Using GNN for refactoring P4 programs

Benedek Csüllög – Máté Tejfel Dr.

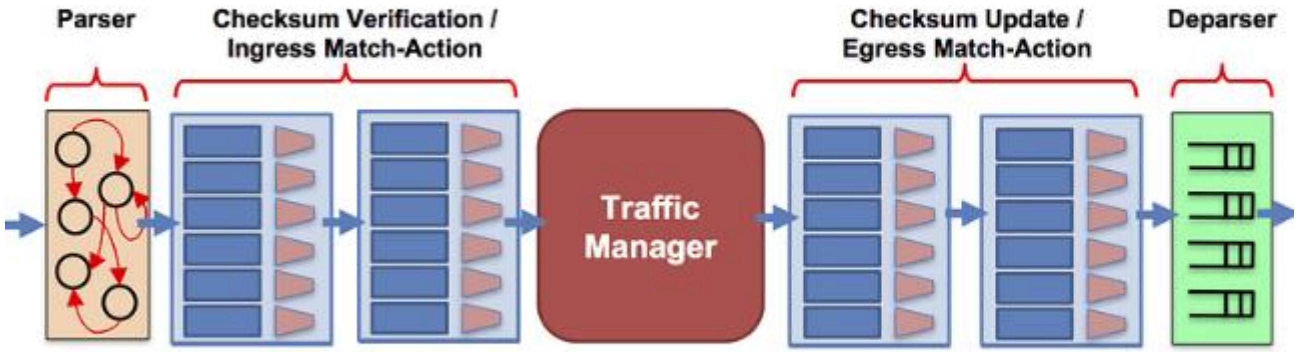
FMF-AI 2025

Content

- Introduction to the P4 Programming Language
- The P4Query Framework
- Motivation
- Basics of GNNs and Their Training
- Current GNN Models and Their Capabilities
- Possible Future Work
- Summary

Programming Protocol-independent Packet Processors - P4

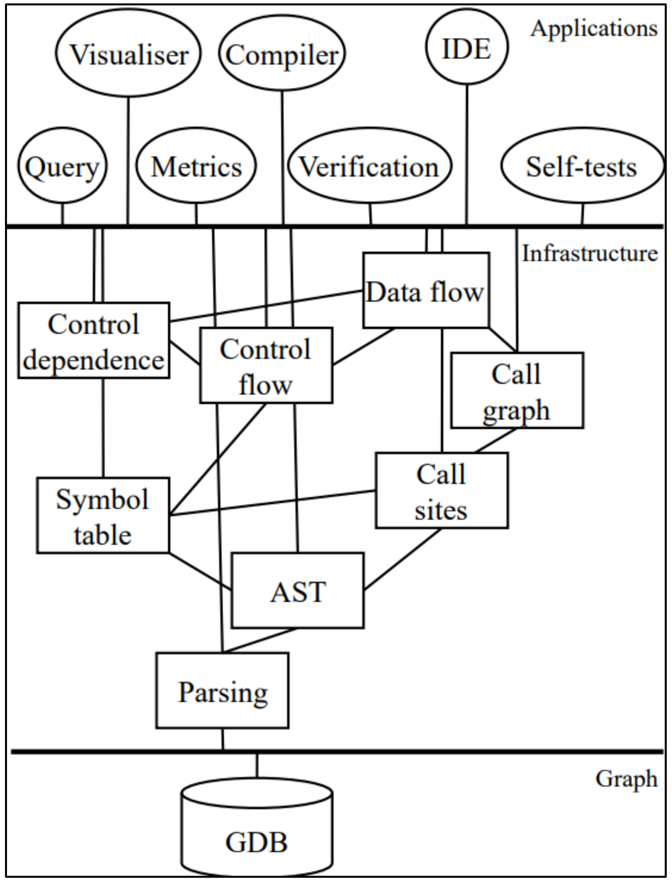
- **Domain-specific programming language for data-plane packet processing.** – P4 lets you describe how a switch/router should parse, match and modify packets at line rate.
- **Match/Action pipeline** – Packets flow through programmable tables that *match* on header fields or metadata and then execute *actions*.
- **Header and metadata updates** – Actions can add, remove or rewrite header fields and adjust metadata, enabling in-network functions such as tunnelling, load-balancing or telemetry.



Source: [P4 architecture - P4 Programming Language]

P4 Query

- **Static analysis framework** for P4
- Contains an internal Gremlin **graph representation**
- Ensures a consistent and **standardized representation** of P4 programs
- P4 programs can naturally be represented as **abstract syntax trees (ASTs)**, generated by the framework



Motivation

Abstract Syntax Trees (**ASTs**) can be naturally represented as directed graphs.

Given this structure, a key question arises:

Can machine learning models learn from both the topology of such graphs and the semantic attributes of their nodes?

Recent advances in deep learning provide a positive answer.

Graph Neural Networks (GNNs) have proven highly effective in learning from graph-structured data, making them suitable for various code understanding and transformation tasks, including refactoring.

Basics of GNNs

- **Graph Neural Networks** are designed to work with graph data structure.
- In these graphs, **nodes represent entities**, and **edges represent relationships** between them.
- GNNs learn node (or graph-level) representations by **iteratively aggregating information from neighboring nodes**.
- The input is a graph, and the output is either:
  - a modified graph (e.g., refactored),
  - or a prediction (e.g., classification or embedding).
- GNNs are well-suited for tasks involving **structural understanding**, such as code refactoring.

Training GNNs

- The model incrementally learns the structure of the AST across multiple training epochs.
- In each epoch, specific portions of the graph are selectively removed to enhance the model's ability to generalize its understanding of the structure.
- Initially, the training may involve the deletion of leaf nodes, which represent the terminal elements of the AST.
- By starting with these simpler nodes, the model is encouraged to focus on the relationships and interactions among the remaining nodes.
- This leads to a deeper understanding of the overall tree structure.
- As the training advances, the strategy can adapt to remove more complex nodes or entire subtree.

Current models

- **Variable renamer**
  - Renames every instance of a given variable in the AST (declaration, initialisation etc.). During running the model the variable to be renamed comes as parameter. Modifies the AST node embeddings.
- **Parameter reorderer**
  - Able to reorder parameter declarations in a function declaration. Modifies the AST's structure.
- **Empty 'else' block detector**
  - Able to recognize an empty else block in an AST.

Trained models – Variable renamer

Goal

- Rename a variable and all of its occurrences in the AST graph.
- Useful for improving readability or resolving naming conflicts.

Training

- A 2-layer GNN is trained to classify node attributes (e. g. ’class’).

```
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}
```

```
header ethernet_t {
    macAddr_t destinationAddress;
    macAddr_t sourceAddress;
    bit<16>   ethernetType;
}
```

Trained models – Variable renamer

Outcome

- The model learns structural and contextual patterns for variable usage.
- Able to reliably rename all instances of a given variable in the graph.

```
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}
```

```
header ethernet_t {
    macAddr_t destinationAddress;
    macAddr_t sourceAddress;
    bit<16>   ethernetType;
}
```

Trained models – Parameter reorderer

Goal

- Reorder function parameters to improve consistency or readability.

Training

- Match parameters based on (type, name) signatures.
- Learn node embeddings and use a feedforward neural network to predict target index positions.

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

```
control MyDeparser( in headers hdr, packet_out packet) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

Trained models – Parameter reorderer

Outcome

- The model predicts a parameter order for a given function.
- Modifies the AST graph based on the predicted permutation.

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

```
control MyDeparser( in headers hdr, packet_out packet) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

Trained models – Empty ’else’ block detector

Goal

- Detect empty 'else' blocks in P4 abstract syntax trees. Identify redundant branches.

Training

- Extract ’else’ nodes and label them as empty or non-empty.
- Node features include class and token value encodings.
- 2-layer GNN performs binary classification, focusing only on ’else’ nodes.

```
if (hdr.ipv4.isValid()) {
    ipv4_lpm.apply();
} else {}
```

Trained models – Empty ’else’ block detector

Outcome

- Learns to classify ’else’ blocks based on structural patterns.
- During inference, outputs predicted class and confidence for each ’else’ node.

```
if (hdr.ipv4.isValid()) {
    ipv4_lpm.apply();
} else {}
```

Possible future work

- Recognize nested 'if' statements that can be collapsed.
- Modify the AST and fulfill header fields automatically.

```
if (hdr.ipv415.isValid()) {
   if (hdr.ipv416.isValid()) {
       ipv415_lpm.apply();
       ipv416_lpm.apply();
   } else {}
} else {}
```

Summary

- **Unified graph representation of P4 programs** using the *P4Query* framework.
- 3 GNN-based model for refactoring:
  - Variable renamer
  - Parameter reorderer
  - Empty ’else’ block detector
- The models learns different apects od the ASTs and try to do modifications or predictions.

Thank you for you attention!