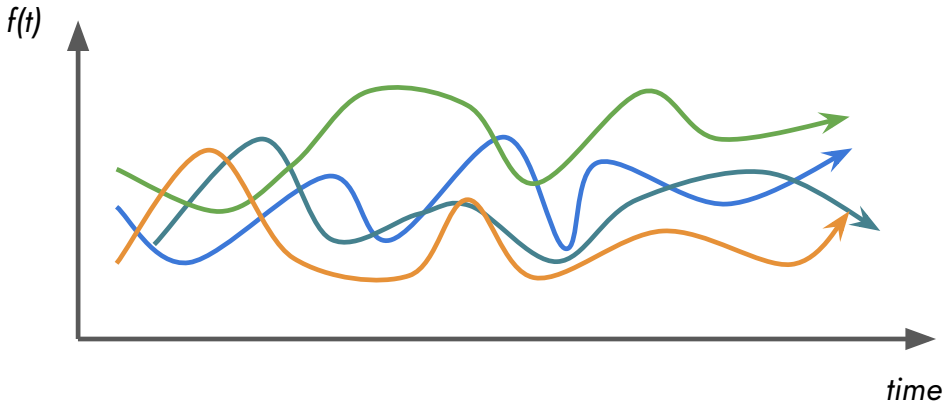
Hallucination-Resilient LLM-Driven Sound and Tunable Static Analysis

Guannan Wei, Zhuo Zhang, Caterina Urban

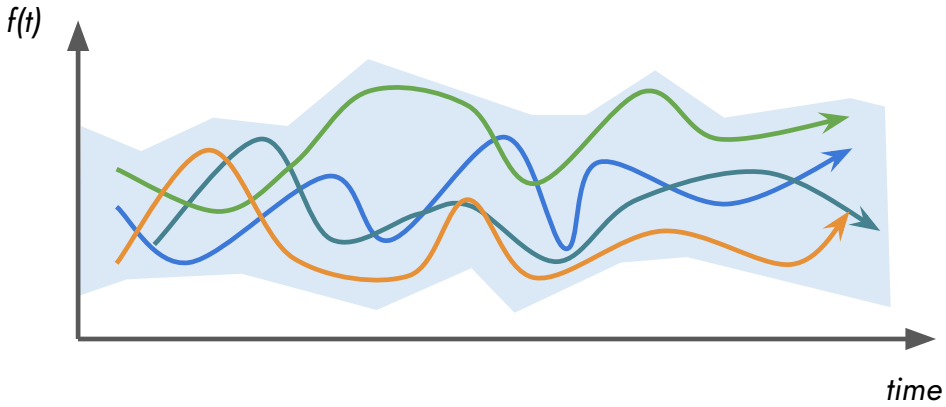
LMPL @ SPLASH/ICFP 2025, Singapore

What is Static Analysis?



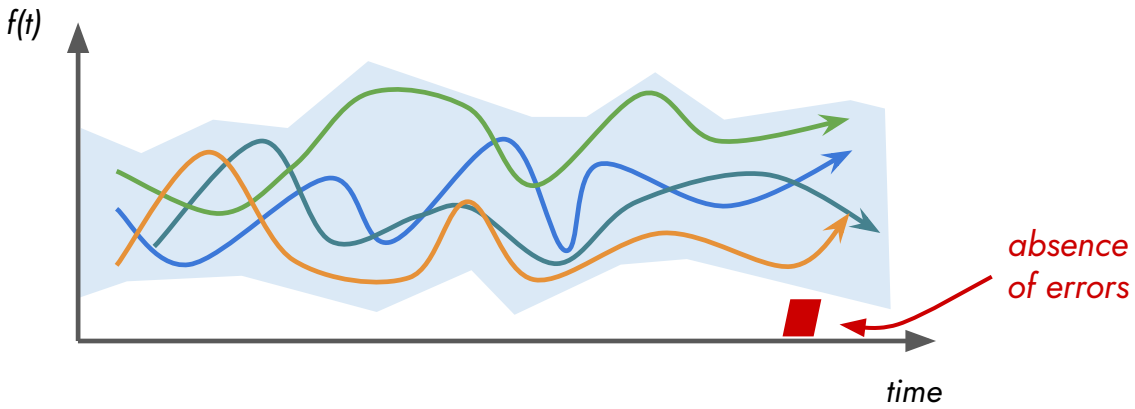
concrete traces

What is Static Analysis?



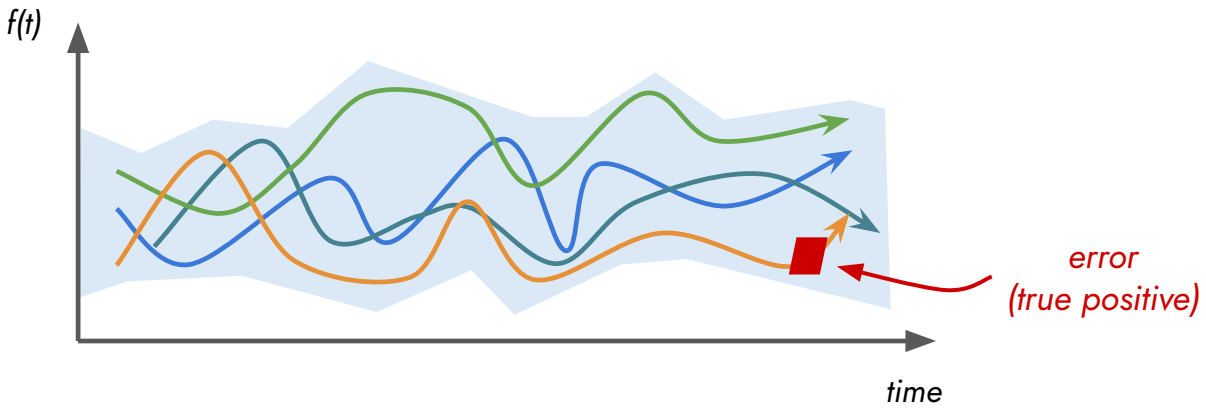
soundness:
over-approximations

What is Static Analysis?



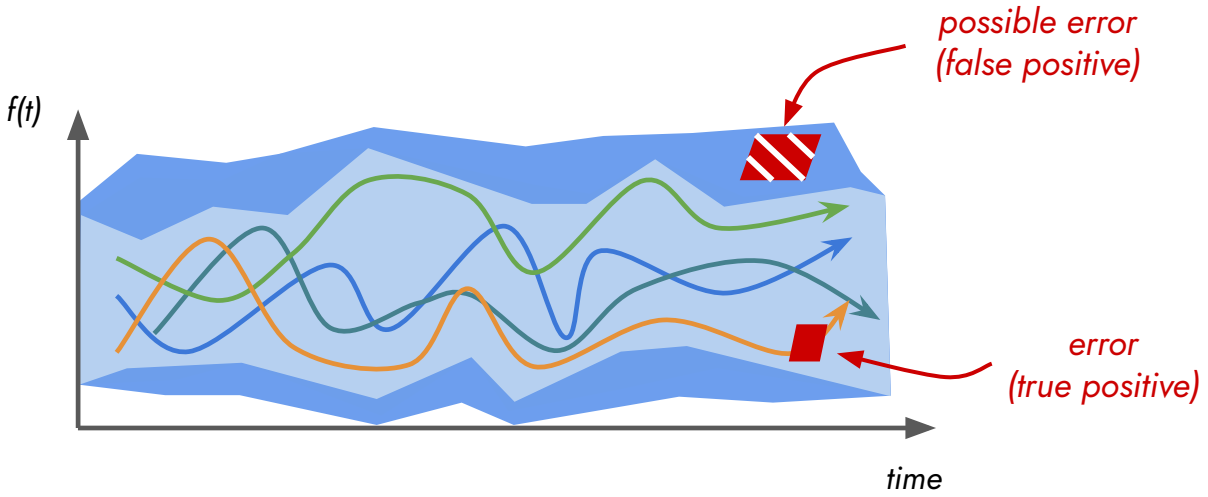
soundness:
over-approximations

What is Static Analysis?



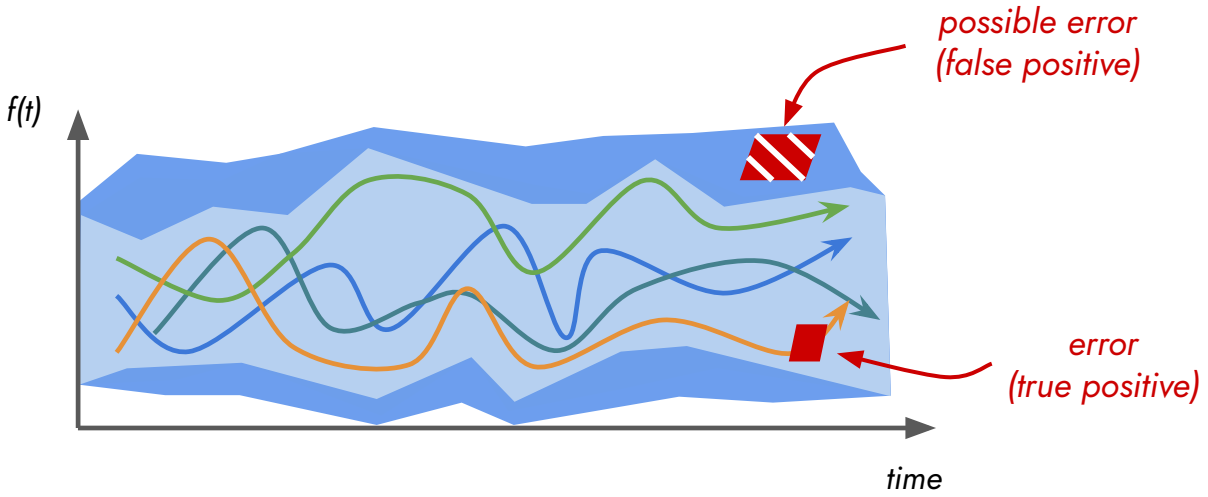
soundness:
over-approximations

What is Static Analysis?



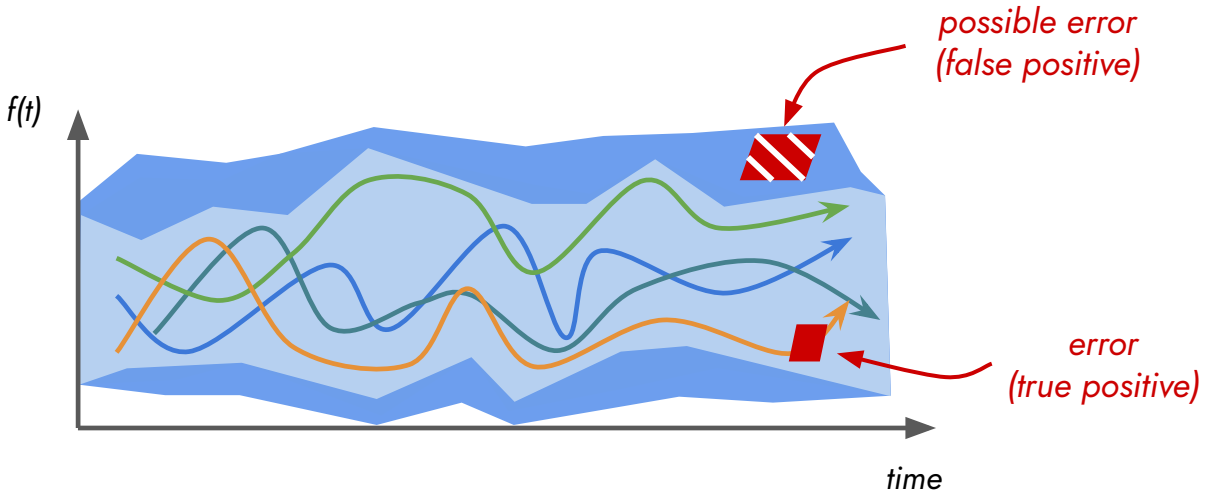
coarser-grained
over-approximations

What is Static Analysis?



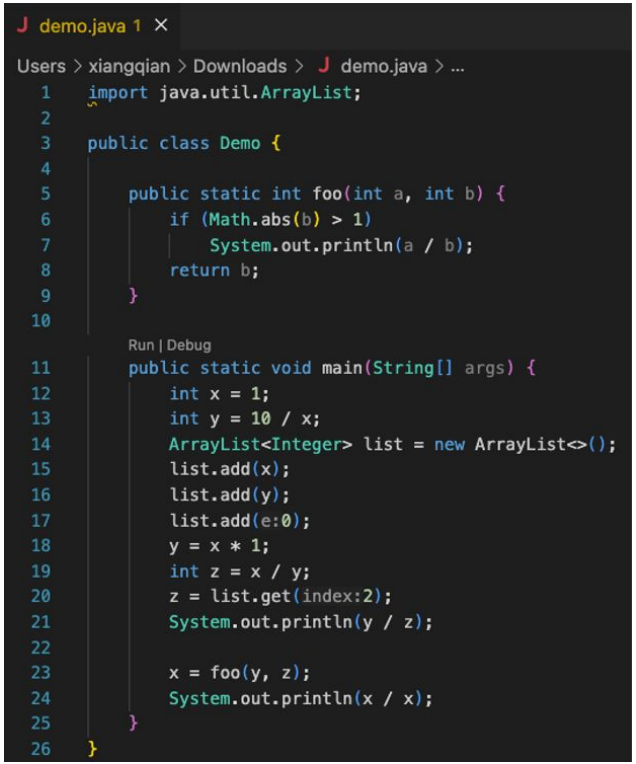
Goals ***Soundness***: *detect all true errors*
Precision: *reduce false errors*

What is Static Analysis?



So, you want to use LLMs for static analysis, but how?

Prompting LLMs



Claude-3.5-Sonnet

Prompting LLMs

demo.java 1

Users > xiangqian > Downloads > demo.java > ...

```java
import java.util.ArrayList;

public class Demo {

    public static int foo(int a, int b) {
        if (Math.abs(b) > 1)
            System.out.println(a / b);
        return b;
    }

    Run | Debug
    public static void main(String[] args) {
        int x = 1;
        int y = 10 / x;
        ArrayList<Integer> list = new ArrayList<>();
        list.add(x);
        list.add(y);
        list.add(e:0);
        y = x * 1;
        int z = x / y;
        z = list.get(index:2);
        System.out.println(y / z);

        x = foo(y, z);
        System.out.println(x / x);
    }
}
```

CHAT

Please mimic the execution of the program and detect divide-by-zero bugs.

demo.java:1-26

prompt

Claude-3.5-Sonnet

Prompting LLMs

demo.java 1

Users > xiangqian > Downloads > demo.java > ...

```java
import java.util.ArrayList;

public class Demo {

    public static int foo(int a, int b) {
        if (Math.abs(b) > 1)
            System.out.println(a / b);
        return b;
    }

    Run | Debug
    public static void main(String[] args) {
        int x = 1;
        int y = 10 / x;
        ArrayList<Integer> list = new ArrayList<>();
        list.add(x);
        list.add(y);
        list.add(e:0);
        y = x * 1;
        int z = x / y;
        z = list.get(index:2);
        System.out.println(y / z);

        x = foo(y, z);
        System.out.println(x / x);
    }
}
```

CHAT

Please mimic the execution of the program and detect divide-by-zero bugs.

demo.java:1-26

prompt

GitHub Copilot

Used 1 reference

I'll analyze the code execution step by step and identify potential divide-by-zero bugs.

Analysis

1. In main() :
   - x = 1
   - y = 10 / x → y = 10 (safe, x = 1)
   - Create ArrayList and add: [1, 10, 0]
   - y = x * 1 → y = 1
   - z = x / y → z = 1 (safe, y = 1)
   - z = list.get(2) → z = 0
   - System.out.println(y / z) → **DIVIDE BY ZERO ERROR!** (y = 1, z = 0)
   - Program would crash here, but if it continued:
   - x = foo(y, z) with y = 1, z = 0
   - Inside foo() : doesn't print since Math.abs(0) ≤ 1
   - Returns z which is 0
   - System.out.println(x / x) → **DIVIDE BY ZERO ERROR!** (x = 0)

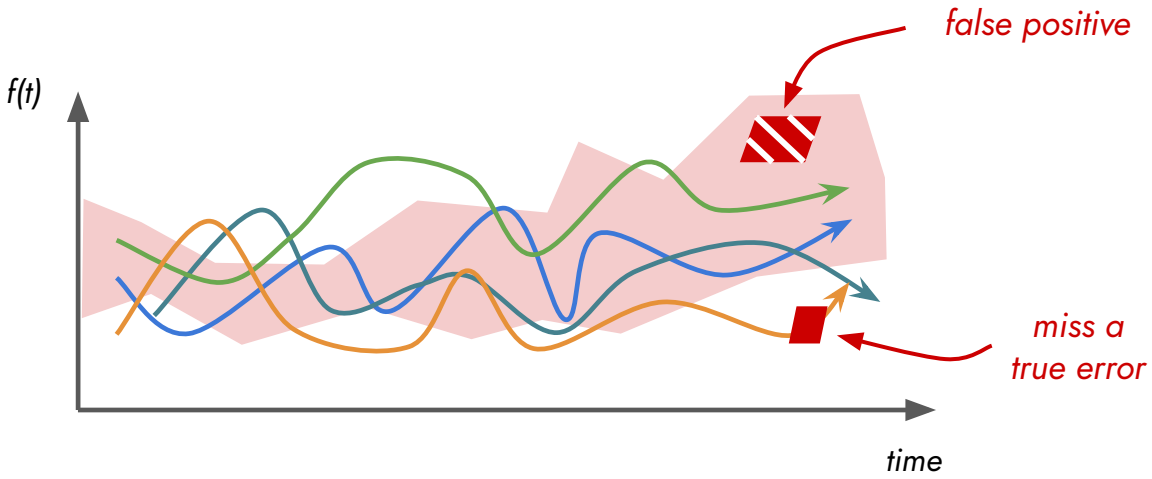
Divide-by-Zero Bugs Found:

1. Line 21: System.out.println(y / z) - when z = 0
2. Line 24: System.out.println(x / x) - when x = 0

reasoning & analysis result

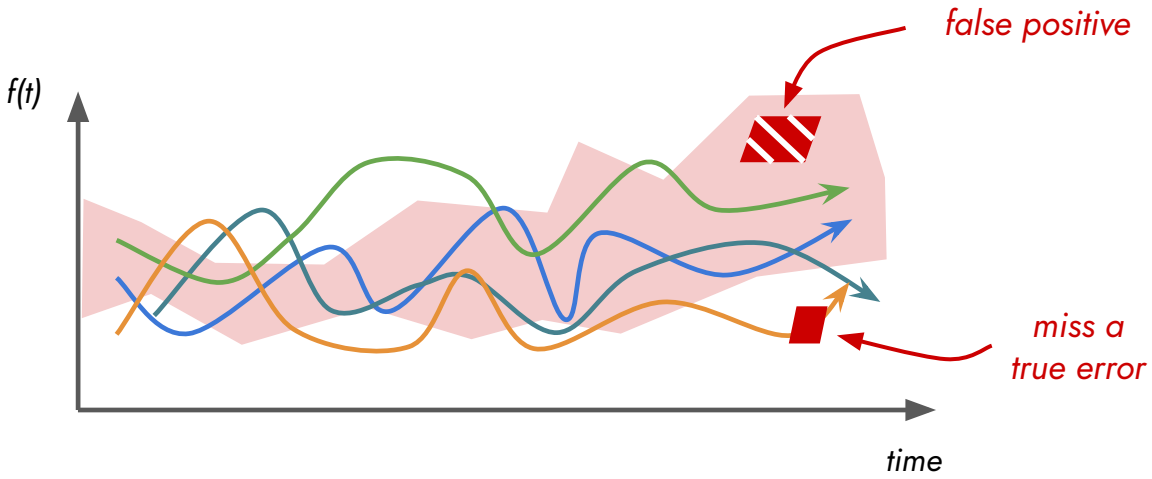
Claude-3.5-Sonnet

Can we trust LLM-reported analysis results?



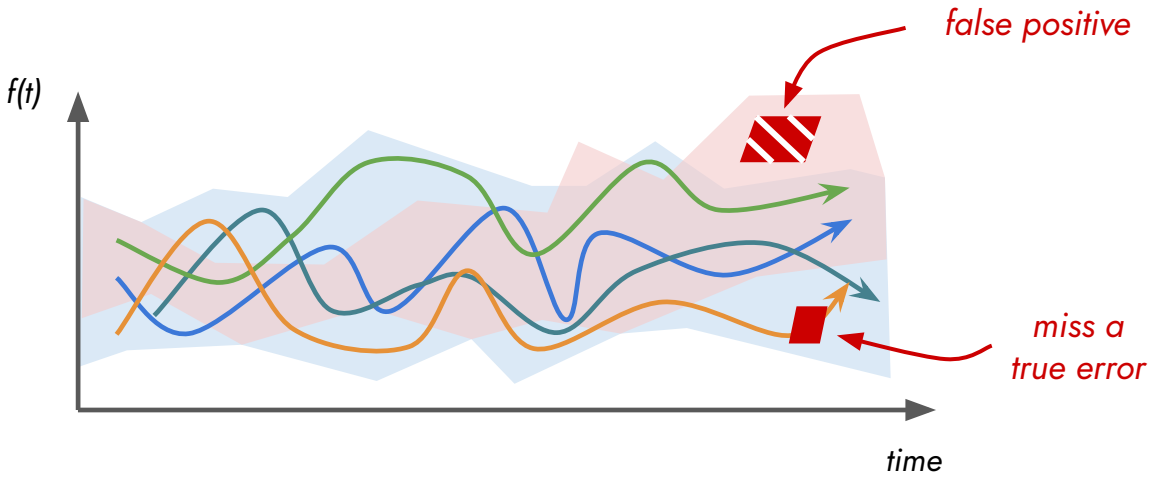
LLMs are untrustworthy because of hallucination.

Can we trust LLM-reported analysis results?



Can we validate the analysis results from LLMs?

Can we trust LLM-reported analysis results?

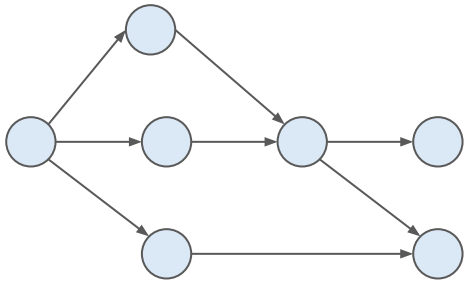


Can we validate the analysis results from LLMs?
Yes, but this becomes a static analysis problem…

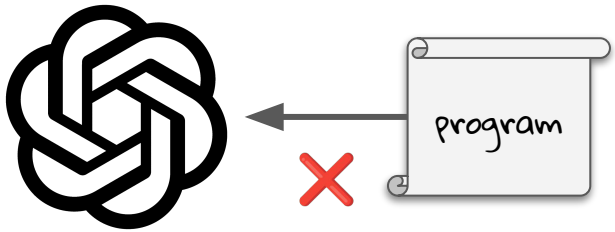
Our Idea: LLMs as Meta-Analysis

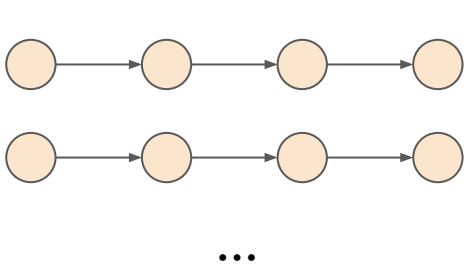
Instead of analyzing target programs, use LLMs to "analyze" the analysis!

analyzer



abstract states

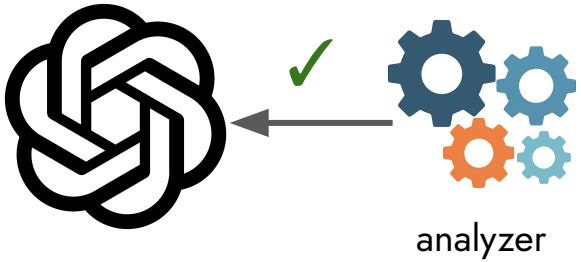


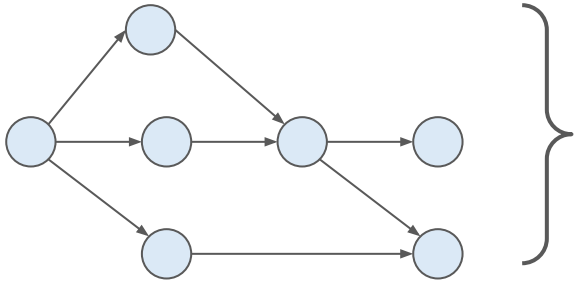


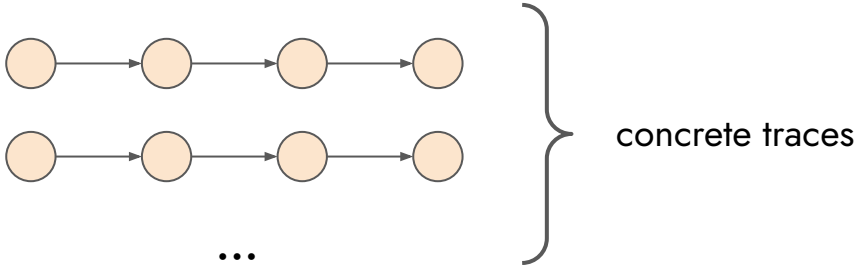
concrete traces

Our Idea: LLMs as Meta-Analysis

Instead of analyzing target programs, use LLMs to "analyze" the analysis!







Our Idea: LLMs as Meta-Analysis

Recipe:

- **Identify precision-critical decisions in the static analysis**
  - Any decision does not hinder soundness,
  - but only impacts precision
- During the analysis, use LLMs as the decision maker
  - Prompt describes the process/implementation of static analysis
  - Prompt contains current abstract state and other (meta-) information
  - Prompt instructs to decide for better precision

Our Idea: LLMs as Meta-Analysis

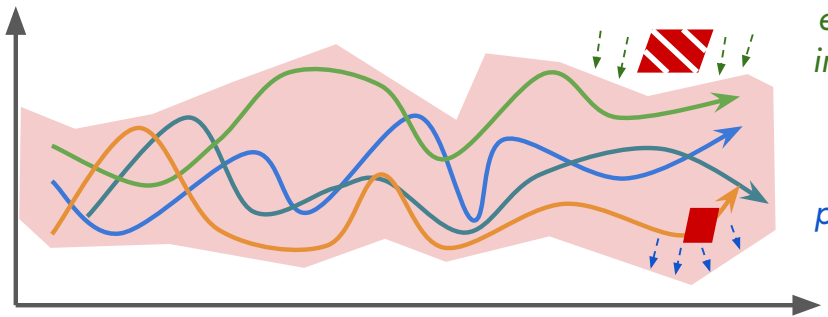
Recipe:

- Identify precision-critical decisions in the static analysis
  - Any decision does not hinder soundness,
  - but only impacts precision
- **During the analysis, use LLMs as the decision maker**
  - Prompt describes the process/implementation of static analysis
  - Prompt contains current *abstract* state and other (meta-) information
  - Prompt instructs to decide for better precision

Our Idea: LLMs as Meta-Analysis

Recipe:

- **Identify precision-critical decisions in the static analysis**
  - Any outcome does not hinder soundness,
  - but only impacts on precision
- **During the analysis, use LLMs as the decision maker**
  - Prompt describes the process/implementation of static analysis
  - Prompt contains current abstract state and other (meta-) information
  - Prompt instructs to decide for precision



effective prompts improve precision

preserve soundness by construction

A Case of Higher-Order Control-Flow Analysis

- Control flow analysis:
  - determines the function to be called at each call-site
  - undecidable problem once data- and control-flow are entangled
    (e.g. when with first-class functions in Scheme, dynamic dispatch in Java, indirect call with pointers in C, etc.)

- Abstracting abstract machine (AAM) framework [ICFP '10]
  - analysis is defined as an abstraction of a concrete abstract machine (e.g. CEK machine)

$$\widehat{\Sigma} \in \widehat{\mathrm{State}} \triangleq (\mathrm{Expr} + \widehat{\mathrm{Value}}) \times \widehat{\mathrm{Env}} \times \widehat{\mathrm{BStore}} \times \widehat{\mathrm{KStore}} \times \widehat{\mathrm{Kont}} \times \widehat{\mathrm{Time}}$$

  - result is an over-approximated transition graph of abstract states

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 1]** Identify precision-critical decisions in the static analysis

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 1]** Identify precision-critical decisions in the static analysis

$$
\begin{aligned}
tick &: \widehat{\mathsf{State}} \to \widehat{\mathsf{Time}} \\
alloc_v &: \widehat{\mathsf{State}} \times \mathsf{Var} \times \widehat{\mathsf{Time}} \to \widehat{\mathsf{BAddr}} \\
alloc_\kappa &: \widehat{\mathsf{State}} \times \mathsf{Expr} \times \widehat{\mathsf{Env}} \times \widehat{\mathsf{BStore}} \times \widehat{\mathsf{Time}} \to \widehat{\mathsf{KAddr}}
\end{aligned}
$$

LLM as abstract address allocator

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 1]** Identify precision-critical decisions in the static analysis

$$
\begin{aligned}
tick &: \widehat{\mathrm{State}} \to \widehat{\mathrm{Time}} \\
alloc_v &: \widehat{\mathrm{State}} \times \mathrm{Var} \times \widehat{\mathrm{Time}} \to \widehat{\mathrm{BAddr}} \\
alloc_\kappa &: \widehat{\mathrm{State}} \times \mathrm{Expr} \times \widehat{\mathrm{Env}} \times \widehat{\mathrm{BStore}} \times \widehat{\mathrm{Time}} \to \widehat{\mathrm{KAddr}}
\end{aligned}
$$

LLM as abstract address allocator

Soundness-safe:
Any allocation strategy from a finite set is sound (a Posteriori Soundness Theorem)

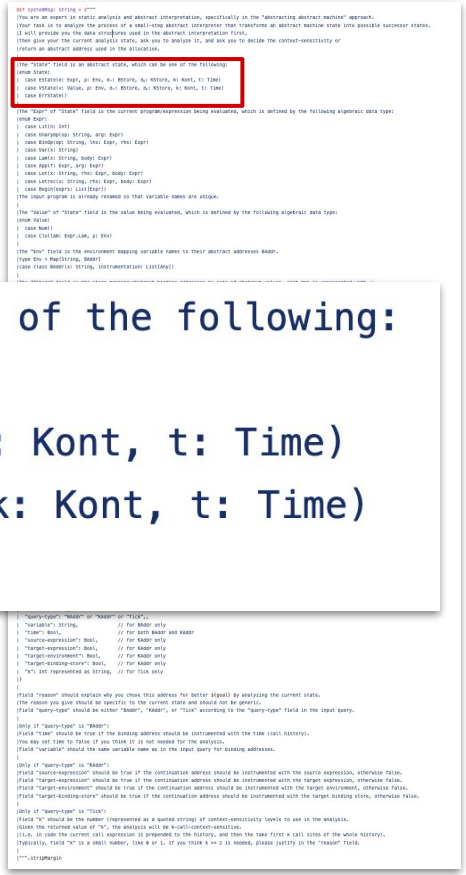
Precision-critical:
Addresses decide the degree of approximation (e.g. context-sensitivity, …)

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 2]** During the analysis, use LLMs as the decision maker.

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 2]** During the analysis, use LLMs as the decision maker.



```
|The "State" field is an abstract state, which can be one of the following:
|enum State:
|  case EState(e: Expr, ρ: Env, σᵥ: BStore, σₖ: KStore, k: Kont, t: Time)
|  case VState(v: Value, ρ: Env, σᵥ: BStore, σₖ: KStore, k: Kont, t: Time)
|  case ErrState()
```

Describe data structures used in the analyzer

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 2]** During the analysis, use LLMs as the decision maker.

```
|Your input is a JSON object with the following schema:
|{
|  "state": State,
|  "query-type": "BAddr" or "KAddr" or "Tick",
|  "variable": String,                    // for BAddr only
|  "time": Time,                          // for BAddr, KAddr, and "Tick"
|  "source-expression": Expr,             // for KAddr only
|  "target-expression": Expr,             // for KAddr only
|  "target-environment": Env,             // for KAddr only
|  "target-binding-store": BStore,        // for KAddr only
|}
```

Describe input JSON format for each query.

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 2]** During the analysis, use LLMs as the decision maker.

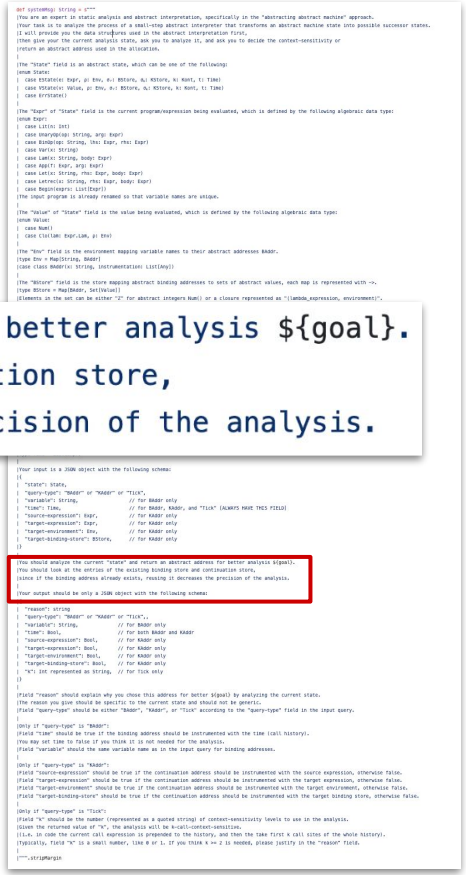
```
|Your output should be only a JSON object with the following schema:
|{
|  "reason": string
|  "query-type": "BAddr" or "KAddr" or "Tick",,
|  "variable": String,                // for BAddr only
|  "time": Bool,                      // for both BAddr and KAddr
|  "source-expression": Bool,         // for KAddr only
|  "target-expression": Bool,         // for KAddr only
|  "target-environment": Bool,        // for KAddr only
|  "target-binding-store": Bool,      // for KAddr only
|  "k": Int represented as String,  // for Tick only
|}
```

Describe output JSON format for each query.

A Case of Higher-Order Control-Flow Analysis

- **[Recipe step 2]** During the analysis, use LLMs as the decision maker.

```
|You should analyze the current "state" and return an abstract address for better analysis ${goal}.
|You should look at the entries of the existing binding store and continuation store,
|since if the binding address already exists, reusing it decreases the precision of the analysis.
```



Explain the logic behind allocation and ask for reasoning with the abstract states.

A Case of Higher-Order Control-Flow Analysis

- A prototype "LLMAAM" implemented in Scala
  `https://github.com/Kraks/llmaam`
- Preliminary evaluation of micro-benchmarks with GPT-4o-mini

	idid		kcfa2		mj09		loop2	
	#N	#E	#N	#E	#N	#E	#N	#E
0CFA+P4F	42	41	307	306	303	302	110	108
0CFA+AAC	33	32	203	202	**179**	**175**	110	108
0CFA+SRC	42	41	246	245	647	644	149	146
1CFA+P4F	33	32	TO	TO	1437	1407	108	106
1CFA+AAC	33	32	336	324	331	323	108	106
1CFA+SRC	42	41	12189	12117	6849	6834	108	106
llmaam	**33**	**32**	**133**	**131**	199	202	**108**	**106**
	33	32	153	150	268	259	108	106
	33	32	272	265	281	273	108	106

#edges and #states as approximation of precision -- smaller is better

Use LLMs to “Analyze” the Analysis!

Recipe for sound, precise, tunable static analysis driven by LLMs:

- Identify precision-critical decisions in static analysis
- During the analysis, use LLMs as meta-analysis and the decision maker

Future opportunities – Analyses:

- Control-flow sensitivity
- Numeric analysis
- Termination analysis
- …

Future opportunities – Optimizations:

- Prompt optimizations
- Fine-tuning
- Reinforcement learning
- …