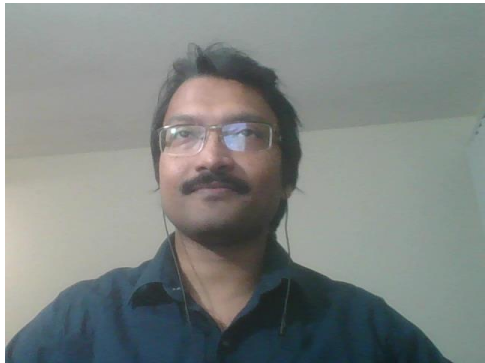
Scalable Validation of Binary Lifters

Sandeep Dasgupta ● Sushant Dinesh ● Deepan Venkatesh ● Vikram S. Adve ● Christopher W. Fletcher
University of Illinois at Urbana Champaign

19 June 2020 @ PLDI'20

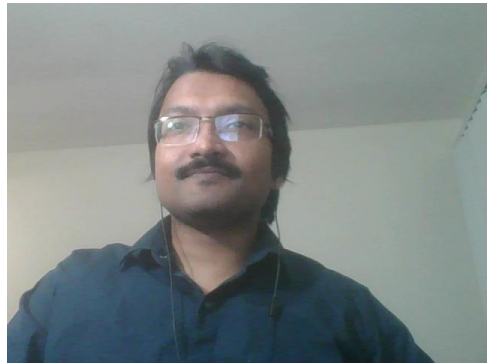


Binary Analysis is Important

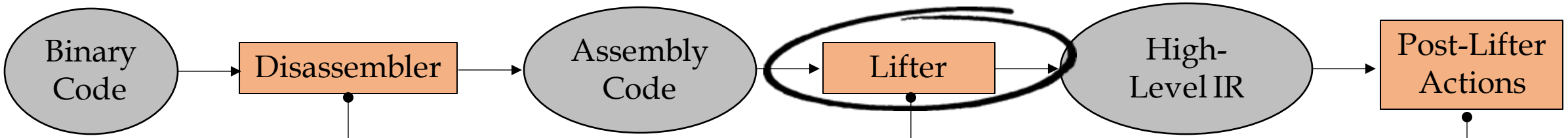
The ability to directly reason about binary is important

A few scenarios where binary analysis is useful

- ❑ Missing source code (e.g. legacy or malware)
- ❑ Avoids trusting compilers



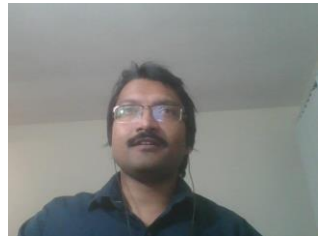
A General Approach for Binary Analysis



Recover high-level primitives
- Instructions
- Control-flow graphs
- Functions boundaries

Enable re-targetability to multiple ISAs.

- Recovery of types, variables, and function prototypes
- Re-target or re-opt for different ISAs



Lifting is Challenging

Manual encoding the effects of binary instructions is hard

- ❑ Vast number of instructions
- ❑ Standard manuals are often ambiguous, buggy, include divergence in the behaviours of variants

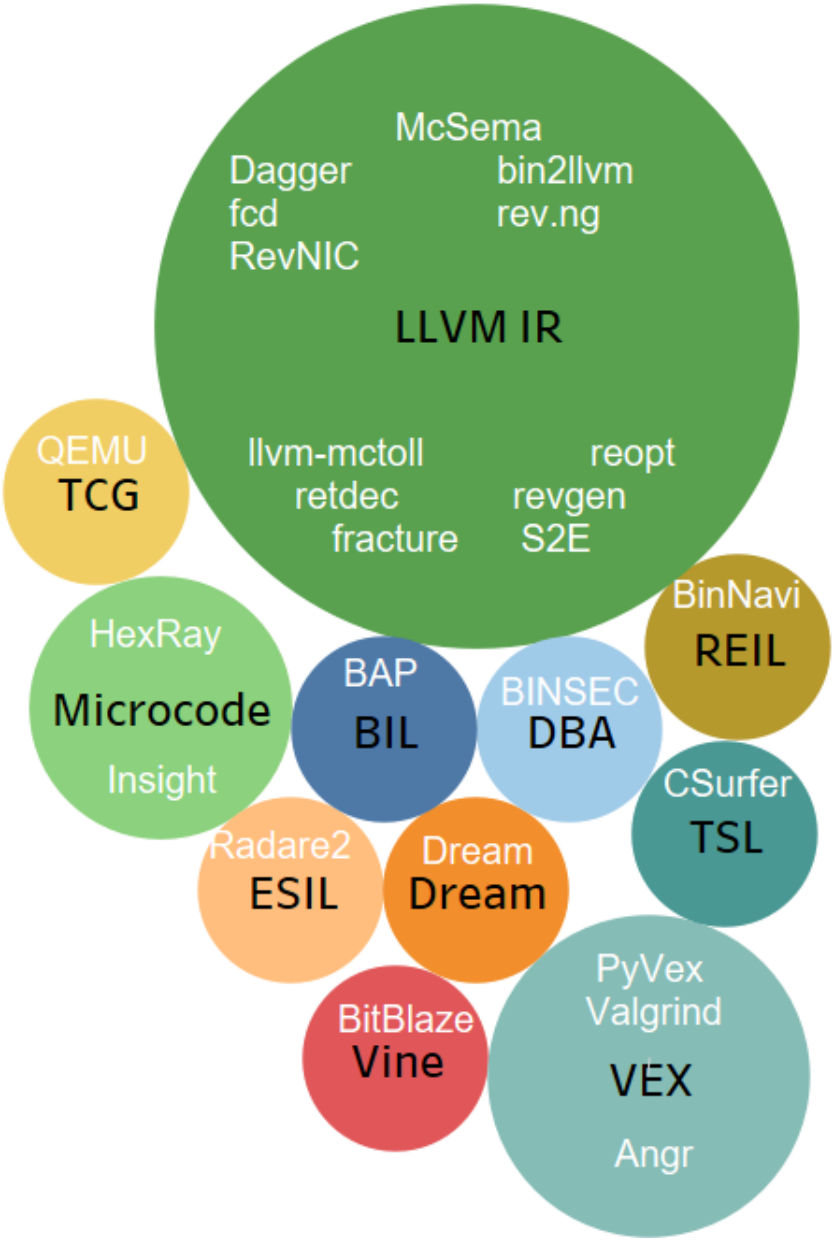
Semantics of Register Variant
(**movsd %xmm1, %xmm0**)

S1. XMM0[63:0] ← XMM1[63:0]
S2. XMM0[127:64] ← (Unmodified)

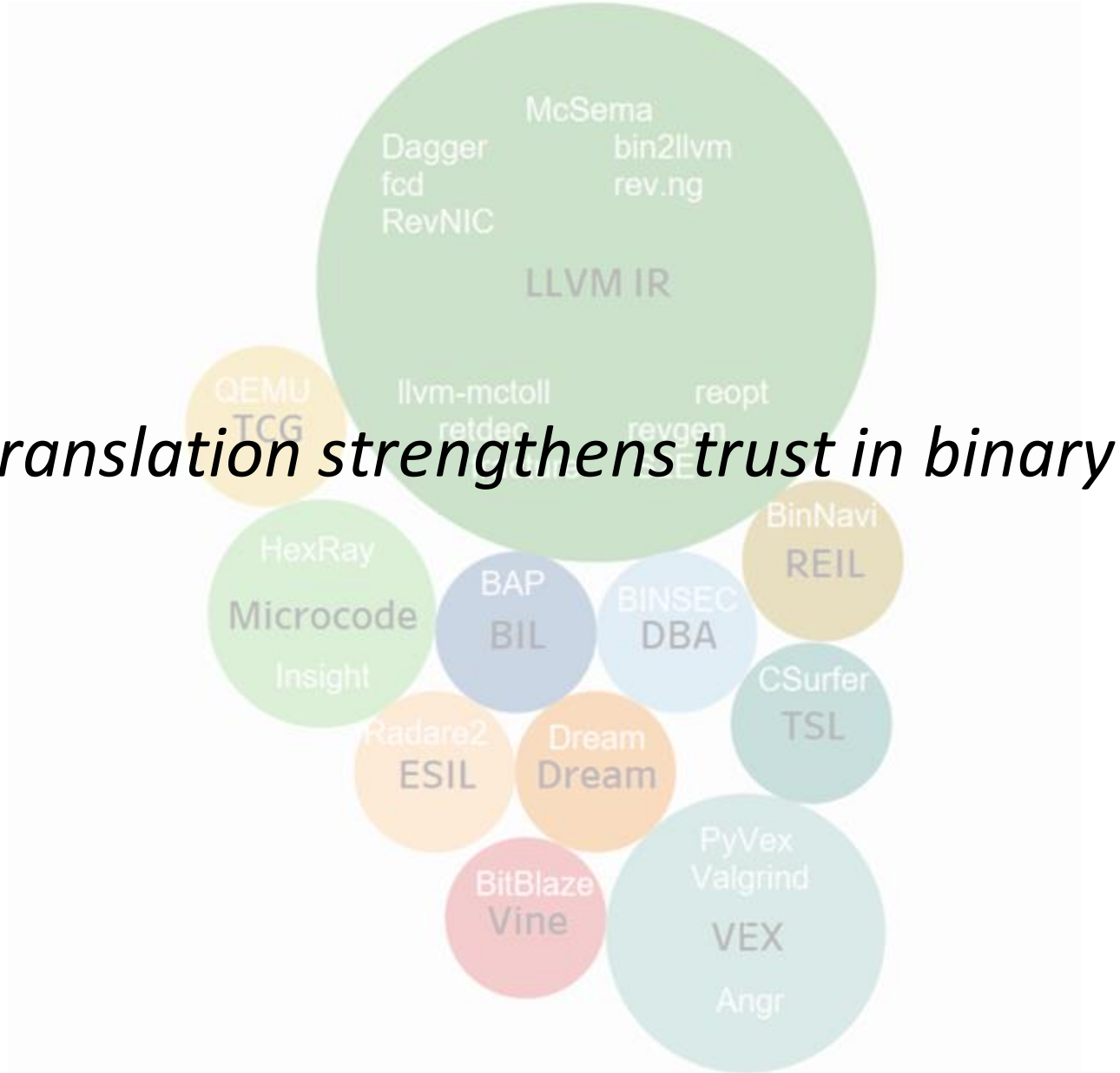
Semantics of Memory Variant
(**movsd (%rax), %xmm0**)

S1. XMM0[63:0] ← MEM_ADDR[63:0]
S2. XMM0[127:64] ← 0

Lifting is Pivotal in Binary Analysis



Validation of Lifting is Critical



Faithful binary translation strengthens trust in binary analysis results

Goal

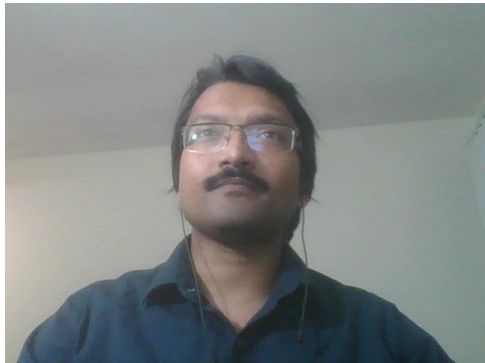
To develop formal and informal techniques to achieve high confidence in the correctness of binary lifting, from a complex machine ISA to a rich IR, by leveraging the semantics of languages involved

Summary of Prior Work

Require random testing	Restricted to instruction- or basic-block-level validation	Require instrumentation of lifter
• Martignoni et al. ISSTA'10 • Chen et al. CLSS'15	▪ Martignoni et al. ISSTA'10, ASPLOS'12 ▪ Chen et al. CLSS'15 ▪ Meandiff - Kim et al. ASE'17	▪ Reopt-vcg, John et al. SpISA'19

Scope of the work

To validate translation from **x86-64** programs to **LLVM IR** using **McSema**



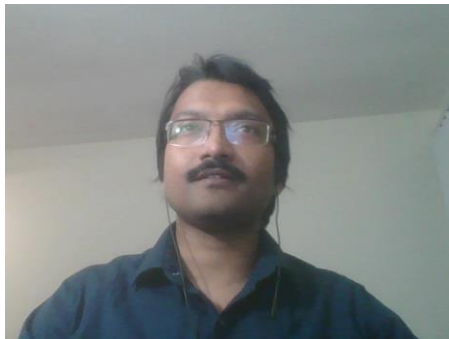
Our Approach: Intuition

Observation

Most binary lifters are designed to perform simple instruction-by-instruction lifting followed by standard IR optimizations to achieve simpler IR code

Intuition

Formal translation validation of single machine instructions can be used as a building block for scalable full-program validation



Our Two-Phase Approach

Phase I **S**ingle-**I**nstruction **T**ranslation-**V**alidation (SITV)

- ❖ Translation-validation of lifted instructions in isolation
- ❖ Leverages our prior work on formalizing x86-64 semantics[PLDI'19]

Phase II **P**rogram-**l**evel **V**alidation (PLV)

- ❖ A scalable approach for full-program validation build on SITV
- ❖ Cheaper than symbolic-execution based equivalence checking

Contributions

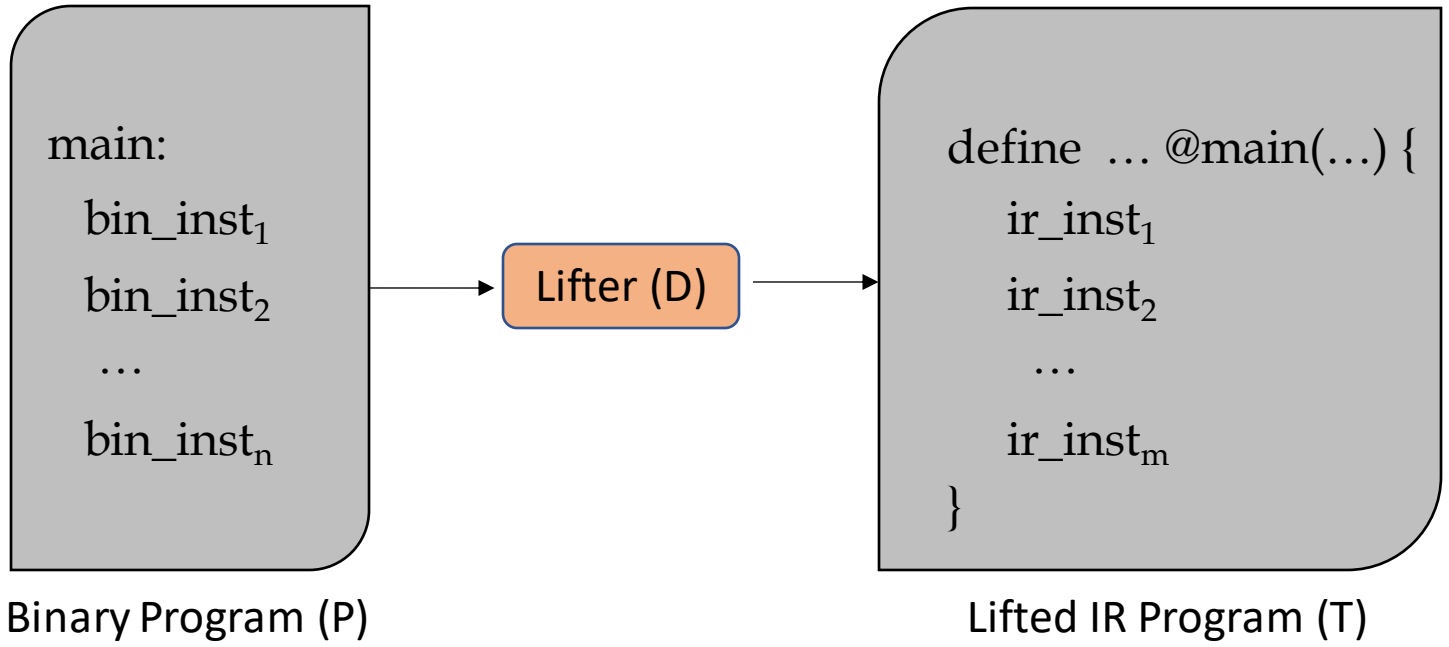
Developing scalable techniques for validating lifters

- ❑ **First SITV framework** for an extensive x86-64 ISA
- ❑ **Revealed Bugs** in a mature lifter like McSema
- ❑ Novel full-program validation **avoiding heavyweight symbolic execution**

Lifter Validation: Our Approach

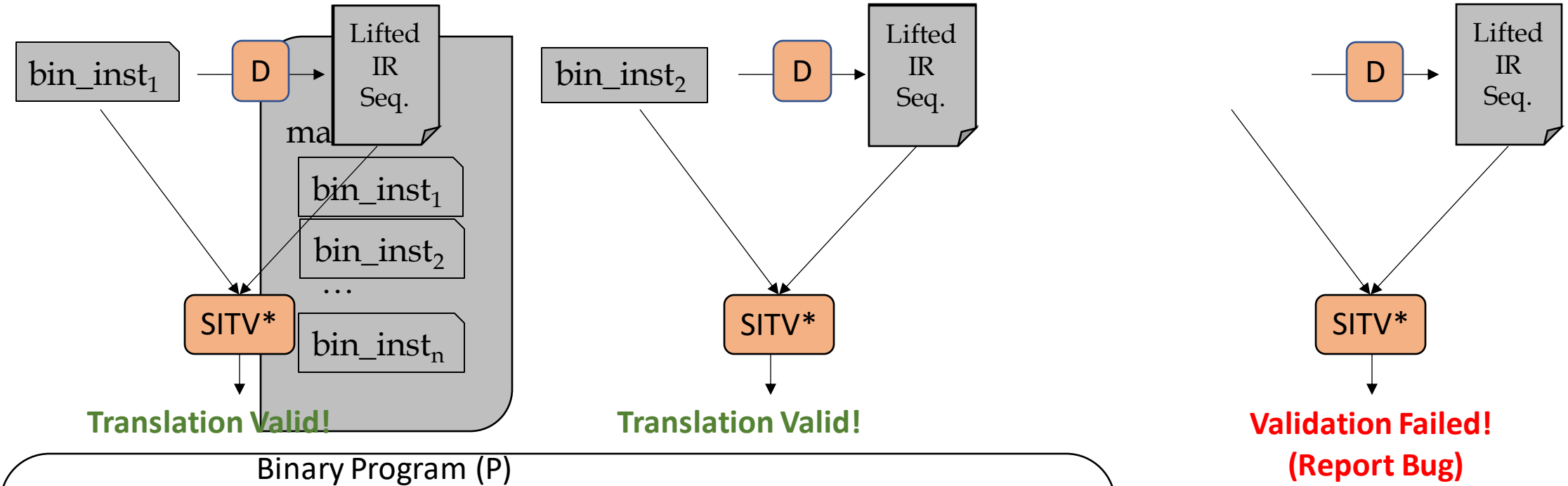
- **Phase I** **S**ingle-**I**nstruction **T**ranslation-**V**alidation (SITV)
- **Phase II** **P**rogram-**l**evel **V**alidation (PLV)

Overall Goal

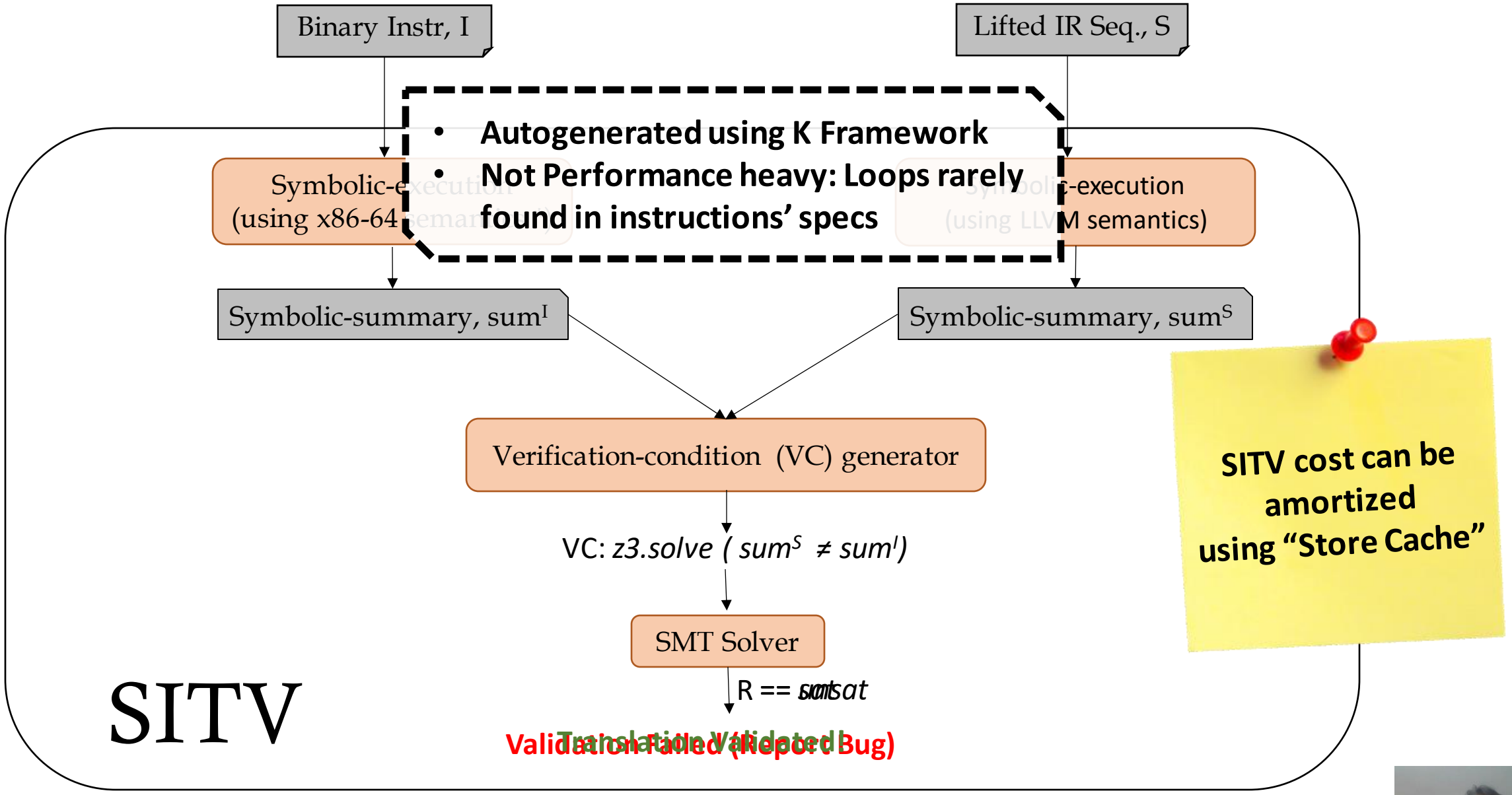


Our goal is to validate the translation from P to T

Single-Instruction Translation Validation

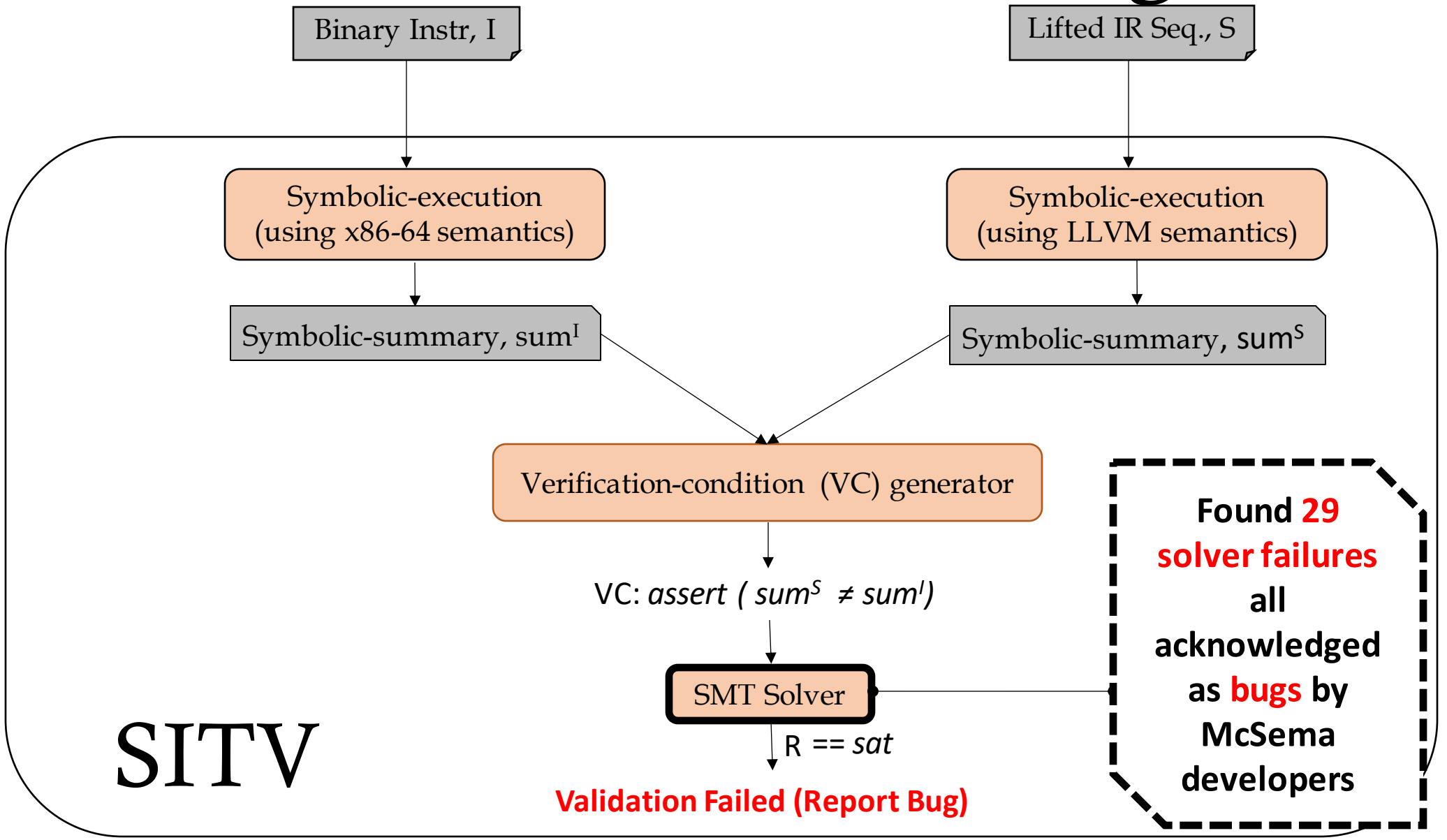


*SITV: Single Instruction Translation Validation Framework



*A Complete Formal Semantics of x86-64 User-Level Instruction Set Architecture, PLDI 2019

SITV: Revealed Bugs



SITV: A Few Reported Bugs

☑ Intel Manual Vol. 2: May 2019

xaddq %rax, %rbx

(1) temp ← %rax + %rbx
(2) %rax ← %rbx
(3) %rbx ← temp

🐞 McSema Implementation

xaddq %rbx, %rbx
(with same operands)

(A) old_rbx ← %rbx
(B) temp ← %rbx + %rbx
(C) %rbx ← temp
(D) %rbx ← old_rbx

SITV: A Few Reported Bugs

Intel Manual Vol. 2: May 2019

cmpxchgl %ecx, %ebx

```
TEMP ← ebx
IF eax = TEMP THEN
  ZF ← 1;
  ebx ← ecx;
ELSE
  ZF ← 0;
  eax ← TEMP;
  ebx ← TEMP;
FI;
```

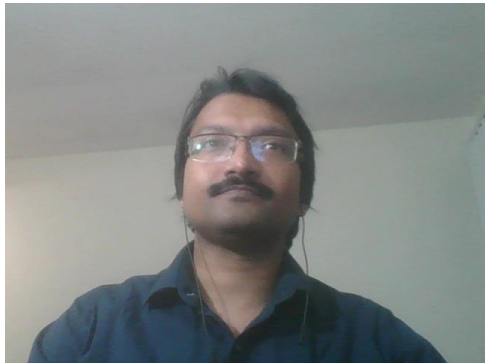
McSema Implementation

cmpxchgl %ecx, %ebx

```
TEMP ← rbx
IF (32'0 ◦ eax) = TEMP THEN
  ZF ← 1;
  ebx ← ecx;
ELSE
  ZF ← 0;
  eax ← TEMP;
  ebx ← TEMP;
FI;
```

Lifter Validation: Our Approach

- **Phase I** **S**ingle-**I**nstruction **T**ranslation-**V**alidation (SITV)
- **Phase II** **P**rogram-**l**evel **V**alidation (PLV)



PLV: Intuition

*Propose an alternate reference program, T', generated by carefully stitching the SIV-validated IR sequences (using **compositional lifting**) to be compared against T*

Semantic equivalence check between T & T' is reduced to a much "cheaper" graph-isomorphism check (using **Matcher**) through the use of semantic preserving transformations (using **Transformer**)

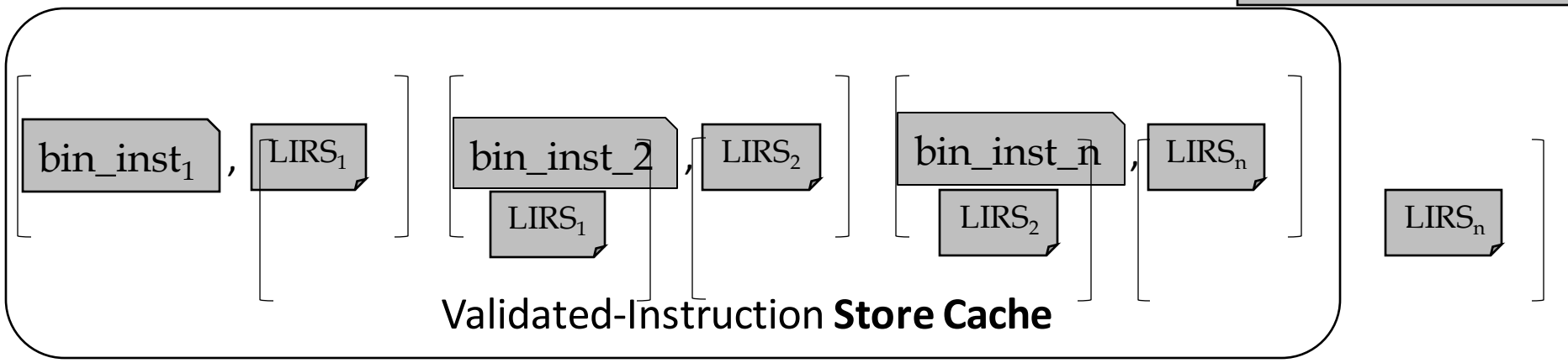
PLV: Compositional Lifting

Binary function (P)

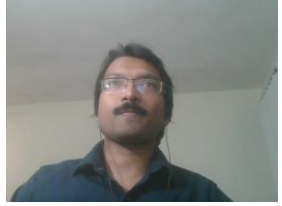
```
main:
  bin_inst1
  bin_inst2
  …
  bin_instn
```

Proposed IR Program, T'

```
define … @main(…) {
  glue code

  glue code

  …
  glue code

}
```



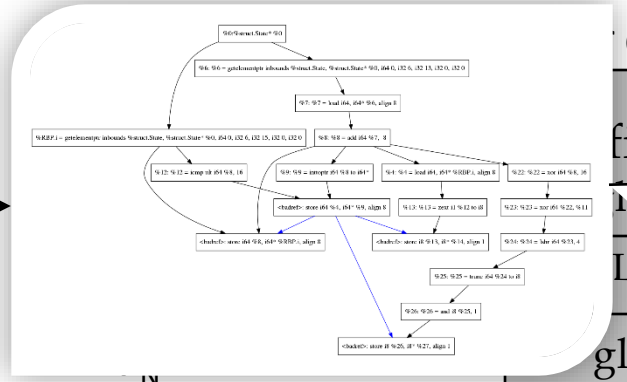
PLV: Transformation & Matching



```
define … @main(…) {
    ir_inst1
    ir_inst2
      …
    ir_instm
}
```

For each function F of T

Transformer (LLVM passes)



Data-Dependence Graph, G_N

each function F' of T'

```
fine … @main(…) {
```

$LIRS_1$

glue code

LIRS

…

glue

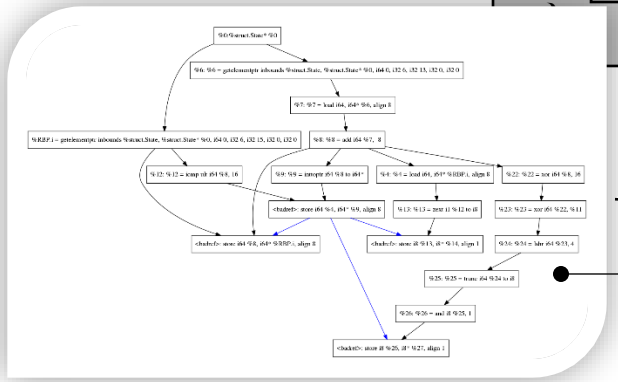
$LIRS_n$

Matcher *
Are G & G'_N isomorphic?

F & F' semantically equivalent

Possible Bug in Lifter

Transformer (LLVM passes)



Data-Dependence Graph, G'_N

Vertex:

Isomorphism of G_N & G'_N → semantic equiv. of F_N & F'_N, follows from *Horwitz et al.

* On the Adequacy of Program Dependence Graphs for Representing Programs, POPL'88

Transformer

- Prunes-off syntactic differences between T & T' except for
  - **Names of virtual registers**, and
  - **Order of non-dependent instructions**

Optimization passes NOT formally-verified

- Transformer uses a list of LLVM optimization passes one for each function pair

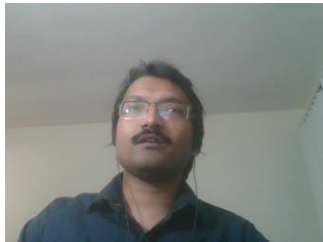
- Pass list is derived using pass sequence autotuning

Autotuning Based Transformer

Used OpenTuner* framework for autotuning

- **Search Space:** Includes 17 LLVM optimization passes (manually discovered)
- **Objective Function:** To maximize number of matching nodes of the candidate data dependence graphs

* OpenTuner: An Extensible Framework for Program Autotuning, PACT'14



PLV: Runtimes

Evaluated PLV on **2348** LLVM single-source benchmark functions

- **Compositional Lifting:** 0.05s – 5.57s, median – 0.63s
- **Autotuning:** 10.7 s - 20 m, median - 6.7 m
- **Matcher:** 0.06s – 119.6s, median – 5 s

PLV: Results

- Proved correctness of 2254 /2348 translations; **success rate - 96%**
- LOC of lifted IR: ranges from **86 – 32105, median - 611**
- Manual inspection shows the remaining **4%** to be **false alarms**

Summary

- Validation of lifters w/o instrumentation or heavyweight equivalence checking is feasible

- Formal translation validation of single machine instructions can be used as a building block for scalable full-program validation