

# Another Excursion



“Trail Ridge Road provides spectacular view of the majestic scenery of RMNP. It is the highest continuous motorway in the United States, with more than eight miles lying above 11,000' and a maximum elevation of 12,183'”

## Route & Pace

All negotiable :-)

## Grade



*The road was designed with a ruling grade generally less than 5% and never exceeding 7%, less than half as steep as the Fall River Road.*

## Bike Rental



Cost: \$40-\$55

Hours: 8am-9pm

# Deciding NetKAT Equivalence using Derivatives

Nate Foster (Cornell)

Dexter Kozen (Cornell)

Matthew Milano (Cornell)

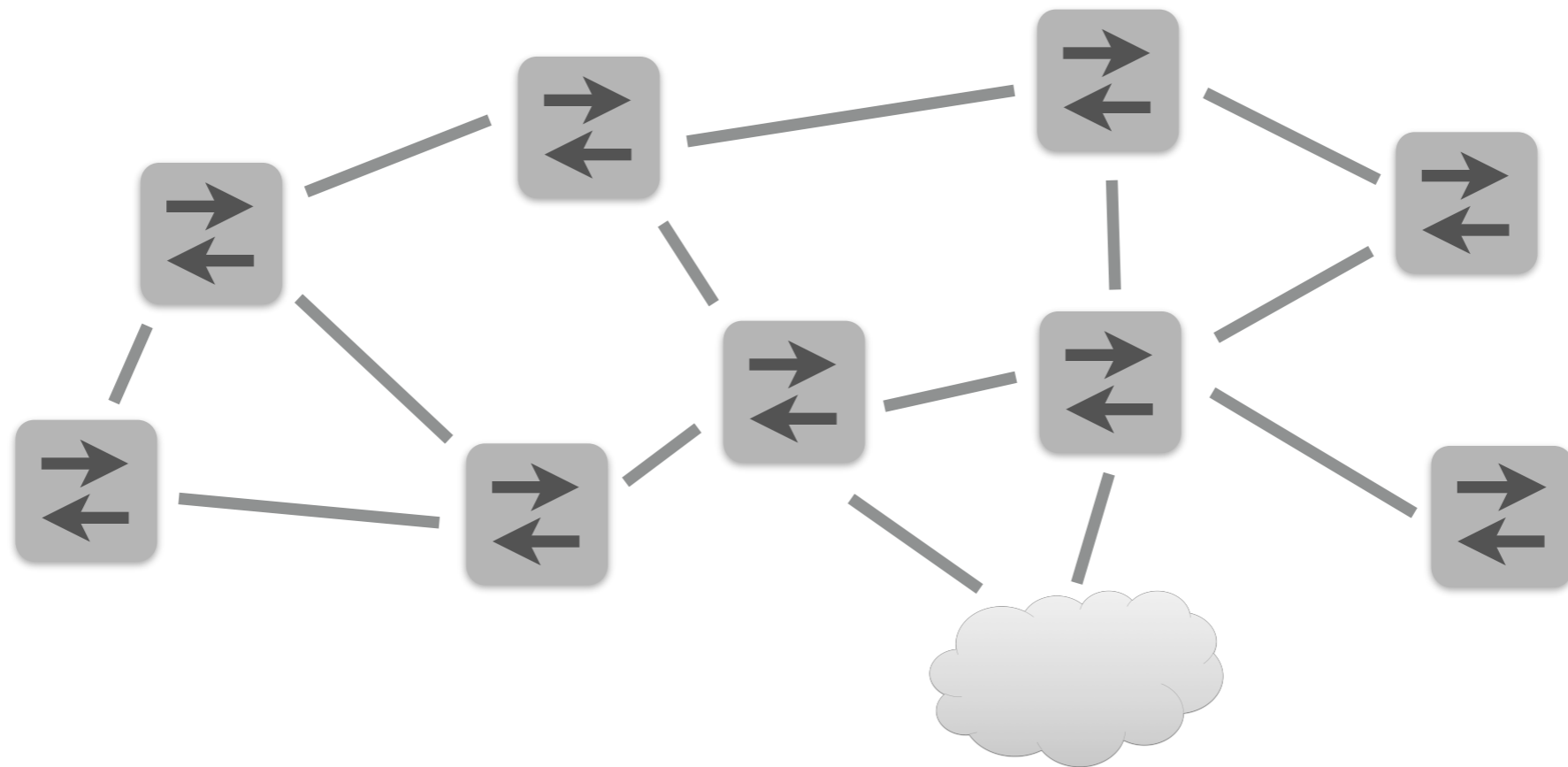
Alexandra Silva (Raboud)

Laure Thompson (Cornell)



# Network Programming Languages

There's been a lot of recent interest in programmable networks ("software-defined networking")...



...my group been designing high-level languages for programming and reasoning about networks

# NetKAT Review

# NetKAT Predicates

*Structures*

$f ::= \mathbf{switch} \mid \mathbf{port} \mid \mathbf{ethsrc} \mid \mathbf{ethdst} \mid \dots$

$pk ::= \{ \mathbf{switch} = n; \mathbf{port} = n; \mathbf{ethsrc} = n; \mathbf{ethdst} = n; \dots \}$

# NetKAT Predicates

*Structures*

$f ::= \mathbf{switch} \mid \mathbf{port} \mid \mathbf{ethsrc} \mid \mathbf{ethdst} \mid \dots$

$pk ::= \{ \mathbf{switch} = n; \mathbf{port} = n; \mathbf{ethsrc} = n; \mathbf{ethdst} = n; \dots \}$

*Syntax*

$a, b, c ::= \mathbf{true}$

(\* false \*)

|  $\mathbf{false}$

(\* true \*)

|  $f = n$

(\* test \*)

|  $a_1 \mathbf{||} a_2$

(\* disjunction \*)

|  $a_1 \mathbf{\&\&} a_2$

(\* conjunction \*)

|  $\mathbf{!} a$

(\* negation \*)

# NetKAT Predicates

## Structures

$f ::= \mathbf{switch} \mid \mathbf{port} \mid \mathbf{ethsrc} \mid \mathbf{ethdst} \mid \dots$

$pk ::= \{ \mathbf{switch} = n; \mathbf{port} = n; \mathbf{ethsrc} = n; \mathbf{ethdst} = n; \dots \}$

## Syntax

$a, b, c ::= \mathbf{true}$	(* false *)
$\mathbf{false}$	(* true *)
$f = n$	(* test *)
$a_1 \parallel a_2$	(* disjunction *)
$a_1 \ \&\& \ a_2$	(* conjunction *)
$! a$	(* negation *)

## Semantics

$\llbracket a \rrbracket \in \mathbf{Packet\ Set}$

$\llbracket \mathbf{true} \rrbracket = \mathbf{Packet}$

$\llbracket \mathbf{false} \rrbracket = \{ \}$

$\llbracket f = n \rrbracket = \{ pk \mid pk.f = n \}$

$\llbracket a_1 \parallel a_2 \rrbracket = \llbracket a_1 \rrbracket \cup \llbracket a_2 \rrbracket$

$\llbracket a_1 \ \&\& \ a_2 \rrbracket = \llbracket a_1 \rrbracket \cap \llbracket a_2 \rrbracket$

$\llbracket ! a \rrbracket = \mathbf{Packet} \setminus \llbracket a \rrbracket$

# NetKAT Policies

*Structures*

$h ::= \langle pk \rangle \mid pk :: h$



# NetKAT Policies

*Structures*

$h ::= \langle pk \rangle \mid pk :: h$

*Syntax*

$p, q, r ::=$	<b>filter</b> a	(* filter *)
	f := n	(* modification *)
	p <sub>1</sub> + p <sub>2</sub>	(* union *)
	p <sub>1</sub> ; p <sub>2</sub>	(* sequence *)
	p*	(* iteration *)
	<b>dup</b>	(* duplication *)

# NetKAT Policies

*Structures*

$h ::= \langle pk \rangle \mid pk :: h$

*Syntax*

$p, q, r ::=$  **filter**  $a$  (\* filter \*)  
|  $f := n$  (\* modification \*)  
|  $p_1 + p_2$  (\* union \*)  
|  $p_1 ; p_2$  (\* sequence \*)  
|  $p^*$  (\* iteration \*)  
| **dup** (\* duplication \*)

*Semantics*

$\llbracket p \rrbracket \in \mathbf{History} \rightarrow \mathbf{History Set}$

$\llbracket \mathbf{filter} a \rrbracket pk :: h = \begin{cases} \{ pk :: h \} & \text{if } pk \in \llbracket a \rrbracket \\ \{ \} & \text{otherwise} \end{cases}$

$\llbracket f := n \rrbracket pk :: h = \{ pk[f:=n] :: h \}$

$\llbracket p_1 + p_2 \rrbracket h = \llbracket p_1 \rrbracket h \cup \llbracket p_2 \rrbracket h$

$\llbracket p_1 \cdot p_2 \rrbracket h = (\llbracket p_1 \rrbracket \cdot \llbracket p_2 \rrbracket) h$

$\llbracket p^* \rrbracket h = (\bigcup_i \llbracket p \rrbracket^i) h$

$\llbracket \mathbf{dup} \rrbracket pk :: h = \{ pk :: pk :: h \}$

# NetKAT Policies

Structures

$h ::= \langle pk \rangle \mid pk :: h$

Syntax

$p, q, r ::= \mathbf{filter} \ a \quad (* \text{ filter } *)$   
 $\quad \mid f := n \quad (* \text{ modification } *)$   
 $\quad \mid p_1 + p_2 \quad (* \text{ union } *)$

**drop**  $\triangleq$  **filter false**

**id**  $\triangleq$  **filter true**

**if a then**  $p_1$  **else**  $p_2 \triangleq (\mathbf{filter} \ a \bullet p_1) + (\mathbf{filter} \ !a \bullet p_2)$

$\llbracket \mathbf{filter} \ a \rrbracket pk :: h = \begin{cases} \{ pk :: h \} & \text{if } pk \in \llbracket a \rrbracket \\ \{ \} & \text{otherwise} \end{cases}$

$\llbracket f := n \rrbracket pk :: h = \{ pk[f:=n] :: h \}$

$\llbracket p_1 + p_2 \rrbracket h = \llbracket p_1 \rrbracket h \cup \llbracket p_2 \rrbracket h$

$\llbracket p_1 \bullet p_2 \rrbracket h = (\llbracket p_1 \rrbracket \bullet \llbracket p_2 \rrbracket) h$

$\llbracket p^* \rrbracket h = (\bigcup_i \llbracket p \rrbracket^i) h$

$\llbracket \mathbf{dup} \rrbracket pk :: h = \{ pk :: pk :: h \}$

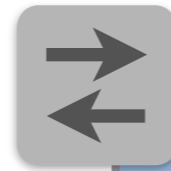
# Reasoning in NetKAT

# Encoding Tables

The forwarding tables  
maintained by switches  
can be encoded using  
conditional policies

# Encoding Tables

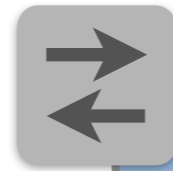
The forwarding tables maintained by switches can be encoded using conditional policies



Pattern	Actions
dstport=22	Drop
srcip=10.0.0.0/8	Forward 1
*	Forward 2

# Encoding Tables

The forwarding tables maintained by switches can be encoded using conditional policies



Pattern	Actions
dstport=22	Drop
srcip=10.0.0.0/8	Forward 1
*	Forward 2

## *Table Normal Form*

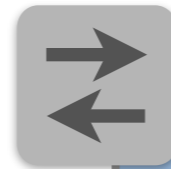
$\text{fwd} ::= f_1 := n_1 \cdot \dots \cdot f_k := n_k + \text{fwd} \mid \mathbf{drop}$

$\text{pat} ::= f = n \cdot \text{pat} \mid \mathbf{true}$

$\text{tbl} ::= \mathbf{if} \text{ pat } \mathbf{then} \text{ fwd } \mathbf{else} \text{ tbl} \mid \text{fwd}$

# Encoding Tables

The forwarding tables maintained by switches can be encoded using conditional policies



Pattern	Actions
dstport=22	Drop
srcip=10.0.0.0/8	Forward 1
*	Forward 2

## *Table Normal Form*

$\text{fwd} ::= f_1 := n_1 \cdot \dots \cdot f_k := n_k + \text{fwd} \mid \text{drop}$

$\text{pat} ::= f = n \cdot \text{pat} \mid \text{true}$

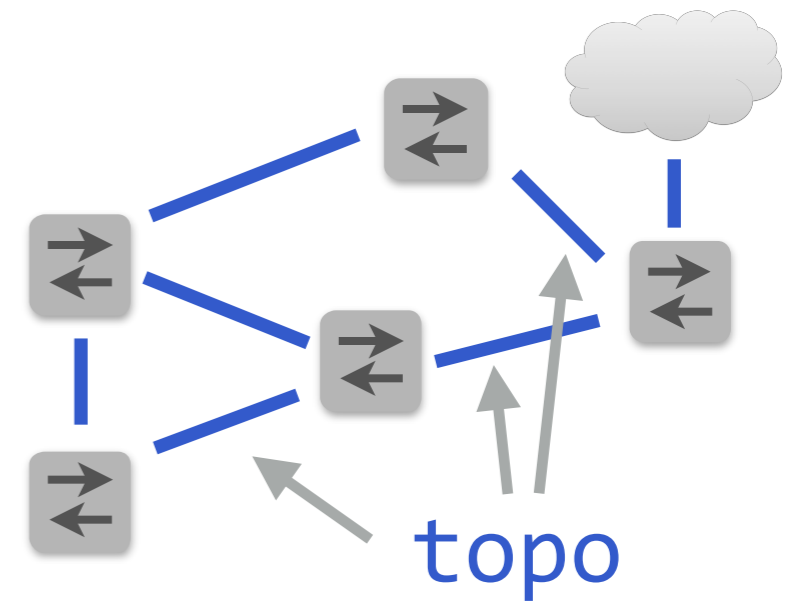
$\text{tbl} ::= \text{if pat then fwd else tbl} \mid \text{fwd}$

```
if dstport=22 then
  drop
else if srcip=10.0.0.0/8 then
  port := 1
else
  port := 2
```



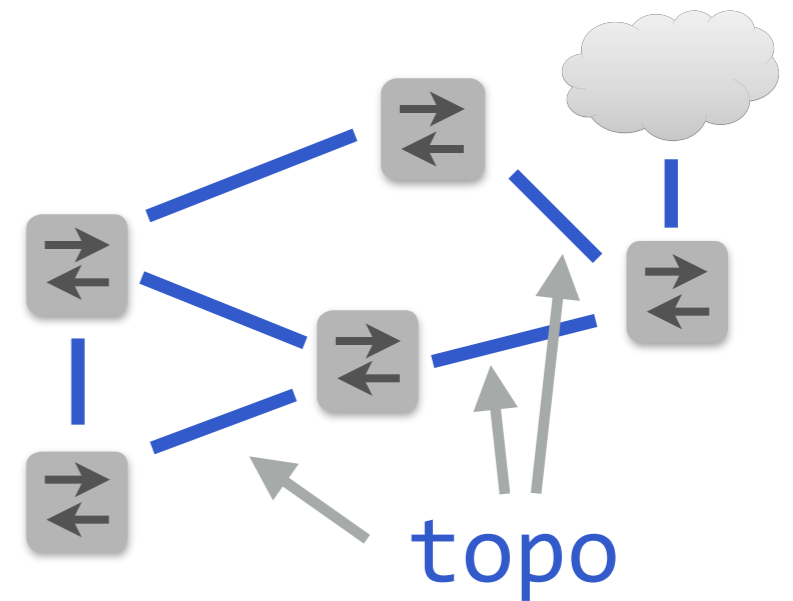
# Encoding Topologies

Links can be modeled as policies that forward packets from one end to the other, and topologies as unions of links



# Encoding Topologies

Links can be modeled as policies that forward packets from one end to the other, and topologies as unions of links



*Topology Normal Form*

$l_{pred} ::= \mathbf{switch}=n \cdot \mathbf{port}=n$

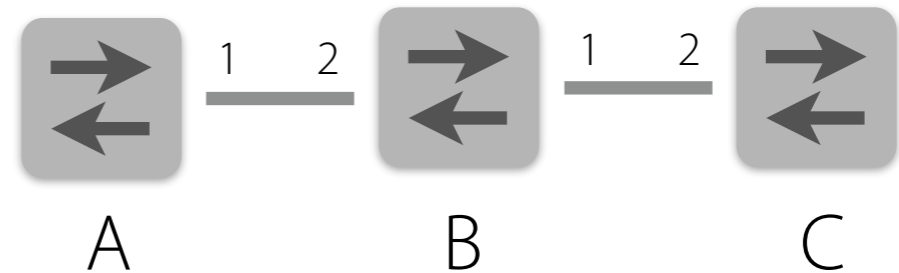
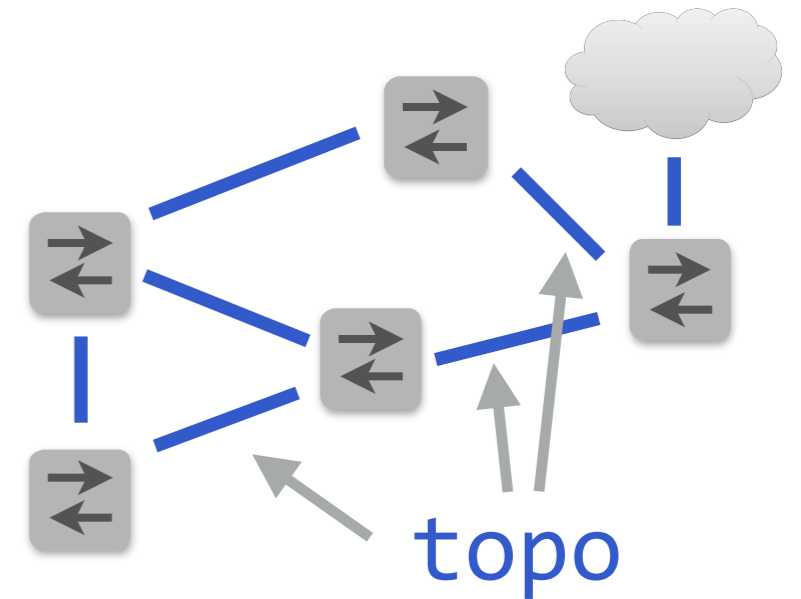
$l_{pol} ::= \mathbf{switch}:=n \cdot \mathbf{port}:=n$

$link ::= l_{pred} \cdot l_{pol}$

$topo ::= link \mathbf{+} topo \mid \mathbf{drop}$

# Encoding Topologies

Links can be modeled as policies that forward packets from one end to the other, and topologies as unions of links



## *Topology Normal Form*

$l_{pred} ::= \mathbf{switch=n} \cdot \mathbf{port=n}$

$l_{pol} ::= \mathbf{switch:=n} \cdot \mathbf{port:=n}$

$link ::= l_{pred} \cdot l_{pol}$

$topo ::= link + topo \mid \mathbf{drop}$

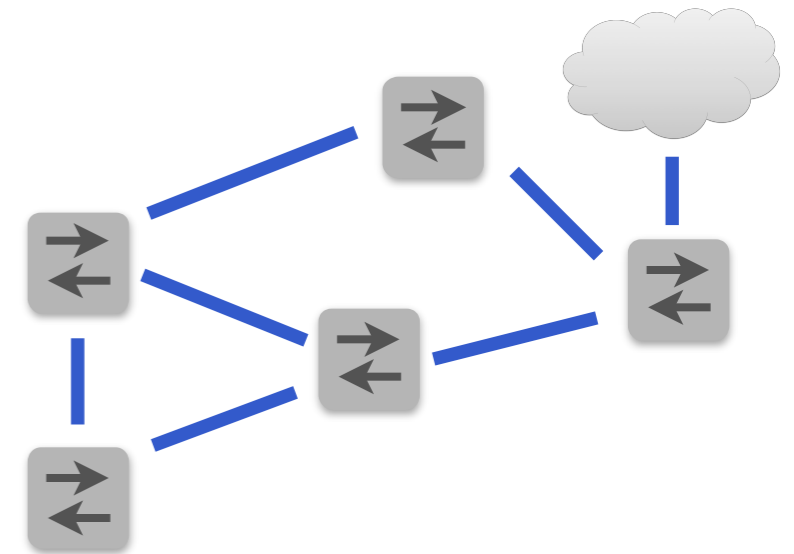
$\mathbf{switch=A \cdot port=1 \cdot switch:=B \cdot port:=2} +$   
 $\mathbf{switch=B \cdot port=2 \cdot switch:=A \cdot port:=1} +$   
 $\mathbf{switch=B \cdot port=1 \cdot switch:=C \cdot port:=2} +$   
 $\mathbf{switch=C \cdot port=2 \cdot switch:=B \cdot port:=1} +$   
 $\mathbf{drop}$

# Encoding Networks

An entire network can be encoded by interleaving policy and topology processing steps arbitrarily many number of times

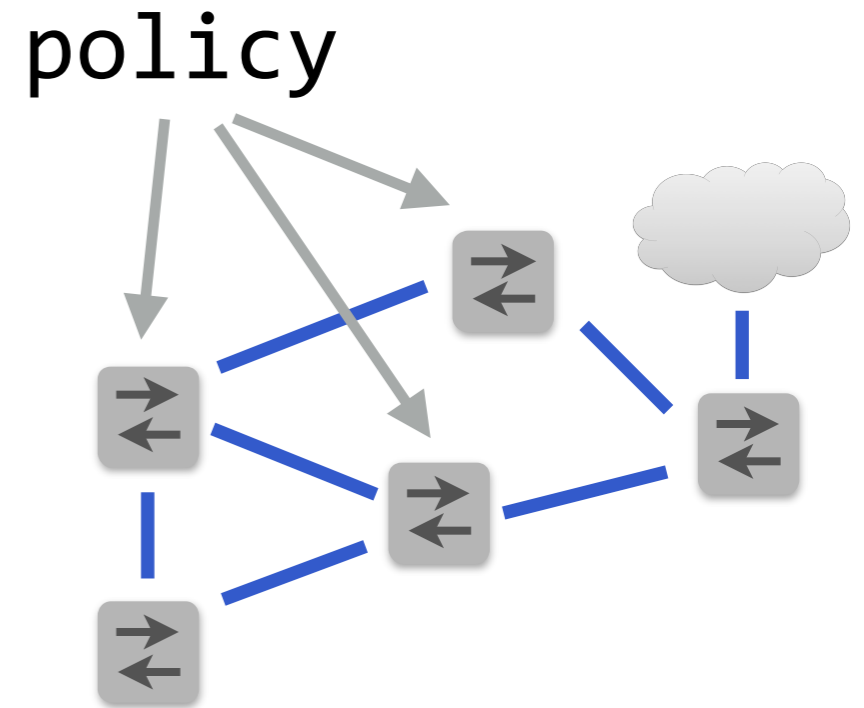
# Encoding Networks

An entire network can be encoded by interleaving policy and topology processing steps arbitrarily many number of times



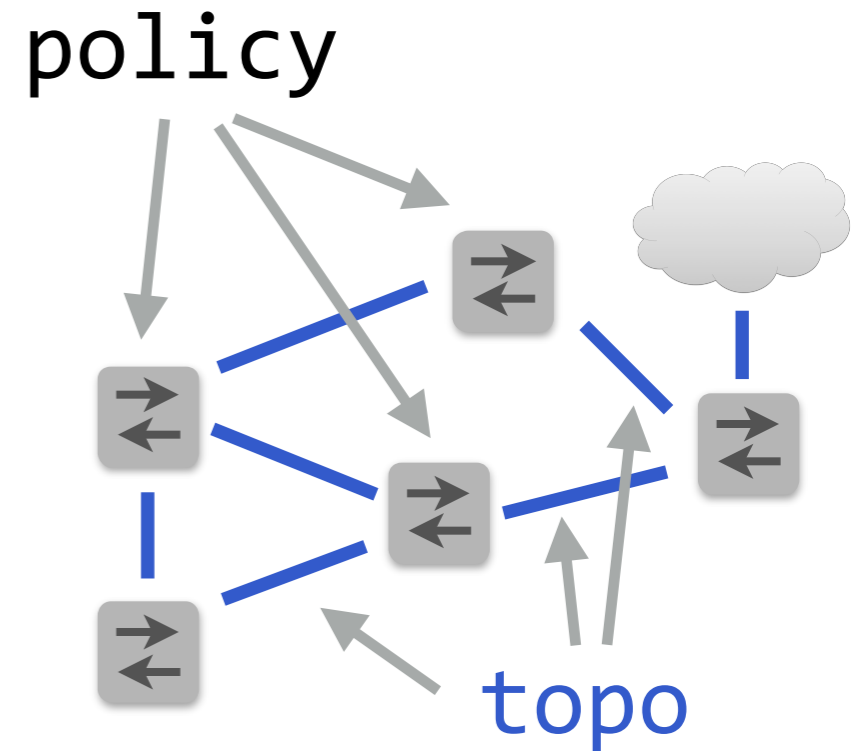
# Encoding Networks

An entire network can be encoded by interleaving policy and topology processing steps arbitrarily many number of times



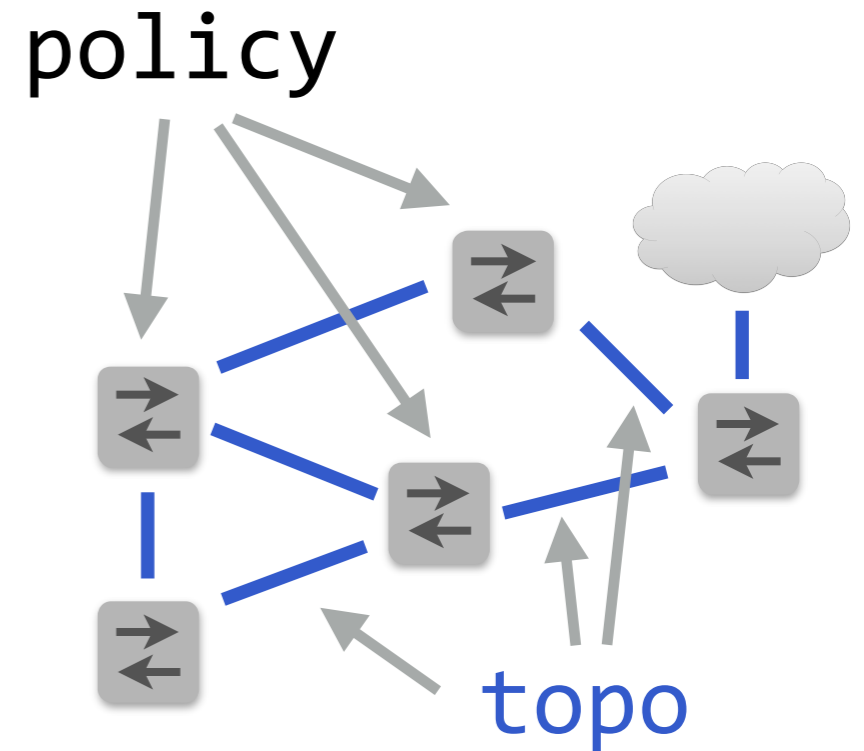
# Encoding Networks

An entire network can be encoded by interleaving policy and topology processing steps arbitrarily many number of times



# Encoding Networks

An entire network can be encoded by interleaving policy and topology processing steps arbitrarily many number of times

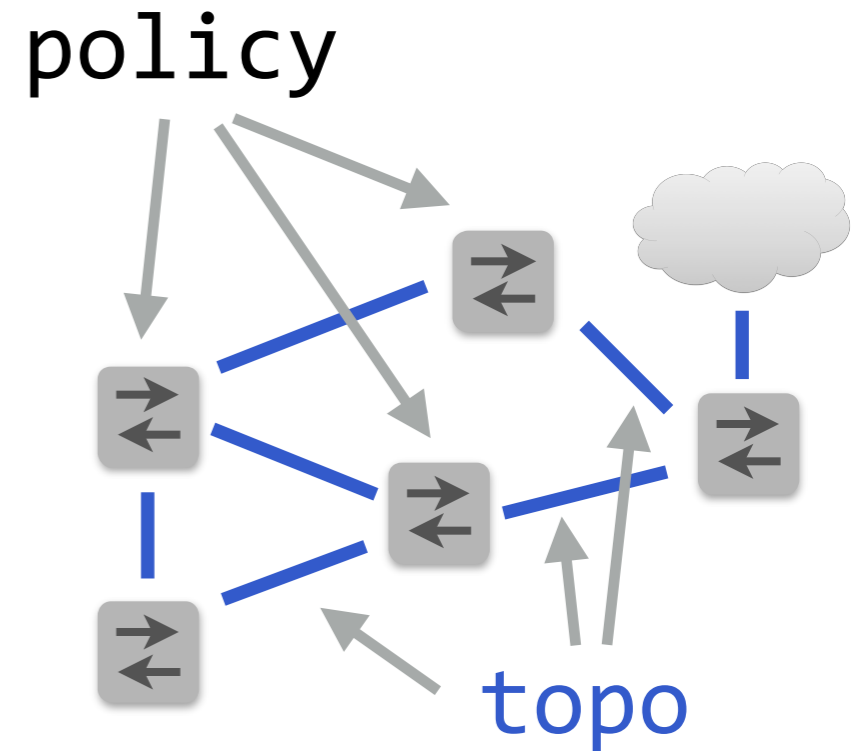


id



# Encoding Networks

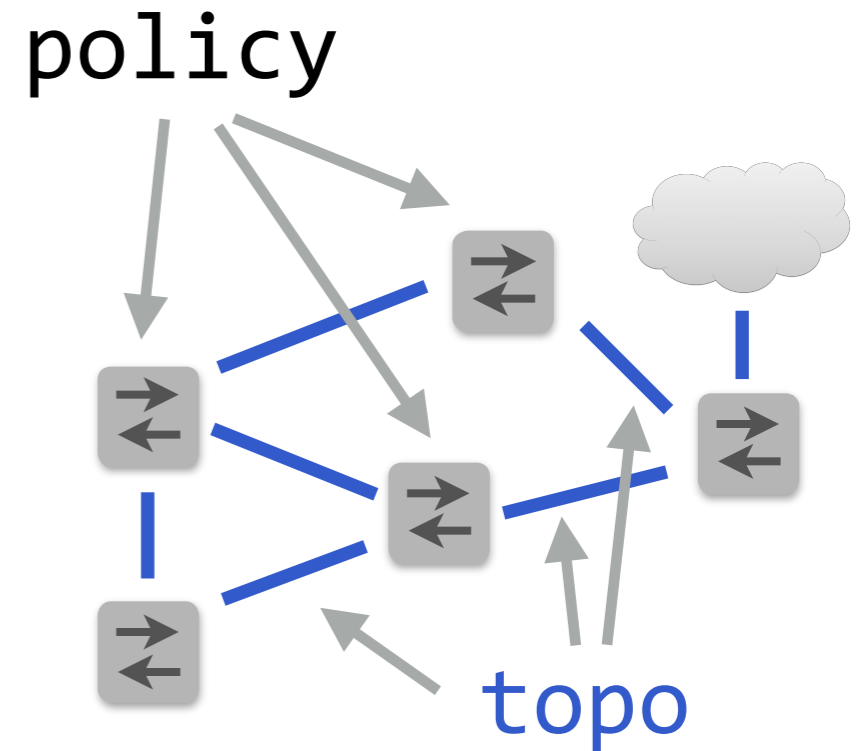
An entire network can be encoded by interleaving policy and topology processing steps arbitrarily many number of times



$$\begin{array}{c} \text{id} \\ + \\ (\text{policy} \bullet \text{topo}) \end{array}$$

# Encoding Networks

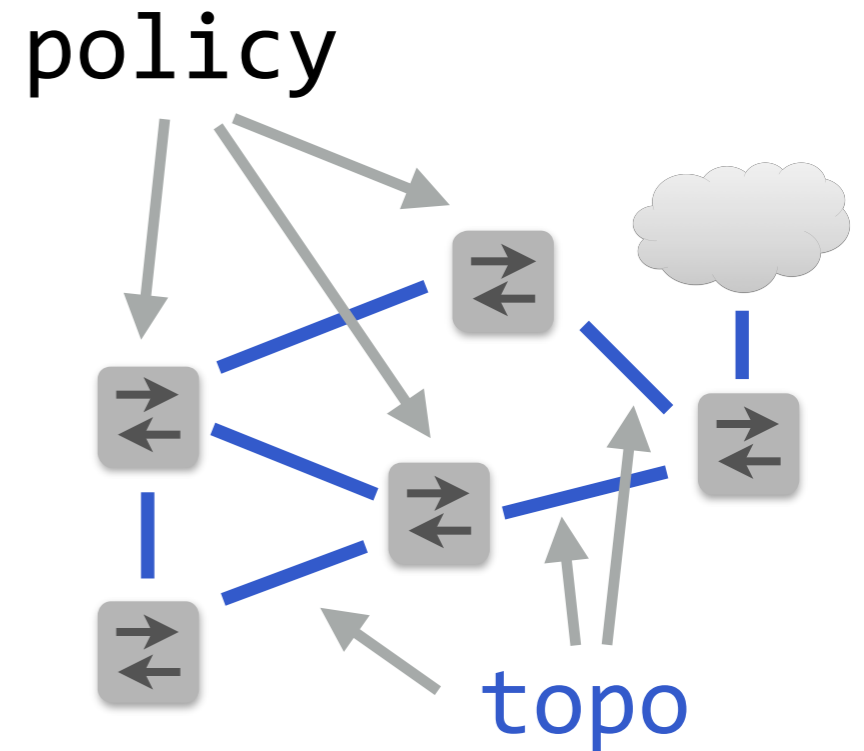
An entire network can be encoded by interleaving policy and topology processing steps arbitrarily many number of times



$$\begin{aligned} & \text{id} \\ & + \\ & (\text{policy} \bullet \text{topo}) \\ & + \\ & (\text{policy} \bullet \text{topo} \bullet \text{policy} \bullet \text{topo}) \end{aligned}$$

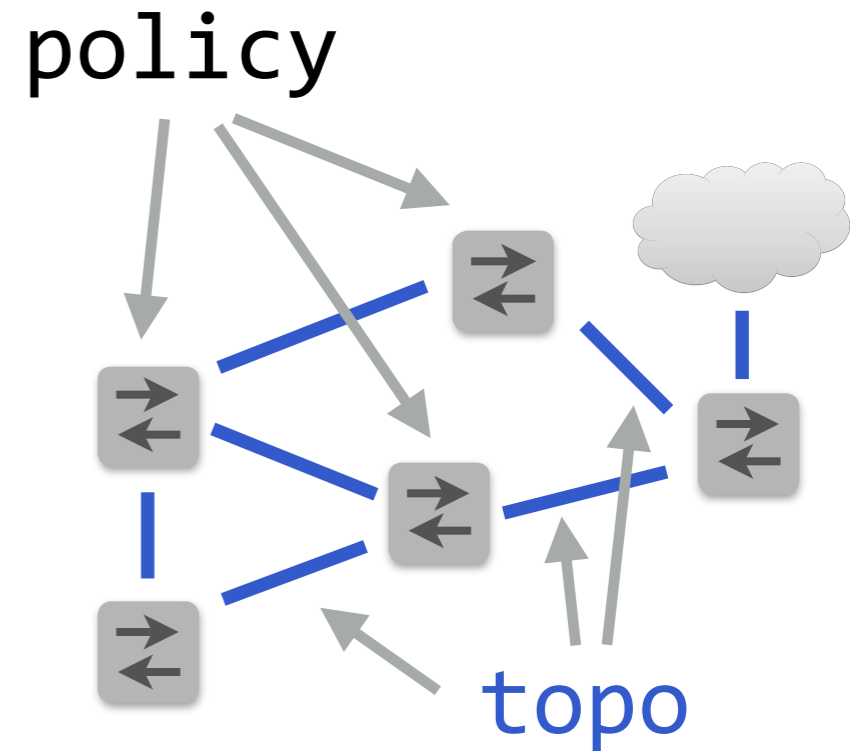
# Encoding Networks

An entire network can be encoded by interleaving policy and topology processing steps arbitrarily many number of times


$$\begin{aligned} & \text{id} \\ & + \\ & (\text{policy} \bullet \text{topo}) \\ & + \\ & (\text{policy} \bullet \text{topo} \bullet \text{policy} \bullet \text{topo}) \\ & + \\ & (\text{policy} \bullet \text{topo} \bullet \text{policy} \bullet \text{topo} \bullet \text{policy} \bullet \text{topo}) \end{aligned}$$

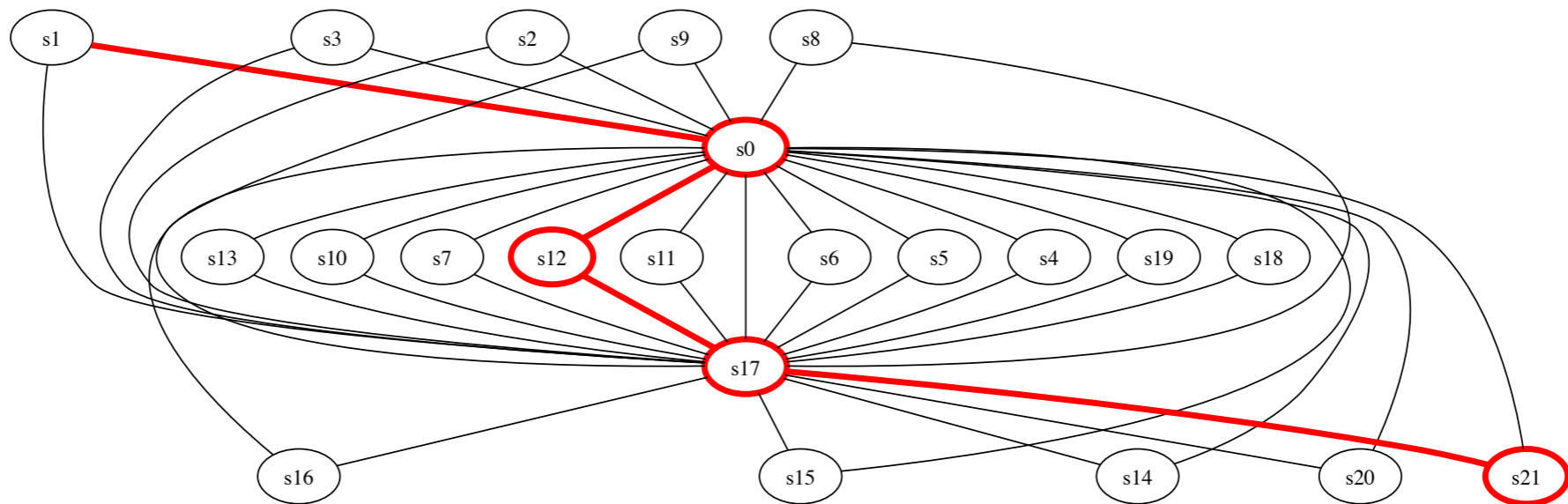
# Encoding Networks

An entire network can be encoded by interleaving policy and topology processing steps arbitrarily many number of times



$$\begin{aligned} & \text{id} \\ & + \\ & (\text{policy} \bullet \text{topo}) \\ & + \\ & (\text{policy} \bullet \text{topo} \bullet \text{policy} \bullet \text{topo}) \\ & + \\ & (\text{policy} \bullet \text{topo} \bullet \text{policy} \bullet \text{topo} \bullet \text{policy} \bullet \text{topo}) \\ & \dots \\ & + \\ & (\text{policy} \bullet \text{topo})^* \end{aligned}$$

# Checking Reachability



Given:

- Ingress predicate:  $\text{switch} = s_1$
- Egress predicate:  $\text{switch} = s_{21}$
- Topology:  $t$
- Switch program:  $p$

Check:

- **$\text{switch} = s_1 \bullet \text{switch} := s_{21} + (p \bullet t)^* \sim (p \bullet t)^*$**
- **$\text{switch} = s_1 \bullet (p \bullet t)^* \bullet \text{switch} = s_{21} \sim \text{drop}$**

# NetKAT Equational Axioms

## Kleene Algebra Axioms

$$p + (q + r) \sim (p + q) + r$$

$$p + q \sim q + p$$

$$p + \mathbf{drop} \sim p$$

$$p + p \sim p$$

$$p \cdot (q \cdot r) \sim (p \cdot q) \cdot r$$

$$p \cdot (q + r) \sim p \cdot q + p \cdot r$$

$$(p + q) \cdot r \sim p \cdot r + q \cdot r$$

$$\mathbf{id} \cdot p \sim p$$

$$p \sim p \cdot \mathbf{id}$$

$$\mathbf{drop} \cdot p \sim \mathbf{drop}$$

$$p \cdot \mathbf{drop} \sim \mathbf{drop}$$

$$\mathbf{id} + p \cdot p^* \sim p^*$$

$$\mathbf{id} + p^* \cdot p \sim p^*$$

$$p + q \cdot r + r \sim r \Rightarrow p^* \cdot q + r \sim r$$

$$p + q \cdot r + q \sim q \Rightarrow p \cdot r^* + q \sim q$$

## Boolean Algebra Axioms

$$a \parallel (b \ \&\& \ c) \sim (a \parallel b) \ \&\& \ (a \parallel c)$$

$$a \parallel \mathbf{true} \sim \mathbf{true}$$

$$a \parallel !a \sim \mathbf{true}$$

$$a \ \&\& \ b \sim b \ \&\& \ a$$

$$a \ \&\& \ !a \sim \mathbf{false}$$

$$a \ \&\& \ a \sim a$$

## Packet Axioms

$$f := n \cdot f' := n' \sim f' := n' \cdot f := n \quad \text{if } f \neq f'$$

$$f := n \cdot f' = n' \sim f' = n' \cdot f := n \quad \text{if } f \neq f'$$

$$f := n \cdot f = n \sim f := n$$

$$f = n \cdot f := n \sim f = n$$

$$f := n \cdot f := n' \sim f := n'$$

$$f = n \cdot f = n' \sim \mathbf{drop} \quad \text{if } n \neq n'$$

$$\mathbf{dup} \cdot f = n \sim f = n \cdot \mathbf{dup}$$

# Metatheory

**Soundness:** If  $\vdash p \sim q$ , then  $\llbracket p \rrbracket = \llbracket q \rrbracket$

**Completeness:** If  $\llbracket p \rrbracket = \llbracket q \rrbracket$ , then  $\vdash p \sim q$

# Metatheory

**Soundness:** If  $\vdash p \sim q$ , then  $\llbracket p \rrbracket = \llbracket q \rrbracket$

**Completeness:** If  $\llbracket p \rrbracket = \llbracket q \rrbracket$ , then  $\vdash p \sim q$

NetKAT equivalence is also decidable! 😊

...But our earlier algorithm was based on determining a non-deterministic algorithm using Savitch's theorem, so it was PSPACE in the best case and the worst case 😞



# Metatheory

**Soundness:** If  $\vdash p \sim q$ , then  $\llbracket p \rrbracket = \llbracket q \rrbracket$

**Completeness:** If  $\llbracket p \rrbracket = \llbracket q \rrbracket$ , then  $\vdash p \sim q$

NetKAT equivalence is also decidable! 😊

...But our earlier algorithm was based on determining a non-deterministic algorithm using Savitch's theorem, so it was PSPACE in the best case and the worst case 😞

**Roadmap:** starting from a language model of NetKAT...

- Develop coalgebraic structure of NetKAT
- Check equivalence using bisimulation
- Deploy a host of cunning tricks to make it fast

# Regular Expression Derivatives Review

# Regular Expressions

*Syntax*

$R ::= \mathbf{0}$	(* empty *)
$c$	(* character *)
$R_1 + R_2$	(* union *)
$R_1 \cdot R_2$	(* concatenation *)
$R^*$	(* Kleene star *)

*Semantics*

$\llbracket R \rrbracket \in \Sigma^*$
$\llbracket \mathbf{0} \rrbracket = \{\}$
$\llbracket c \rrbracket = \{c\}$
$\llbracket R_1 + R_2 \rrbracket = \llbracket R_1 \rrbracket \cup \llbracket R_2 \rrbracket$
$\llbracket R_1 \cdot R_2 \rrbracket = \llbracket R_1 \rrbracket \cdot \llbracket R_2 \rrbracket$
$\llbracket R^* \rrbracket = ( \cup_i \llbracket R \rrbracket^i )$

# Language Derivatives

*Semantic*

$$\partial_c R = \{ w \mid c \cdot w \in \llbracket R \rrbracket \}$$

# Language Derivatives

*Semantic*

$$\partial_c R = \{ w \mid c \cdot w \in \llbracket R \rrbracket \}$$

*Syntactic*

*Continuation map*

$$D_c(\mathbf{0}) = \mathbf{0}$$

$$D_c(b) = \begin{cases} c & \text{if } c = b \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$D_c(R_1 + R_2) = D_c(R_1) + D_c(R_2)$$

$$D_c(R_1 \cdot R_2) = D_c(R_1) \cdot R_2 + E(R_1) \cdot D_c(R_2)$$

$$D_c(R^*) = D_c(R) \cdot R^*$$

# Language Derivatives

*Semantic*

$$\partial_c R = \{ w \mid c \cdot w \in \llbracket R \rrbracket \}$$

*Syntactic*

*Continuation map*

$$D_c(\mathbf{0}) = \mathbf{0}$$

$$D_c(b) = \begin{cases} c & \text{if } c = b \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$D_c(R_1 + R_2) = D_c(R_1) + D_c(R_2)$$

$$D_c(R_1 \cdot R_2) = D_c(R_1) \cdot R_2 + E(R_1) \cdot D_c(R_2)$$

$$D_c(R^*) = D_c(R) \cdot R^*$$

*Observation map*

$$E(\mathbf{0}) = E(c) = \mathbf{0}$$

$$E(R_1 + R_2) = E(R_1) \parallel E(R_2)$$

$$E(R_1 \cdot R_2) = E(R_1) \&\& E(R_2)$$

$$E(R^*) = \mathbf{1}$$

# Language Derivatives

*Semantic*

$$\partial_c R = \{ w \mid c \cdot w \in \llbracket R \rrbracket \}$$

*Syntactic*

*Continuation map*

**Theorem [Brzozowski '64]:** every regular expression has a finite number of derivatives (modulo ACI equivalence)

$$D_c(R_1 \cdot R_2) = D_c(R_1) \cdot R_2 + E(R_1) \cdot D_c(R_2)$$

$$D_c(R^*) = D_c(R) \cdot R^*$$

*Observation map*

$$E(\mathbf{0}) = E(c) = \mathbf{0}$$

$$E(R_1 + R_2) = E(R_1) \parallel E(R_2)$$

$$E(R_1 \cdot R_2) = E(R_1) \&\& E(R_2)$$

$$E(R^*) = \mathbf{1}$$

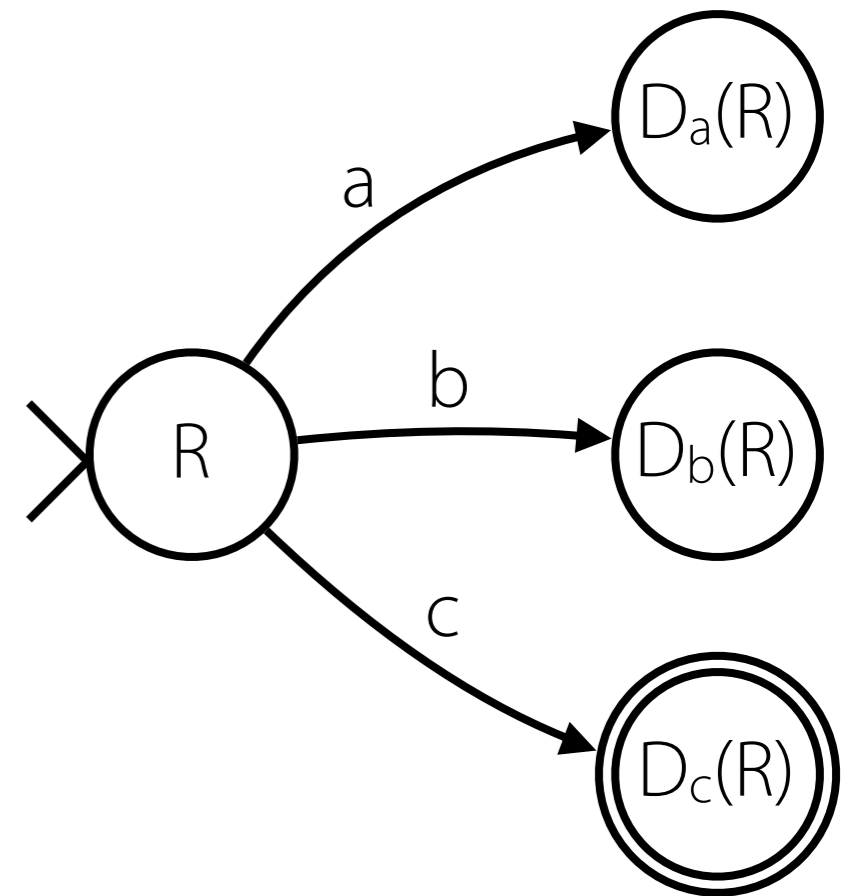
# Building Automata using Derivatives

## Automaton:

- Label initial state by  $R$
- Transition from  $R_i$  to  $R_i'$  on  $c$  if  $D_c(R_i) = R_i'$
- Label state  $R_i$  as final if  $E(R_i) = 1$
- Only generate new state for expressions not seen previously, modulo ACI

## Advantages:

- Extremely simple
- Easy to make lazy
- Easy to extend with negative operators
- Easy to optimize by recognizing coarser equivalences (language equivalence leads to minimal automaton)

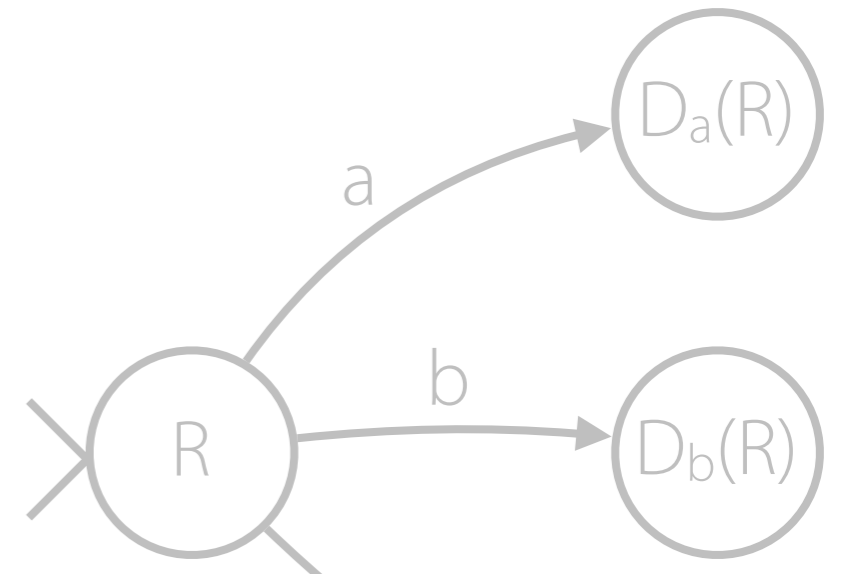




# Building Automata using Derivatives

## Automaton:

- Label initial state by  $R$
- Transition from  $R_i$  to  $R_i'$  on  $c$  if  $D_c(R_i) = R_i'$
- Label state  $R_i$  as final if  $E(R_i) = 1$
- Only generate new state for expressions



Can also build NFAs using a variant called the Antimirov derivative

## Advantages:

- Extremely simple
- Easy to make lazy
- Easy to extend with negative operators
- Easy to optimize by recognizing coarser equivalences (language equivalence leads to minimal automaton)

# NetKAT Derivatives

# Reduced NetKAT

*Complete tests*

$\alpha ::= \mathbf{switch} = n \bullet \mathbf{port} = n$

*Complete assignments*

$\beta ::= \mathbf{switch} := n \bullet \mathbf{port} := n$

*Reduced terms*


$p, q ::= \alpha$	(* complete test *)
$\beta$	(* complete assignment *)
$p + q$	(* union *)
$p \bullet q$	(* sequence *)
$p^*$	(* Kleene star *)
<b>dup</b>	(* Duplication *)

# Reduced NetKAT

*Complete tests*

$\alpha ::= \mathbf{switch} = n \bullet \mathbf{port} = n$

*For simplicity, only consider two fields*



*Complete assignments*

$\beta ::= \mathbf{switch} := n \bullet \mathbf{port} := n$

*Reduced terms*


$p, q ::= \alpha$	(* complete test *)
$\beta$	(* complete assignment *)
$p + q$	(* union *)
$p \bullet q$	(* sequence *)
$p^*$	(* Kleene star *)
<b>dup</b>	(* Duplication *)

# Reduced NetKAT

*Complete tests*

$\alpha ::= \mathbf{switch} = n \bullet \mathbf{port} = n$

*For simplicity, only consider two fields*



*Complete assignments*

$\beta ::= \mathbf{switch} := n \bullet \mathbf{port} := n$

*Reduced terms*

$p, q ::= \alpha$	(* complete test *)
$\beta$	(* complete assignment *)
$p + q$	(* union *)
$p \bullet q$	(* sequence *)
$p^*$	(* Kleene star *)
<b>dup</b>	(* Duplication *)

**Lemma:** For every NetKAT term  $p$ , there is a reduced NetKAT term  $p'$  such that  $\vdash p \sim p'$

# Regular Interpretation

Can interpret reduced terms as regular languages over an “alphabet” of complete tests, complete assignments, and **dup**:

# Regular Interpretation

Can interpret reduced terms as regular languages over an “alphabet” of complete tests, complete assignments, and **dup**:

*Regular Interpretation:*  $R(p) \subseteq (A \cup B \cup \{\mathbf{dup}\})^*$

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

$$R(\beta) = \{\beta\}$$

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

$$R(p \cdot q) = R(p) \cdot R(q)$$

$$R(p^*) = R(p)^*$$

$$R(\mathbf{dup}) = \{\mathbf{dup}\}$$

# Regular Interpretation

Can interpret reduced terms as regular languages over an “alphabet” of complete tests, complete assignments, and **dup**:

*Regular Interpretation:*  $R(p) \subseteq (A \cup B \cup \{\mathbf{dup}\})^*$

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

$$R(\beta) = \{\beta\}$$

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

$$R(p \cdot q) = R(p) \cdot R(q)$$

$$R(p^*) = R(p)^*$$

$$R(\mathbf{dup}) = \{\mathbf{dup}\}$$

Unfortunately  $\llbracket p \rrbracket = \llbracket q \rrbracket$  does not imply  $R(p) = R(q)$



# Regular Interpretation

Can interpret reduced terms as regular languages over an “alphabet” of complete tests, complete assignments, and **dup**:

*Regular Interpretation:*  $R(p) \subseteq (A \cup B \cup \{\mathbf{dup}\})^*$

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

$$R(\beta) = \{\beta\}$$

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

$$R(p \cdot q) = R(p) \cdot R(q)$$

$$R(p^*) = R(p)^*$$

$$R(\mathbf{dup}) = \{\mathbf{dup}\}$$

Unfortunately  $\llbracket p \rrbracket = \llbracket q \rrbracket$  does not imply  $R(p) = R(q)$

## Counterexample:

$$\begin{aligned} & \underline{\text{switch}=1 \cdot \text{port}=1} \cdot \underline{\text{switch}=1 \cdot \text{port}=2} \sim \\ & \underline{\text{switch}=1 \cdot \text{port}=1} \cdot \underline{\text{switch}=2 \cdot \text{port}=1} \end{aligned}$$

# Language Model

# Language Model

*Language Interpretation:*  $G(p) \subseteq A \cdot (B \cdot \{\mathbf{dup}\})^* \cdot B$

$$G(a) = \{a \cdot \pi_a\}$$

$$G(\beta) = \{a \cdot \beta \mid a \in A\}$$

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

$$G(p \cdot q) = G(p) \diamond G(q)$$

$$G(p^*) = G(p)^*$$

$$G(\mathbf{dup}) = \{a \cdot \beta_a \cdot \mathbf{dup} \cdot \beta_a \mid a \in A\}$$

# Language Model

*Language Interpretation:*  $G(p) \subseteq A \cdot (B \cdot \{\mathbf{dup}\})^* \cdot B$

$$G(a) = \{a \cdot \pi_a\}$$

$$G(\beta) = \{a \cdot \beta \mid a \in A\}$$

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

$$G(p \cdot q) = G(p) \diamond G(q)$$

$$G(p^*) = G(p)^*$$

$$G(\mathbf{dup}) = \{a \cdot \beta_a \cdot \mathbf{dup} \cdot \beta_a \mid a \in A\}$$

*Guarded  
strings*



# Language Model

*Language Interpretation:*  $G(p) \subseteq A \cdot (B \cdot \{\mathbf{dup}\})^* \cdot B$

$$G(a) = \{a \cdot \pi_a\}$$

$$G(\beta) = \{a \cdot \beta \mid a \in A\}$$

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

$$G(p \cdot q) = G(p) \diamond G(q)$$

$$G(p^*) = G(p)^*$$

$$G(\mathbf{dup}) = \{a \cdot \beta_a \cdot \mathbf{dup} \cdot \beta_a \mid a \in A\}$$

*Guarded strings*

*Guarded concatenation*

# Language Model

*Language Interpretation:*  $G(p) \subseteq A \cdot (B \cdot \{\mathbf{dup}\})^* \cdot B$

$$G(a) = \{a \cdot \pi_a\}$$

$$G(\beta) = \{a \cdot \beta \mid a \in A\}$$

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

$$G(p \cdot q) = G(p) \diamond G(q)$$

$$G(p^*) = G(p)^*$$

$$G(\mathbf{dup}) = \{a \cdot \beta_a \cdot \mathbf{dup} \cdot \beta_a \mid a \in A\}$$

*Guarded strings*

*Guarded concatenation*

**Example:**  $a_1 \cdot \beta_2 \cdot \mathbf{dup} \cdot \beta_3 \cdot \mathbf{dup} \cdot \dots \cdot \mathbf{dup} \cdot \beta_n$

# Language Model

*Language Interpretation:*  $G(p) \subseteq A \cdot (B \cdot \{\mathbf{dup}\})^* \cdot B$

$$G(\alpha) = \{\alpha \cdot \pi_\alpha\}$$

$$G(\beta) = \{\alpha \cdot \beta \mid \alpha \in A\}$$

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

$$G(p \cdot q) = G(p) \diamond G(q)$$

$$G(p^*) = G(p)^*$$

$$G(\mathbf{dup}) = \{\alpha \cdot \beta_\alpha \cdot \mathbf{dup} \cdot \beta_\alpha \mid \alpha \in A\}$$

*Guarded strings*

*Guarded concatenation*

**Example:**  $\alpha_1 \cdot \beta_2 \cdot \mathbf{dup} \cdot \beta_3 \cdot \mathbf{dup} \cdot \dots \cdot \mathbf{dup} \cdot \beta_n$

**Intuition:** models trajectories through the network

# Language Model

*Language Interpretation:*  $G(p) \subseteq A \cdot (B \cdot \{\mathbf{dup}\})^* \cdot B$

$$G(\alpha) = \{\alpha \cdot \pi_\alpha\}$$

$$G(\beta) = \{\alpha \cdot \beta \mid \alpha \in A\}$$

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

$$G(p \cdot q) = G(p) \diamond G(q)$$

$$G(p^*) = G(p)^*$$

$$G(\mathbf{dup}) = \{\alpha \cdot \beta_\alpha \cdot \mathbf{dup} \cdot \beta_\alpha \mid \alpha \in A\}$$

*Guarded strings*

*Guarded concatenation*

**Example:**  $\alpha_1 \cdot \beta_2 \cdot \mathbf{dup} \cdot \beta_3 \cdot \mathbf{dup} \cdot \dots \cdot \mathbf{dup} \cdot \beta_n$

**Intuition:** models trajectories through the network

**Theorem:**  $\llbracket p \rrbracket = \llbracket q \rrbracket$  if and only if  $G(p) = G(q)$



# NetKAT Derivatives

**Goal:** match all of the guarded strings of the form

$$A \cdot (B \cdot \{\mathbf{dup}\})^* \cdot B$$

in the set denoted by a given NetKAT term  $p$

**Continuation map**  $D_{\alpha\beta}(p)$ :

- Attempts to match  $\alpha \cdot \beta \cdot \mathbf{dup}$  at the start of string
- Returns the residual NetKAT term, if successful or **drop** if not
- Note that we elide **dup** to streamline the notation

**Observation map**  $E_{\alpha\beta}(p)$ :

- Tries to match the final  $\alpha \cdot \beta$  at the end of the string
- Returns a term equivalent to **true** if successful or **false** if not

# NetKAT Derivatives

# NetKAT Derivatives

*Continuation Map:*

$$D_{\alpha\beta}(\mathbf{f} = \mathbf{n}) = D_{\alpha\beta}(\mathbf{f} := \mathbf{n}) = \mathbf{drop}$$

$$D_{\alpha\beta}(\mathbf{dup}) = \alpha \cdot [\alpha = \beta]$$

$$D_{\alpha\beta}(p + q) = D_{\alpha\beta}(p) + D_{\alpha\beta}(q)$$

$$D_{\alpha\beta}(p \cdot q) = D_{\alpha\beta}(p) \cdot q + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot D_{\gamma\beta}(q)$$

$$D_{\alpha\beta}(p^*) = D_{\alpha\beta}(p) \cdot p^* + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot D_{\gamma\beta}(p^*)$$

# NetKAT Derivatives

*Continuation Map:*

$$D_{\alpha\beta}(\mathbf{f} = \mathbf{n}) = D_{\alpha\beta}(\mathbf{f} := \mathbf{n}) = \mathbf{drop}$$

$$D_{\alpha\beta}(\mathbf{dup}) = \alpha \cdot [\alpha = \beta]$$

$$D_{\alpha\beta}(p + q) = D_{\alpha\beta}(p) + D_{\alpha\beta}(q)$$

$$D_{\alpha\beta}(p \cdot q) = D_{\alpha\beta}(p) \cdot q + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot D_{\gamma\beta}(q)$$

$$D_{\alpha\beta}(p^*) = D_{\alpha\beta}(p) \cdot p^* + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot D_{\gamma\beta}(p^*)$$

*Observation Map:*

$$E_{\alpha\beta}(\mathbf{f} = \mathbf{n}) = [\alpha = \beta \leq \mathbf{f} = \mathbf{n}]$$

$$E_{\alpha\beta}(\mathbf{dup}) = \mathbf{drop}$$

$$E_{\alpha\beta}(\mathbf{f} := \mathbf{n}) = [\mathbf{f} := \mathbf{n} = p_{\beta}]$$

$$E_{\alpha\beta}(p + q) = E_{\alpha\beta}(p) + E_{\alpha\beta}(q)$$

$$E_{\alpha\beta}(p \cdot q) = \sum_{\gamma} E_{\alpha\gamma}(p) \cdot E_{\gamma\beta}(q)$$

$$E_{\alpha\beta}(p^*) = [\alpha = \beta] + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot E_{\gamma\beta}(p^*)$$

# NetKAT Derivatives

*Continuation Map:*

$$D_{\alpha\beta}(\mathbf{f} = \mathbf{n}) = D_{\alpha\beta}(\mathbf{f} := \mathbf{n}) = \mathbf{drop}$$

$$D_{\alpha\beta}(\mathbf{dup}) = \alpha \cdot [\alpha = \beta]$$

$$D_{\alpha\beta}(p + q) = D_{\alpha\beta}(p) + D_{\alpha\beta}(q)$$

$$D_{\alpha\beta}(p \cdot q) = D_{\alpha\beta}(p) \cdot q + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot D_{\gamma\beta}(q)$$

**Lemma [Foster et al. '14]:** every NetKAT term has a finite number of derivatives (modulo ACI equivalence)

$$E_{\alpha\beta}(\mathbf{f} = \mathbf{n}) = [\alpha = \beta \leq \mathbf{f} = \mathbf{n}]$$

$$E_{\alpha\beta}(\mathbf{dup}) = \mathbf{drop}$$

$$E_{\alpha\beta}(\mathbf{f} := \mathbf{n}) = [\mathbf{f} := \mathbf{n} = p_{\beta}]$$

$$E_{\alpha\beta}(p + q) = E_{\alpha\beta}(p) + E_{\alpha\beta}(q)$$

$$E_{\alpha\beta}(p \cdot q) = \sum_{\gamma} E_{\alpha\gamma}(p) \cdot E_{\gamma\beta}(q)$$

$$E_{\alpha\beta}(p^*) = [\alpha = \beta] + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot E_{\gamma\beta}(p^*)$$

# Matrix Representation

***Observation:*** can streamline definitions using matrices

# Matrix Representation

**Observation:** can streamline definitions using matrices

*Continuation Map:*

$$D(\mathbf{f} = \mathbf{n}) = D(\mathbf{f} := \mathbf{n}) = \mathbf{drop}$$

$$D(\mathbf{dup}) = J$$

$$D(p + q) = D(p) + D(q)$$

$$D(p \cdot q) = D(p) \cdot I(q) + E(p) \cdot D(q)$$

$$D(p^*) = E(p^*) \cdot D(p) \cdot I(p^*)$$

# Matrix Representation

**Observation:** can streamline definitions using matrices

*Continuation Map:*

$$D(\mathbf{f} = \mathbf{n}) = D(\mathbf{f} := \mathbf{n}) = \mathbf{drop}$$

$$D(\mathbf{dup}) = J$$

$$D(p + q) = D(p) + D(q)$$

$$D(p \cdot q) = D(p) \cdot I(q) + E(p) \cdot D(q)$$

$$D(p^*) = E(p^*) \cdot D(p) \cdot I(p^*)$$

*Matrix with as  
on diagonal and 0s  
everywhere else*





# Matrix Representation

**Observation:** can streamline definitions using matrices

*Continuation Map:*

$$D(\mathbf{f} = \mathbf{n}) = D(\mathbf{f} := \mathbf{n}) = \mathbf{drop}$$


$$D(\mathbf{dup}) = J$$

$$D(p + q) = D(p) + D(q)$$

$$D(p \cdot q) = D(p) \cdot I(q) + E(p) \cdot D(q)$$

$$D(p^*) = E(p^*) \cdot D(p) \cdot I(p^*)$$

*Matrix with as  
on diagonal and 0s  
everywhere else*



*Observation Map:*

$$E(\mathbf{f} = \mathbf{n}) = \dots$$

$$E(\mathbf{dup}) = \mathbf{false}$$

$$E(\mathbf{f} := \mathbf{n}) = \dots$$

$$E(p + q) = E(p) + E(q)$$

$$E(p \cdot q) = E(p) \cdot E(q)$$

$$E(p^*) = E(p)^*$$

# Implementation and Experiments

# Implementation Highlights

## **Representations:**

- Bases encode sets of complete tests and assignments
- “Spines” encode sets of terms
- Sparse matrix library

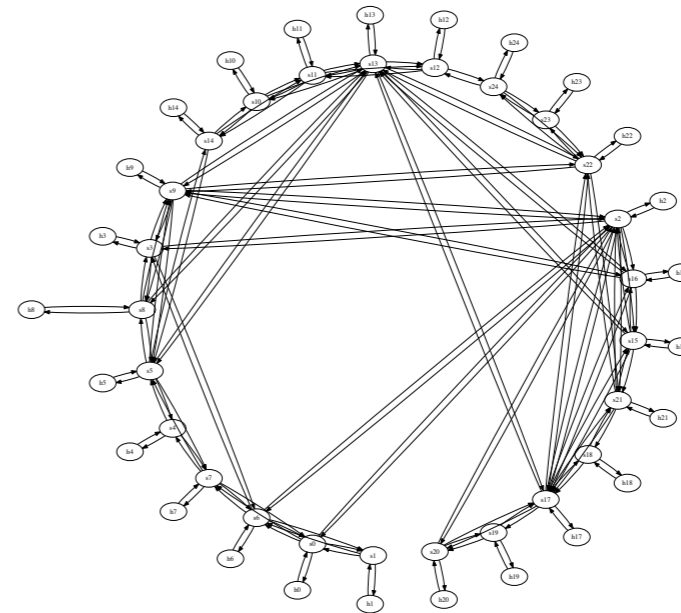
## **Algorithmic optimizations:**

- Smart constructors
- Hash consing
- Memoization
- Base set compaction
- Fast multiplication
- Fast fixpoints
- Union-find in bisimulation

# Experiments

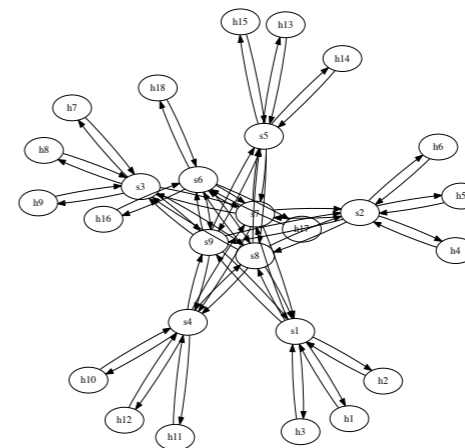
## Networks:

- Topology Zoo
- FatTree
- Stanford Backbone



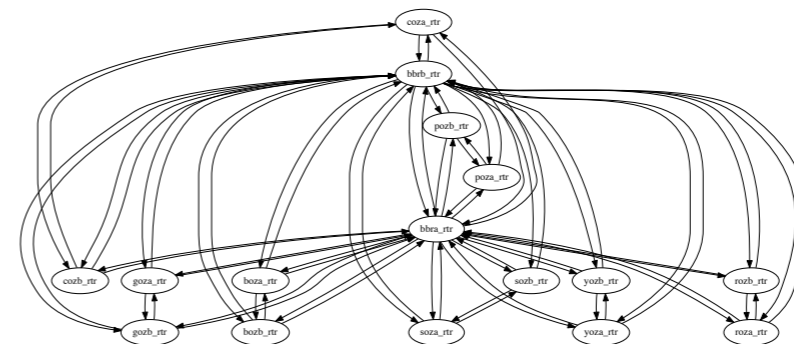
## Policies:

- Shortest-path forwarding
- Stanford production policy



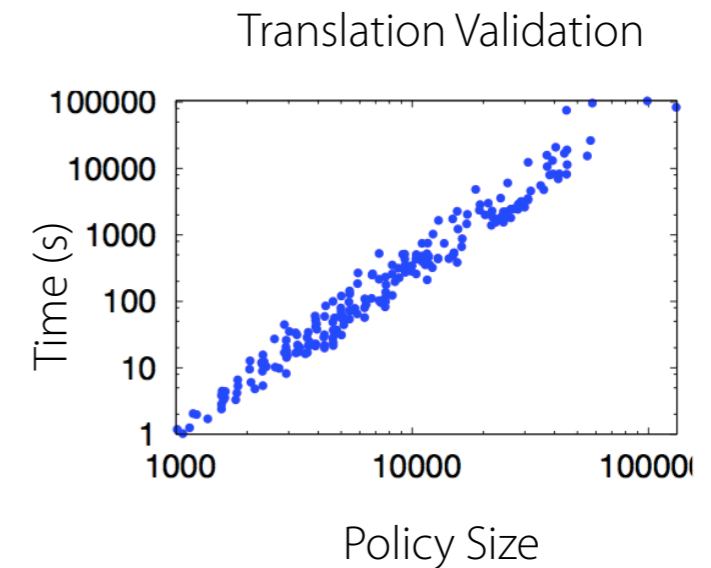
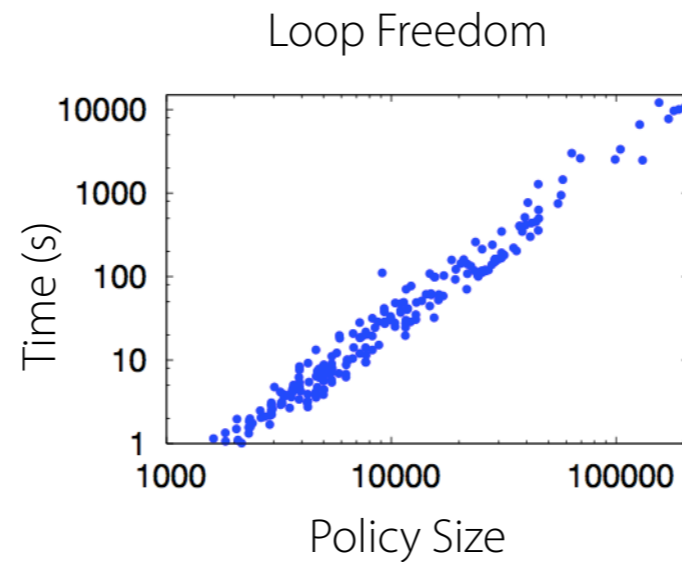
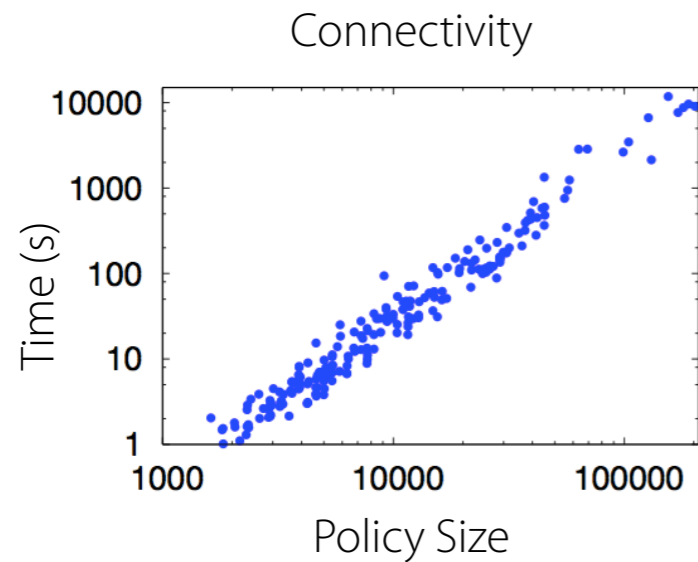
## Questions:

- Point-to-point reachability
- All-Pairs connectivity
- Loop freedom
- Translation validation

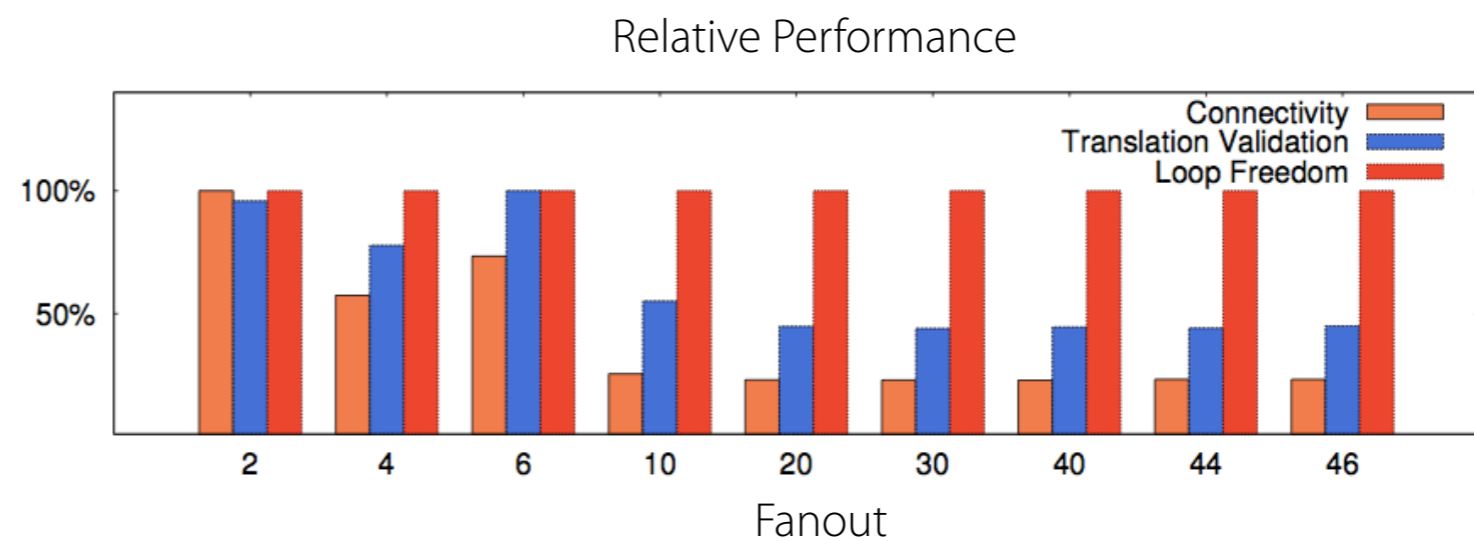
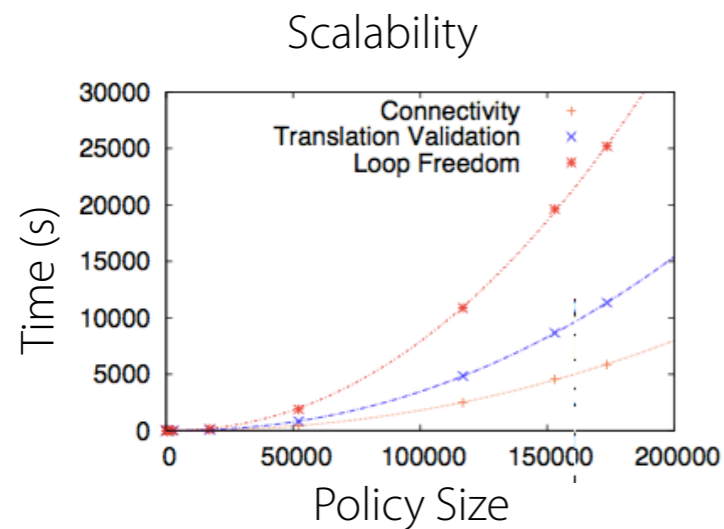


# Results

## Topology Zoo



## FatTree



## Stanford Backbone

Point-to-point reachability in 0.67s (vs 13s for HSA)

# Conclusion

- Still (!) lots of great PL problems in networking
- SDN is an enabling technology for this kind of research
- NetKAT is a new framework for programming and reasoning about network behavior
- Brzozowski derivatives are an elegant technique for building automata that has borne fruit for 40 years and counting...
- Ongoing work
  - Proof carrying code
  - Probabilistic NetKAT
  - Network-wide optimizations using matrices

# Thank you!

## ***Collaborators***

- Dexter Kozen (Cornell)
- Matthew Milano (Cornell)
- Alexandra Silva (Nijmegen)
- Laure Thompson (Cornell)

## ***Papers, code, etc.***

<http://frenetic-lang.org/>



**frenetic**