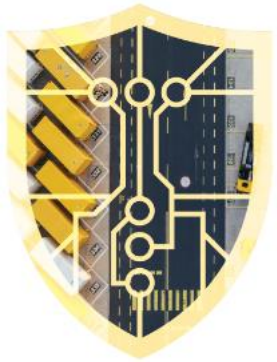
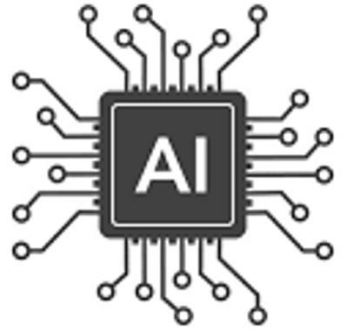




When Static Analysis Meets LLMs

A Neuro-Symbolic Approach

Chengpeng Wang

Email: wang6590@purdue.edu

Homepage: https://chengpeng-wang.github.io/

Oct. 29th , 2024

PURDUE UNIVERSITY

Programming in the AI Era

- *Copy-and-paste* the code from intelligent search engine
- *Prompt-and-paste* the code from LLM bots
- *Comment-and-select* the code recommended by LLM-powered IDE

A Nature Shift of Programming

- Shift from writing code to validating correctness
  - Writing code is cheap
  - Validating correctness is expensive

- Program analysis: Automate validating code correctness
  - Dynamic testing
  - Static analysis

- In this talk, we only discuss static analysis

Outline

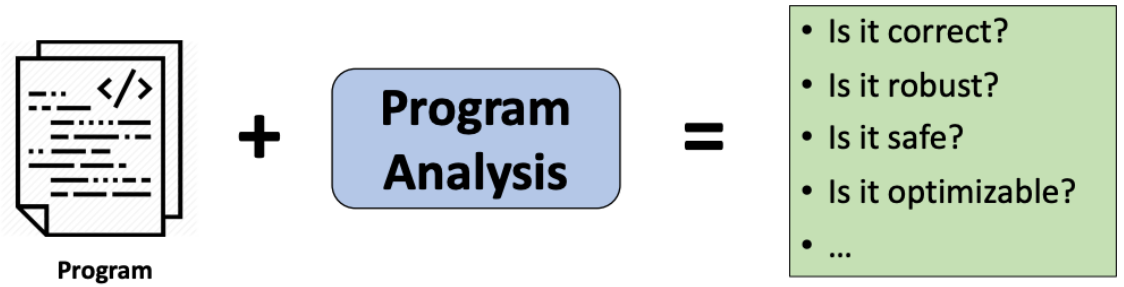
- Background of Static Analysis
- Recent Research on LLMs for Static Analysis
  - LLMDFA [NeurIPS' 24]
- Neuro-Symbolic Static Analysis

Outline

- Background of Static Analysis
- Recent Research on LLMs for Static Analysis
  - LLMDFA [NeurIPS' 24]
- Neuro-Symbolic Static Analysis

Static Analysis

- Statically analyze the behavior of a given program.
  - For any input, reason whether a specific assertion (program property) can always hold?
  - For specific inputs, reason whether a specific assertion (program property) may be violated?

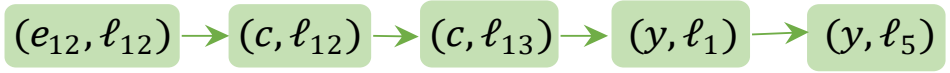


- Application:
  - Compiler optimization
  - Bug detection / Security auditing

Example: Divide-by-Zero Bug Detection

- Program property
  - y at Line 5 may have zero value.
- Analysis: Data Dependency Analysis
  - c at Line 13 depends on c at Line 12
  - y can be zero at Line 1
  - Hence, y at Line 5 can be zero
- Implication
  - Divide-by-Zero bug

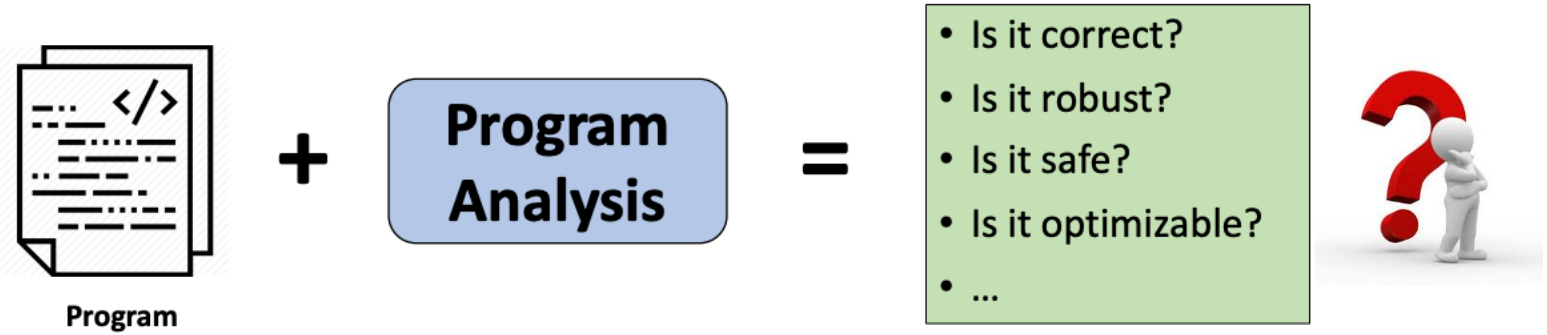
```
 1 public static void bar(int x, int y){
 2   if (x != 0)
 3     return (y * 1.0 / x);
 4   else
 5     return (x * 1.0 / y); //bug
 6 }
 7 public static void main(){
 8   int a = 0; //zero
 9   int b = parseInt("123");
10   System.out.println(bar(b, a));
11   String arg = args[0];
12   int c = parseInt(arg); //potential zero
13   System.out.println(bar(a, c));
14   c = b;
15   System.out.println(bar(a, c));
16 }
```



Data-flow Path

Static Analysis as Undecidable Problem

- (**Rice Theorem**) Verifying a semantic property is undecidable for a program in Turing-complete language.



- Unable to design a specific algorithm that always give a correct answer in finite time.
  - Over/under-approximation: Introduce false positives/negatives

Constraint Solving in Static Analysis

- SMT solvers serve as fundamental building blocks in static analysis

- Encode the program semantics with logical constraints
  - Symbolic execution: Klee
  - Path-sensitive data-flow analysis: Pinpoint
  - Memory-safety detector: Facebook Infer

Great Success: Separation Logic and Facebook Infer

Professor Peter O'Hearn is Awarded the 2016 Gödel Prize

Peter O'Hearn, Professor of Computer Science and member of UCL's Programming Principles, Logic and Verification Research (PPLV) Group, has been awarded the 2016 Gödel Prize. Peter will receive the award jointly with Stephen Brookes, Professor of Computer Science at Carnegie Mellon University, at the 43rd International Colloquium On Automata, Languages and Programming (ICALP 2016), 12-15 July 2016, in Rome, Italy.

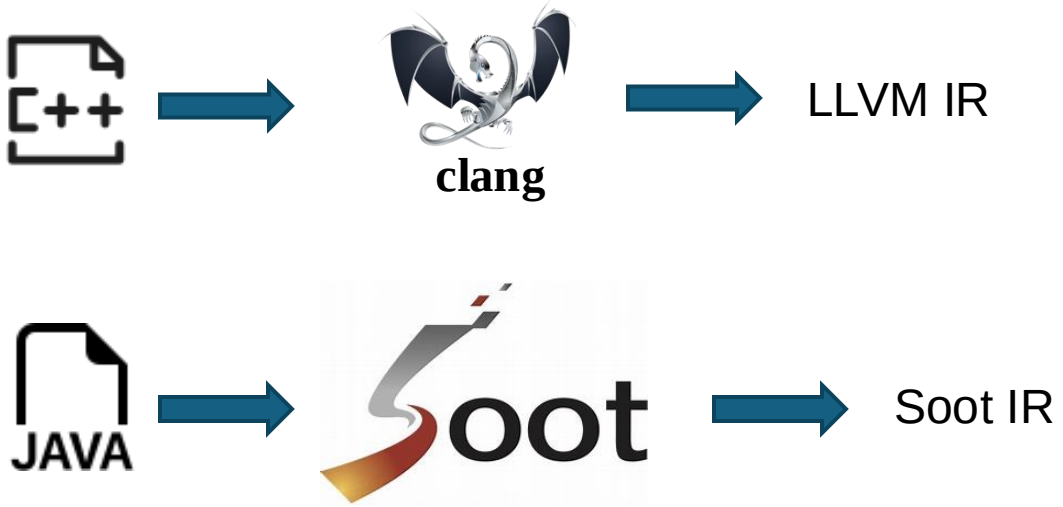
Peter's paper Resources, Concurrency and Logical Reasoning introduces and advances the idea of Concurrent Separation Logic (CSL), which has had a far-reaching impact in both theoretical and practical realms.

For the last thirty years experts have regarded shared-memory concurrency as one of the greatest challenges in program verification. CSL is a revolutionary advance on previous approaches. It builds on the Separation Logic for sequential programs due to Peter and the late John Reynolds, using the separation idea to support modular reasoning.

Mainstream Static Bug Detectors

Research Gap: Compilation Reliance

- Existing semantic analysis requires intermediate representations (IRs) of programs generated by successful compilation
  - Fail to discover security vulnerabilities in the incomplete code during the development




```java
public static String getFindBugsFullVersion() { return Version.VERSION_STRING; }

public static boolean isFindBugsError(@NotNull final Throwable error) {
  /ThrowableResultOfMethodCallIgnored/
  final Throwable cause = ErrorUtil.getCause(error);
  final StackTraceElement[] stack = error.getStackTrace();
  if (stack != null && stack.length > 0) { // just in case - think of -XX:-OmitStackTraceInFastThrow
    final String className = stack[0].getClassName();
    if (className != null) {
      return className.startsWith("edu.umd.cs.findbugs.") || className.startsWith("org.apache.bcel.");
    }
  }
  return false;
}
```

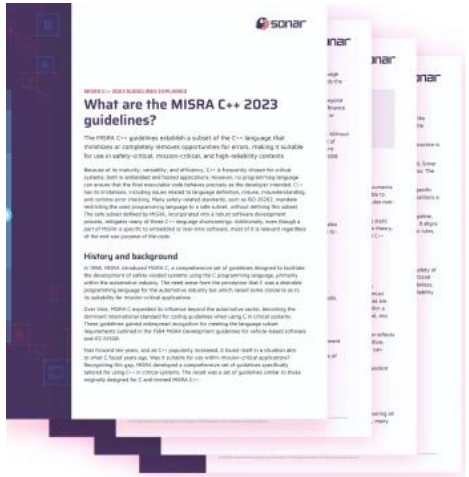
Research Gap: Customization

- Require comprehensive specifications
- Require hacking the compilation infrastructure
  - Example: Traverse abstract syntax tree (AST) or LLVM IR for C/C++ analysis

```
public static void main(){
  int a = 0;
  int b = new_function("123");
  System.out.println(bar(b, a));
  String arg = args[0];
  int c = new_function(arg);
  System.out.println(bar(a, c));
  c = b;
  System.out.println(bar(a, c));
}
```

Divide-by-zero bug detection

Customized specification



Dynamic code execution should not be vulnerable to injection attacks

Vulnerability

NoSQL operations should not be vulnerable to injection attacks

Vulnerability

HTTP request redirections should not be open to forging attacks

Vulnerability

Deserialization should not be vulnerable to injection attacks

Vulnerability

More bug types

Reshaping Static Analysis with LLMs

- LLMs: Magic Box
  - Input: Program + Questions
  - Output: Answers
- New opportunities
  - SMT solvers -> LLMs
- New challenges
  - ???

Peter O'Hearn · 1 度
Researcher at Meta AI (FAIR), Prof at University College ...
1 天前 ·

Some news. I've joined Meta AI (FAIR), to do research related to reasoning.

There are many interesting and challenging technical problems to work on. These include teaching neural nets to reason about code, interfacing them with symbolic reasoning components, using reasoning and semantics to help with code synthesis, and much more. Reasoning and planning are also a central part central to FAIR's broader vision for nextgen AI systems, for human-level machine intelligence (or AGI), and I fully expect to do work on reason+planning for other-than-code as well.

Additionally, I'm attracted to the open research culture of FAIR, which emphasizes collaboration and publication of research to the community, and I like the way Meta AI has embraced open source with Llama.

This all promises to be great fun!

Outline

- Background of Static Analysis
- Recent Research on LLMs for Static Analysis
  - LLMDFA [NeurIPS' 24]
- Neuro-Symbolic Static Analysis

Chengpeng Wang, Wuqi Zhang, Zian Su, Xiangzhe Xu, Xiaoheng Xie, and Xiangyu Zhang. **LLMDFA: Analyzing dataflow in code with large language model**. Advances in Neural Information Processing Systems, 2024.

Data-flow Analysis Problem

- Given the sets of sources/sinks in specific forms, determine whether the value of a source can reach a sink.

- Example: DBZ bug detection, Cross-Site Scripting(XSS) bug detection

```
1 public static void bar(int x, int y){
2   if (x != 0)
3     return (y * 1.0 / x);
4   else
5     return (x * 1.0 / y); //bug
6 }
7 public static void main(){
8   int a = 0; //zero
9   int b = parseInt("123");
10  System.out.println(bar(b, a));
11  String arg = args[0];
12  int c = parseInt(arg); //potential zero
13  System.out.println(bar(a, c));
14  c = b;
15  System.out.println(bar(a, c));
16 }
```

```
/* read string from file into data */
streamFileInput = new FileInputStream(file);
readerInputStream = new InputStreamReader(streamFileInput, "UTF-8");
readerBuffered = new BufferedReader(readerInputStream);
/* POTENTIAL FLAW: Read data from a file */
/* This will be reading the first "line" of the file, which
 * could be very long if there are little or no newlines in the file */
data = readerBuffered.readLine();

if (data != null)
{
    /* POTENTIAL FLAW: Display of data in web page after using replaceAll() to remove script tags,
    response.getWriter().println("<br>bad(): data = " + data.replaceAll("(<script>)", ""));
}
```

Understanding Programs with LLMs

- LLMs are good interpreters of programs in different languages.
  - Chain-of-Thought: What's the value of `z` used at the line 14? Please think step by step.

```
1  public class Demo        {
2    public static          int    foo(  int    a,   int    b){
3      if    ( Math.abs(b) > 1)
4          System.out.println(a / b);
5      return      b;
6    }
7    public static void             main(  String[] args){
8      int    x = 1;
9      int    y = 10 / x;
10     x = 0;
11     y = x + 1;
12     int    z = x / y;
13     z = x;
14     System.out.println(y / z);
15     x  =  foo(y, z);
16     System.out.println(x           /   x);
17   }
18 }
```

ChatGPT Poe

The value of the variable z used at line 14 is 0.

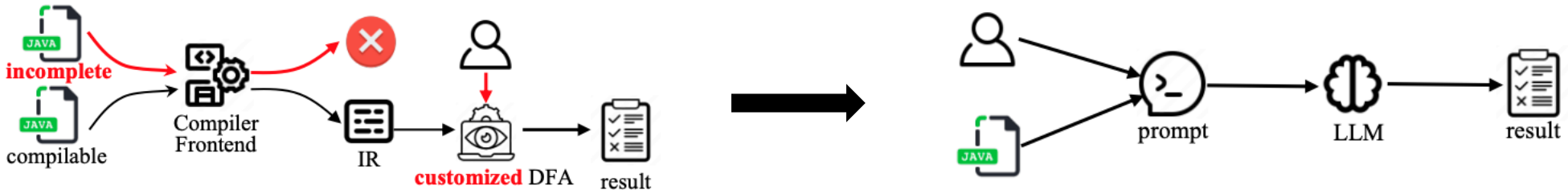
Let's go through the code step by step:

1. `int x = 1;` - Initializes x with the value 1.
2. `int y = 10 / x;` - Calculates y as the result of dividing 10 by the value of x, which is 1. So y is assigned the value 10.
3. `x = 0;` - Updates the value of x to 0.
4. `y = x + 1;` - Assigns y as the result of adding the value of x (0) and 1. So y is assigned the value 1.
5. `int z = x / y;` - Calculates z as the result of dividing the value of x (0) by the value of y (1). So z is assigned the value 0.
6. `z = x;` - Updates the value of z to the value of x, which is 0.
7. `System.out.println(y / z);` - Prints the result of dividing y (1) by z (0), which is not defined and would result in an exception.
8. `x = foo(y, z);` - Calls the `foo` method with arguments y (1) and z (0). The `foo` method divides *a* by b only if the absolute value of b is greater than 1. Since b is 0, the division is not performed, and the `foo` method returns 0. So x is assigned the value 0.
9. `System.out.println(x / x);` - Prints the result of dividing x (0) by x (0), which would result in an exception since the division by zero is not defined.

Therefore, the value of z used at line 14 is 0.

Our Vision

- LLMs for Data-flow Analysis
  - Analyze code as specific kind of textual data
    - No need to build the code
  - Easy to customize the analysis
    - Analyze code via prompting
  - Align with program semantics
    - No need to propose any abstraction upon the code



Current Static Analysis

Our Proposal

New Challenge

- Hallucination degrades the accuracy
  - Does the value of the variable z defined at the line 13 depends on the value of the variable x defined at the line 10?

```
1  public class Demo        {
2    public static          int    foo(  int    a,   int    b){
3      if    ( Math.abs(b) > 1)
4        System.out.println(a / b);
5      return     b;
6    }
7    public static void            main(  String[] args){
8      int    x = 1;
9      int    y = 10 / x;
10     x = 0;
11     y = x + 1;
12     int    z = x / y;
13     z = x;
14     System.out.println(y / z);
15     x  =  foo(y, z);
16     System.out.println(x           /   x);
17   }
18 }
```

ChatGPT Poe

No, the value of the variable z defined at line 13 does not depend on the value of the variable x defined at line 10.

Let's go through the code again to understand the sequence of operations:

1. `int x = 1;` - Initializes x with the value 1.
2. `int y = 10 / x;` - Calculates y as the result of dividing 10 by the value of x, which is 1. So y is assigned the value 10.
3. `x = 0;` - Updates the value of x to 0.
4. `y = x + 1;` - Assigns y as the result of adding the value of x (0) and 1. So y is assigned the value 1.
5. `int z = x / y;` - Calculates z as the result of dividing the value of x (0) by the value of y (1). So z is assigned the value 0.
6. `z = x;` - Updates the value of z to the value of x, which is 0.

As we can see, the assignment of the value 0 to z at line 6 does not depend on the value of x at line 10. It simply assigns the value of x (0) to z.

Therefore, the value of the variable z defined at line 13 does not depend on the value of the variable x defined at line 10.

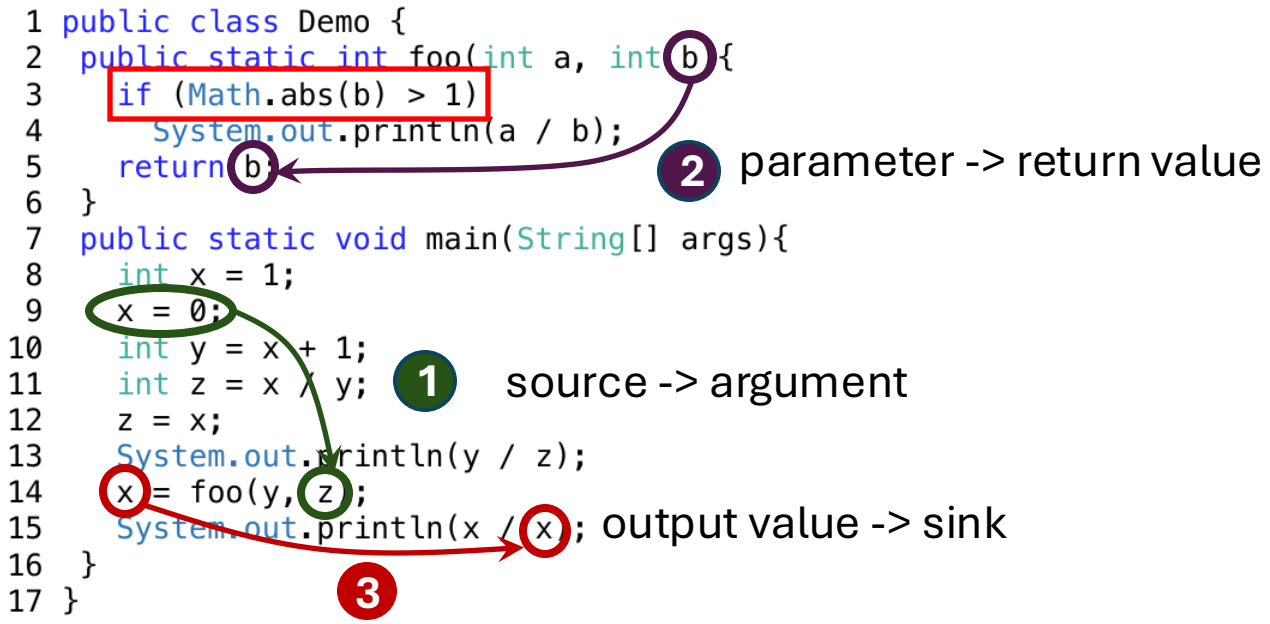
Central Issue: Mitigating Hallucinations

- Tricky case
  - Lengthy *program* + complex *bug*

- Divide-and-Conquer
  - Decompose the program into small code snippets
  - Decompose the bug specification into several simpler properties
  - Solve and merge

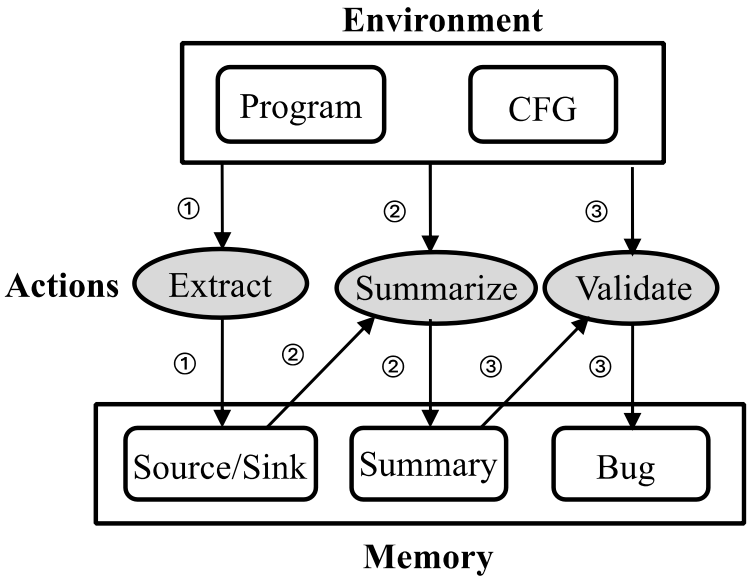
Key Idea: Summary-based Data-flow Analysis

- Example: DBZ detection



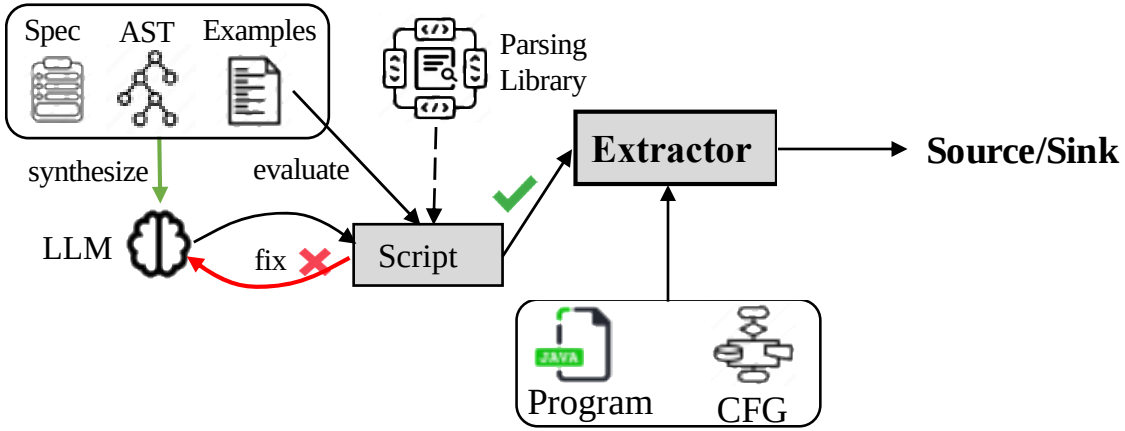
LLMDFA: Agent-Centric Solution

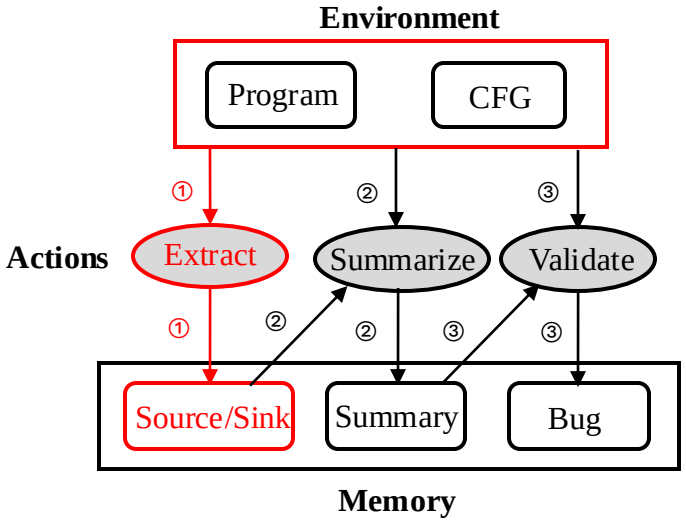
- Instantiate summary-based DFA with LLMs
  - P-I: Source/sink identification
  - P-II: Dataflow summarization
  - P-III: Path feasibility validation



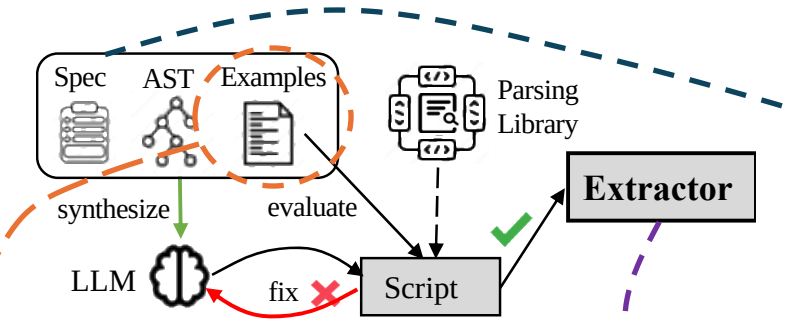
Extract

- Utilize LLMs to synthesize source/sink extractors
  - Structural search in most cases
    - Tree traversal + string match
  - Multi-modal synthesis problem
    - Spec: Programming-by-Natural Language
    - AST + Example code: Programming-by-Example





Example: Source/Sink Extractor Synthesis



Role: You are a good programmer and familiar with AST of programs.
Description: Please write the Python script traversing AST and identify sources/sinks for data flow analysis.

Source/Sink Info: There are several forms of sources/sinks: [Spec]. Also, we offer several example programs containing sources/sinks and their corresponding ASTs: [Example Programs + ASTs]

Synthesis Task: Please write a Python script to extract the sources/sinks on AST. You may refer to the AST structure of the example programs, and a skeleton AST traverser program. [skeleton]

Fixing Task: Here is the synthesized result of last round: [script]. When executing the script, we encounter the following error: [error message]. Here are missed sources/sinks are missed: [missed ones]. Here are the variables misidentified as sources/sinks: [incorrect ones]. Please fix it and return a runnable script.

Prompt Template

```
int x1 = 0; // src: x1
double x2 = 0.0; // src: x2
float x3 = 0.0f; // src: x3

var s = "0";
int x4 = parseInt(s); //src: x4

int z = 1;
int a = z / x; // sink: x
int b = z % y; // sink: y
int c = x + y;
```

Examples used for synthesizing extractors

```
def is_interesting(node):
  return (node.type=="binary_expr"
      and (node.op == "%" or
           node.op == "/"))
def traverse(node, sinks):
  if is_interesting(node):
    sinks.append(node.sec_operand)
  return sinks
  for child in node.children:
    sinks = traverse(child, sinks)
sinks = traverse(ast_root, [])
```

Synthesized Sink Extractor for DBZ detection

Summarize

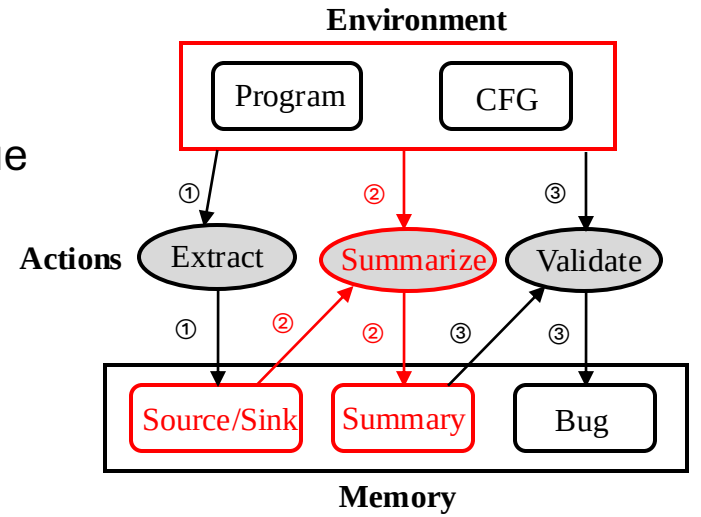
- A dataflow path from source to sink consists of several intra-procedural dataflow summaries
  - Focus on intra-procedural dataflows only

```
1  public class Demo {
2   public static int foo(int a, int b){
3     if (Math.abs(b) > 1)
4       System.out.println(a / b);
5     return b;
6   }
7   public static void main(String[] args){
8     int x = 1;
9     x = 0;
10    int y = x + 1;
11    int z = x / y;
12    z = x;
13    System.out.println(y / z);
14    x = foo(y, z);
15    System.out.println(x / x);
16  }
17 }
```

2 parameter -> return value

1 source -> argument

3 output value -> sink



Example: Dataflow Summarization

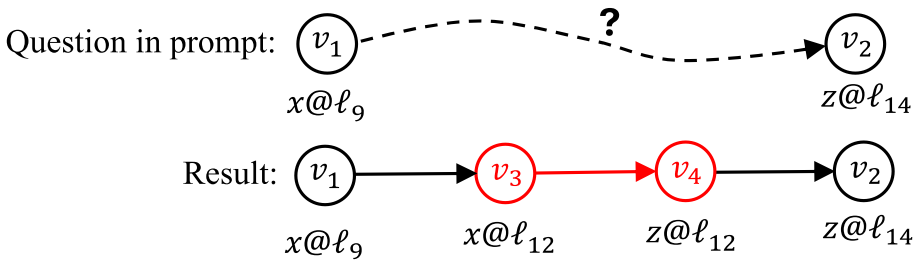
Role: You are a good Java programmer. You are good at understanding the semantics of Java programs.
Description: Determine whether two variables at two lines have the same value.

Here are several **rules**:
(1) If they are the same variable and not overwritten between two lines, the answer should be yes.
(2) If the variable a is assigned with the value of the variable b, then answer should be yes. [Other rules]

Here are several **examples**:
Example 1: User: [Program] [Question]
System: [Answer: Yes] [Explanation: y is assigned with x at line 2 and not over-written between lines 2 and 3. Hence, the value of y at line 3 is the same as x defined at line 1. The answer is Yes.]
[Other examples]

Question: Now I give you a function: [FUNCTION]
Please answer: Does [VAR1] used at line [L1] have the same value as [VAR2] defined at line [L2] ?
Please think it step by step. Return Yes/No with the explanation.

Prompt Template

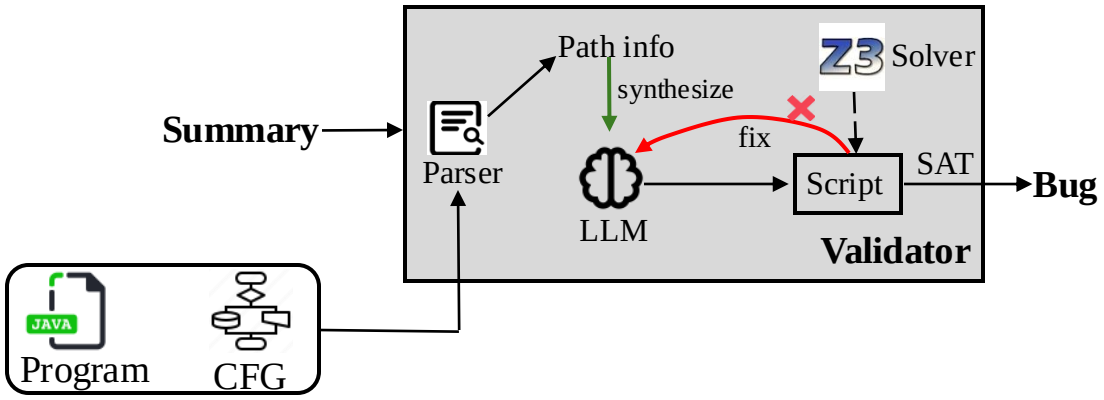


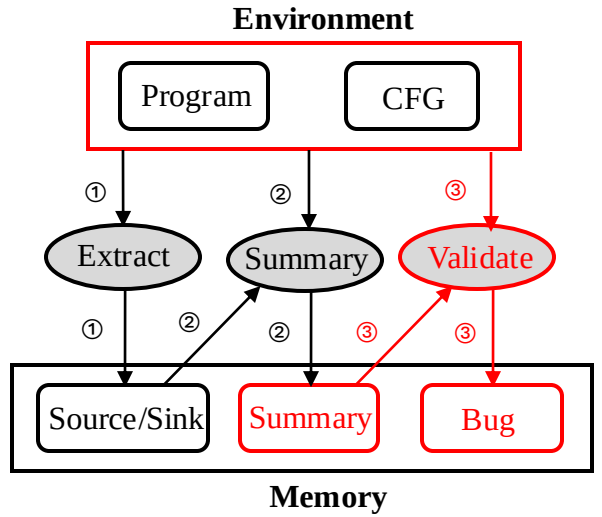

```
1 public class Demo {
2  public static int foo(int a, int b){
3    if (Math.abs(b) > 1)
4      System.out.println(a / b);
5    return b;
6  }
7  public static void main(String[] args){
8    int x = 1;
9    x = 0;
10   int y = x + 1;
11   int z = x / y;
12   z = x;
13   System.out.println(y / z);
14   x = foo(y, z);
15   System.out.println(x / x);
16 }
17 }
```

1 source -> argument

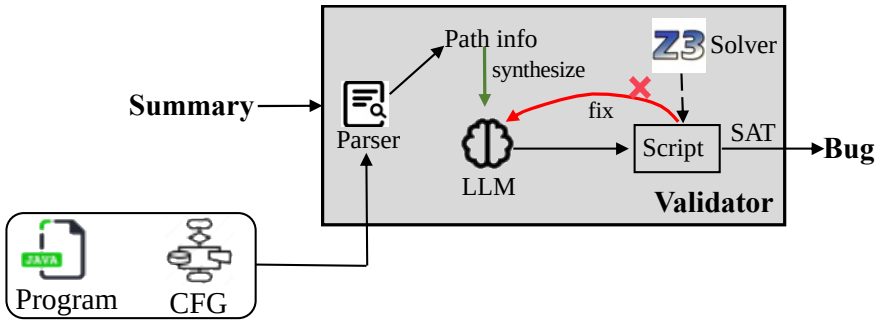
Validate

- Leverage LLMs to synthesize path validators
  - Stitch summaries across functions
  - Ask LLMs to encode and solve path conditions
    - Utilize Z3 Python binding (Z3: Domain Expert)





Example: Path Validator Synthesis



Role: You are a good programmer and familiar with Z3 python binding . **Description**: Please write a Python program using Z3 python binding to encode the path condition.
Path Info: Here is a path: [path]. Note that the value of [variable] is 0. Line [line number] is in the [true/false] branch of the if-statement, of which the condition is [branch condition].
Synthesis Task: Please write a Python script to solve the path condition using Z3 python binding. You can refer to the skeleton : [skeleton]
Fixing Task: Here is the synthesized result of last round: [script]. When executing the synthesized script, we encounter the following error: [error message]. Please fix the bug and return a runnable script.

Prompt Template

```
1  public class Demo {
2   public static int foo(int a, int b){
3     if (Math.abs(b) > 1)
4       System.out.println(a / b);
5     return b;
6   }
7   public static void main(String[] args){
8     int x = 1;
9     x = 0;
10    int y = x + 1;
11    int z = x / y;
12    z = x;
13    System.out.println(y / z);
14    x = foo(y, z);
15    System.out.println(x / x);
16  }
17 }
```

2 parameter -> sink

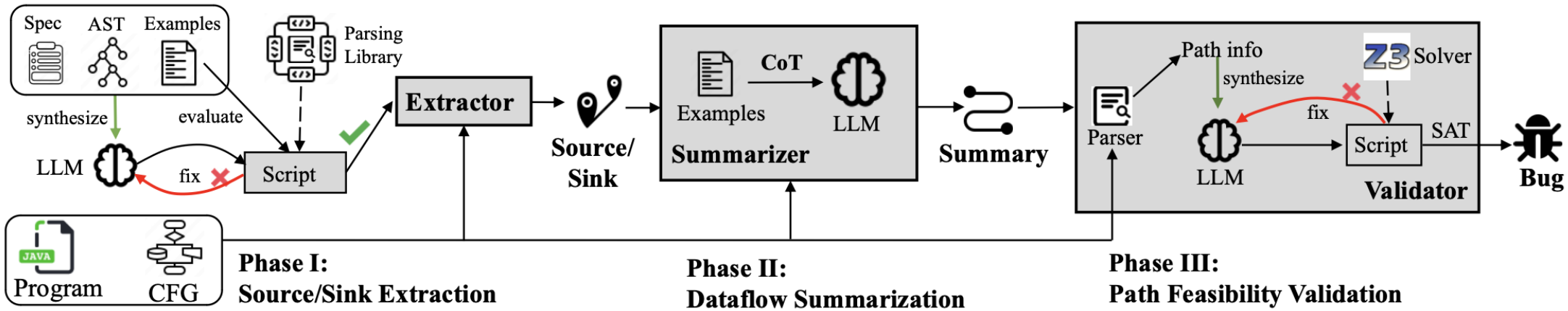
1 source -> argument

```
from z3 import *
s = Solver()
b = Int('b')
s.add(b == 0)
s.add(Abs(b) > 1)
print(s.check())
```

UNSAT!

Synthesized Path Validator

Workflow of LLMDFA



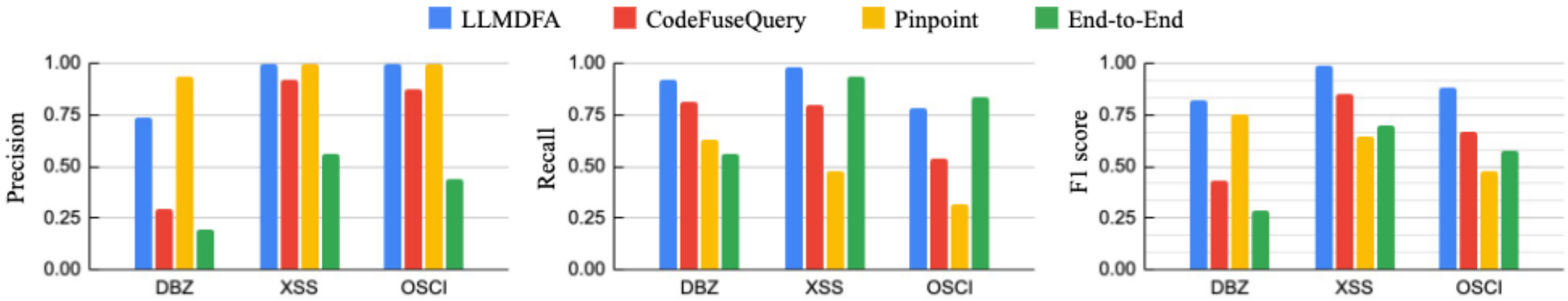
Main Results: Juliet Test Suite

➢LLMDFA

Bug	Phase	gpt-3.5			gpt-4			gemini-1.0			claude-3		
		P (%)	R (%)	F1	P (%)	R (%)	F1	P (%)	R (%)	F1	P (%)	R (%)	F1
DBZ	Extract	100.00	100.00	1.00	100.00	100.00	1.00	100.00	100.00	1.00	100.00	100.00	1.00
	Summarize	90.95	97.57	0.94	95.32	98.43	0.97	83.57	82.47	0.83	89.26	92.38	0.91
	Validate	81.58	99.20	0.90	89.76	100.00	0.95	79.83	93.73	0.86	85.74	94.52	0.90
	Detection	**73.75**	**92.16**	**0.82**	**81.38**	**95.75**	**0.87**	**66.57**	**74.21**	**0.70**	**76.91**	**82.67**	**0.80**
XSS	Extract	100.00	100.00	1.00	100.00	100.00	1.00	100.00	100.00	1.00	100.00	100.00	1.00
	Summarize	86.52	96.25	0.91	97.84	99.76	0.99	88.79	97.31	0.93	94.17	97.83	0.96
	Validate	100.00	100.00	1.00	100.00	98.91	0.99	100.00	99.07	1.00	100.00	95.29	0.98
	Detection	**100.00**	**92.31**	**0.96**	**100.0**	**98.64**	**0.99**	**100.00**	**94.60**	**0.97**	**100.0**	**86.49**	**0.93**
OSCI	Extract	100.00	100.00	1.00	100.00	100.00	1.00	100.00	100.00	1.00	100.00	100.00	1.00
	Summarize	89.57	85.76	0.88	94.58	93.12	0.94	87.21	96.54	0.92	98.26	97.87	0.98
	Validate	100.00	97.14	0.99	100.00	100.00	1.00	100.00	98.13	0.99	100.00	100.00	1.00
	Detection	**100.00**	**78.38**	**0.88**	**100.00**	**89.19**	**0.94**	**100.00**	**94.59**	**0.97**	**100.00**	**97.30**	**0.99**

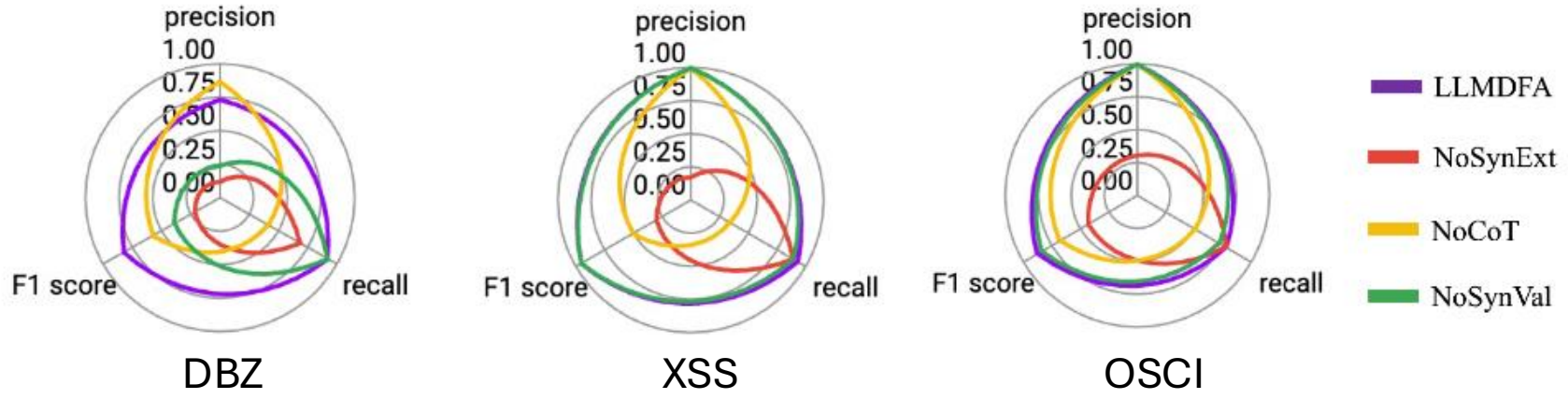
Main Results: Juliet Test Suite

- Comparison with baselines



Ablation Study: LLMDFA with GPT-3.5

- NoSynExt: Identify sources/sinks with LLMs via few-shot CoT
- NoCoT: Determine data-flow summaries without CoT
- NoSynVal: Examine path feasibility with LLMs without invoking solvers



DBZ XSS OSCI

LLMDFA: Takeaway

- Agent-centric solution
  - Tool using/synthesis
    - Used tool
      - Light-weighted static analyzer (Parser), Formal method (Z3 SMT solver)
    - Created tool
      - Source/sink extractors, path validators
  - Agentless (No planning)
    - Difficult for LLMs to discover an effective policy in static analysis

Outline

- Background of Static Analysis
- Recent Research on LLMs for Static Analysis
  - LLMDFA [NeurIPS' 24]
- Neuro-Symbolic Static Analysis

LLMDFA: Neuro-Symbolic Approach

- Symbolic analysis
  - Parsing-based source/sink extraction
  - SMT solving-based path feasibility validation

- Neural analysis
  - Code interpreter
  - Tool synthesizer

The Role of LLMs: Code Interpreter

➢Analyze incomplete code

```
1  public class Demo        {
2    public static          int    foo(  int    a,   int    b){
3       if    ( Math.abs(b) > 1)
4           System.out.println(a / b);
5       return      b;
6    }
7    public static void             main(  String[] args){
8       int    x = 1;
9       int    y = 10 / x;
10      x = 0;
11      y = x + 1;
12      int    z = x / y;
13      z = x;
14      System.out.println(y / z);
15      x  =  foo(y, z);
16      System.out.println(x            /   x);
17   }
18 }
```

ChatGPT Poe

The value of the variable z used at line 14 is 0.

Let's go through the code step by step:

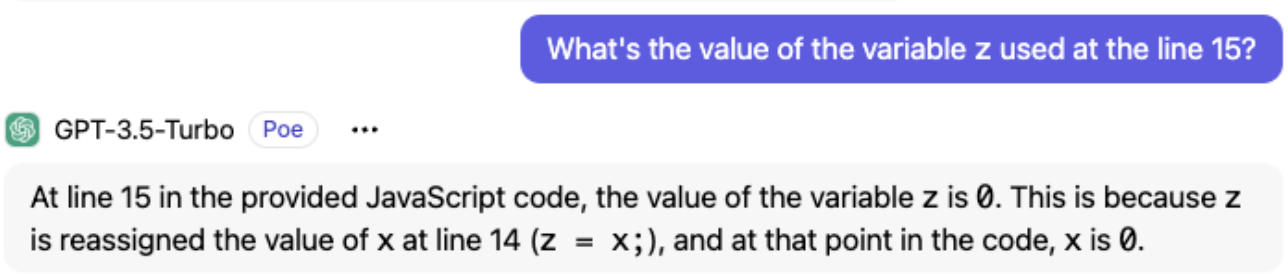
1. `int x = 1;` - Initializes x with the value 1.
2. `int y = 10 / x;` - Calculates y as the result of dividing 10 by the value of x, which is 1. So y is assigned the value 10.
3. `x = 0;` - Updates the value of x to 0.
4. `y = x + 1;` - Assigns y as the result of adding the value of x (0) and 1. So y is assigned the value 1.
5. `int z = x / y;` - Calculates z as the result of dividing the value of x (0) by the value of y (1). So z is assigned the value 0.
6. `z = x;` - Updates the value of z to the value of x, which is 0.
7. `System.out.println(y / z);` - Prints the result of dividing y (1) by z (0), which is not defined and would result in an exception.
8. `x = foo(y, z);` - Calls the `foo` method with arguments y (1) and z (0). The `foo` method divides a by b only if the absolute value of b is greater than 1. Since b is 0, the division is not performed, and the `foo` method returns 0. So x is assigned the value 0.
9. `System.out.println(x / x);` - Prints the result of dividing x (0) by x (0), which would result in an exception since the division by zero is not defined.

Therefore, the value of z used at line 14 is 0.

The Role of LLMs: Code Interpreter

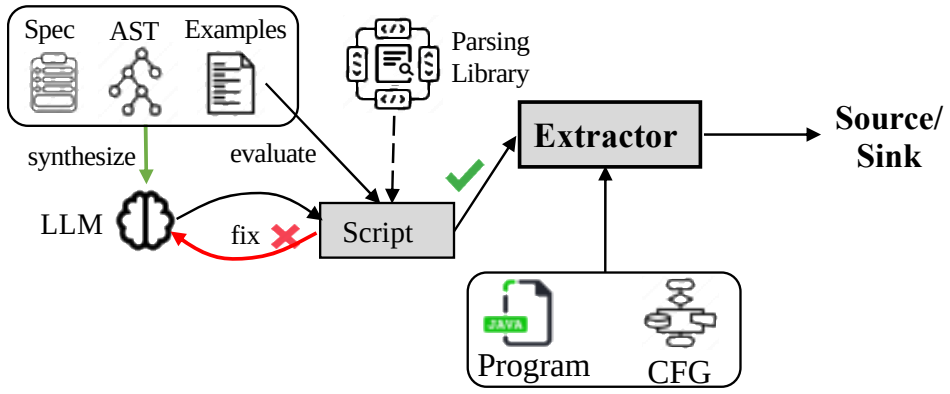
- Analyze programs in multiple languages

```
1. function foo(a, b) {
2.   if (Math.abs(b) > 1) {
3.     console.log(a / b);
4.   }
5.   return b;
6. }
7.
8. function main() {
9.   let x = 1;
10.   let y = 10 / x;
11.   x = 0;
12.   y = x + 1;
13.   let z = x / y;
14.   z = x;
15.   console.log(y / z);
16.   x = foo(y, z);
17.   console.log(x / x);
18. }
19.
20. main();
```

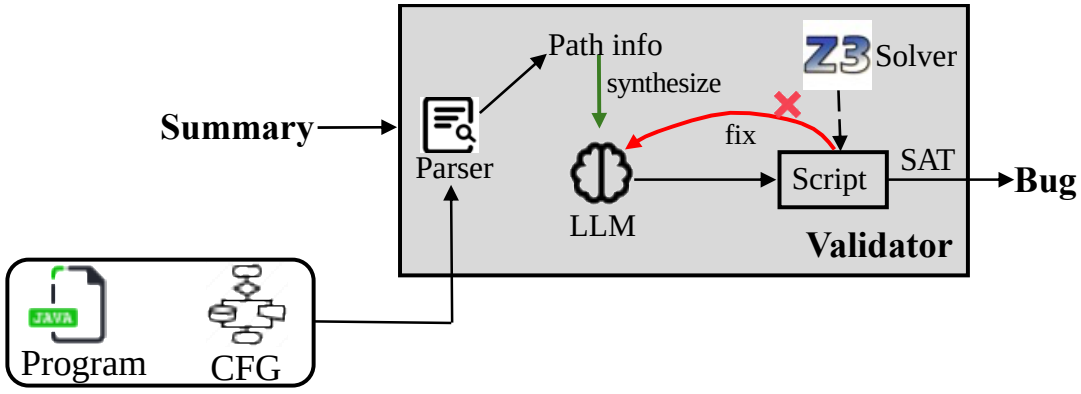


The Role of LLMs: Program Synthesizer

- Apply external tools autonomously
  - Reduce original task to domain-specific program synthesis



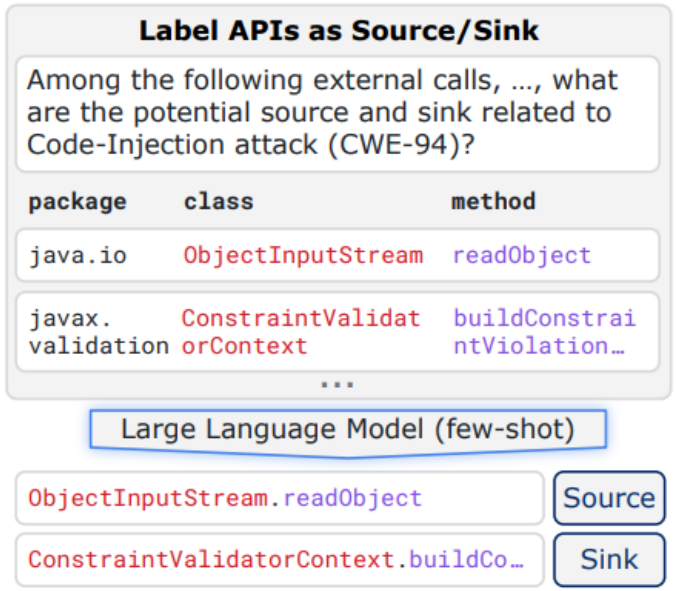
Source/sink Extraction

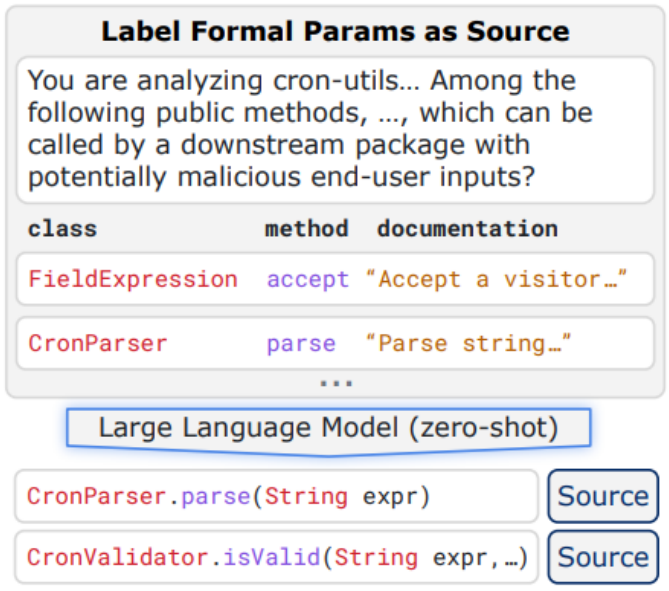


Path feasibility validation

Another Role of LLMs: Knowledge Base

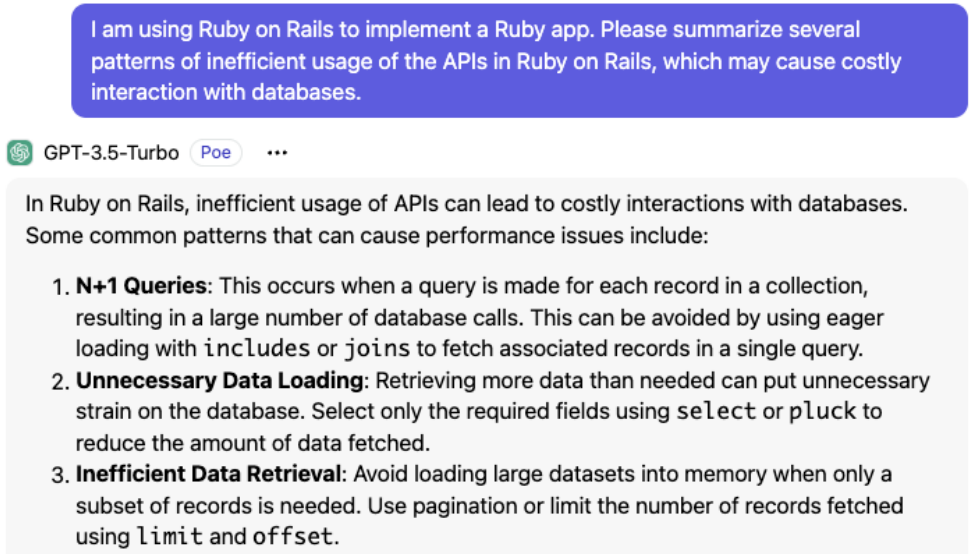
➢Enhance existing static analyzers with more library specifications

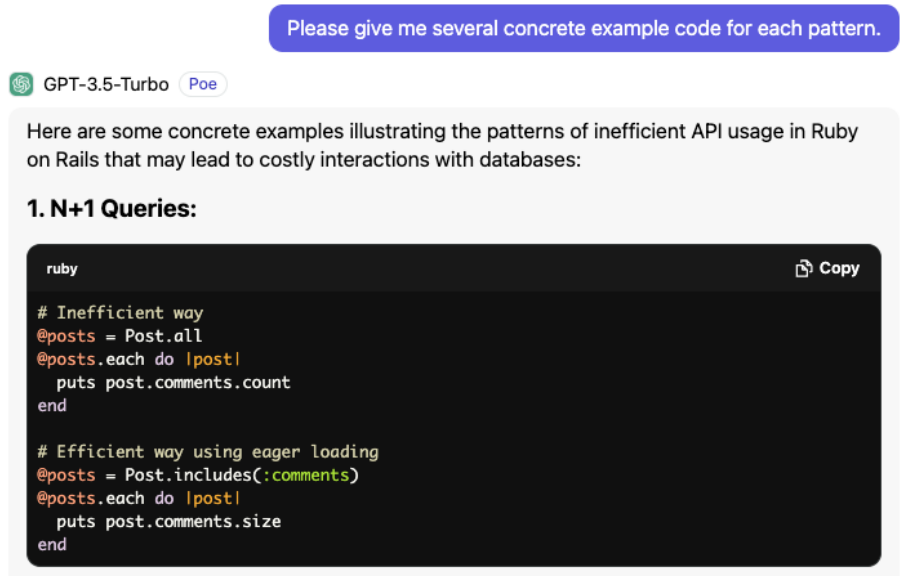




Another Role of LLMs: Knowledge Base

- Detect hard-to-formulated bugs
  - E.g., performance bugs caused by library function usage





Neuro-Symbolic Static Analysis

- Left half brain: Rigorous reasoning
  - Traditional program analyzers
  - Other external tools: SMT solver, parser
- Right half brain: Intuitive thinking
  - Different roles of LLMs
    - Code interpreter
    - Program synthesizer
    - Knowledge base
    - Etc

Large Language Model vs Large Reasoning Model

- Large Reasoning Models (LRMs), e.g., OpenAI o1, target general reasoning tasks
- Example: Analyzing intra-file taint flows in TaintBench

Metrics	LLMDFA	GPT-3.5	GPT-4	GPT-4o-mini	OpenAI o1
Precision	66.27%	40.00%	63.93%	46.97%	62.03%
Recall	78.57%	22.86%	55.71%	44.29%	70%
F1 Score	0.72	0.29	0.60	0.46	0.66

Future Directions

- Domain-specific Reasoning
  - LRMs may not effectively solve all the domain-specific problems
  - Performance concerns: Time/Token costs
  - Discover more roles of LLMs
  - Planning + Reinforcement learning

- Hallucination Mitigation
  - New prompting strategies
  - Tool using
  - Semantic alignment

Q&A

- Chengpeng Wang
- Email: wang6590@purdue.edu
- Homepage: https://chengpeng-wang.github.io/

LLM-PLSE-paper

LLMDFA