Automated Black-box Verification of Networking Systems
Collaborators

Nate Foster
Matteo Sammartino
Stefan Zetzsche
Dexter Kozen
Steffen Smolka
Many of today’s high-level languages were designed in an era when computers looked like this...
But nowadays, computers look like this...
And applications are structured like this...
We need new kinds of abstractions and tools for programming these networked systems!

- Centralized
- Sequential
- Functional

- Distributed
- Concurrent
- Interactive
Specify communication
Optimize performance
Guarantee security
Software-Defined Networking
Networking

“The last bastion of mainframe computing” [Hamilton 2009]

- Modern computers
  - implemented with commodity hardware
  - programmed using general-purpose languages, standard interfaces
- Networks
  - built and programmed the same way since the 1970s
  - low-level, special-purpose devices implemented on custom hardware
  - routers and switches that do little besides maintaining routing tables and forwarding packets
  - configured locally using proprietary interfaces
  - network configuration ("tuning") largely a black art
Networking

- Difficult to implement end-to-end routing policies and optimizations that require a global perspective
- Difficult to extend with new functionality
- Effectively impossible to reason precisely about behavior
Software-Defined Networking

A clean-slate architecture based on two key ideas:
• Generalize network devices
• Separate control and forwarding
Software-Defined Networks

Your Program goes here!

Ox Controller Platform
or POX, Beacon, Floodlight, others

OpenFlow API

OpenFlow-compatible switches
Pica8, Dell, NEC, HP, many others
A first step: the OpenFlow API [McKeown & al., SIGCOMM 08]

- specifies capabilities and behavior of switch hardware
- a language for manipulating network configurations
- very low-level: easy for hardware to implement, difficult for humans to write and reason about

But...

- is platform independent
- provides an open standard that any vendor can implement
OpenFlow Switch

General-purpose packet-processing device that can be used to implement switches, routers, firewalls, etc.

Key data structure is a flow table containing a prioritized list of match-action rules and counters.
Switch to controller:
- switch_connected
- switch_disconnected
- port_status
- packet_in
- stats_reply

Controller to switch:
- flow_mod
- packet_out
- stats_request

OpenFlow Controller

<table>
<thead>
<tr>
<th>Match</th>
<th>Actions</th>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Major Trend in Industry

Backbone network runs OpenFlow

Bought by VMware for $1.2B
Verification of networks

Trend in PL&Verification after Software-Defined Networks

- Design *high-level languages* that model essential network features
- Develop *semantics* that enables reasoning precisely about behaviour
- Build *tools* to synthesise low-level implementations automatically

- Frenetic [Foster & al., ICFP 11]
- Pyretic [Monsanto & al., NSDI 13]
- Maple [Voellmy & al., SIGCOMM 13]
- FlowLog [Nelson & al., NSDI 14]
- Header Space Analysis [Kazemian & al., NSDI 12]
- VeriFlow [Khurshid & al., NSDI 13]
- NetKAT [Anderson & al., POPL 14]
- and many others . . .
But…

What if there is no formal model?

Does the low-level implementation really do what it is supposed to do?
What we propose

Build black-box model via interactions with the system

Automated Modelling

Automated Verification

Properties
Automata learning (Angluin ’87)

**Finite alphabet** of system’s actions $A$

Set of system behaviours is a **regular language** $\mathcal{L} \subseteq A^*$

---

**Learner**

**Membership Query**

Q: $w \in \mathcal{L}$?  
A: Y/N

**Equivalence Query**

Q: $\mathcal{L}(H) = \mathcal{L}$?  
where $H$ is a **hypothesis automaton**

A: Y / N + counterexample

---

**Oracle**

$\mathcal{L}$

**Minimal DFA** accepting $\mathcal{L}$
"Lazy" testing and model-checking

Good for scalability!
Many interesting applications

• Detect TLS implementations flaws [USENIX Sec. Sym. ’15]
• TCP implementations [CAV ’16]
• Analysis of botnet protocols [CCS ’10]
• Bank cards …
To each application domain its model...

Non-deterministic

Probabilistic

Mealy Machines

Weighted

Alternating

Universal

Register

Buchi

Do I need to write my automata learning algorithm from scratch?

NO! Category Theory can help!
Categorical Automata Learning Framework

calf-project.org

Gerco van Heerdt  
UCL

Joshua Moerman  
Radboud University

Bartek Klin  
Warsaw University

Michal Szynwelski  
Warsaw University

Maverick Chardet  
ENS Lyon

Tiago Ferreira  
UCL Intern
Different automata, same structure

DFAs

\[ F = (-) \times A \]

Automaton type

Q \times A

\[ \downarrow \delta_Q \]

init_Q \rightarrow Q \rightarrow \text{out}_Q

1

q_0 \in Q

Initial state selector

2

F \subseteq Q

Collection of state outputs

"Collection" of states (not necessarily a set!)
A general framework

Abstract observation data structure

Hypothesis automaton

Target minimal automaton

abstract closedness and consistency

General correctness theorem

Guidelines for implementation

CALF: Categorical Automata Learning Framework (CSL ’17)
Gerco van Heerdt, Matteo Sammartino, Alexandra Silva
Other automata & optimizations

Change automaton type

<table>
<thead>
<tr>
<th>Set</th>
<th>Nom</th>
<th>Vect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nom</td>
<td>Nominal automata</td>
<td>Weighted automata</td>
</tr>
</tbody>
</table>

Change main data structure

- Observation tables
- Discrimination trees

Plug monads in

- Powerset
- Powerset with intersection
- Double powerset
- Maybe monad

- NFAs
- Universal automata
- Alternating automata
- Partial automata

Learning Nominal Automata (POPL ’17)
Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, Michal Szynwelski
Infinite alphabets

infinite-state, but finitely representable automata
Other automata & optimizations

Change automaton type
- Nominal automata
- Weighted automata

Change main data structure
- Observation tables
- Discrimination trees

Plug monads in
- Powerset
- Powerset with intersection
- Double powerset
- Maybe monad
- NFAs
- Universal automata
- Alternating automata
- Partial automata

Learning Nominal Automata (POPL ’17)
Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, Michal Szynwelski

Optimising Automata Learning via Monads
Gerco van Heerdt, Matteo Sammartino, Alexandra Silva
(arXiv:1704.08055)
Connections with other techniques

Extensions

Automata Learning

Minimization

Testing

Optimizations
What we propose

Build black-box model via interactions with the system

Automated Modelling

Properties

Automated Verification
Language to describe behaviours

NetKAT

= Kleene algebra with tests (KAT)

+ additional specialized constructs particular to network topology and packet switching

(0 + 1(01*0)*1)*
\{multiples of 3 in binary\}

\((ab)^{*} a = a(ba)^{*}\)
\{a, aba, ababa, \ldots\}

\((a + b)^{*} = a^* (ba^*)^*\)
\{all strings over \{a, b\}\}
(K, B, +, ·, *, −, 0, 1),  B ⊆ K

- (K, +, ·, *, 0, 1) is a Kleene algebra
- (B, +, ·, −, 0, 1) is a Boolean algebra
- (B, +, ·, 0, 1) is a subalgebra of (K, +, ·, 0, 1)

KAT = simple imperative language

- p, q, r, ...
- a, b, c, ...

If b then p else q = b;p + !b;q

While b do p = (bp)*!b
Deductive Completeness and Complexity

- deductively complete over language, relational, and trace models
- subsumes propositional Hoare logic (PHL)
- deductively complete for all relationally valid Hoare-style rules

\[
\begin{align*}
\{b_1\} & p_1 \{c_1\}, \ldots, \{b_n\} p_n \{c_n\} \\
\{b\} & p \{c\}
\end{align*}
\]

- decidable in PSPACE

Applications

- protocol verification
- static analysis and abstract interpretation
- verification of compiler optimizations
a packet $\pi$ is an assignment of constant values $n$ to fields $x$

a packet history is a nonempty sequence of packets

$\pi_1 :: \pi_2 :: \cdots :: \pi_k$

the head packet is $\pi_1$

NetKAT

- assignments $x \leftarrow n$
  assign constant value $n$ to field $x$ in the head packet

- tests $x = n$
  if value of field $x$ in the head packet is $n$, then pass, else drop

- dup
  duplicate the head packet
Networks in NetKAT

sw=6;pt=8;dst := 10.0.1.5;pt:=5

For all packets located at port 8 of switch 6, set the destination address to 10.0.1.5 and forward it out on port 5.
Networks in NetKAT

The behaviour of an entire network can be encoded in NetKAT by interleaving steps of processions by switches and topology.

\[
policy \quad \text{topo} \\
\vdots \\
(policy; \text{topo}); policy \\
+ \\
(policy; \text{topo}; policy; \text{topo}); policy \\
\vdots \\
(policy; \text{topo})^*; policy
\]
Semantics

packet history $\langle p, \ldots \rangle$ \rightarrow (policy; topo)^*; policy

set of packet histories $\{ \langle q, \ldots \rangle, \langle r, \ldots \rangle \}$

\[
\begin{align*}
[e] : H & \rightarrow 2^H \\
[x \leftarrow n](\pi_1 :: \sigma) & \triangleq \{ \pi_1[n/x] :: \sigma \} \\
[x = n](\pi_1 :: \sigma) & \triangleq \begin{cases} 
\{ \pi_1 :: \sigma \} & \text{if } \pi_1(x) = n \\
\emptyset & \text{if } \pi_1(x) \neq n
\end{cases} \\
[dup](\pi_1 :: \sigma) & \triangleq \{ \pi_1 :: \pi_1 :: \sigma \}
\end{align*}
\]
Verification using NetKAT

Reachability
- Can host A communicate with host B? Can every host communicate with every other host?

Security
- Does all untrusted traffic pass through the intrusion detection system located at C?

Loop detection
- Is it possible for a packet to be forwarded around a cycle in the network?
Verification using NetKAT

Soundness and Completeness [Anderson et al. 14]

- $\vdash p = q$ if and only if $\llbracket p \rrbracket = \llbracket q \rrbracket$

Decision Procedure [Foster et al. 15]

- NetKAT coalgebra
- efficient bisimulation-based decision procedure
- implementation in OCaml
- deployed in the Frenetic suite of network management tools
For all packets located at port 8 of switch 6, set the destination address to 10.0.1.5 and forward it out on port 5.
Current explorations

- Forwarding/Filtering behaviour
- Concurrency
- Large data domains

NetKat  CALF  CKA

in collaboration with Amazon
Other research directions

Software Analysis

- Learning the “correct ways” of using undocumented code
- Learning-based automated test generation

Hardware Analysis

- Analysing concurrency in hardware, in collaboration with ARM