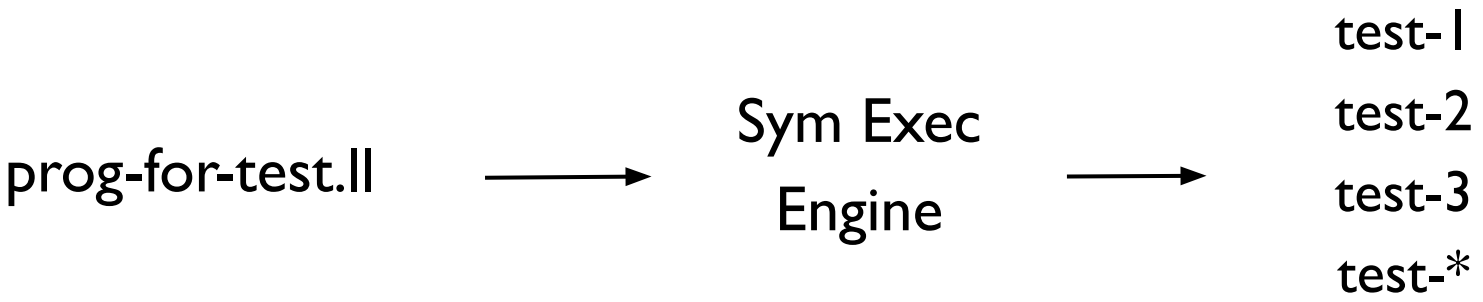
LLSC: A Parallel Symbolic Execution Compiler for LLVM IR

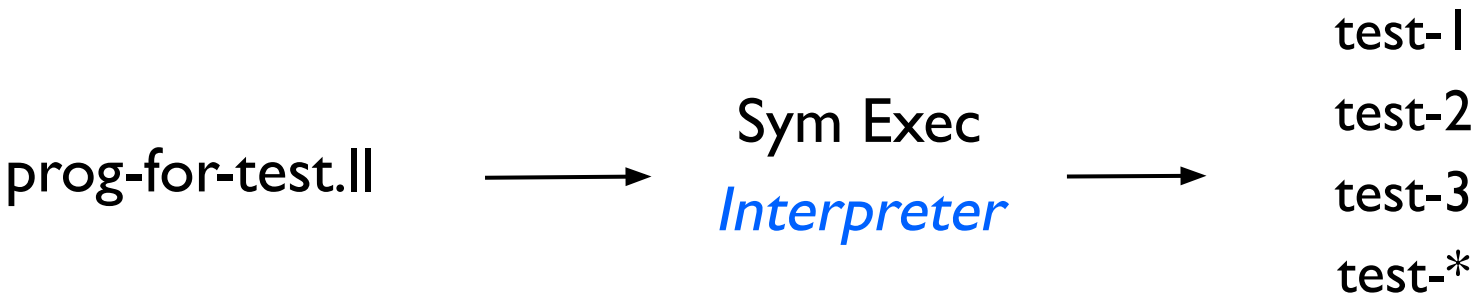
Guannan Wei, Shanyin Tan,
Oliver Bračevac, and Tiark Rompf

ESEC/FSE 2021, Online
Demonstration Track

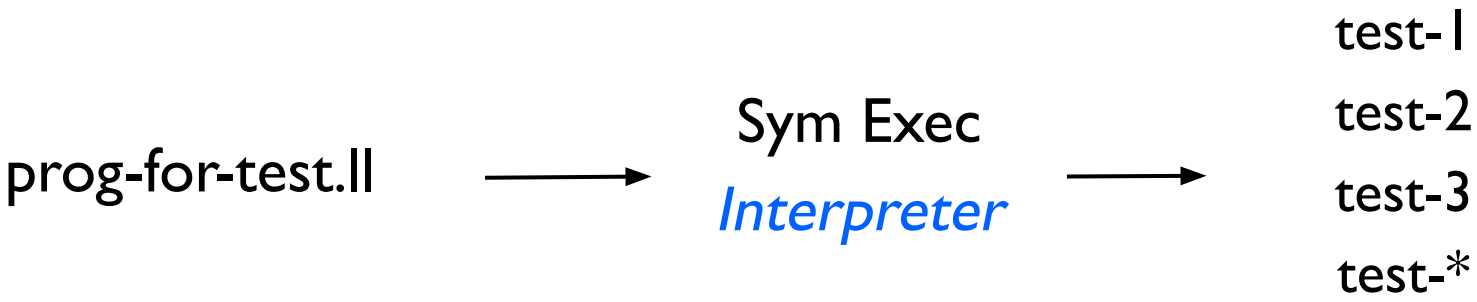
Symbolic Execution



Symbolic Execution



Symbolic Execution



Slow, because interpretation overhead

Q: How to eliminate the interpretation overhead?

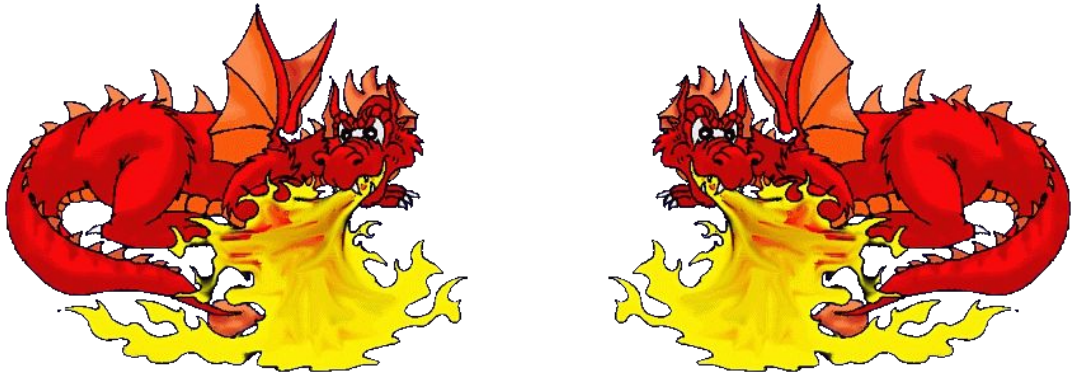
Q: How to eliminate the interpretation overhead?

A: Compilation.

Symbolic Execution by Compilation

Generating code that performs
symbolic execution
without interpretation overhead

Symbolic Execution by Compilation



How to conquer the complexity of compilation + symbolic execution?

Image from https://www.cs.utexas.edu/~pingali/CS375/2010Sp/

Symbolic Execution by Compilation

LLSC: derive a symbolic compiler from
a symbolic interpreter by *metaprogramming*

The 1st Projection of Futamura

Yoshihiko Futamura

Compilation = Specializing an interpreter to a program

Generated code = The specialized interpreter to that program

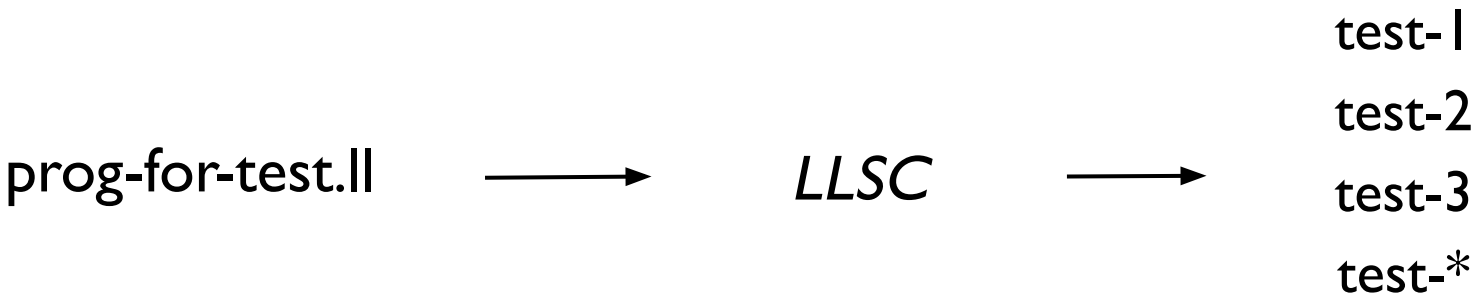
The 1st Projection of Futamura
by Staging

A staged interpreter is a compiler.

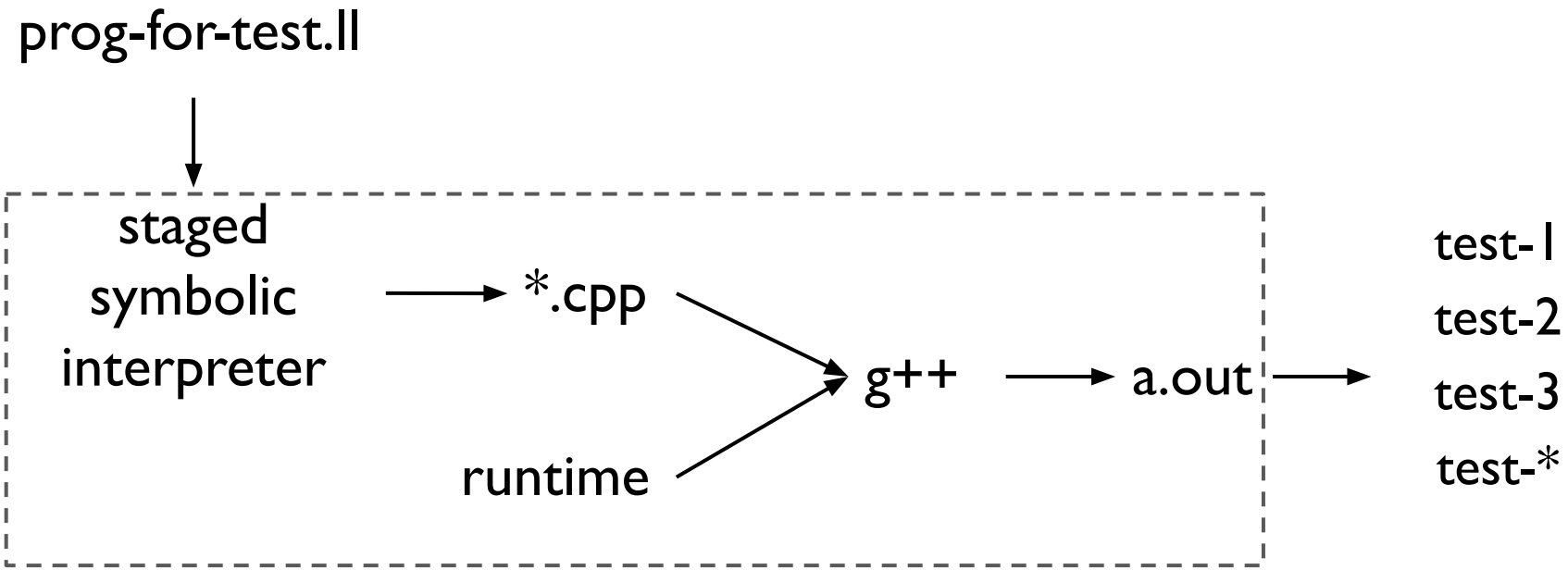
The 1st Projection of Futamura
by Staging

A staged *symbolic* interpreter is a *symbolic* compiler.

How does *LLSC* work?



How does *LLSC* work?



LLSC

- A *staged symbolic interpreter* for LLVM IR written in Scala

LLSC

- A *staged symbolic interpreter* for LLVM IR written in Scala

- Generates C++ code to symbolically execute multiple paths *in parallel*, and to generate test cases

LLSC

- A *staged symbolic interpreter* for LLVM IR written in Scala
- Generates C++ code to symbolically execute multiple paths *in parallel*, and to generate test cases
- Generated C++ code runs much *faster* than interpretation to explore multiple execution paths

Thanks for watching!

More results, source code, Docker image available at:

https://continuation.passing.style/llsc