Network Programming
with frenetic

Nate Foster (Cornell)
Arjun Guha (UMass)
Mark Reitblatt (Cornell)
Cole Schlesinger (Princeton)

FMCAD ’13 Tutorial
Network Programming
with frenetic


Arjun Guha (UMass)
Mark Reitblatt (Cornell)
Cole Schlesinger (Princeton)

FMCAD ’13 Tutorial
Networks Today

There are hosts...
Networks Today

Connected by switches...
Networks Today

There are also servers...
Networks Today

Connected by routers...
Networks Today

And a load balancer...
Networks Today

And a gateway router...
Networks Today

There are other ISPs...
Networks Today

So we need to run BGP...
Networks Today

And we need a firewall to filter incoming traffic...
Networks Today

There are also wireless hosts...
Networks Today

So we need wireless gateways...
Networks Today

And yet more middleboxes for lawful intercept...
Networks Today

Each color represents a different set of control plane protocols and algorithms...
Software-Defined Networking

A clean-slate programmable network architecture
A Major Trend in Networking

Backbone network runs OpenFlow

Bought for $1.2B (mostly cash)
**Vision:** program networks using a high-level language, generate low-level machine code using a compiler, and verify formal properties of networks automatically.
Vision: program networks using a high-level language, generate low-level machine code using a compiler, and verify formal properties of networks automatically.
Tutorial Outline
Tutorial Outline

Part I: Ox
- OpenFlow Overview
- Ox Applications

Part II: Frenetic
- Frenetic Overview
- Frenetic Applications

Part III: Formal methods
- Update consistency
- Verification and reasoning
OpenFlow Overview
OpenFlow Architecture

OpenFlow Switch

Controller

Ox Controller Platform
or POX, Beacon, Floodlight, etc.

OpenFlow API

OpenFlow Switch

OpenFlow-compatible switches
Pica8, Dell, NEC, HP, and many others
OpenFlow Architecture

Your Program goes here!

Ox Controller Platform
or POX, Beacon, Floodlight, etc.

OpenFlow API

OpenFlow Switch

OpenFlow-compatible switches
Pica8, Dell, NEC, HP, and many others
Controller

packet_in

packet_out all ports

OpenFlow Switch
Can write *any* packet processing function we want in OCaml
<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>All Packets</td>
<td>All Ports</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Priority</td>
<td>Pattern</td>
<td>Action</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>All Packets</td>
<td>All Ports</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
OpenFlow API

Switch to controller:
- switch_connected
- switch_disconnected
- packet_in
- stats_reply

Controller to switch:
- packet_out
- flow_mod
- stats_request
Ox Programming
open OxPlatform
open OpenFlow0x01_Core

module MyApplication = struct

include OxStart.DefaultTutorialHandlers

let switch_connected (sw : switchId) : unit =
  send_flow_mod sw 0l (add_flow prio pat act);
  send_flow_mod sw 0l (add_flow prio pat act);
  send_flow_mod sw 0l (add_flow prio pat act)

let packet_in (sw : switchId) (xid : xid) (pk : packetIn) : unit =
  send_packet_out sw 0l
  { output_payload = pk.input_payload;
    port_id = None;
    apply_actions = actions
  }
end

module Controller = OxStart.Make (MyApplication)
Frenetic Overview
Machine Languages

OpenFlow is a machine language

Programmers must think in terms of low-level concepts such as:

- Flow tables
- Matches
- Priorities
- Timeouts
- Events
- Callbacks

Key issue: programs don’t compose!
Can’t write (firewall; routing)
Current Controllers

(Monitor | Route | Load Balance) ; Firewall

Controller Platform
Current Controllers

One monolithic application

(Monitor | Route | Load Balance) ; Firewall

Controller Platform
Current Controllers

One monolithic application

(Monitor | Route | Load Balance); Firewall

Controller Platform

Challenges:
- Writing, testing, and debugging programs
- Reusing code across applications
- Porting applications to new platforms
Language-Based Approach

Monitor | Route | Load Balance | Firewall

Compiler | Run-Time System

Controller Platform

- - - - - - - - - - - - - - - - -
Language-Based Approach

One module for each task

Monitor | Route | Load Balance | Firewall

Compiler | Run-Time System

Controller Platform

- - - - - - - - - -

[Diagram showing the layers and modules of a language-based approach]
Language-Based Approach

One module for each task

Benefits:
- Easier to write, test, and debug programs
- Can reuse modules across applications
- Possible to port applications to new platforms
Frenetic is a programming language

Programmers work in terms of natural constructs:

- Functions
- Predicates
- Relational operators
- Logical properties

Compiler bridges the gap between these abstractions and their implementations in OpenFlow
Frenetic is a programming language

Programmers work in terms of natural constructs:

- Functions
- Predicates
- Relational operators
- Logical properties

Compiler bridges the gap between these abstractions and their implementations in OpenFlow
Frenetic predicates denote sets of located packets

**Syntax**

- `switch = s` (* switch location *)
- `inPort = n` (* port location *)
- `field = v` (* field value *)
- `pred1 && pred2` (* conjunction *)
- `pred1 || pred2` (* disjunction *)
- `!pred` (* negation *)

**Examples**

```
switch = 01 && !(inPort = 1)
dlSrc = 00:00:00:00:00:01
dlTyp = ip && nwProto = icmp
```
Policies

Frenetic policies denote functions from located packets to sets of located packets

Syntax

- `fwd(n)` (* forward *)
- `all` (* flood *)
- `pass` (* identity *)
- `drop` (* drop *)
- `field v -> v` (* modify *)
- `if pred then pol1 else pol2` (* conditional *)
- `let x = pol1 in pol2` (* variable binding *)
- `pol1 + pol2` (* parallel composition *)
- `pol1 ; pol2` (* sequential composition *)

Examples

if switch = 01 then all else drop
firewall; (monitor + route)
Frenetic Programming
The full Frenetic system contains a number of additional features including:

- Functions for transforming packets
- Stateful applications (learning, NAT, firewall)
- Integrated testing and debugging

Example

```haskell
let my_nat =
  let priv, pub = nat(publicIP = 10.0.0.254) in
    if switch = 1 && inPort = 1 then (priv; fwd(2))
    else if switch = 1 && inPort = 2 then (pub; fwd(1))
  in
monitorPolicy(learn; my_nat)
```
The **frenetic** System

**Frenetic DSL**
implemented using

**Frenetic**
implemented using

**Ox**

Domain-specific language
- predicates and policies
- monitoring
- mac learning
- network address translation

OCaml embedding
- predicates and policies
- queries

OCaml OpenFlow Platform
- similar to Nox, Pox, Floodlight, etc.
The **frenetic** System

- predicates
- policies
- queries

Stream of snapshots over time

**Frenetic**

implemented using

**Ox**

OCaml embedding

- predicates and policies
- queries

OCaml OpenFlow Platform

- similar to Nox, Pox, Floodlight, etc.
Frenetic in OCaml

Can also implement dynamic and stateful programs in Frenetic

Use CStruct and Lwt libraries internally

```
type pred =
  | Hdr of ptrn
  | OnSwitch of switchId
  | Or of pred * pred
  | And of pred * pred
  | Not of pred
  | Everything
  | Nothing

type switchEvent =
  | SwitchUp of switchId * SwitchFeatures.t
  | SwitchDown of switchId

type pol =
  | HandleSwitchEvent of (switchEvent -> unit)
  | Action of action
  | Filter of pred
  | Union of pol * pol
  | Seq of pol * pol
  | ITE of pred * pol * pol
...
```
Frenetic @ Home

TP-Link TL-WR1043ND

$50

Open firmware:
www.dd-wrt.com
Frenetic @ Home

TP-Link TL-WR1043ND

$50

Open firmware: www.dd-wrt.com
Consistent Updates
Example: Access Control

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>🌺 Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>🌺 Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>🌺 Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>
Example: Access Control

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>🧛‍♂️</td>
<td>Web</td>
<td>Allow</td>
</tr>
<tr>
<td>🧛‍♂️</td>
<td>Non-web</td>
<td>Drop</td>
</tr>
<tr>
<td>🧛‍♂️</td>
<td>Any</td>
<td>Allow</td>
</tr>
</tbody>
</table>

Configuration A

Process black-hat traffic on F1
Process white-hat traffic on \{F2,F3\}
Example: Access Control

<table>
<thead>
<tr>
<th>Security Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src</td>
</tr>
<tr>
<td>🧛‍♂️</td>
</tr>
<tr>
<td>🧛‍♂️</td>
</tr>
<tr>
<td>🧛‍♂️</td>
</tr>
</tbody>
</table>

**Configuration A**
- Process black-hat traffic on F1
- Process white-hat traffic on \{F2,F3\}
Example: Access Control

- **Security Policy**
  
<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

- **Configuration A**
  - Process black-hat traffic on F1
  - Process white-hat traffic on \{F2,F3\}

- **Configuration B**
  - Process black-hat traffic on \{F1,F2\}
  - Process white-hat traffic on F3
Network Updates

Challenges

• The network is a distributed system
• Can only update one element at a time

Our Approach

• Provide programmers with a construct for updating the entire network at once
• Semantics ensures “reasonable” behavior
• Engineer efficient implementations:
  - Compiler constructs update protocols
  - Optimizations applied automatically
Atomic Updates

• Seem sensible...
• but costly to implement...
• and difficult to reason about, due to behavior on in-flight packets
Update Semantics

Atomic Updates
- Seem sensible...
- but costly to implement...
- and difficult to reason about, due to behavior on in-flight packets

Per-Packet Consistent Updates
Every packet processed with old or new configuration, but not a mixture of the two

Per-Flow Consistent Updates
Every set of related packets processed with old or new configuration, but not a mixture of the two
Update Semantics

Atomic Updates
- Seem sensible...
- but costly to implement...
- and difficult to reason about, due to behavior on inflight packets

Per-Packet Consistent Updates
Every packet processed with old or new configuration, but not a mixture of the two

Per-Flow Consistent Updates
Every set of related packets processed with old or new configuration, but not a mixture of the two

Theorem (Universal Property Preservation)
An update is per-packet consistent if and only if it preserves all safety properties.
Implementation

**Two-phase commit**
- Build versioned internal and edge switch configurations
- Phase 1: Install internal configuration
- Phase 2: Install edge configuration

**Pure Extension**
- Update strictly adds paths

**Pure Retraction**
- Update strictly removes paths

**Sub-space updates**
- Update modifies a small number of paths
Verification
Recent Network Outages

We discovered a misconfiguration on this pair of switches that caused what's called a “bridge loop” in the network.

A network change was [...] executed incorrectly [...] more “stuck” volumes and added more requests to the re-mirroring storm.

Service outage was due to a series of internal network events that corrupted router data tables.

Experienced a network connectivity issue [...] interrupted the airline's flight departures, airport processing and reservations systems.
Existing Tools

There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig '10]
- AntEater [SIGCOMM '11]
- NICE [NSDI '12]
- Header Space Analysis [NSDI '12]
- VeriFlow [HotSDN '12]
- and many others...

This is exciting, because the networking community is starting to get serious about using formal methods.
Switch flow tables cannot be updated atomically

- Flow tables are huge
- Instructions only add/delete individual entries
- Must update the table live, while it is processing packets

\[ \text{table}_{sw} \subseteq f|_{sw} \]
Non-Atomic Updates

<table>
<thead>
<tr>
<th>Priority</th>
<th>Predicate</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Priority</th>
<th>Predicate</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Priority</th>
<th>Predicate</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Priority</th>
<th>Predicate</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>SSH</td>
<td>Drop</td>
</tr>
<tr>
<td>5</td>
<td>dst_ip = H1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>5</td>
<td>dst_ip = H2</td>
<td>Fwd 2</td>
</tr>
</tbody>
</table>
Non-Atomic Updates

Priority | Predicate | Action
---------|-----------|--------
5        | dst_ip = H1 | Fwd 1
5        | dst_ip = H2 | Fwd 2

update re-ordering

Priority | Predicate | Action
---------|-----------|--------
10       | SSH       | Drop
5        | dst_ip = H1 | Fwd 1
5        | dst_ip = H2 | Fwd 2

⊆

Priority | Predicate | Action
---------|-----------|--------
10       | SSH       | Drop
Our Approach

- Write programs in a high-level declarative network programming language
- Use a compiler and run-time system to generate low-level instructions
- Certify the compiler and run-time system using the Coq proof assistant
- Extract to OCaml and run on real hardware
Certified Software Systems

Recent success stories

• seL4 microkernel [SOSP ‘09]
• CompCert compiler [CACM ‘09]

Tools

Textbooks
Certified Software Systems

Recent success stories
- seL4 microkernel [SOSP ‘09]
- CompCert compiler [CACM ’09]

Tools
- Isabelle
- ACL2

Textbooks
- Software Foundations
- Certified Programming with Dependent Types
Certified Software Systems

Recent success stories
- seL4 microkernel [SOSP ’09]
- CompCert compiler [CACM ’09]

Tools
ACL2

Textbooks
Software Foundations
Certified Programming with Dependent Types

Write code
Prove correct

Lemma \( \text{inter\_wildcard\_other} \) : \( \forall x, \text{Wildcard\_inter} x \text{\_Wildcard\_All} x = x. \)
Proof.
intros; destruct x; auto.
Qed.

Lemma \( \text{inter\_wildcard\_other\_1} \) : \( \forall x, \text{Wildcard\_inter} x \text{\_Wildcard\_All} x = x. \)
Proof.
intros; destruct x; auto.
Qed.

Lemma \( \text{inter\_exact\_same} \) : \( \forall x, \text{Wildcard\_inter} (\text{Wildcard\_Exact} x) (\text{Wildcard\_Exact} x) = \text{Wildcard\_Exact} x. \)
Proof.
intros. unfold \text{Wildcard\_inter}. destruct (eqdec x x); intuition.
Qed.
Certified Software Systems

Recent success stories
- seL4 microkernel [SOSP '09]
- CompCert compiler [CACM '09]

Tools
- Isabelle
- ACL2

Textbooks
- Certified Programming with Dependent Types
- Software Foundations

Write code
Prove correct
Extract code

Inductive pred : Type :=
| OnSwitch : Switch -> pred
| TelBoot : Boot -> pred

Lemma inter_wildcard_other : forall x, Wildcard_inter WildcardAll x x.
Proof int Qed.

Lemma inter_wildcard_other1 : forall x, Wildcard_inter x WildcardAll = x.
Proof intros; destruct x; auto.
Qed.

Lemma inter_exact_same : forall x, Wildcard_inter (WildcardExact x) (WildcardExact x) = WildcardExact x.
Proof intros.
unfold Wildcard_inter.
destruct (eqdec x x); intuition.
Qed.
Certified Software Systems

Recent success stories

- seL4 microkernel [SOSP ’09]
- CompCert compiler [CACM ’09]

Tools

- Isabelle
- ACL2

Textbooks

- Software Foundations
- Certified Programming with Dependent Types

Write code
Prove correct
Extract code
Certified executable
Our Approach

- Write network programs in a high-level declarative programming language
- Use a compiler and run-time system to generate low-level instructions
- Certify the compiler and run-time system using the Coq proof assistant
- Extract to OCaml and execute on real network hardware
Our Approach

• Write network programs in a high-level declarative programming language
• Use a compiler and run-time system to generate low-level instructions
• Certify the compiler and run-time system using the Coq proof assistant
• Extract to OCaml and execute on real network hardware
Compiler Correctness

Formalization Highlights

- Library of algebraic properties of tables
- New tactic for proving equalities on bags
- General-purpose table optimizer
- Key invariant: all synthesized predicates are well-formed (w.r.t. protocol types)

Theorem

Theorem compile_correct :
  \forall pol \ sw \ pt \ pk,
  netcore_eval pol sw pt pk =
  table_eval (compile pol sw) pt pk.
OpenFlow: an open, standardized protocol for programming switches
Dell, HP, NEC, Pica8, OpenVSwitch, etc.
OpenFlow Specification

42 pages...

...of informal prose

...diagrams and flow charts

...and C struct definitions
Syntax

<table>
<thead>
<tr>
<th>Devices</th>
<th>Switch</th>
<th>S</th>
<th>::=</th>
<th>S(sw, pts, RT, inp, outp, cm, outm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller</td>
<td>C</td>
<td></td>
<td>::=</td>
<td>C(sw, f_in, f_out)</td>
</tr>
<tr>
<td>Link</td>
<td>L</td>
<td></td>
<td>::=</td>
<td>L(loc, pk, loc, pk)</td>
</tr>
<tr>
<td></td>
<td>OpenFlow Link to Controller</td>
<td></td>
<td>::=</td>
<td>M(sw, SMS, CMS)</td>
</tr>
</tbody>
</table>

| Packets and Locations | Packet               | pk     | ::= | abstract                           |
|                      | Switch ID            | sw     | ::= | N                                  |
|                      | Port ID              | pt     | ::= | N                                  |
|                      | Location             | loc    | ::= | sw × pt                            |
|                      | Located Packet       | lp     | ::= | loc × pk                           |

| Controller Components | Controller state     | f_loc | ::= | abstract                           |
|                      | Controller input relation | f_in | ::= | sw × CM × σ → σ                    |
|                      | Controller output relation | f_out | ::= | σ → sw × CM × σ                    |

| Switch Components    | Rule table            | [RT]   | ::= | abstract                           |
|                      | Rule table interpretation | ΔRT   | ::= | abstract                           |
|                      | Rule table modifier   | ΔRT    | ::= | abstract                           |
|                      | Ports on switch       | pts   | ::= | {pt_1, ..., pt_n}                  |
|                      | Consumed packets      | cmp   | ::= | {b_{pt_1}, ..., b_{pt_n}}         |
|                      | Produced packets      | op    | ::= | {b_{pt_1}, ..., b_{pt_n}}         |
|                      | Messages from controller | m   | ::= | {CM_1, ..., CM_n}                 |
|                      | Messages to controller | omt   | ::= | {CM_1, ..., CM_n}                 |

| Link Components      | Endpoints             | loc    | ::= | loc where loc ≠ loc_{in, out}     |
|                      | Packets from loc_{in, out} | pk    | ::= | [pt_1, ..., pt_n]                 |

| Controller Link      | Message queue from controller | SMS  | ::= | SM_1 + SM_2                       |
|                      | Message queue to controller | CMS  | ::= | CM_1 + CM_2                       |

Abstract OpenFlow Protocol

| Message from controller | SM | ::= | FlowMod ΔRT, PktOut pt |                  |
| Message to controller   | CM | ::= | PktIn pt pk | BarrierReply n |

Semantics

\[(\text{outm}', \text{outm}) = \{\text{RT}\}\{\text{pk}\}\]

\[S(sw, pts, RT, \{\text{pk}\}) \equiv \text{inp, outp, cm, outm} \quad \Rightarrow S(sw, pts, RT, \text{inp}', \text{outp}' \equiv \text{outp}, \text{cm}, \text{outm}') \equiv \text{outm})\]  
\[(\text{Pkt-Process})\]

\[S(sw, pts, RT, \text{inp}', \text{outp}, \text{cm}, \text{outm}) \quad \Rightarrow S(sw, pts, RT, \text{inp}', \text{outp}', \text{cm}, \text{outm}) \quad \Rightarrow L(sw, pts, pk, loc)\]  
\[(\text{Send-Write})\]

\[L(sw, pts, pk, loc) \quad \Rightarrow L(sw, pts, RT, \text{inp}', \text{outp}, \text{cm}, \text{outm}) \quad \Rightarrow L(sw, pts, pk, loc) \quad \Rightarrow \text{inp}', \text{outp}, \text{cm}, \text{outm})\]  
\[(\text{Recvd-Write})\]

\[\text{RT} = \text{args}(\Delta RT, RT)\]

\[S(sw, pts, RT, \text{inp}', \text{outp}, \{\text{FlowMod}\Delta RT\} \equiv \text{cm}, \text{outm}) \quad \Rightarrow S(sw, pts, RT') \quad \Rightarrow \text{inp}', \text{outp}, \text{cm}, \text{outm})\]  
\[(\text{Switch-FlowMod})\]

\[\text{pts} \subseteq \text{pts}\]

\[S(sw, pts, RT, \text{inp}', \text{outp}, \{\text{PktOut pk}\} \equiv \text{outm}, \text{outm}) \quad \Rightarrow S(sw, pts, RT, \text{inp}', \{\text{outm}, \text{outm}\} \equiv \text{outm})\]  
\[(\text{Switch-PktOut})\]

\[f_{\text{ctrl}}(\sigma) = \langle sw, SM, \sigma \rangle\]

\[
\mathcal{C}(\sigma, f_{\text{in}}, f_{\text{out}}) \mid M(sw, CMS) \quad \Rightarrow \mathcal{C}(\sigma', f_{\text{in}}', f_{\text{out}}') \mid M(sw, CMS) + \text{CMS}\]
\[\quad \text{(Ctrl-Send)}\]

\[f_{\text{recv}}(\sigma, CM) = \sigma';\]

\[
\mathcal{C}(\sigma, f_{\text{in}}, f_{\text{out}}) \mid M(sw, CMS, CMS) \quad \Rightarrow \mathcal{C}(\sigma', f_{\text{in}}', f_{\text{out}}') \mid M(sw, CMS)\]
\[\quad \text{(Ctrl-Recv)}\]

\[\text{SM} \neq \text{BarrierRequest n}\]

\[M(sw, CMS) + [\text{CMS}] \quad \Rightarrow M(sw, CMS) \quad \Rightarrow \text{M(sw, CMS)}\]
\[\quad \text{(Switch-Ctrl-Barrier)}\]

Models all features related to packet forwarding, and all essential asynchrony.
/* Fields to match against flows */
struct ofp_match {
  uint32_t wildcards;       /* Wildcard fields. */
  uint16_t in_port;         /* Input switch port. */
  uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
  uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
  uint16_t dl_vlan;         /* Input VLAN. */
  uint8_t dl_vlan_pcp;      /* Input VLAN priority. */
  uint8_t pad1[1];          /* Align to 64-bits. */
  uint16_t dl_type;         /* Ethernet frame type. */
  uint8_t nw_tos;           /* IP ToS (DSCP field, 6 bits). */
  uint8_t nw_proto;         /* IP protocol or lower 8 bits of ARP opcode. */
  uint8_t pad2[2];          /* Align to 64-bits. */
  uint32_t nw_src;          /* IP source address. */
  uint32_t nw_dst;          /* IP destination address. */
  uint16_t tp_src;          /* TCP/UDP source port. */
  uint16_t tp_dst;          /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
/* Fields to match against flows */

struct ofp_match {
  uint32_t wildcards;  /* Wildcard fields. */
  uint16_t in_port;   /* Input switch port. */
  uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
  uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
  uint16_t dl_vlan; /* Input VLAN. */
  uint8_t dl_vlan_pcp; /* Input VLAN priority. */
  uint8_t pad1[1]; /* Align to 64-bits. */
  uint16_t dl_type; /* Ethernet frame type. */
  uint8_t nw_tos; /* IP ToS (DSCP field, 6 bits). */
  uint8_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode. */
  uint8_t pad2[2]; /* Align to 64-bits. */
  uint32_t nw_src; /* IP source address. */
  uint32_t nw_dst; /* IP destination address. */
  uint16_t tp_src; /* TCP/UDP source port. */
  uint16_t tp_dst; /* TCP/UDP destination port. */
};

OFP_ASSERT(sizeof(struct ofp_match) == 40);
/* Fields to match against flows */

struct ofp_match {
    uint32_t wildcards;    /* Wildcard fields. */
    uint16_t in_port;      /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan;      /* Input VLAN. */
    uint8_t dl_vlan_pcp;   /* Input VLAN priority. */
    uint8_t pad1[1];       /* Align to 64-bits. */
    uint16_t dl_type;      /* Ethernet frame type. */
    uint8_t nw_tos;        /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto;      /* IP protocol or lower 8 bits of ARP opcode. */
    uint8_t pad2[2];       /* Align to 64-bits. */
    uint32_t nw_src;       /* IP source address. */
    uint32_t nw_dst;       /* IP destination address. */
    uint16_t tp_src;       /* TCP/UDP source port. */
    uint16_t tp_dst;       /* TCP/UDP destination port. */
};

OFP_ASSERT(sizeof(struct ofp_match) == 40);

Record Pattern : Type := MkPattern {
    dlSrc : Wildcard EthernetAddress;
    dlDst : Wildcard EthernetAddress;
    dlType : Wildcard EthernetType;
    dlVlan : Wildcard VLAN;
    dlVlanPcp : Wildcard VLANPriority;
    nwSrc : Wildcard IPAddress;
    nwDst : Wildcard IPAddress;
    nwProto : Wildcard IPProtocol;
    nwTos : Wildcard IPTypeOfService;
    tpSrc : Wildcard TransportPort;
    tpDst : Wildcard TransportPort;
    inPort : Wildcard Port
};
/* Fields to match against flows */

struct ofp_match {
    uint32_t wildcards; /* Wildcard fields */
    uint16_t in_port; /* Input switch port */
    uint8_t dl_vlan; /* Input VLAN */
    uint8_t pad1[1]; /* Align to 64-bits */
    uint8_t pad2[2]; /* Align to 64-bits */
    uint16_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode */
    uint16_t nw_tos; /* IP ToS (DSCP field, 6 bits) */
    uint16_t nw_dst; /* IP destination address */
    uint32_t nw_src; /* IP destination address */
    uint16_t dl_vlan_pcp; /* Input VLAN priority */
    uint16_t dl_type; /* Ethernet frame type */
    uint16_t dl_vlan_pcp; /* Input VLAN priority */
    uint16_t dl_src[OFP_ETH_ALEN]; /* Ethernet destination address */
    uint16_t dl_dst[OFP_ETH_ALEN]; /* Ethernet source address */
    uint16_t tp_dst; /* TCP/UDP destination port */
    uint16_t tp_src; /* TCP/UDP source port */
    uint32_t nw_tos; /* IP ToS (DSCP field, 6 bits) */
    uint32_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode */
    uint16_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode */
    uint16_t nw_tos; /* IP ToS (DSCP field, 6 bits) */
    uint16_t nw_dst; /* IP destination address */
    uint32_t nw_src; /* IP destination address */
    uint16_t dl_vlan_pcp; /* Input VLAN priority */
    uint16_t dl_type; /* Ethernet frame type */
    uint16_t dl_vlan_pcp; /* Input VLAN priority */
    uint16_t dl_src[OFP_ETH_ALEN]; /* Ethernet destination address */
    uint16_t dl_dst[OFP_ETH_ALEN]; /* Ethernet source address */

    Record Pattern : Type := MkPattern {
        d1Src := Wildcard EthernetAddress;
        d1Dst := Wildcard EthernetAddress;
        d1Type := Wildcard EthernetType;
        d1Vlan := Wildcard VLAN;
        d1VlanPcp := Wildcard VLANPriority;
        nwSrc := Wildcard IPAddress;
        nwTos := Wildcard IPTypeOfService;
        tpSrc := Wildcard TransportPort;
        tpDst := Wildcard TransportPort;
        inPort := Wildcard Port
    };

    OFP_ASSERT(sizeof(struct ofp_match) == 40);

    forward:
        let d1Src := Wildcard EthernetAddress.
        let d1Dst := Wildcard EthernetAddress.
        let d1Type := Wildcard EthernetType.
        let d1Vlan := Wildcard VLAN.
        let d1VlanPcp := Wildcard VLANPriority.
        let nwSrc := Wildcard IPAddress.
        let nwTos := Wildcard IPTypeOfService.
        let tpSrc := Wildcard TransportPort.
        let tpDst := Wildcard TransportPort.
        let inPort := Wildcard Port.

    Definition Pattern_inter (p p':Pattern) :=
        let d1Src := Wildcard EthernetAddress.
        let d1Dst := Wildcard EthernetAddress.
        let d1Type := Wildcard EthernetType.
        let d1Vlan := Wildcard VLAN.
        let d1VlanPcp := Wildcard VLANPriority.
        let nwSrc := Wildcard IPAddress.
        let nwTos := Wildcard IPTypeOfService.
        let tpSrc := Wildcard TransportPort.
        let tpDst := Wildcard TransportPort.
        let inPort := Wildcard Port.

    Define exact_pattern (pk : Packet) (pt : Word16.T) : Pattern :=
        MkPattern
            (WildcardExact (ptd1Src pk))
            (WildcardExact (ptd1Dst pk))
            (WildcardExact (ptd1Type pk))
            (WildcardExact (ptd1Vlan pk))
            (WildcardExact (ptd1VlanPcp pk))
            (WildcardExact (ptnwSrc pk))
            (WildcardExact (ptnwTos pk))
            (WildcardExact (pttpSrc pk))
            (WildcardExact (pttpDst pk))
            (WildcardExact pt).

        negb (Pattern_is_empty (Pattern_inter (exact_pattern pk pt) pat)).
Forwarding

Detailed model of matching, forwarding, and flow table update

```c
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards; /* Wildcard fields. */
    uint32_t in_port; /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan; /* Input VLAN. */
    uint8_t dl_vlan_pcp; /* Input VLAN priority. */
    uint8_t pad1[1]; /* Align to 64-bits. */
    uint16_t dl_type; /* Ethernet frame type. */
    uint16_t nw_tos; /* IP TOS (DSCP field, 6 bits). */
    uint16_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode. */
    uint8_t pad2[2]; /* Align to 64-bits. */
    uint32_t nw Src; /* IP source address. */
    uint32_t nw_dst; /* IP destination address. */
    uint16_t tp-src; /* TCP/UDP source port. */
    uint16_t tp-dst; /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
```

let d1Dst := Wildcard_inter_EthernetAddress.eqdec (ptrnD1Dst p) (ptrnD1Dst p') in
let d1Type := Wildcard_inter_Word16.eqdec (ptrnD1Type p) (ptrnD1Type p') in
let d1Vlan := Wildcard_inter_Word16.eqdec (ptrnD1Vlan p) (ptrnD1Vlan p') in
let d1VlanPcp := Wildcard_inter_Word8.eqdec (ptrnD1VlanPcp p) (ptrnD1VlanPcp p') in
let nwSrc := Wildcard_inter_Word32.eqdec (ptrnNwSrc p) (ptrnNwSrc p') in
let nwDst := Wildcard_inter_Word32.eqdec (ptrnNwDst p) (ptrnNwDst p') in
let nwProto := Wildcard_inter_Word8.eqdec (ptrnNwProto p) (ptrnNwProto p') in
let nwTos := Wildcard_inter_Word8.eqdec (ptrnNwTos p) (ptrnNwTos p') in
let tpSrc := Wildcard_inter_Word16.eqdec (ptrnTpSrc p) (ptrnTpSrc p') in
let tpDst := Wildcard_inter_Word16.eqdec (ptrnTpDst p) (ptrnTpDst p') in
let inPort := Wildcard_inter_Word16.eqdec (ptrnInPort p) (ptrnInPort p') in
MkPattern d1Src d1Dst d1Type d1Vlan d1VlanPcp
    nwSrc nwDst nwProto nwTos
tpSrc tpDst
inPort.

Definition exact_pattern (pk : Packet) (pt : Word16.T) : Pattern :=
    MkPattern
        (WildcardExact (pktD1Src pk)) (WildcardExact (pktD1Dst pk))
        (WildcardExact (pktD1Type pk))
        (WildcardExact (pktD1Vlan pk)) (WildcardExact (pktD1VlanPcp pk))
        (WildcardExact (pktNwSrc pk)) (WildcardExact (pktNwDst pk))
        (WildcardExact (pktNwProto pk)) (WildcardExact (pktNwTos pk))
        (Wildcard_of_option (pktTpSrc pk)) (Wildcard_of_option (pktTpDst pk))
        (WildcardExact pt).

    negb (Pattern_is_empty (Pattern_inter (exact_pattern pk pt) pat)).
Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”
Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”

Definition \( \text{InBuf} := \text{Bag Packet} \).
Definition \( \text{OutBuf} := \text{Bag Packet} \).
Definition \( \text{OFInBuf} := \text{Bag SwitchMsg} \).
Definition \( \text{OFOutBuf} := \text{Bag CtrlMsg} \).
Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”

Essential asynchrony: packet buffers, message reordering, and barriers

Definition $\text{InBuf} := \text{Bag Packet}$.
Definition $\text{OutBuf} := \text{Bag Packet}$.
Definition $\text{OFInBuf} := \text{Bag SwitchMsg}$.
Definition $\text{OFOOutBuf} := \text{Bag CtrlMsg}$.
Weak Bisimulation

$(H_1, \text{email})$
Weak Bisimulation

(H₁,✉) → (S₁,pt₁,✉)
Weak Bisimulation

\[(H_1, \text{✉️}) \rightarrow (S_1, pt_1, \text{✉️}) \rightarrow (S_2, pt_1, \text{✉️})\]
Weak Bisimulation

$$(H_1, \text{Envelope}) \rightarrow (S_1, pt_1, \text{Envelope}) \rightarrow (S_2, pt_1, \text{Envelope}) \rightarrow (H_2, \text{Envelope})$$
Weak Bisimulation

\((H_1, \text{message}) \rightarrow (S_1, \text{pt}_1, \text{message}) \rightarrow (S_2, \text{pt}_1, \text{message}) \rightarrow (H_2, \text{message})\)
Weak Bisimulation

(H₁,✉₁) → (S₁,pt₁,✉₁) → (S₂,pt₁,✉₁) → (H₂,✉₁)
Weak Bisimulation

\[(H_1, \text{message}) \rightarrow (S_1, pt_1, \text{message}) \rightarrow (S_2, pt_1, \text{message}) \rightarrow (H_2, \text{message})\]
Theorem: NetCore abstract semantics is weakly bisimilar to Featherweight OpenFlow + NetCore controller
Parameterized Weak Bisimulation

Invariants

- **Safety**: at all times, the rules installed on switches are a *subset* of the controller function
- **Liveness**: the controller eventually processes all packets diverted to it by switches

Theorem

```module RelationDefinitions :=
    FwOF.FwOFRelationDefinitions.Make (AtomsAndController).
...
```

```theorem fwof_abst_weak_bisim :
    weak_bisimulation
    concreteStep
    abstractStep
    bisim_relation.
```
Equational Reasoning
Network Features

What network features should a logical framework model?*

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation
- Path union

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation
- Path union

*Focusing just on packet forwarding
Network Features

What network features should a logical framework model?*

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation
- Path union
- Path iteration

*Focusing just on packet forwarding
NetKAT

\[
f ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \\
v ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \\
a, b, c ::= \text{true} \quad (* \text{true} *) \\
\mid \text{false} \quad (* \text{false} *) \\
\mid f = v \quad (* \text{test} *) \\
\mid a_1 \mid a_2 \quad (* \text{disjunction} *) \\
\mid a_1 \& a_2 \quad (* \text{conjunction} *) \\
\mid ! a \quad (* \text{negation} *) \\
p, q, r ::= \text{filter} \ a \\
\mid f ::= v \quad (* \text{modification} *) \\
\mid p_1 \mid p_2 \quad (* \text{union} *) \\
\mid p_1 ; p_2 \quad (* \text{sequence} *) \\
\mid p^* \quad (* \text{iteration} *) \\
\mid \text{dup} \quad (* \text{duplication} *)
\]
NetKAT

\[ f ::= \text{switch} | \text{inport} | \text{srcmac} | \text{dstmac} | \ldots \]
\[ v ::= 0 | 1 | 2 | 3 | \ldots \]
\[ a,b,c ::= \text{true} \quad \text{(* true *)} \]
\[ \quad | \text{false} \quad \text{(* false *)} \]
\[ \quad | f = v \quad \text{(* test *)} \]
\[ \quad | a_1 \mid a_2 \quad \text{(* disjunction *)} \]
\[ \quad | a_1 \& a_2 \quad \text{(* conjunction *)} \]
\[ \quad | \neg a \quad \text{(* negation *)} \]
\[ p,q,r ::= \text{filter} \ a \quad \text{(* filter *)} \]
\[ \quad | f ::= v \quad \text{(* modification *)} \]
\[ \quad | p_1 \mid p_2 \quad \text{(* union *)} \]
\[ \quad | p_1 ; p_2 \quad \text{(* sequence *)} \]
\[ \quad | p^* \quad \text{(* iteration *)} \]
\[ \quad | \text{dup} \quad \text{(* duplication *)} \]

\[ \text{if } a \text{ then } p_1 \text{ else } p_2 \triangleq (\text{filter } a; p_1) \mid (\text{filter } \neg a; p_2) \]
Example: Reachability

Given:
- Ingress predicate i
- Egress predicate e
- Topology t
- Switch program p

Test:
```
filter i; dup; (p; dup; t)*; filter e ~ filter false
```
Kleene Algebra with Tests

The design of NetKAT is not an accident!

Its foundation rests upon canonical mathematical structure:
• Regular operators (|, ;, and *) encode paths through topology
• Boolean operators (&, |, and !) encode switch tables

This is called a **Kleene Algebra with Tests (KAT)** [Kozen ’96]

KAT has an accompanying proof system for showing equivalences of the form p ~ q
Kleene Algebra with Tests

The design of NetKAT is not an accident!

Its foundation rests upon canonical mathematical structure:
• Regular operators (|, ;, and *) encode paths through topology
• Boolean operators (&, |, and !) encode switch tables

This is called a Kleene Algebra with Tests (KAT) [Kozen ’96]

KAT has an accompanying proof system for showing equivalences of the form $p \sim q$

**Theorems**
• **Soundness**: programs related by the axioms are equivalent
• **Completeness**: equivalent programs are related by the axioms
• **Decidability**: there is an algorithm for deciding equivalence
NetKAT Equational Theory

Kleene Algebra
\[ p \mid (q \mid r) \sim (p \mid q) \mid r \]
\[ p \mid q \sim q \mid p \]
\[ p \mid \text{filter false} \sim p \]
\[ p \mid p \sim p \]
\[ p ; (q ; r) \sim (p ; q) ; r \]
\[ p ; (q \mid r) \sim p ; q \mid p ; r \]
\[ (p \mid q) ; r \sim p ; r \mid q ; r \]
\[ \text{filter true} ; p \sim p \]
\[ p \sim p ; \text{filter true} \]
\[ \text{filter false} ; p \sim \text{filter false} \]
\[ p ; \text{filter false} \sim \text{filter false} \]
\[ \text{filter true} \mid p ; p^* \sim p^* \]
\[ \text{filter true} \mid p^* ; p \sim p^* \]
\[ p \mid q ; r \mid r \sim r \Rightarrow p^* ; q \mid r \sim r \]
\[ p \mid q ; r \mid q \sim q \Rightarrow p ; r^* \mid q \sim q \]

Boolean Algebra
\[ a \mid (b \& c) \sim (a \mid b) \& (a \mid c) \]
\[ a \mid \text{true} \sim \text{true} \]
\[ a \mid !a \sim \text{false} \]
\[ a \& a \sim a \]

Packet Algebra
\[ f := n; f' := n' \sim f' := n' ; f := n \quad \text{if } f \neq f' \]
\[ f := n; f' := n' \sim f' := n' ; f := n \quad \text{if } f \neq f' \]
\[ f := n; f = n \sim f := n \]
\[ f = n; f := n \sim f = n \]
\[ f := n; f := n' \sim f := n' \]
\[ f = n; f = n' \sim \text{filter false} \quad \text{if } n \neq n' \]
\[ \text{dup} ; f = n \sim f = n ; \text{dup} \]
Application: Optimization

Given a program and a topology:

Want to be able to answer questions like:

“Will my network behave the same if I put the firewall rules on A, or on switch B (or both)?”
Application: Optimization

Given a program and a topology:

Want to be able to answer questions like:

“Will my network behave the same if I put the firewall rules on A, or on switch B (or both)?”

Formally, does the following equivalence hold?

\[(\text{filter} \; \text{switch} = \text{A} ; \text{firewall}; \text{routing}) \mid (\text{filter} \; \text{switch} = \text{B}; \text{routing}) \]

\[\sim\]

\[(\text{filter} \; \text{switch} = \text{A} ; \text{routing}) \mid (\text{filter} \; \text{switch} = \text{B}; \text{firewall}; \text{routing})\]
Optimization Proof
Optimization Proof

\[\text{in; } (p_A; t)^*; p_A; \text{out}\]

\[\equiv \{ \text{definition in, out, and } p_A \} \]
\[s_A; \text{ssh}; ((s_A; \neg\text{ssh}; p + s_B; p); t)^*; p_A; s_B\]

\[\equiv \{ \text{KAT-Invariant} \} \]
\[s_A; \text{ssh}; ((s_A; \neg\text{ssh}; p + s_B; p); t; \text{ssh})^*; p_A; s_B\]

\[\equiv \{ \text{KA-Seq-Dist-R} \} \]
\[s_A; \text{ssh}; (s_A; \neg\text{ssh}; p; t; \text{ssh} + s_B; p; t; \text{ssh})^*; p_A; s_B\]

\[\equiv \{ \text{KAT-Commute} \} \]
\[s_A; \text{ssh}; (s_A; \text{drop}; p; t + s_B; p; t; \text{ssh})^*; p_A; s_B\]

\[\equiv \{ \text{BA-Contra} \} \]
\[s_A; \text{ssh}; (s_A; \text{drop}; p; t + s_B; p; t; \text{ssh})^*; p_A; s_B\]

\[\equiv \{ \text{KA-Seq-Zero, KA-Zero-Seq, KA-Plus-Comm, KA-Plus-Zero} \} \]
\[s_A; \text{ssh}; (s_B; p; t; \text{ssh})^*; p_A; s_B\]

\[\equiv \{ \text{KA-Unroll-L} \} \]
\[s_A; \text{ssh}; (\text{id} + (s_B; p; t; \text{ssh}); (s_B; p; t; \text{ssh})^*); p_A; s_B\]

\[\equiv \{ \text{KA-Seq-Dist-L and KA-Seq-Dist-R} \} \]
\[(s_A; \text{ssh}; p_A; s_B)^* \]
\[(s_A; \text{ssh}; s_B; p; t; \text{ssh}; (s_B; p; t; \text{ssh})^*); p_A; s_B\]

\[\equiv \{ \text{KAT-Commute} \} \]
\[(s_A; s_B; \text{ssh}; p_A)^* +\]
\[(s_A; s_B; \text{ssh}; p; t; \text{ssh}; (s_B; p; t; \text{ssh})^*); p_A; s_B\]

\[\equiv \{ \text{PA-Contra} \} \]
\[(\text{drop}; \text{ssh}; p_A)^* +\]
\[(\text{drop}; \text{ssh}; p; t; \text{ssh}; (s_B; p; t; \text{ssh})^*); p_A; s_B\]

\[\equiv \{ \text{KA-Zero-Seq, KA-Plus-Idem} \} \]
\[\text{drop}\]

\[\equiv \{ \text{KA-Seq-Zero, KA-Zero-Seq, KA-Plus-Idem} \} \]
\[s_A; (p_B; t)^*; (\text{ssh}; \text{drop}; p + s_B; \text{drop}; p; s_B)^*\]

\[\equiv \{ \text{PA-Contra and BA-Contra} \} \]
\[s_A; (p_B; t)^*; (\text{ssh}; s_A; s_B; p + s_B; \text{ssh}; \neg\text{ssh}; p; s_B)^*\]

\[\equiv \{ \text{KAT-Commute} \} \]
\[s_A; (p_B; t)^*; (\text{ssh}; s_A; s_B; p + s_B; \text{ssh}; \neg\text{ssh}; p; s_B)^*\]

\[\equiv \{ \text{KA-Seq-Dist-L and KA-Seq-Dist-R} \} \]
\[(s_A; (p_B; t)^*; \text{ssh}; (s_A; p + s_B; \neg\text{ssh}; p); s_B)^* \]

\[\equiv \{ \text{KAT-Commute} \} \]
\[s_A; \text{ssh}; (p_B; t)^*; (s_A; p + s_B; \neg\text{ssh}; p); s_B\]

\[\equiv \{ \text{definition in, out, and } p_B \} \]
\[\text{in; } (p_B; t)^*; p_B; \text{out}\]
Wrapping Up
Conclusion

- Networks are an promising area for applications of formal methods
- Software-defined networking is a new architecture that makes it easy to deploy formal verification tools
- Frenetic is a high-level language for programming networks and reasoning about their behavior:
  - Consistent updates
  - Machine-verified compiler and run-time system
  - Equational reasoning in NetKAT
Thank you!

The Team
Carolyn Anderson
Nate Foster
Arjun Guha
Jean-Baptiste Jeannin
Dexter Kozen
Mark Reitblatt
Jen Rexford
Cole Schlesinger
David Walker
Carbon

frenetic
http://frenetic-lang.org

Formal Foundations for Networking
Seminar 30-0613, Feb 2015
• Nikolaj Bjørner (MSR)
• Nate Foster (Cornell)
• Brighten Godfrey (UIUC)
• Pamela Zave (AT&T)