- Improve scalability
- Prove relative termination
- Reason about probabilities
- Synthesis
- Connect our tool flow with an approximate compiler

CONTRIBUTIONS

1. Applied automated differential program verification (SymDiff) for reasoning about approximations
2. Showed that mutual summaries naturally express many relaxed specifications for approximations
3. Improved precision and automation using SMT-based checking and invariant inference
4. Proved feasibility of applying automated verification to the domain of approximate computing