NetKAT: Semantic Foundations for Networks

Nate Foster
Cornell University
Software-Defined Networking

Key ideas: generalize devices, separate control and forwarding
Software-Defined Networking

Key ideas: generalize devices, separate control and forwarding
Current Controllers

Monitor | Route | Load Balance | Firewall

Controller Platform
Current Controllers

One monolithic application

Monitor | Route | Load Balance | Firewall

Controller Platform
Current Controllers

Challenges:
• Writing, testing, and debugging programs
• Reusing code across applications
• Porting applications to new platforms
<table>
<thead>
<tr>
<th>Route</th>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Route Table

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstip=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>dstip=10.0.0.2</td>
<td>Fwd 2</td>
</tr>
</tbody>
</table>

### Monitor Table

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcip=1.2.3.4</td>
<td>Count</td>
</tr>
<tr>
<td>Route</td>
<td>Monitor</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
</tbody>
</table>

### Route

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstip=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>dstip=10.0.0.2</td>
<td>Fwd 2</td>
</tr>
</tbody>
</table>

### Monitor

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcip=1.2.3.4</td>
<td>Count</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Route</th>
<th>Monitor</th>
</tr>
</thead>
</table>

### Route

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.1</td>
<td>Fwd 1, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.2</td>
<td>Fwd 2, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4</td>
<td>Count</td>
</tr>
<tr>
<td>dstip=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>dstip=10.0.0.1</td>
<td>Fwd 2</td>
</tr>
<tr>
<td>Pattern</td>
<td>Actions</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.1</td>
<td>Fwd 1, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.2</td>
<td>Fwd 2, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4</td>
<td>Count</td>
</tr>
<tr>
<td>dstip=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>dstip=10.0.0.2</td>
<td>Fwd 2</td>
</tr>
<tr>
<td>Pattern</td>
<td>Actions</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.1</td>
<td>Fwd 1, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.2</td>
<td>Fwd 2, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4</td>
<td>Count</td>
</tr>
<tr>
<td>dstip=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>dstip=10.0.0.2</td>
<td>Fwd 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcpdst = 22</td>
<td>Drop</td>
</tr>
<tr>
<td>*</td>
<td>Fwd ?</td>
</tr>
<tr>
<td>Pattern</td>
<td>Actions</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.1</td>
<td>Fwd 1, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.2</td>
<td>Fwd 2, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4</td>
<td>Count</td>
</tr>
<tr>
<td>dstip=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>dstip=10.0.0.2</td>
<td>Fwd 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcpdst = 22</td>
<td>Drop</td>
</tr>
<tr>
<td>*</td>
<td>Fwd ?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcip=1.2.3.4, tcpdst = 22</td>
<td>Count, Drop</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.1</td>
<td>Fwd 1, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.2</td>
<td>Fwd 2, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4</td>
<td>Count</td>
</tr>
<tr>
<td>tcpdst = 22</td>
<td>Drop</td>
</tr>
<tr>
<td>dstip=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>dstip=10.0.0.2</td>
<td>Fwd 2</td>
</tr>
</tbody>
</table>
Language-Based Approach

Monitor | Route | Load Balance | Firewall

Compiler | Run-Time System

Controller Platform

---
Language-Based Approach

One module for each task
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
Key ideas:

- A language abstraction between programs and hardware
- Constructs for reading state and specifying forwarding policies
- Support for modular composition through policy combiners
- Run-time system pushes rules to switches reactively
Key ideas:

- **NetCore policy language**
  
  \[
  e ::= h:w
  \ | \ switch s
  \ | \ inspect e f
  \ | \ e1 & e2
  \ | \ !e1
  \]

  \[
  S ::= \{ \ s1, \ldots, \ sk \} \]

  \[
  t ::= e \rightarrow S
  \ | \ t1 \& t1
  \ | \ !t
  \]

- **Compiler pushes forwarding rules to switches proactively**

- **Reactive specialization handles features that cannot be translated**
Key ideas:

• NetCore

P ::= f=n
  | switch s
  | P1 | P2
  | P1 & P2

A ::= drop
  | id
  | fwd n
  | flood

C ::= P => A
  | C1 => C2

• Sequential composition

• Virtual fields
Language IV

PLDI’13

Key ideas:

- Network-wide semantics
  \[ [p](lp) = M' \]
  \[ p \models \{lp\} + M \rightarrow M + M' \]

- Detailed “featherweight” model of software-defined networks

- Machine-checked proofs of correctness in Coq

- First real deployment
- Established a beachhead for network programming languages
- Got a lot of folks thinking seriously about modular composition
- Details of the compiler and run-time system interesting
Report Card

😊
- Established a beachhead for network programming languages
- Got a lot of folks thinking seriously about modular composition
- Details of the compiler and run-time system interesting

😢
- Key design choices revisited on each iteration
- Each semantics had a precise definition but was rather ad hoc
- Unclear how new features should interact with old ones
- Could not reason equationally about network-wide behavior
Language Features

What features should a network programming language provide?*

*Focusing just on packet forwarding
Language Features

What features should a network programming language provide?*

*Focusing just on packet forwarding
Language Features

What features should a network programming language provide?*

- Packet predicates
- Packet transformations

*Focusing just on packet forwarding*
What features should a network programming language provide?*

- Packet predicates
- Packet transformations

*Focusing just on packet forwarding
Language Features

What features should a network programming language provide?*

- Packet predicates
- Packet transformations
- Path construction

*Focusing just on packet forwarding
Language Features

What features should a network programming language provide?*

- Packet predicates
- Packet transformations
- Path construction

*Focusing just on packet forwarding
Language Features

What features should a network programming language provide?*

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

*Focusing just on packet forwarding
What features should a network programming language provide?*

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

*Focusing just on packet forwarding
What features should a network programming language provide?

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

*Focusing just on packet forwarding*
Language Features

What features should a network programming language provide?*

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

*Focusing just on packet forwarding
Language Features

What features should a network programming language provide?*

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

*Focusing just on packet forwarding
Language Features

What features should a network programming language provide?*

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

*Focusing just on packet forwarding
NetKAT
NetKAT

\[ f ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \]
NetKAT

\[
f ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots
\]

\[
v ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots
\]
NetKAT

\[ f ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \]
\[ v ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \]
\[ a,b,c ::= \text{true} \quad \text{(* true *)} \]
NetKAT

\[ f ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \]
\[ v ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \]
\[ a, b, c ::= \text{true} \quad \text{(* true *)} \]
\[ \mid \text{false} \quad \text{(* false *)} \]
NetKAT

\[ f ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \]
\[ v ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \]
\[ a,b,c ::= \text{true} \quad (* \text{true} *) \]
\[ \mid \text{false} \quad (* \text{false} *) \]
\[ \mid f = v \quad (* \text{test} *) \]
NetKAT

\[
\begin{align*}
f & ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \\
v & ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \\
a, b, c & ::= \text{true} \quad (* \text{true} *) \\
& \quad \mid \text{false} \quad (* \text{false} *) \\
& \quad \mid f = v \quad (* \text{test} *) \\
& \quad \mid a_1 \mid a_2 \quad (* \text{disjunction} *)
\end{align*}
\]
NetKAT

\[ f ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \]
\[ v ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \]
\[ a,b,c ::= \text{true} \quad \text{(* true *)} \]
\[ \mid \text{false} \quad \text{(* false *)} \]
\[ \mid f = v \quad \text{(* test *)} \]
\[ \mid a_1 \mid a_2 \quad \text{(* disjunction *)} \]
\[ \mid a_1 \& a_2 \quad \text{(* conjunction *)} \]
NetKAT

\[ f ::= \text{switch} | \text{inport} | \text{srcmac} | \text{dstmac} | \ldots \]

\[ v ::= 0 | 1 | 2 | 3 | \ldots \]

\[ a, b, c ::= \text{true} \quad (\text{true}) \]

\[ | \quad \text{false} \quad (\text{false}) \]

\[ | \quad f = v \quad (\text{test}) \]

\[ | \quad a_1 | a_2 \quad (\text{disjunction}) \]

\[ | \quad a_1 \& a_2 \quad (\text{conjunction}) \]

\[ | \quad \neg a \quad (\text{negation}) \]
NetKAT

\[
\begin{align*}
f &::= \text{switch} | \text{inport} | \text{srcmac} | \text{dstmac} | \ldots \quad (* \text{true} *) \\
v &::= 0 | 1 | 2 | 3 | \ldots \quad (* \text{false} *) \\
a,b,c &::= \text{true} \\
| \text{false} \\
| f = v \\
| a_1 | a_2 \\
| a_1 \& a_2 \\
| ! a \\
p,q,r &::= \text{filter} \quad (* \text{filter} *) \quad (* \text{disjunction} *) \\
\end{align*}
\]
NetKAT

\[ f ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \]
\[ v ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \]
\[ a, b, c ::= \text{true} \quad (* \text{true} *) \]
\[ \mid \text{false} \quad (* \text{false} *) \]
\[ \mid f = v \quad (* \text{test} *) \]
\[ \mid a_1 \mid a_2 \quad (* \text{disjunction} *) \]
\[ \mid a_1 \& a_2 \quad (* \text{conjunction} *) \]
\[ \mid \lnot a \quad (* \text{negation} *) \]
\[ p, q, r ::= \text{filter} \ a \quad (* \text{filter} *) \]
\[ \mid f ::= v \quad (* \text{modification} *) \]
NetKAT

\[ f ::= \textit{switch} \mid \textit{inport} \mid \textit{srcmac} \mid \textit{dstmac} \mid \ldots \]
\[ v ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \]
\[ a,b,c ::= \texttt{true} \]
\[ \mid \texttt{false} \]
\[ \mid f = v \]
\[ \mid a_1 \mid a_2 \]
\[ \mid a_1 \land a_2 \]
\[ \mid \neg a \]
\[ p,q,r ::= \texttt{filter} a \]
\[ \mid f ::= v \]
\[ \mid p_1 \mid p_2 \]

(* true *)

(* false *)

(* test *)

(* disjunction *)

(* conjunction *)

(* negation *)

(* filter *)

(* modification *)

(* union *)
NetKAT

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

\[
\begin{align*}
 f & ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \\
 v & ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \\
 a, b, c & ::= \text{true} \\
 & \quad \mid \text{false} \\
 & \quad \mid f = v \\
 & \quad \mid a_1 \mid a_2 \\
 & \quad \mid a_1 \& a_2 \\
 & \quad \mid !a \\
 p, q, r & ::= \text{filter} a \\
 & \quad \mid f := v \\
 & \quad \mid p_1 \mid p_2 \\
 & \quad \mid p_1 ; p_2 \\
 & \quad \mid p^* 
\end{align*}
\]
NetKAT

\[ f ::= \text{switch} \mid \text{inport} \mid \text{srcmac} \mid \text{dstmac} \mid \ldots \]

\[ v ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \]

\[ a, b, c ::= \text{true} \]

\[ | \text{false} \]

\[ | f = v \]

\[ | a_1 \mid a_2 \]

\[ | a_1 \& a_2 \]

\[ | ! a \]

\[ p, q, r ::= \text{filter} a \]

\[ | f ::= v \]

\[ | p_1 \mid p_2 \]

\[ | p_1 ; p_2 \]

\[ | p^* \]

\[ | \text{dup} \]

(* true *)

(* false *)

(* test *)

(* disjunction *)

(* conjunction *)

(* negation *)

(* filter *)

(* modification *)

(* union *)

(* sequence *)

(* iteration *)

(* duplication *)
NetKAT

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

\[ \text{if} \ a \ \text{then} \ p_1 \ \text{else} \ p_2 \triangleq (\text{filter} \ a; \ p_1) \mid (\text{filter} \ !a; \ p_2) \]
Basic Primitives

if srcip = 10.0.0.1 & !(dstport = 22) then
  port := 1
else
  port := 2

Firewall

Controller Platform
Basic Primitives

```
if srcip = 10.0.0.1 & !(dstport = 22) then
  port := 1
else
  port := 2
```

Firewall

Controller Platform

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstport=22</td>
<td>Drop</td>
</tr>
<tr>
<td>srcip=10.0.0.1</td>
<td>Forward 1</td>
</tr>
<tr>
<td>*</td>
<td>Forward 2</td>
</tr>
</tbody>
</table>
if srcip = 1.2.3.4 then port := 3

else if dstip = 10.0.0.1 then port := 2

if dstip = 10.0.0.1 then port := 1
Union

if srcip = 1.2.3.4 then port := 3

if dstip = 10.0.0.1 then port := 1
else if dstip = 10.0.0.1 then port := 2

Monitor

Controller Platform

Route

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.1</td>
<td>Forward 1, Forward 3</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.2</td>
<td>Forward 2, Forward 3</td>
</tr>
<tr>
<td>srcip=1.2.3.4</td>
<td>Forward 3</td>
</tr>
<tr>
<td>dstip=10.0.0.1</td>
<td>Forward 1</td>
</tr>
<tr>
<td>dstip=10.0.0.2</td>
<td>Forward 2</td>
</tr>
</tbody>
</table>
if srcip = *0 then dstip := 10.0.0.1
else if srcip = *1 then dstip := 10.0.0.2

if dstip = 10.0.0.1 then port := 1
else if dstip = 10.0.0.2 then port := 2
if srcip = *0 then dstip := 10.0.0.1
else if srcip = *1 then dstip := 10.0.0.2

if dstip = 10.0.0.1 then port := 1
else if dstip = 10.0.0.2 then port := 2

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcip=*0</td>
<td>dstip:=10.0.0.1, Forward 1</td>
</tr>
<tr>
<td>srcip=*1</td>
<td>dstip:=10.0.0.2, Forward 2</td>
</tr>
</tbody>
</table>
**Iteration**

```plaintext
if dstip = 192.168.0.0/16 then
  port := B
else if port = A & dstport = 80 then
  port := 1

if dstip = 10.0.0.0/8 then
  port := A
else if port = B & dstport = 22 then
  port := 2
```

**Controller Platform**

<table>
<thead>
<tr>
<th>Tenant A</th>
<th>Tenant B</th>
</tr>
</thead>
</table>

( ) *
**Iteration**

```
if dstip = 192.168.0.0/16 then
  port := B
else if port = A & dstport = 80 then
  port := 1

if dstip = 10.0.0.0/8 then
  port := A
else if port = B & dstport = 22 then
  port := 2
```

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstip=10.0.0.0/8, dstport=80</td>
<td>Forward 1</td>
</tr>
<tr>
<td>dstip=192.168.0.0/16, dstport=22</td>
<td>Forward 2</td>
</tr>
<tr>
<td>*</td>
<td>Drop</td>
</tr>
</tbody>
</table>
Semantic Foundation

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 $p \sim q$
Semantic Foundation

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

**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

- \( p \mid (q \mid r) \sim (p \mid q) \mid r \)
- \( p \mid q \sim q \mid p \)
- \( p \mid \text{filter false} \sim p \)
- \( p \mid p \sim p \)
- \( p; (q; r) \sim (p; q); r \)
- \( p; (q \mid r) \sim p; q \mid p; r \)
- \( (p \mid q); r \sim p; r \mid q; r \)

#### Boolean Algebra

- \( a \mid (b \& c) \sim (a \mid b) \& (a \mid c) \)
- \( a \mid \text{true} \sim \text{true} \)
- \( a \mid \text{!a} \sim \text{true} \)
- \( a \& b \sim b \& a \)
- \( a \& \text{!a} \sim \text{false} \)
- \( a \& a \sim a \)

#### Packet Algebra

- \( f := n; f' := n' \sim f' := n'; f := n \) if \( f \neq f' \)
- \( f := n; f' = n' \sim f' = n'; f := n \) if \( f \neq f' \)
- \( f := n; f = n \sim f := n \)
- \( f = n; f := n \sim f = n \)
- \( f := n; f := n' \sim f := n' \)
- \( f = n; f = n' \sim \text{filter false} \) if \( n \neq n' \)
- \( \text{dup}; f = n \sim f = n; \text{dup} \)
Application: Reachability

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

Test:
\[ \text{filter } i; \text{dup; (p; dup; t)*; filter } e \sim \text{filter false} \]
Application: Optimization

Given a program and a topology:

```
  ─────── A ─────── B ───────
  ┌───────┐           ┌───────┐
  │      │           │      │
  └───────┘           └───────┘
```

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?

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

\sim

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

\[
\begin{aligned}
&\text{in; } (p_A; t)^*; p_A; \text{ out} \\
\equiv &\{ \text{ definition in, out, and } p_A \} \\
&s_A; \text{ ssh; } ((s_A; \neg \text{ ssh; } p + s_B; p); t)^*; p_A; s_B \\
\equiv &\{ \text{ KAT-Invariant } \\
&s_A; \text{ ssh; } ((s_A; \neg \text{ ssh; } p + s_B; p); t; s_B)^*; p_A; s_B \\
\equiv &\{ \text{ KA-Seq-Dist-R } \\
&s_A; \text{ ssh; } (s_A; \neg \text{ ssh; } p + s_B; p; t; s_B)^*; p_A; s_B \\
\equiv &\{ \text{ KAT-Commute } \\
&s_A; \text{ ssh; } (s_A; \neg \text{ ssh; } p + s_B; p; t; s_B)^*; p_A; s_B \\
\equiv &\{ \text{ BA-Contra } \\
&s_A; \text{ ssh; } (s_A; \neg \text{ ssh; } t)^*; p_A; s_B \\
\equiv &\{ \text{ KA-Seq-Zero, KA-Seq-Comm, KA-Plus-Comm, KA-Plus-Zero } \\
&s_A; \text{ ssh; } (s_B; p; t; s_B)^*; p_A; s_B \\
\equiv &\{ \text{ KA-Unroll-L } \\
&s_A; \text{ ssh; } (\text{id + (s_B; p; t; s_B)}; (s_B; p; t; s_B)^*); p_A; s_B \\
\equiv &\{ \text{ KA-Seq-Dist-L and KA-Seq-Dist-R } \\
&(s_A; \text{ ssh; } p_A; s_B)^+ \\
&(s_A; \text{ ssh; } s_B; p; t; s_B; (s_B; p; t; s_B)^*); p_A; s_B \\
\equiv &\{ \text{ definition in, out, and } p_B \} \\
&\text{ in; } (p_B; t)^*; p_B; \text{ out} \\
\end{aligned}
\]
Application: Security

Each module controls a different portion of the traffic
Application: Security

Each module controls a different portion of the traffic

A slice is a lightweight abstraction for expressing isolated programs:
Application: Security

Each module controls a different portion of the traffic

A slice is a lightweight abstraction for expressing isolated programs:

\[ s ::= \{i\} \ p \ \{e\} \]
A slice is a lightweight abstraction for expressing isolated programs:

\[ s ::= \{i\} \ p \ \{e\} \]

Slices can be desugared into NetKAT by a simple translation
Application: Security

Each module controls a different portion of the traffic

A slice is a lightweight abstraction for expressing isolated programs:

\[ s ::= \{i\} \ p \ \{e\} \]

Slices can be desugared into NetKAT by a simple translation

**Theorem:** \((s_1 \mid s_2;\text{topo})^* = (s_1;\text{topo})^* \mid (s_2;\text{topo})^*\)
Ongoing Work

• NetKAT co-algebraically
  - Brzozowski derivative (DFA)
  - Antimirov partial derivative (NFA)
  - Algorithms for deciding equivalence

• Global compilation
  - Simulate automata state using header bits
  - Optimizations to reduce state space
  - Useful for compiling virtual paths, trees, etc.

• Non-deterministic NetKAT
  - Fault-tolerance
  - In-network load-balancing

• Verification tools
  - Z3-based backend
  - Automata-based backend (in progress)
Thank you!

NetKAT collaborators
• Carolyn Anderson (Swarthmore)
• Arjun Guha (UMass Amherst)
• Jean-Baptiste Jeannin (Cornell)
• Dexter Kozen (Cornell)
• Cole Schlesinger (Princeton)
• David Walker (Princeton)

Formal Foundations for Networks
Seminar 30-0613, February 2015
Co-organizers:
• Nikolaj Bjørner (MSR)
• Nate Foster (Cornell)
• Brighten Godfrey (UIUC)
• Pamela Zave (AT&T)

https://github.com/frenetic-lang/frenetic
https://github.com/frenetic-lang/pyretic
Completeness

Syntax

Atoms \( \alpha, \beta \triangleq f_1 = n_1; \cdots; f_k = n_k \)

Assignments \( \pi \triangleq f_1 \leftarrow n_1; \cdots; f_k \leftarrow n_k \)

Join-irreducibles \( x, y \in At; (P; \text{dup})^*; P \)

Simplified axioms for \( At \) and \( P \)

\( \pi \equiv \pi; \alpha \pi \quad \alpha; \text{dup} \equiv \text{dup}; \alpha \quad \sum_\alpha \alpha \equiv \text{id}, \)

\( \alpha \equiv \alpha; \pi_\alpha \quad \pi; \pi' \equiv \pi' \quad \alpha; \beta \equiv \text{drop}, \alpha \neq \beta \)

Join-irreducible concatenation

\( \alpha; p; \pi \uplus \beta; q; \pi' = \begin{cases} \alpha; p; q; \pi' & \text{if } \beta = \alpha \pi \\ \text{undefined} & \text{if } \beta \neq \alpha \pi \end{cases} \)

\( A \uplus B = \{ p \uplus q \mid p \in A, q \in B \} \subseteq I \)

Regular Interpretation: \( R(p) \subseteq (P + At + \text{dup})^* \)

\( R(\pi) = \{ \pi \} \)

\( R(p + q) = R(p) \cup R(q) \)

\( R(\alpha) = \{ \alpha \} \)

\( R(p; q) = \{ xy \mid x \in R(p), y \in R(q) \} \)

\( R(\text{dup}) = \{ \text{dup} \} \)

\( R(p^*) = \bigcup_{n \geq 0} R(p^n) \)

with \( p^0 = 1 \) and \( p^{n+1} = p; p^n \)

Language Model: \( G(p) \subseteq I = At; (P; \text{dup})^*; P \)

\( G(\pi) = \{ \alpha; \pi \mid \alpha \in At \} \)

\( G(p + q) = G(p) \cup G(q) \)

\( G(\alpha) = \{ \alpha; \pi_\alpha \} \)

\( G(p; q) = G(p) \uplus G(q) \)

\( G(\text{dup}) = \{ \alpha; \pi_\alpha; \text{dup}; \pi_\alpha \mid \alpha \in At \} \)

\( G(p^*) = \bigcup_{n \geq 0} G(p^n) \)