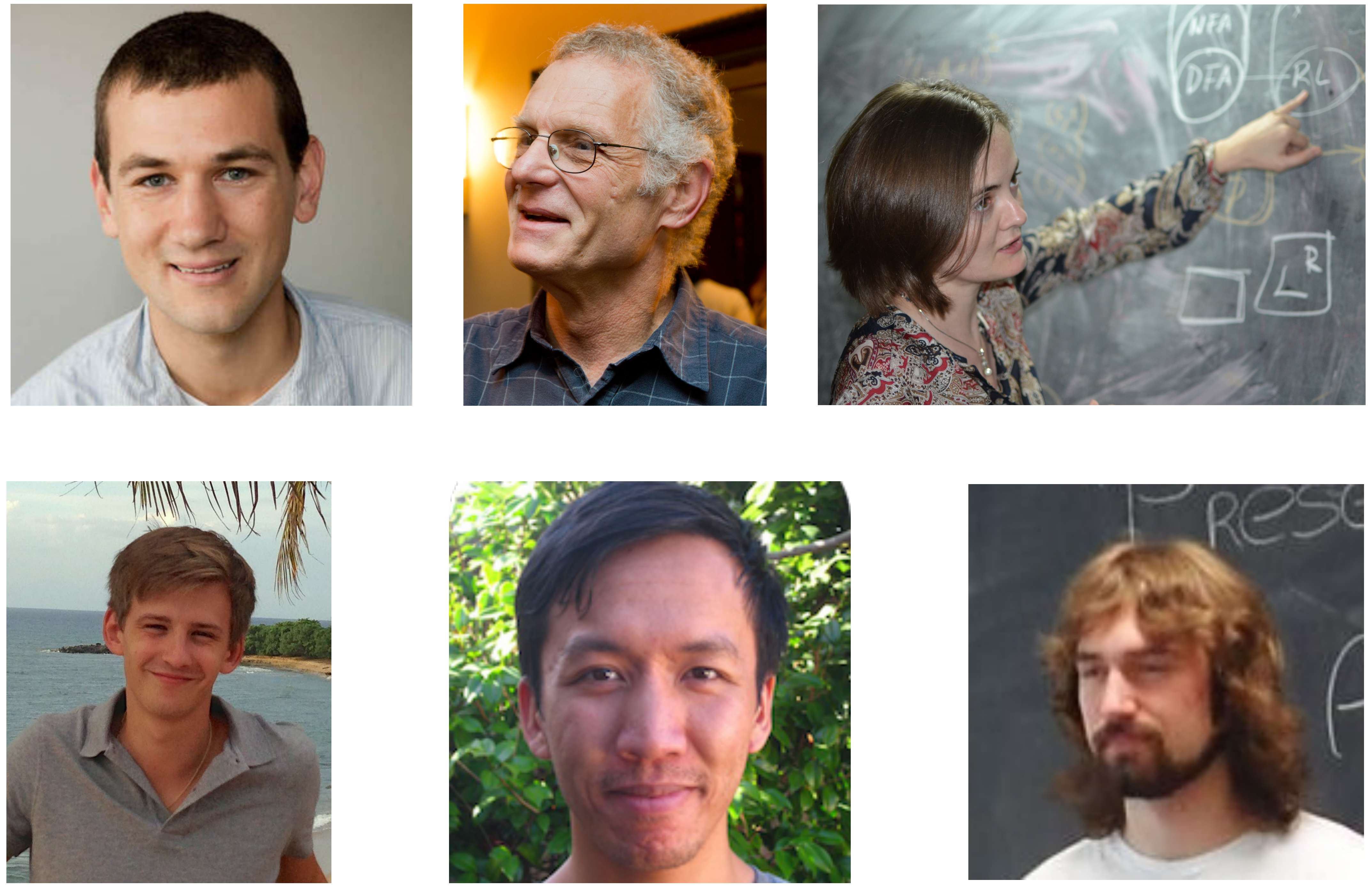
Guarded Kleene Algebra with Tests

Alexandra Silva

Credits



This talk

This talk

Verification Problem

Reachability in Networks

This talk

Verification Problem

Reachability in Networks

Solution

Use of Kleene Algebra
and Automata

This talk

Verification Problem
Reachability in Networks

Solution
Use of Kleene Algebra and Automata

Great Performance
10x state of the art 😊

This talk

Verification Problem
Reachability in Networks

Solution
Use of Kleene Algebra
and Automata

Great Performance
10x state of the art 😊

Did not make
complete sense 🤔

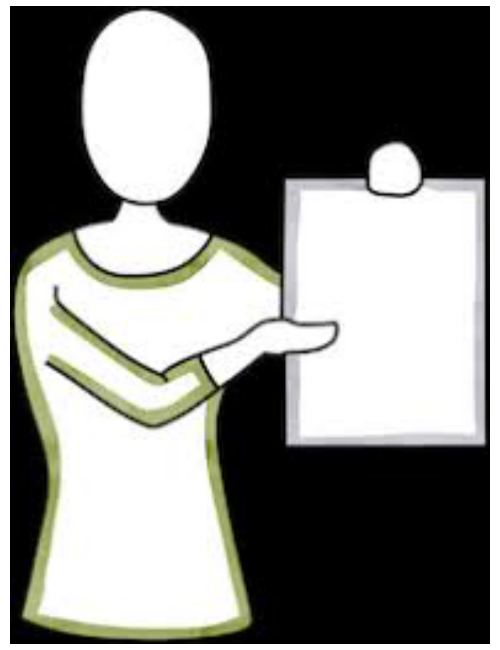
This talk

Verification Problem
Reachability in Networks

Solution
Use of Kleene Algebra
and Automata

Great Performance
10x state of the art

Did not make
complete sense



Let’s start from the beginning…

Verification of networks

Trend in PL&Verification after Software-Defined Networks

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

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

NetKAT

NetKAT

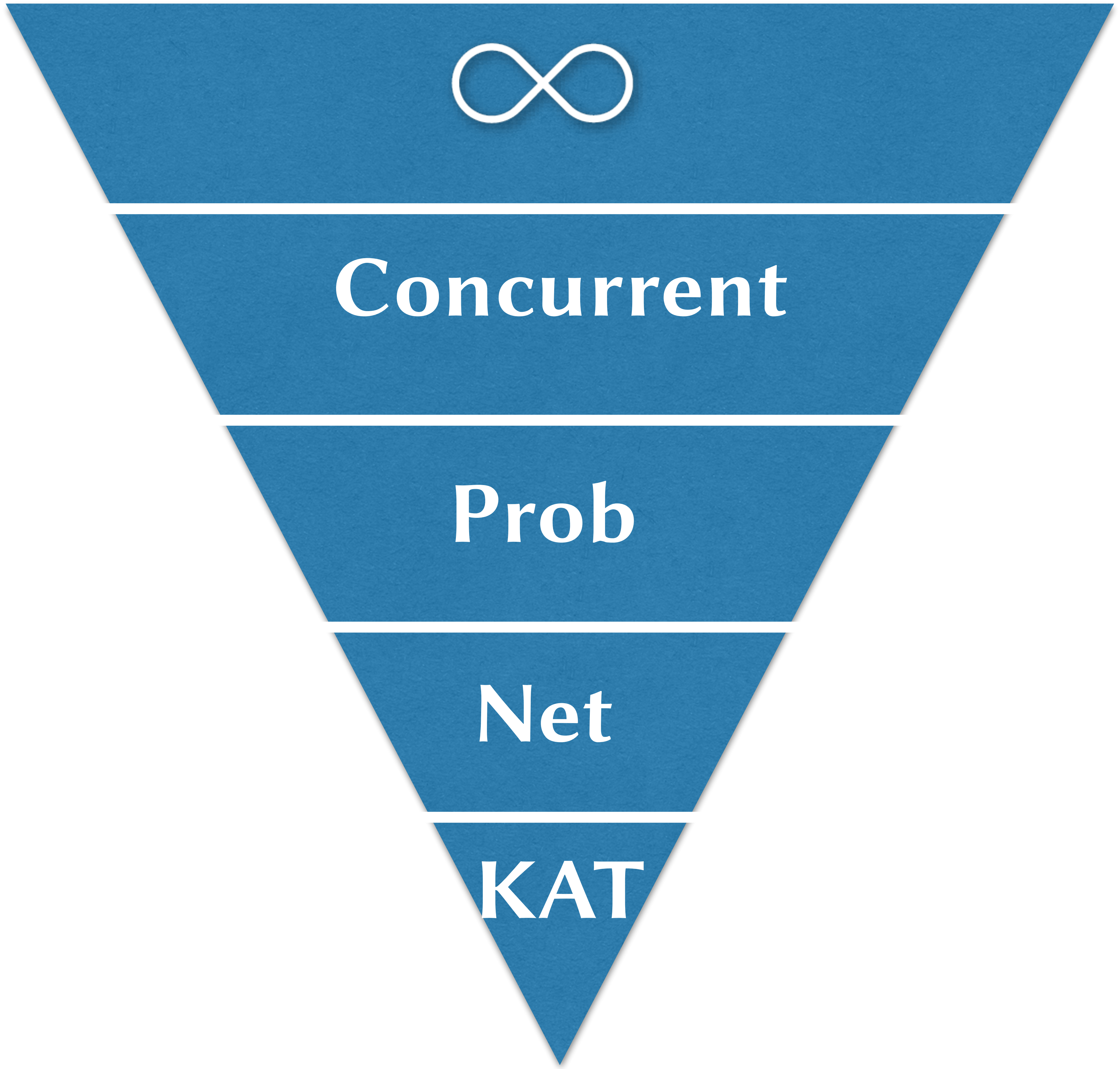
$=$

Kleene algebra with tests (KAT)

$+$

additional specialized constructs particular to network topology and packet switching

The KAT tower principle



∞

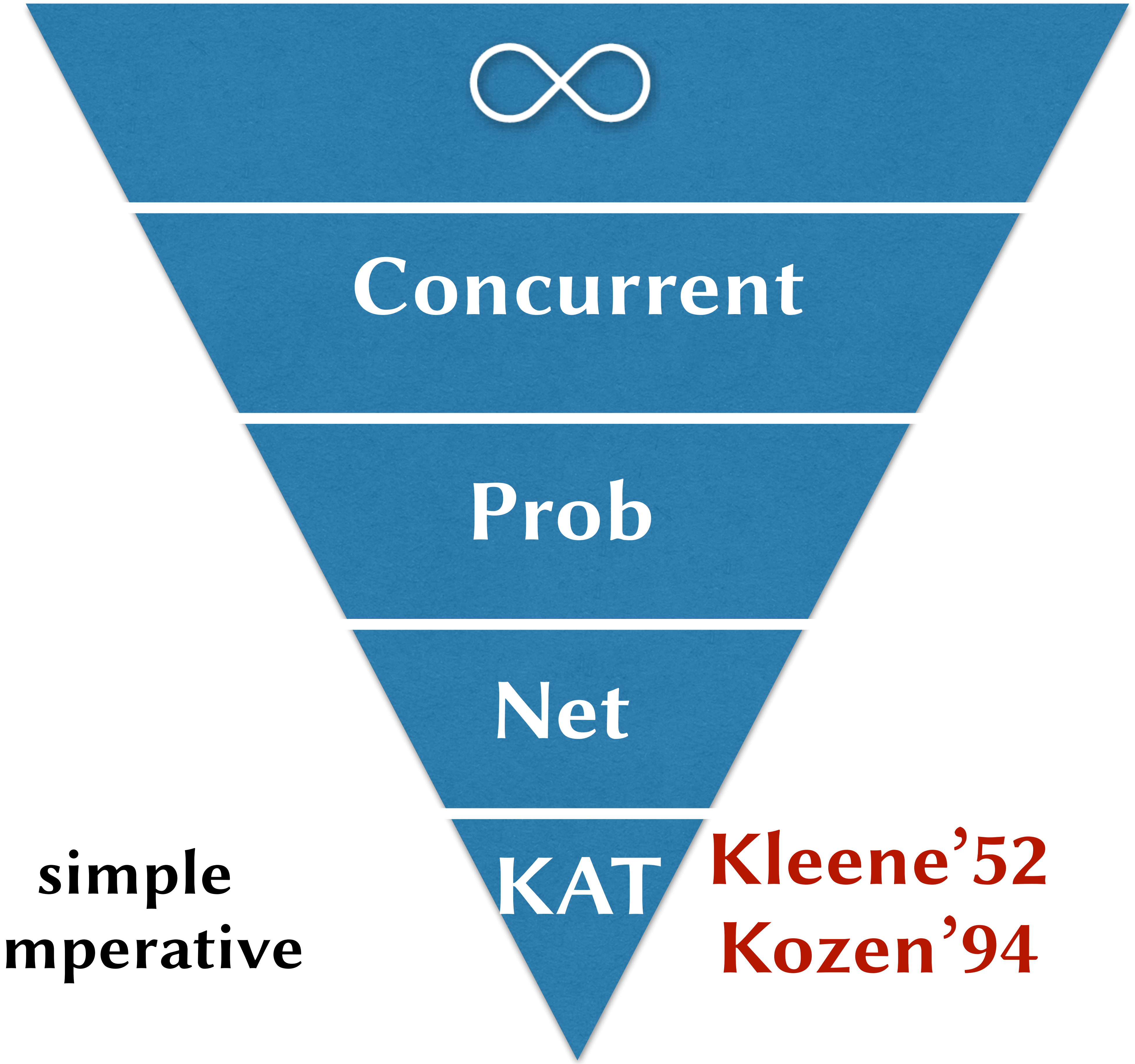
Concurrent

Prob

Net

KAT

The KAT tower principle



∞

Concurrent

Prob

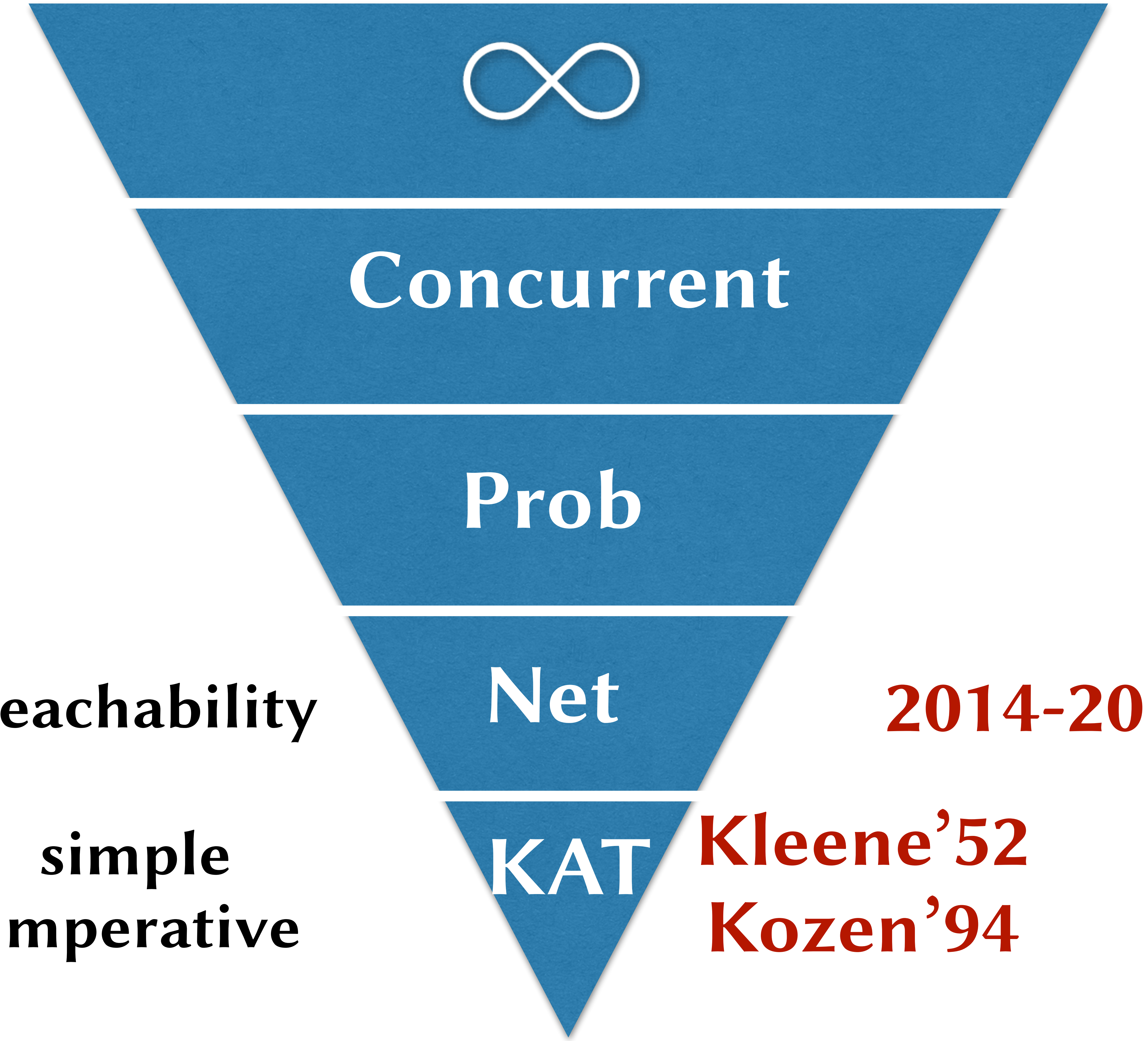
Net

KAT

simple
imperative

Kleene'52
Kozen'94

The KAT tower principle



∞

Concurrent

Prob

Net

KAT

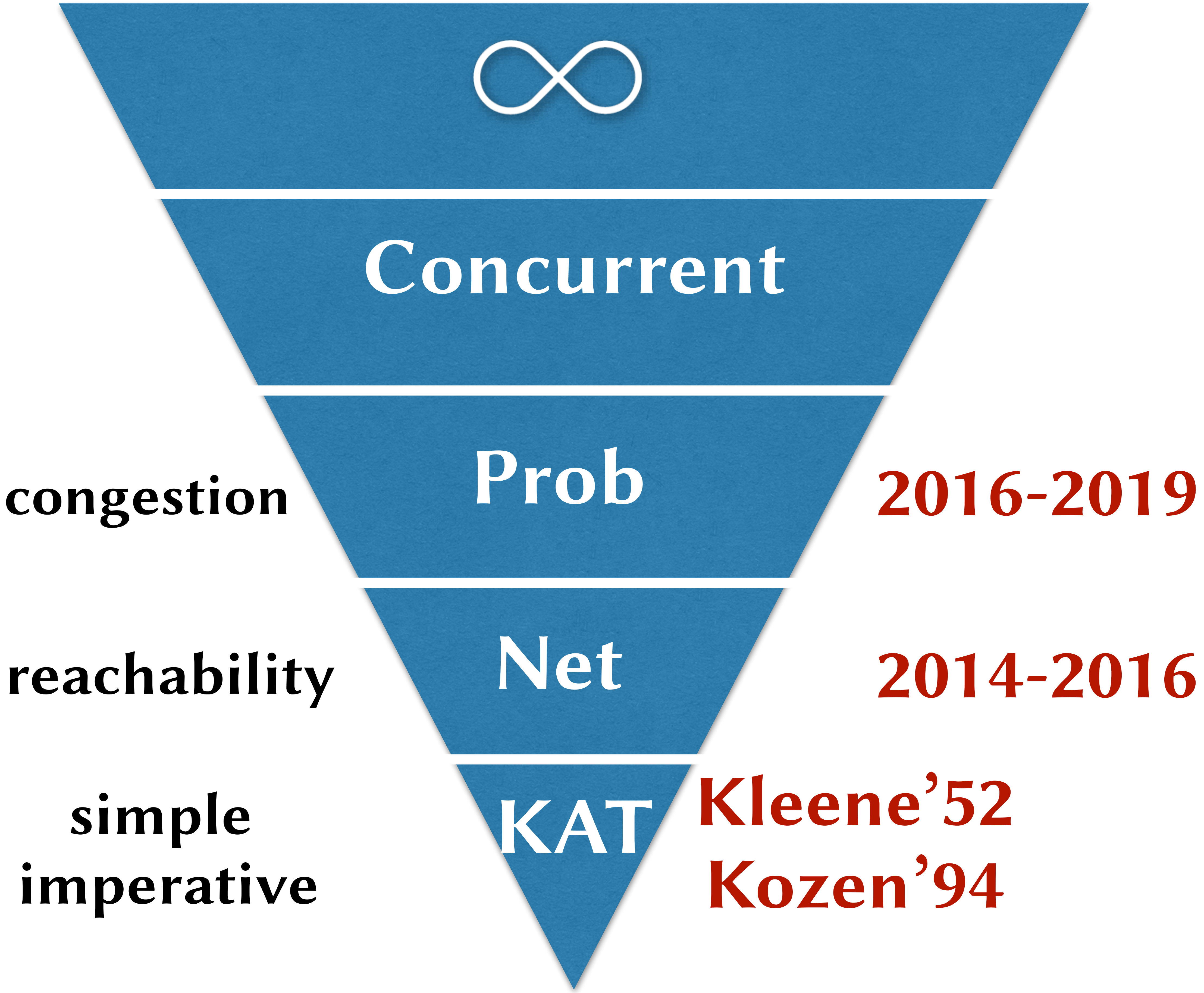
reachability

simple imperative

2014-2016

Kleene'52
Kozen'94

The KAT tower principle



∞

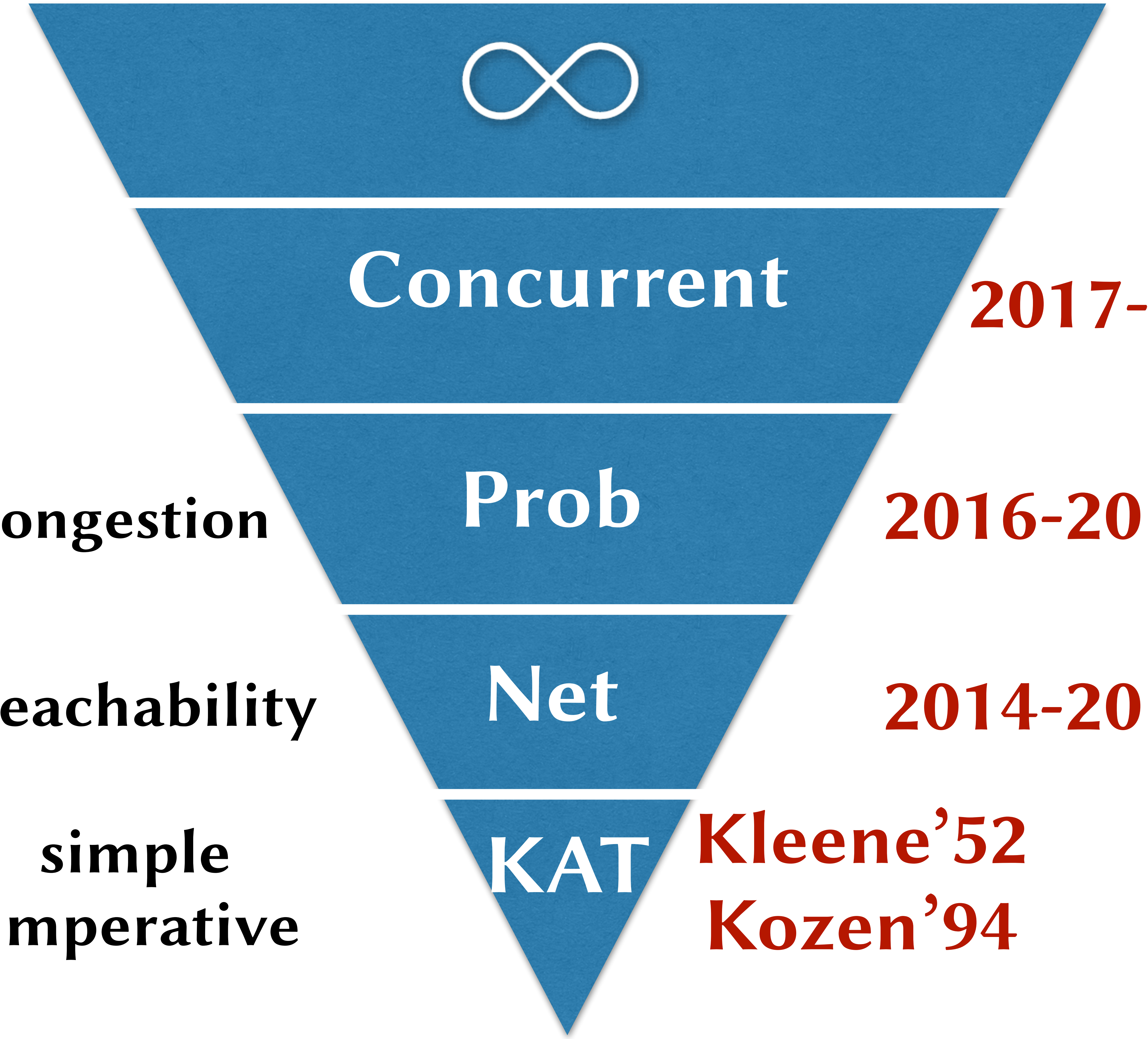
Concurrent

congestion | Prob | 2016-2019

reachability | Net | 2014-2016

simple imperative | KAT | Kleene'52 Kozen'94

The KAT tower principle



∞

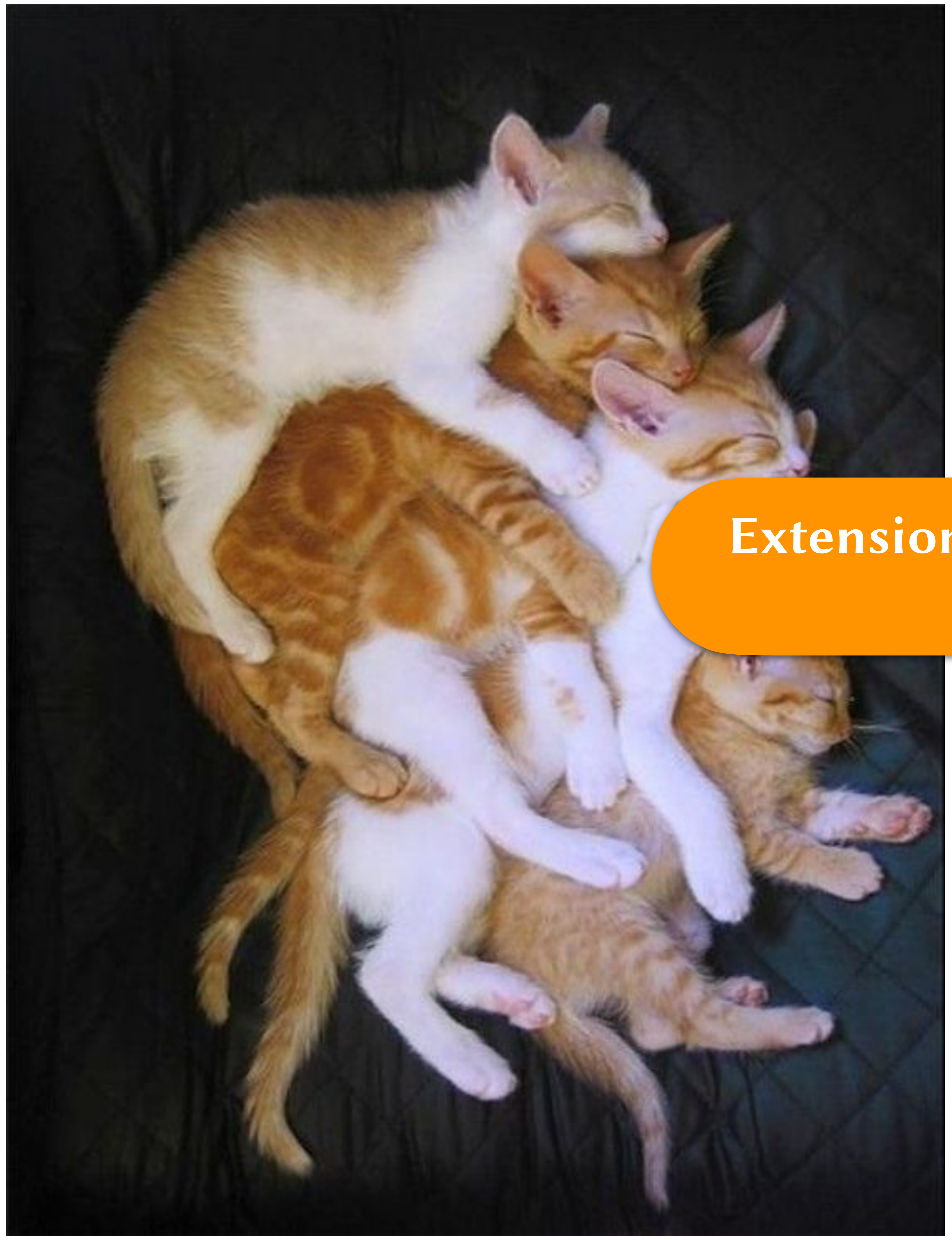
Concurrent 2017-2022

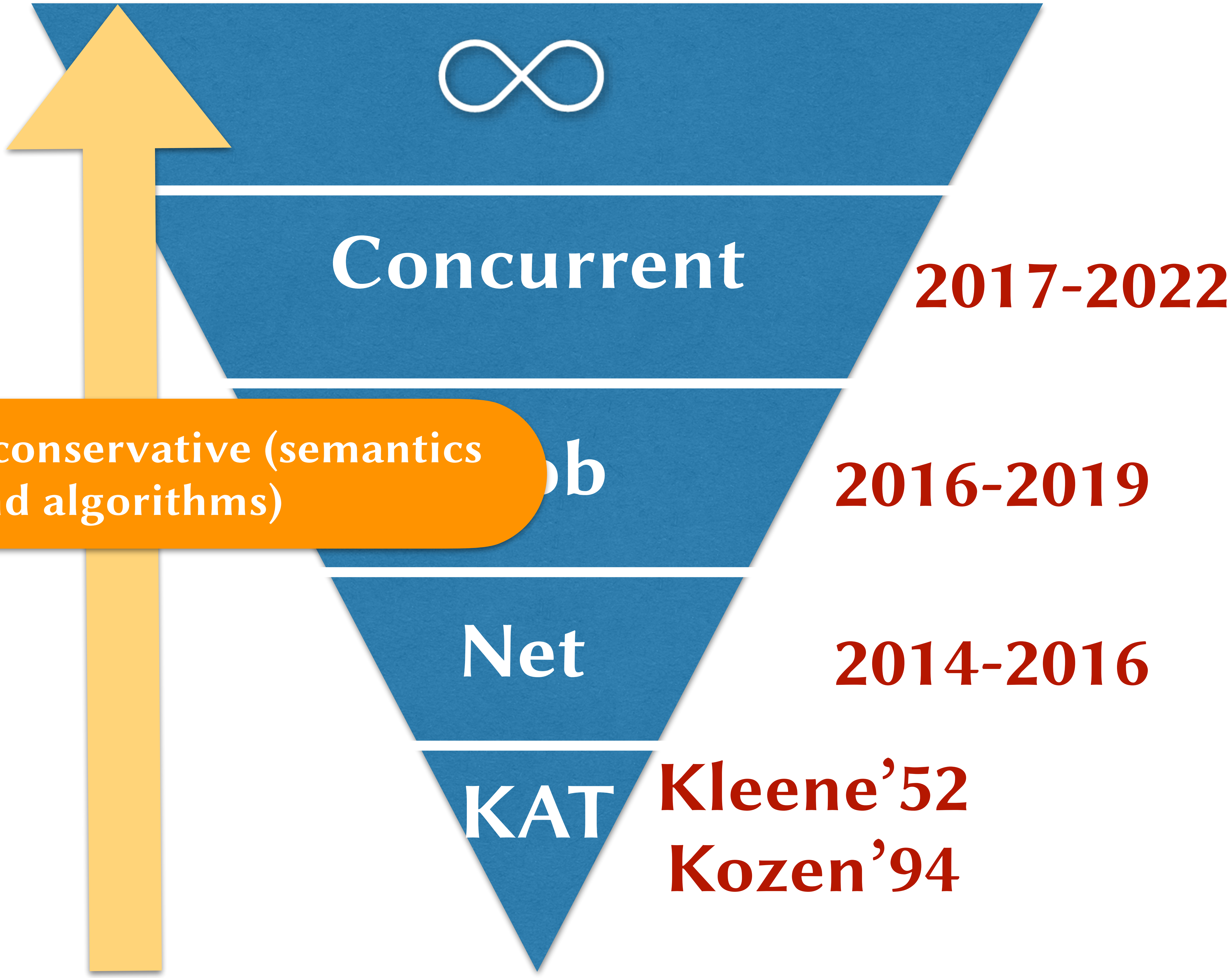
congestion — Prob — 2016-2019

reachability — Net — 2014-2016

simple imperative — KAT — Kleene’52 Kozen’94

The KAT tower principle





∞

Concurrent 2017-2022

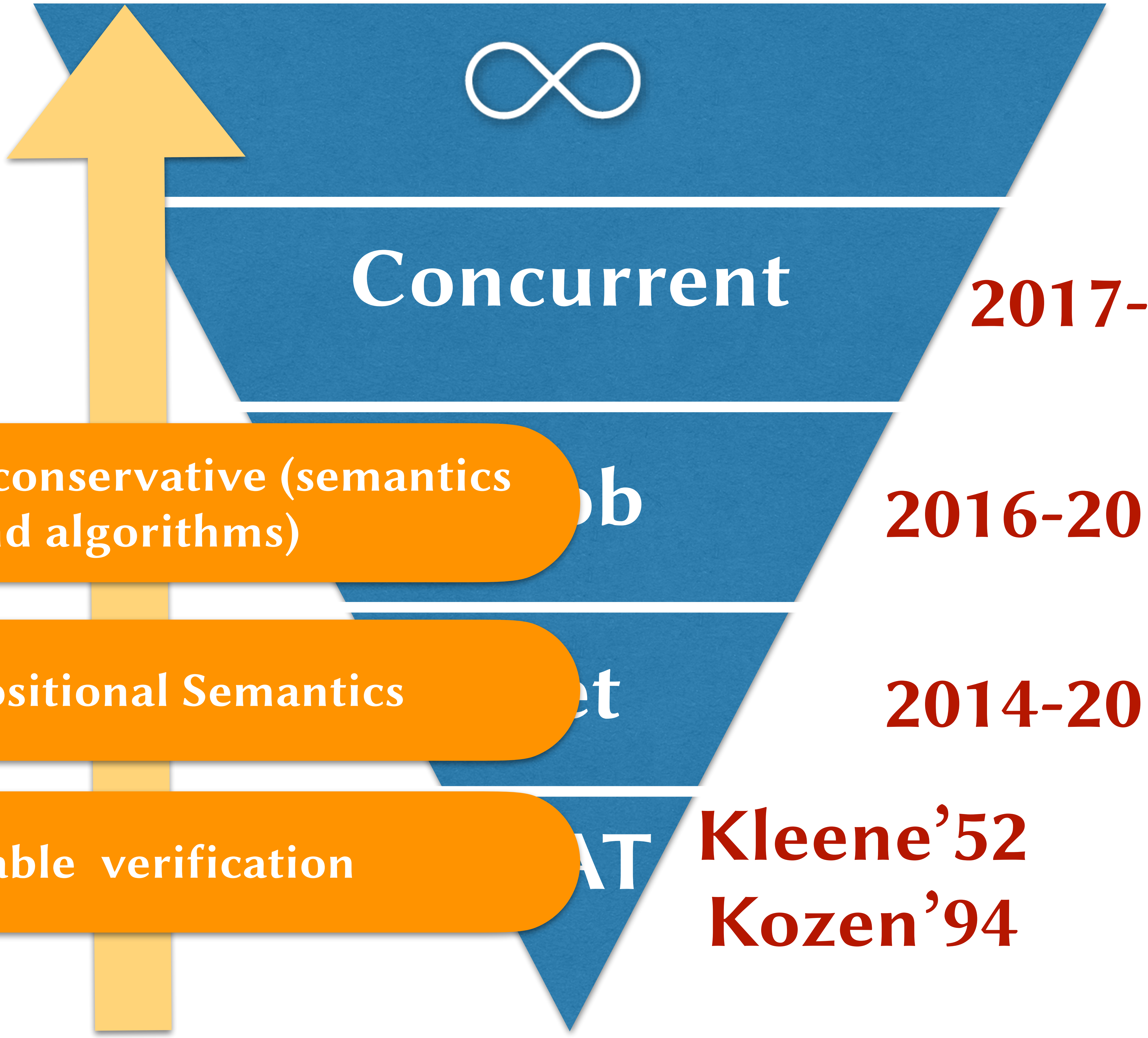
ob 2016-2019

Net 2014-2016

KAT Kleene'52 Kozen'94

Extensions conservative (semantics and algorithms)

The KAT tower principle



∞

Concurrent

2017-2022

Extensions conservative (semantics and algorithms)

b

2016-2019

Compositional Semantics

t

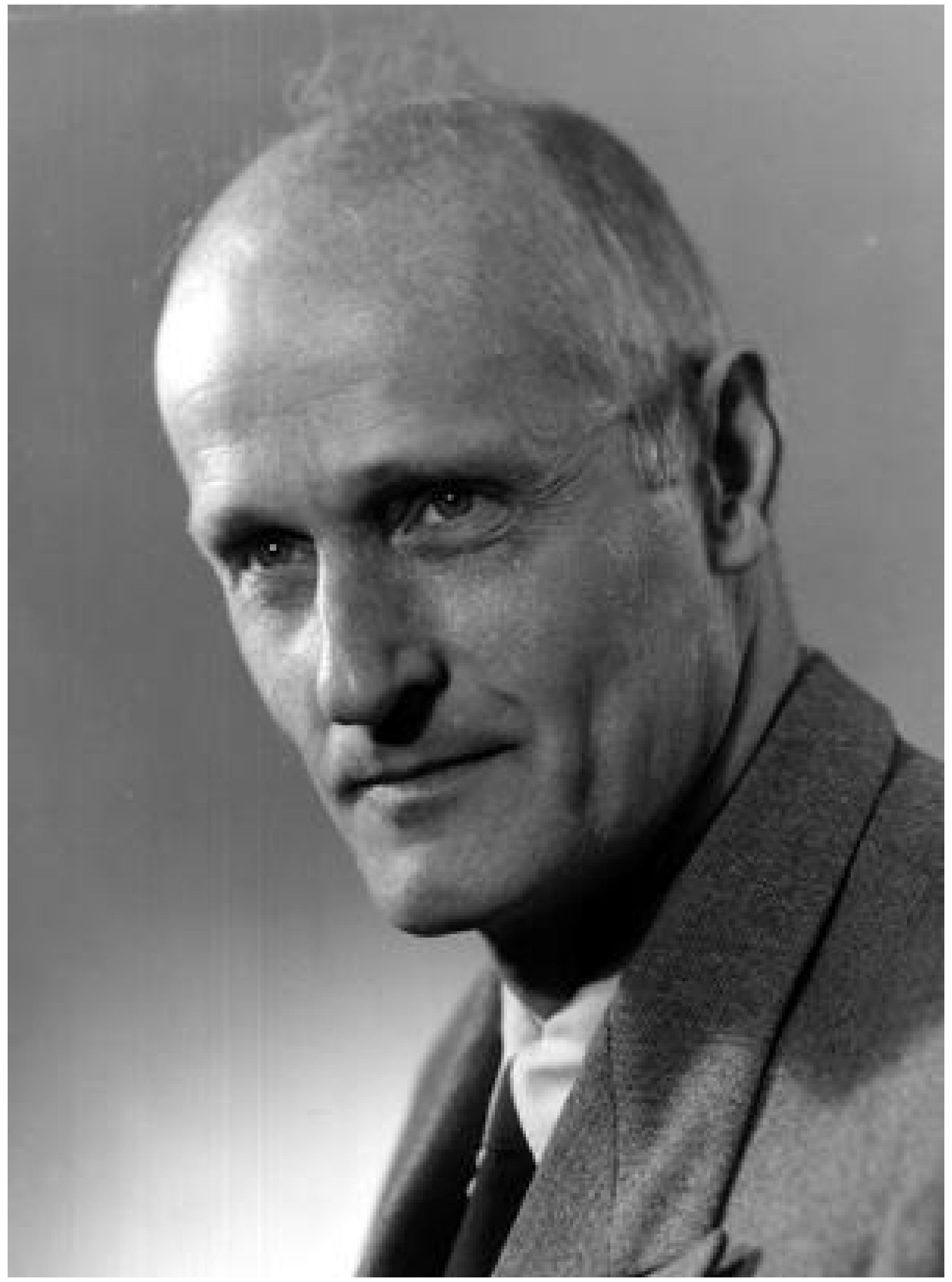
2014-2016

Scalable verification

AT

Kleene'52
Kozen'94

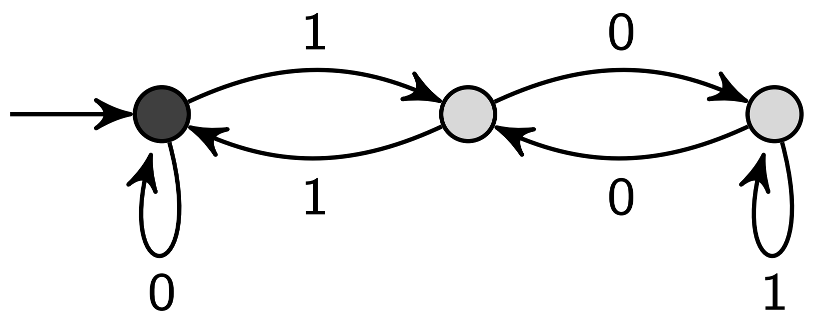
NetKAT



Stephen Cole Kleene
(1909–1994)

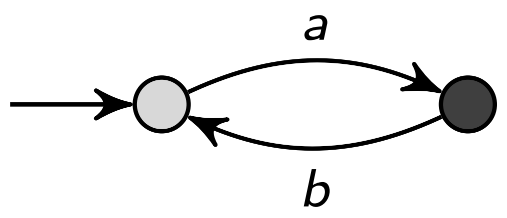
$(0 + 1(01^*0)^*1)^*$

$\{\text{multiples of 3 in binary}\}$



$(ab)^*a = a(ba)^*$

$\{a, aba, ababa, \ldots\}$



$(a + b)^* = a^*(ba^*)^*$

$\{\text{all strings over } \{a, b\}\}$

NetKAT

$(K, B, +, \cdot, ^*, ^-, 0, 1), \quad B \subseteq K$

- $(K, +, \cdot, ^*, 0, 1)$ is a Kleene algebra
- $(B, +, \cdot, ^-, 0, 1)$ is a Boolean algebra
- $(B, +, \cdot, 0, 1)$ is a subalgebra of $(K, +, \cdot, 0, 1)$

- $p, q, r, \ldots$ range over K
- $a, b, c, \ldots$ range over B

NetKAT

$(K, B, +, \cdot, ^*, ^-, 0, 1), \quad B \subseteq K$

- $(K,$
- $(B,$
- $(B,$

- p, q
- $a, b,$

KAT = simple imperative language

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

While b **do** p = (bp)*!b

NetKAT

- a packet π is an assignment of constant values n to fields x
- a packet history is a nonempty sequence of packets
  $\pi_1 :: \pi_2 :: \cdots :: \pi_k$
- the head packet is π_1

NetKAT

- assignments $x \leftarrow n$
  assign constant value n to field x in the head packet
- tests $x = n$
  if value of field x in the head packet is n, then pass, else drop
- dup
  duplicate the head packet

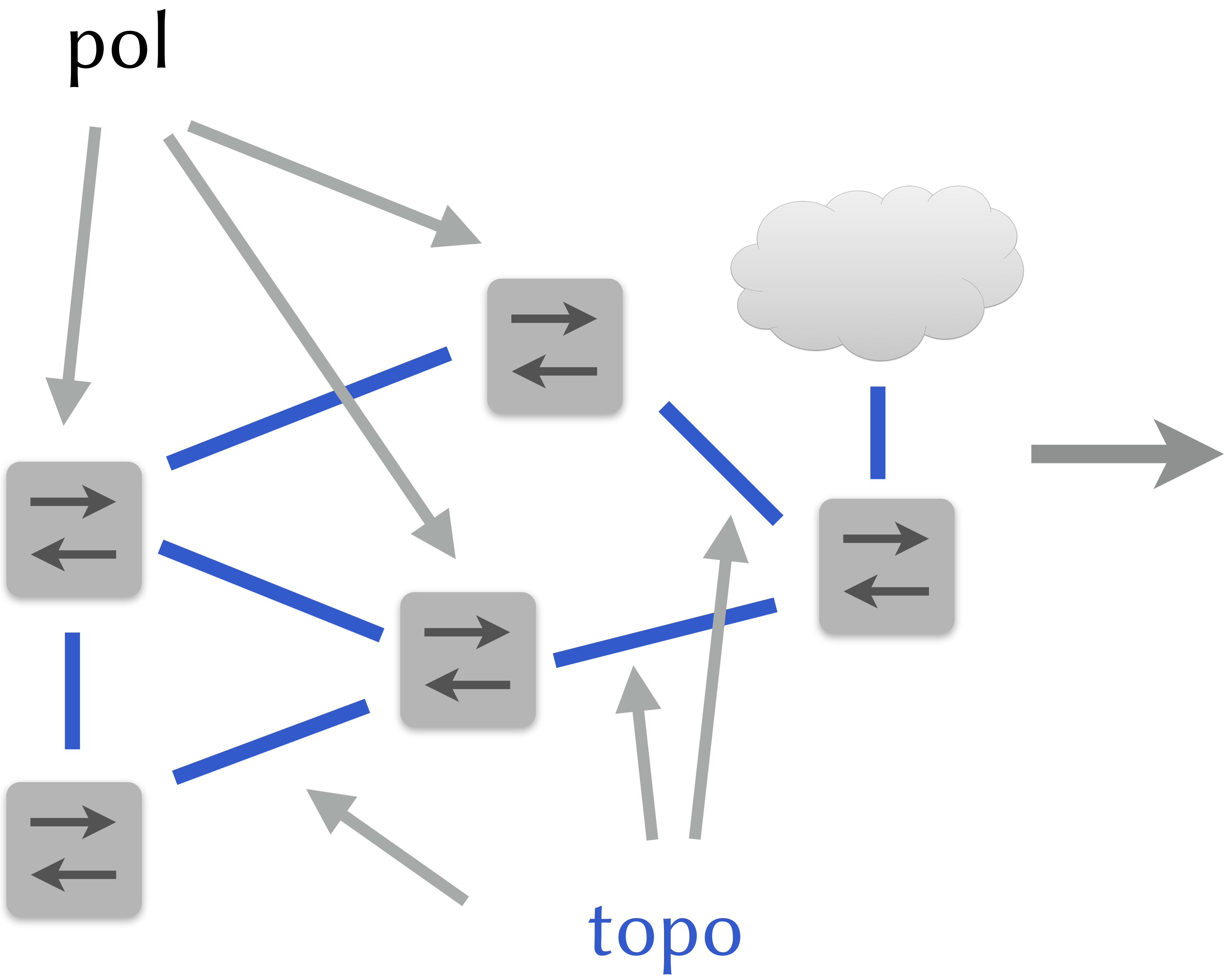
Networks in NetKAT

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

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

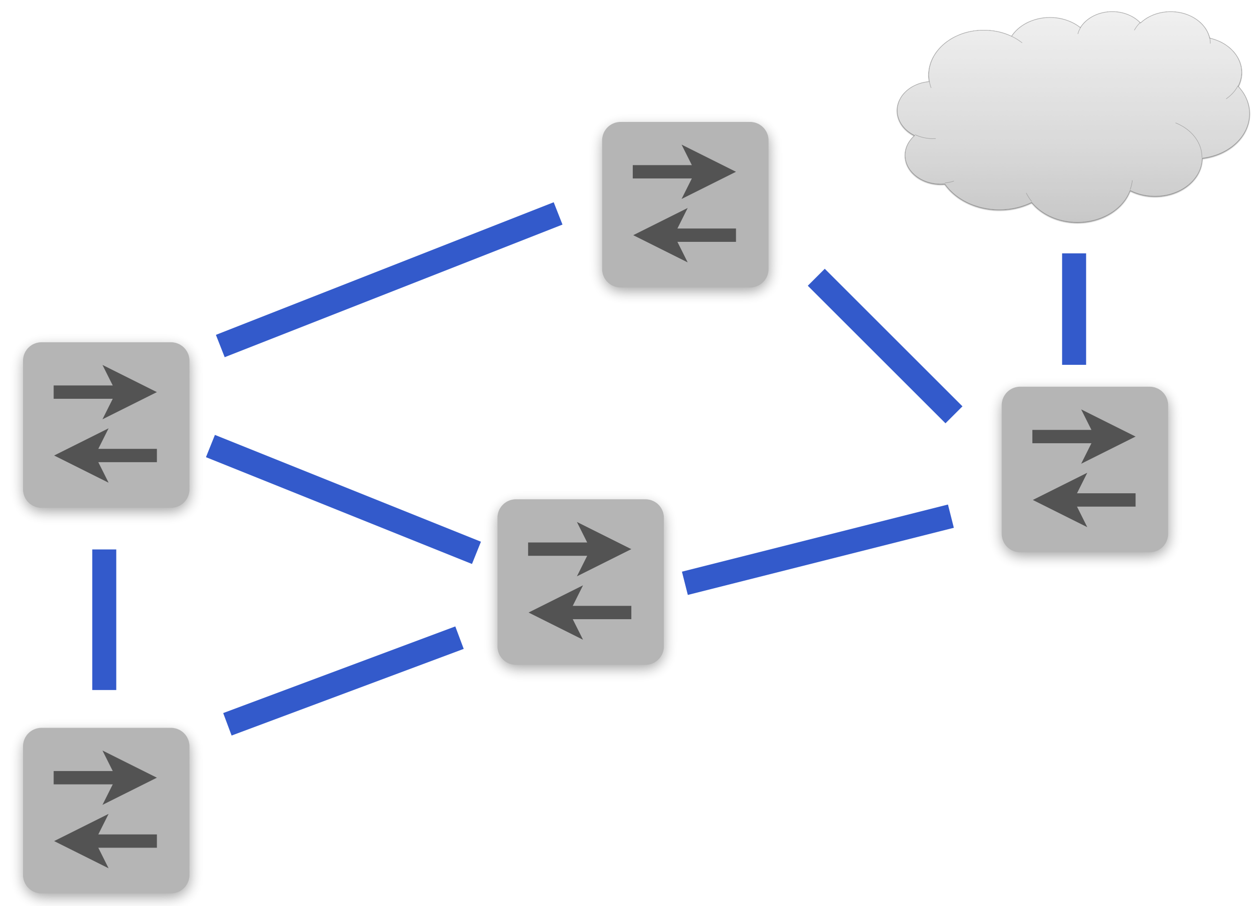
Encoding Networks

...and entire networks can be encoded by iterating the processing done by the switches and topology



$(\mathrm{pol} \bullet \mathrm{topo})^*$

Formal Reasoning



$$\models \varphi$$

Given a network encoded this way, we'd like to be able to automatically answer questions like:

"Does the network isolate A and B?"

Can reduce this question (and others) to equivalence

$$\mathbf{A \bullet (pol \bullet topo)^* \bullet B \equiv false}$$

Verification using NetKAT

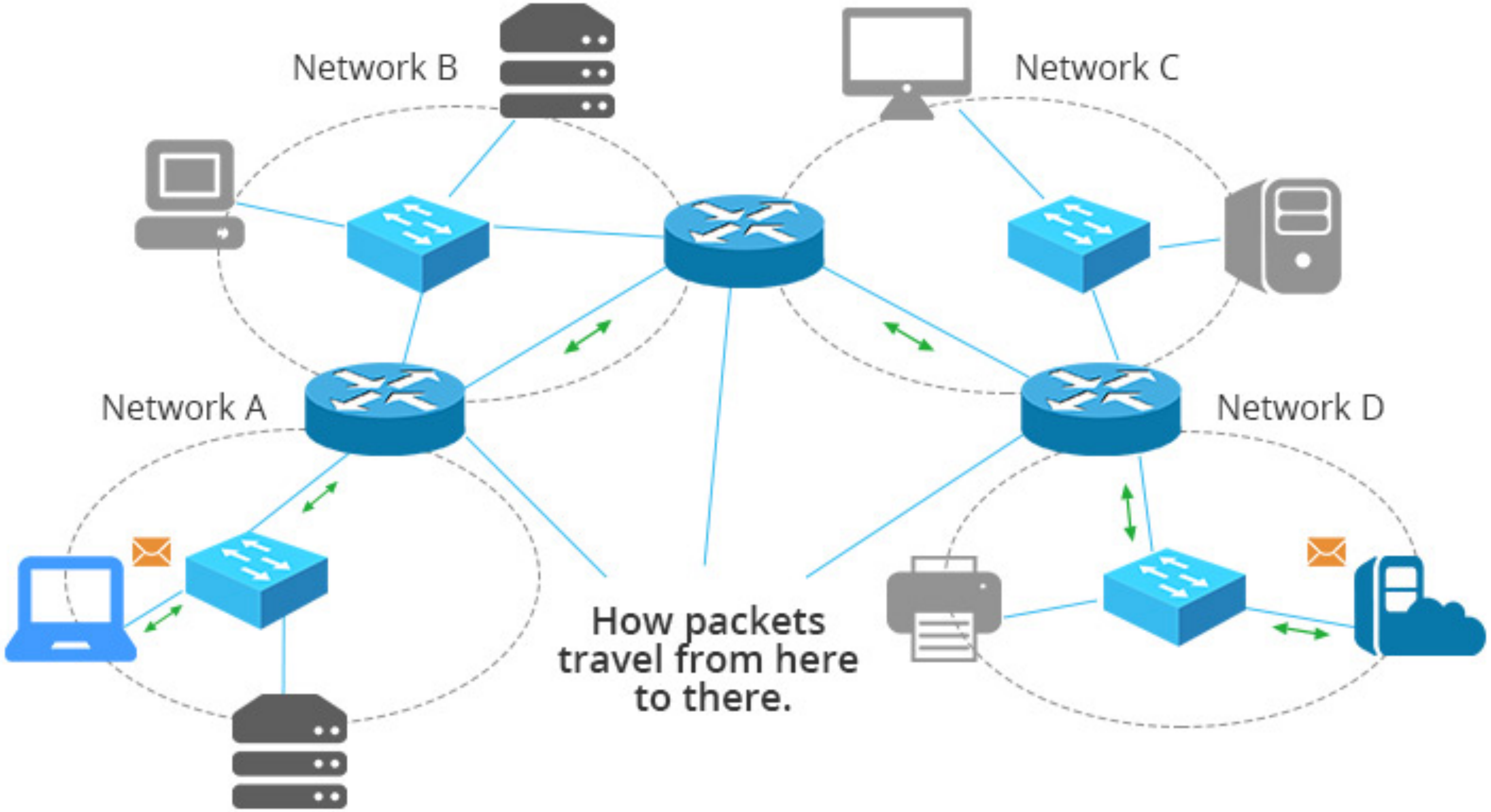
Soundness and Completeness [Anderson et al. 14]

- $\vdash p = q$ if and only if $[\![p]\!] = [\![q]\!]$

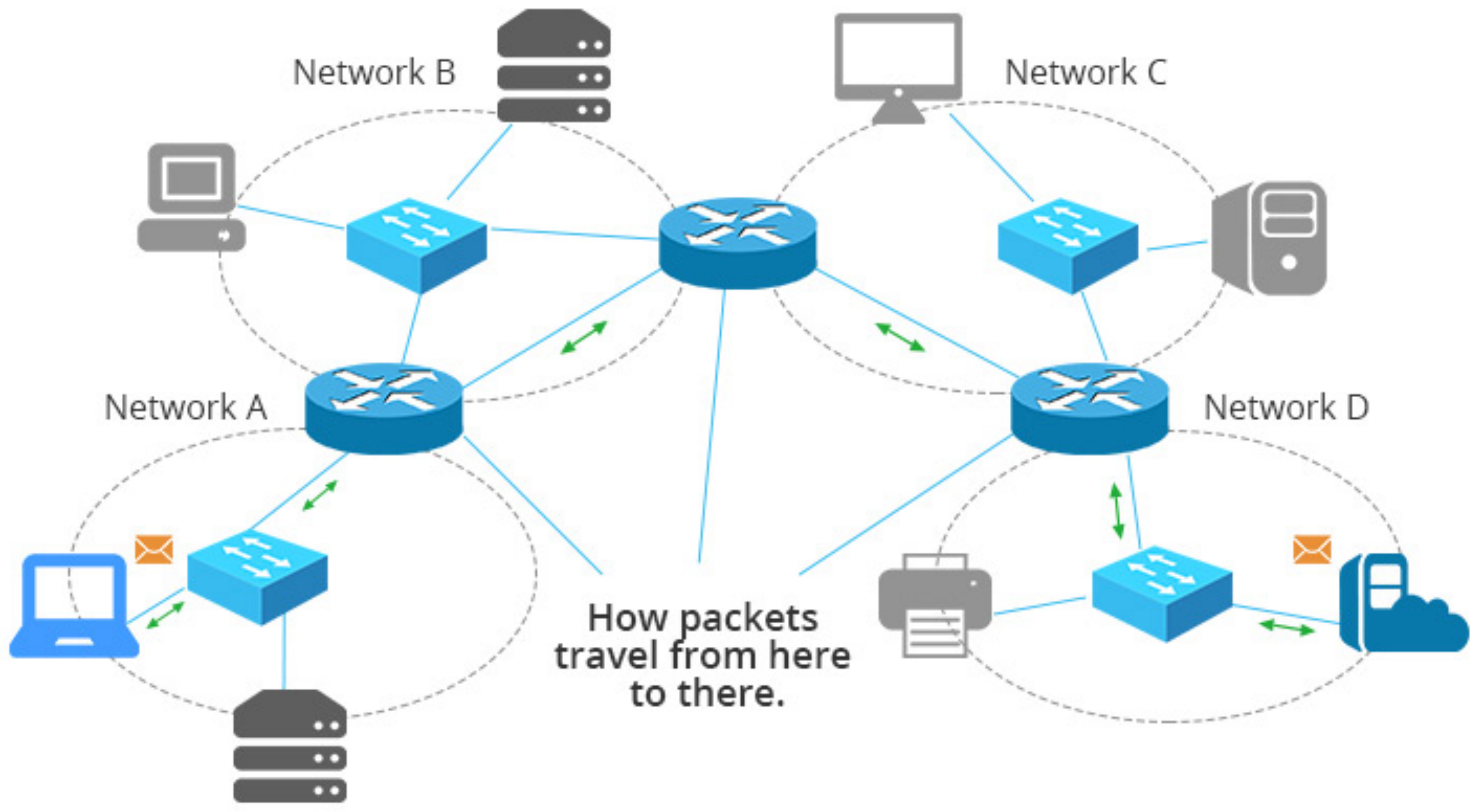
Decision Procedure [Foster et al. 15]

- NetKAT coalgebra
- efficient bisimulation-based decision procedure
- implementation in OCaml
- deployed in the Frenetic suite of network management tools

Verification using NetKAT



Verification using NetKAT



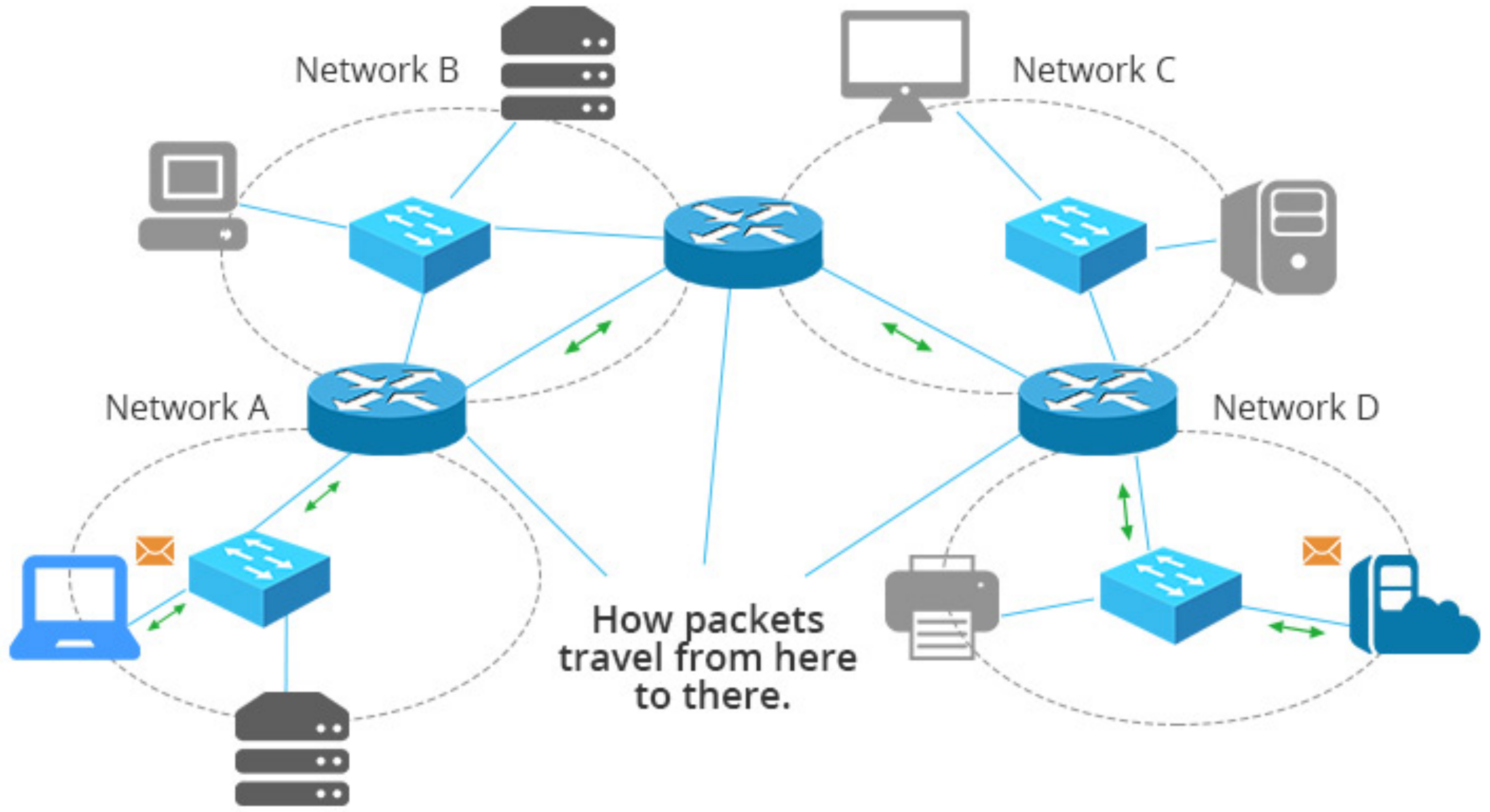
Verification using NetKAT

Switch forwarding tables

Pattern	Actions
dstport=22	Drop
srcip=10.0.0.1	Forward 1
*	Forward 2

encoding

```
if dstport=22 then false
else if srcip=10.0.0.1 then port := 1
else port := 2
```



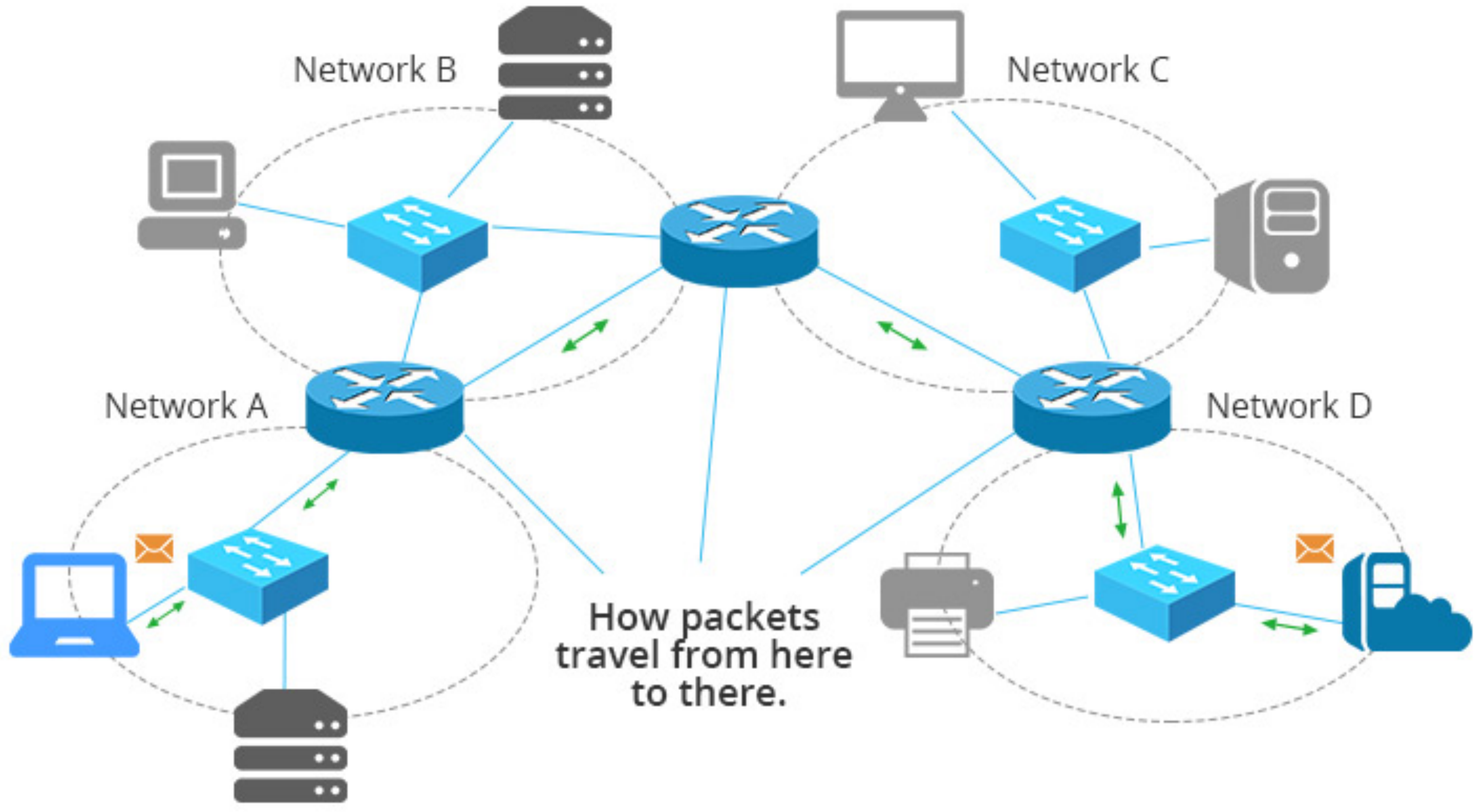
Verification using NetKAT

Switch forwarding tables

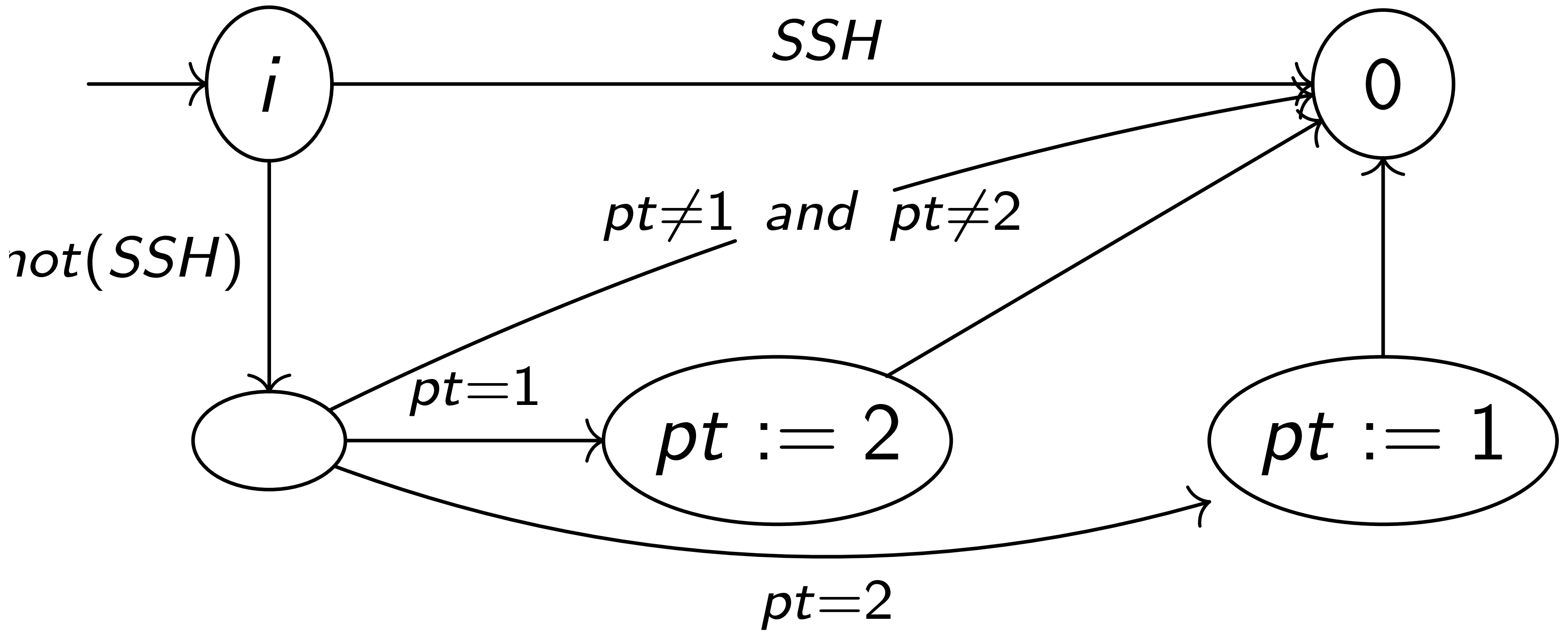
Pattern	Actions
`dstport=22`	`Drop`
`srcip=10.0.0.1`	`Forward 1`
`*`	`Forward 2`

↓ encoding

if dstport=22 **then false**
else if srcip=10.0.0.1 **then** port **:=** 1
else port **:=** 2



Operational Semantics: Automata



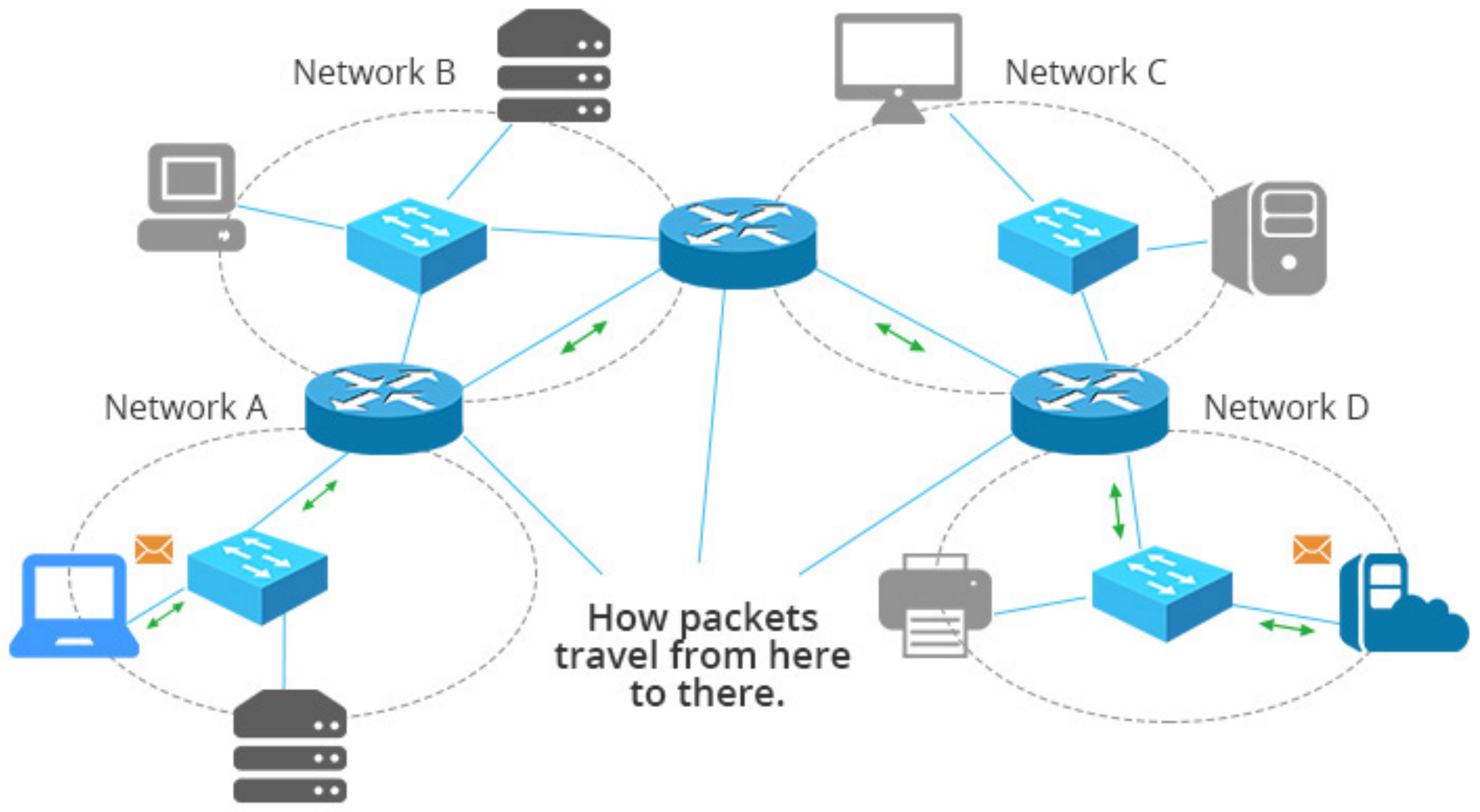
Verification using NetKAT

Switch forwarding tables

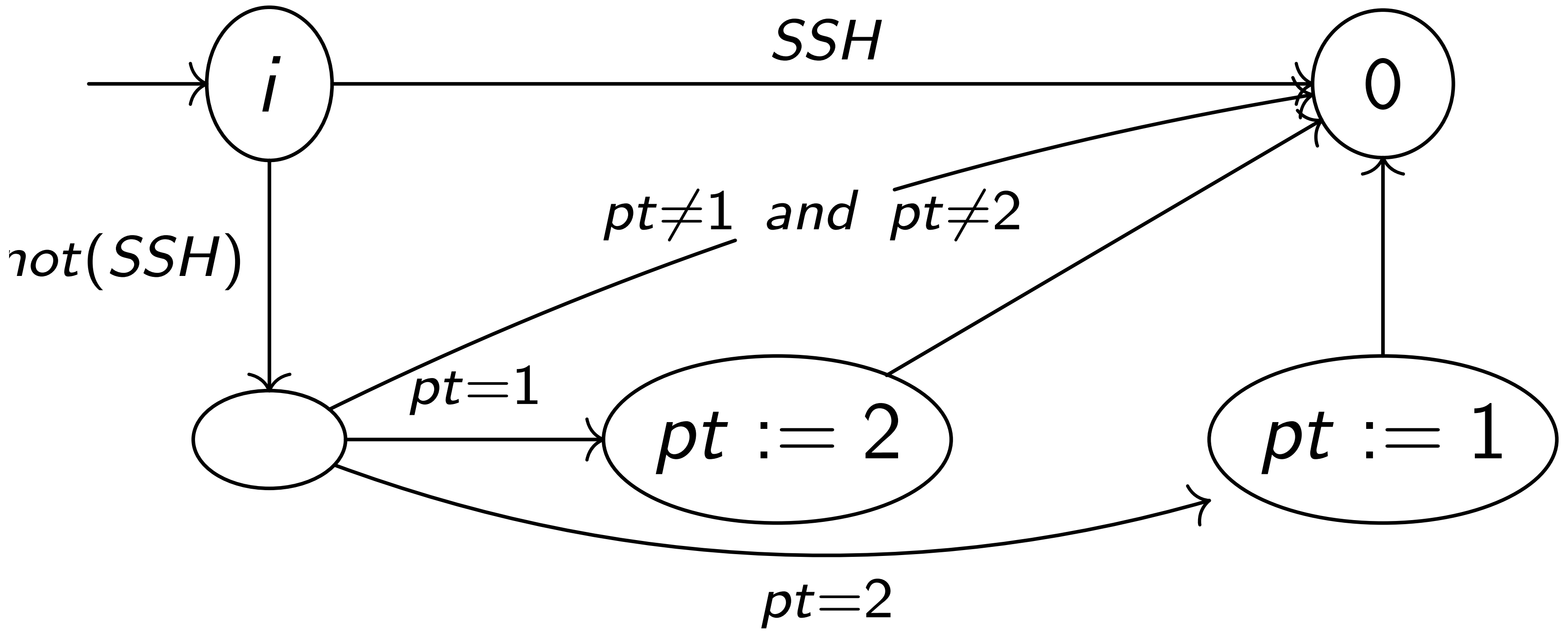
Pattern	Actions
dstport=22	Drop
srcip=10.0.0.1	Forward 1
*	Forward 2

↓ encoding

```
if dstport=22 then false
else if srcip=10.0.0.1 then port := 1
else port := 2
```



Operational Semantics: Automata



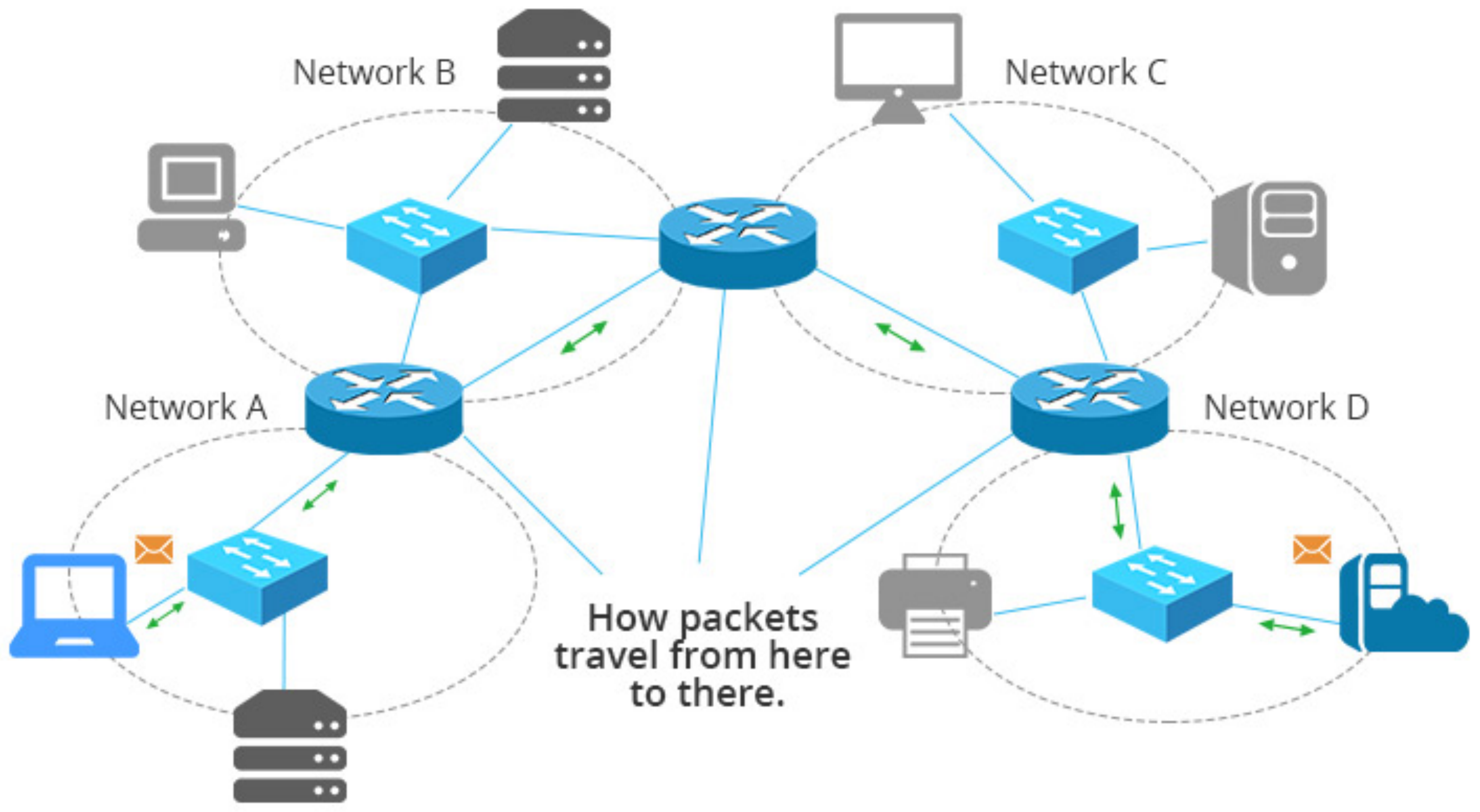
Verification using NetKAT

Switch forwarding tables

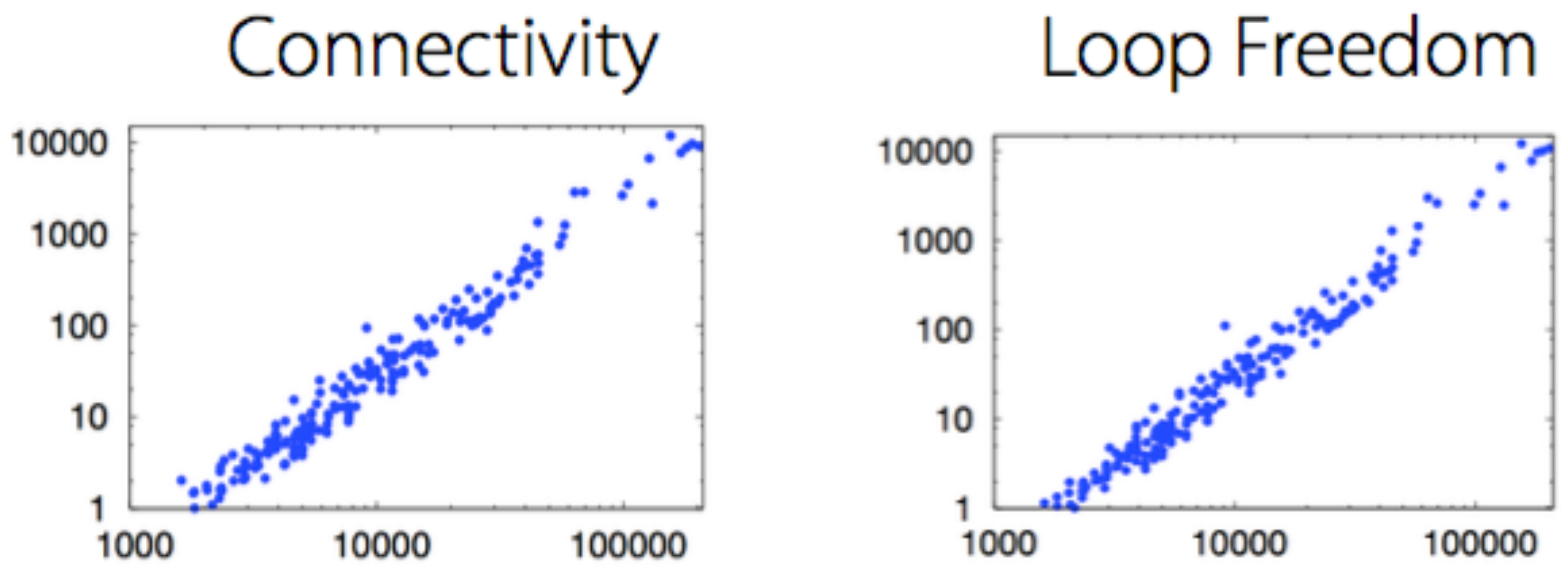
Pattern	Actions
`dstport=22`	`Drop`
`srcip=10.0.0.1`	`Forward 1`
`*`	`Forward 2`

encoding

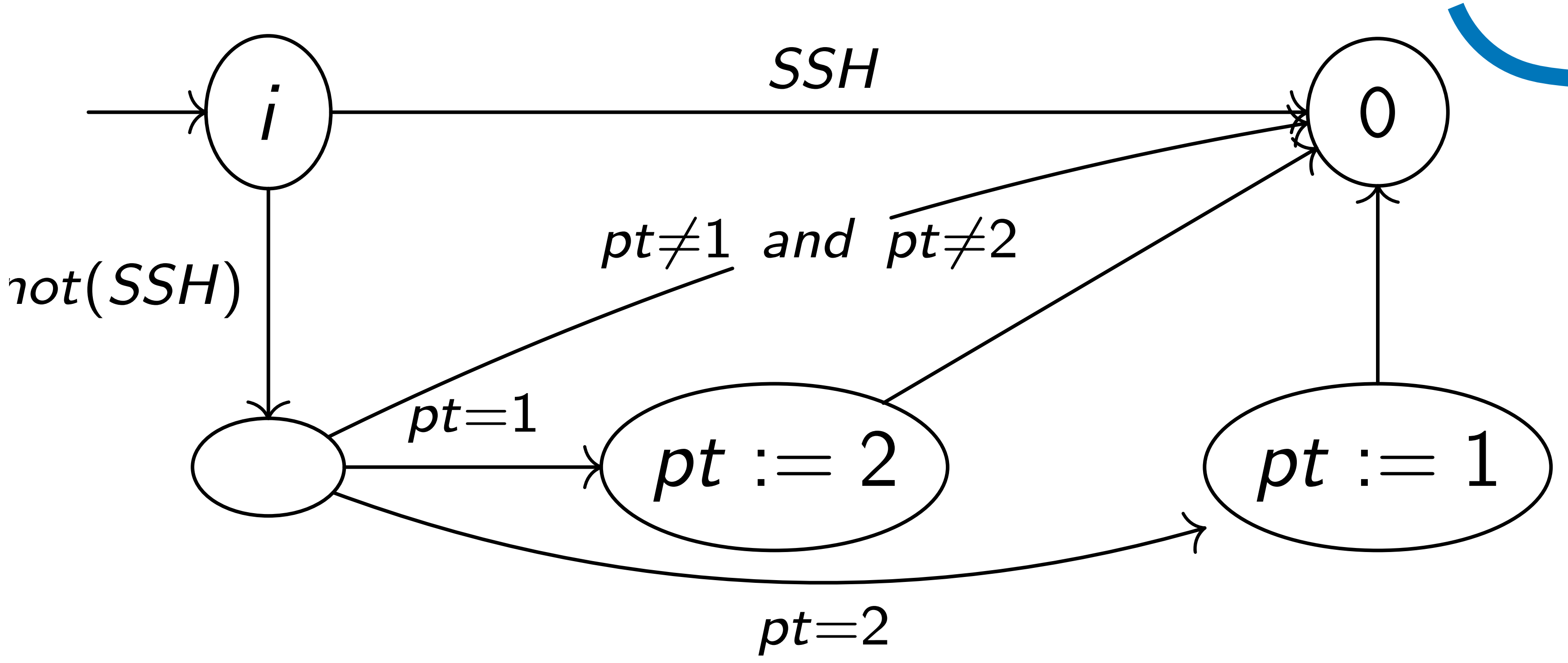
```
if dstport=22 then false
else if srcip=10.0.0.1 then port := 1
else port := 2
```



[POPL '15]

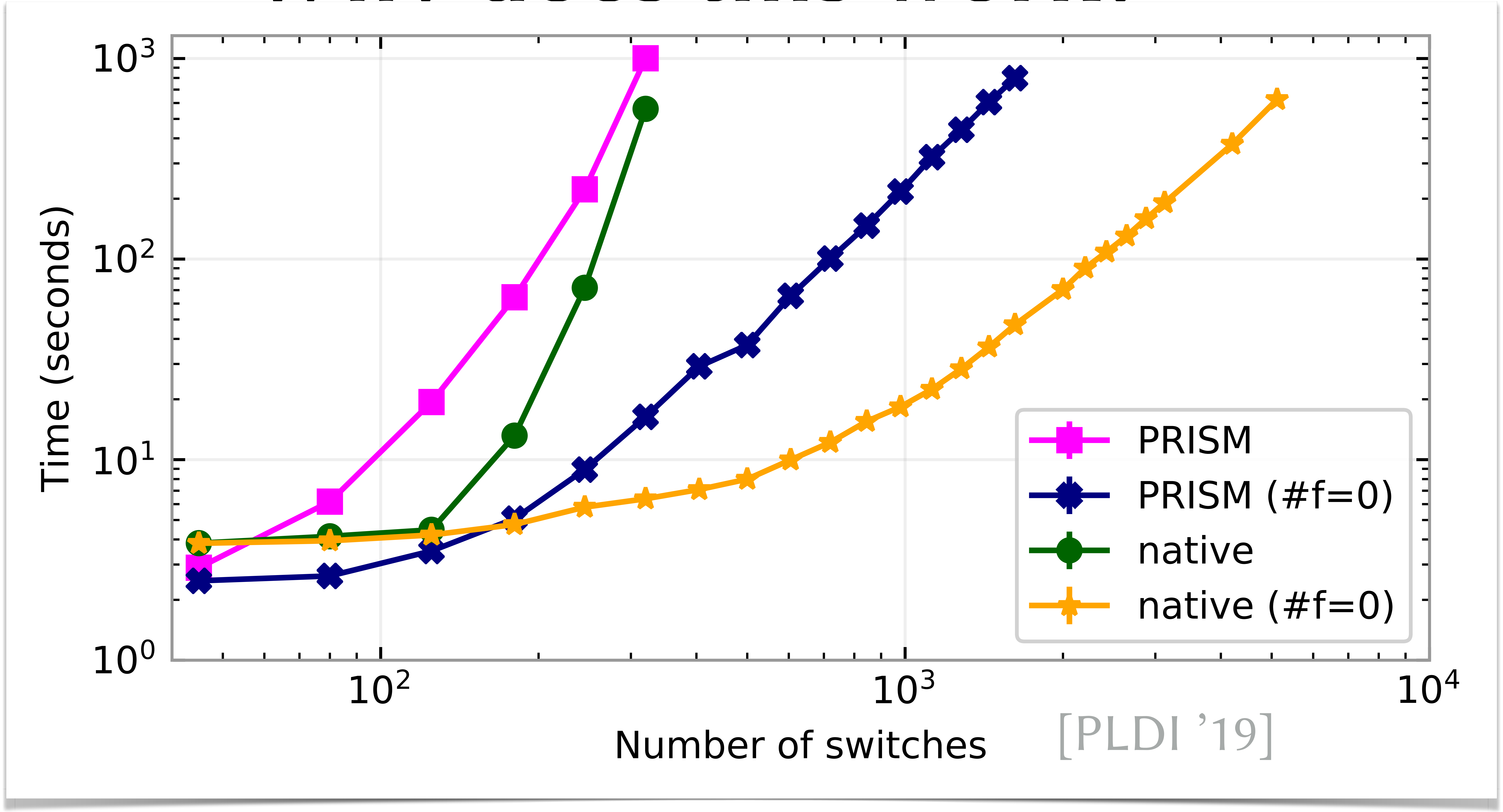


Operational Semantics: Automata



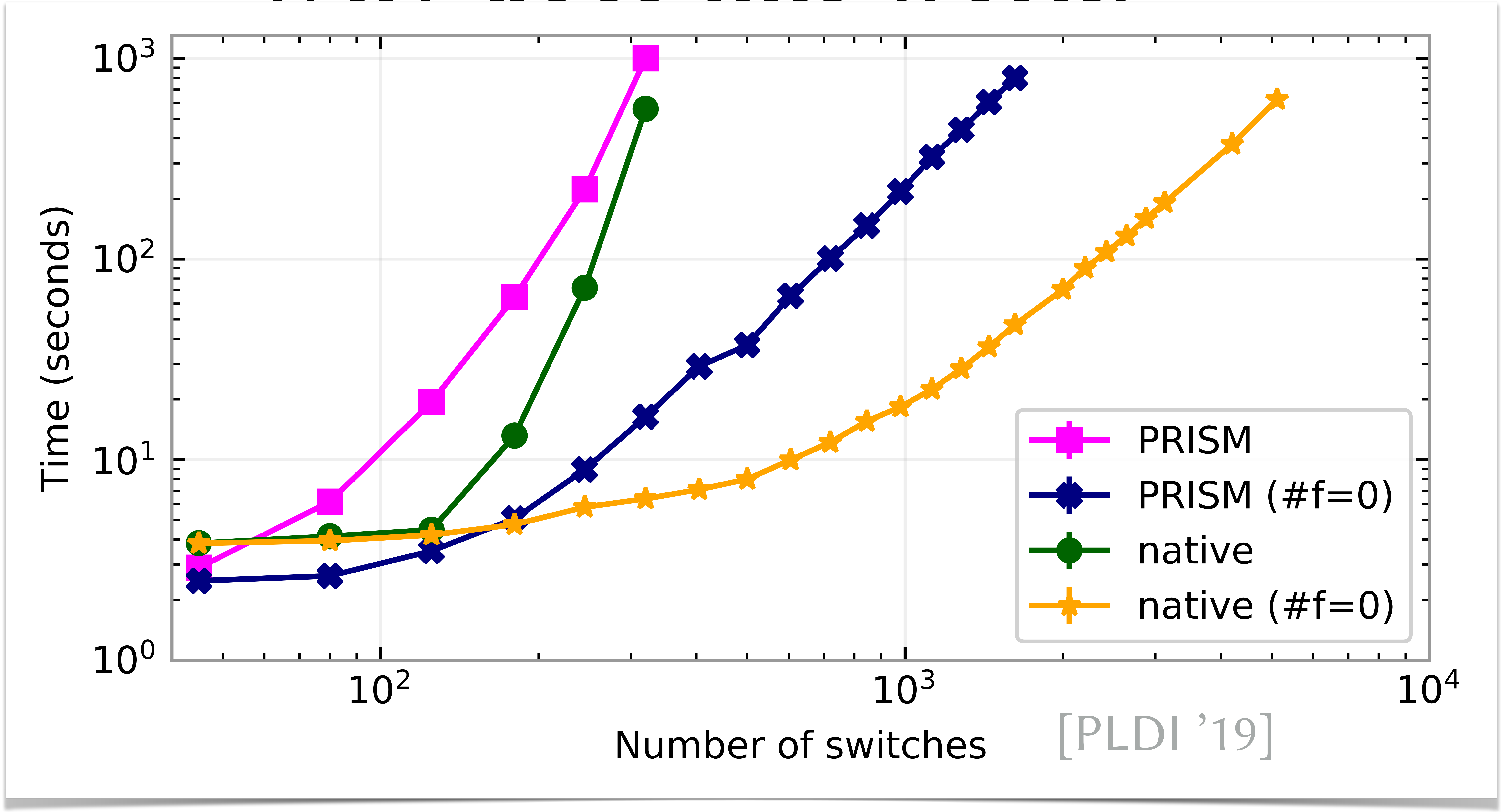
Prototype implementation
Topology Zoo benchmarks

Why does this work?



[PLDI '19]

Why does this work?



[PLDI ’19]

Theorem [POPL ’14]: Deciding equivalence is PSPACE-complete

> “I can’t tell PSPACE from outer space”
>
> —A prominent academic

This Talk

Guarded KAT: a restriction that is reasonably expressive *and* efficient

This Talk

Guarded KAT: a restriction that is reasonably expressive *and* efficient

Key Results:

- Decidable equivalence in (near) linear time
- Sound and complete axiomatization
- Automata model and Kleene Theorem

This Talk

Guarded KAT: a restriction that is reasonably expressive *and* efficient

This Talk

Guarded KAT: a restriction that is reasonably expressive *and* efficient

Switch forwarding tables

Pattern	Actions
`dstport=22`	`Drop`
`srcip=10.0.0.1`	`Forward 1`
`*`	`Forward 2`

encoding

```
if dstport=22 then false
else if srcip=10.0.0.1 then port := 1
else port := 2
```

This Talk

Guarded KAT: a restriction that is reasonably expressive *and* efficient

Switch forwarding tables

Pattern	Actions
`dstport=22`	`Drop`
`srcip=10.0.0.1`	`Forward 1`
`*`	`Forward 2`

encoding

```
if dstport=22 then false
else if srcip=10.0.0.1 then port := 1
else port := 2
```

Guarded choice

This Talk

Guarded KAT: a restriction that is reasonably expressive *and* efficient

Switch forwarding tables

Pattern	Actions
`dstport=22`	`Drop`
`srcip=10.0.0.1`	`Forward 1`
`*`	`Forward 2`

encoding

```
if dstport=22 then false
else if srcip=10.0.0.1 then port := 1
else port := 2
```

Guarded choice

$$\mathrm{A} \bullet (\mathrm{pol} \bullet \underline{\mathrm{topo})^{*}} \bullet \mathrm{B} \equiv$$

Guarded iteration

GKAT Overview

GKAT Syntax

GKAT Syntax

Parameters

GKAT Syntax

Parameters

- finite set of actions p, q, r ∈ Action

GKAT Syntax

Parameters

- finite set of actions $p, q, r \in \text{Action}$
- finite set of primitive tests $t \in \text{Test}$

GKAT Syntax

Parameters

- finite set of actions p, q, r ∈ Action
- finite set of primitive tests t ∈ Test

Syntax

GKAT Syntax

Parameters

- finite set of actions p, q, r ∈ Action
- finite set of primitive tests t ∈ Test

Syntax

$b, c, d \in \mathrm{BExp} ::= \ 0 \mid 1 \mid t \in \mathrm{Test} \mid b \cdot c \mid b + c \mid \neg b$

GKAT Syntax

Parameters

- finite set of actions p, q, r ∈ Action
- finite set of primitive tests t ∈ Test

Syntax

Boolean algebra

b, c, d ∈ BExp ::= 0 | 1 | t ∈ Test | b·c | b+c | ¬b

GKAT Syntax

Parameters

- finite set of actions p, q, r ∈ Action
- finite set of primitive tests t ∈ Test

Syntax

Boolean algebra

b, c, d ∈ BExp ::= 0 | 1 | t ∈ Test | b·c | b+c | ¬b

e, f, g ∈ Exp ::=

GKAT Syntax

Parameters

- finite set of actions p, q, r ∈ Action
- finite set of primitive tests t ∈ Test

Syntax

Boolean algebra

b, c, d ∈ BExp ::= 0 | 1 | t ∈ Test | b·c | b+c | ¬b

e, f, g ∈ Exp ::=
| b ∈ BExp **assert** b
| p ∈ Action **do** p

GKAT Syntax

Parameters

- finite set of actions $p, q, r \in \mathrm{Action}$
- finite set of primitive tests $t \in \mathrm{Test}$

Syntax

Boolean algebra

$$b, c, d \in \mathrm{BExp} ::= \; 0 \mid 1 \mid t \in \mathrm{Test} \mid b \cdot c \mid b + c \mid \neg b$$

$$e, f, g \in \mathrm{Exp} ::=$$

$\mid \; b \in \mathrm{BExp}$	**assert** b
$\mid \; p \in \mathrm{Action}$	**do** p
$\mid \; e \cdot f$	e **;** f
$\mid \; e +_b f$	**if** b **then** e **else** f
$\mid \; e^{b}$	**while** b **do** e

GKAT Syntax

assert b
do p
e;f
if b then e else f
while b do e

Semantics
+
Program equivalence ?

GKAT Syntax

assert b
do p
e;f
if b then e else f
while b do e

$e \cdot 0 \equiv 0 \equiv 0 \cdot e$

$e \cdot 1 \equiv e \equiv 1 \cdot e$

$(e \cdot f) \cdot g \equiv e \cdot (f \cdot g)$

…

Semantics
\+
Program equivalence

?

Relational Semantics

Relational Semantics

Parameters: interpretation i = (State, eval, sat)

Relational Semantics

Parameters: interpretation i = (State, eval, sat)

- set of states σ ∈ State

Relational Semantics

Parameters: interpretation i = (State, eval, sat)

- set of states σ ∈ State
- for each action p: relation eval(p) ⊆ State × State

Relational Semantics

Parameters: interpretation i = (State, eval, sat)

- set of states σ ∈ State
- for each action p: relation eval(p) ⊆ State × State
- for each primitive test t: subset of states sat(t) ⊆ State

Relational Semantics

Parameters: interpretation i = (State, eval, sat)

- set of states σ ∈ State
- for each action p: relation eval(p) ⊆ State × State
- for each primitive test t: subset of states sat(t) ⊆ State

Semantics $B_i[\![e]\!] \subseteq$ State × State

Relational Semantics

Parameters: interpretation i = (State, eval, sat)

- set of states σ ∈ State
- for each action p: relation eval(p) ⊆ State × State
- for each primitive test t: subset of states sat(t) ⊆ State

Semantics $B_i[\![e]\!] \subseteq \text{State} \times \text{State}$

e	$B_i[\![e]\!]$
b	sat(b)

Relational Semantics

Parameters: interpretation i = (State, eval, sat)

- set of states σ ∈ State
- for each action p: relation eval(p) ⊆ State × State
- for each primitive test t: subset of states sat(t) ⊆ State

Semantics $B_i[\![e]\!] \subseteq \text{State} \times \text{State}$

e	$B_i[\![e]\!]$
b	sat(b)
p	eval(p)

Relational Semantics

Parameters: interpretation $i = (\text{State}, \text{eval}, \text{sat})$

- set of states $\sigma \in \text{State}$
- for each action p: relation $\text{eval}(p) \subseteq \text{State} \times \text{State}$
- for each primitive test t: subset of states $\text{sat}(t) \subseteq \text{State}$

Semantics $B_i[\![e]\!] \subseteq \text{State} \times \text{State}$

e	$B_i[\![e]\!]$
b	sat(b)
p	eval(p)
$e +_b f$	$\text{sat}(b) \circ B_i[\![e]\!] \cup \text{sat}(!b) \circ B_i[\![f]\!]$

Relational Semantics

Parameters: interpretation i = (State, eval, sat)

- set of states $\sigma \in \text{State}$
- for each action p: relation $\text{eval}(p) \subseteq \text{State} \times \text{State}$
- for each primitive test t: subset of states $\text{sat}(t) \subseteq \text{State}$

Semantics $B_i[\![e]\!] \subseteq \text{State} \times \text{State}$

e	$B_i[\![e]\!]$
b	$\text{sat}(b)$
p	$\text{eval}(p)$
$e +_b f$	$\text{sat}(b) \circ B_i[\![e]\!] \cup \text{sat}(!b) \circ B_i[\![f]\!]$
$e \cdot f$	$B_i[\![e]\!] \circ B_i[\![f]\!]$

Relational Semantics

Parameters: interpretation i = (State, eval, sat)

- set of states $\sigma \in$ State
- for each action p: relation eval(p) $\subseteq$ State × State
- for each primitive test t: subset of states sat(t) $\subseteq$ State

Semantics $B_i[\![e]\!] \subseteq$ State × State

e	$B_i[\![e]\!]$
b	sat(b)
p	eval(p)
$e +_b f$	$\text{sat}(b) \circ B_i[\![e]\!] \cup \text{sat}(!b) \circ B_i[\![f]\!]$
$e \cdot f$	$B_i[\![e]\!] \circ B_i[\![f]\!]$
e^c	$\text{sat}(c) \circ B_i[\![e]\!] \circ B_i[\![e^c]\!] \cup \text{sat}(!c)$

Axioms

Guarded Union

To warm up, let’s look at axioms for guarded union…

Guarded Union

To warm up, let's look at axioms for guarded union...

U1: $e +_b e \equiv e$

Guarded Union

To warm up, let’s look at axioms for guarded union…

U1: $e +_b e \equiv e$

U2: $e +_b e' \equiv e' +_{!b} e$

Guarded Union

To warm up, let’s look at axioms for guarded union...

U1: $e +_b e \equiv e$

U2: $e +_b e' \equiv e' +_{!b} e$

U3: $(e1 +_b e2) +_c e3 \equiv e1 +_{bc} (e2 +_c e3)$

Guarded Union

To warm up, let's look at axioms for guarded union…

U1: $e +_{b} e \equiv e$

U2: $e +_{b} e' \equiv e' +_{!b} e$

U3: $(e1 +_{b} e2) +_{c} e3 \equiv e1 +_{bc} (e2 +_{c} e3)$

U4: $e +_{b} e' \equiv be +_{b} e'$

Guarded Union

To warm up, let's look at axioms for guarded union...

U1: $e +_b e \equiv e$

U2: $e +_b e' \equiv e' +_{!b} e$

U3: $(e1 +_b e2) +_c e3 \equiv e1 +_{bc} (e2 +_c e3)$

U4: $e +_b e' \equiv be +_b e'$

U5: $(e1 +_b e2) \cdot f \equiv e1 \cdot f +_b e2 \cdot f$

Derivable equivalences

Theorem: `e` $+_{\texttt{b}}$ `0` ≡ `be`

Derivable equivalences

Theorem: $\texttt{e} +_{\texttt{b}} \texttt{0} \equiv \texttt{be}$

$\texttt{e} +_{\texttt{b}} \texttt{0}$

Derivable equivalences

Theorem: $e +_b 0 \equiv be$

$e +_b 0$

≡ { U4: e +b e' ≡ be +b e'}

$be +_b 0$

Derivable equivalences

Theorem: $\mathtt{e} +_{\mathtt{b}} \mathtt{0} \equiv \mathtt{be}$

$\mathtt{e} +_{\mathtt{b}} \mathtt{0}$

$\equiv$ { U4: e +b e' ≡ be +b e'}

$\mathtt{be} +_{\mathtt{b}} \mathtt{0}$

$\equiv$ { U2: $\mathtt{e} +_{\mathtt{b}} \mathtt{e'} \equiv \mathtt{e'} +_{\mathtt{!b}} \mathtt{e}$}

$\mathtt{0} +_{\mathtt{!b}} \mathtt{be}$

Derivable equivalences

Theorem: $e +_b 0 \equiv be$

$e +_b 0$

$\equiv$ { U4: e +b e' ≡ be +b e'}

$be +_b 0$

$\equiv$ { U2: $e +_b e' \equiv e' +_{!b} e$}

$0 +_{!b} be$

$\equiv$ { Boolean algebra & $0 \equiv 0 \cdot e$}

$!b \cdot b \cdot e +_{!b} be$

Derivable equivalences

Theorem: $e +_b 0 \equiv be$

$e +_b 0$

$\equiv$ { U4: $e +b e' \equiv be +b e'$}
$be +_b 0$

$\equiv$ { U2: $e +_b e' \equiv e' +_{!b} e$}
$0 +_{!b} be$

$\equiv$ { Boolean algebra & $0 \equiv 0 \cdot e$}
$!b \cdot b \cdot e +_{!b} be$

$\equiv$ { U4: $e +_b e' \equiv be +_b e'$}
$be +_{!b} be$

Derivable equivalences

Theorem: $e +_b 0 \equiv be$

$e +_b 0$

$\equiv$ { U4: e +b e' ≡ be +b e'}

$be +_b 0$

$\equiv$ { U2: $e +_b e' \equiv e' +_{!b} e$}

$0 +_{!b} be$

$\equiv$ { Boolean algebra & $0 \equiv 0 \cdot e$}

$!b \cdot b \cdot e +_{!b} be$

$\equiv$ { U4: $e +_b e' \equiv be +_b e'$}

$be +_{!b} be$

$\equiv$ { U1: $e +_b e \equiv e$ }

be □

Guarded Iteration

For guarded iteration, we use a “Salomaa-style” characterization of the fixed point

Guarded Iteration

For guarded iteration, we use a "Salomaa-style" characterization of the fixed point

Termination

W1: $t \equiv e \cdot t +_b f$ and $E(e) \equiv 0 \Rightarrow$
$t \equiv e^b \cdot f$

Guarded Iteration

For guarded iteration, we use a “Salomaa-style” characterization of the fixed point

Termination

W1: $t \equiv e \cdot t +_b f$ and $E(e) \equiv 0 \Rightarrow$
$t \equiv e^b \cdot f$

W2: $e^b \equiv 1 +_{!b} e \cdot e^b$

Guarded Iteration

For guarded iteration, we use a “Salomaa-style” characterization of the fixed point

Termination

$$\mathtt{W1:}\quad t \equiv e \cdot t +_b f \text{ and } E(e) \equiv 0 \Rightarrow t \equiv e^b \cdot f$$

$$\mathtt{W2:}\quad e^b \equiv 1 +_{!b} e \cdot e^b$$

$$\mathtt{DL:}\quad (e +_b 1)^c \equiv (be)^c$$

Eliminate ∞ loops

Decision Procedure

Decision Procedure

$$\forall i\colon$$

$$B_i[\![\mathbf{e}]\!] = B_i[\![\mathbf{f}]\!]$$

?

Decision Procedure

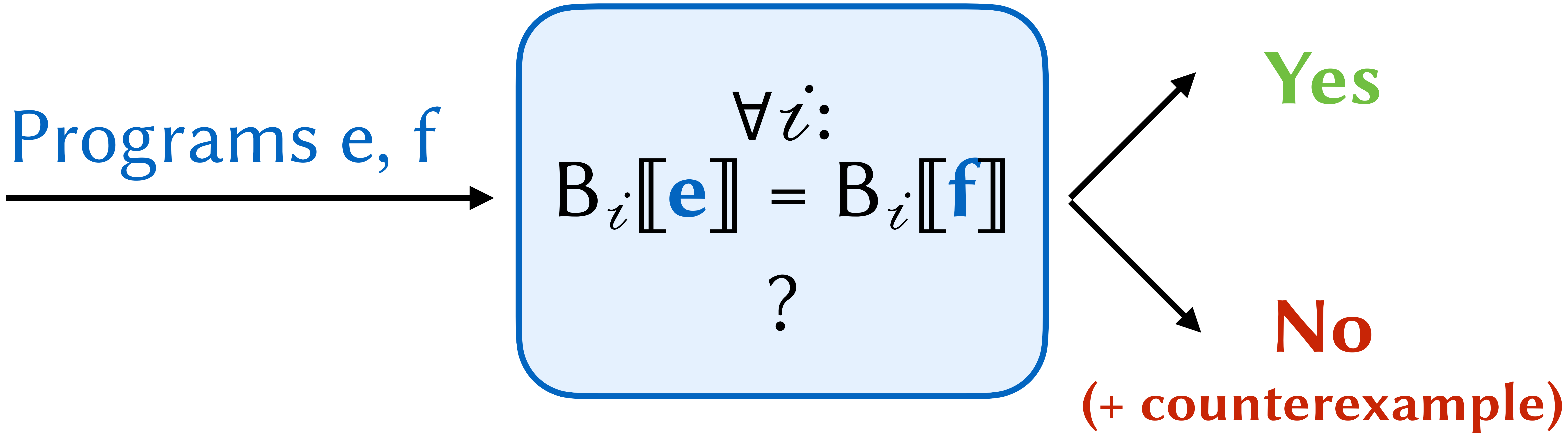
Programs e, f

$$\forall i\text{:}$$

$$\mathrm{B}_i[\![\mathbf{e}]\!] = \mathrm{B}_i[\![\mathbf{f}]\!]$$

?

Decision Procedure



Programs e, f

$\forall i$:

$B_i[\![\mathbf{e}]\!] = B_i[\![\mathbf{f}]\!]$

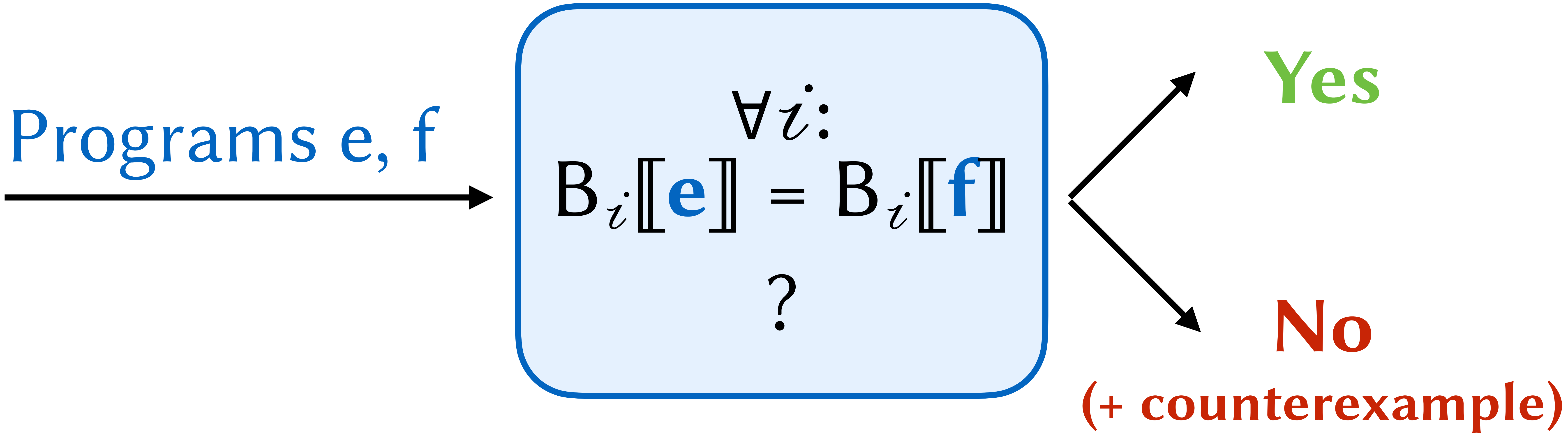
?

Yes

No

(+ counterexample)

Decision Procedure



Programs e, f

$\forall i$:
$B_i[\![\mathbf{e}]\!] = B_i[\![\mathbf{f}]\!]$
?

Yes

No
(+ counterexample)

Key Challenge:

There are infinitely many interpretations i!

Overview

1. **Develop "universal semantics"** (aka free model)
   i) $[\![e]\!] = [\![f]\!] \iff \forall i.\ B_i[\![e]\!] = B_i[\![f]\!]$
   ii) $[\![e]\!]$ is a set of strings (i.e., formal languages)

2. **Develop automaton model** (aka coalgebra)
   i) algorithm $e \mapsto A_e$
   ii) automaton A_e recognizes language $[\![e]\!]$
   iii) $|A_e| \in O(|e|)$
   iv) A_e is deterministic

3. **Decide $e \equiv f$**
   i) check bisimilarity $A_e \sim A_f$
   ii) using Hopcroft-Karp: $O^*(|A_e| + |A_f|)$

Language Model

Language Model

Interpret program as **set of successful "runs"** it induces:

$$[\![p]\!] := \{ \text{ runs of p } \}$$

Language Model

Interpret program as **set of successful "runs"** it induces:

$$[\![p]\!] := \{ \text{ runs of } p \}$$

Atomic predicates ("truth assignments")

$$\text{Atom} := \{ \prod_{t \in \text{Test}} \text{lit}_t \mid \text{lit}_t \in \{t, \neg t\} \}$$

Language Model

Interpret program as **set of successful "runs"** it induces:

$$[\![p]\!] := \{ \text{ runs of } p \}$$

Atomic predicates ("truth assignments")

$$\text{Atom} := \{ \bigsqcap_{t \in \text{Test}} \text{lit}_t \mid \text{lit}_t \in \{t, \neg t\} \}$$

Runs are finite strings of the form

$$\alpha_0 \, p_1 \, \alpha_1 \ldots p_n \, \alpha_n \in \text{Atom} \cdot (\text{Action} \cdot \text{Atom})^*$$

initial state

actions

final state

Language Model

Interpret program as **set of successful “runs"**:

$$[\![e]\!] \subseteq \text{Atom}\cdot(\text{Action}\cdot\text{Atom})^*$$

Language Model

Interpret program as **set of successful “runs"**:

$$[\![e]\!] \subseteq \text{Atom}\cdot(\text{Action}\cdot\text{Atom})^*$$

$$[\![b]\!] := \{\, \alpha \in \text{Atom} \mid \alpha \Longrightarrow b \text{ is tautology} \,\}$$

Language Model

Interpret program as **set of successful “runs"**:

$[\![e]\!] \subseteq \text{Atom}\cdot(\text{Action}\cdot\text{Atom})^*$

$[\![b]\!] := \{\, \alpha \in \text{Atom} \mid \alpha \Longrightarrow b \text{ is tautology} \,\}$

$[\![p]\!] := \{\, \alpha p \beta \mid \alpha, \beta \in \text{Atom} \,\}$

Language Model

Interpret program as **set of successful "runs"**:

$[\![e]\!] \subseteq \mathrm{Atom}\cdot(\mathrm{Action}\cdot\mathrm{Atom})^*$

$[\![b]\!] := \{\, \alpha \in \mathrm{Atom} \mid \alpha \Longrightarrow b \text{ is tautology} \,\}$

$[\![p]\!] := \{\, \alpha p \beta \mid \alpha, \beta \in \mathrm{Atom} \,\}$

$[\![e\cdot f]\!] := \{\, v\alpha w \mid v\alpha \in [\![e]\!], \alpha w \in [\![f]\!] \,\}$

Language Model

Interpret program as **set of successful "runs"**:

$[\![e]\!] \subseteq \text{Atom}\cdot(\text{Action}\cdot\text{Atom})^*$

$[\![b]\!] := \{\, \alpha \in \text{Atom} \mid \alpha \Rightarrow b \text{ is tautology} \,\}$

$[\![p]\!] := \{\, \alpha p \beta \mid \alpha, \beta \in \text{Atom} \,\}$

$[\![e \cdot f]\!] := \{\, v\alpha w \mid v\alpha \in [\![e]\!], \alpha w \in [\![f]\!] \,\}$

$[\![e +_b f]\!] := [\![b \cdot e]\!] \cup [\![\neg b \cdot f]\!]$

Language Model

Interpret program as **set of successful “runs"**:

$[\![e]\!] \subseteq \mathrm{Atom}\cdot(\mathrm{Action}\cdot\mathrm{Atom})^*$

$[\![b]\!] := \{ \alpha \in \mathrm{Atom} \mid \alpha \Longrightarrow b \text{ is tautology} \}$

$[\![p]\!] := \{ \alpha p \beta \mid \alpha, \beta \in \mathrm{Atom} \}$

$[\![e\cdot f]\!] := \{ v\alpha w \mid v\alpha \in [\![e]\!], \alpha w \in [\![f]\!] \}$

$[\![e +_b f]\!] := [\![b\cdot e]\!] \cup [\![\neg b\cdot f]\!]$

$[\![e^b]\!] := \{ w \in [\![(b\cdot e)^n\cdot(\neg b)]\!] \mid n \geq 0 \}$

Language Model

Interpret program as **set of successful "runs"**:

$$[\![e]\!] \subseteq \text{Atom}\cdot(\text{Action}\cdot\text{Atom})^*$$

$$[\![b]\!] := \{\, \alpha \in \text{Atom} \mid \alpha \Longrightarrow b \text{ is tautology} \,\}$$

$$[\![p]\!] := \{\, \alpha p \beta \mid \alpha, \beta \in \text{Atom} \,\}$$

$$[\![e\cdot f]\!] := \{\, v\alpha w \mid v\alpha \in [\![e]\!], \alpha w \in [\![f]\!] \,\}$$

$$[\![e +_b f]\!] := [\![b\cdot e]\!] \cup [\![\neg b\cdot f]\!]$$

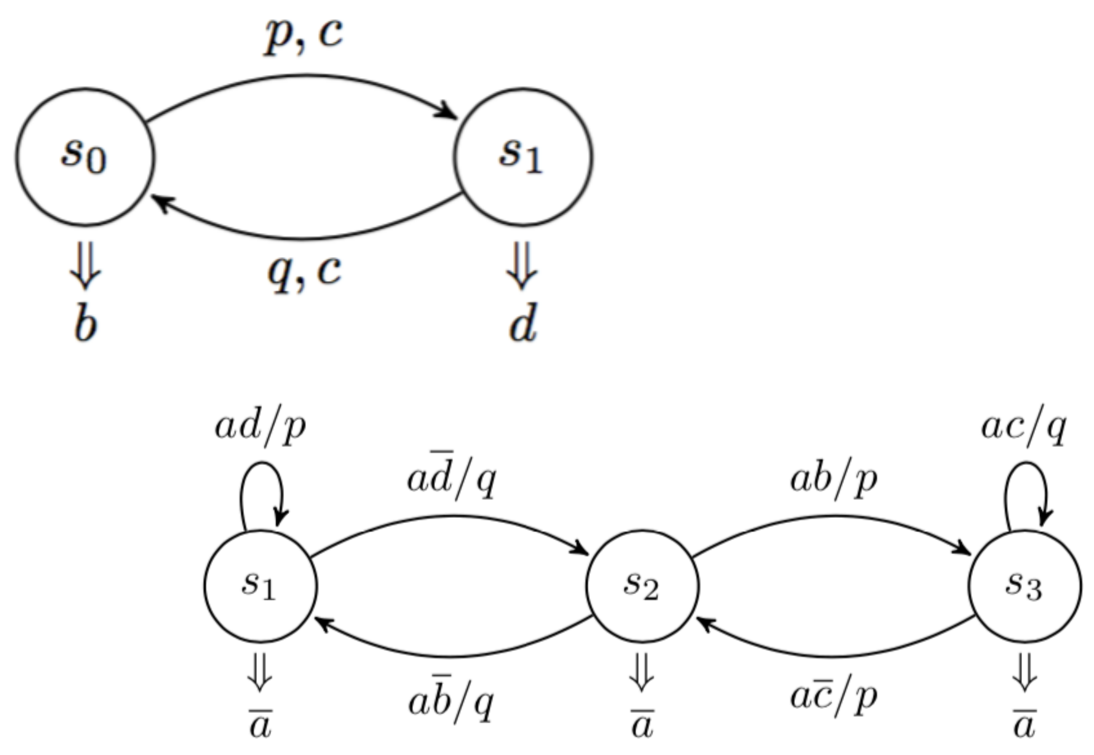
$$[\![e^b]\!] := \{\, w \in [\![(b\cdot e)^n\cdot(\neg b)]\!] \mid n \geq 0 \,\}$$

Theorem[Soundness] : Axioms sound with respect to the Language Model:

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

Kleene Theorem

Automata



Programs

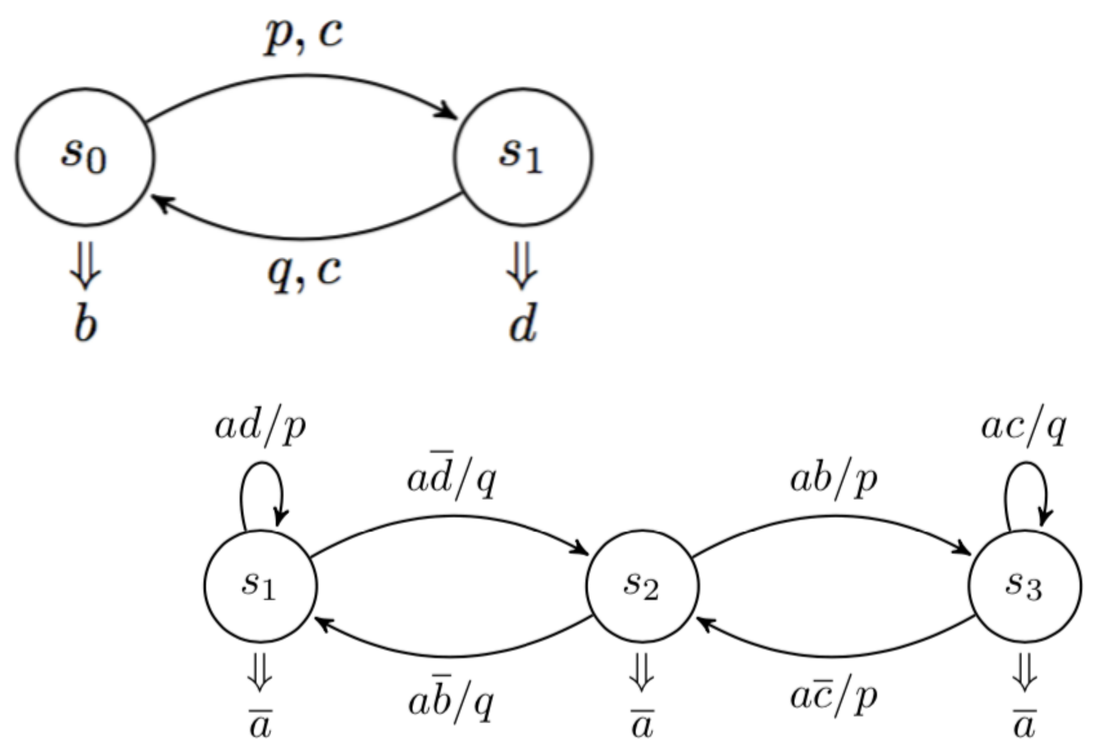
$e^{(bc)} \cdot e^{(c)}$

$(e1 +_b e2) \cdot f$

$(e^{(b)} \cdot f)^{(c)}$

Kleene Theorem

Automata



Programs

$e^{(bc)} \cdot e^{(c)}$

$(e1 +_b e2) \cdot f$

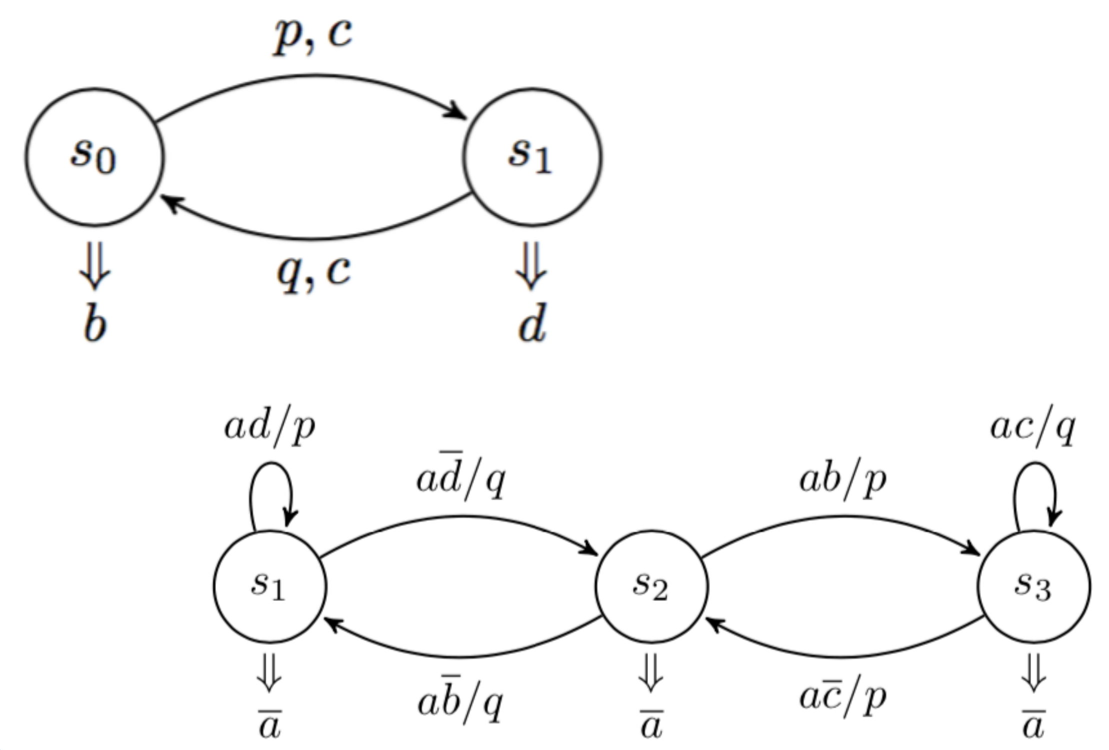
$(e^{(b)} \cdot f)^{(c)}$

check bisimilarity $A_e \sim A_f$

$e \equiv f$

Kleene Theorem

Automata ⟺ **Programs**



$e^{(bc)} \cdot e^{(c)}$

$(e1 +_b e2) \cdot f$

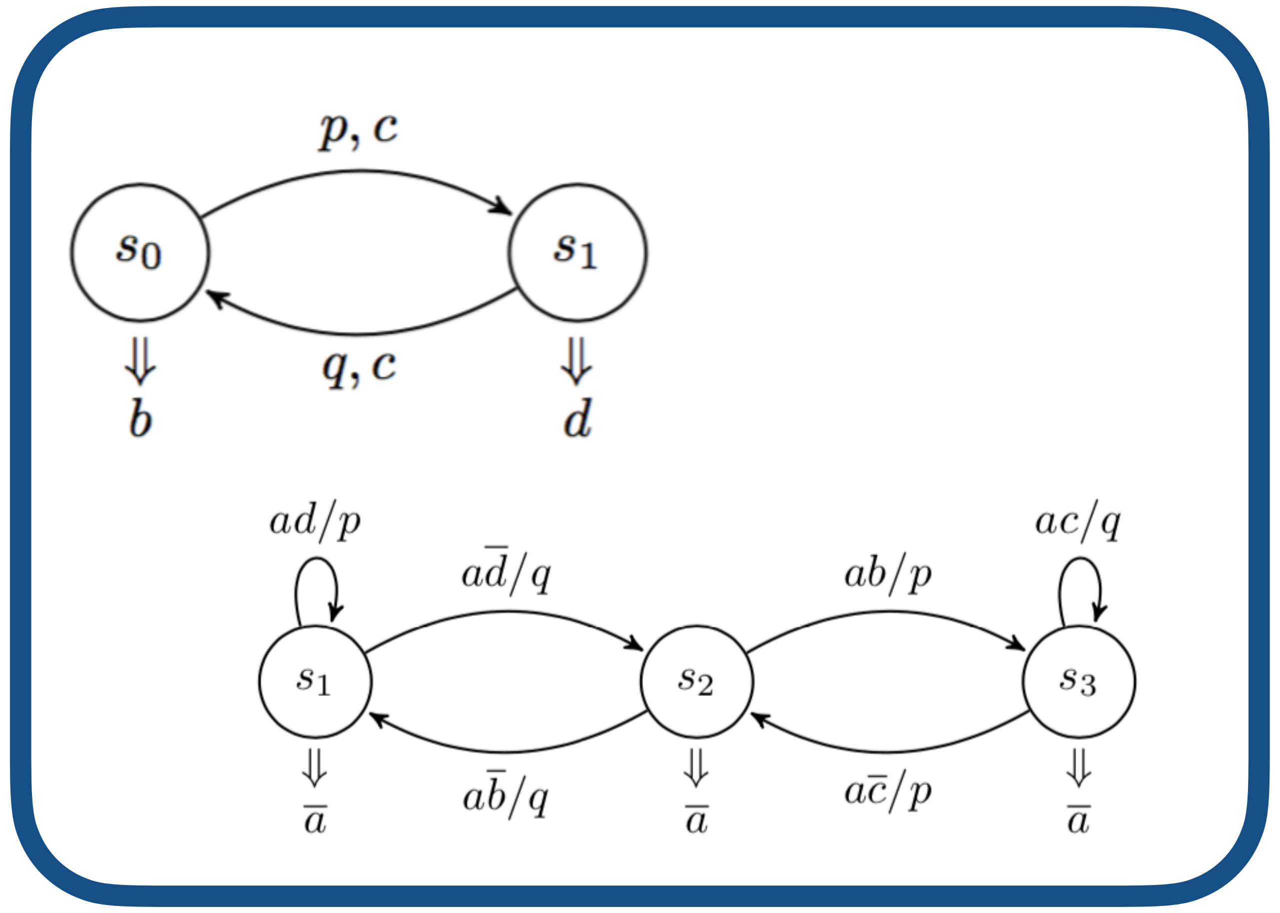
$(e^{(b)} \cdot f)^{(c)}$

check bisimilarity $A_e \sim A_f$ ⟸ $e \equiv f$

$A_1 \sim A_2$ ⟹ $e_1 \equiv e_2$

Kleene Theorem

Automata



Programs

$\mathtt{e}^{(\mathtt{bc})} \cdot \mathtt{e}^{(\mathtt{c})}$

$(e1 +_b e2) \cdot f$

$(\mathtt{e}^{(\mathtt{b})} \cdot \mathtt{f})^{(\mathtt{c})}$

check bisimilarity $A_e \sim A_f$ **Decidability** $e \equiv f$

+

$A_1 \sim A_2$ **Completeness** $e_1 \equiv e_2$

Automata

(Un)Successful termination

State of program

$$S \xrightarrow{\delta} (2 + \Sigma \times S)^{\mathsf{At}}$$

Automata

(Un)Successful termination

State of program

$$S \xrightarrow{\delta} (2 + \Sigma \times S)^{\mathsf{At}}$$

$L(s) \subseteq \mathrm{Atom}\cdot(\mathrm{Action}\cdot\mathrm{Atom})^*$

Semantics
=
Guarded Language

Automata

(Un)Successful termination

State of program

$$S \xrightarrow{\delta} (2 + \Sigma \times S)^{\mathsf{At}}$$

$L(s) \subseteq \text{Atom}\cdot(\text{Action}\cdot\text{Atom})^*$

$\alpha \in L(s) \iff \delta(s)(\alpha)=1$

Semantics = Guarded Language

Automata

(Un)Successful termination

State of program

$$S \xrightarrow{\delta} (2 + \Sigma \times S)^{\mathsf{At}}$$

$$L(s) \subseteq \mathrm{Atom}\cdot(\mathrm{Action}\cdot\mathrm{Atom})^*$$

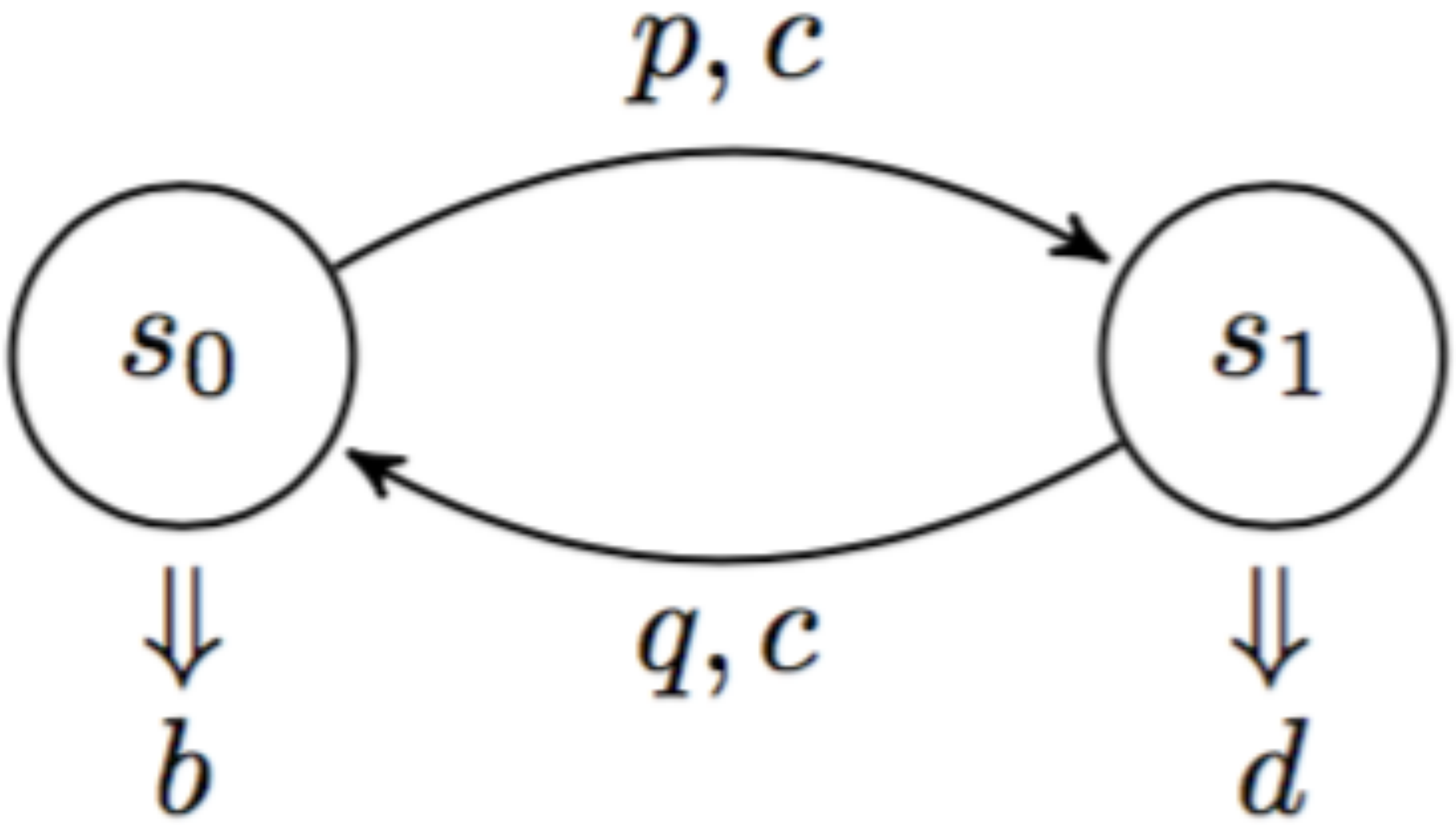
$$\alpha \in L(s) \iff \delta(s)(\alpha)=1$$

$$\alpha p w \in L(s) \iff \delta(s)(\alpha)= \langle p,s'\rangle \text{ and } w \in L(s')$$

Semantics
=
Guarded Language

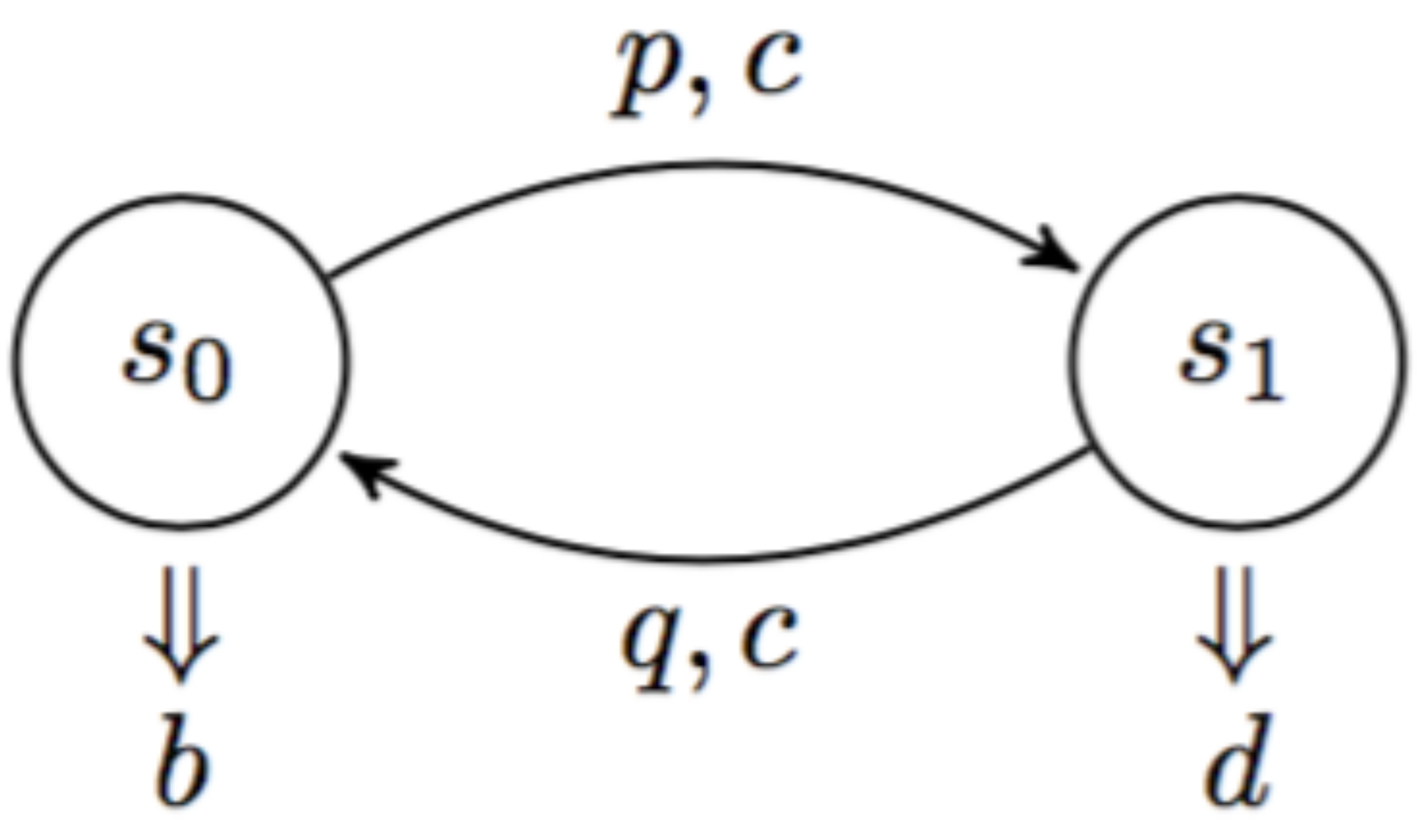
Challenge

Not all automata correspond to a GKAT program!



Challenge

Not all automata correspond to a GKAT program!



```
while !b do
  assert c;
  p
  if c then
    q
  else
    break
done
```

Well-Nested Loops

Idea

- Characterize automata that correspond to well-nested GKAT programs
- Intuitively, we need a way capture the uniform interface between each well-nested loop and its surrounding context

Conclusion

- GKAT is a framework to reason about simple while programs.
- Efficient fragment of KAT
- Equational theory is surprisingly subtle

Conclusion

- GKAT is a framework to reason about simple while programs.
- Efficient fragment of KAT
- Equational theory is surprisingly subtle

Ongoing work

Conclusion

- GKAT is a framework to reason about simple while programs.
- Efficient fragment of KAT
- Equational theory is surprisingly subtle

Ongoing work

- **Finer-grained semantics**

Conclusion

- GKAT is a framework to reason about simple while programs.
- Efficient fragment of KAT
- Equational theory is surprisingly subtle

Ongoing work

- **Finer-grained semantics**
  - alternative language model including infinite strings

Conclusion

- GKAT is a framework to reason about simple while programs.
- Efficient fragment of KAT
- Equational theory is surprisingly subtle

Ongoing work

- **Finer-grained semantics**
  - alternative language model including infinite strings
  - can distinguish \`**while** true **do** p\` from \`**assert** false\`

Conclusion

- GKAT is a framework to reason about simple while programs.
- Efficient fragment of KAT
- Equational theory is surprisingly subtle

Ongoing work

- **Finer-grained semantics**
  - alternative language model including infinite strings
  - can distinguish `**while** true **do** p` from `**assert** false`
- Probabilistic extension

Conclusion

- GKAT is a framework to reason about simple while programs.
- Efficient fragment of KAT
- Equational theory is surprisingly subtle

Ongoing work

- **Finer-grained semantics**
  - alternative language model including infinite strings
  - can distinguish `**while** true **do** p` from `**assert** false`
- Probabilistic extension
  - efficient fragment of ProbNetKAT

Conclusion

- GKAT is a framework to reason about simple while programs.
- Efficient fragment of KAT
- Equational theory is surprisingly subtle

Ongoing work

- **Finer-grained semantics**
  - alternative language model including infinite strings
  - can distinguish \`**while** true **do** p\` from \`**assert** false\`
- Probabilistic extension
  - efficient fragment of ProbNetKAT
  - reasoning about congestion, failure resilience, reliability