Kleene Algebra with Tests
Lecture 4

Agenda
1. What can we do with NetKAT

Review: NetKAT is a KAT with an interpreted action and test. The semantics assumes that there is one packet. Given a packet from the beginning, a regular expression transform the packet.

In the original NetKAT paper, there is another action called "dup/obs". One can use it like "sw←2; dup; sw←5", which specifies that the packet is sent to switch 2, then the state is recorded, and then the packet is sent to switch 5.
A use case could be: to enforce that all ssh packet do not go to swith 3, we can use dup to record the state history and test it.

Operational Semantics

$$P \to 2^P$$

$$[\![f \leftarrow n]\!](\pi) = \{\pi[n/f]\}$$

$$[\![f = n]\!](\pi) = \begin{cases} \emptyset & \pi(f) \neq n \\ \{\pi\} & \pi(f) = n \end{cases}$$

$$[\![e + e']\!](\pi) = [\![e]\!](\pi) \cup [\![e']\!](\pi)$$

$$[\![e ; e']\!](\pi) = \bigcup_{\pi' \in [\![e']\!](\pi)} [\![e']\!](\pi')$$

Note that in NetKAT, each test is only satisfied by one packet, as the test is performed on each field. Therefore, tests and packets are isomorphic

Axioms

NetKAT satisfies all axioms of KAT with the following additional packet axioms:

$$f \leftarrow n; f \leftarrow m \equiv f \leftarrow m$$
$$f \leftarrow n; g \leftarrow m \equiv g \leftarrow m; f \leftarrow n$$
$$f \leftarrow n; g = m \equiv g = m; f \leftarrow n$$
$$f \leftarrow n; f = n \equiv f \leftarrow n$$
$$f = n; f \leftarrow n \equiv f = n$$
$$\sum_{n \in Val} f = n \equiv 1$$
$$f = n; f = m \equiv 0$$
$$f = n; dup \equiv dup; f = n$$

With these axioms, we can prove

$$[\![e]\!] = [\![f]\!] \Leftrightarrow e \equiv f$$

NetKAT has an automaton model that gives an efficient decision procedure for equivalence. The automaton model also has a compiler that compiles to openflow, p4 and BDDs. Most recently, this automaton model has been shown to compile to SPPs (BDD on steroids).

Network Modeling

Let's see how to use NetKAT to model a network and check properties.

A switch can be described using match-action table, e.g.

Condition	action
type = ssh	drop
dst = 3	$pt \leftarrow 5 + pt \leftarrow 3$

Such table is translated to NetKAT expression

$$(sw=k).\sum_e t_e ; a_e$$

which is the sum of conditions concatenated with actions.

Note that this is a simplified version. In the real one, the test is a conjunction of negation of prior conditions and the current condition. This implies that the order in the table matters.

A link from switch k, port 1 to switch j, port 2 can be described by

$$link_{kj} \triangleq sw=k ; pt=1 ; sw \leftarrow j ; pt \leftarrow 2$$

Now, we can combine network topology and switch behavior to form the network.

$$top \triangleq \sum_{\ell \in Link} \ell$$

$$Switch \triangleq \sum_{s \in sw} s$$

$$N \triangleq (top ; switch)^*$$

Application: reachability
Is B reachable from A?
We can test by checking whether the following holds:

$$sw=A; top; (switch; top)^*; sw=B \equiv 0$$

This expression tries out all pathes from A to B; if it is equivalent to the emptyset, it means that it's not reachable.
This is called emptyness test, which can be implemented extremely efficiently (100x faster than the baseline).

Application: forwarding Loop
Can a packet come back to where it has been?
We test whether the following holds:

$$\alpha; (top; switch); (top; switch)^*; \alpha \equiv 0$$

Since we cant run on all possible α, we choose a predicate over which port/switch a packet can come in:

$$in; (top; switch); (top; switch)^*; in \equiv 0$$

Application: Access control
See slides.