P4's action-execution semantics and conditional operators

Anirudh Sivaraman

Massachusetts Institute of Technology

P4's action-execution semantics

1. Consider the statements:
   modify_field(hdr.fieldA, 1);
   modify_field(hdr.fieldB, hdr.fieldA);
2. Assume `hdr.fieldA = 0` initially.
3. If executed sequentially: `hdr.fieldB = 1`.
4. P4's semantics (7.1.1 of rc1 draft): "Both actions are started at the same time."
5. `hdr.fieldB = 0`.

Parallel semantics can be unintuitive

1. Feedback from P4 tutorial at SIGCOMM.
2. Ben Pfaff's email: "I don't think I'd noticed anything about parallel versus serial semantics before; maybe it is new. It will take some care in a software implementation."

Proposal: Change action execution semantics to sequential

1. Does this break any existing code?
2. Are programmers already using sequential semantics?

Analyzing programmer intent in existing P4 programs

1. Python script that uses the p4-hlir python module.
2. Analyze read and write sets for each action primitive in a compound action.

   modify_field(hdr.fieldA, 1);
   modify_field(hdr.fieldB, hdr.fieldA);
3. Read sets = {}, {hdr.fieldA}
4. Write sets = {hdr.fieldA}, {hdr.fieldB}
5. Read set for compound action = {hdr.fieldA}
6. Write set for compound action = {hdr.fieldA, hdr.fieldB}
7. Intersection of read/write sets = {hdr.fieldA}
8. Flag any compound action with an intersection between the read and write set.

Results on p4lang/p4factory/targets/switch.p4

1. 211 compound actions.
2. 163 with no read/write set intersection.
3. 48 with non-null read write set intersection.

Digging deeper

1. Of the flagged 48, 43 have write-after-read dependencies.

```
terminate_tunnel_inner_ethernet_ipv4 {

modify_field(qos_metadata.outer_dscp,l3_metadata.lkp_ip_tc);
    modify_field(l3_metadata.lkp_ip_tc,inner_ipv4.diffserv);
}
decap_vxlan_inner_ipv6 {
    copy_header(ethernet,inner_ethernet);
    remove_header(inner_ethernet);
}
```

2. Will work with both parallel and sequential semantics.
3. Sequential makes the intent clearer.

Digging deeper

1. 5 were written using sequential semantics.

```
egress_port_mirror {
    modify_field(i2e_metadata.mirror_session_id, session_id);
    clone_egress_pkt_to_egress(session_id, p4_field_list.e2e_mirror_info);
}
field_list e2e_mirror_info {
    i2e_metadata.mirror_session_id;

}
```

2. (May have cleverly exploited parallel semantics, but checked that sequential was the author's intent).

What we learned from switch.p4

1. Switching to sequential semantics is unlikely to break switch.p4.
2. P4 programmers already use sequential semantics.

We don't lose any expressive power

1. Any thing that's parallel (e.g. swap) can be expressed with a sequential construct.

   pkt.a = pkt.b;
   pkt.b = pkt.a;

   becomes

   pkt.tmp = pkt.a;
   pkt.a = pkt.b;
   pkt.b = pkt.tmp;

2. Many things cannot be done with just parallel statements e.g. atomic read, modify, write.

   pkt.tmp = register;
   pkt.tmp = pkt.tmp + 1;
   register = pkt.tmp;

Does this complicate the compiler?

1. Yes, the compiler backend might need to gracefully reject some code.
2. Backend will limit the longest dependency chain within a compound action.

But expressions (2.6 of rc1 draft) cause the same problem anyway

1. For instance,
   pkt.a = (pkt.b + (pkt.c - (pkt.d <<(pkt.e >>pkt.f))));
   is valid P4 code, which some backends will reject.
2. It's equivalent to this sequential version
   pkt.tmp1 = pkt.e >>pkt.f;
   pkt.tmp2 = pkt.d <<pkt.tmp1;
   pkt.tmp3 = pkt.c - pkt.tmp2;
   pkt.a = pkt.b + pkt.tmp3;
3. Can transform the second into the first by propagating expressions.
4. I think rejecting sequential code is no worse than rejecting complex expressions.

Conclusion

1. As P4 programmers, we don't lose anything (and gain quite a bit) by switching to sequential semantics.
2. The compiler needs a little more work, which is going to happen anyway.
3. Analysis script, results, and slides available at: https://github.com/anirudhSK/p4-semantics

One last thing: The conditional operator

1. Section 2.6 of rc1 proposes min/max operatrors
2. The min operator expressed as code is

```
(arith_expr1 < arith_expr2) ? arith_expr1 : arith_expr2;
```

3. I propose we generalize this to:

```
(bool_expr) ? arith_expr1 : arith_expr2;
```