Kleene Algebra with Tests

Nate Foster
Justin Hsu
Tobias Kappé
Dexter Kozen
Steffen Smolka
Alexandra Silva

Kleene **A**lgebra with **T**ests

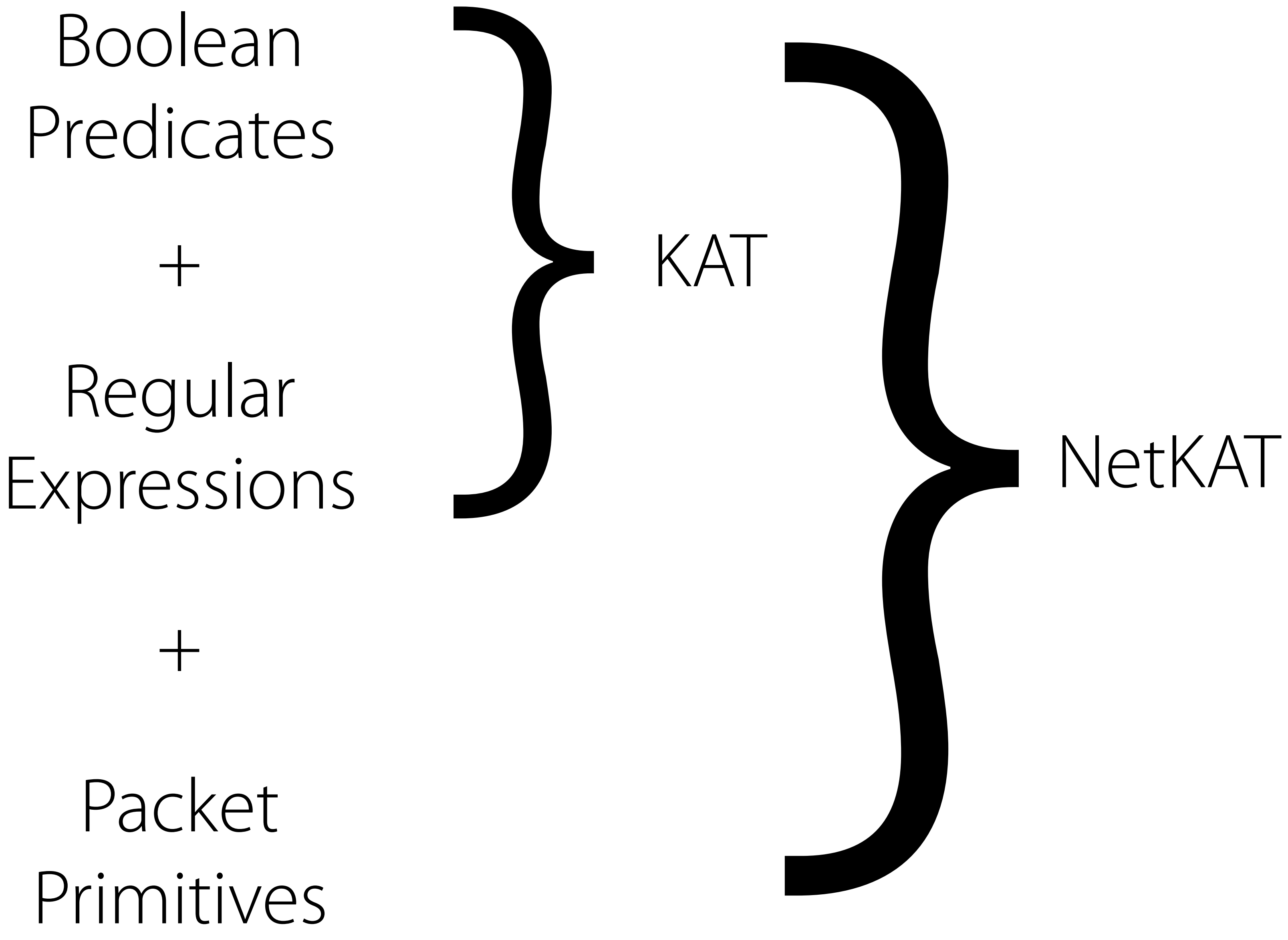
Nate Foster
Justin Hsu
Tobias Kappé
Dexter Kozen
Steffen Smolka
Alexandra Silva

Guarded **K**leene **A**lgebra with **T**ests

Nate Foster
Justin Hsu
Tobias Kappé
Dexter Kozen
Steffen Smolka
Alexandra Silva

NetKAT [POPL ’14]

P ::=
| **false**
| **true**
| field = val
| field **:=** val
| !p
| p_1 **+** p_2
| p_1 • p_2
| p*
| **A→B**



Boolean Predicates

+

Regular Expressions

KAT

+

Packet Primitives

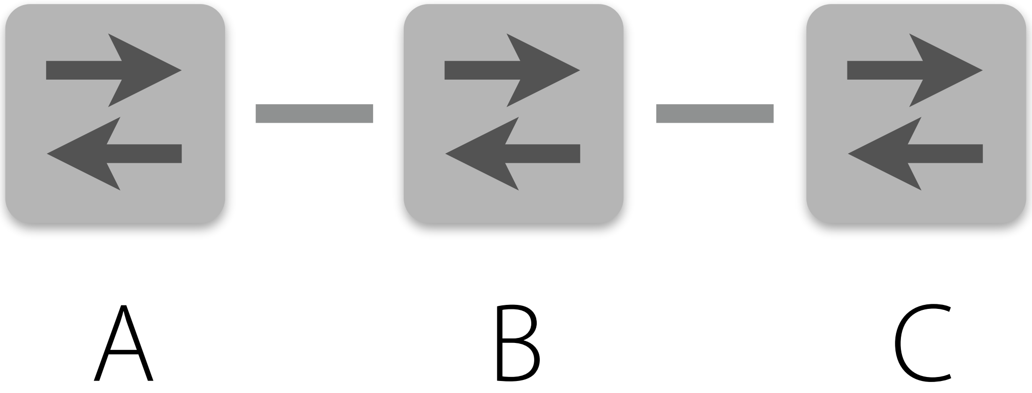
NetKAT

Encoding Routers

It is straightforward to encode router forwarding tables and network topologies into NetKAT…

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

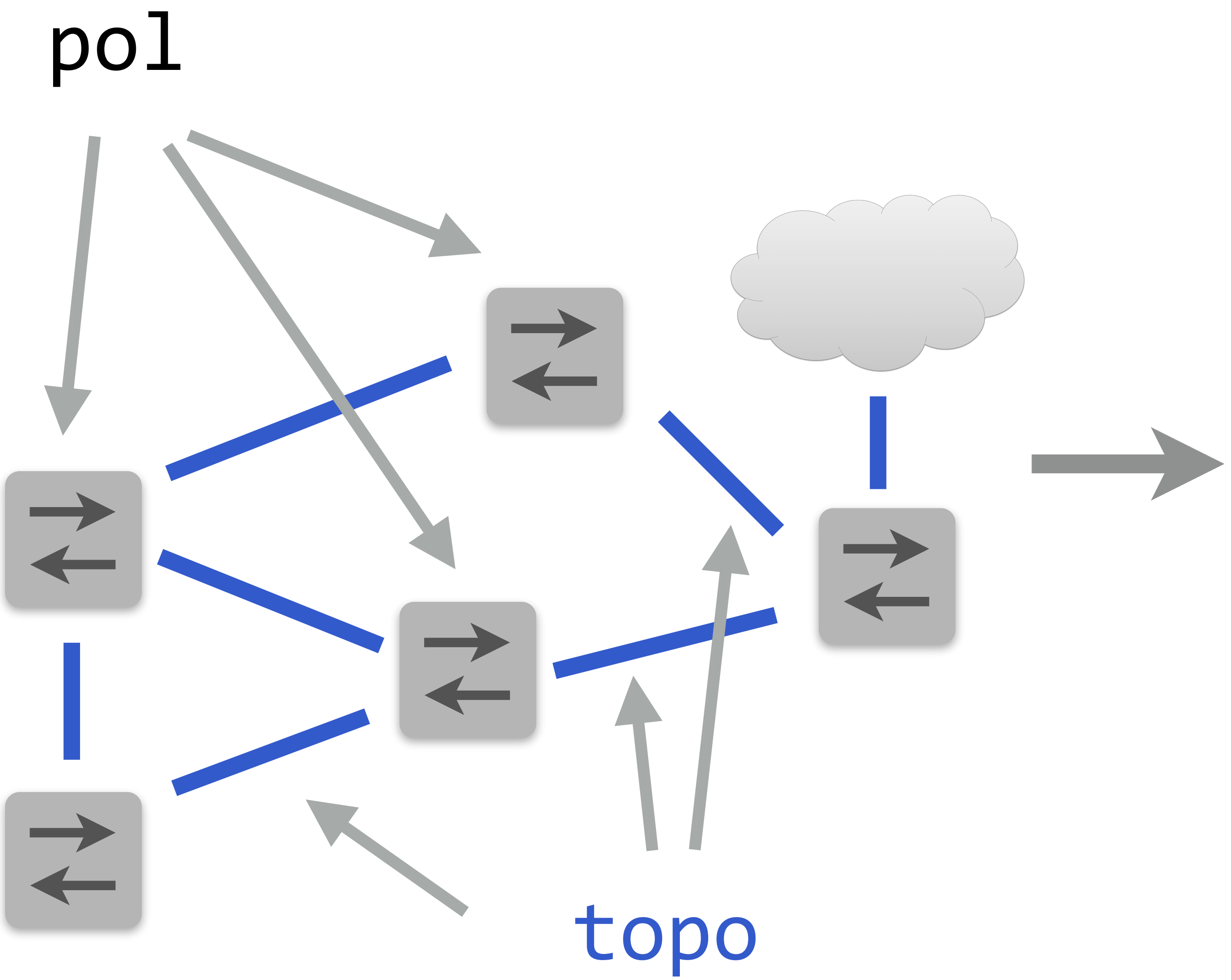
```
if dstport=22 then false
elsif srcip=10.0.0.1 then port := 1
else port := 2
```



A → B + B → A + B → C + C → B

Encoding Networks

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

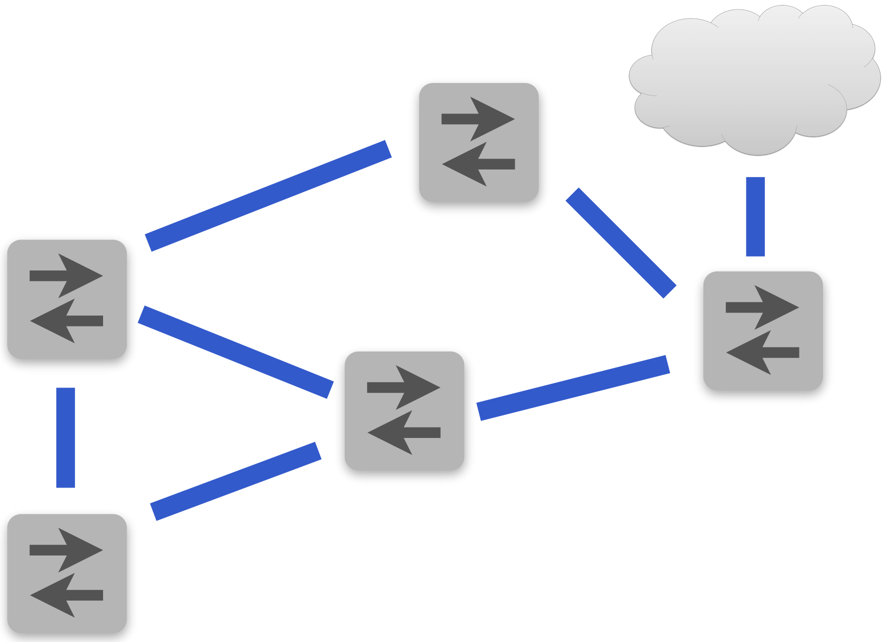


pol

topo

(pol • topo)*

Formal Reasoning



$\models \phi$

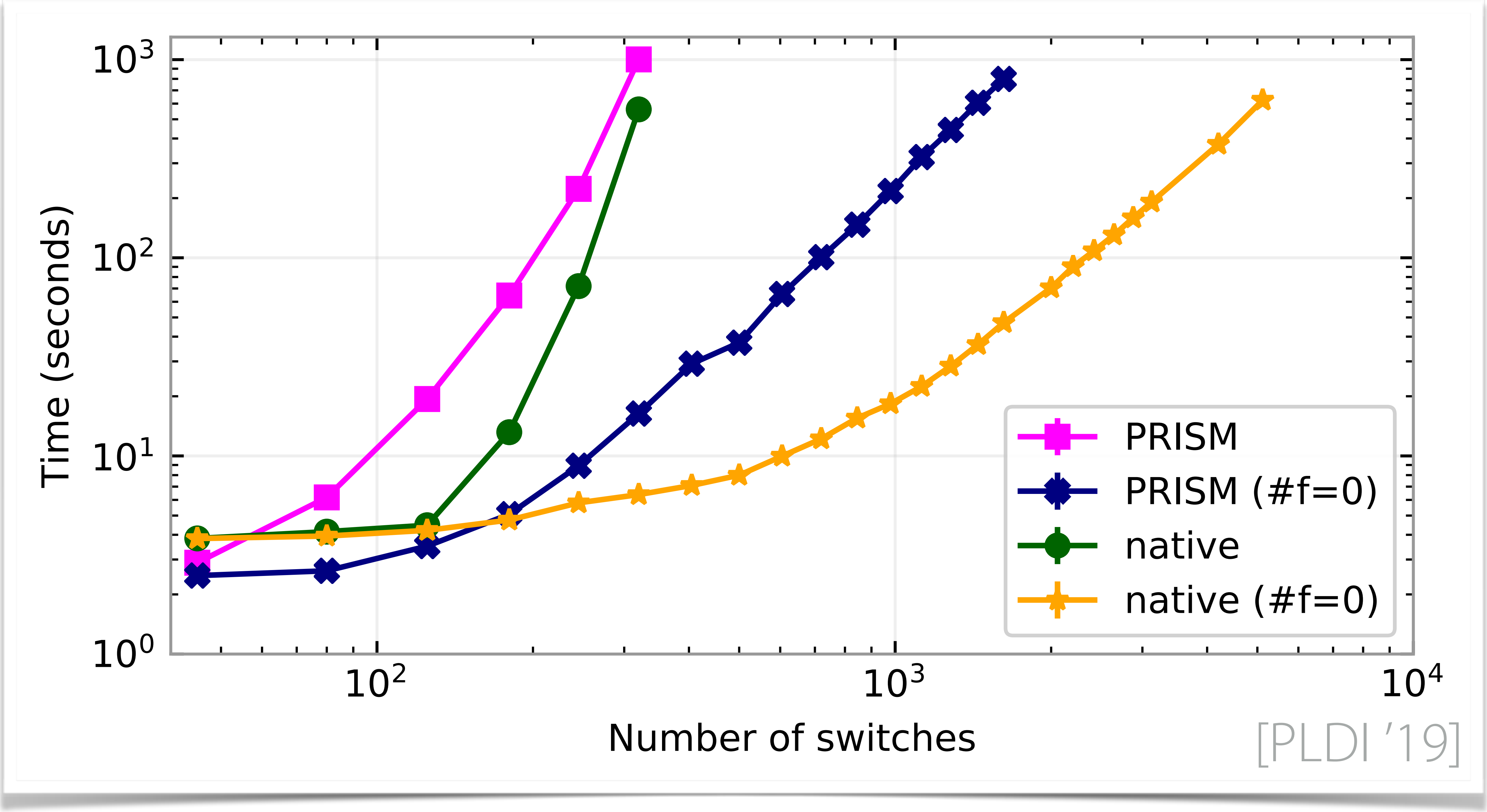
Given a network encoded this way, we'd like to be able to automatically answer questions like:

"Does the network isolate A and B?"

Can reduce this question (and others) to equivalence

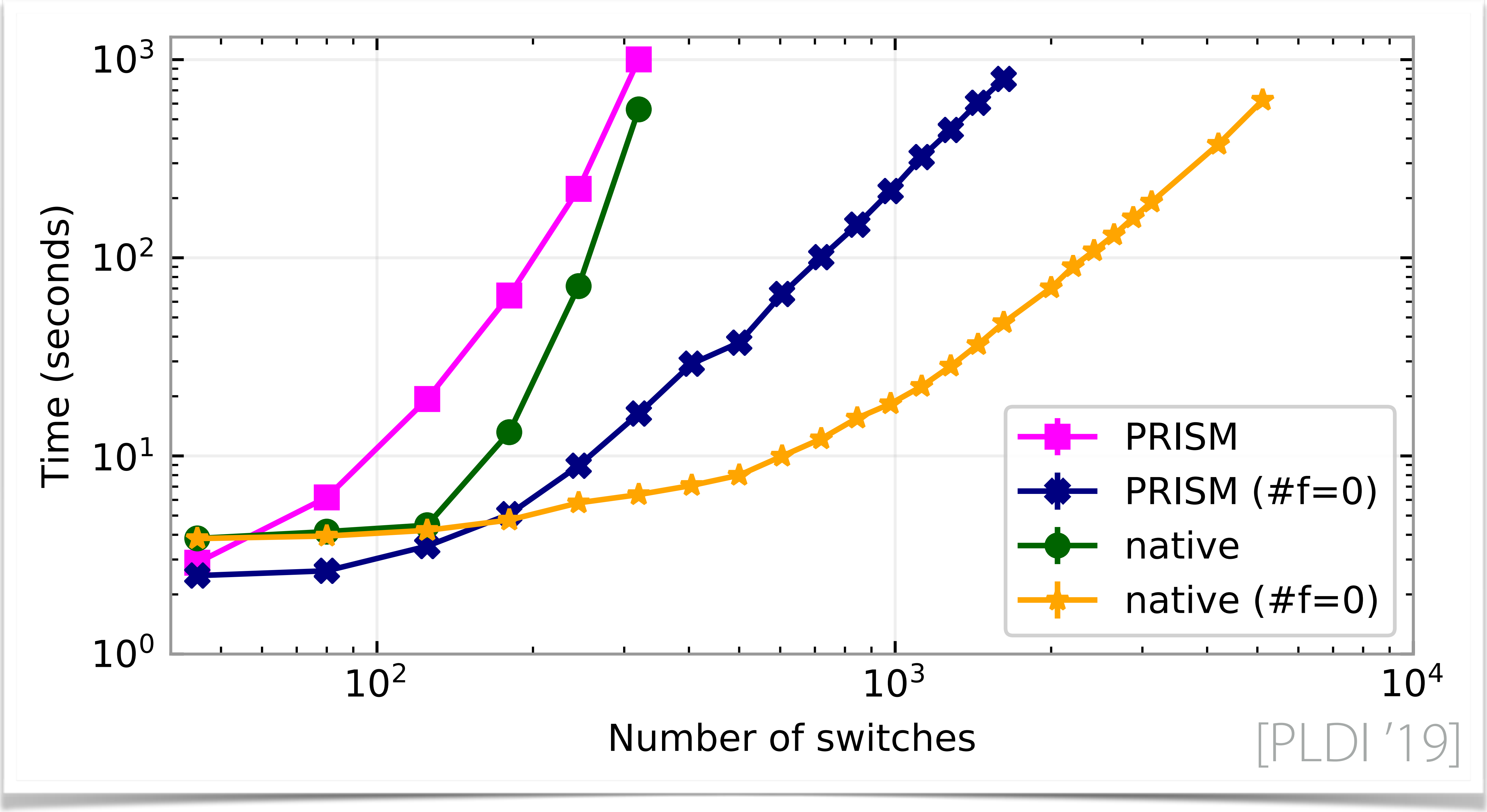
$$A \bullet (\mathrm{pol} \bullet \mathrm{topo})^* \bullet B \equiv \mathrm{false}$$

Why does this work?

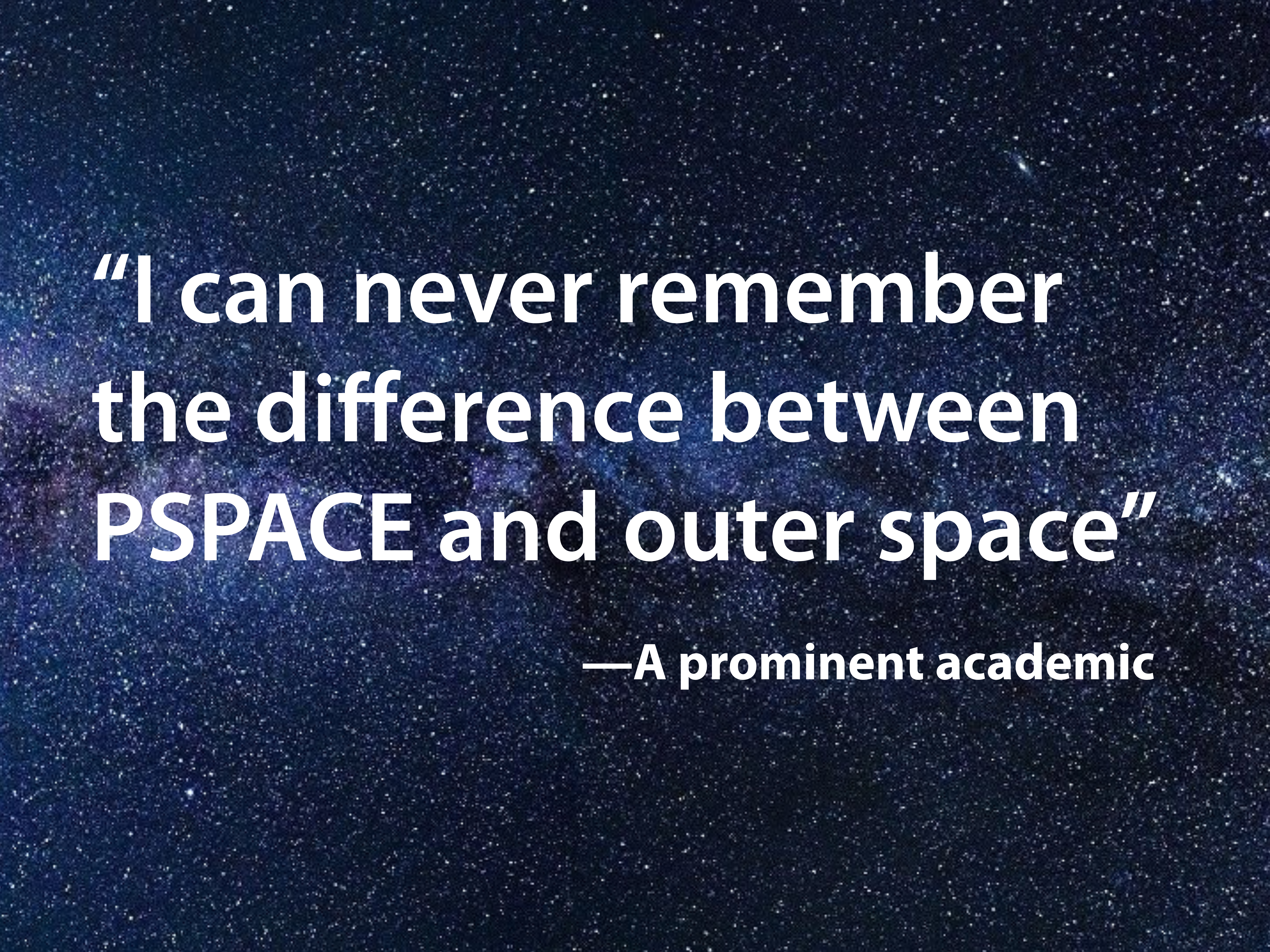


[PLDI ’19]

Why does this work?



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



“I can never remember the difference between PSPACE and 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

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 ∈ Action
- finite set of **primitive tests** t ∈ 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

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

GKAT Syntax

Parameters

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

Syntax

Boolean algebra

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

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

GKAT Syntax

Parameters

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

Syntax

Boolean algebra

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

e, f, g ∈ Exp ::=

|	b ∈ BExp	**assert** b
|	p ∈ Action	**do** p

GKAT Syntax

Parameters

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

Syntax

Boolean algebra

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

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

$\mid \ b \in \text{BExp}$	**assert** b
$\mid \ p \in \text{Action}$	**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 $B_i[\![e]\!] \subseteq \text{State} \times \text{State}$

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

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 = (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

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

Semantics $B_i\llbracket e \rrbracket \subseteq \text{State} \times \text{State}$

e	$B_i\llbracket e \rrbracket$
`b`	sat(b)
`p`	eval(p)
`e +`$_b$` f`	$\text{sat}(b) \circ B_i\llbracket e \rrbracket \cup \text{sat}(!b) \circ B_i\llbracket f \rrbracket$
`e·f`	$B_i\llbracket e \rrbracket \circ B_i\llbracket f \rrbracket$

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

Axioms

Guarded Union

To warm up, look at the axioms for guarded union…

Guarded Union

To warm up, look at the axioms for guarded union…

U1: $e +_b e \equiv e$

Guarded Union

To warm up, look at the axioms for guarded union…

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

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

Guarded Union

To warm up, look at the 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, look at the 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, look at the 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: $\texttt{e} \ +_{\texttt{b}} \ \texttt{0} \equiv \texttt{be}$

Derivable equivalences

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

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

Derivable equivalences

Theorem: $e +_b 0 \equiv be$

$e +_b 0$

$\equiv$ { U4: e +b e' ≡ be +b e'}

$be +_b 0$

Derivable equivalences

Theorem: $e +_b 0 \equiv be$

$e +_b 0$

$\equiv$ { U4: e +b e' ≡ be +b e'}

$be +_b 0$

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

$0 +_{!b} be$

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 & $0 \equiv 0\cdot e$}

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

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

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

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

$be +_{!b} be$

$\equiv$ { 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

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

$$t \equiv e^b \cdot f$$

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

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

$$t \equiv e^b \cdot f$$

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

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

Eliminate ∞ loops

Termination and Continuation

$E(b) = b$

$E(p) = 0$

$E(e^c) = !c$

$E(e +_b f) = b \cdot E(e) + !b \cdot E(f)$

$E(e \cdot f) = E(e) \cdot E(f)$

Termination and Continuation

$E(b) = b$

$E(p) = 0$

$E(e^c) = !c$

$E(e +_b f) = b \cdot E(e) + !b \cdot E(f)$

$E(e \cdot f) = E(e) \cdot E(f)$

$D(b) = 0$

$D(p) = p$

$D(e^c) = c \cdot D(e) \cdot e^c$

$D(e +_b f) = D(e) +_b D(f)$

$D(e \cdot f) = D(f) +_{E(e)} D(e) \cdot f$

Termination and Continuation

$E(b) = b$

$E(p) = 0$

$E(e^c) = !c$

$E(e +_b f) = b \cdot E(e) + !b \cdot E(f)$

$E(e \cdot$

Theorem[FT]: $e \equiv 1 +_{E(e)} D(e)$

$D(b) = 0$

$D(p) = p$

$D(e^c) = c \cdot D(e) \cdot e^c$

$D(e +_b f) = D(e) +_b D(f)$

$D(e \cdot f) = D(f) +_{E(e)} D(e) \cdot f$

Example: Reasoning about Loops

Theorem: $e^{c} \equiv e^{bc} \cdot e^{c}$

Example: Reasoning about Loops

Theorem: $e^{c} \equiv e^{bc} \cdot e^{c}$

e^{c}

Example: Reasoning about Loops

Theorem: $e^c \equiv e^{bc} \cdot e^c$

$$
\begin{aligned}
& e^c \\
= \; & \{ \;\; \mathtt{U1} \;\; \} \\
& e^c \; +_{bc} \; e^c
\end{aligned}
$$

Example: Reasoning about Loops

Theorem: $e^c \equiv e^{bc} \cdot e^c$

```
   e^c
= { U1 }
   e^c +_bc e^c
= { Productive Loop Lemma }
   (!E(e)·D(e))^c +_bc e^c
```

Lemma[Productive]

$e^c \equiv (!E(e) \cdot D(e))^c$

Example: Reasoning about Loops

Theorem: $e^c \equiv e^{bc} \cdot e^c$

$$
\begin{aligned}
& e^c \\
= \; & \{ \text{ U1 } \} \\
& e^c +_{bc} e^c \\
= \; & \{ \text{ Productive Loop Lemma } \} \\
& (!E(e) \cdot D(e))^c +_{bc} e^c \\
= \; & \{ \text{ W2, U4 and BA } \} \\
& !E(e) \cdot D(e) \cdot (!E(e) \cdot D(e))^c +_{bc} e^c
\end{aligned}
$$

Lemma[Productive]

$e^c \equiv (!E(e) \cdot D(e))^c$

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

Example: Reasoning about Loops

Theorem: $e^c \equiv e^{bc} \cdot e^c$

$$
\begin{aligned}
& e^c \\
= \; & \{ \texttt{ U1 } \} \\
& e^c \; +_{bc} \; e^c \\
= \; & \{ \texttt{ Productive Loop Lemma } \} \\
& (!E(e) \cdot D(e))^c \; +_{bc} \; e^c \\
= \; & \{ \texttt{ W2, U4 and BA } \} \\
& !E(e) \cdot D(e) \cdot (!E(e) \cdot D(e))^c \; +_{bc} \; e^c \\
= \; & \{ \texttt{ Productive Loop Lemma } \} \\
& !E(e) \cdot D(e) \cdot e^c \; +_{bc} \; e^c
\end{aligned}
$$

Lemma[Productive]

$e^c \equiv (!E(e) \cdot D(e))^c$

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

Example: Reasoning about Loops

Theorem: $e^{c} \equiv e^{bc} \cdot e^{c}$

e^{c}

$=$ { U1 }

$e^{c} +_{bc} e^{c}$

$=$ { Productive Loop Lemma }

$(!E(e) \cdot D(e))^{c} +_{bc} e^{c}$

$=$ { W2, U4 and BA }

$!E(e) \cdot D(e) \cdot (!E(e) \cdot D(e))^{c} +_{bc} e^{c}$

$=$ { Productive Loop Lemma }

$!E(e) \cdot D(e) \cdot e^{c} +_{bc} e^{c}$

$=$ { W1 }

$(!E(e) \cdot D(e))^{bc} \cdot e^{c}$

Lemma[Productive]

$e^{c} \equiv (!E(e) \cdot D(e))^{c}$

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

W1 $t \equiv e \cdot t +_{b} f \Rightarrow$

$t \equiv e^{b} \cdot f$

Example: Reasoning about Loops

Theorem: $e^{c} \equiv e^{bc} \cdot e^{c}$

e^{c}

$= \{\ \mathrm{U1}\ \}$

$e^{c} +_{bc} e^{c}$

$= \{$ Productive Loop Lemma $\}$

$(!E(e) \cdot D(e))^{c} +_{bc} e^{c}$

$= \{$ W2, U4 and BA $\}$

$!E(e) \cdot D(e) \cdot (!E(e) \cdot D(e))^{c} +_{bc} e^{c}$

$= \{$ Productive Loop Lemma $\}$

$!E(e) \cdot D(e) \cdot e^{c} +_{bc} e^{c}$

$= \{\ \mathrm{W1}\ \}$

$(!E(e) \cdot D(e))^{bc} \cdot e^{c}$

$= \{$ Productive Loop Lemma $\}$

$e^{bc} \cdot e^{c}$ □

Lemma[Productive]

$e^{c} \equiv (!E(e) \cdot D(e))^{c}$

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

W1 $t \equiv e \cdot t +_{b} f \Rightarrow$

$t \equiv e^{b} \cdot f$

Decision Procedure

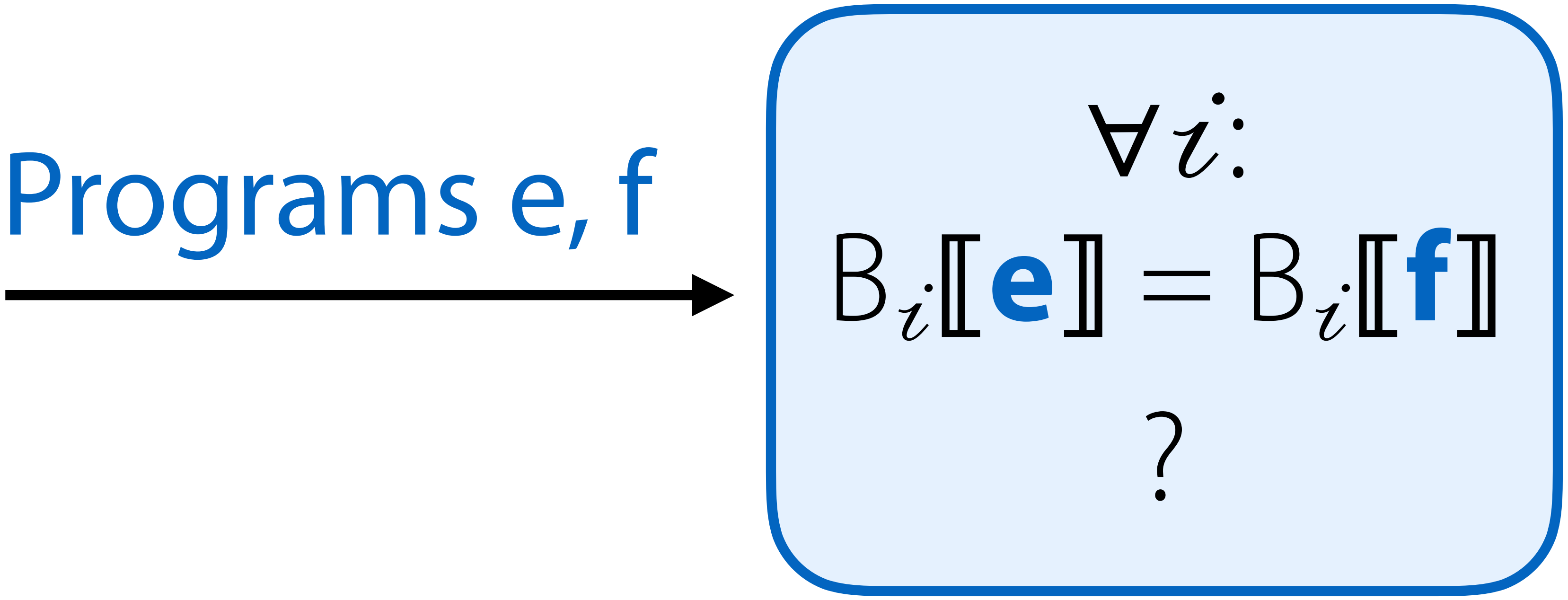
Decision Procedure

$\forall i$:

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

?

Decision Procedure



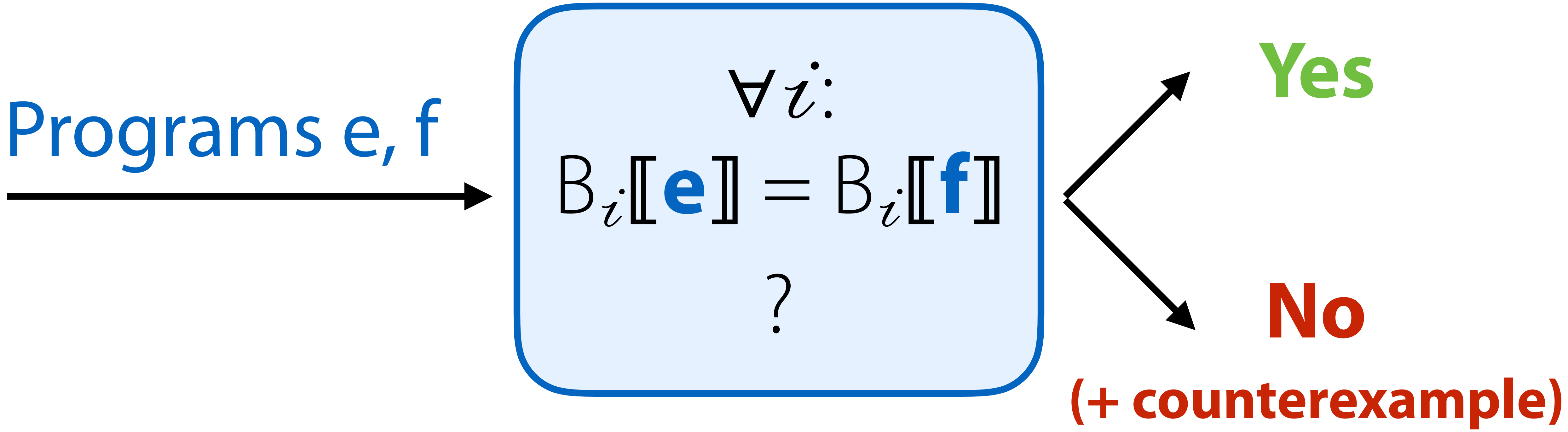
Programs e, f

$\forall i$:

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

?

Decision Procedure



Programs e, f

$$\forall i:$$

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

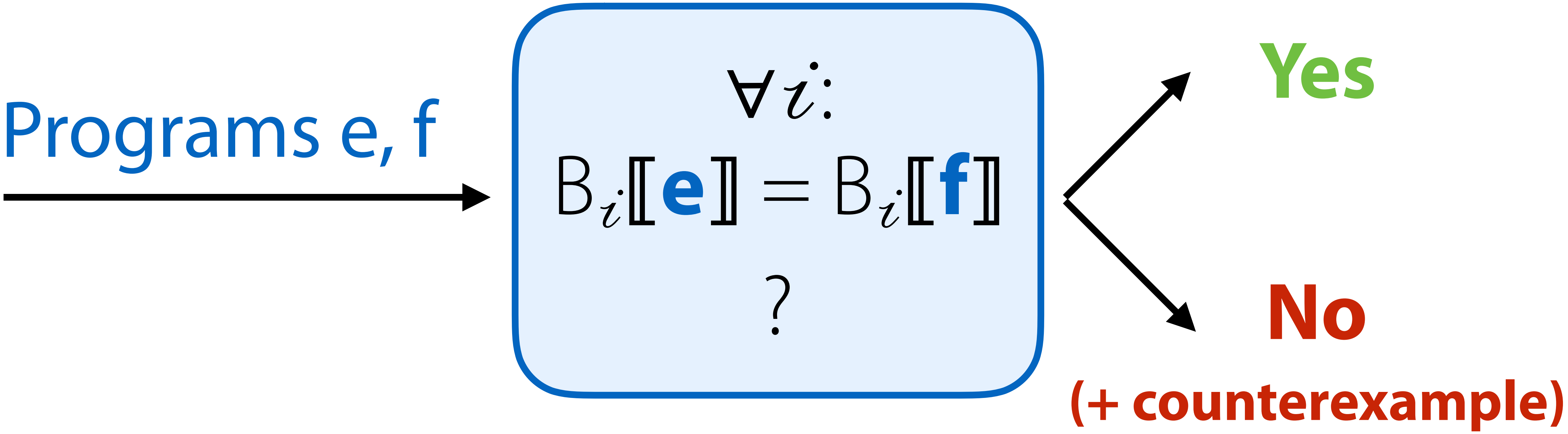
?

Yes

No

(+ counterexample)

Decision Procedure



Programs e, f

$$\forall i: \; \mathrm{B}_i\llbracket \mathbf{e} \rrbracket = \mathrm{B}_i\llbracket \mathbf{f} \rrbracket \; ?$$

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 language)

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")

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

Runs are finite strings of the form

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

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"**:

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

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

Language Model

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

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

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

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

Language Model

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

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

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

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

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

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"**:

$$\llbracket e \rrbracket \subseteq \text{Atom} \cdot (\text{Action} \cdot \text{Atom})^*$$

$$\llbracket b \rrbracket := \{ \alpha \in \text{Atom} \mid \alpha \Rightarrow b \text{ is tautology} \}$$

$$\llbracket p \rrbracket := \{ \alpha p \beta \mid \alpha, \beta \in \text{Atom} \}$$

$$\llbracket e \cdot f \rrbracket := \{ v\alpha w \mid v\alpha \in \llbracket e \rrbracket, \alpha w \in \llbracket f \rrbracket \}$$

$$\llbracket e +_b f \rrbracket := \llbracket b \cdot e \rrbracket \cup \llbracket \neg b \cdot f \rrbracket$$

$$\llbracket e^b \rrbracket := \{ w \in \llbracket (b \cdot e)^n \cdot (\neg b) \rrbracket \mid n \geq 0 \}$$

Language Model

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

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

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

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

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

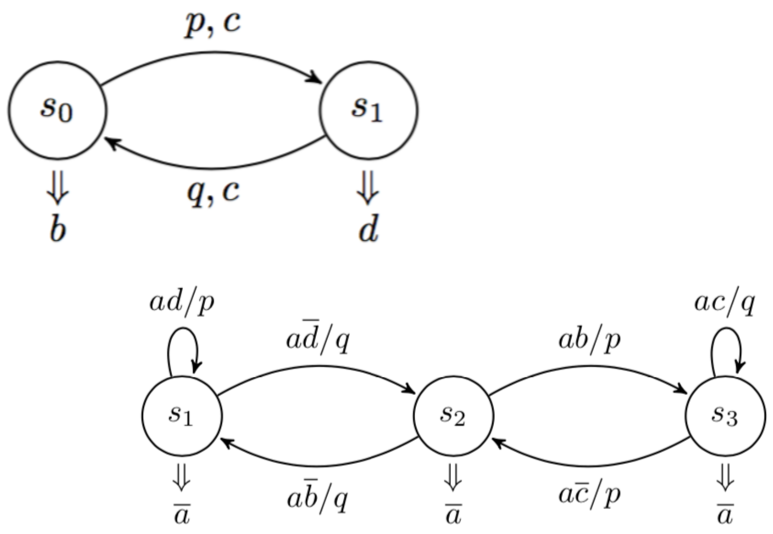
$$[\![e +_b f]\!] := [\![b \cdot e]\!] \cup [\![\neg b \cdot f]\!]$$

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

Theorem[Soundness] : Axioms sound with respect to the Language Model: $\mathbf{e} \equiv \mathbf{f} \Rightarrow [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$

Kleene Theorem

Automata



↔

Programs

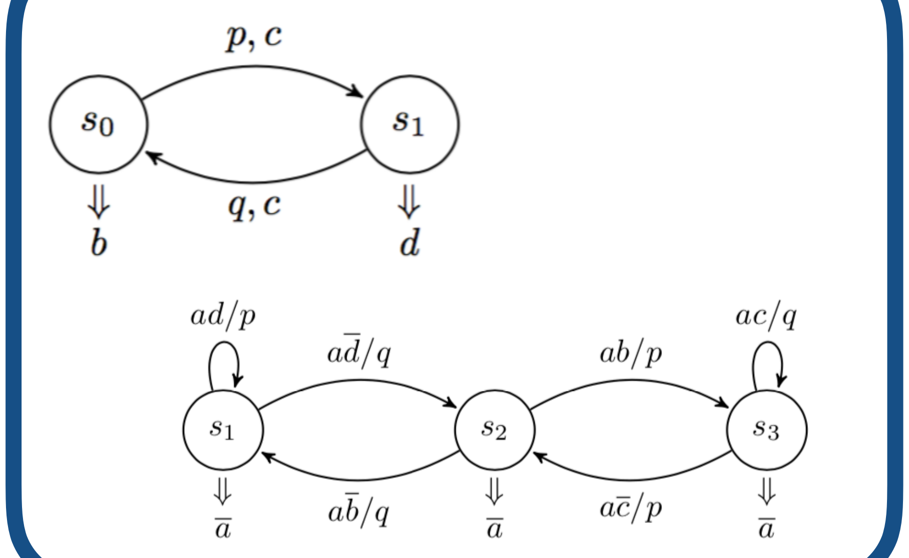
$\mathtt{e}^{(\mathtt{bc})} \cdot \mathtt{e}^{(\mathtt{c})}$

$(\mathtt{e1} +_{\mathtt{b}} \mathtt{e2}) \cdot \mathtt{f}$

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

Kleene Theorem

Automata



Programs

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

$(e1 +_b e2) \cdot f$

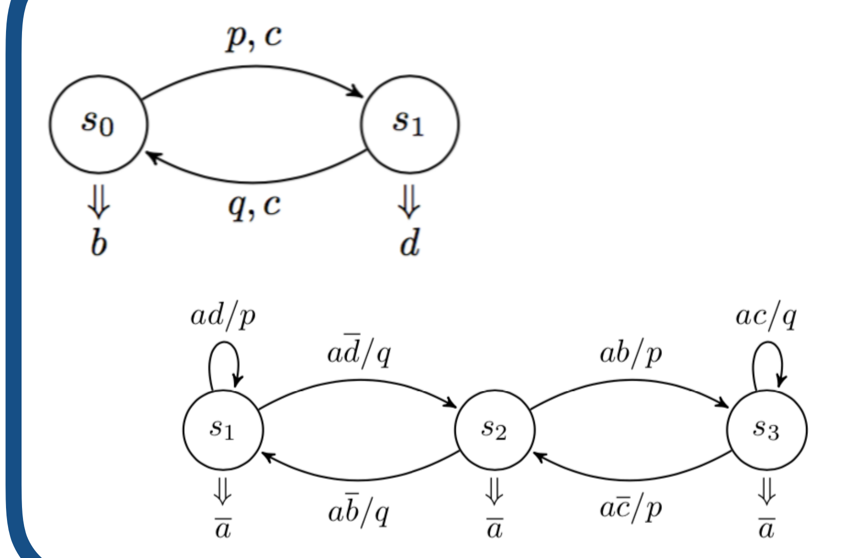
$(e^{(b)} \cdot f)^{(c)}$

check bisimilarity $A_e \sim A_f$

$\mathbf{e} \equiv \mathbf{f}$

Kleene Theorem

Automata



⟷

Programs

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

$(e1 +_b e2) \cdot f$

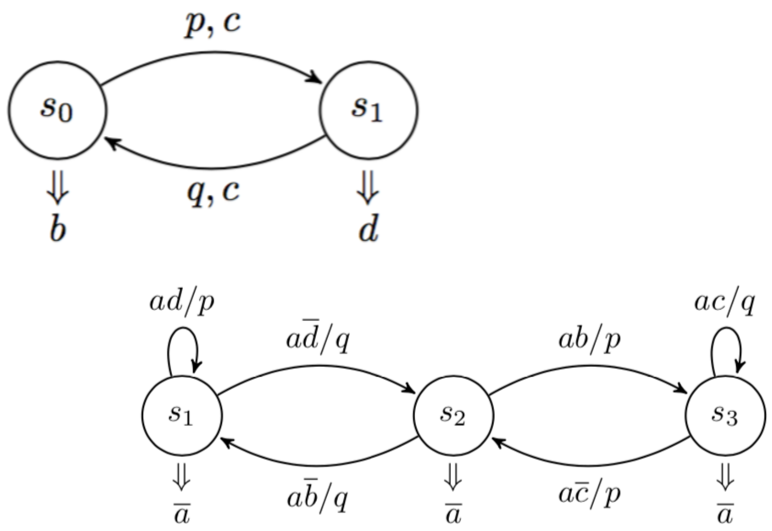
$(e^{(b)} \cdot f)^{(c)}$

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

$A_1 \sim A_2$ → **$e_1 \equiv e_2$**

Kleene Theorem

Automata



Programs

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

$(e1 +_b e2) \cdot f$

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

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

+

$A_1 \sim A_2$ **Completeness** $\mathbf{e_1 \equiv e_2}$

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

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

Semantics = Guarded Language

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

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

Automata

(Un)Successful termination

State of program

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

Semantics = Guarded Language

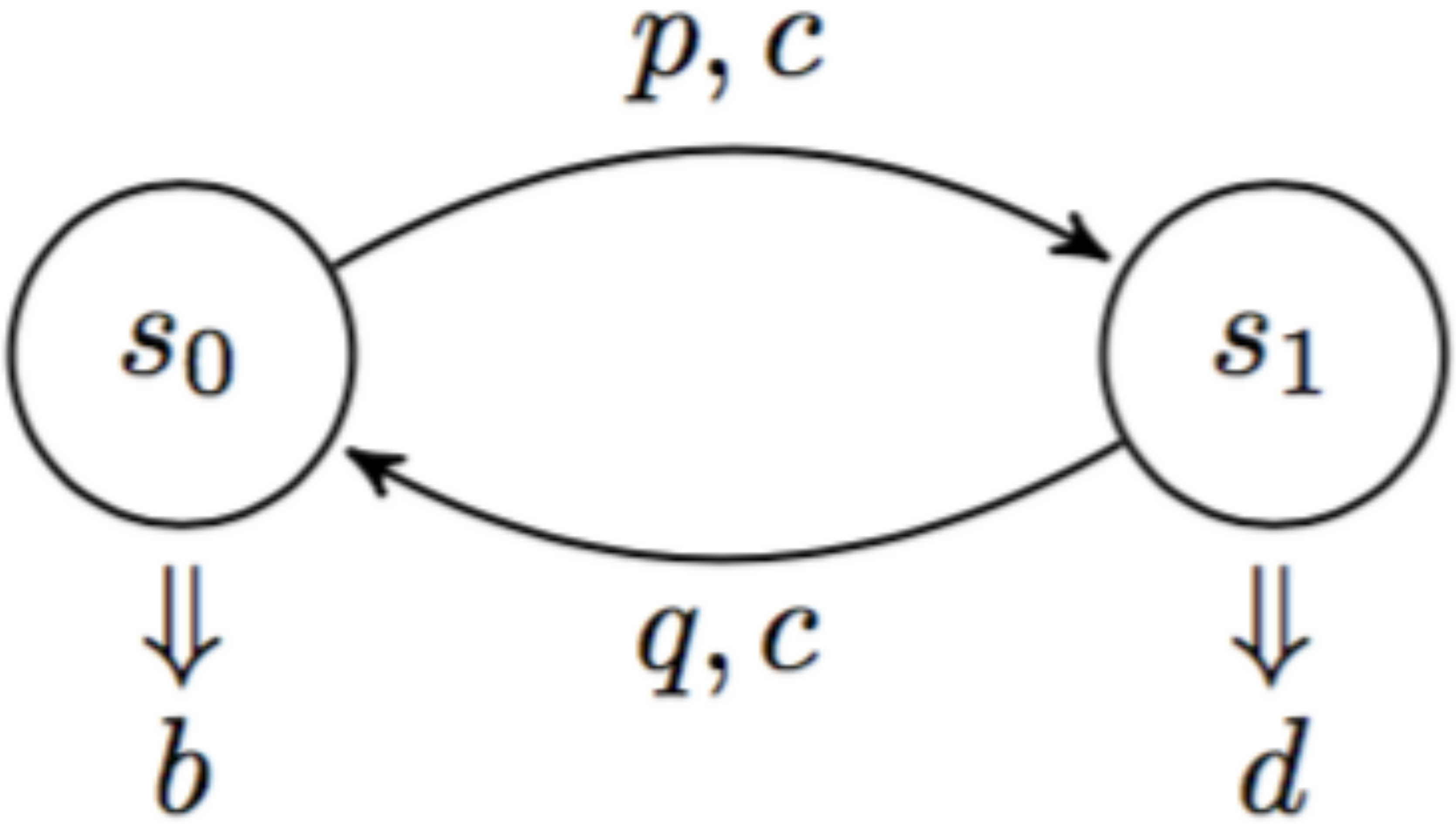
$L(s) \subseteq \mathrm{Atom} \cdot (\mathrm{Action} \cdot \mathrm{Atom})^*$

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

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

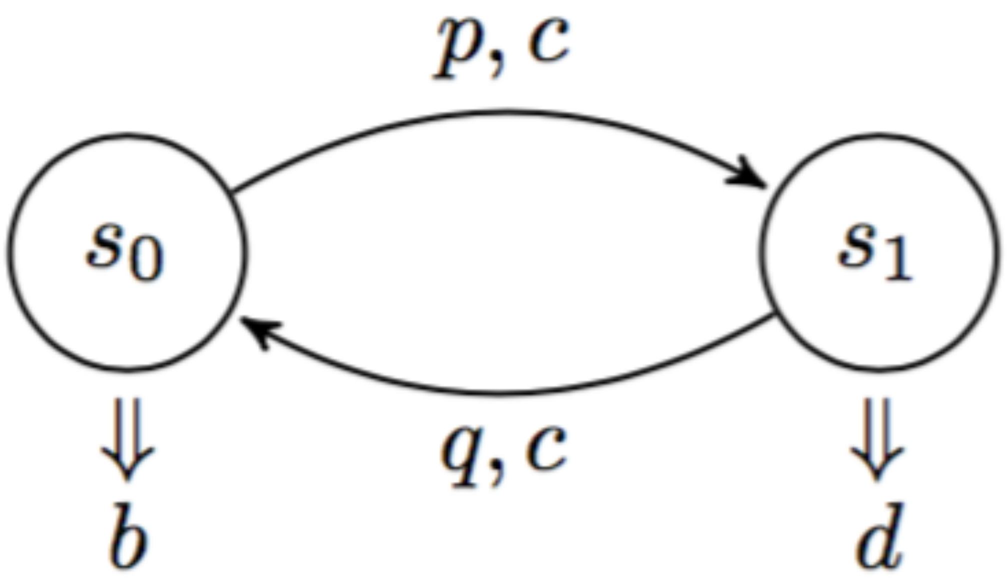
Challenge

Not all automata correspond to a GKAT program!



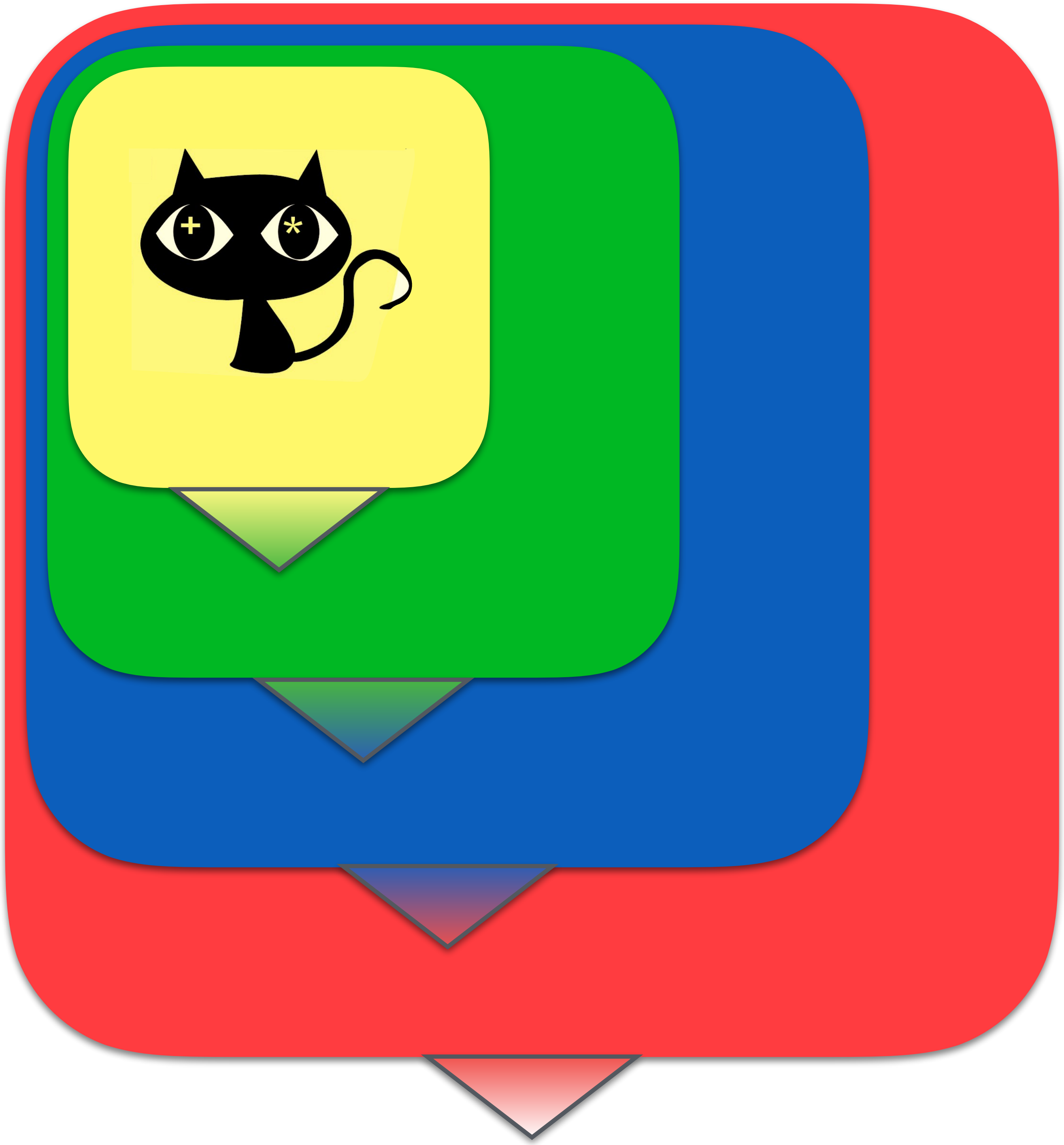
Challenge

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

Uniformity

Pseudo State

Call an arbitrary element h of G X a *pseudo state:*

$$\forall \alpha.\ h(\alpha) \in (2 + \Sigma \times X)$$

Uniformity

Pseudo State

Call an arbitrary element h of G X a *pseudo state:*

$$\forall \alpha.\ h(\alpha) \in (2 + \Sigma \times X)$$

Uniform Extension

Given⟨X, δ⟩, the *uniform extension* of Y by h is the coalgebra ⟨X, δ{h,Y}⟩ where

$$\delta\{h,Y\}(x)(\alpha) = \begin{cases} h(\alpha) & \text{if } x \in X \text{ and } \delta(x)(\alpha) = 1 \\ \delta(x)(\alpha) & \text{otherwise} \end{cases}$$

Simple Coalgebras

The set of *simple* coalgebras is defined inductively:

- If δ_X has no transitions, then $\langle X, \delta_X \rangle$ is simple
- If $\langle X, \delta_X \rangle$ and $\langle Y, \delta_y \rangle$ are simple, and $h \in G(X + Y)$, then $\langle X + Y, (\delta_X + \delta_Y)\{h, X\} \rangle$ is simple

Simple Coalgebras

The set of *simple* coalgebras is defined inductively:

- If δ_X has no transitions, then $\langle X, \delta_X \rangle$ is simple
- If $\langle X, \delta_X \rangle$ and $\langle Y, \delta_y \rangle$ are simple, and $h \in G(X + Y)$, then $\langle X + Y, (\delta_X + \delta_Y)\{h,X\} \rangle$ is simple

Theorem: Every simple coalgebra corresponds to a (well-nested) GKAT program

Thompson Construction

Thompson Construction

Expression	States X	Continuation δ	Initial ι(α)

Thompson Construction

Expression	States X	Continuation δ	Initial ι(α)
b	∅	∅	$[\alpha \leq b]$

Thompson Construction

Expression	States X	Continuation δ	Initial ι(α)
b	$\varnothing$	$\varnothing$	$[\alpha \leq b]$
p	$\{*\}$	$* \mapsto 1$	$(p, *)$

Thompson Construction

Expression	States X	Continuation δ	Initial ι(α)
b	$\varnothing$	$\varnothing$	$[\alpha \le b]$
p	$\{*\}$	$* \mapsto 1$	$(p, *)$
$e \;\; +_b \;\; f$	$X_e + X_f$	$\delta_e + \delta_f$	$l_e\,\alpha \le b$ $l_e\,\alpha \le\, !b$

Thompson Construction

Expression	States X	Continuation δ	Initial ι(α)
b	$\varnothing$	$\varnothing$	$[\alpha \leq b]$
p	$\{*\}$	$* \mapsto 1$	$(p, *)$
$e \ +_b \ f$	$X_e + X_f$	$\delta_e + \delta_f$	$\iota_e \, \alpha \leq b$ $\iota_e \, \alpha \leq !b$
$e \ \cdot \ f$	$X_e + X_f$	$(\delta_e + \delta_f)\{\iota_f, X_e\}$	ι_f if $\alpha = 1$ ι_e if $\alpha \neq 1$

Thompson Construction

Expression	States X	Continuation δ	Initial ι(α)
b	$\varnothing$	$\varnothing$	$[\alpha \leq b]$
p	$\{*\}$	$* \mapsto 1$	$(p, *)$
$e \; +_b \; f$	$X_e + X_f$	$\delta_e + \delta_f$	$\iota_e \; \alpha \leq b$ $\iota_e \; \alpha \leq !b$
$e \; \cdot \; f$	$X_e + X_f$	$(\delta_e + \delta_f)\{\iota_f, X_e\}$	ι_f if $\alpha = 1$ ι_e if $\alpha \neq 1$
e^c	X_e	$(\delta_e)\{\iota_e, X_e\}$	1 if $\alpha \leq !c$ 0 if $\alpha \leq c \; \iota_e(\alpha) = 1$ $\iota_e(\alpha)$ if $\alpha \leq c \; \iota_e(\alpha) \neq 1$

Thompson Construction

Expression	States X	Continuation δ	Initial ι(α)
b	∅	∅	$[\alpha \leq b]$
e +			
e			ι_e if $\alpha \neq 1$
e^c	X_e	$(\delta_e)\{\iota_e, X_e\}$	1 if $\alpha \leq\, !c$ 0 if $\alpha \leq c\ \iota_e(\alpha) = 1$ $\iota_e(\alpha)$ if $\alpha \leq c\ \iota_e(\alpha) \neq 1$

Theorem: The coalgebra generated by the Thompson construction, namely,

$$\langle \{\iota\} + X_e, \delta_{X_e} \cup \{ \iota \mapsto \iota_e \} \rangle$$

is simple

Wrapping Up

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

Thank you!

Thank you!