

TeTRIS:

General-purpose Fuzzing for Translation Bugs in Source-to-Source Code Transpilers

Yeaseen Arafat
University of Utah

Stefan Nagy
University of Utah

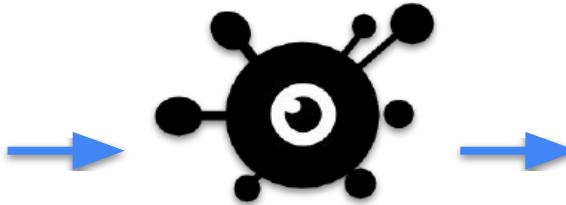


Transpilers: Automated Cross-language Code Translation

- High-level goal: translate code from one programming language to another

```
int func(){  
    int y, x, res;  
    if (y < 5) {  
        goto LABEL_RES;  
    }  
    if (x < 20) {  
        goto LABEL_END;  
    }  
    LABEL_RES:  
        res = x*y;  
    LABEL_END:  
        return res;  
}
```

Original C code



Transpiler
(e.g., C2Rust)

```
pub unsafe extern "C" fn func() ->  
libc::c_int {  
    let mut y: libc::c_int = 0;  
    let mut x: libc::c_int = 0;  
    let mut res: libc::c_int = 0;  
    if y < 5  
        { cur_blk = 1; }  
    else if x < 20  
        { cur_blk = 2; }  
    else  
        { cur_blk = 1; }  
    match cur_blk {  
        1 => { res = x*y; }  
        _ => {}  
    }  
    return res;  
}
```

Translated Rust code

Transpilers: Automated Cross-language Code Translation

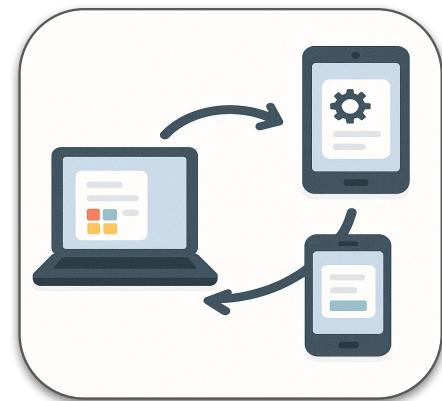
- **High-level goal:** translate code from one programming language to another
 - An emerging backbone of modern **translation-oriented** software development tasks



Software Migration



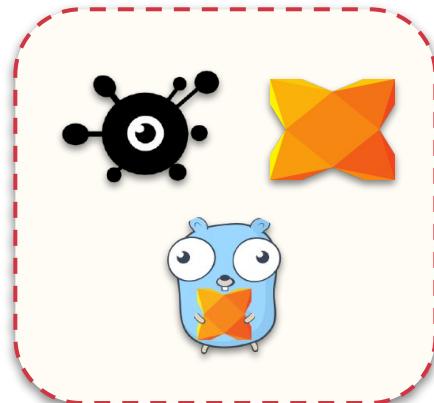
Security Retrofitting



Code Interoperability

Transpilers: Automated Cross-language Code Translation

- **High-level goal:** translate code from one programming language to another
 - An emerging backbone of modern **translation-oriented** software development tasks
 - Success of these development tasks often undermined by **incorrect code translation**



Buggy Transpilers



Mis-translated Code



Downstream Failures

Transpilers: Automated Cross-language Code Translation

- **High-level goal:** translate code from one programming language to another
 - An emerging backbone of modern **translation-oriented** software development tasks
 - Success of these development tasks often undermined by **incorrect code translation**



Problem: what kinds of **translation errors** do transpilers face **in practice?**



Buggy Transpilers



Mis-translated Code



Downstream Failures

Transpilers: Challenges to Successful Code Translation

- Manually distilled reported transpiler bugs into **three distinct categories**

Transpilers: Challenges to Successful Code Translation

- Manually distilled reported transpiler bugs into **three distinct categories**

Bug Category	Transpiler	GitHub Issue No. & Error Description
Syntactic Mishandling	CxGo	#42: Mis-included variable <code>re-declaration</code>
	C2Go	#848: Mis-recovered <code>switch-case</code> values
	C2Nim	#217: Mis-recovered <code>struct</code> members

Example:

CxGo #42

```
#define func(arg){  
    int x = arg;  
}  
if (0) {  
    func(5);  
    func(6);  
}
```

Original C code

```
if false{  
    var x int = 5  
    _ = x  
    var x int = 6  
    _ = x  
}
```

Mis-translated Go

Transpilers: Challenges to Successful Code Translation

- Manually distilled reported transpiler bugs into **three distinct categories**

Bug Category	Transpiler	GitHub Issue No. & Error Description
Syntactic Mishandling	CxGo	#42: Mis-included variable <code>re-declaration</code>
	C2Go	#848: Mis-recovered <code>switch-case</code> values
	C2Nim	#217: Mis-recovered <code>struct</code> members
Type Conversion	C2Rust	#486: Mis-recovered <code>int→float</code> conversion
	Zig Translate-C	#20005: Mis-recovered <code>bool→int</code> conversion
	Zig Translate-C	#20638: Mis-recovered <code>int→bool</code> conversion

Example:

Zig #20005

```
#define func(x) (  
    2 ? 4 : 8  
)
```

Original C code

```
if(@as(c_int,2))  
    @as(c_int,4);  
else  
    @as(c_int,8);
```

Mis-translated Zig

Transpilers: Challenges to Successful Code Translation

- Manually distilled reported transpiler bugs into **three distinct categories**

Bug Category	Transpiler	GitHub Issue No. & Error Description
Syntactic Mishandling	CxGo	#42: Mis-included variable <code>re-declaration</code>
	C2Go	#848: Mis-recovered <code>switch-case</code> values
	C2Nim	#217: Mis-recovered <code>struct</code> members
Type Conversion	C2Rust	#486: Mis-recovered <code>int→float</code> conversion
	Zig Translate-C	#20005: Mis-recovered <code>bool→int</code> conversion
	Zig Translate-C	#20638: Mis-recovered <code>int→bool</code> conversion
Code Fragment Omissions	C2Nim	#240: Unrecovered standard <code>int</code> types
	C2Rust	#896: Unrecovered <code>call</code> instruction
	Go2Hx	#19: Unrecovered <code>array</code> size
	HxCpp	#225: Unrecovered optional <code>int</code> arguments

Example:

C2Rust #896

```
int func() {  
    g = 1;  
    return g;  
}  
int main() {  
    -func();  
}
```

Original C code

```
fn func() -> c_int {  
    g = 1 as c_int;  
    return g;  
}  
fn main() {  
    // -func();  
}
```

Mis-translated Rust

Transpilers: Challenges to Successful Code Translation

- Manually distilled reported transpiler bugs into **three distinct categories**

Bug Category	Transpiler	GitHub Issue No. & Error Description
Syntactic Mishandling	CxGo	#42: Mis-included variable <code>re-declaration</code>
	C2Go	#848: Mis-recovered <code>switch-case</code> values

Example:

C2Rust #896

```
int func() {  
    g = 1;  
    return g;  
}  
int main() {
```

Motivation: how can **mis-translation errors** be uncovered **automatically?**

Conversion	Zig Translate-C	#20638: Mis-recovered <code>int→bool</code> conversion
Code Fragments	C2Nim	#240: Unrecovered standard <code>int</code> types
	C2Rust	#896: Unrecovered <code>call</code> instruction
	Go2Hx	#19: Unrecovered <code>array</code> size
	HxCpp	#225: Unrecovered optional <code>int</code> arguments

```
fn func() -> c_int {  
    g = 1 as c_int;  
    return g;  
}  
fn main() {  
    // -func();  
}
```

Mis-translated Rust

Prior Work: Approaches for Fuzzing Code Processors

- **Observation:** current approaches **ill-suited** to today's diverse transpilers

Fuzzer
AFL++
libFuzzer
AFL-Comp-Fuzz
Polyglot
CSmith
YARPGen

Prior Work: Approaches for Fuzzing Code Processors

- **Observation:** current approaches **ill-suited** to today's diverse transpilers

Fuzzer	C1: Language Agnostic
AFL++	✓
libFuzzer	✓
AFL-Comp-Fuzz	✓
Polyglot	✓
CSmith	✗
YARPGen	✗



Supports **only C**
(e.g., Clang, GCC)



Any language
(e.g., C, Py, Go)

Prior Work: Approaches for Fuzzing Code Processors

- **Observation:** current approaches **ill-suited** to today's diverse transpilers

Fuzzer	C1: Language Agnostic	C2: Minimal Lang. Spec.
AFL++	✓	None
libFuzzer	✓	None
AFL-Comp-Fuzz	✓	None
Polyglot	✓	✓
CSmith	✗	✗
YARPGen	✗	✗



Supports **only C**
(e.g., Clang, GCC)

Fuzzer	Core	Spec
CSmith	~38.9 K LoC	
Polyglot	~7.0 K	~1.5 K

Prior Work: Approaches for Fuzzing Code Processors

- Observation: current approaches **ill-suited** to today's diverse transpilers

Fuzzer	C1: Language Agnostic	C2: Minimal Lang. Spec.	C3: Upholds Code Validity
AFL++	✓	None	✗
libFuzzer	✓	None	✗
AFL-Comp-Fuzz	✓	None	✗
Polyglot	✓	✓	✗
CSmith	✗	✗	✓
YARPGen	✗	✗	✓



Supports **only C**
(e.g., Clang, GCC)

Fuzzer	Semantic Validity
CSmith	~100%
Polyglot	~20%

Prior Work: Approaches for Fuzzing Code Processors

- **Observation:** current approaches **ill-suited** to today's diverse transpilers

Fuzzer	C1: Language Agnostic	C2: Minimal Lang. Spec.	C3: Upholds Code Validity
AFL++	✓	None	✗
libFuzzer	✓	None	✗
AFL-Comp-Fuzz	✓	None	✗
Polyglot	✓	✓	✗
CSmith	✗	✗	✓
YARPGen	✗	✗	✓



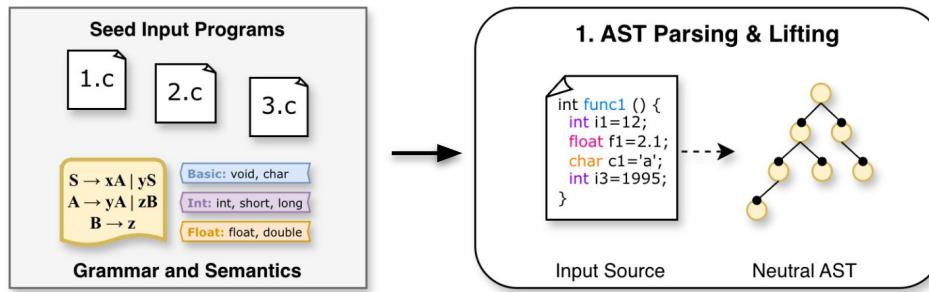
Supports **only C**
(e.g., Clang, GCC)

Fuzzer	Semantic Validity
CSmith	~100%
Polyglot	~20%

- **Takeaways:** transpiler fuzzing must balance **both generality and code validity**

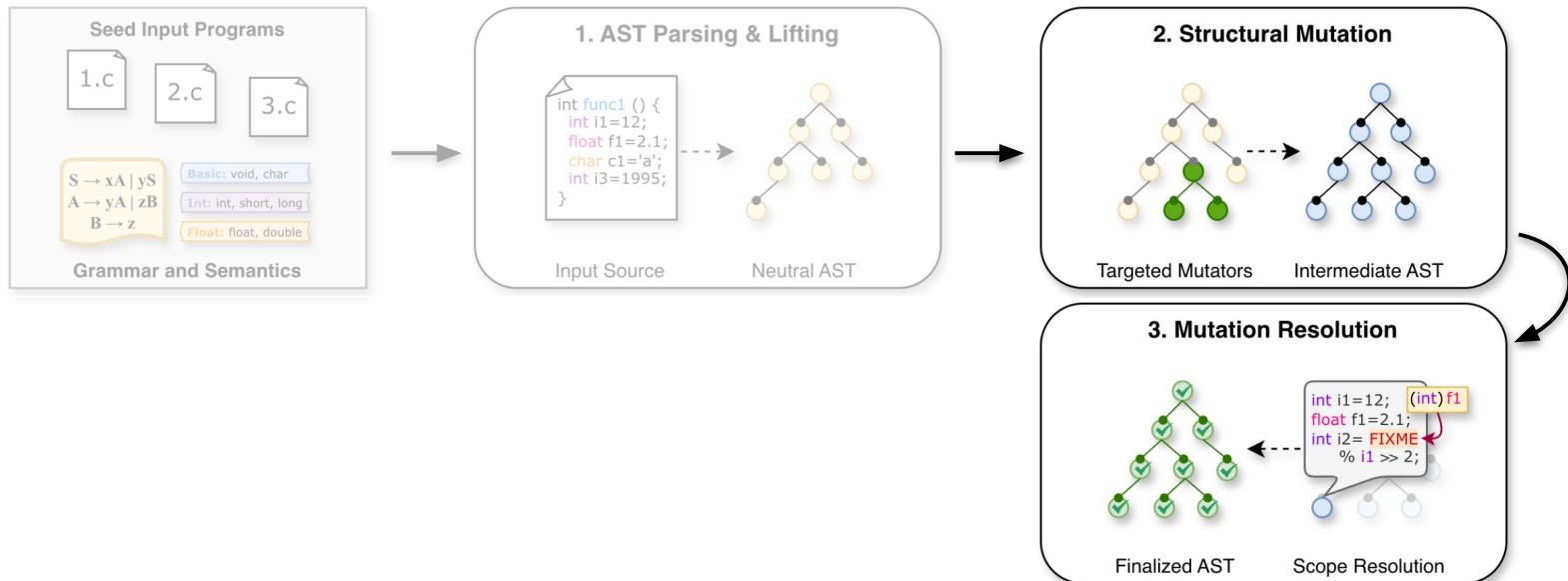
TeTRIS: Testing Transpilers Regardless of Input Source

- Key idea: grammar-guided mutation with language-level semantic guardrails



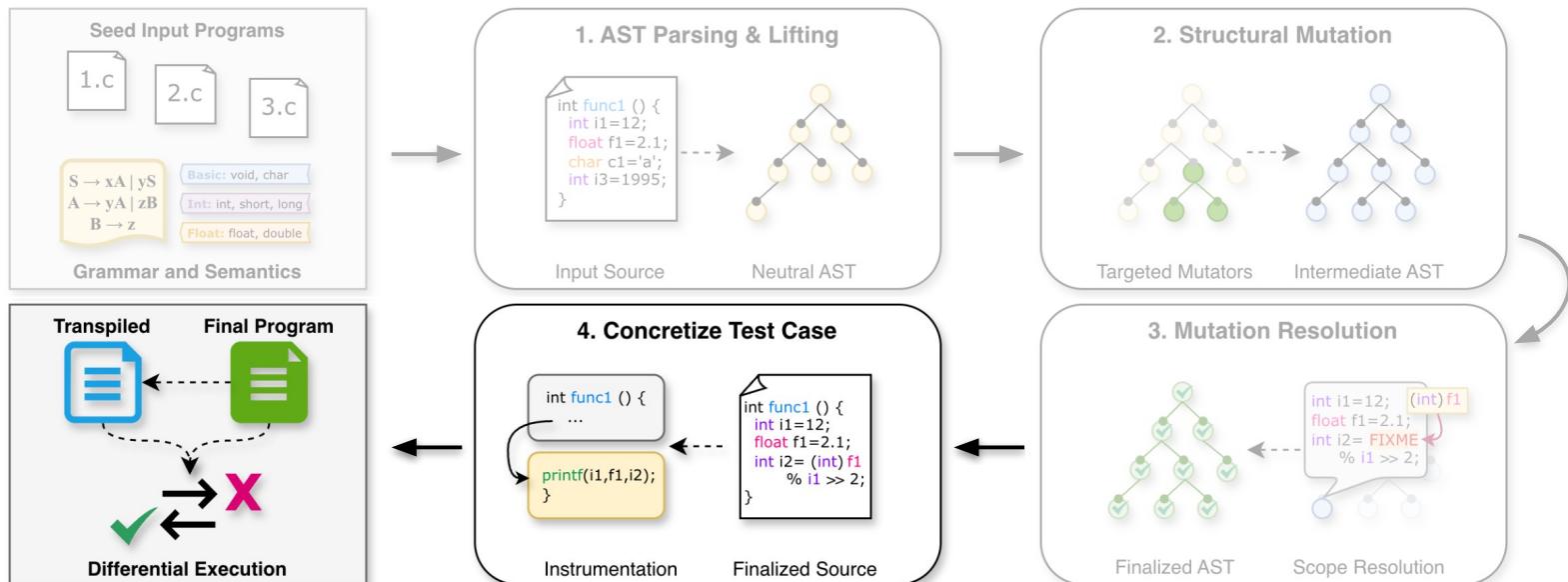
TeTRIS: Testing Transpilers Regardless of Input Source

- Key idea: grammar-guided mutation with language-level semantic guardrails



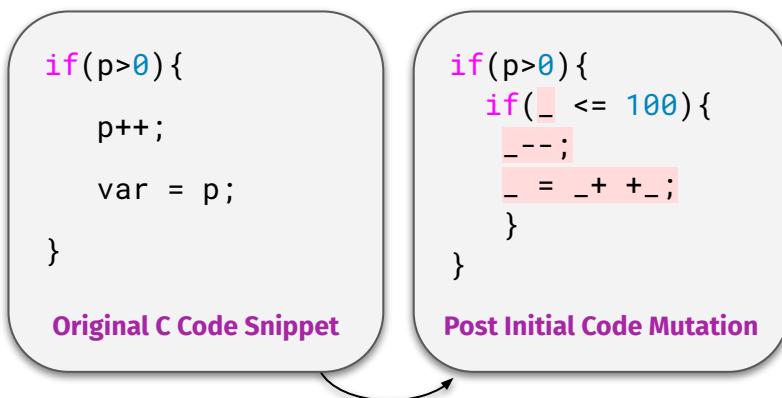
TeTRIS: Testing Transpilers Regardless of Input Source

- Key idea: **grammar-guided mutation** with **language-level semantic guardrails**
 - Transpiler-specific bug oracles: transpiler crashes, non-runnable code, differential execution



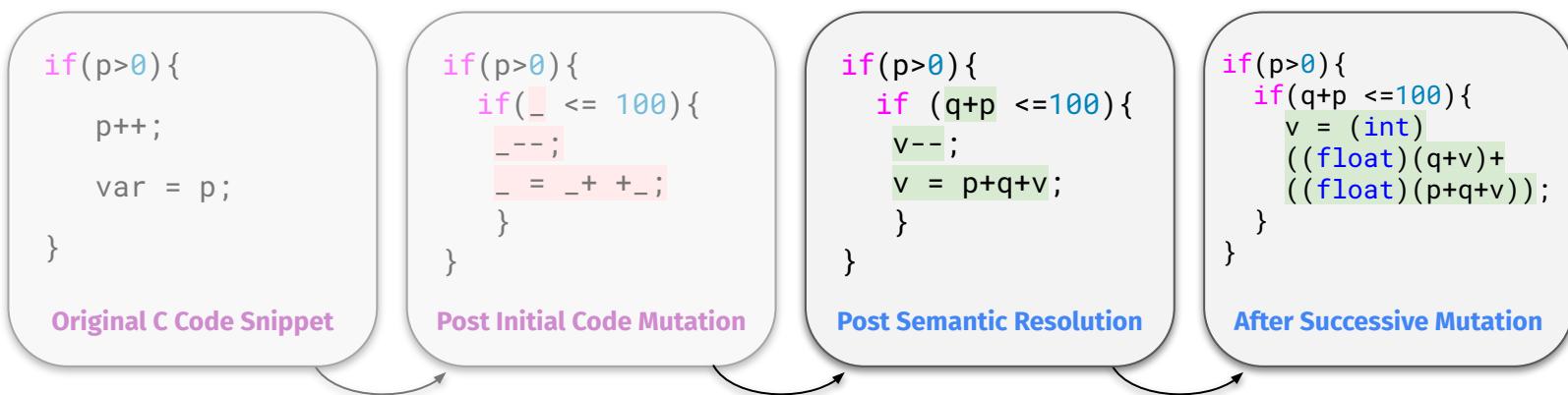
TeTRIS: Testing Transpilers Regardless of Input Source

- Key idea: **grammar-guided mutation** with **language-level semantic guardrails**
 - Transpiler-specific bug oracles: transpiler crashes, non-runnable code, differential execution



TeTRIS: Testing Transpilers Regardless of Input Source

- Key idea: **grammar-guided mutation** with **language-level semantic guardrails**
 - Transpiler-specific bug oracles:** transpiler crashes, non-runnable code, differential execution
 - Generated programs gradually cover deeper language **syntax and semantic combinations**



TeTRIS: Testing Transpilers Regardless of Input Source

- Key idea: **grammar-guided mutation** with **language-level semantic guardrails**
 - Transpiler-specific bug oracles:** transpiler crashes, non-runnable code, differential execution
 - Generated programs gradually cover deeper language **syntax and semantic combinations**



Fuzzer	Core	C Spec	Go Spec	Haxe Spec
CSmith	~38.9K LoC		✗	✗
Polyglot	~7.0 K	~1.5 K	✗	✗
TeTRIS	~5.6 K	811 LoC	797 LoC	785 LoC

- Outcome: **broad transpiler support** with only minimal language specification

Evaluation: Overview

- **Fundamental questions:**
 1. Is TeTRIS capable of **generating valid inputs?**
 2. Can TeTRIS **more thoroughly** test transpilers?
 3. Does TeTRIS uncover **more translation bugs?**
- Competing code processor fuzzers:
 - CSmith (language-specific C compiler fuzzer)
 - AFL++, AFL-Compiler-Fuzzer, Polyglot (gen.-purpose)

Evaluation: Overview

- **Fundamental questions:**
 1. Is TeTRIS capable of **generating valid inputs?**
 2. Can TeTRIS **more thoroughly** test transpilers?
 3. Does TeTRIS uncover **more translation bugs?**
- Competing code processor fuzzers:
 - CSmith (language-specific C compiler fuzzer)
 - AFL++, AFL-Compiler-Fuzzer, Polyglot (gen.-purpose)
- Benchmarked on **seven transpilers** spanning **six distinct** language-to-language pairings
 - Standard evaluation procedure ($5 \times 24\text{hr}$ trials each)

Transpiler	In Lang.	Out Lang.
C2Rust	C	Rust
CxGo, C4go	C	Go
Zig Translate-C	C	Zig
Go2Hx	Go	Haxe
HxCpp	Haxe	C++
HxPy	Haxe	Python3

Evaluation: Generated Code Validity

- Measured **mean generated code validity** across all fuzzers per transpiler
 - Proportion of each fuzzer's 24-hr corpora accepted as valid by each targeted transpiler
 - Evaluated on transpilers compatible with grey-box fuzzing (CxGo, C4Go, C2Rust, HxCpp, HxPy)

Evaluation: Generated Code Validity

- Measured **mean generated code validity** across all fuzzers per transpiler
 - Proportion of each fuzzer's 24-hr corpora accepted as valid by each targeted transpiler
 - Evaluated on transpilers compatible with grey-box fuzzing (CxGo, C4Go, C2Rust, HxCpp, HxPy)

Metric	TeTRIS vs. CSmith
Code Validity	- 0.25×

Evaluation: Generated Code Validity

- Measured **mean generated code validity** across all fuzzers per transpiler
 - Proportion of each fuzzer's 24-hr corpora accepted as valid by each targeted transpiler
 - Evaluated on transpilers compatible with grey-box fuzzing (CxGo, C4Go, C2Rust, HxCpp, HxPy)

Metric	TeTRIS vs. CSmith	TeTRIS vs. Polyglot	TeTRIS vs. AFL-Comp-Fuzz	TeTRIS vs. AFL++
Code Validity	- 0.25×	+ 5.24×	+ 27.42×	+ 30.69×

- Takeaways:** TeTRIS balances **code validity and broad transpiler support**
 - Comparable performance to C-specific CSmith despite supporting many more languages

Evaluation: Transpiler Code Coverage

- Measured **mean transpiler code coverage** for all fuzzers per transpiler
 - Enumerated basic block coverage of transpiler binaries using AFL-QEMU binary tracing
 - Evaluated on transpilers compatible with QEMU coverage tracing (CxGo, C2Rust, HxCpp, HxPy)

Evaluation: Transpiler Code Coverage

- Measured **mean transpiler code coverage** for all fuzzers per transpiler
 - Enumerated basic block coverage of transpiler binaries using AFL-QEMU binary tracing
 - Evaluated on transpilers compatible with QEMU coverage tracing (CxGo, C2Rust, HxCpp, HxPy)

Metric	TeTRIS vs. CSmith
Code Coverage	- 10.25%

Evaluation: Transpiler Code Coverage

- Measured **mean transpiler code coverage** for all fuzzers per transpiler
 - Enumerated basic block coverage of transpiler binaries using AFL-QEMU binary tracing
 - Evaluated on transpilers compatible with QEMU coverage tracing (CxGo, C2Rust, HxCpp, HxPy)

Metric	TeTRIS vs. CSmith	TeTRIS vs. Polyglot	TeTRIS vs. AFL-Comp-Fuzz	TeTRIS vs. AFL++
Code Coverage	- 10.25%	+ 11.31%	+ 23.62%	+ 21.70%

- Takeaways:** TeTRIS probes more of transpilers' **overall translation logic**
 - Trades-off some **transpiler-specific depth** for covering a **broader range of behaviors**

Evaluation: Discovered Transpiler Bugs

- Analyzed **total unique bugs** throughout all transpiler fuzzing campaigns
 - Bug categories:** transpiler crashes, post-translation crashes, divergent runtime outputs
 - All bugs manually deduplicated and reported to their respective transpiler developers

Evaluation: Discovered Transpiler Bugs

- Analyzed **total unique bugs** throughout all transpiler fuzzing campaigns
 - Bug categories:** transpiler crashes, post-translation crashes, divergent runtime outputs
 - All bugs manually deduplicated and reported to their respective transpiler developers

Metric	CSmith
Total (old +new)	2

Evaluation: Discovered Transpiler Bugs

- Analyzed **total unique bugs** throughout all transpiler fuzzing campaigns
 - Bug categories:** transpiler crashes, post-translation crashes, divergent runtime outputs
 - All bugs manually deduplicated and reported to their respective transpiler developers

Metric	CSmith	Polyglot	AFL-Comp-Fuzz	AFL++	TeTRIS
Total (old +new)	2	0	0	0	16

Evaluation: Discovered Transpiler Bugs

- Analyzed **total unique bugs** throughout all transpiler fuzzing campaigns
 - Bug categories:** transpiler crashes, post-translation crashes, divergent runtime outputs
 - All bugs manually deduplicated and reported to their respective transpiler developers

Metric	CSmith	Polyglot	AFL-Comp-Fuzz	AFL++	TeTRIS
Total (old +new)	2	0	0	0	16
Newly- Found	0	0	0	0	12

- Takeaways:** TeTRIS's diverse test cases expose **more transpiler defects**
 - Since submission, **all 12** TeTRIS-found new bugs **are confirmed and/or fixed**

Case Studies: TeTRIS-found Transpiler Bugs

- **CxGo #76:** unhandled **bool-to-float implicit type conversion**
 - CxGo adds **external functions** to accommodate Go's lack of implicit type conversion

```
float x = 1.25;  
  
x += 100 + (4 < 4.5);  
  
x += 9.9 + (4 < 4.5);
```

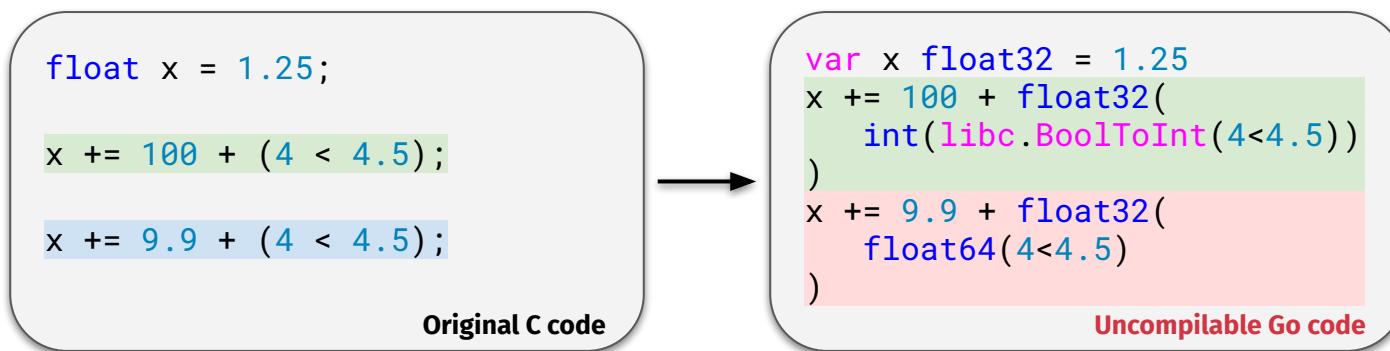
Original C code



```
var x float32 = 1.25  
x += 100 + float32(  
    int(libc.BoolToInt(4<4.5))  
)  
x += 9.9 + float32(  
    float64(4<4.5)  
)
```

Case Studies: TeTRIS-found Transpiler Bugs

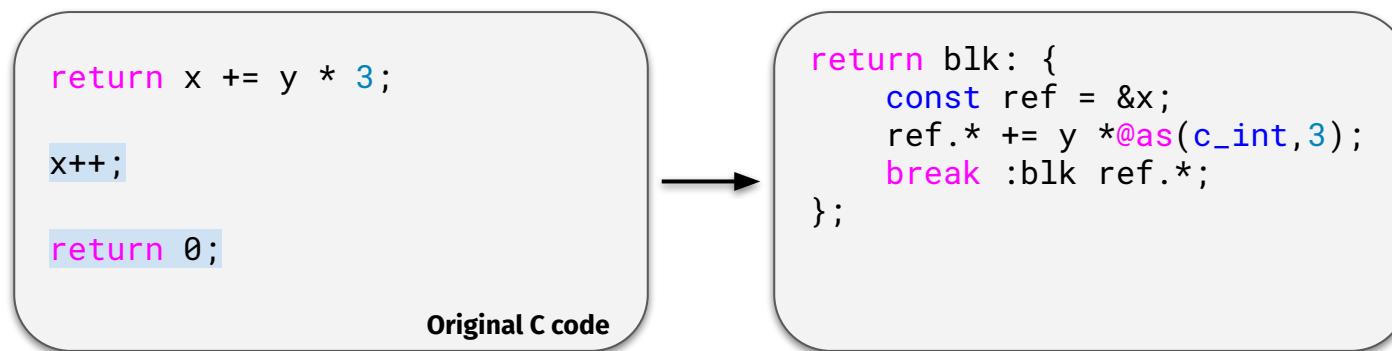
- **CxGo #76:** unhandled **bool-to-float implicit type conversion**
 - CxGo adds **external functions** to accommodate Go's lack of implicit type conversion
 - Although bool-to-integer conversions supported, CxGo **missed conversions to floats**



- **Takeaways:** real-world codebases span a **wide range of type semantics**
 - TeTRIS's random programs surface mishandling of **expected type-to-type conversions**

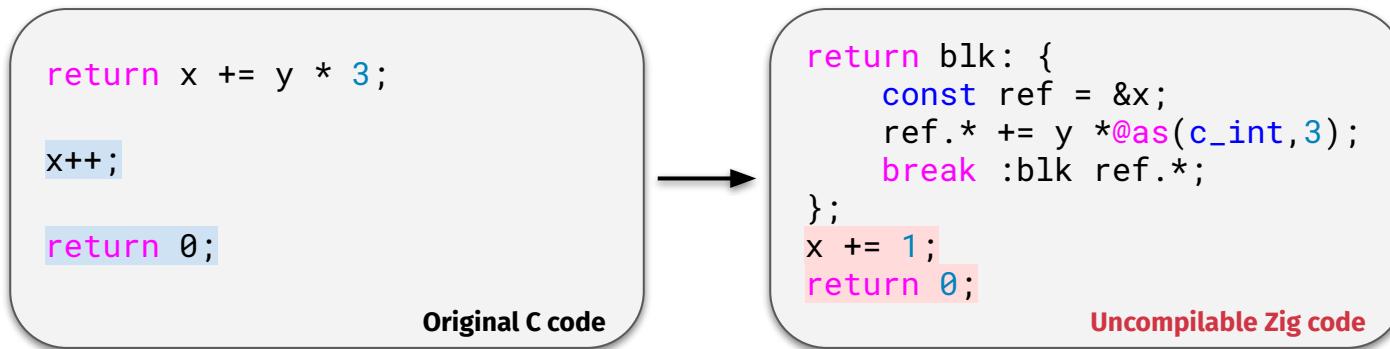
Case Studies: TeTRIS-found Transpiler Bugs

- **Zig Translate-C #21871:** mis-inclusion of **unreachable code statements**
 - While **unreachable code** is permitted in C, it is strictly prohibited in the Zig language



Case Studies: TeTRIS-found Transpiler Bugs

- **Zig Translate-C #21871:** mis-inclusion of **unreachable code statements**
 - While **unreachable code** is permitted in C, it is strictly prohibited in the Zig language
 - Zig Translate-C correctly prunes most unreachable statements, **but wrongly kept these**



- **Takeaways:** transpilers' **internal code analyses** are also error-prone
 - TeTRIS uncovers how these analyses fail toward **enhancing transpilers' reliability**

Case Studies: TeTRIS-found Transpiler Bugs

- Go2Hx #179: non-equivalent **comparisons between zeroed arrays**
 - Go supports **value-level array comparisons**, while Haxe only by reference

```
x := [4]int{0,0,0,0}
if x != [4]int{0,0,0,0} {
    panic("ERR: unequal!")
}
```

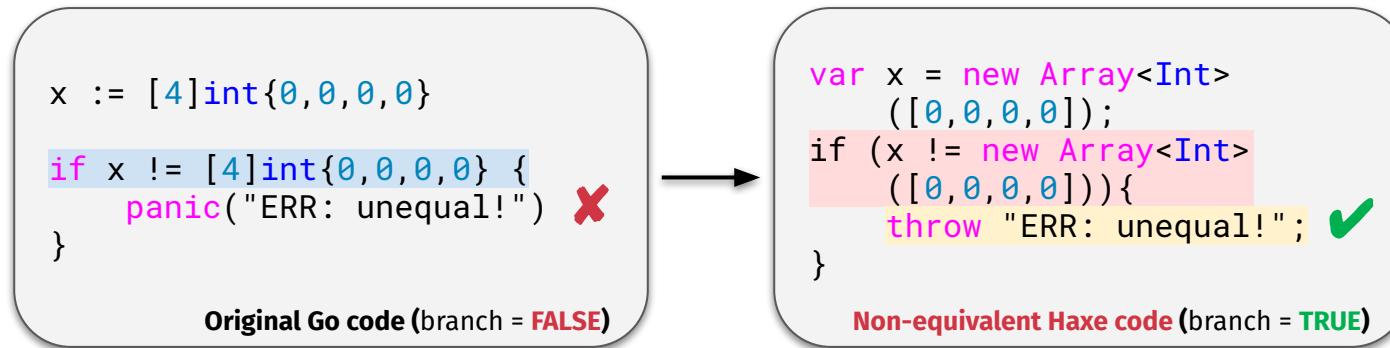
Original Go code (branch = FALSE)



```
var x = new Array<Int>
([0,0,0,0]);
if (
)
{
    throw "ERR: unequal!";
}
```

Case Studies: TeTRIS-found Transpiler Bugs

- **Go2Hx #179:** non-equivalent **comparisons between zeroed arrays**
 - Go supports **value-level array comparisons**, while Haxe only by reference
 - Go2Hx fails to properly account for this, resulting in **divergent executions**



- **Takeaways:** transpilers need to resolve **language-level feature differences**
 - TeTRIS's coverage of diverse code constructs surfaces **subtle translation edge-cases**

Conclusion: Why TeTRIS?

- Transpiler bugs **impede translation-oriented tasks**
 - Code migration, interoperability, and security retrofitting
 - Hence, **testing transpilers proactively** is key to fixing them
- Prior fuzzers are **ineffective on today's transpilers**
 - Language-specific fuzzers (e.g., CSmith) remain **inflexible**
 - General-purpose fuzzers (e.g., Polyglot) face **invalid code**
 - Hence, vast majority of transpiler logic **remains untested**

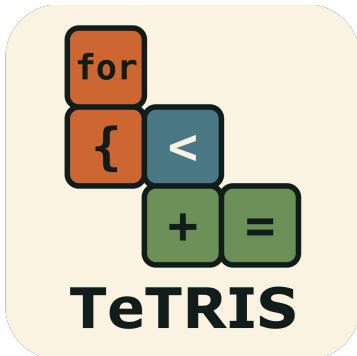
Conclusion: Why TeTRIS?

- Transpiler bugs **impede translation-oriented tasks**
 - Code migration, interoperability, and security retrofitting
 - Hence, **testing transpilers proactively** is key to fixing them
- Prior fuzzers are **ineffective on today's transpilers**
 - Language-specific fuzzers (e.g., CSmith) remain **inflexible**
 - General-purpose fuzzers (e.g., Polyglot) face **invalid code**
 - Hence, vast majority of transpiler logic **remains untested**
- Our solution: **TeTRIS**
 - The first fuzzer designed specifically for transpilers' needs
 - **Outcome:** effective fuzzing for today's diverse transpilers, spanning **C**-, **Go**-, and **Haxe**-targeting translation workflows

Key Results:

5–30× higher validity
11–23% more coverage
12 new transpiler bugs
(all since confirmed)

Thank you!



github.com/FuturesLab/TeTRIS

Contact:

✉️ y.arafat@utah.edu

𝕏 [@yeaseen_](https://twitter.com/yeaseen_)

🦋 [@yeaseen.bsky.social](https://bsky.social/@yeaseen)

FUTURES³
LAB FUTURE TECHNOLOGY FOR USABLE, RELIABLE, &
EFFICIENT SECURITY OF SOFTWARE & SYSTEMS