Software Bug Localization with Markov Logic

Sai Zhang, Congle Zhang
University of Washington

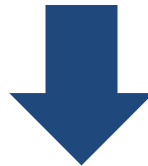
Presented by **Todd Schiller**

Computer Science & Engineering
UNIVERSITY of WASHINGTON

Software bug localization: finding the likely buggy code fragments

A **software** system
(source code)

Some **observations**
(test results, code coverage, bug history, code dependencies, etc.)



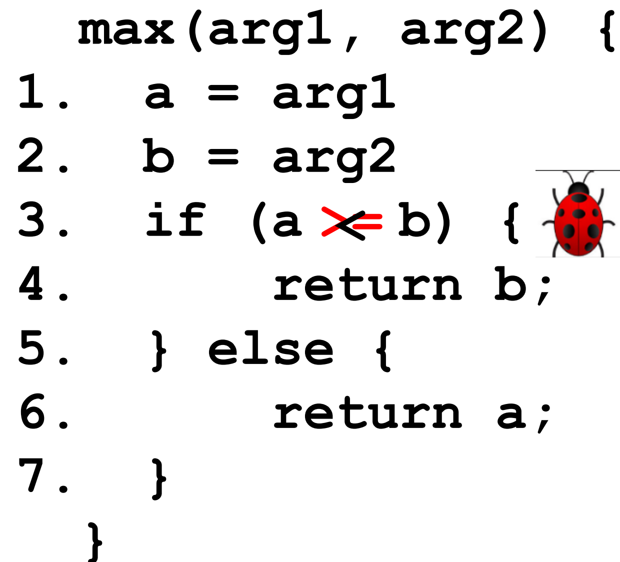
A ranked list of likely buggy code fragments

An example bug localization technique (Tarantula [Jones'03])

- **Input**: a program + passing tests + failing tests
- **Output**: a list of buggy statements

```
3. if (a >= b) {
4. return b;
1. a = arg1
2. b = arg2
5. } else {
6.    return a;
```

Example:



Te

arg1 = 1
arg2 = 2

arg
arg

✘ ✘ ✓

Tarantula's ranking heuristic

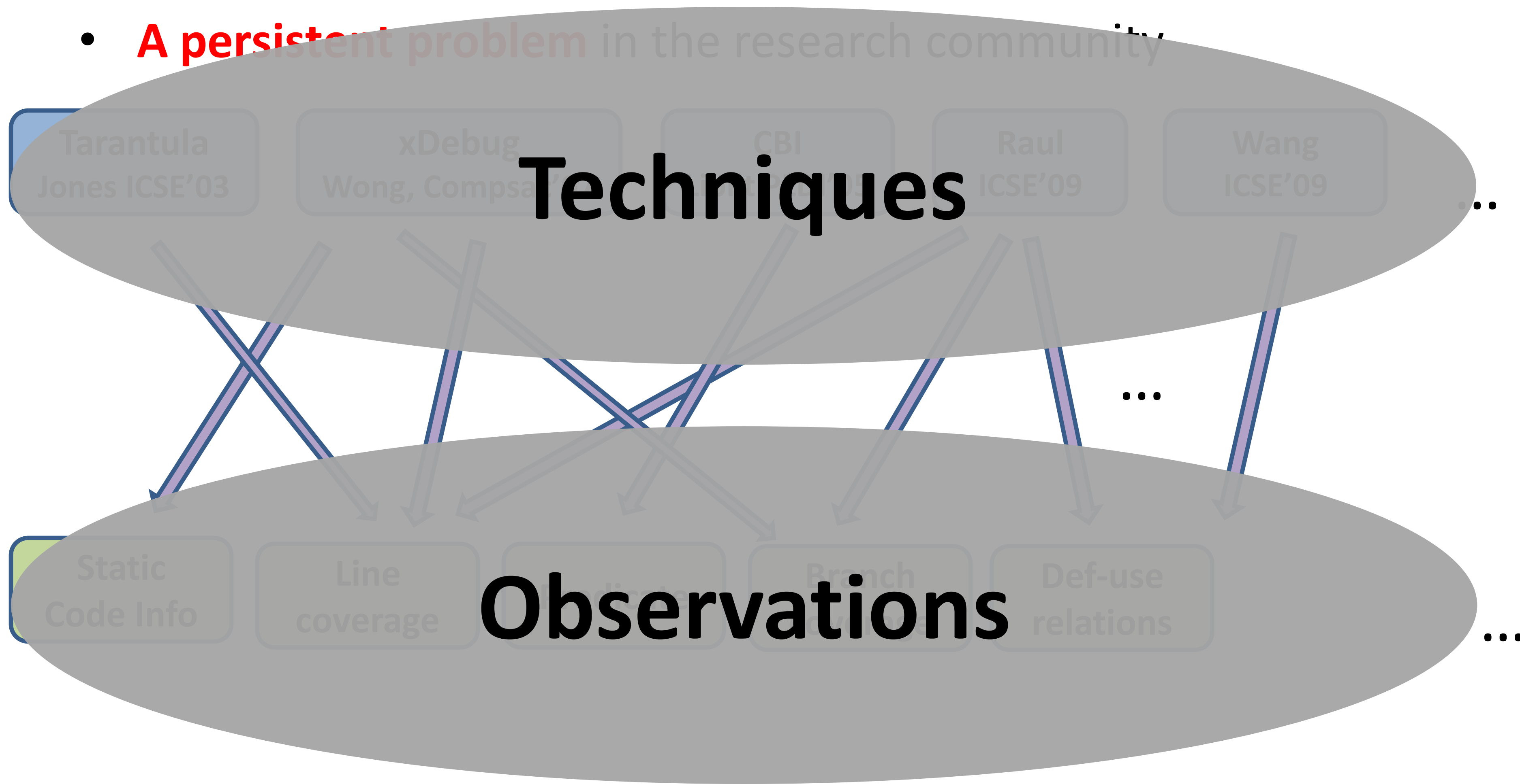
For a statement: s

$$\text{Suspiciousness(s)} = \frac{\%fail(s)}{\%fail(s) + \%pass(s)}$$

This heuristic is effective in practice [Jones'05]

Problem: existing techniques lack an interface layer

- Heuristics are **hand crafted**
- Techniques are often defined in an **ad-hoc** way
- **A persistent problem** in the research community

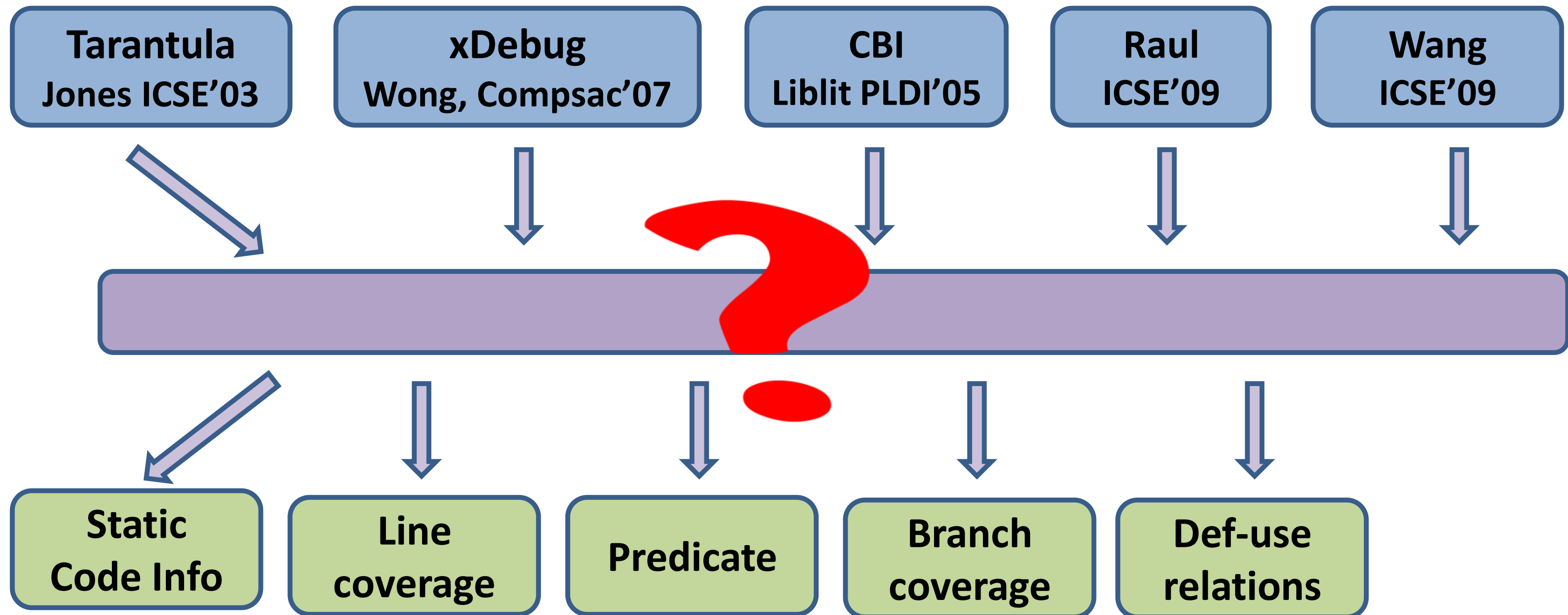


Adding an interface layer

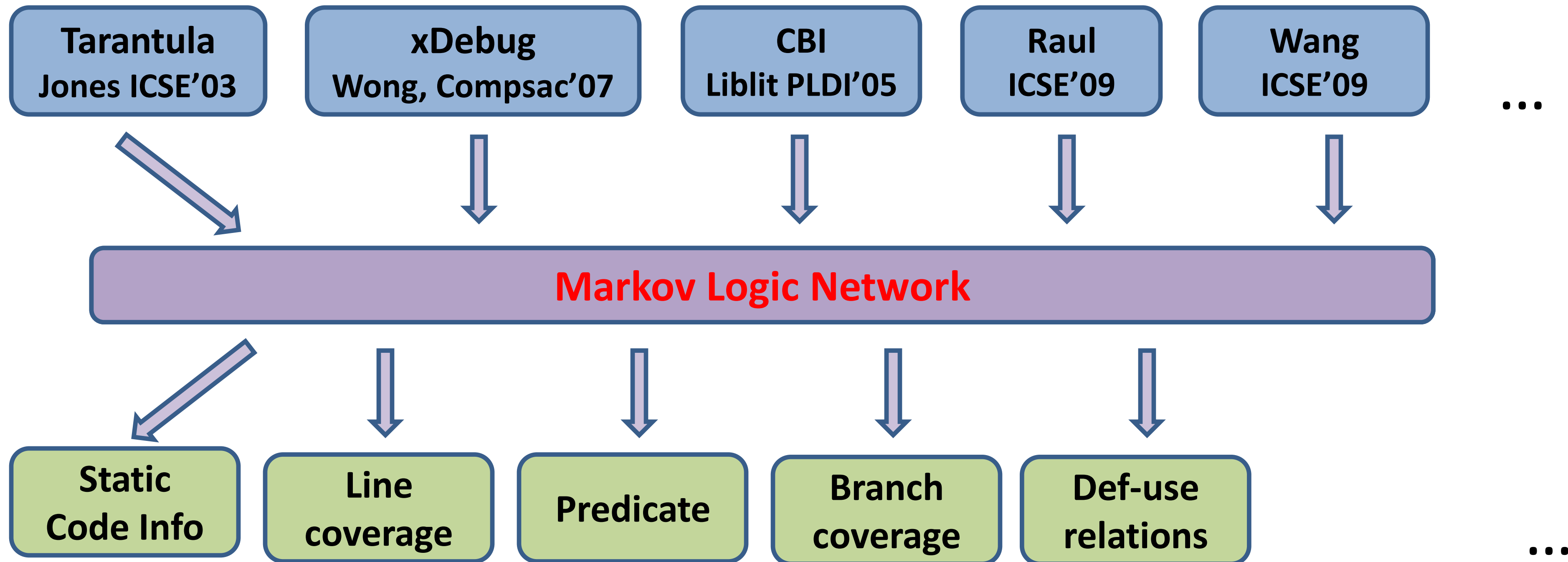
Why an interface layer?

- **Focus on key design insights**
- **Avoid “magic numbers “ in heuristics**
- **Fair basis for comparison**
- **Fast prototyping**

Who should be the interface layer?



Markov logic network as an interface layer



Why Markov Logic Network [Richardson'05]?

- Use first order logic to express key insights
  - E.g., estimate the likelihood of **`cancer(x)`** for people **`x`**

    <u>Example rules:</u>

```
smoke(x) => cancer(x)
smoke(x) ∧ frie...) => smoke(y)
...ds(x, ...ds(x, z)
```

smoke causes cancer

you will smoke if your friend...

friends of friends are friends

Why Markov Logic Network [Richardson'05]?

- Use first order logic to express key insights
  - E.g., estimate the likelihood of **`cancer(x)`** for people **`x`**

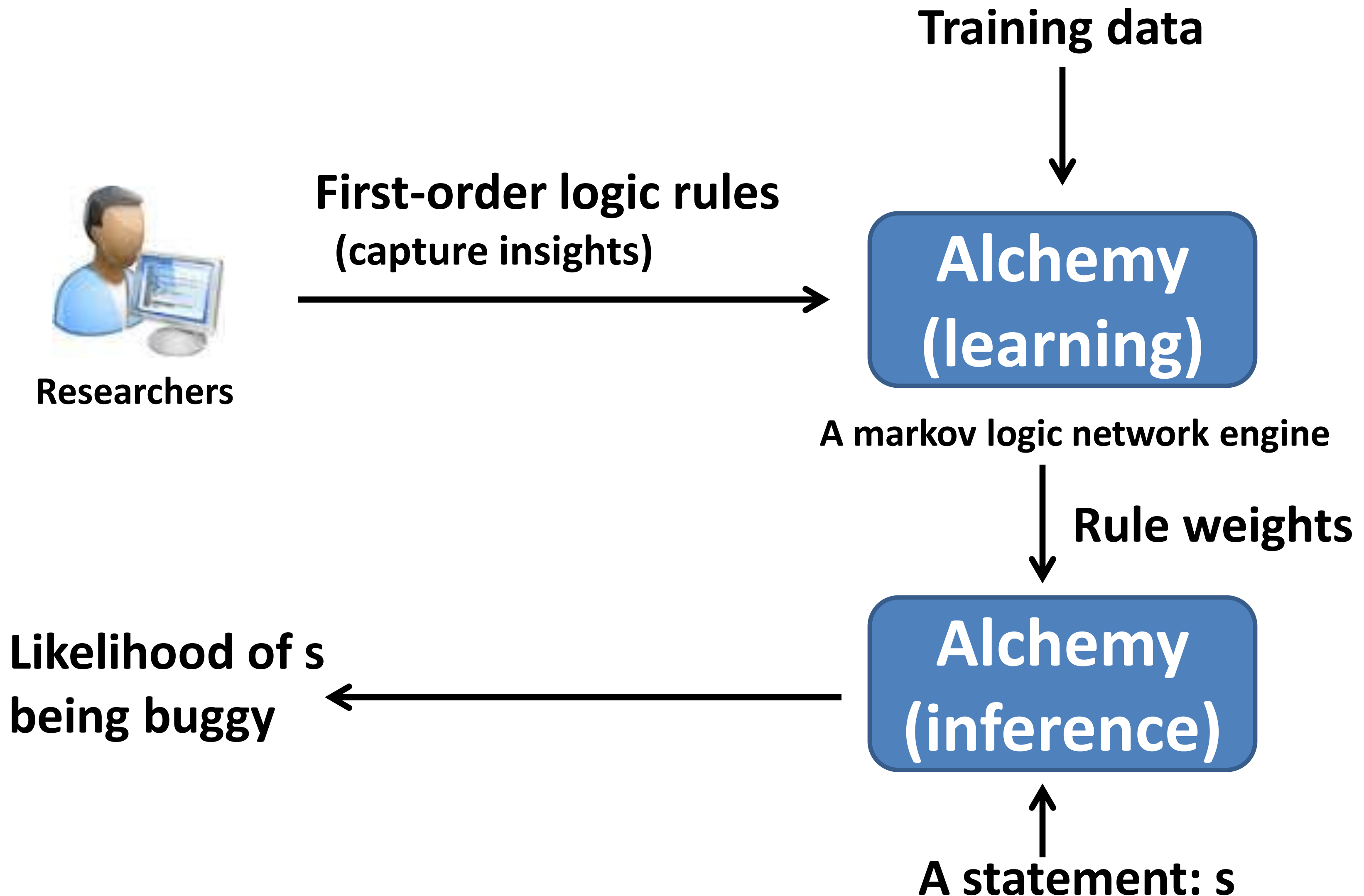
    Example rules:

w1	**`smoke(x) => cancer(x)`**
w2	**`smoke(x) ∧ friend(x,y) => smoke(y)`**
w3	**`friends(x, y) ∧ friends(y, z) => friends(x, z)`**

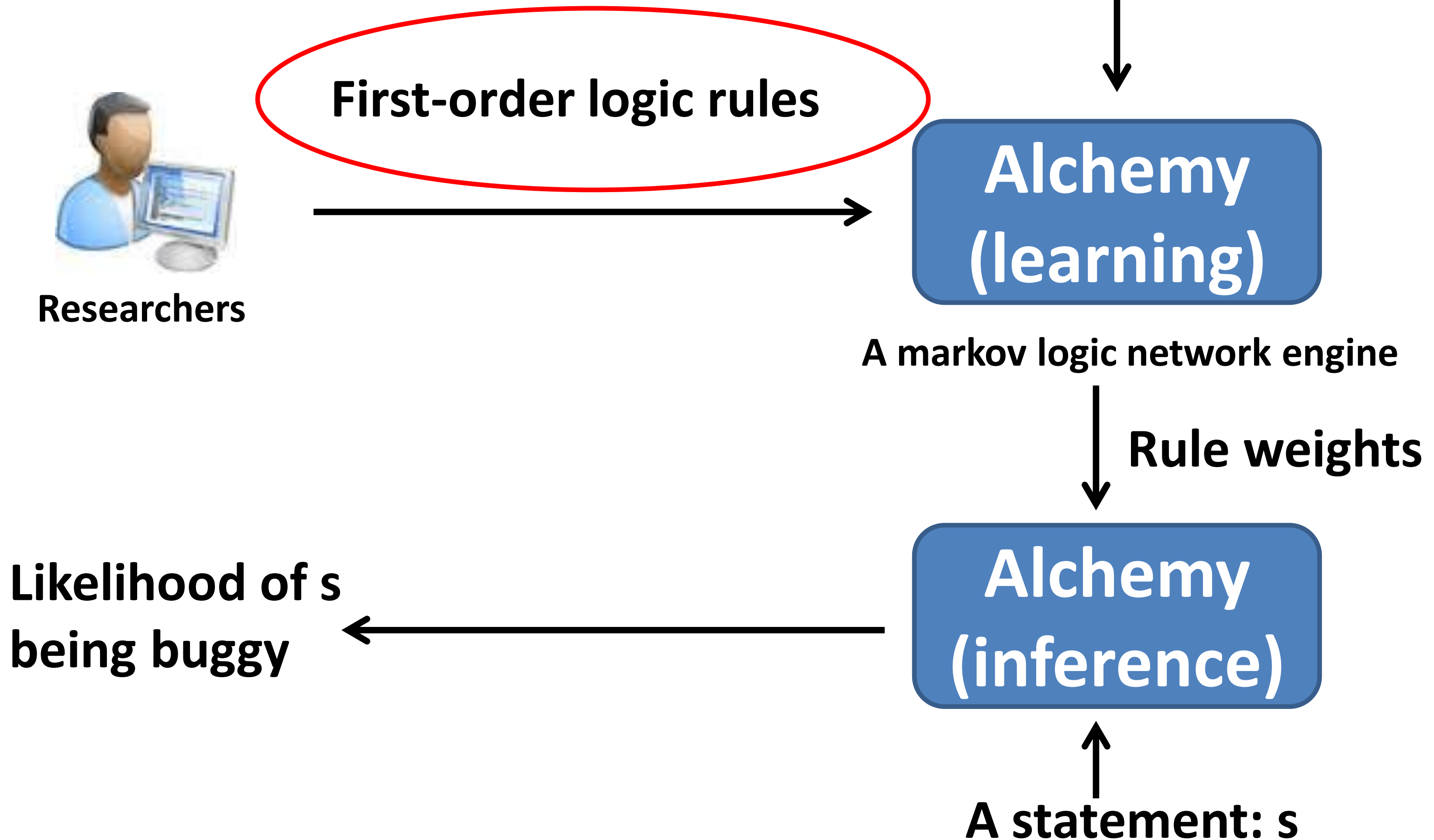
- Efficient weight learning and inference
  - Learning rule weights from training data
  - Estimate **`cancer(x)`** for a new data point

    (details omitted here)

Markov logic for bug localization



Markov logic for bug localization



Our prototype: MLNDebugger

- First-order rules

```
1. cover(test, s) ∧ fail(test) => buggy(s)
2. cover(test, s) ∧ pass(test) => ¬ buggy(s)
3. co
4. da
```

A statement covered by a passing test is not buggy

If a statement ... data flow dep... statem...

A statement that was buggy before is buggy

```
if(foo(x)) {
    bar();
}
```

Buggy!

Correct!

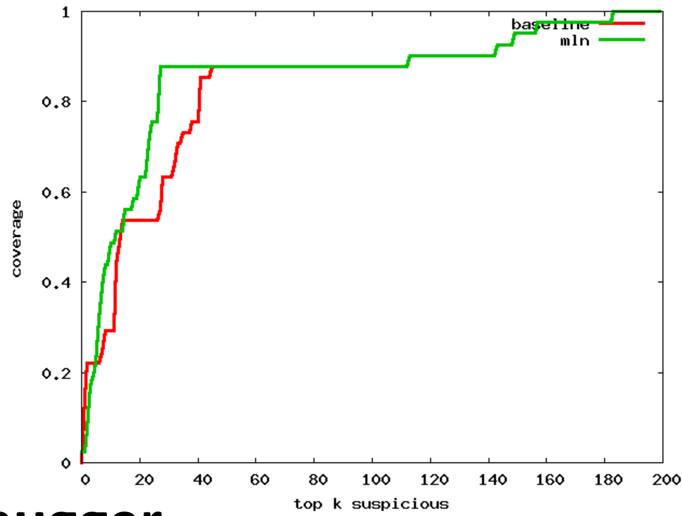
foo() Buggy!

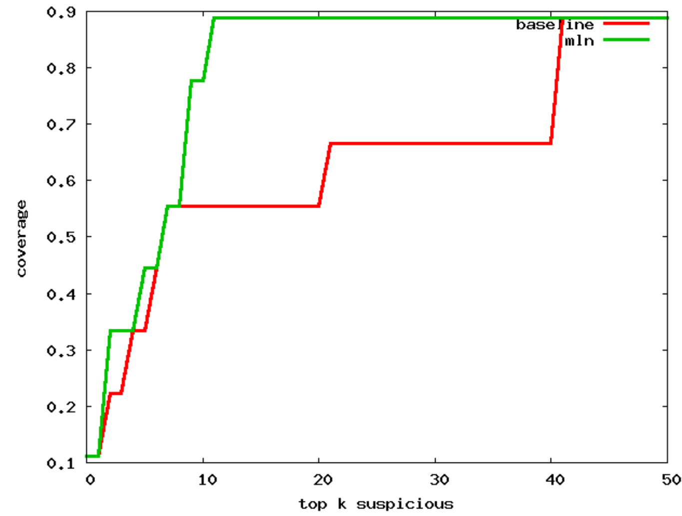
Correct!

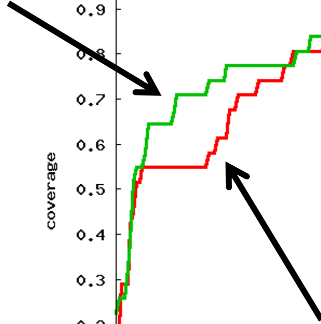
Evaluating MLNDebugger on 4 Siemens benchmarks

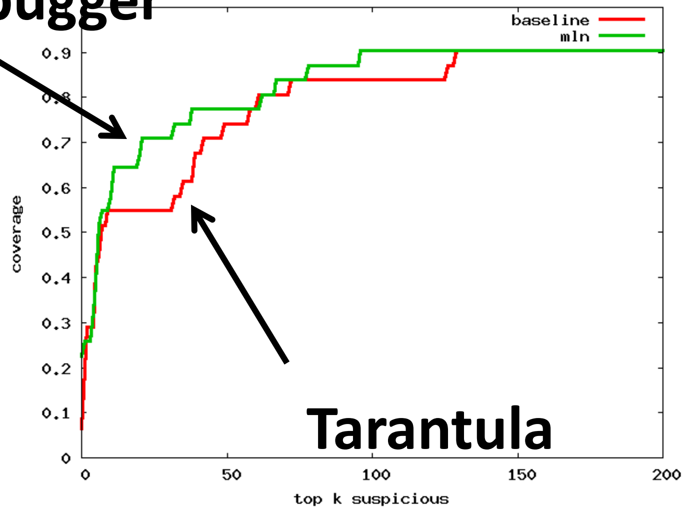
- 80+ seeded bugs
  - 2/3 as training set
  - 1/3 as testing set
- Measurement on the testing set
  - Return **top k suspicious** statements, check the percentage of **buggy ones** they can cover.
- Baseline: Tarantula [Jones' ICSE 2003]

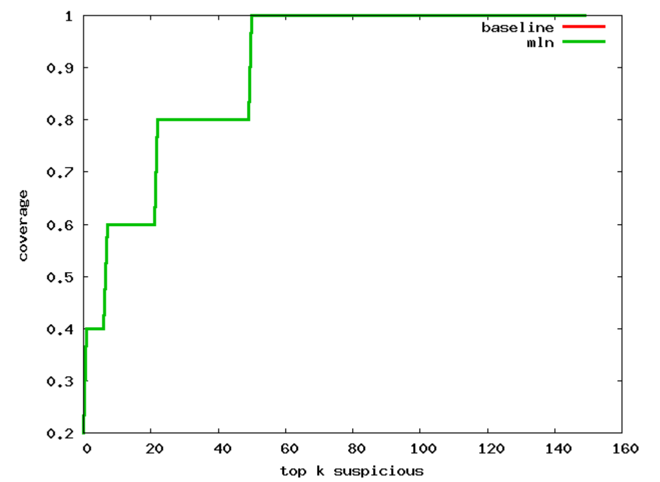
Experimental results











More in the paper…

- Formal definition
- Inference algorithms
- Implementation details
- Implications to the bug localization research

Contributions

- The first unified framework for automated debugging
  - Markov logic network as an interface layer: expressive, concise, and elegant
- A proof-of-concept new debugging technique using the framework
- An empirical study on 4 programs
  - 80+ versions, 8000+ tests
  - Outperform a well-known technique