D-Helix: A Generic Decompiler Testing Framework Using Symbolic Differentiation

Muqi Zou, Arslan Khan, Ruoyu Wu, Han Gao,
Antonio Bianchi, and Dave (Jing) Tian

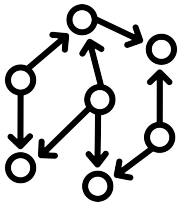
Purdue University

Decompilation Process

Binary

1010
1010

IR Language
(e.g LLVM IR, Pcode
and Vex IR)



Heuristics

Disassembler → Lifter → Decompiler → Printer

High Level Language
(e.g. C Code)

Motivation

Decompiler	Semantic-Preserving Heuristics	SLOC
Dream / Dream++	✓	12.9k
Foxdec	✓	2,924k
Phoenix	✓	-
Retdec	✗	2,437k
Ghidra	✗	4,258k
Reko	✗	6,764k
angr	✗	246.8k
Radeco	✗	40.5k
Rellic	✗	25.3k
llvm-cbe	✗	10.9k
rev.ng-c	✗	-
Hex-Rays	✗	-
JEB	✗	-
BinNinja	✗	-

Only 4 decompilers:

- *Mostly CFG related*

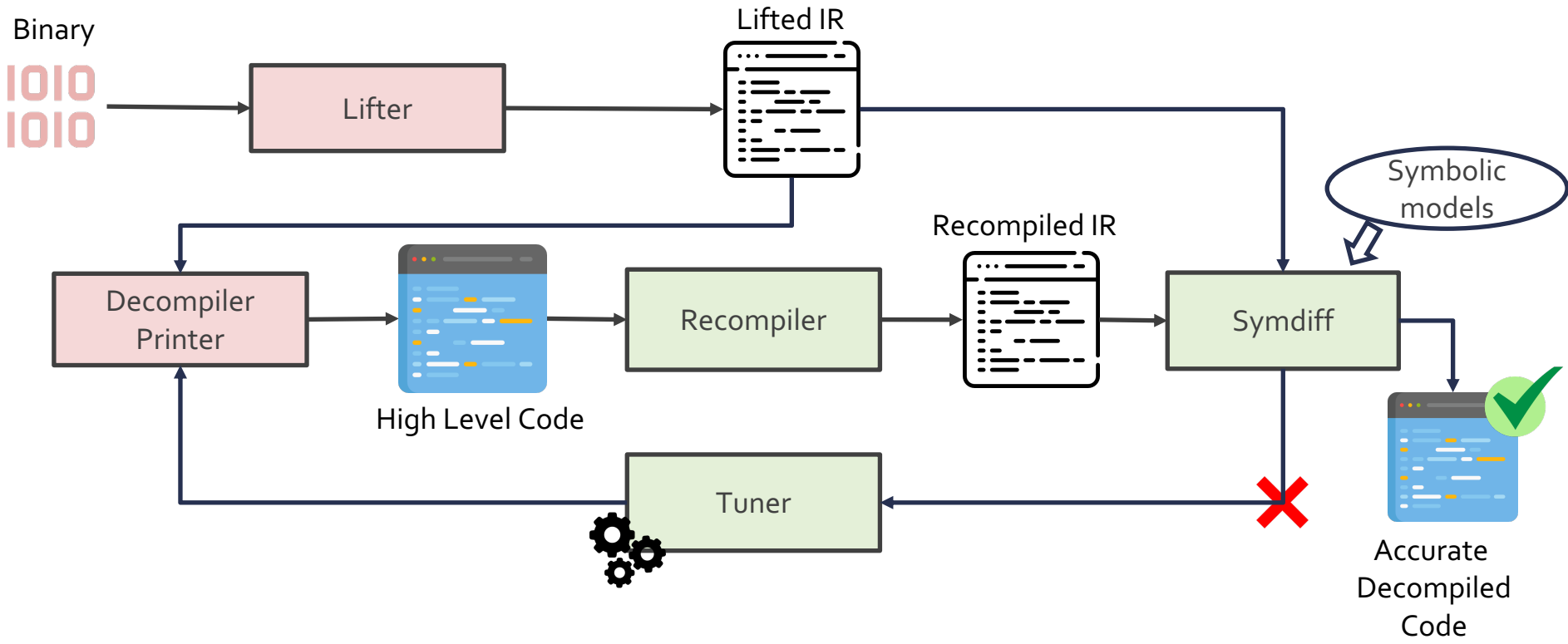
Inaccuracies in other decompilers:

- *Easy to be detected:*
  - *E.g., Special symbols : undefined (Ghidra)*
- ***Still exists semantic inaccuracies cannot be noticed without testing***

D-helix

Hence, we propose ***D-Helix***, a generic decompiler testing framework that can automatically vet the decompilation correctness.

Pipeline


Binary
1010
1010
Lifter
Lifted IR
Symbolic models
Recompiled IR
Decompiler Printer
High Level Code
Recompiler
Symdiff
Tuner
Accurate Decompiled Code

D-Helix components

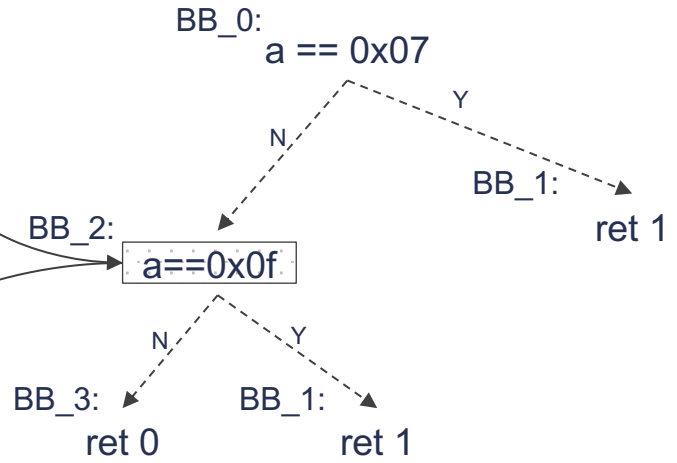
- Recompiler: error logs, automatic , iterative, function level.
- Symdiff: symbolic execution, SMT solver, IR level.
- Tuner: heuristic toggling, root cause identification.

Symbolic models example

Pseudo Code

```
1: f(uchar a){
2: if (a == 0x07)
3:       return 1;
4: if (a == 0x0f)
5:       return 1;
6: return 0;}
```

Constraint Graph



Symbolic Model

```
1: BB_0_2_1  = (1 32bits)
2: BB_0_2_3  = (0 32bits)
3: BB_0_2 = (ITE a==0x0f
4:     BB_0_2_1 BB_0_2_3)
5: BB_0_1 = (1 32bits)
6: BB_0 = (ITE a==0x07
       BB_0_1 BB_0_2)
```

Evaluation

- Evaluate Ghidra and angr.
- Test 56k functions from 2,004 binaries including coreutils, util-linux and top trending projects on Github.
- Find 25 bugs (17 unknown) in the two decompilers.

Evaluation, Symdiff

- Around 80% bugs in Ghidra :
  1. Function prototype recovery (43%)
  2. Literal value recovery (18%)
  3. Type recovery (18%)
- Around 80% bugs in angr:
  1. Function prototype recovery (40%)
  2. CFG recovery (20%)
  3. Missing instructions (20%)

Bug example:
(Ground Truth vs. Decompiled Code)

f(int a) vs. f(void)
x < 128 vs. x < -128
-0x18 vs. &DAT_ffffe8

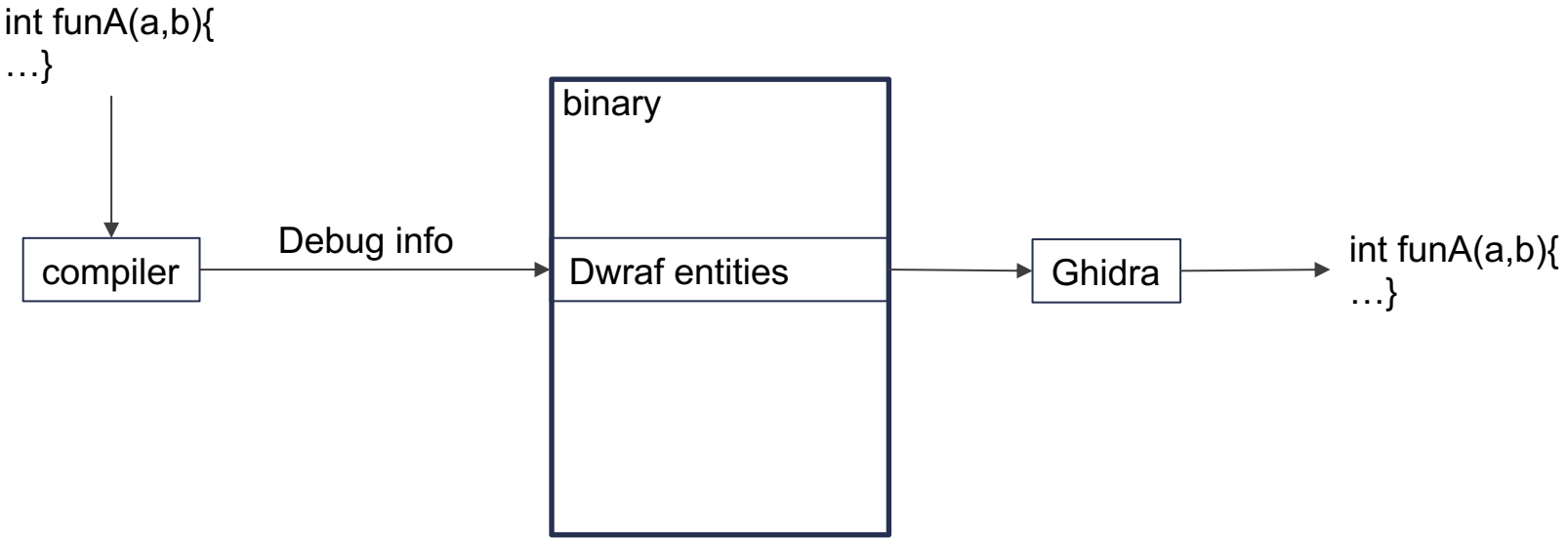
return a < 26; vs. return a;
while(){if…} vs. while(){…}
int a = 23; vs. //a is not declared

Evaluation, Tuner

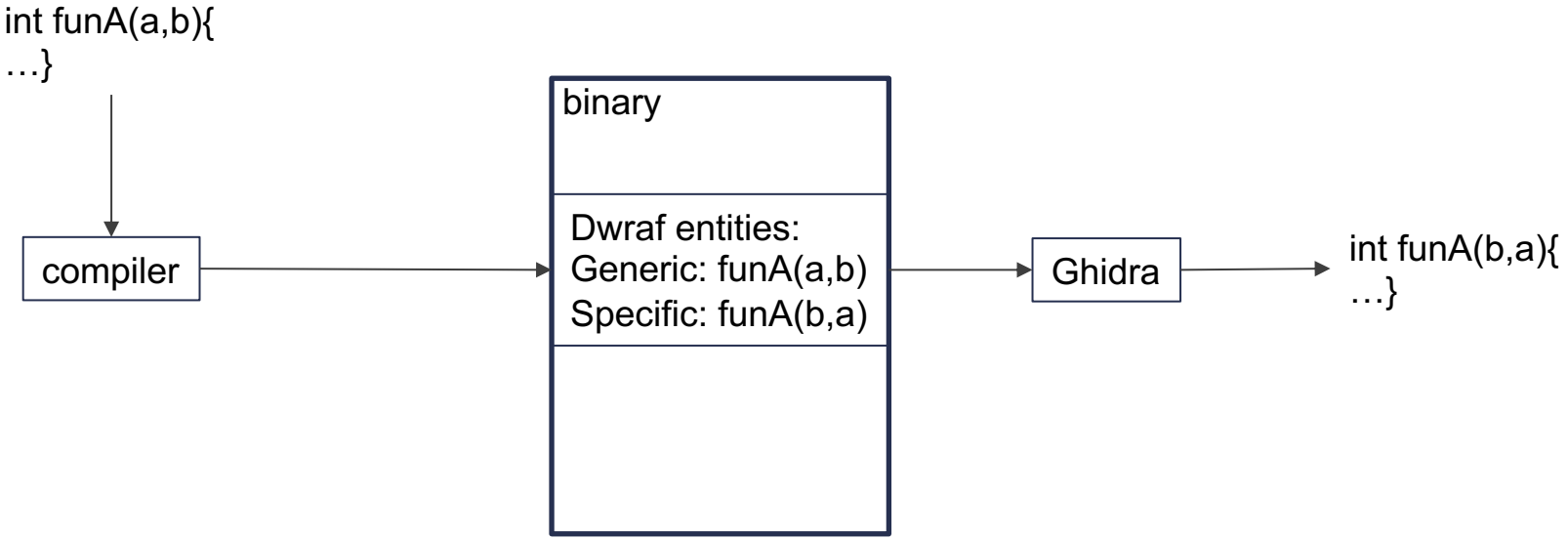
- Fix 16.3% functions in Ghidra :
  - DWARF, RuleSubvarSext, Apply Data Archives ,X86 Constant Reference Analyze.

- Other bugs in Ghidra and angr cannot be fixed:
  - Implementation bugs not related with heuristics.
  - Tuner still helps developers by ruling out some heuristic conjunctures.

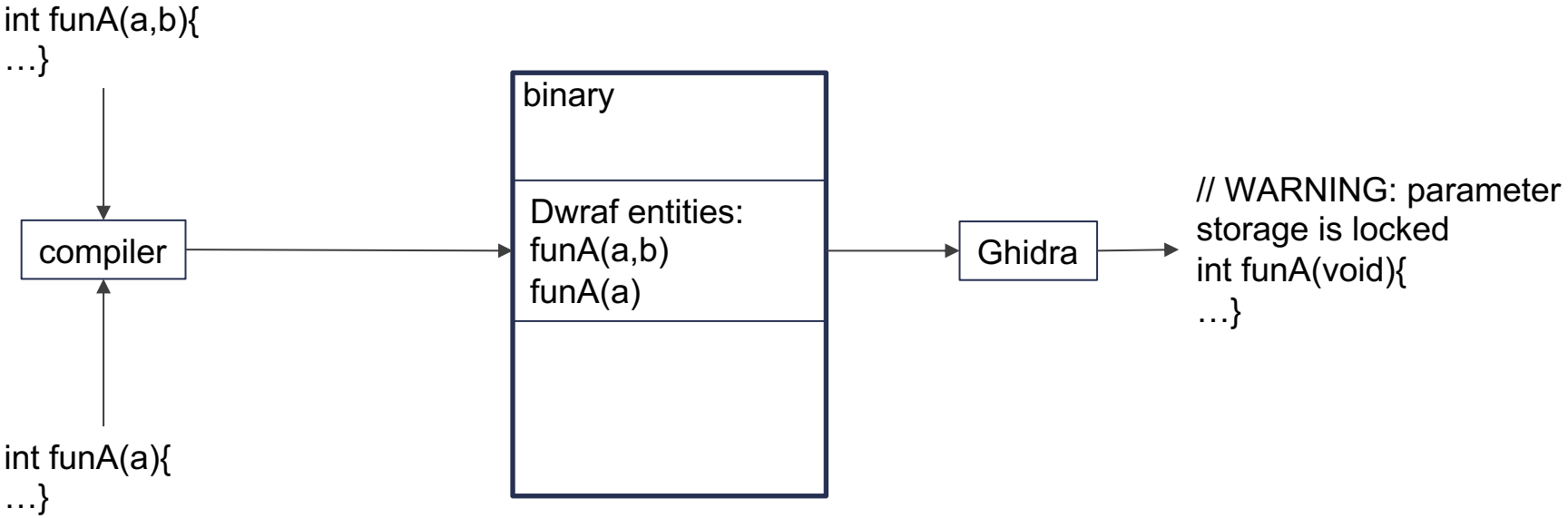
Case study: incorrect usage of DWARF information



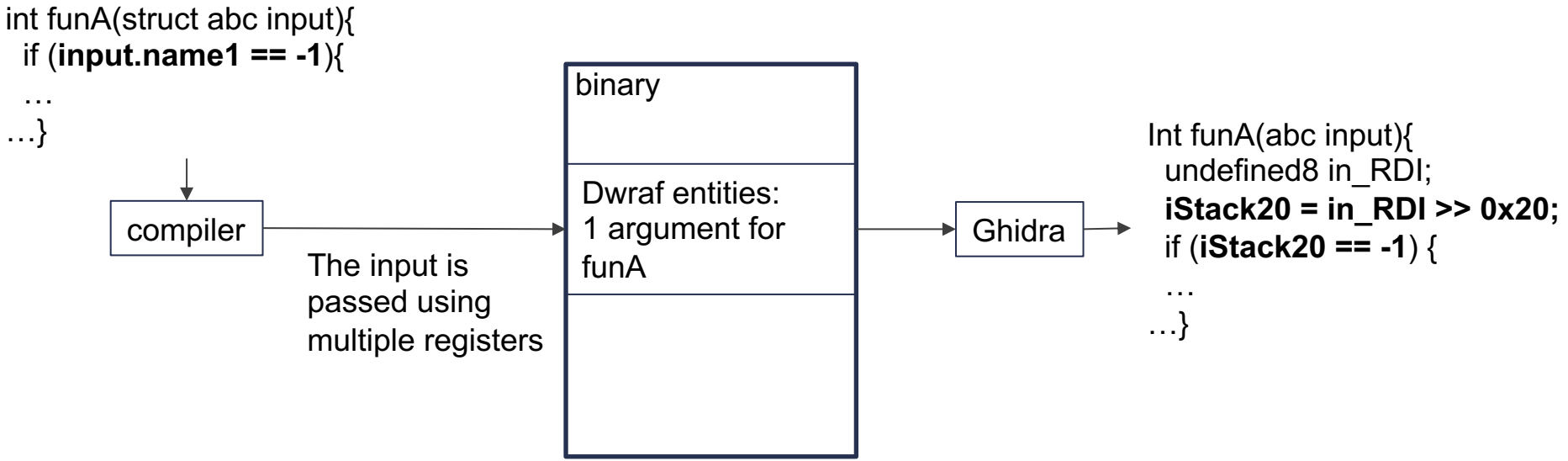
Case study: incorrect usage of DWARF information



Case study: incorrect usage of DWARF information



Case study: incorrect usage of DWARF information



All three cases can be fixed by disabling DWARF in Ghidra.

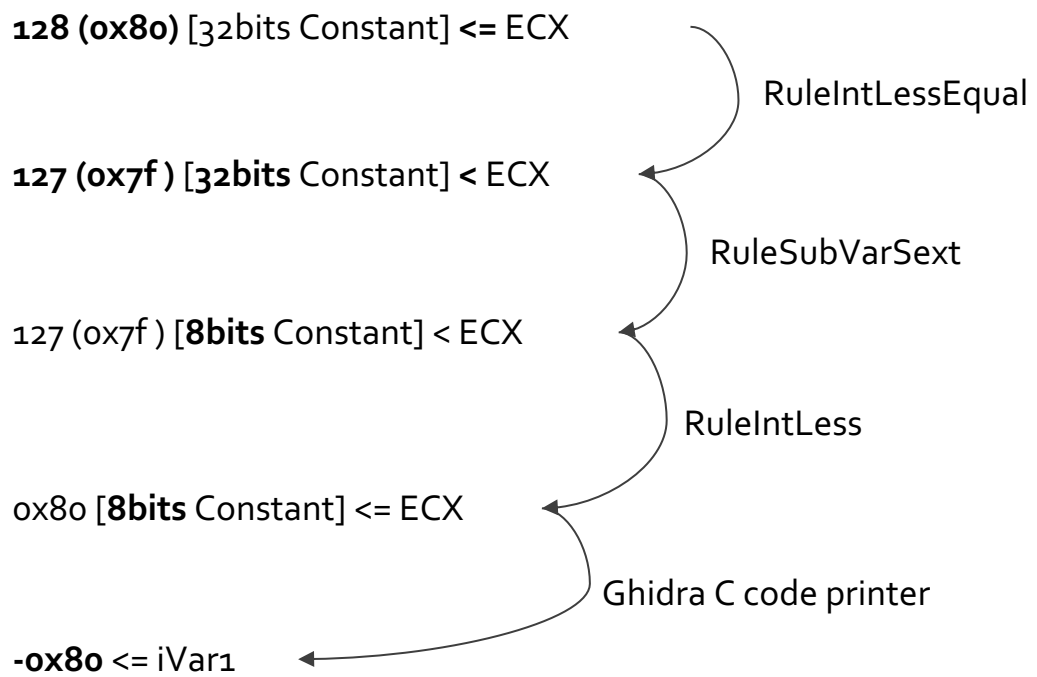
Case study: Constant 128 is decompiled as -128

Source code:

```
...
  char temp3;
  temp3 = getchar();
  if(temp3 < 128) {
...
```

Decompiled code:

```
...
  char iVar1;
  iVar1 = getchar();
  if(iVar1 < -0x80) {
...
```



This can be fixed by disabling the RuleIntLessEqual or RuleSubvarSext in Ghidra.

Summary

- D-Helix, a generic decompiler testing framework that can automatically vet the decompilation correctness.
- Find 25 bugs (17 unknown) in the two decompilers.
- D-Helix is open-source:
  - https://github.com/purseclab/Dhelix

Acknowledgement

- Thanks for timely response from Ghidra and angr teams
- This work was supported in part by NSF under grant NSF CNS-2145744, the Office of Naval Research (ONR) under grant N000014-23-1-2157, and the Defense Advanced Research Projects Agency (DARPA) under contract number N6600120C4031.

Thank you! Questions?

zou116@purdue.edu