Righting Software

James R. Larus, Thomas Ball, Manuvir Das, Robert DeLine,
Manuel Fähndrich, Jon Pincus, Sriram K. Rajamani, and
Ramanathan Venkatapathy
IEEE Software, May-June 2004

Summary by Prof. Thomas LaToza

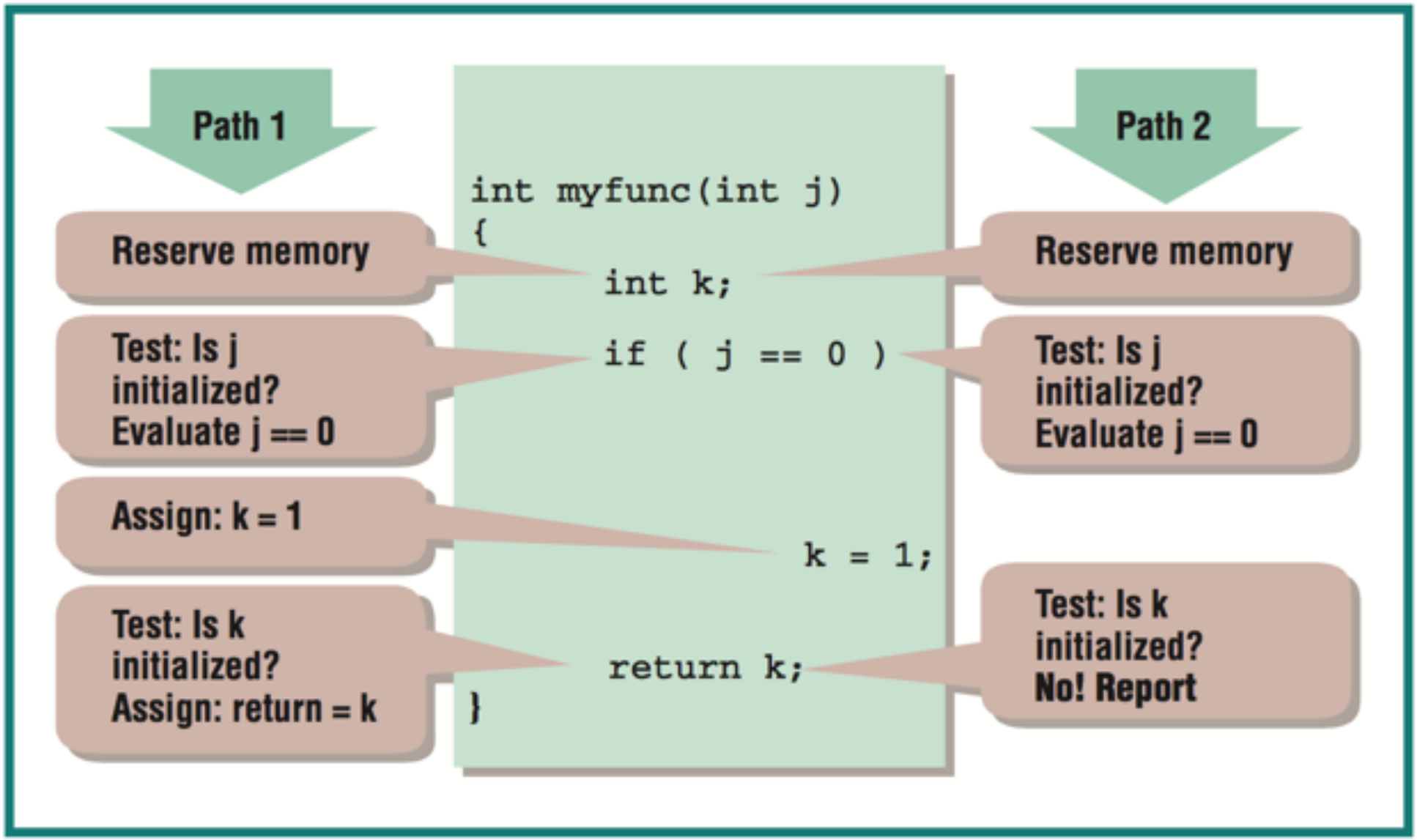
SWE 795, Spring 2017

Software Engineering Environments

Detecting Errors with Static Analysis

- Goal: Find important defects through static analysis.
  - Errors in using PL features such as referencing uninitialized variable
  - Errors in API usage such as closing a file descriptor twice
- NOT to determine that program definitely behaves as expected
  - Only to rule out programs that are obviously buggy

Key Idea: Path Exploration



Challenge: Path Feasibility

```
FILE* f;
if (complex_calc1())
     f = fopen(...);
...
if (complex_calc2())
     fclose(f);
```

- Determining if path is feasible is hard
- Can't always succeed
- What happens when you don't?
- Show an error that might be spurious (Sound)
- Don't show an error that might be spurious (Complete)
- Show an error if you're fairly confident (Heuristics)

(Some) MS Static Analysis Tools

- Prefix—traverse interprocedrual paths, builds models of functions, limit # of paths & use heuristics

- Prefast—use local intraprocedural analysis to find idioms such as wrong param values

- Slam—find protocol violations by building & refining Boolean program abstractions

- ESP—less precise, more scalable interprocedural analysis by pruning irrelevant branches

- Valut—safe version of C that enables modular checking by adding annotations

History

- Tools initially built in late 90s and 2000s at Microsoft
- Used widely internally & in some cases externally (e.g., device drivers)
- Slam tool credited with helping reduce blue screens in MS Windows (other innovations helped too)

Questions for discussion

- Overall reaction to the paper

- Would you use such a system for your everyday programming?
  - Why or why not?

- In what circumstances are these tools most useful?