Finding Bugs in Compilers for Programmable Packet Processing

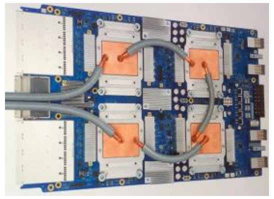
Fabian Ruffy, Tao Wang, and Anirudh Sivaraman

p4gauntlet.github.io

NEW YORK UNIVERSITY

Computation is Moving to Accelerators

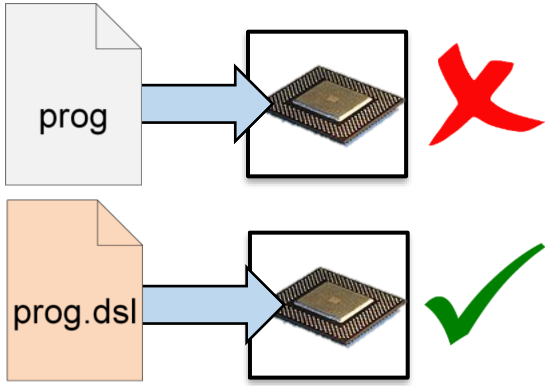
- **Accelerators** for machine learning
  - Google TPU, Intel VPU
- **Cloud FPGAs** for specialized tasks
  - Microsoft Catapult, Amazon F1
  - DPUs (Fungible, NVIDIA Bluefield)
- **Programmable Networks**
  - SmartNICs (Xilinx, Pensando)
  - Programmable switch chips (Barefoot, Cisco, Broadcom)



Accelerators and Domain-Specific Languages

- Force the developer to **"think" in accelerator-specific abstractions**

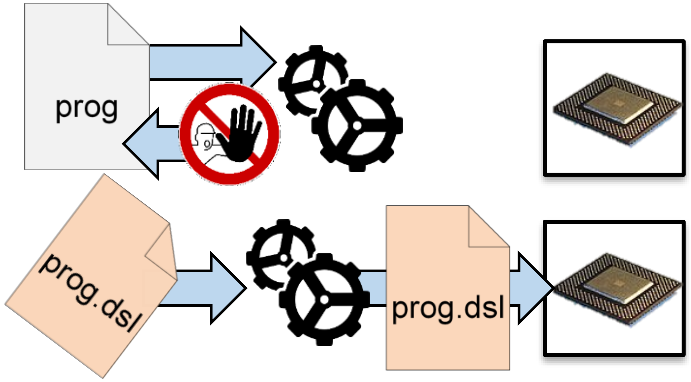
- **Examples**
  - TensorFlow HLO for deep learning models
  - P4/NPL for packet processing



Consequence → DSLs are often constrained and **not** Turing-complete!

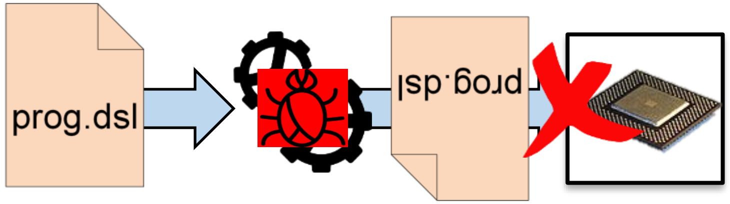
Compilers and Domain-Specific Languages

- DSLs typically require a custom compiler, which…
  - enforces the restrictions for the target accelerator
  - translates high-level spec into device-specific instructions
  - applies domain-specific optimizations
- Increase in accelerators leads to…
  - …more domain-specific compilers to deal with



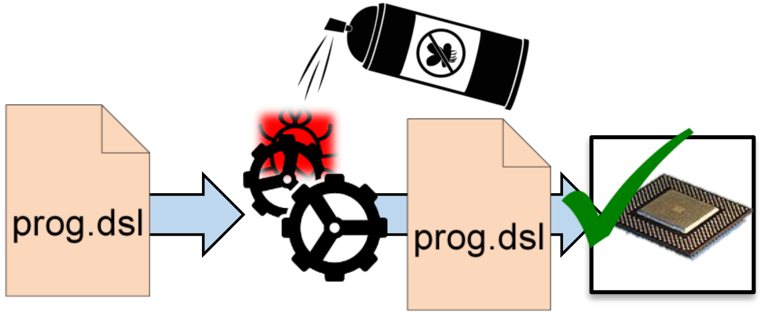
What about Bugs in These Compilers?

- Compilers for these DSLs may have bugs
  - Newer → not as well-tested as general-purpose GCC, LLVM, ICC
  - Often compile for mission-critical paths → high impact of faults
  - applies domain-specific optimizations
- How do we make sure that these compilers are reliable?



Exploit Constrained DSLs!

- **Observation**: DSLs only need to express restricted functionality
- If we constrain our DSL just right we can…
  - …efficiently apply formal methods
  - …revive old techniques from compiler and testing literature



Our Work: Bug-finding Techniques for $P4_{16}$

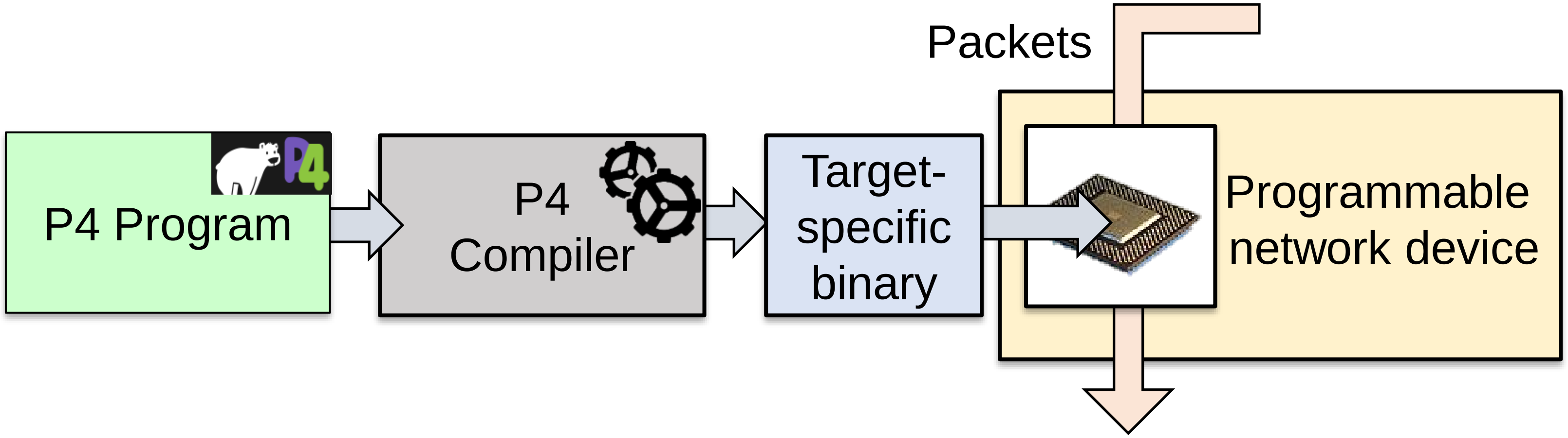
- We describe
  - How to find bugs in compilers for the $P4_{16}$ DSL
  - How we revive old compiler techniques to find bugs
  - **Gauntlet**, our tool suite that finds bugs in $P4_{16}$ compilers

Broader Takeaways

- Designing a DSL well can lead to effective analysis tools
  - Limiting undefined behavior eases code generation
  - Restrictions make expressive semantics possible
- $P4_{16}$ is such a semantics-friendly DSL. This helped us...
  - …identify **more than 90** bugs within eight months of testing
  - …apply translation validation at scale without false positives
  - …integrate translation validation into the CI pipeline of P4C

What is P4?

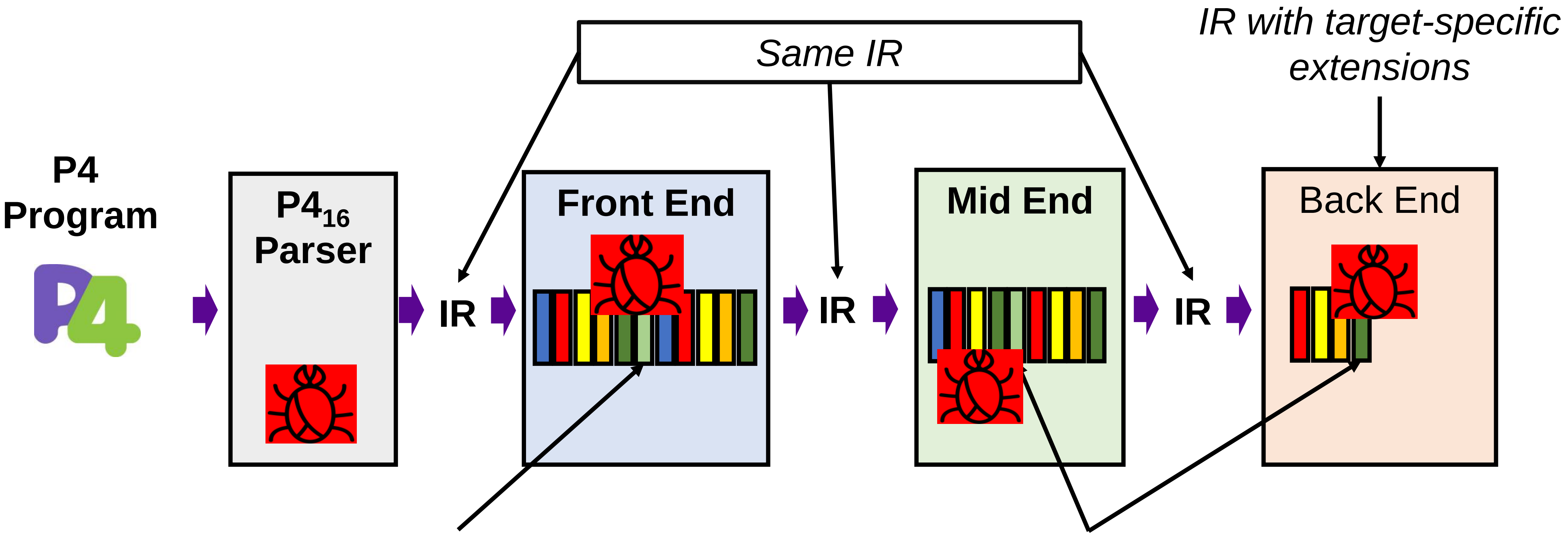
- DSL for network data planes
  - Specifies how an incoming packet header is parsed
  - Allows the implementation of custom network protocols
- Open and **standardized**



P4: Current Landscape

- Back ends
  - Intel (Barefoot) Tofino, Cisco Silicone One, Xilinx Alveo
- Users
  - Google, Broadcom, Nokia, Orange…
- $P4_{16}$ DSL has a reference compiler: P4C
  - Has status similar to LLVM/GCC; represents the P4 spec
  - P4C transforms input → streamlines and optimizes code

Compiler Context: P4C



IR with target-specific extensions

Same IR

P4 Program

P4$_{16}$ Parser

IR

Front End

IR

Mid End

IR

Back End

Target-independent compiler passes (25+ distinct transformations)

Target-specific compiler passes

IR: Intermediate Representation

Stages of Testing a Compiler

Increased Precision

LEVEL	INPUT CLASS	CAN THE COMPILER HANDLE…
1	Sequence of ASCII characters	…large input sizes?
2	Sequence of words, etc.	…an invalid token?
3	Syntactically correct program	… a missing bracket?
4	Type-correct program	…adding int to a struct?
5	Statically conforming program	… a variable that is not defined?
6	Dynamically conforming program	…transforming expressions?

Differential testing for software., McKeeman, William M., *Digital Technical Journal, 1998*

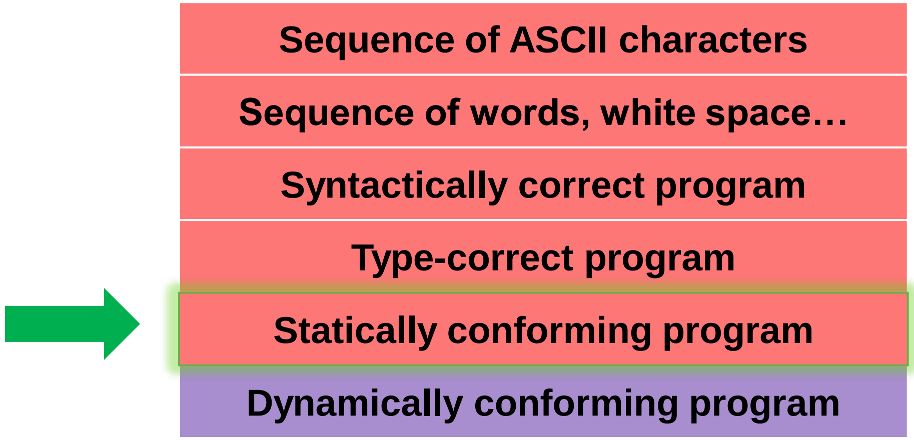
Two Types of Bugs

- Crash Bug
  - “Obvious” bug
  - Program that causes the compiler to exit abnormally
  - All bugs up to level **5**
- Miscompilation or “Semantic Bug”
  - No error raised, **but** behavior of program is altered
  - Typically caused by misbehaving compiler passes
  - Level **6**

Sequence of ASCII characters
Sequence of words, white space…
Syntactically correct program
Type-correct program
Statically conforming program
Dynamically conforming program

How to Crash the Compiler?

- **Random** programs
- We target level 5
  - Generate random programs that are valid



Sequence of ASCII characters

Sequence of words, white space…

Syntactically correct program

Type-correct program

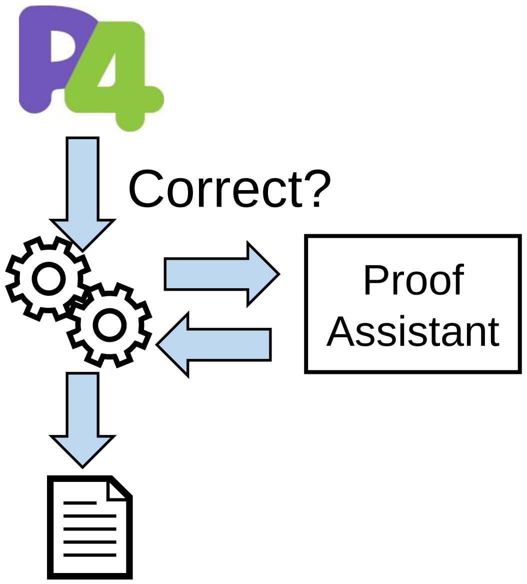
Statically conforming program

Dynamically conforming program

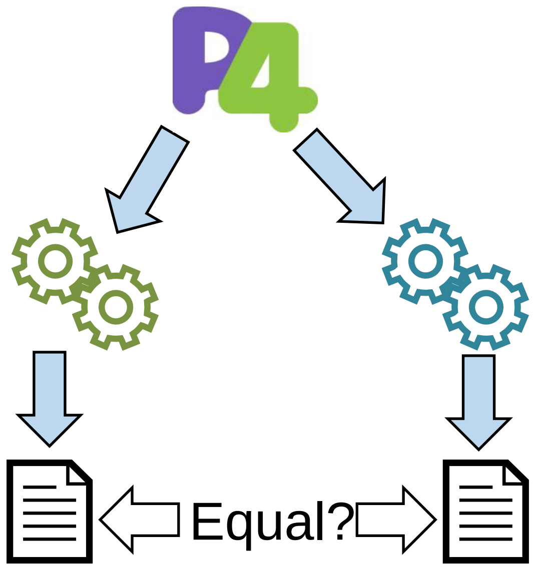
- Identify programs that cause a non-zero exit code
  - Could also be a program that is incorrectly rejected
- **Bonus**: Use the generated programs to find semantic bugs

Handling Semantic Bugs in the Compiler

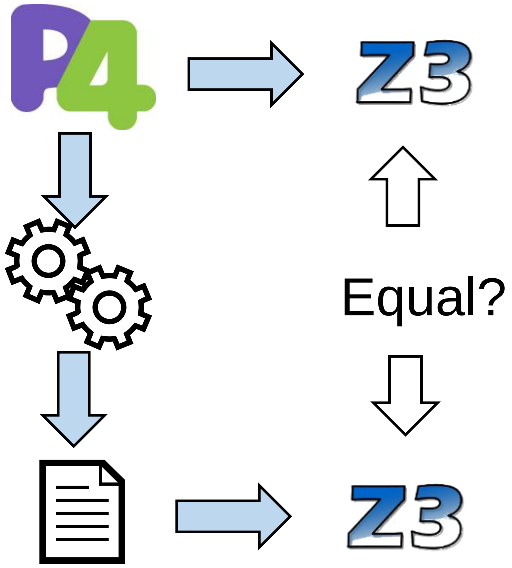
Compiler Verification? ✗



Differential Testing? ✗



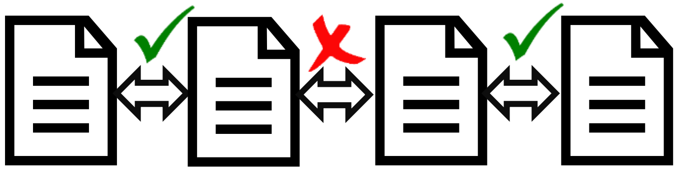
Translation Validation? ✓



Why does Translation Validation Work for P4?

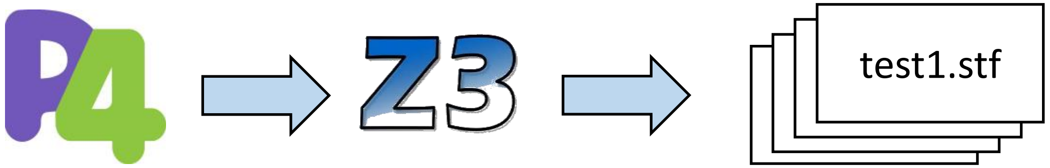
- Historically limited because of undecidability
- **But**, P4’s properties are a great fit for formal methods
  - Language core not Turing-complete
  - Program-structure provides well-defined state
    - Input/output and state known at program start

We can compare entire programs!



Model-Based Testing

- Cannot use translation validation for closed-source compilers
  - No access to the IR
  - Output binary obfuscated and semantics unknown
- Idea: Reuse program semantics to infer input and output
  - Requires end-to-end test framework
  - Input/output pairs are computed based on program branches



The Gauntlet Framework for P4

- Toolbox of testing software
  - Random code generator
  - Interpreter that converts $P4_{16}$ to Z3
  - Translation validation and testing pipeline
- Three concrete techniques for finding bugs
  1. Random code generation to find crash bugs
  2. Translation validation to identify semantic bugs
  3. **Bonus**: Model-based testing for closed-source compilers

Normalized Z3 Semantics: Example

P4 Program

```
struct Hdr {bit<48> mac_dst;
            bit<48> mac_src;
            bit<16> eth_type; }

control in(inout Hdr hdr, in bit<8> flag) {
    main {
        if (flag = 0) {
            hdr.eth_type = Ox800; // IPv4
        } else {
            hdr.eth_type = 0x86DD; // IPv6
        }
    }
}
```

Semantic Representation

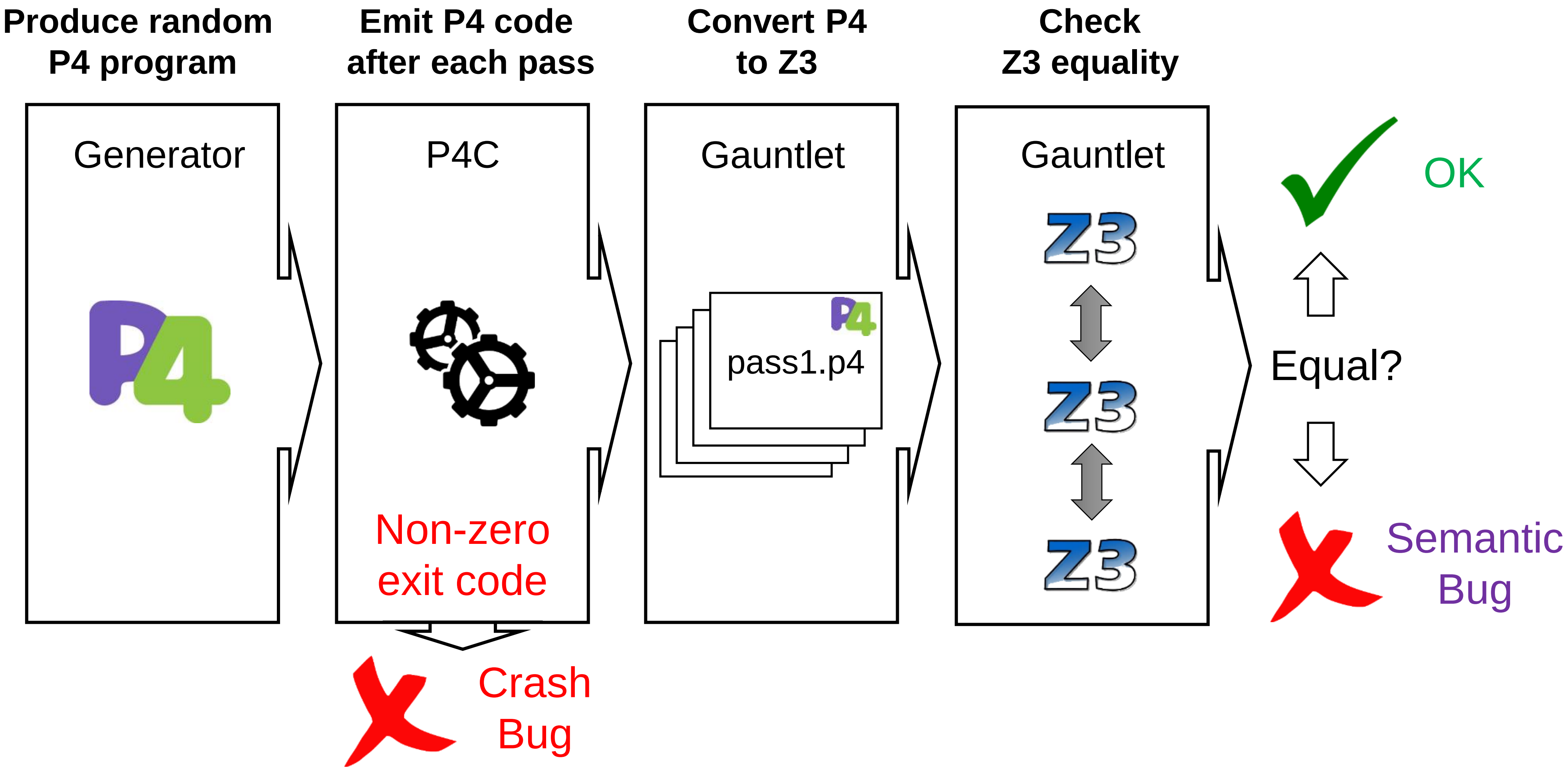
```
Symbolic input: hdr, flag
Symbolic output: hdr_out

hdr_out = if (flag == 0):
                  Hdr(hdr.mac_dst,
                      hdr.mac_src,
                      0x800)
          otherwise:
                  Hdr(hdr.mac_dst,
                      hdr.mac_src,
                      0x86DD)
```

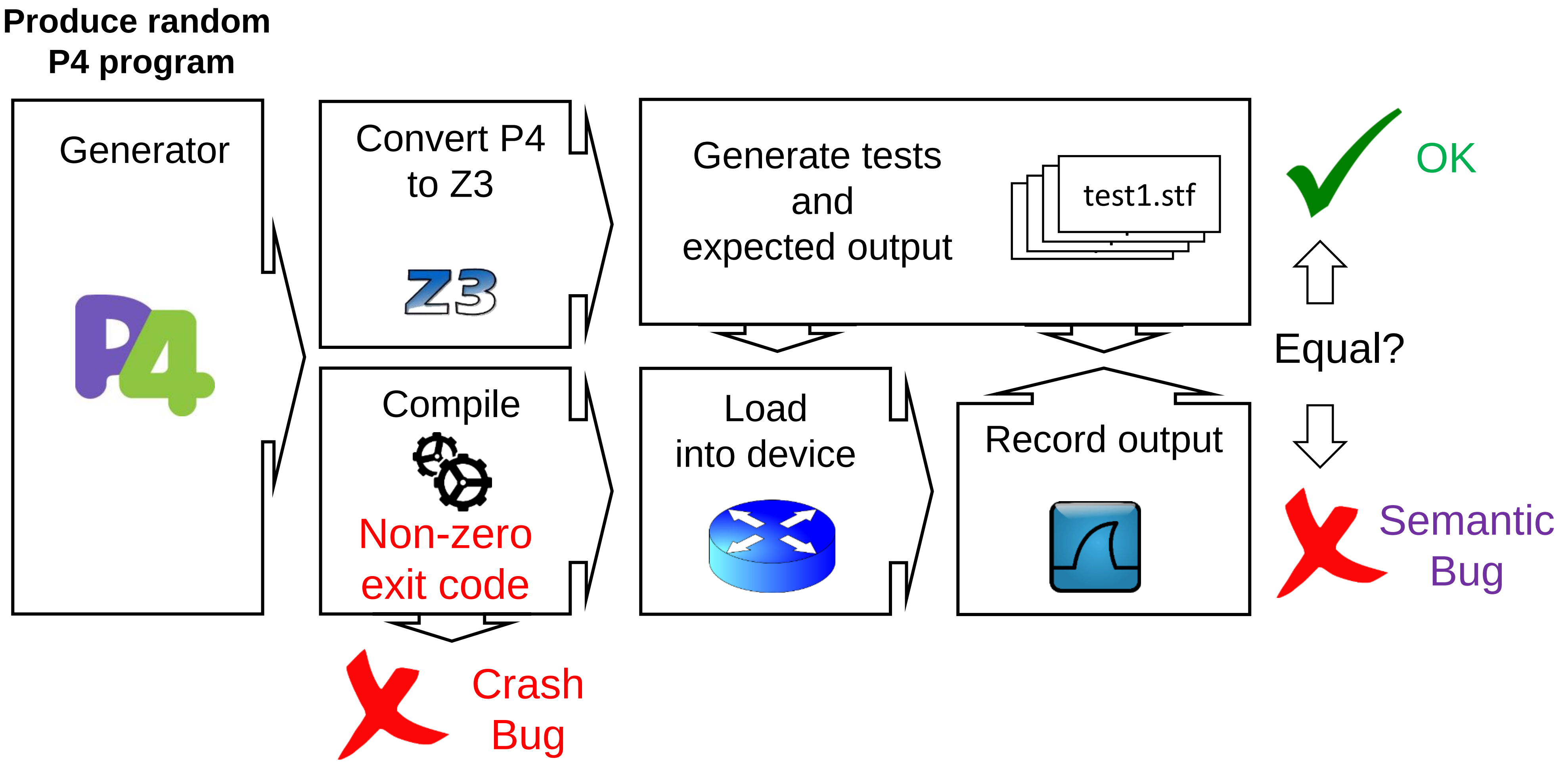
Generating a Random Program for P4C

- Program generator modelled after Csmith
    - But does **not** avoid undefined behavior → Simpler
- “Grow” the AST by picking from legal P4$_{16}$ expressions
- Code generation is guided by P4$_{16}$ specification
    - A correctly rejected, generated program is a bug in our tool
- **Small** fragments of the language sufficient to detect bugs
    - Branching is limited → Performance not a concern

The Gauntlet Validation Workflow



Bonus: Model-Based Testing



Results

- Found **96** compiler bugs in 8 months
  - **62** compiler crashes (25/62 in the compiler for the Tofino network chip)
  - **34** semantic bugs (7/34 in the compiler for the Tofino network chip)
- Some observations
  - Crashes were largely caused by an assertion firing
  - Handling side-effects correctly is difficult
- Resulted in **6** specification changes

Future Work

- Develop semantics for instruction set architectures
  - Extend translation validation to back ends
  - Ensure correctness during the entire compilation process
- Detect other classes of compiler bugs
  - Identify when an optimization should have been applied
  - Identify compiler passes negatively affecting performance
  - Again repurpose techniques from compiler testing literature

Summary

- A well-designed device DSL can lead to effective analysis tools
- P4 is a semantics-friendly DSL, which helped us build **Gauntlet**
- With **Gauntlet** we were able to...
  - …apply translation validation at scale without false positives
  - …identify **more than 90** bugs within eight months of testing
  - …integrate translation validation into the CI pipeline of P4C

Thank you
for listening!

Contact

Fabian Ruffy (fruffy@nyu.edu)
Tao Wang (tw1921@nyu.edu)
Anirudh Sivaraman (anirudh@cs.nyu.edu)

Project Repository

p4gauntlet.github.io