AUTOMATED SOFTWARE TESTING: PROJECTS AND SOME CHALLENGES

Marcelo d'Amorim (damorim@cin.ufpe.br)

Assistant Professor at Federal University of Pernambuco (UFPE)

August 01, 2012 (visit Prof. Ralf Lämmel)

Software errors are expensive

Software is everywhere

Annual cost of software errors to the US economy is ~60B dollars [NIST2002]

Main approaches for finding errors

- Software Testing
  - Run code and observe effects
- Static Analysis
  - Analyze code without running it

~20B dollars can be saved with better infrastructure [NIST2002]

My Research Goal

Develop new techniques and improve existing techniques for Program Analysis (Software Testing)

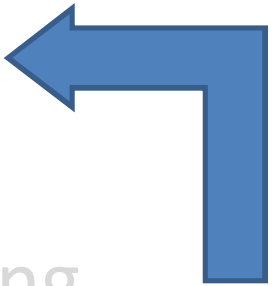
Current Students

- Mateus Borges, UROP since 2009
- João Paulo Oliveira*, MS since 2010
- Sabrina Souto, PhD since 2011

* co-advised with Fernando Castor (Cin-UFPE)

Projects and Students

- Improve Symbolic Execution
  - Mateus Borges
- Improve Software Model Checking
  - João Paulo Oliveira
- Improve Reliability of Softwar



Focus more on sequential programs

Projects and Students

- Improve Symbolic Execution
  - Mateus Borges
- Improve Software Model Checking
  - João Paulo Oliveira
- Improve Reliability of Software Product Lines

Focus more on concurrent programs. In particular, data-races.

Projects and Students

- Improve Symbolic Execution
  - Mateus Borges
- Improve Software Model Checking
  - João Paulo Oliveira
- Improve Reliability of Software Product Lines

How to understand,
test, and evolve SPLs?

(Super-) Short agenda

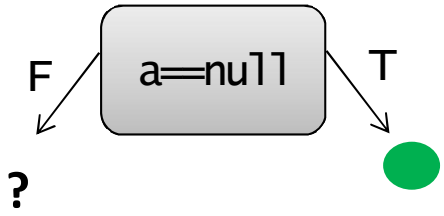
- I will talk (~10min) about Mateus' work and about some open problems (/ideas)
- Next, João Paulo and Sabrina will talk about their work

Symbolic Execution (SE)

Example from Microsoft SE Tool PEX

Code to generate inputs for:

```
void coverMe(int[] a) {
  if (a == null) return;
  if (a.length > 0)
    if (a[0] == 1234567890)
      throw new Exception("bug");
}
```



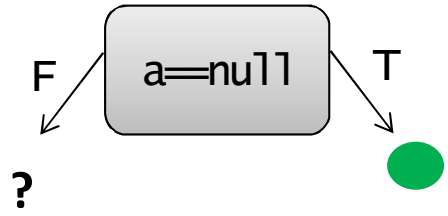
Constraints to solve	Inputs	Observed constraints
	null	a==null

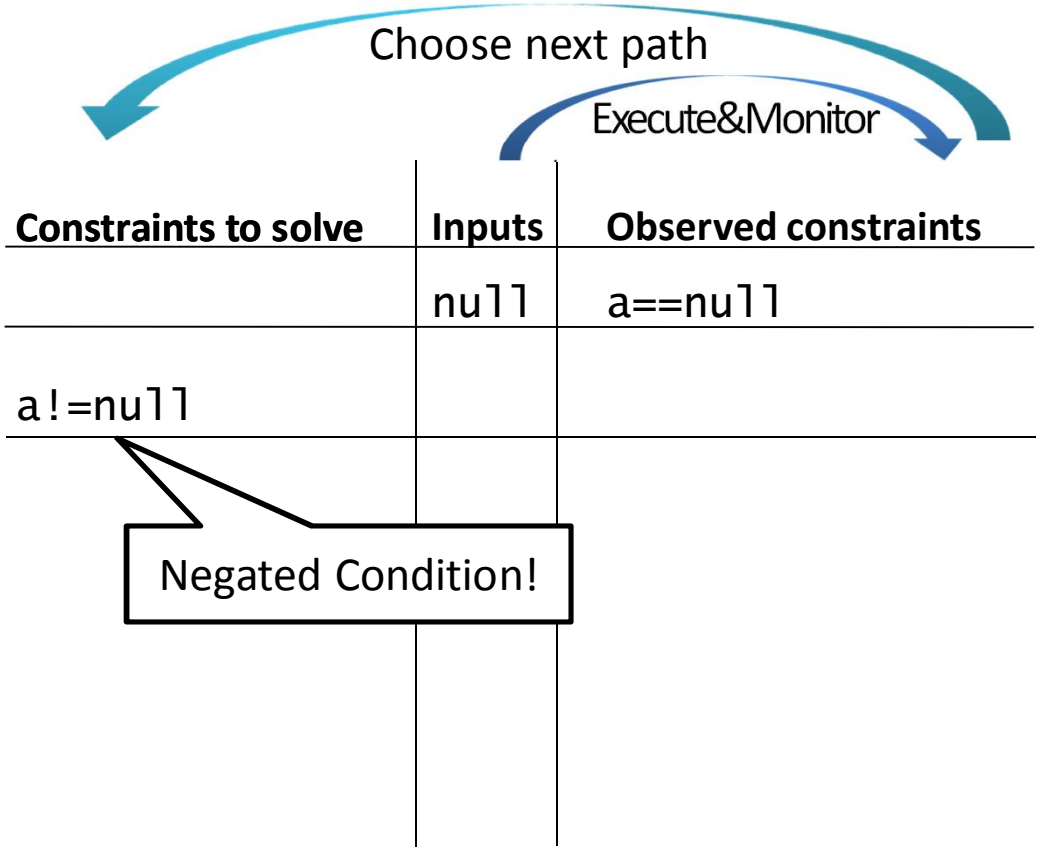
Symbolic Execution (SE)

Example from Microsoft SE Tool PEX

Code to generate inputs for:

```
void coverMe(int[] a) {
  if (a == null) return;
  if (a.length > 0)
    if (a[0] == 1234567890)
      throw new Exception("bug");
}
```



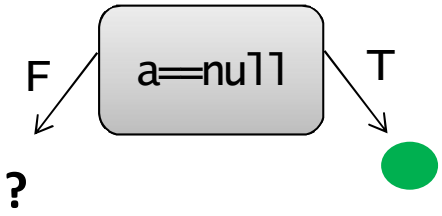


Symbolic Execution (SE)

Example from Microsoft SE Tool PEX

Code to generate inputs for:

```
void coverMe(int[] a) {
  if (a == null) return;
  if (a.length > 0)
    if (a[0] == 1234567890)
      throw new Exception("bug");
}
```



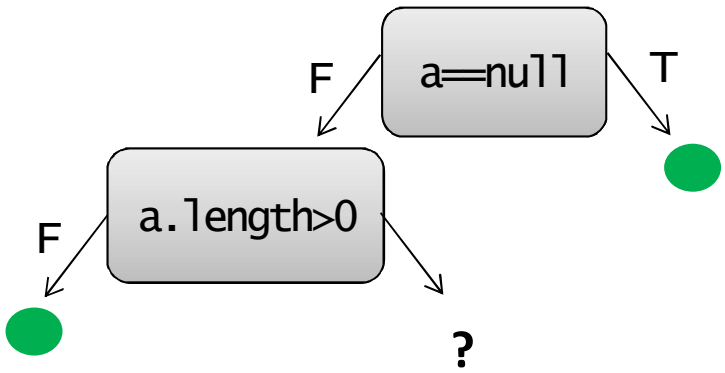
Constraints to solve	Inputs	Observed constraints
	null	a==null
a!=null	{}	

Symbolic Execution (SE)

Example from Microsoft SE Tool PEX

Code to generate inputs for:

```
void coverMe(int[] a) {
  if (a == null) return;
  if (a.length > 0)
    if (a[0] == 1234567890)
      throw new Exception("bug");
}
```



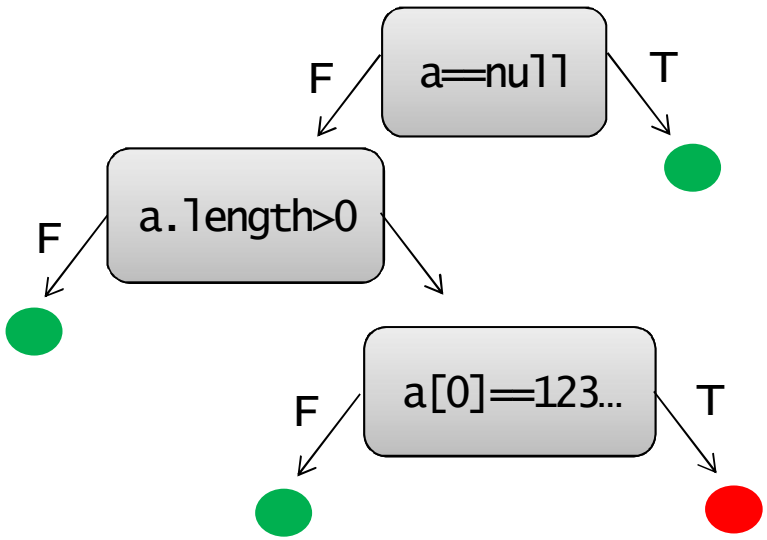
Constraints to solve	Inputs	Observed constraints
	null	a==null
a!=null	{}	a!=null && !(a.length>0)

Symbolic Execution (SE)

Example from Microsoft SE Tool PEX

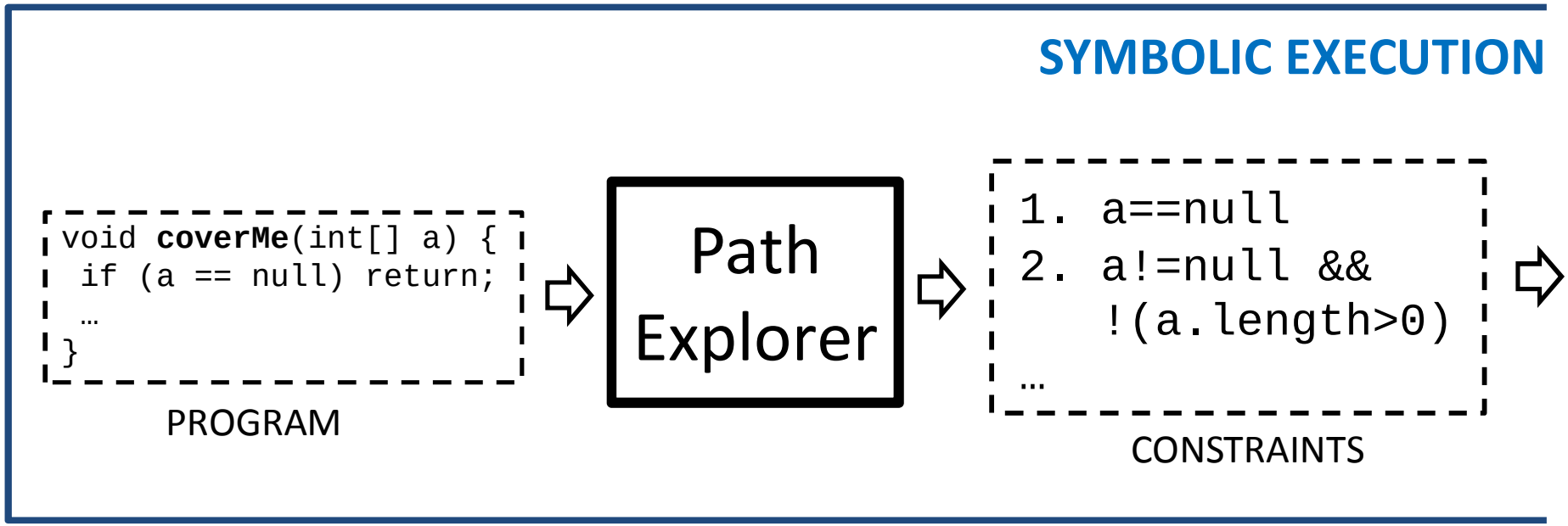
Code to generate inputs for:

```
void coverMe(int[] a) {
  if (a == null) return;
  if (a.length > 0)
    if (a[0] == 1234567890)
      throw new Exception("bug");
}
```



Constraints to solve	Inputs	Observed constraints
	null	a==null
a!=null	{}	a!=null && !(a.length>0)
a!=null && a.length>0	{0}	a!=null && a.length>0 && a[0]!=1234567890
a!=null && a.length>0 && a[0]==1234567890	{123..}	a!=null && a.length>0 && a[0]==1234567890

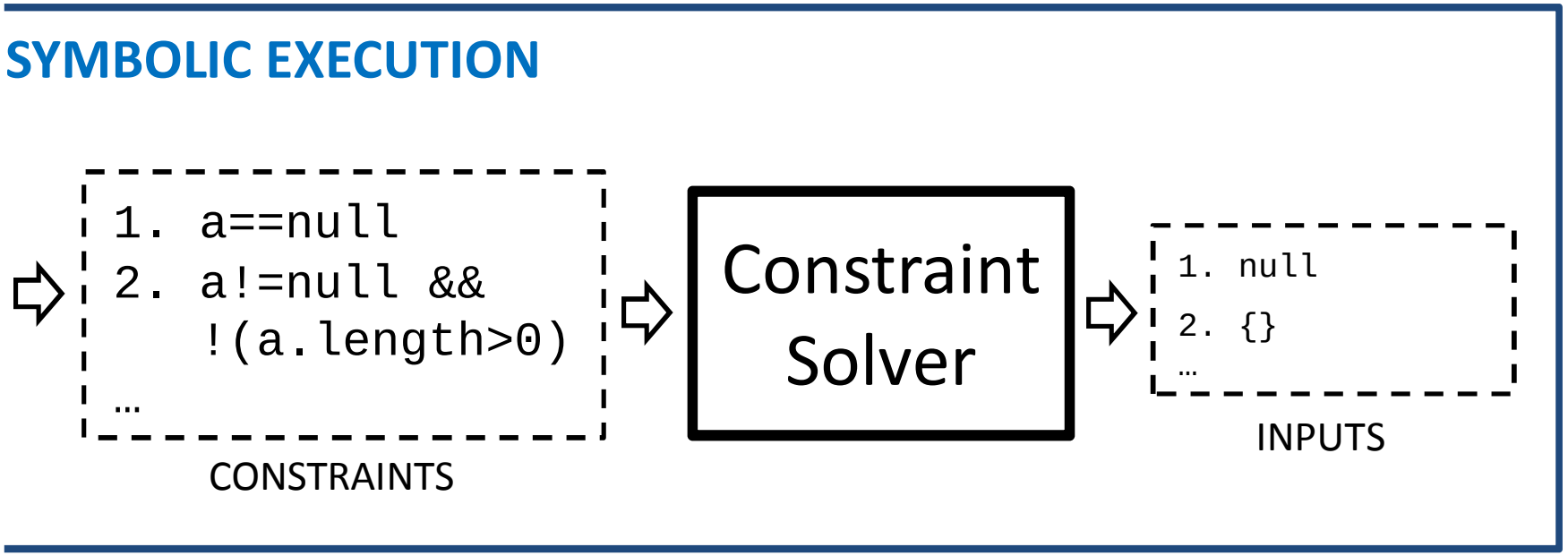
Two Components of SE (1)



Explores all program paths (up to bounds)

Two Components of SE (2)

contd.



Finds inputs from constraints

Mateus' Problem

- Improve constraint solving for Symbolic Execution
  - Focus is on complex* constraints
    - Aerospace safety: floating-point (fp)
    - Web-security vulnerabilities: string + linear integer

- Definition for **complex** constraints:
  - Constraints that involve intractable (e.g., fp arithmetic) or undecidable theories (e.g. non-linear integer arithmetic)
  - Constraints that are nontrivial to model on current off-the-shelf SMT solvers (e.g., strings + integers)

Results of his work

- CORAL: a meta-heuristic solver for dealing with floating-points and math functions
  - NASA Formal Methods (NFM'11)
  - International Conference on Software Testing Verification & Validation (ICST'12)
- Currently:
  - Google Summer of Code Student, improving Symbolic Path Finder (NASA SE tool)
- Future:
  - Improve String Solving and Look for Real Errors

SOME CHALLENGES
(YOU CAN ALSO READ AS: LIMITATIONS/THOUGHTS)

Problem 1: How to test Highly-Dynamic Systems?

- Context
  - Today's systems are more heterogeneous and dynamic. E.g., use geographical data, provide advanced user-interfaces, integrate with many sub-systems, etc.
- Problem
  - These systems are (very?) hard to test
- Solution (?)
  - Investigate novel ways to model (/mock) sub-systems

Problem 2: Utility of techniques

- Context
  - Each year several new analysis techniques are proposed in research venues
- Problem
  - Hard to evaluate new techniques (non-incremental), time-consuming to develop, questionable utility, etc.
- Solution (?)
  - Propose canonical symbolic representations of analysis information and protocol for collaboration

Problem 3: Utility of test automation (note: this is the topic of my talk)

- Context
  - Test Automation does not preclude user intervention. E.g., human still needs to evaluate quality of generated test, inform oracles, etc.
- Problem
  - Sometimes test automation creates different problems for the human to deal with ☹
- Solution (?)
  - (1) Enable the machine to help the human tester (as opposed to solving the problem). (2) Enable the machine to continuously accept user-intervention.

Problem 3: Utility of test automation (note: this is the topic of my talk)

- Context
- (1) Enable the machine to help the human tester (as opposed to solving the problem). (2) Enable the machine to continuously accept user-intervention.

My opinion on this: Even though Test Automation has shown useful in some contexts (e.g., as a complement to human testing), it is also **not** so useful in some contexts! But it is important to keep advancing the field with the limitations in mind. This way we could see more effective complementation upcoming. In particular, new interesting ways of interaction between human-and-machine for testing.