TeTRIS: Fuzzing for Code Translation Bugs in Source-to-Source Code Transpilers

Yeaseen Arafat, Stefan Nagy

Introduction

- **Transpilers** are an emergent class of automated tools that convert code between languages, enabling interoperability, migration, and memory safety.
- **TeTRIS:** A general-purpose fuzzer for testing source-to-source code transpilers, enhancing syntactic and semantic testing of transpilers.
  - Tested against **four** leading fuzzers across **seven** transpilers, including **C2Rust** (C to Rust), **CxGo** and **C4Go** (C to Go), **Go2Hx** (Go to Haxe), and **HxCpp** (Haxe to C++), among others.
  - Superior in generating valid tests, achieving extensive coverage, and bug discovery.
  - Revealed **12 new translation bugs**, with **all** confirmed by the transpilers' developers.

Bug Type	Transpiler	Brief Error Description
Syntactical	CxGo	Mis-included variable re-declaration
	C2Go	Mis-recovered `switch-case` values
	C2Nim	Mis-recovered `struct` members
	HxCpp	Mis-recovered null `array` in callee
	Zig Translate-C	Mis-recovered `array` type and size
Type Conversion	C2Rust	Mis-recovered `int`→`float` conversion
	Zig Translate-C	Mis-recovered `bool`→`int` conversion
	Zig Translate-C	Mis-recovered `int`→`bool` conversion
Code Fragment	C2Go	Unrecovered `struct` name
	C4Go	Unrecovered `unary` operator
	C2Nim	Unrecovered standard `int` types
	C2Rust	Unrecovered bitfield `struct`
	C2Rust	Unrecovered `call` instruction
	Go2Hx	Unrecovered `array` size
	HxCpp	Unrecovered optional `int` arguments

Previously known translation bugs

Background

- **CSmith** cannot adapt to the diverse languages modern transpilers handle, such as Haxe and Go.
- **AFL++** and **AFL-Compiler-Fuzzer** almost always produce syntactically-invalid code.
- **Polyglot** generates syntactically-valid code but struggles with generating semantically-valid code, achieving only 2.X% semantic accuracy.

Fuzzer	Supports all Transpilers	Code Validity Syntactic	Code Validity Semantic	Extensible Mutation
AFL++	✔	✗	✗	✔
libFuzzer	✔	✗	✗	✔
AFL-Compiler-Fuzzer	✔	✗	✗	✔
Polyglot	✔	✔	✗	✔
CSmith	✗	✔	✔	✗
RustSmith	✗	✔	✔	✗
YARPGen	✗	✔	✔	✗
TeTRIS (our framework)	✔	✔	✔	✔

Available fuzzing techniques and their fundamental characteristics with respect to fuzzing transpilers.

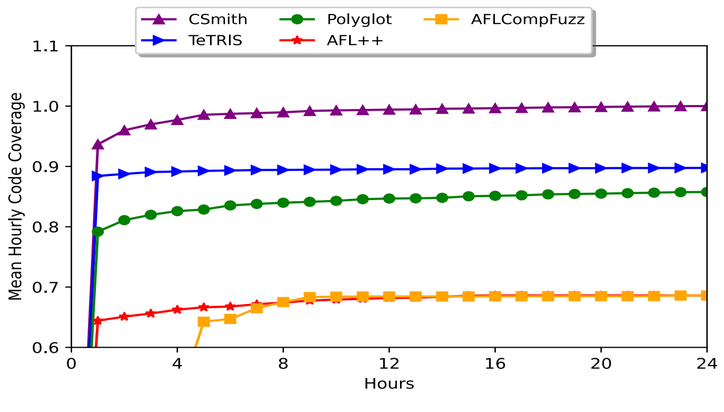
Our fuzzing approach combines **(1) the flexibility of language-agnostic fuzzers with (2) the correctness of language-specific ones—**extending comprehensive testing across today's ever-growing transpiler ecosystem.

Evaluation: Validity Rate

Transpiler	TeTRIS %VALID	Polyglot %VALID	AFL-CompFuzz %VALID	AFL++ %VALID	CSmith %VALID
C2Rust	77.39%	19.56%	7.48%	4.94%	95.57%
CxGo	71.84%	8.18%	1.53%	2.72%	n/a
C4Go	55.41%	6.67%	0.82%	1.13%	n/a
HxCpp	75.72%	n/a	0.83%	0.79%	n/a
HxPy	77.80%	n/a	1.95%	1.72%	n/a
Mean:	71.63%	11.47%	2.52%	2.26%	95.57%

Test case validity rate per transpilers.

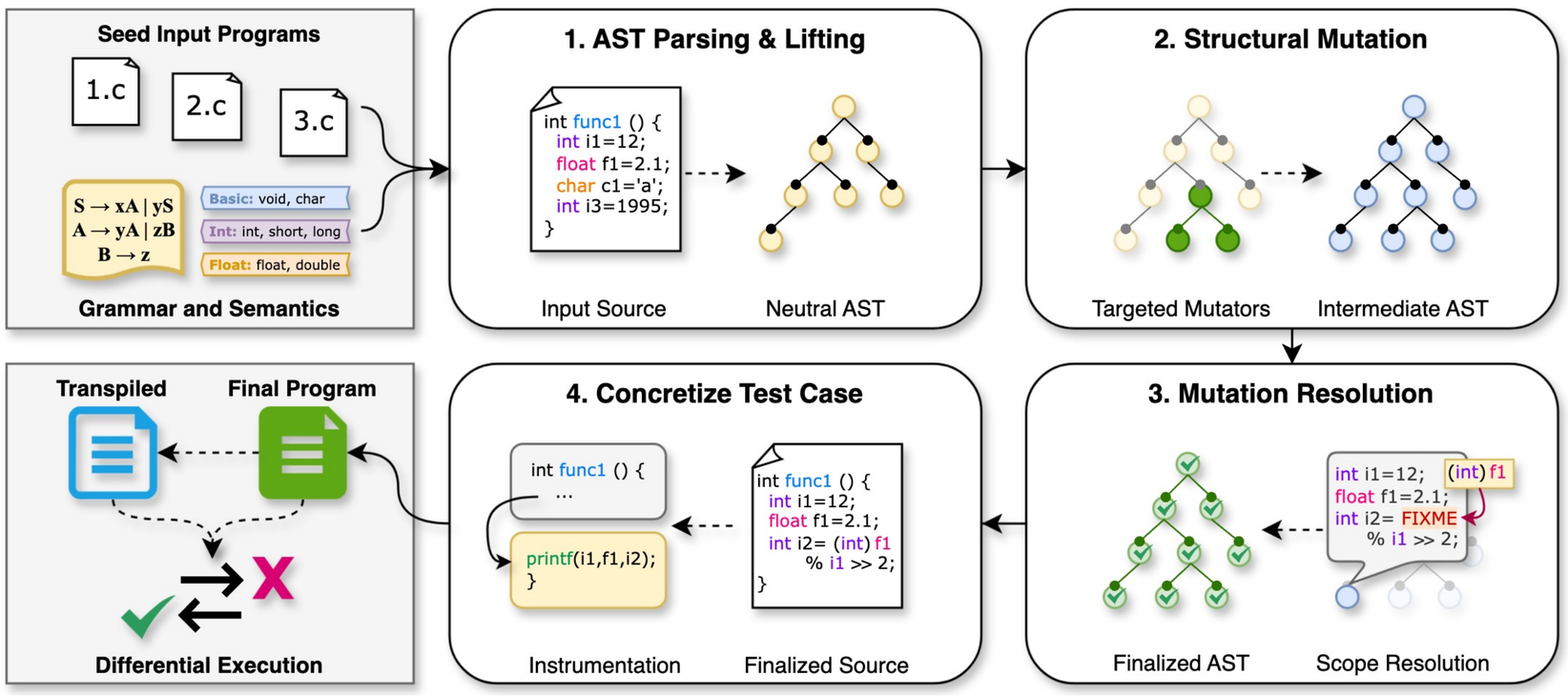
Evaluation: Transpiler Code Coverage



C2Rust Code Coverage Graph

How TeTRIS Fuzzes Transpilers

The following shows a visualization of TeTRIS's four high-level steps and a side-by-side example visualizing TeTRIS's successive operations on a single input

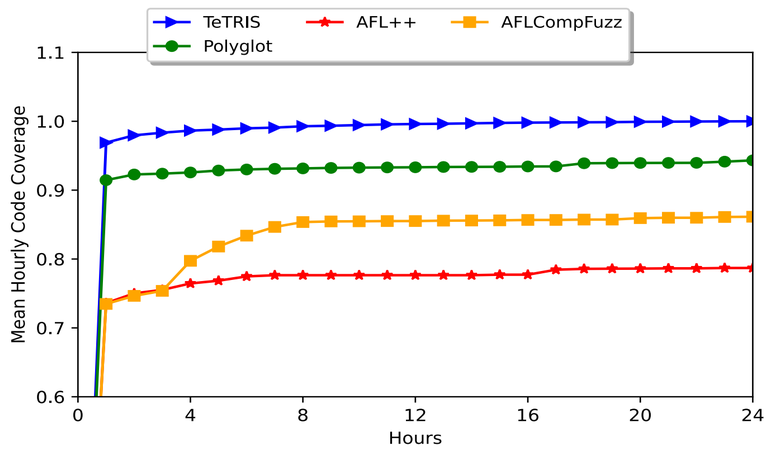



```
1 if ( p > 0 ){
2
3   p++;
4   var = p;
5
6 }
```
(a) Seed program.

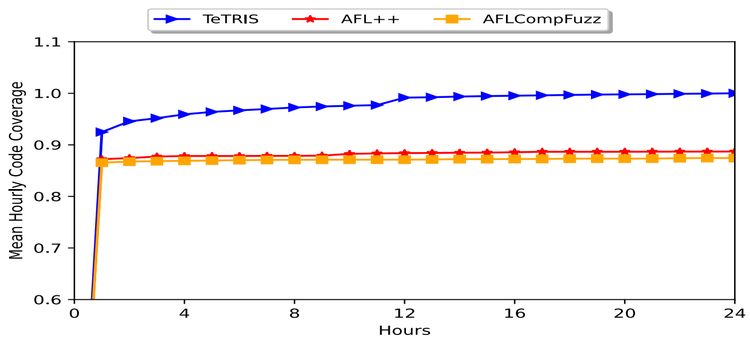
```
1 if ( p > 0 ){
2   if( _ <= 100
        ){
3     _--;
4     _ = _+_+_;
5   }
6 }
```
(b) After mutation.

```
1 if ( p > 0 ){
2   if (q+p <= 100 ){
3     var--;
4     var = p+q+var;
5   }
6 }
```
(c) After resolution.

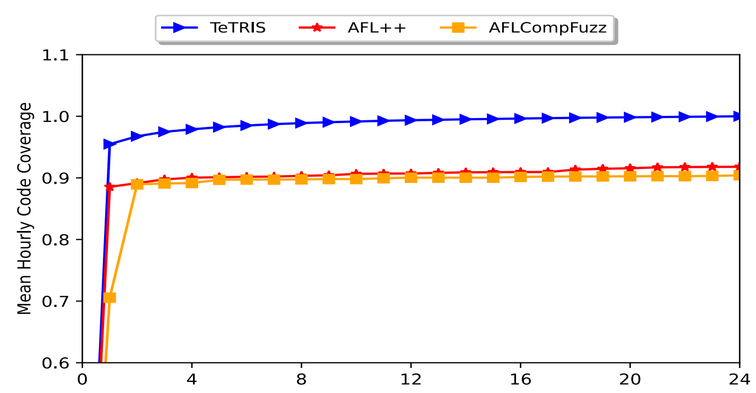
```
1 if( p > 0 ){
2   if( q+p <= 100 ){
3     var =
          (int)((float)(q+var)
        + (float)(p+q+var));
4   }
5 }
```
(d) After successive mutation/resolution.



CxGo Code Coverage Graph



HxCpp Code Coverage Graph



HxPy Code Coverage Graph

Evaluation: Bug Discovery

Transpiler	TeTRIS NEW	TeTRIS CONF	Polyglot NEW	Polyglot CONF	AFL-CompFuzz NEW	AFL-CompFuzz CONF	AFL++ NEW	AFL++ CONF	CSmith NEW	CSmith CONF
C2Rust	0	0	0	0	0	0	0	0	0	0
CxGo	7	7	0	0	0	0	0	0	n/a	n/a
C4Go	2	2	0	0	0	0	0	0	n/a	n/a
Go2Hx	2	2	n/a	n/a	0	0	0	0	n/a	n/a
HxCpp	0	0	n/a	n/a	0	0	0	0	n/a	n/a
HxPy	0	0	n/a	n/a	0	0	0	0	n/a	n/a
Zig Translate-C	1	1	0	0	0	0	0	0	n/a	n/a
Total:	**12**	**12**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Total new and confirmed bugs by per transpiler.

Transpiler	Bug Type	How Detected	Brief Error Description
C2Rust	Syntactical	Post-Transpile Failure	Mis-recovering bitfields contained in `union`
CxGo	Syntactical	Post-Transpile Failure	Mis-recovering 3-D `array` element pointer deref
CxGo	Syntactical	Post-Transpile Failure	Mis-recovering compound literal (e.g., `&(int)1`)
CxGo	Type Conversion	Post-Transpile Failure	Mis-recovering implicit conversion (`bool` → `float`)
CxGo	Type Conversion	Post-Transpile Failure	Mis-recovering explicit conversion (`bool` → `float`)
CxGo	Type Conversion	Post-Transpile Failure	Mis-recovering explicit conversion (`float` → `int`)
CxGo	Code Fragment	Runtime Divergence	Mis-recovering operations on `static` variable
CxGo	Code Fragment	Transpiler Failure	Crash parsing unreachable `switch` body code
C4Go	Type Conversion	Post-Transpile Failure	Unsupported type conversion (`bool` → `int`)
C4Go	Code Fragment	Post-Transpile Failure	Unrecovered initialization of value in `union`
Go2Hx	Syntactical	Transpiler Failure	Unrecovered `generic` types for function args
Go2Hx	Code Fragment	Runtime Divergence	Mis-recovering arch-specific type (`haxe.Int64`)
Go2Hx	Code Fragment	Runtime Divergence	Mis-recovering zeroed `array` comparisons
Zig Translate-C	Syntactical	Post-Transpile Failure	Mis-recovering bitfields contained in `struct`
Zig Translate-C	Syntactical	Post-Transpile Failure	Mis-including unreachable post-`return` code
Zig Translate-C	Code Fragment	Post-Transpile Failure	Unrecovered compound literal (e.g., `&(int)1`)

TeTRIS-found bugs

Evaluation: Bug Case Studies

The following shows several TeTRIS-found transpiler bugs

```
1 x += 100 + (4 < 4.5);
2 x += 9.9 + (float)(4 < 4.5);
```
(a) Original C.

```
1 x += 100 + float32(int(libc.BoolToInt(4
      < 4.5)))
2 x += 9.9 + float32(4 < 4.5) Error!
```
(b) CxGo Go Output.

```
1 x := [4]int{0, 0, 0, 0}
2 if x != [4]int{0, 0, 0, 0} {
3     panic("Error: not equal!")
4 }
```
(a) Original Go.

```
1 var x = new Array<Int>([0, 0, 0, 0]);
2 if (x != new Array<Int>([0, 0, 0, 0])) {
3       throw "Error: not equal!";
4 }
```
(b) Go2Hx Haxe Output.

```
1 x += 100 + (4 < 4.5);
2 x += 9.9 + (4 < 4.5);
```
(a) Original C.

```
1 x += 100 + float32(int(libc.BoolToInt(4
      < 4.5)))
2 x += 9.9 + float32(float64(4 < 4.5))
      Error!
```
(b) CxGo Go Output.

```
1 return x +=
      y*3;
2 x++;
3 return 0 ;
```
(a) Original C.

```
1 return blk: {
2   const ref = &x;
3   ref.* += y *
        @as(c_int,3);
4   break :blk ref.*;
5 };
6 x += 1;
7 return 0;
```
(b) Zig Translate-C Output.

Conclusion

- **TeTRIS** achieves the **highest transpiler coverage**, test case **validity**, and **bug discovery** of all fuzzers.
- **Future Directions**:
  - Supporting other transpiler ecosystems, such as JSweet, which translates Java to Javascript.
  - Extending to other tools (e.g., compilers, decompilers).

KAHLERT SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH
FUTURES³ LAB FUTURE TECHNOLOGY FOR USABLE, RELIABLE, & EFFICIENT SECURITY OF SOFTWARE & SYSTEMS