MNK Labs & Consulting

A Few Extensions to P4 for Code Modularity

Alex Bernstein and Hemant Singh <alex/hemant@mnkcg.com>

MNK Labs & Consulting

https://mnkcg.com

Background

- Code reuse: want to be able to extend P4 programs without the need to rewrite the base program
  - e.g. parsers, individual parser states, controls, structs, headers…
  - recall Vladimir's presentation back in June
- This presentation will present a few potential extensions to the P4-16 language which were originally designed for extending parsers, but may have use with other constructs

Potential P4 extensions

- New `override` keyword
  - tells compiler to extend various P4 constructs (e.g., parser, parser state, struct, control, package, etc.)
- New `super` keyword
  - In parser state transition block, use `super.`*`state`*`.transition`
    - *`state`* represents name of a base parser state; uses base transition code without using the extract code
  - `super` can be used for other constructs (e.g. control)…

Examples: Parser `override`

```
// tells compiler to patch previously
// defined parser with the same name ("base_parser")
parser base_parser(…) override {
    state ethernet override {  // override previously defined state definition
        ...
    }

    state new_state {  // if no override, then add the state as a new one
        ...
    }
}
```

Examples: `super.state.transition`

base parser

```
parser base_parser {
    state parse_ipv4_state {
        extract(hdr.ipv4);
        transition accept;
    }

    state parse_ethernet {
        extract(hdr.eth);
        transition (hdr.eth.ethertype) {
            IPV4: parse_ipv4_state;
            default: accept;
        }
    }
}
```

extended parser

```
parser base_parser override {
    state parse_ipv6_state {
        extract(hdr.ipv6);
        transition accept;
    }

    state parse_ethernet override {
        extract(hdr.eth);
        transition (hdr.eth.ethertype) {
            IPV6: parse_ipv6_state;
            // no need to extract, only switch based
            // on already extracted header
            default: super.parse_ethernet.transition;
        }
    }
}
```

Examples: `super.state.transition`

merged parser

```
parser base_parser {
    state parse_ipv6_state {
        extract(hdr.ipv6);
        transition accept;
    }

    state _parse_ipv4_state {
        extract(hdr.ipv4);
        transition accept;
    }

    // original base state
    state parse_ethernet_transition {
        transition (hdr.eth.ethertype) {
            IPV4: parse_ipv4_state;
            default: accept;
        }
    }

    // extended state
    state parse_ethernet {
        extract(hdr.eth);
        transition (hdr.eth.ethertype) {
            IPV6: ipv6_state;
            // replace super reference with the name of the newly created transition part of the base state
            default: parse_ethernet_transition;
        }
    }
}
```

Proof-of-concept parser merging

1. Base P4 program (base.p4) remains unchanged
2. New P4 program (new.p4), reusing base.p4, written using P4-16 extensions (“p4++”)
3. new.p4 compiled using modified p4c frontend, generating the merged P4 program
4. Merged P4 program compiled with target (e.g. Tofino) compiler

Finer Control: update body with super

```
control foo(…) override {
    action moo(…) override {
        // new code here
        super.moo(); // call base action.
    }
    apply {
        // new code
        super.apply(); // call apply method from the base code.
        // more new code
    }
}
```

Finer Control Reuse Using `override`

```
// Augmenting action parameter and action body is supported.
// Defaults to adding new body to end.
action foo(...) override {  // tells compiler to patch previously defined action.
    ...
}

// tells compiler to allow adding new action.
action New_action() {
    ...
}

table moo override {
    key override = {  // Patch new key element
        // new key element.
    }

    actions override = {  // Patch to existing actions.
        New_action;
    }

    default_action override = action_x;  // tells compiler to use this action as default_action.
}
```

Thank you

Hemant Singh<hemant@mnkcg.com>

Extra Slides

Issues with extending code

- Extending deparser–hard problem
- If merged code does not fit in stages of target, target compiler fails compiling–so be it
- Not fitting in ASIC stages is pre-existing problem before P4–dealt with manual code shuffling
- Automatic resource allocation of target hardware not possible without access to target's p4c backend source or until backend is made modular

Other uses for `override`

```
package V1Switch(base_parser, new_ingress,
new_deparser) override; // tells compiler to
ignore (drop) previous package declaration and
use this one.
```

Control Reuse

- There is more flexibility in Control definition than in Parser
- Finer patching is discussed in another slide
- Coarse control patching is shown below

Base code abstracts some functionality into modular controls that users can use as building blocks.

```
control new_main() {
   base1.apply();
   base2.apply();
   custom_code.apply();
   base3.apply();
}
```

Base code puts some hooks in its code

```
control new_main() {
   new_before.apply();
   // base code is here
   new_after.apply();}
```

Packaging Thoughts

- Archive base.p4/vendor.p4 akin to a .jar file used with Java
- P4-Ansible already provides Python shell commands to inspect P4 code in base.p4; shell akin to jar file perusing tool
- Maybe add manifest to archive, support digital signing
- We started with merging vendor and customer P4 programs, this is why we use two separate P4 programs which is also akin to base Java code in .jar file to develop on top of
- For code resuse we could use single file with base and new code

Keywords: `default.yyy`

- `new.p4` has no select in parser state and only includes a transition

```
state parse_udp {
    packet.extract(hdr.udp);
      transition parse_rtp;
}
```

- Use “default.xxx” to hint compiler to change “default: accept;” to “default:parse_rtp;” in base state

```
state parse_udp override {
      packet.extract(hdr.udp);
      transition super.parse_udp.transition.default.parse_rtp;
}
```