RIT | Golisano College of Computing and Information Sciences

P4 Safety Verification

Mark Drobnak (mtd8050@rit.edu) | Advisor: Dr. Minseok Kwon (jmk@cs.rit.edu)

Introduction and Background

- P4 is a domain specific language for programming network switches in SDNs. P4 enables rapid experimentation and deployment of network protocols.
- Networking requirements are rapidly changing and rising in complexity. Software Defined Networking (SDN) introduces opportunities for advancing network protocols.
- P4 programs can have bugs and safety issues which cause undesirable behavior in SDN systems.
- p4-analyzer is introduced to detect these issues in P4 programs.

Undesirable P4 Behaviors

- P4 programs can have many undesirable behaviors:
  - **Use-before-initialization**: Using a variable before it has been initialized with a value.
  - **Header-use-before-valid**: Reading from a header before it has been read from the input packet.
  - **Undefined-forwarding**: Failing to forward or drop a packet.
  - ...and more
- Most of these issues can be detected statically.

p4-analyzer

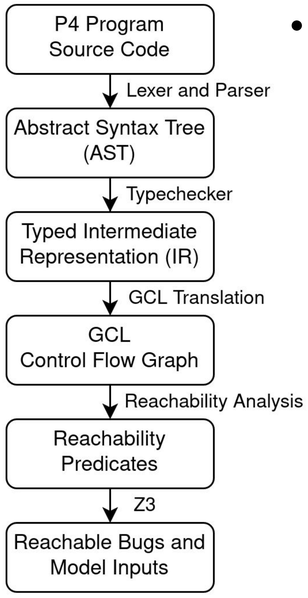
```
struct metadata_t { bool value; }

control my_control(inout metadata_t meta) {
    bool var;

    apply {
        if (meta.value) {
            // Bug: 'var' is uninitialized
            meta.value = var;
        }
    }
}
```

Figure 1. A P4 program with a bug.

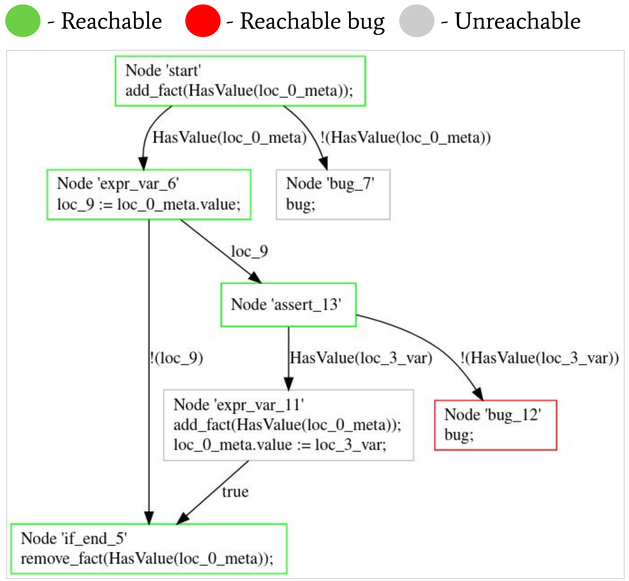
p4-analyzer



- p4-analyzer merges good ideas from previous works:
  - From p4v [3]: Translation to **GCL**.
    - Allows for a more semantic analysis of P4 programs.
  - From bf4 [2]: Translation to a **Control Flow Graph** (CFG) and **bug node injection**.
    - During GCL translation, bug nodes are inserted whenever undesirable behavior could occur.
  - From Petr4 [1]: The **type checker** and some of the **GCL translation** are based on the static and dynamic P4 semantics introduced by Petr4
    - The formal foundations from Petr4 give strong guarantees about the correctness of p4-analyzer.

Figure 2. The flow of data in p4-analyzer.

Analysis of the Figure 1 P4 Program



Reachability Predicates

start = true
bug_7 = false
bug_14 = loc_0_meta.value

Z3 Model for bug_14

loc_0_meta = (metadata_t true)

GCL Model for bug_14

loc_0_meta = { value = true }

P4 Model for bug_14

meta = (metadata_t) { value = true }

Figure 3. Computed GCL CFG, reachability predicates, and model of buggy input.

Contributions

- p4-analyzer can detect non-trivial bugs in P4 programs.
  - Its foundation is **theoretically stronger** than previous work due to the combination of semantic analysis and use of Petr4's formal type system calculus.
- The translation of reachability predicates to Z3 includes a **novel translation** of advanced P4 types into Z3 types.
  - Struct types are translated into Z3 data types. This simplifies translation to and from Z3.
  - Example:

```
struct metadata_t {
    bool value;
}
Z3: meta = (metadata_t true)
P4: meta = (metadata_t) { value = true }
```

- Advanced uninitialized variable bug detection
  - The p4c reference compiler does simple uninitialized variable usage detection, but fails in more advanced cases.
  - bf4, being based off of p4c, inherits this simple bug detection.
  - p4-analyzer introduces an analysis of P4 programs which detects non-trivial instances of uninitialized variable usage.

Conclusions and Future Work

- p4-analyzer performs advanced static analysis of P4.
- Support more of P4, including integers, bit strings, and headers.
  - p4-analyzer currently only supports a subset of P4.
    - Control blocks which use booleans and structs.
- Analyze P4 match-action tables and infer a control plane interface.
  - This would enable detecting control-plane-interface bugs.
- Investigate syntactic vs semantic analysis of P4.

RIT | Rochester Institute of Technology

References

1. Doenges et al. (2021). Petr4: Formal Foundations for P4 Data Planes. *Proc. ACM Program. Lang.*, *5*(POPL). https://doi.org/10.1145/3434322
2. Dumitrescu et al. (2020). Bf4: Towards Bug-Free P4 Programs. *Sigcomm '20*, 571–585. https://doi.org/10.1145/3387514.3405888
3. Liu et al. (2018). P4v: Practical Verification for Programmable Data Planes. *Sigcomm '18*, 490–503. https://doi.org/10.1145/3230543.3230582

Source Code: https://github.com/AzureMarker/p4-analyzer