Beyond Testing

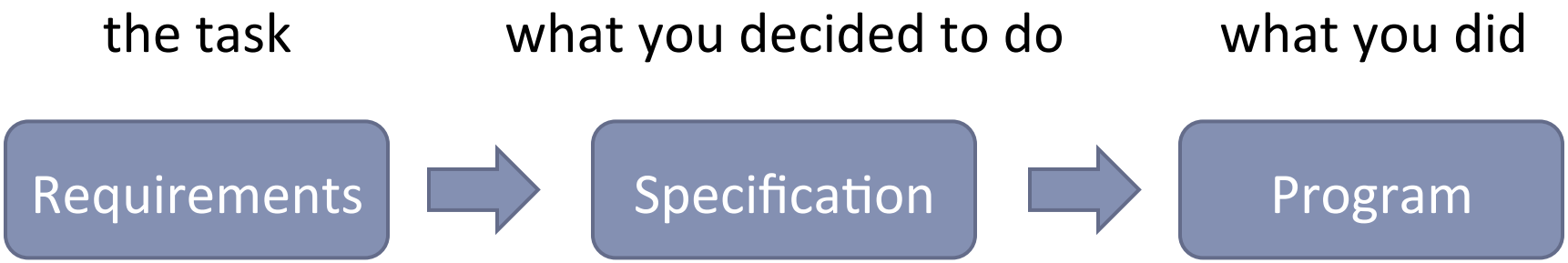
Paolo Giarrusso
(with slides from Tillmann Rendel)

The Central Question

Is My Program Correct?

What Is Correct?

- A correct program fullfills the specification.
- A correct specification reflects the requirements.



Source of Errors #1:

Specification does not reflect requirements.

Source of Errors #2:

Implementation does not fullfill specification.

Is My **Program** Correct?

- (Today, focus on correctness of programs, not specifications)

Correctness is Undecidable

Rice's Theorem:

For any non-trivial property of partial functions,
there is no general and effective method
to decide whether an algorithm
computes a partial function with that property

- By Rice's theorem, it is undecidable whether a program fullfills a specification.

Correctness is Undecidable

Rice's Theorem:

For any non-trivial property of partial functions,
there is no general and effective method
to **decide** whether **an algorithm**
computes a partial function with that property

- What do those words exactly mean?

Without violating Rice's theorem, we can:

- Check syntactic properties of a program
  - "This program does not manipulate files, hence it cannot destroy disk data!"
- Try to check semantic properties, without *always deciding* them
  - We will get false positives, false negatives, or both.

Approximate Correctness

Search For Errors

- Soundness
  - If an error is found, it is really an error
- Completeness
  - If no error is found, there is no error.

Prove Correctness

- Soundness
  - If code is proven correct, it is really correct
- Completeness
  - If the code is not proven correct, there is an error.

by Rice‘s theorem, we can‘t be sound **and** complete

How To Search For Errors

Run The Program

- Testing
  - Run the program once
- Test case generation
  - Run the program often
- Debugging
  - Watch the program running

Don‘t Run The Program

- FindBugs
  - Look for bug patterns
- Weak typing

How To Prove Correctness

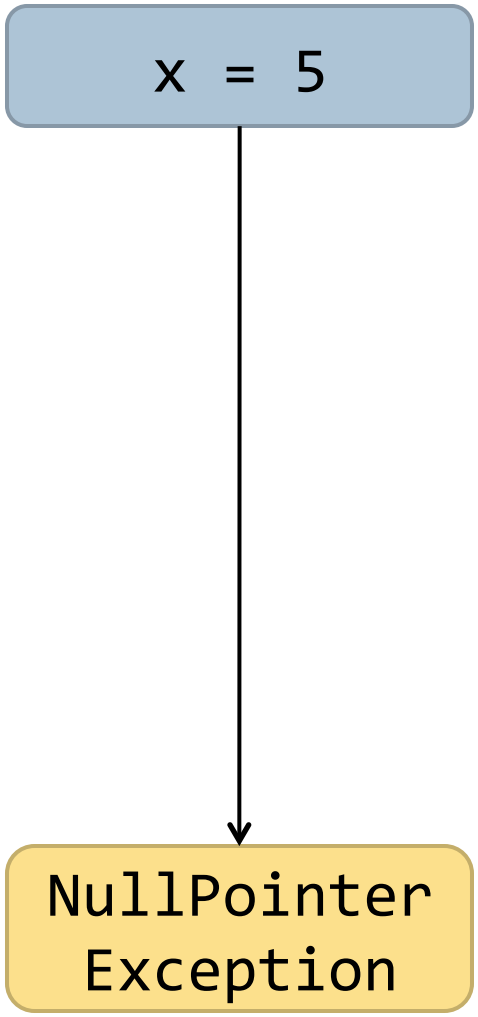
„Run“ The Program

- Model Checking
  - Check **all** paths

Don‘t Run The Program

- Static Analysis
- Strong Typing
- Theorem Proving
- Proof-Carrying Code
  - Attach proof to code
- Dependent Types
  - Specify correctness in types

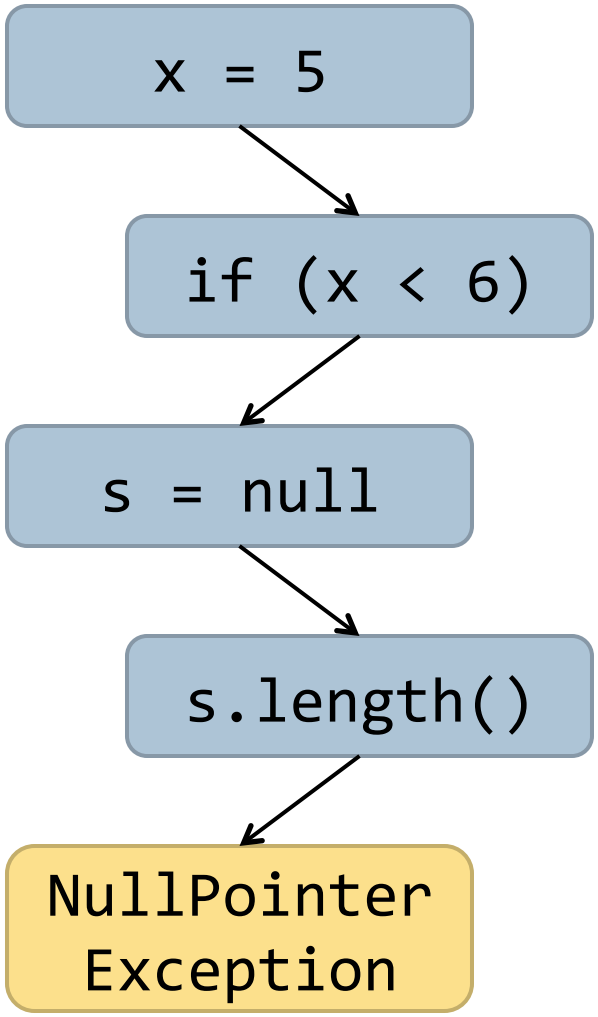
Testing



- Run the program
- Inspect the result

- Sound
  - All found errors are real
- Incomplete
  - Can‘t find all errors
- A lot of work
  - Need to specify test cases

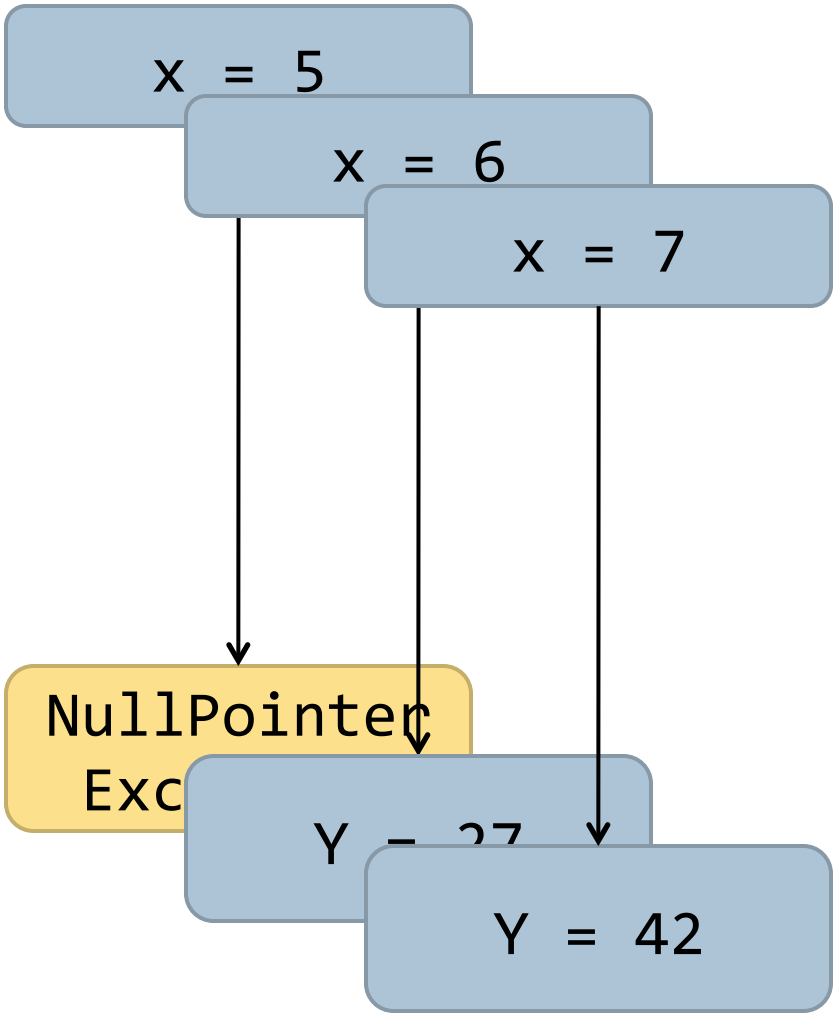
Debugging



- Watch the program run
- Inspect intermediate state
- Understand the problem

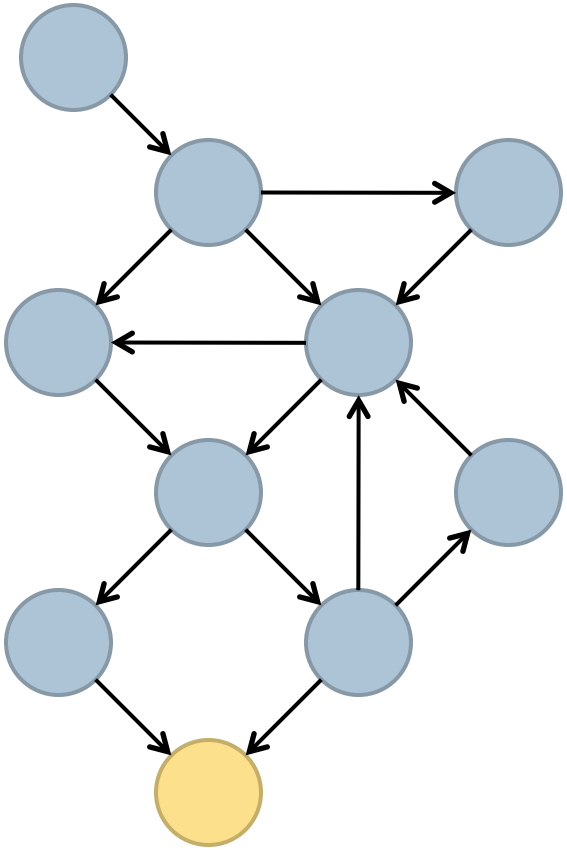
- Demo: Eclipse Debugger

Test Case Generation



- Generate many test cases
- Sound and Incomplete
  - Like testing
- Less Work
  - Get test cases generated
- Do we generate good test cases, or are they all trivial?
- Demo: ScalaCheck

Model Checking



- Explore the state space
- Sound
  - All errors are real.
- Complete
  - All paths explored.
- Problem
  - State space too big, maybe even infinite
- Demo: Java Pathfinder

Static Analysis

- Analyze source code
  (without running the program)
- Sound
  - Results describe runtime behavior correctly
- Incomplete
  - Not all runtime behavior can be predicted
- Example: FindBugs