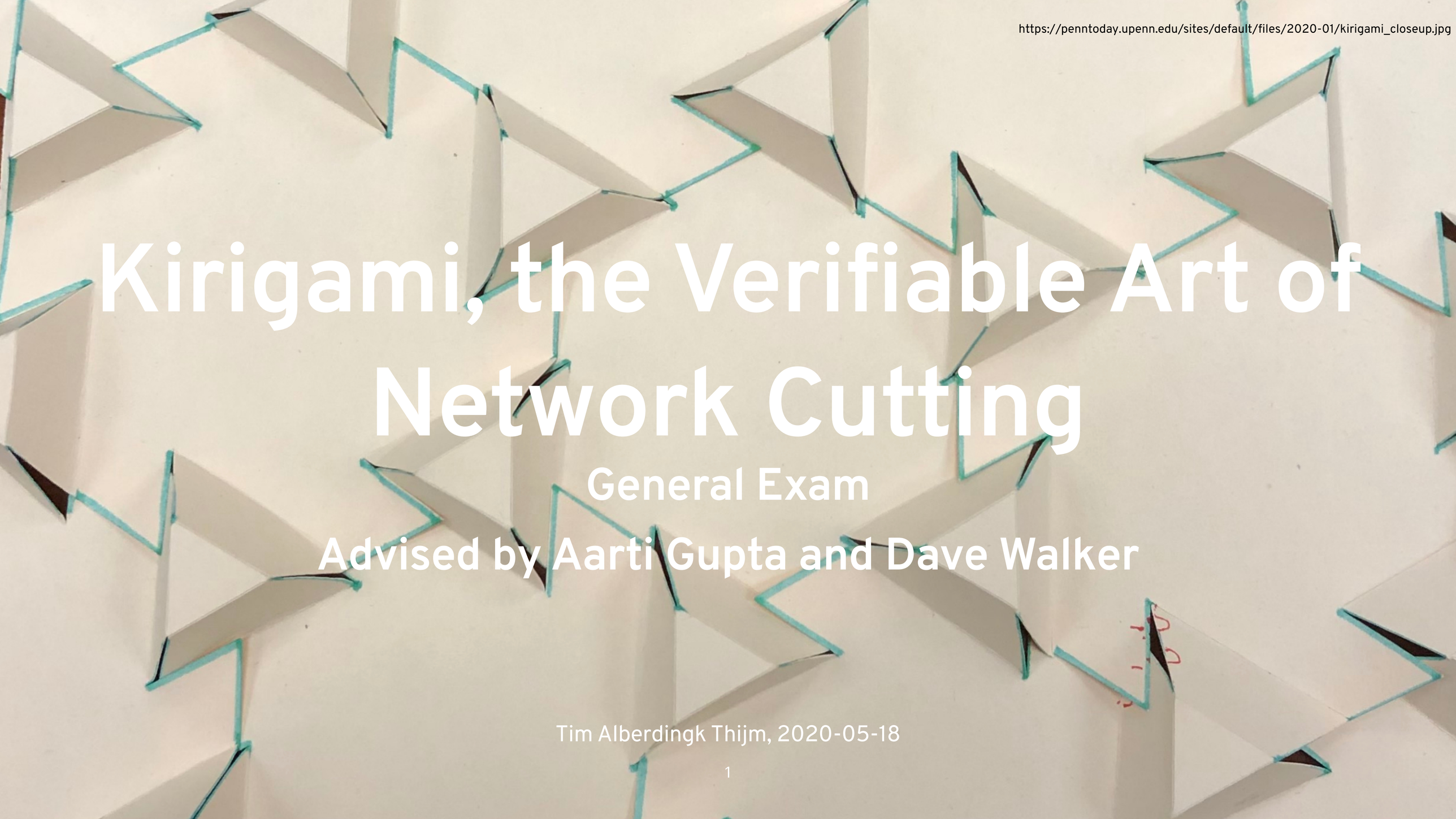


https://penntoday.upenn.edu/sites/default/files/2020-01/kirigami_closeup.jpg

Kirigami, the Verifiable Art of Network Cutting

General Exam

Advised by Aarti Gupta and Dave Walker

Tim Alberdingk Thijm, 2020-05-18

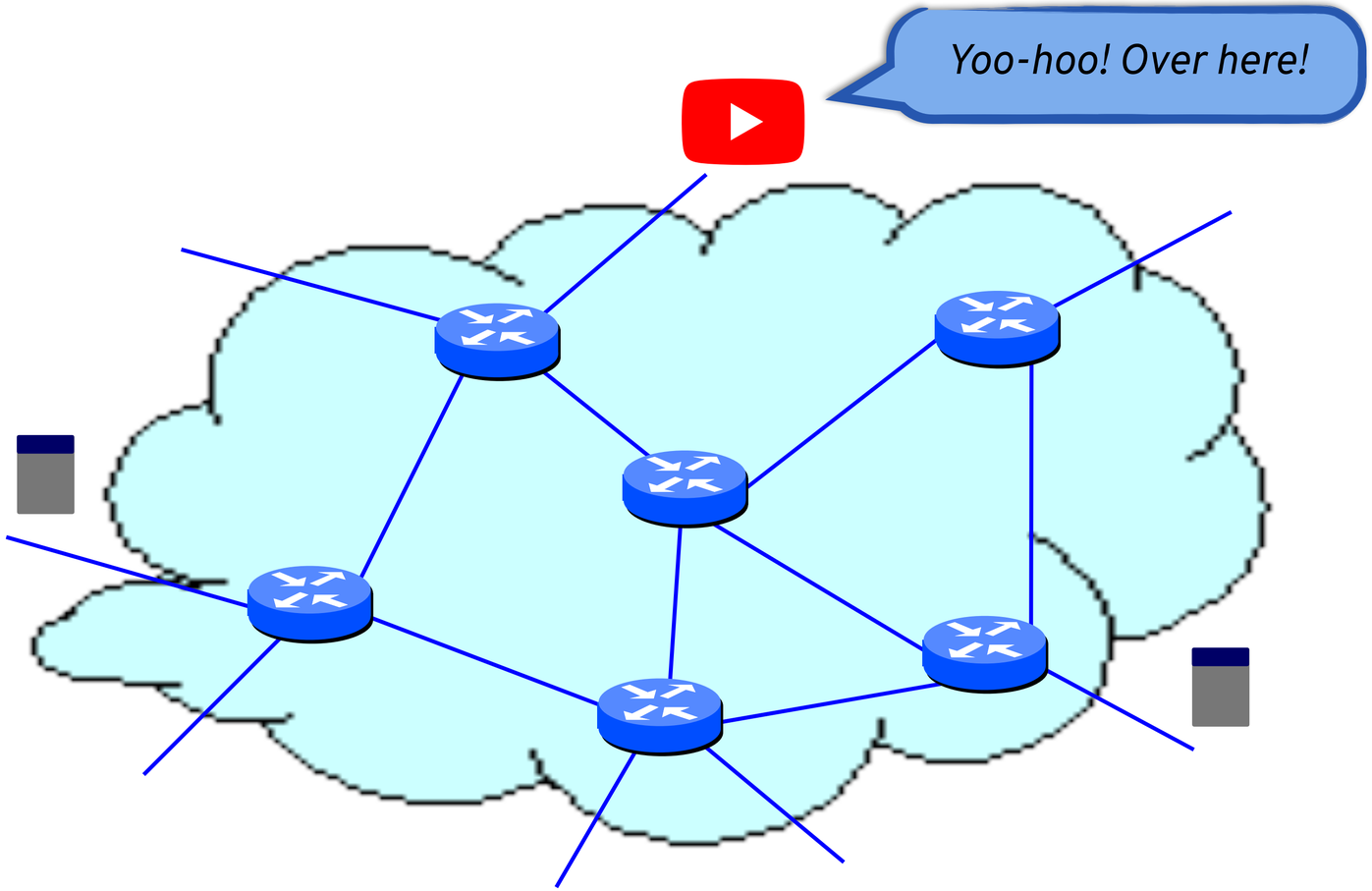
Network Routing

A (Simplified) Control Plane Protocol

Some images from Jen Rexford's COS561 slides

Network Routing

A (Simplified) Control Plane Protocol



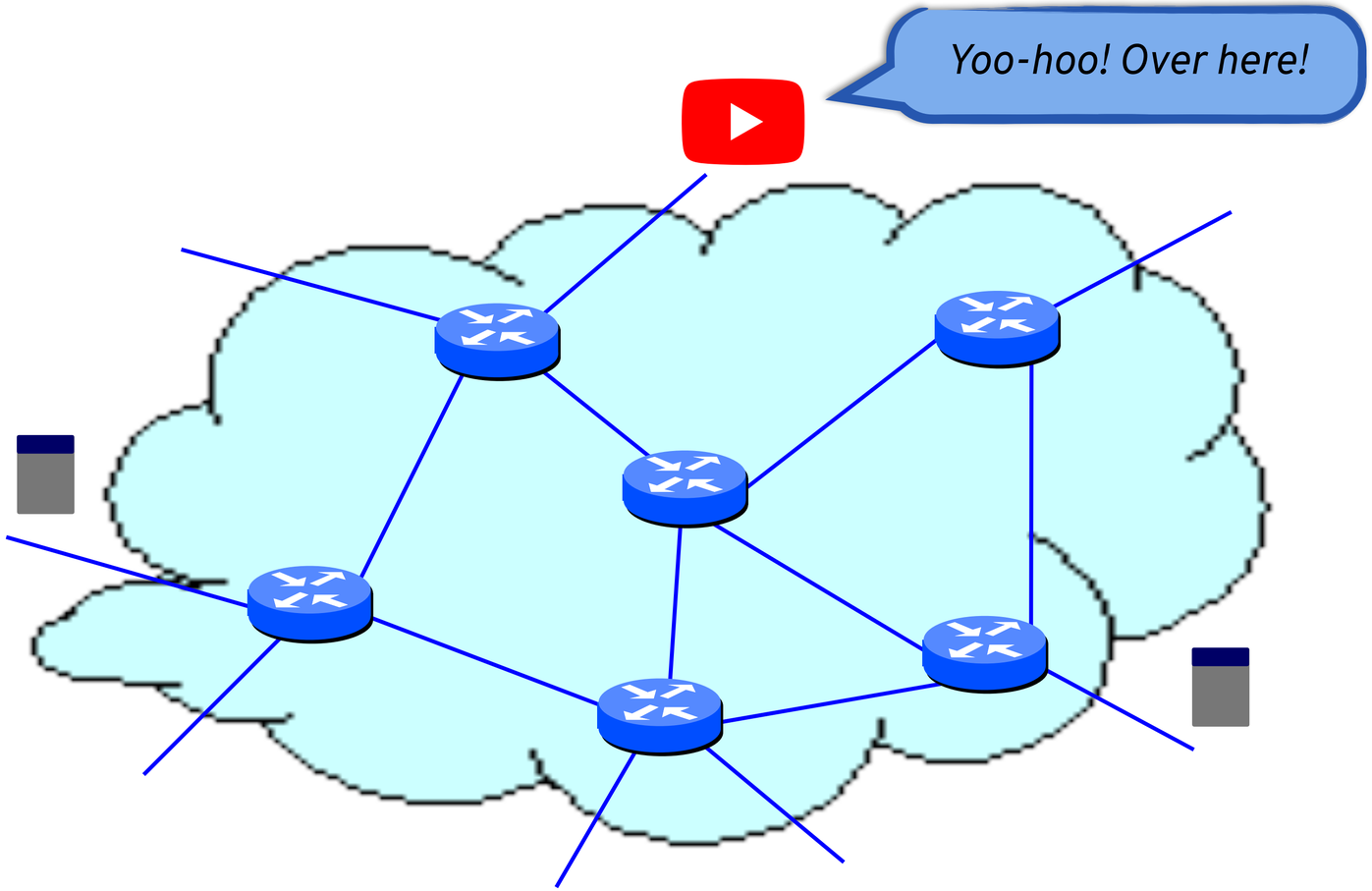
Some images from Jen Rexford's COS561 slides

Network Routing

A (Simplified) Control Plane Protocol

Routers can:

- *Send messages*
- *Receive messages*
- *Compare routes by incentives*



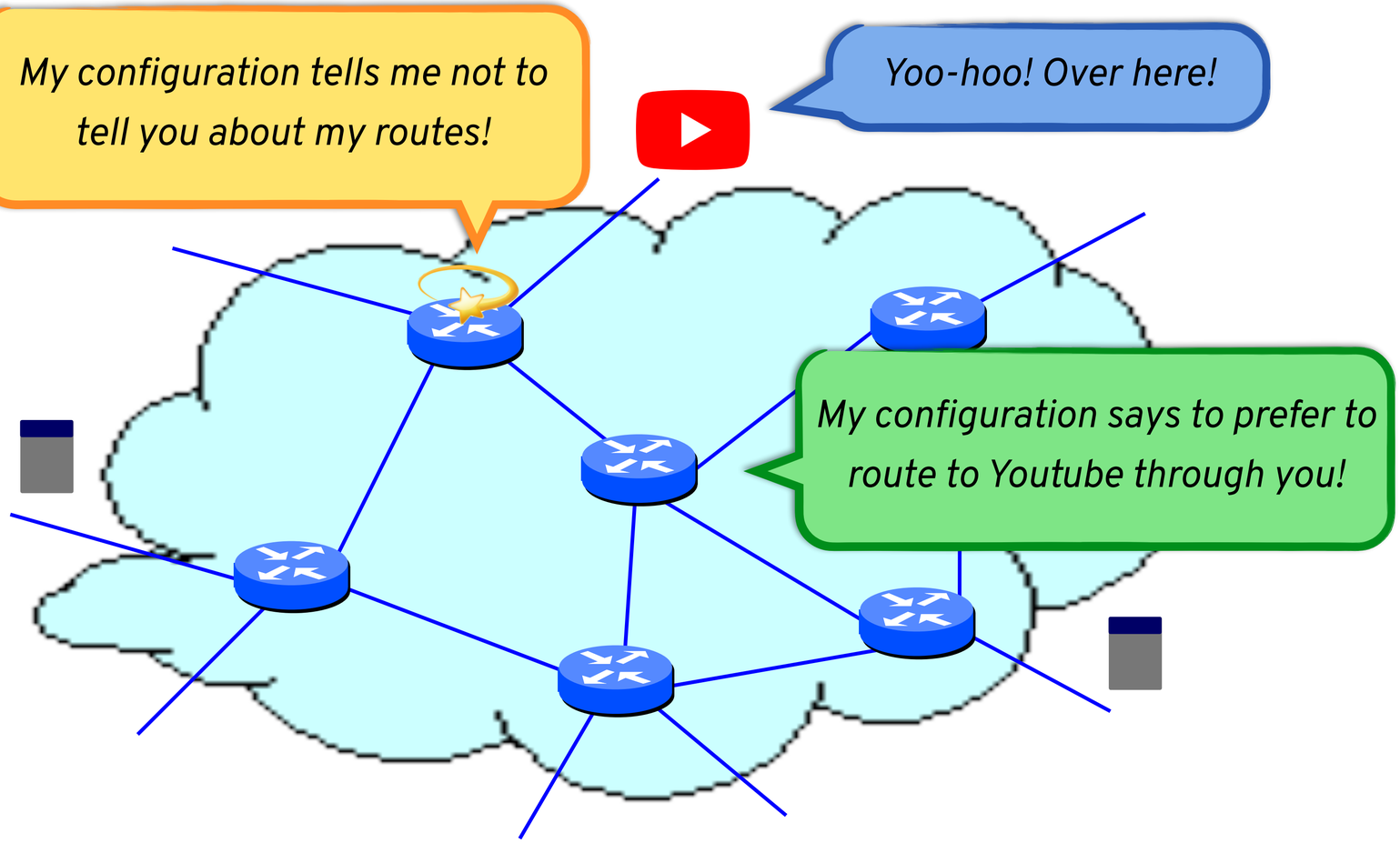
Some images from Jen Rexford's COS561 slides

Network Routing

A (Simplified) Control Plane Protocol

Routers can:

- *Send messages*
- *Receive messages*
- *Compare routes by incentives*



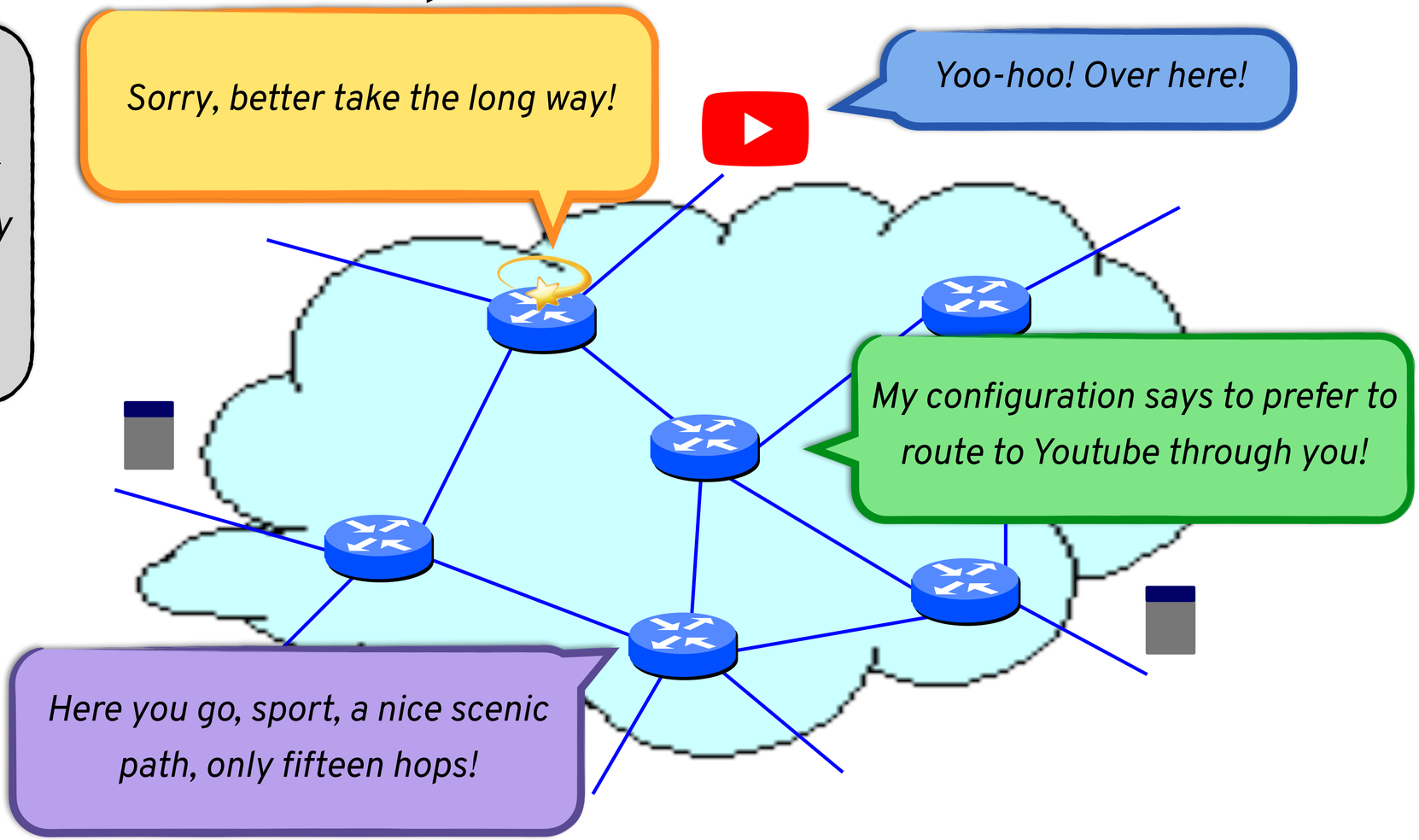
Some images from Jen Rexford's COS561 slides

Network Routing

A (Simplified) Control Plane Protocol

Routers can:

- *Send messages*
- *Receive messages*
- *Compare routes by incentives*



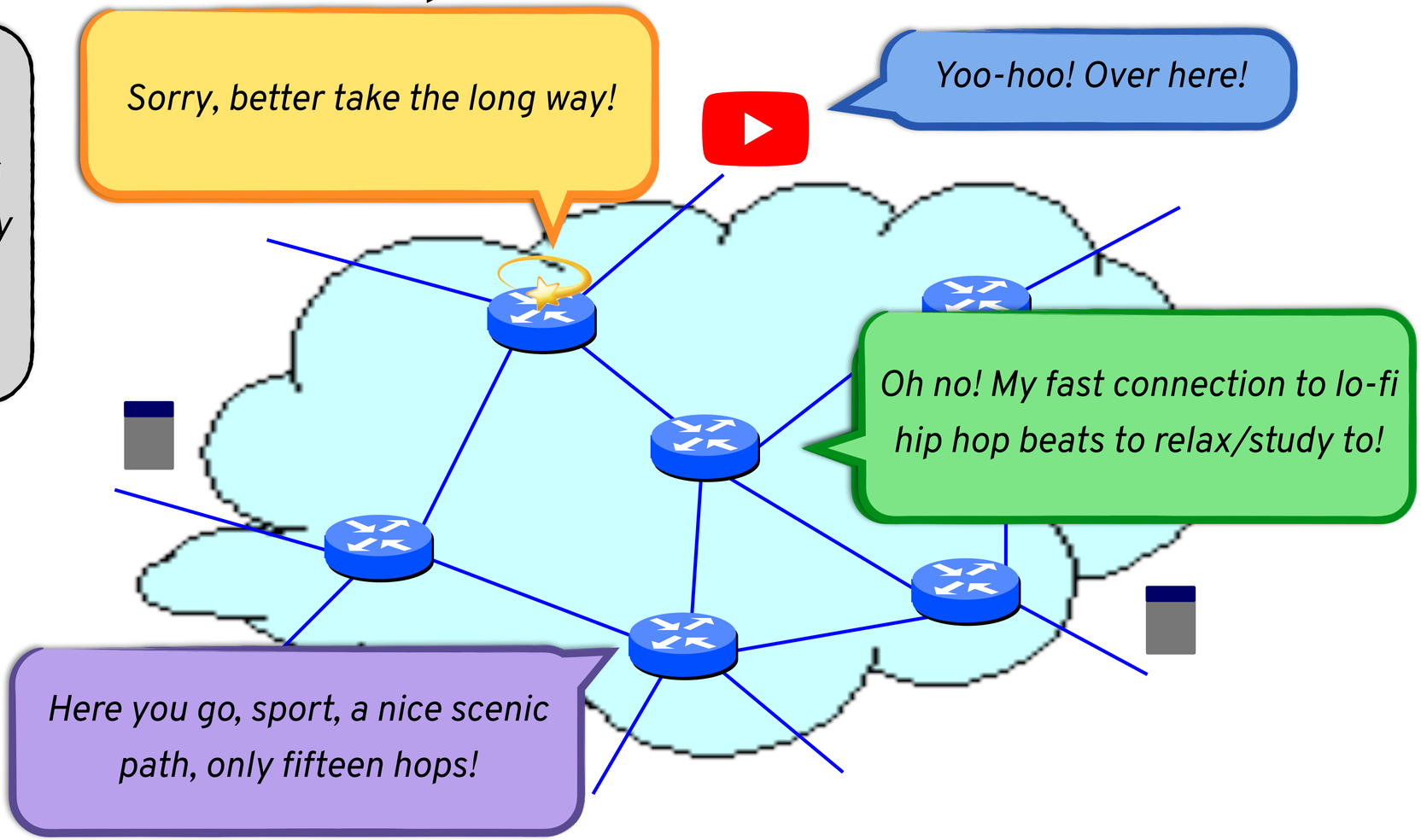
Some images from Jen Rexford's COS561 slides

Network Routing

A (Simplified) Control Plane Protocol

Routers can:

- *Send messages*
- *Receive messages*
- *Compare routes by incentives*
- *Have bugs*



Some images from Jen Rexford's COS561 slides

Network Routing

A (Simplified) Control Plane Protocol

Routers can:

- *Send messages*
- *Receive messages*
- *Compare routes by incentives*
- *Have bugs*

Some images from Jen Rexford's COS561 slides

Solver-Aided Network Verification

In A Nutshell

[Beckett et al., SIGCOMM 2017]

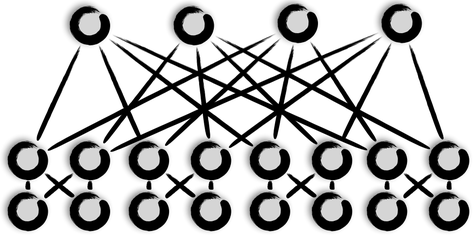
[Gember-Jacobson et al., SIGCOMM 2016]

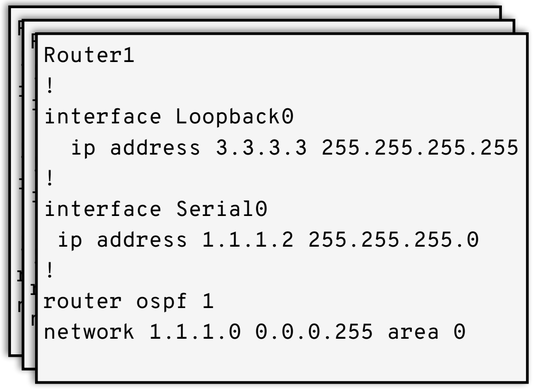
[Anderson et al., SIGPLAN 2014]

[Mai et al., SIGCOMM 2011]

Solver-Aided Network Verification

In A Nutshell





[Beckett et al., SIGCOMM 2017]

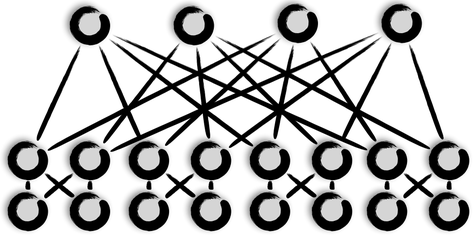
[Gember-Jacobson et al., SIGCOMM 2016]

[Anderson et al., SIGPLAN 2014]

[Mai et al., SIGCOMM 2011]

Solver-Aided Network Verification

In A Nutshell



```
Router1
!
interface Loopback0
  ip address 3.3.3.3 255.255.255.255
!
interface Serial0
 ip address 1.1.1.2 255.255.255.0
!
router ospf 1
network 1.1.1.0 0.0.0.255 area 0
```

$$\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$$

[Beckett et al., SIGCOMM 2017]

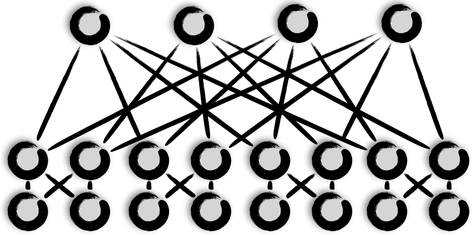
[Gember-Jacobson et al., SIGCOMM 2016]

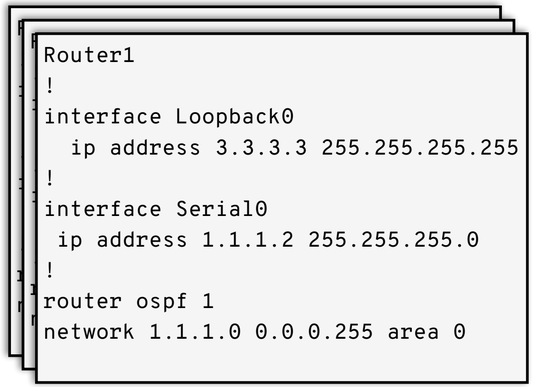
[Anderson et al., SIGPLAN 2014]

[Mai et al., SIGCOMM 2011]

Solver-Aided Network Verification

In A Nutshell





P

$\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$

[Beckett et al., SIGCOMM 2017]

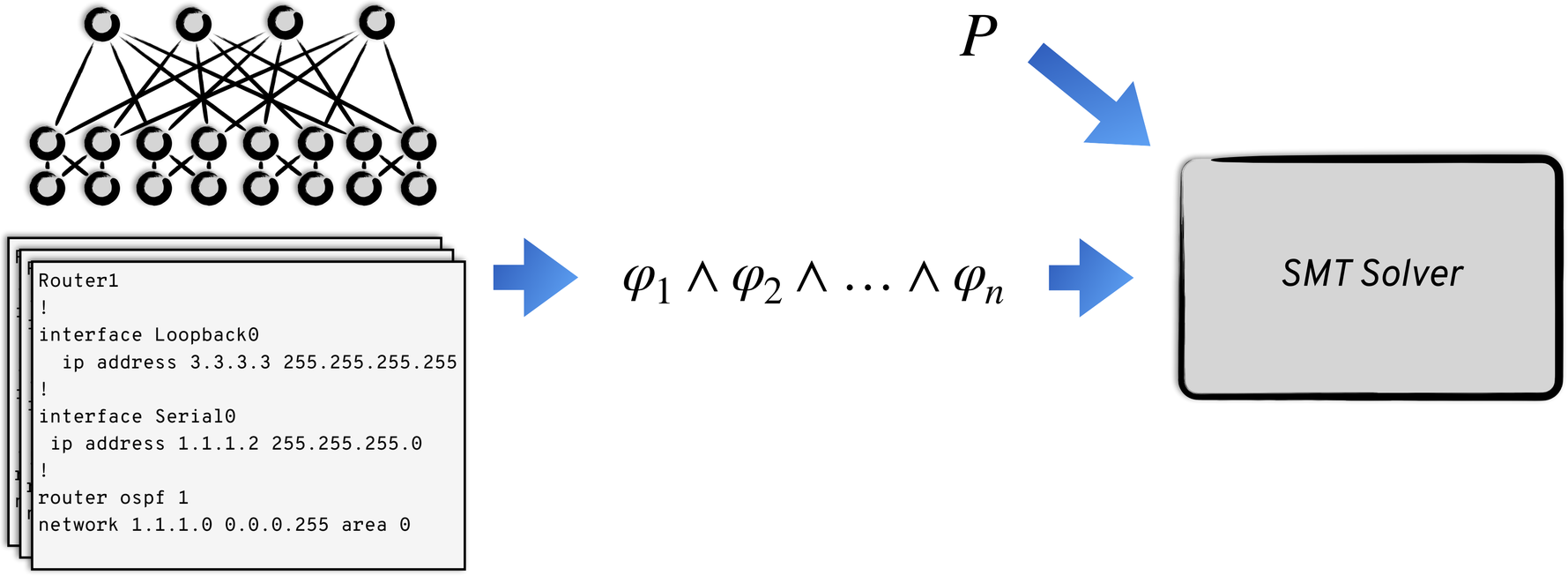
[Gember-Jacobson et al., SIGCOMM 2016]

[Anderson et al., SIGPLAN 2014]

[Mai et al., SIGCOMM 2011]

Solver-Aided Network Verification

In A Nutshell



[Beckett et al., SIGCOMM 2017]

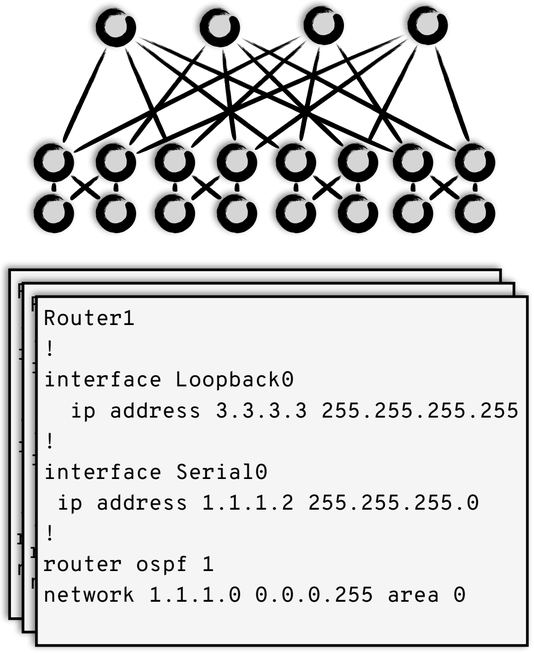
[Gember-Jacobson et al., SIGCOMM 2016]

[Anderson et al., SIGPLAN 2014]

[Mai et al., SIGCOMM 2011]

Solver-Aided Network Verification

In A Nutshell



```
Router1
!
interface Loopback0
  ip address 3.3.3.3 255.255.255.255
!
interface Serial0
 ip address 1.1.1.2 255.255.255.0
!
router ospf 1
network 1.1.1.0 0.0.0.255 area 0
```

$\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$

P

SMT Solver

Verified ✓

[Beckett et al., SIGCOMM 2017]

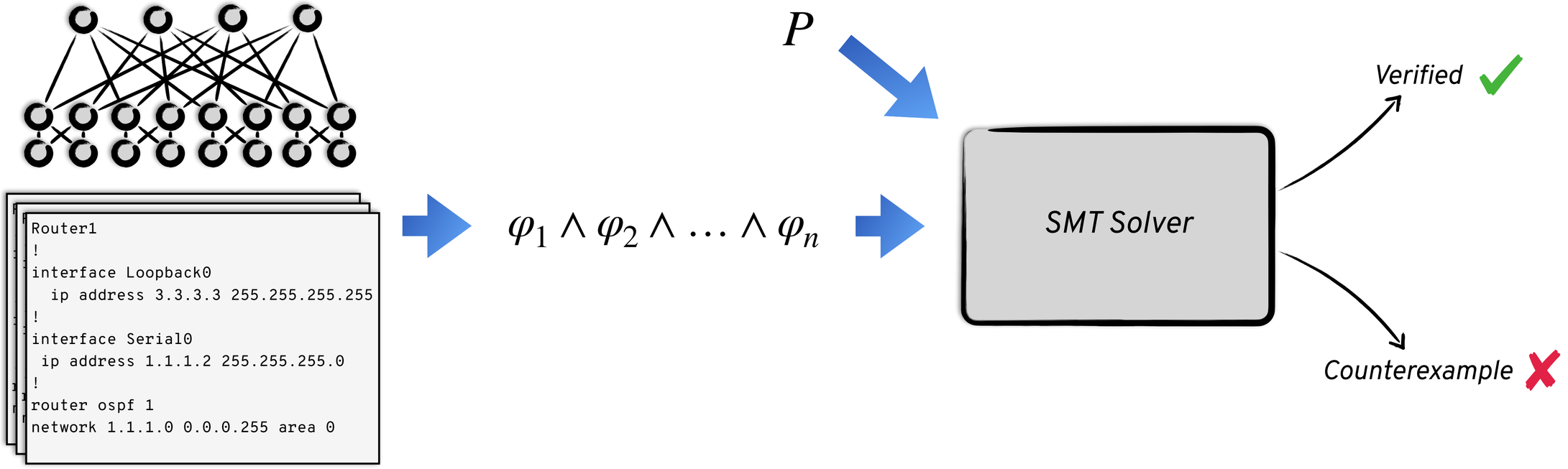
[Gember-Jacobson et al., SIGCOMM 2016]

[Anderson et al., SIGPLAN 2014]

[Mai et al., SIGCOMM 2011]

Solver-Aided Network Verification

In A Nutshell



[Beckett et al., SIGCOMM 2017]

[Gember-Jacobson et al., SIGCOMM 2016]

[Anderson et al., SIGPLAN 2014]

[Mai et al., SIGCOMM 2011]

SMT Verification Scalability

[Beckett et al., SIGCOMM 2017]

SMT Verification Scalability

- # SMT Variables $\propto$ Network Size

[Beckett et al., SIGCOMM 2017]

SMT Verification Scalability

- # SMT Variables $\propto$ Network Size
- Time to check certain properties can be exponential in the number of variables!

[Beckett et al., SIGCOMM 2017]

SMT Verification Scalability

- # SMT Variables ∝ Network Size
- Time to check certain properties can be exponential in the number of variables!

(lower is better)

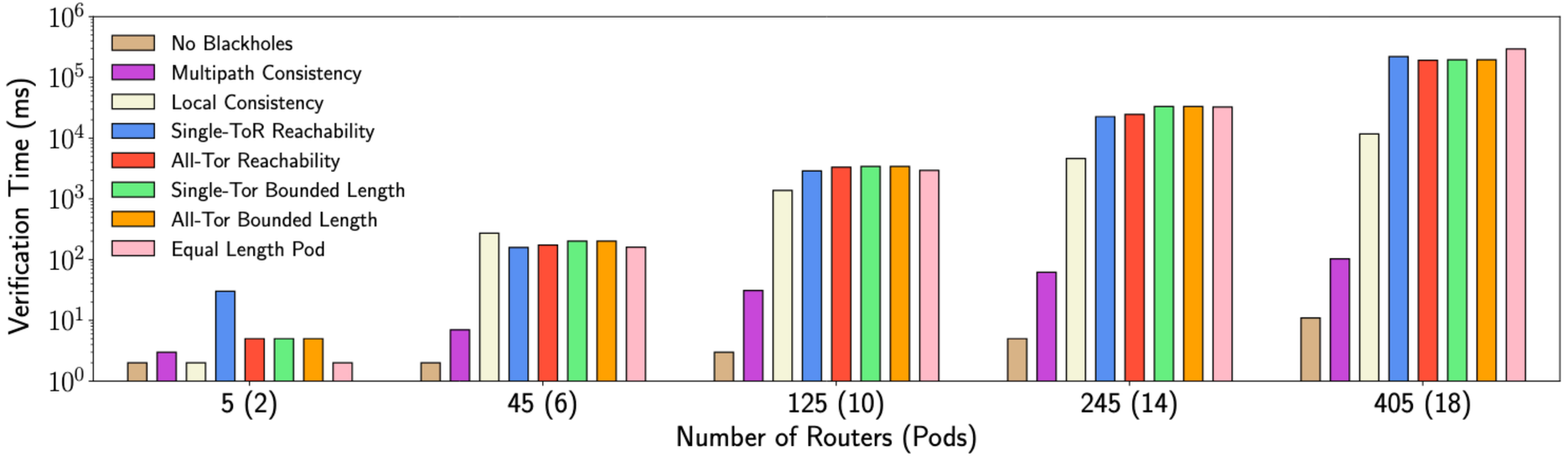


Figure 8: Verification time for synthetic configurations for different properties and network sizes.

[Beckett et al., SIGCOMM 2017]

SMT Verification Scalability

- # SMT Variables ∝ Network Size
- Time to check certain properties can be exponential in the number of variables!

(lower is better)

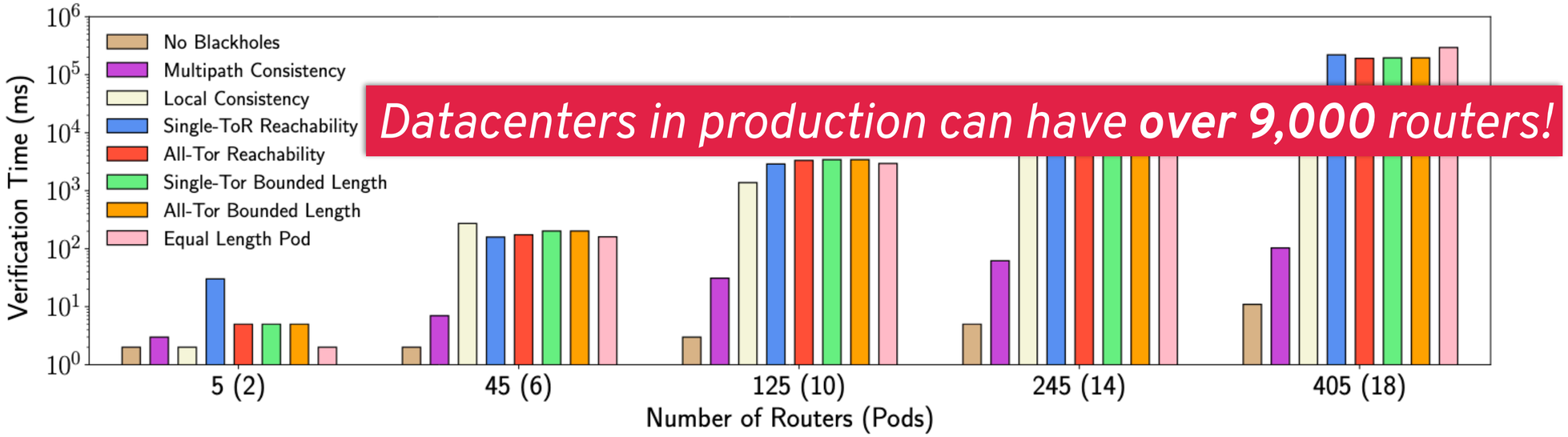
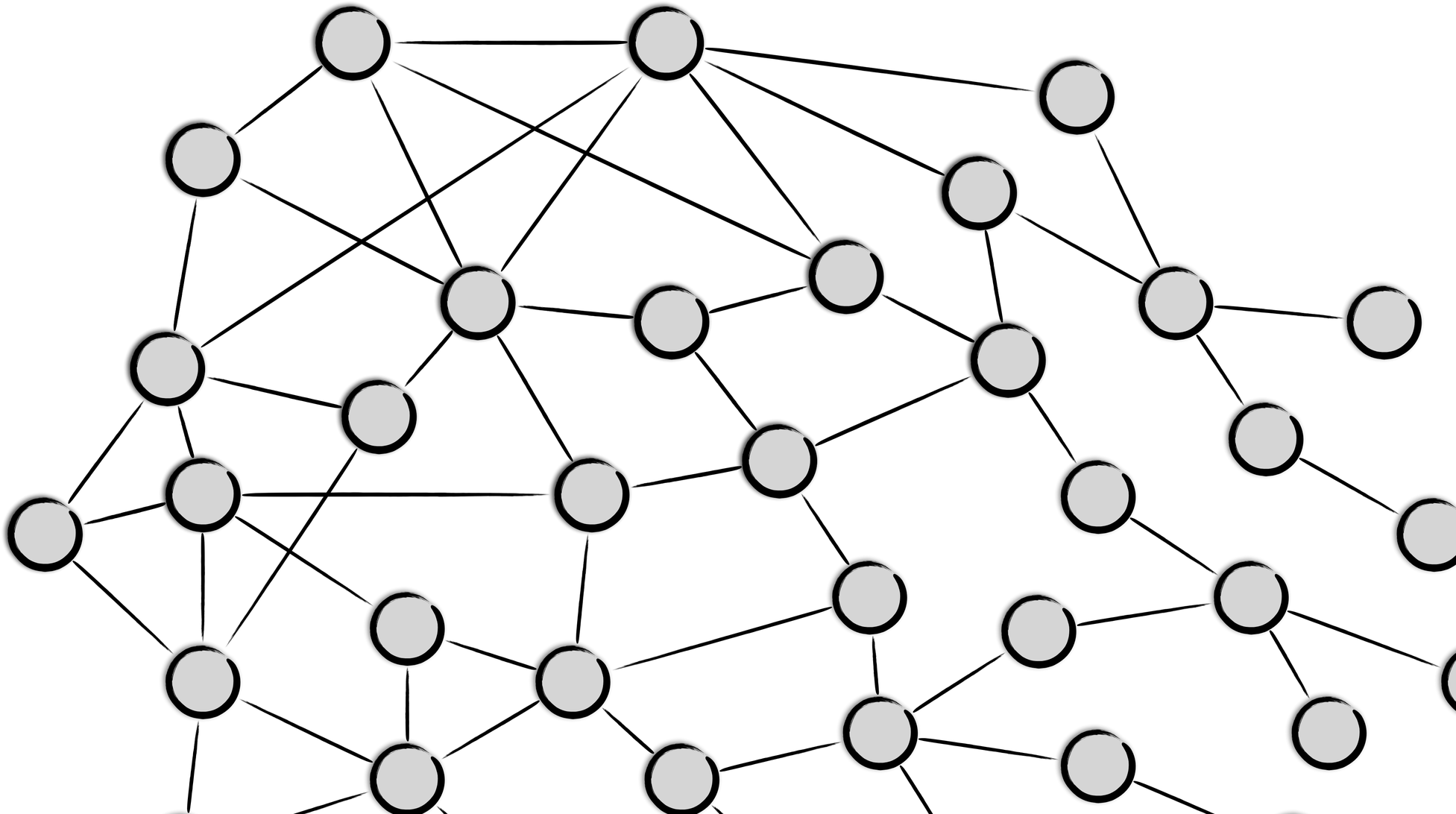


Figure 8: Verification time for synthetic configurations for different properties and network sizes.

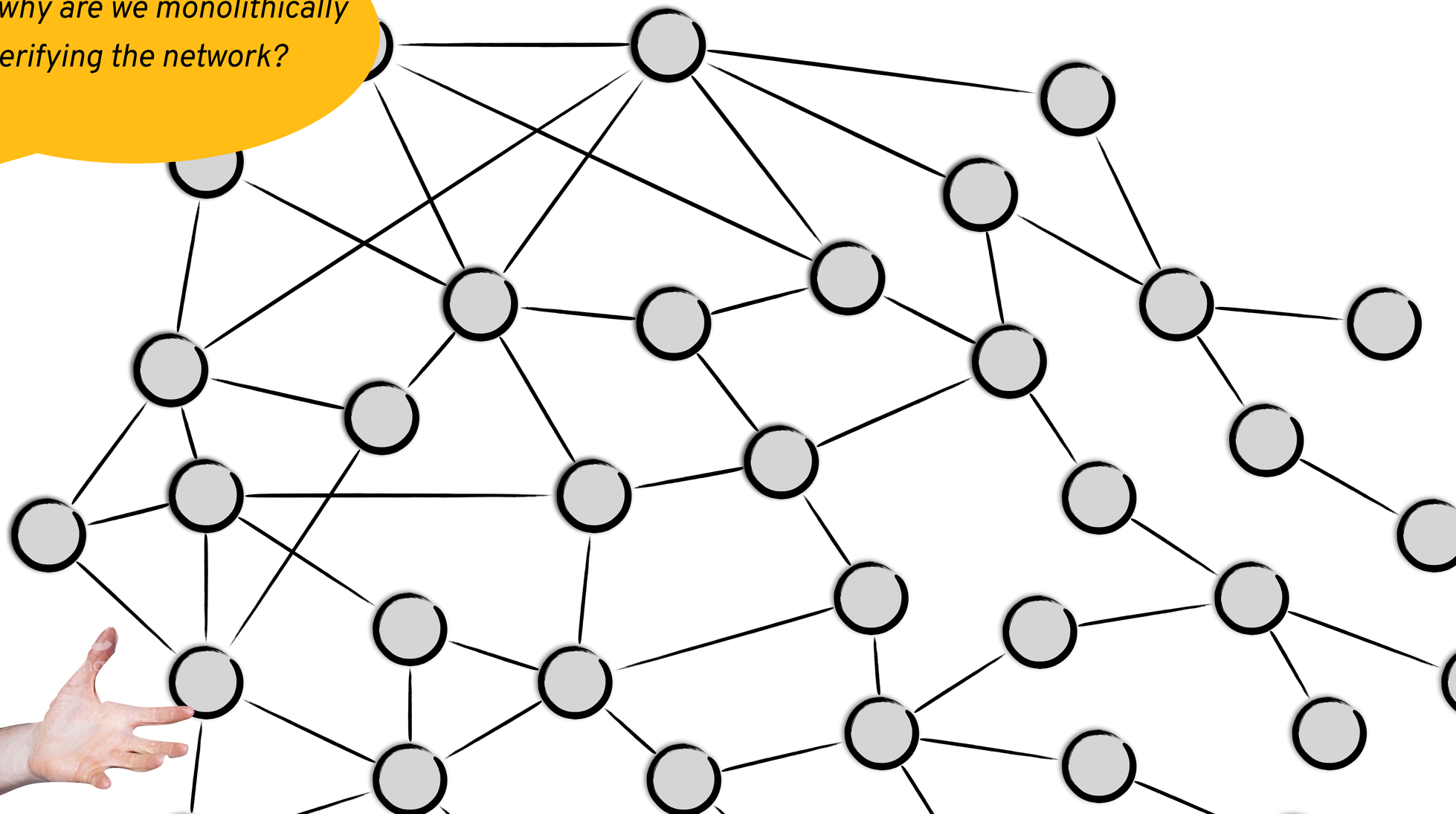
[Beckett et al., SIGCOMM 2017]

Is All Hope Lost?

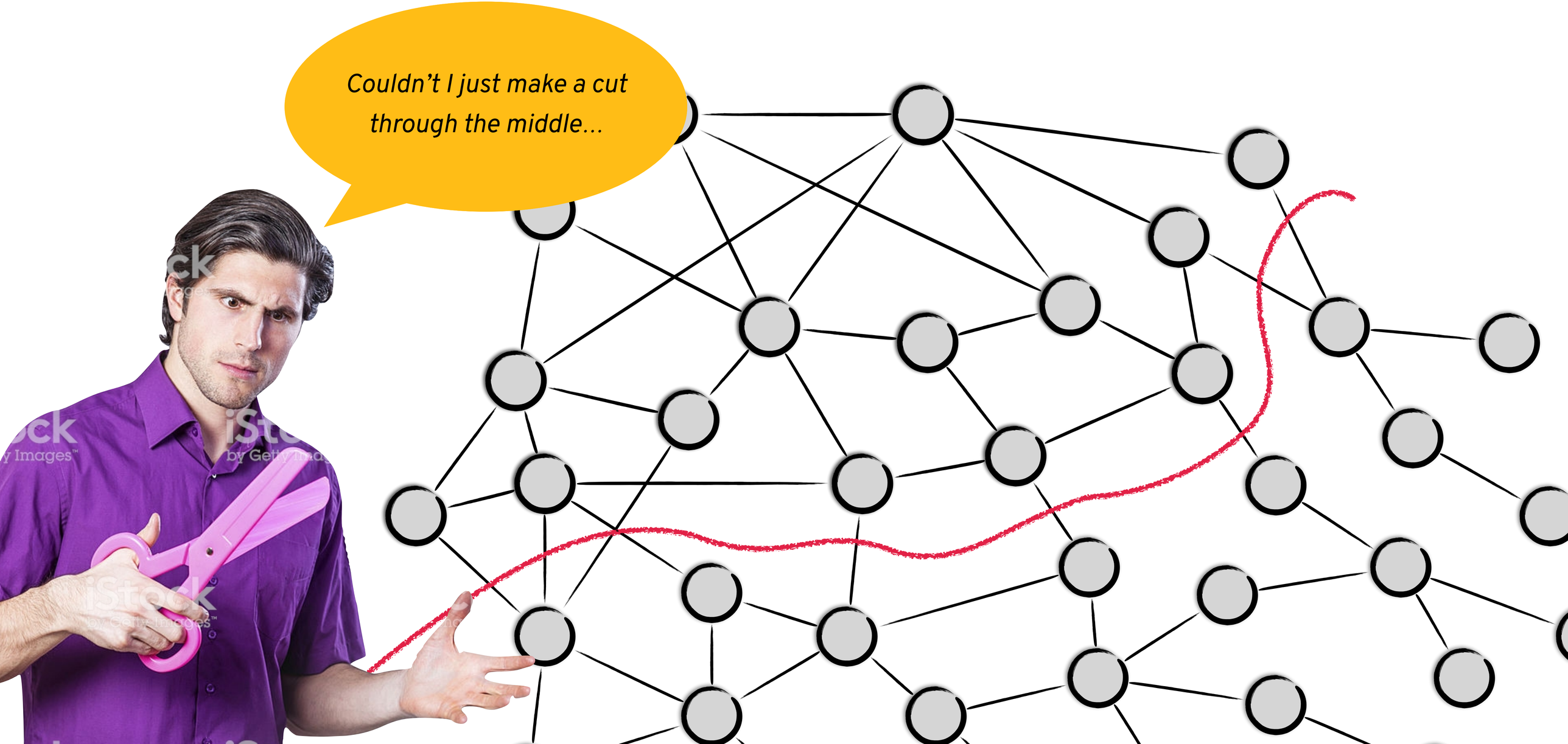


Is All Hope Lost?

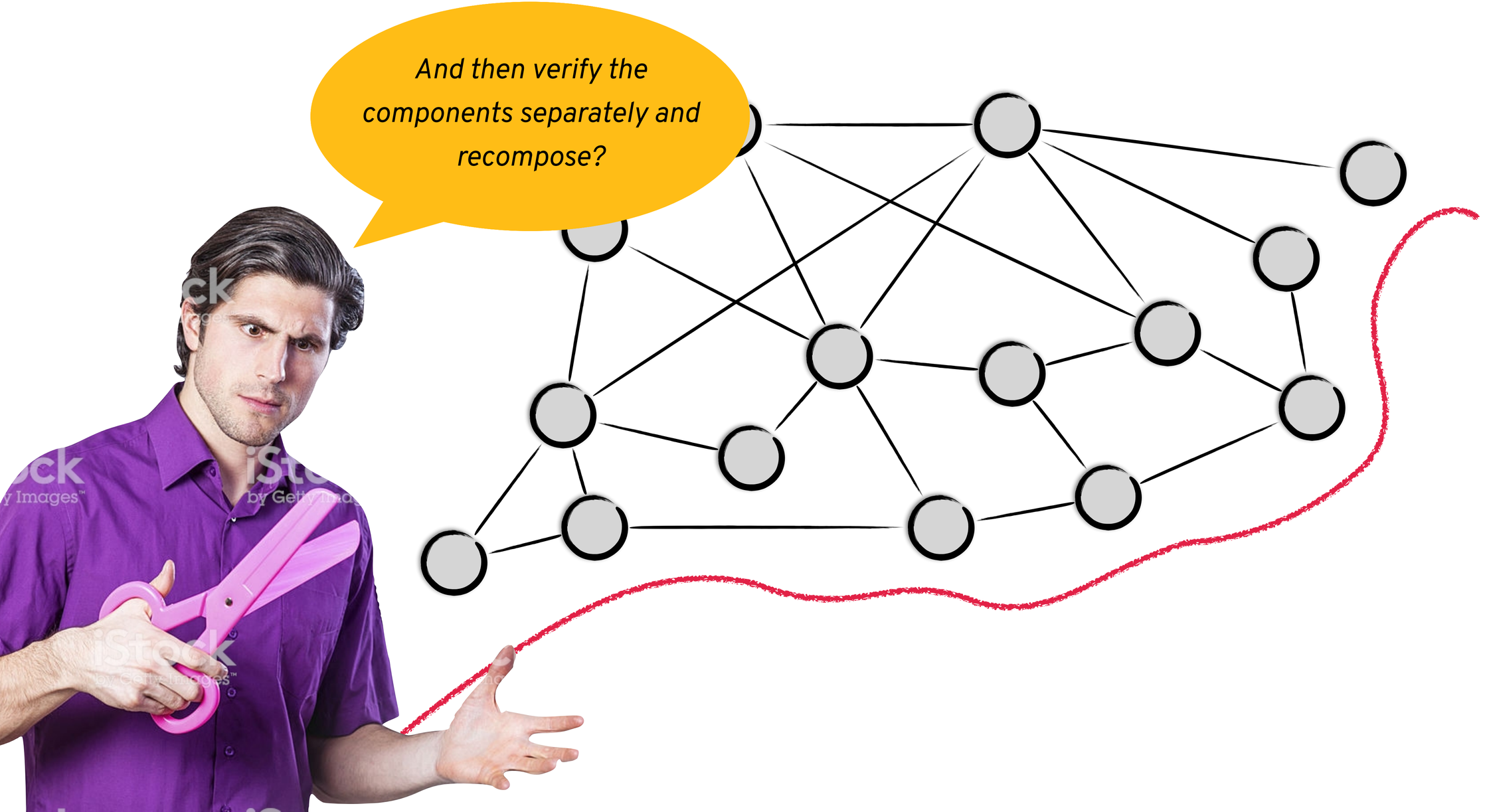
Wait, why are we monolithically verifying the network?



Is All Hope Lost?



Is All Hope Lost?



In other words…

Can we use the inherent *compositional design* of networks to *verify them compositionally*?

In other words…

Can we

co

rks

to

tionally?

Compositional Verification: verify each component separately, such that correctness of the monolithic system is guaranteed!

Roadmap

Opening up the Stable Routing Problem

The Kirigami Algorithm

Implementing Kirigami in NV

Results and Future Directions

Opening up the Stable Routing Problem

The Stable Routing Problem (SRP)

(Prior Work)

[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$R = (\quad , \quad , \quad , \quad , \quad)$$

[Beckett et al., SIGCOMM 2018]

[Griffin and Sobrinho, SIGCOMM 2005]

[Sobrinho, IEANEP 2005]

[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

topology of network

$$R = (G, \quad , \quad , \quad , \quad)$$

[Beckett et al., SIGCOMM 2018]

[Griffin and Sobrinho, SIGCOMM 2005]

[Sobrinho, IEANEP 2005]

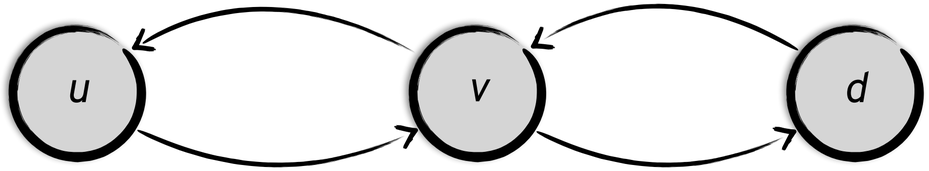
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

topology of network

$$R = (G, \quad , \quad , \quad , \quad)$$



[Beckett et al., SIGCOMM 2018]

[Griffin and Sobrinho, SIGCOMM 2005]

[Sobrinho, IEANEP 2005]

[Griffin et al., IEANEP 2002]

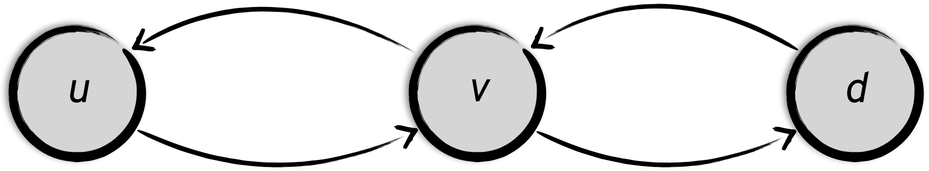
The Stable Routing Problem (SRP)

(Prior Work)

topology of network

$$R = (\ G,\ A,\quad ,\quad ,\quad)$$

set of attributes



[Beckett et al., SIGCOMM 2018]

[Griffin and Sobrinho, SIGCOMM 2005]

[Sobrinho, IEANEP 2005]

[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

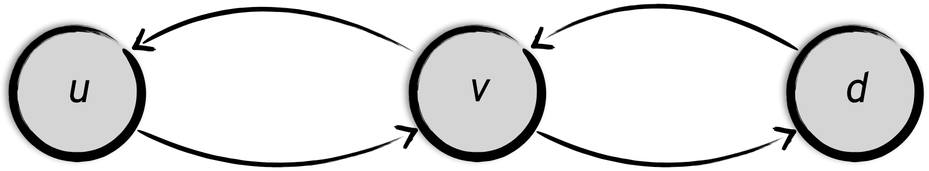
(Prior Work)

topology of network

$$R = (\ G,\ A, \mathbf{init},\quad ,\qquad\qquad)$$

set of attributes

initial attribute at each node



[Beckett et al., SIGCOMM 2018]

[Griffin and Sobrinho, SIGCOMM 2005]

[Sobrinho, IEANEP 2005]

[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

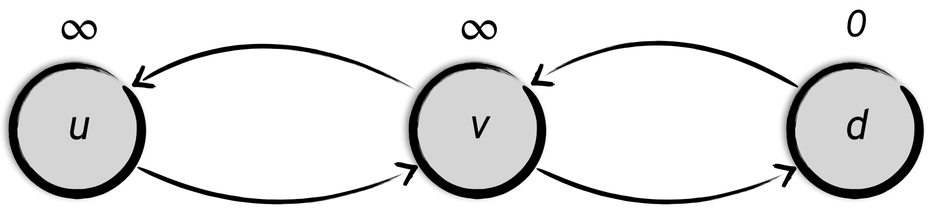
(Prior Work)

topology of network

$$R = (\ G,\ A, \mathbf{init},\quad ,\qquad\qquad)$$

set of attributes

initial attribute at each node



[Beckett et al., SIGCOMM 2018]

[Griffin and Sobrinho, SIGCOMM 2005]

[Sobrinho, IEANEP 2005]

[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

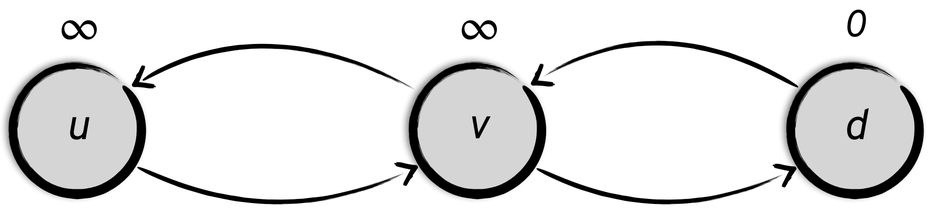
how to merge received attributes

topology of network

$$R = (G, A, \mathbf{init}, \oplus, \quad)$$

set of attributes

initial attribute at each node



[Beckett et al., SIGCOMM 2018]

[Griffin and Sobrinho, SIGCOMM 2005]

[Sobrinho, IEANEP 2005]

[Griffin et al., IEANEP 2002]

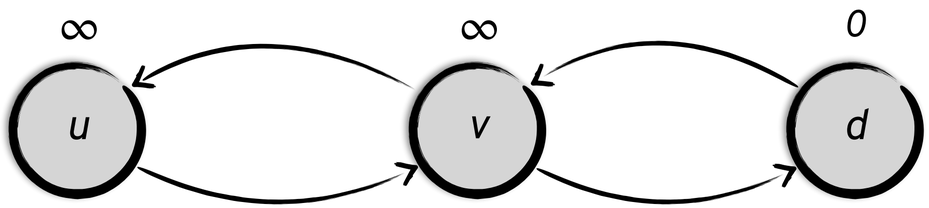
The Stable Routing Problem (SRP)

(Prior Work)

how to merge received attributes

topology of network

$$R = (G, A, \mathbf{init}, \oplus, \mathbf{trans})$$



set of attributes

how to transfer attributes along edges

initial attribute at each node

[Beckett et al., SIGCOMM 2018]

[Griffin and Sobrinho, SIGCOMM 2005]

[Sobrinho, IEANEP 2005]

[Griffin et al., IEANEP 2002]

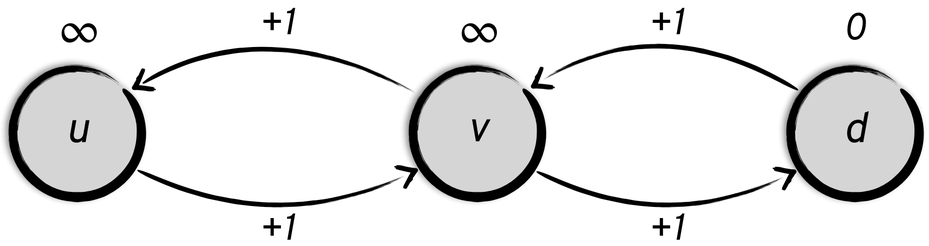
The Stable Routing Problem (SRP)

(Prior Work)

how to merge received attributes

topology of network

$$R = (G, A, \textbf{init}, \oplus, \textbf{trans})$$



set of attributes

how to transfer attributes along edges

initial attribute at each node

[Beckett et al., SIGCOMM 2018]

[Griffin and Sobrinho, SIGCOMM 2005]

[Sobrinho, IEANEP 2005]

[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

how to merge received attributes

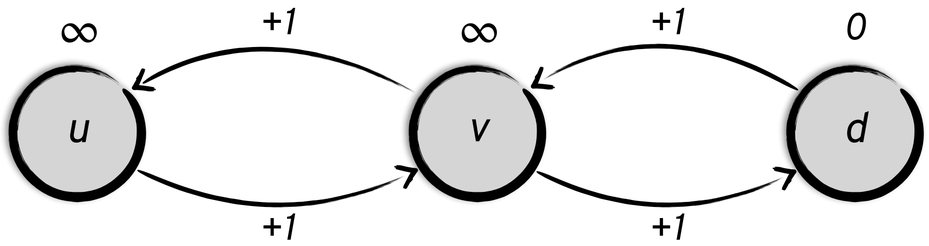
topology of network

$$R = (G, A, \textbf{init}, \oplus, \textbf{trans})$$

set of attributes

how to transfer attributes along edges

initial attribute at each node

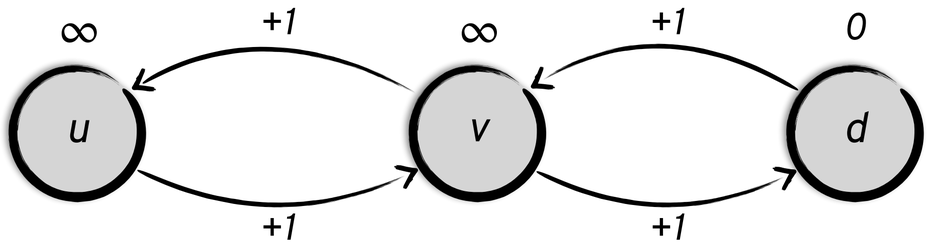


Can every router reach the destination d in under 10 hops?

[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)



Can every router reach the destination d in under 10 hops?

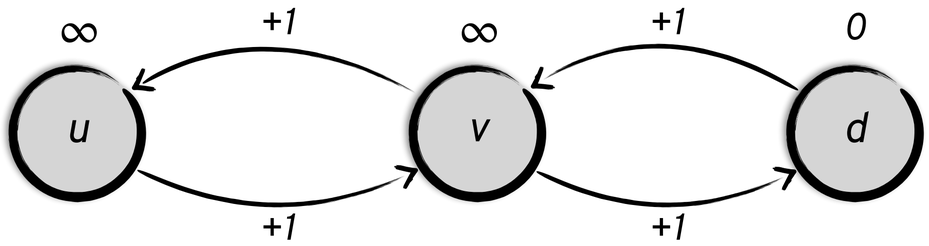
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 0



Can every router reach the destination d in under 10 hops?

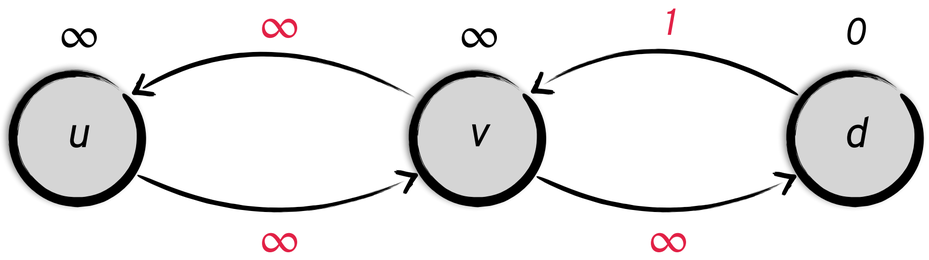
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 1



Can every router reach the destination d in under 10 hops?

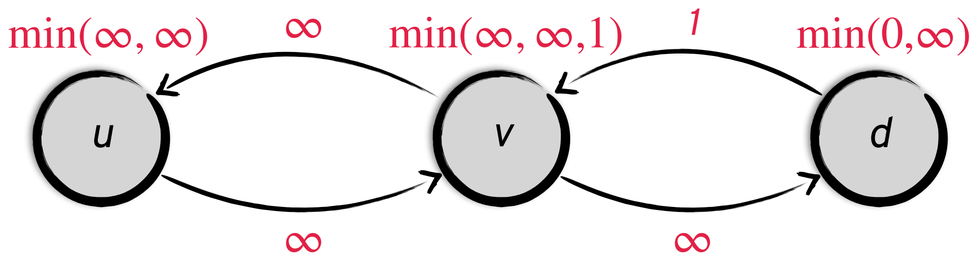
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 1



Can every router reach the destination d in under 10 hops?

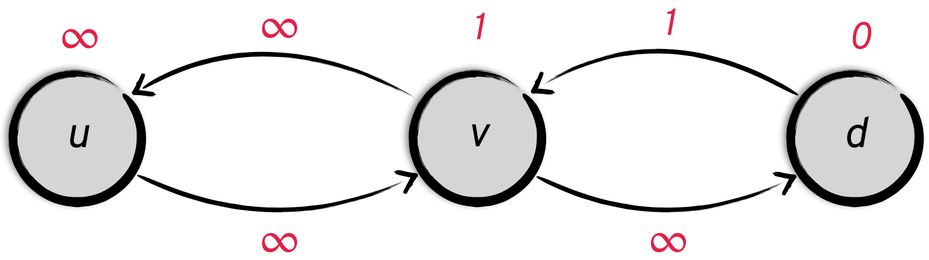
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 1



Can every router reach the destination d in under 10 hops?

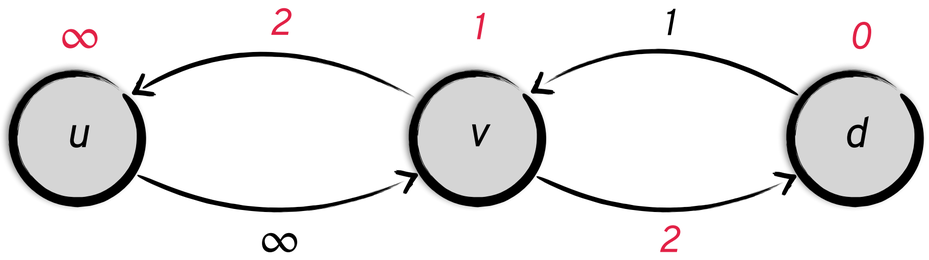
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 2



Can every router reach the destination d in under 10 hops?

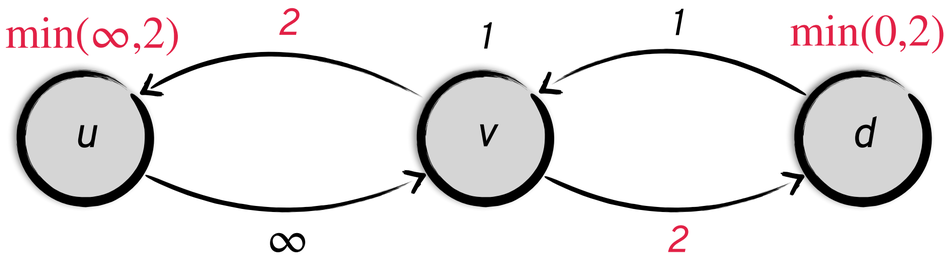
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 2



Can every router reach the destination d in under 10 hops?

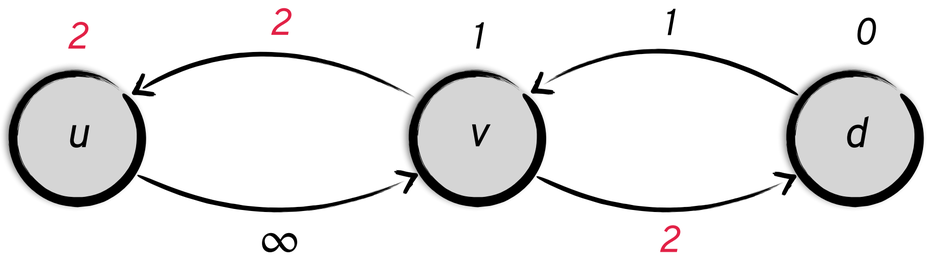
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 2



Can every router reach the destination d in under 10 hops?

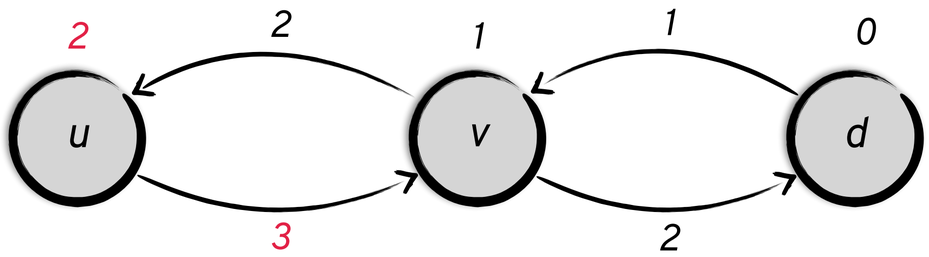
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 3



Can every router reach the destination d in under 10 hops?

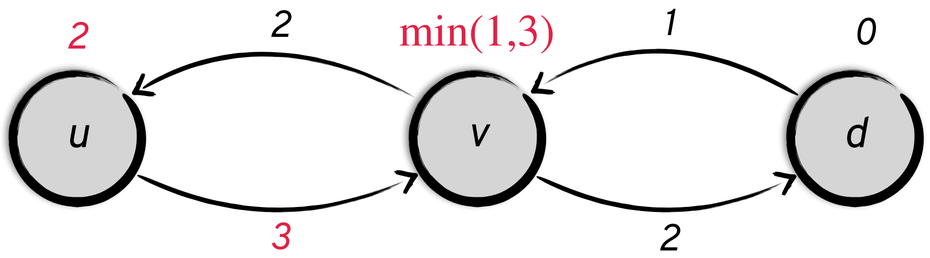
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 3



Can every router reach the destination d in under 10 hops?

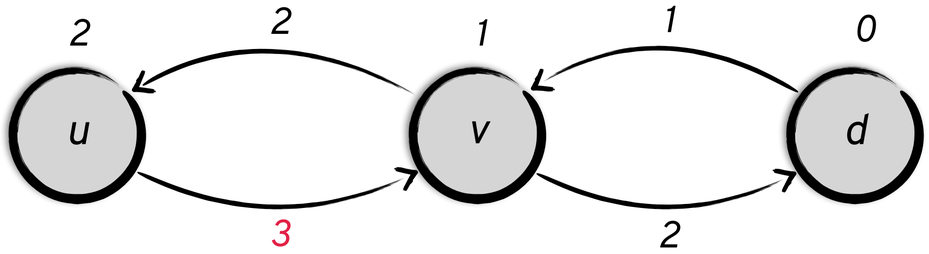
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 3



Can every router reach the destination d in under 10 hops?

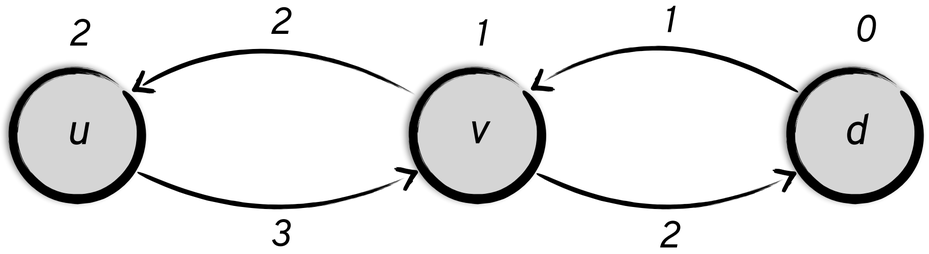
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 4



Can every router reach the destination d in under 10 hops?

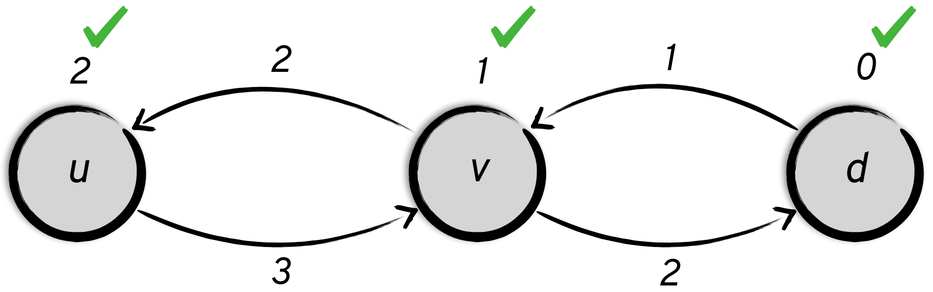
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 4



Can every router reach the destination d in under 10 hops? ✓

[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

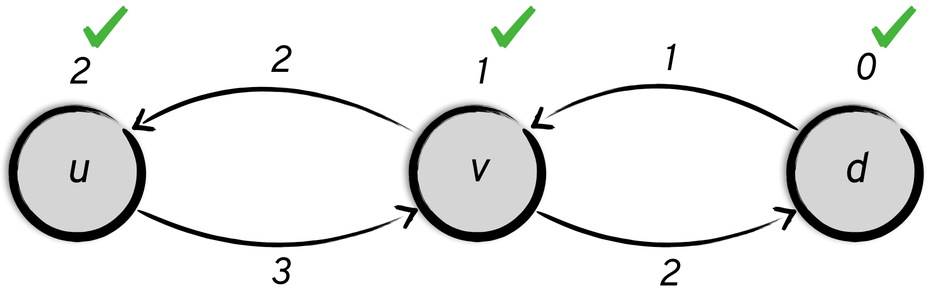
(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

captures *converged* solutions:

$$\mathscr{L}(u) = 2$$
$$\mathscr{L}(v) = 1$$
$$\mathscr{L}(d) = 0$$

Step: 4



Can every router reach the destination d in under 10 hops? ✓

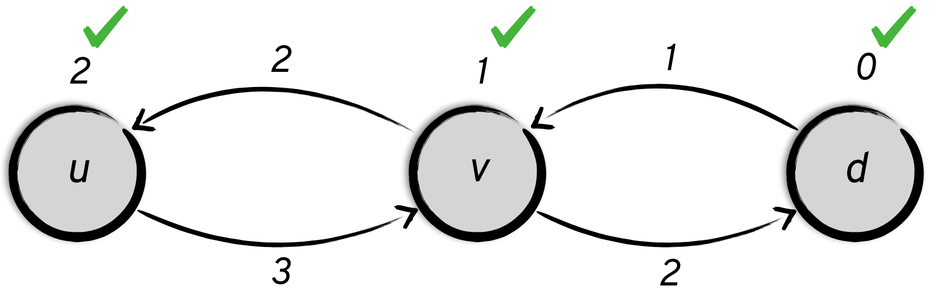
[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

The Stable Routing Problem (SRP)

(Prior Work)

$$A = \mathbb{N}$$
$$\text{init}(n) = \text{if } n = d \text{ then } 0 \text{ else } \infty$$
$$a \oplus b = \min(a, b)$$
$$\text{trans}(e, x) = x + 1$$

Step: 4



captures *converged* solutions:

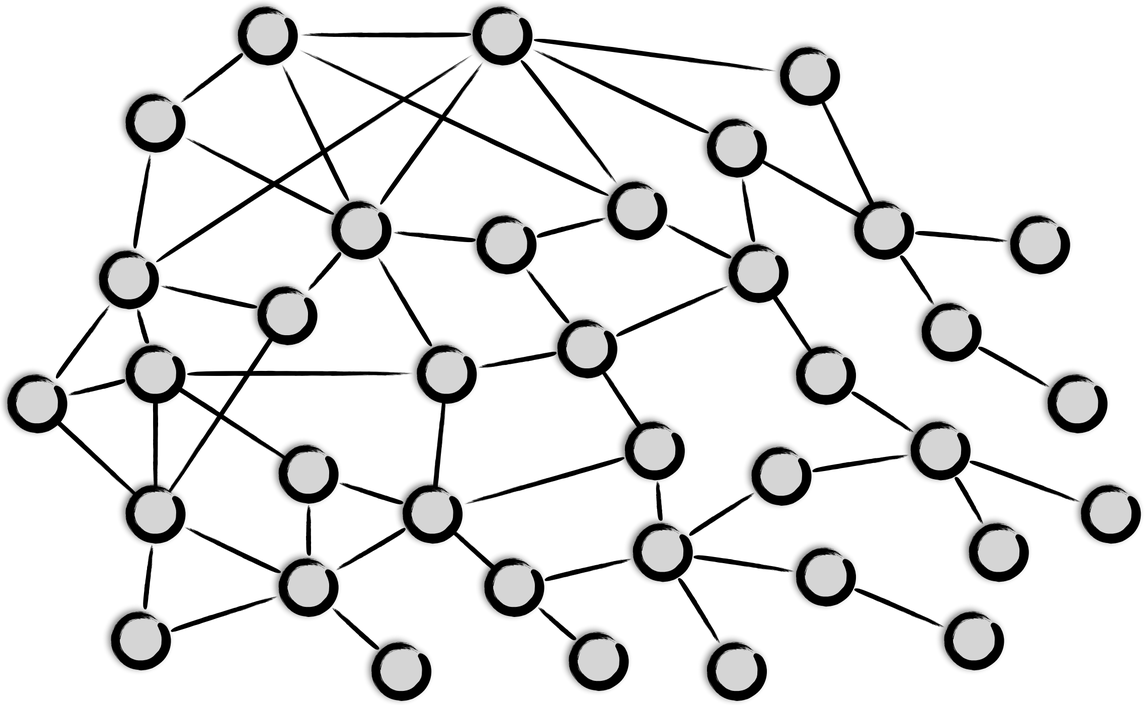
$$\mathscr{L}(u) = \min(\infty, \mathscr{L}(v) + 1)$$
$$\mathscr{L}(v) = \min(\infty, \mathscr{L}(u) + 1, \mathscr{L}(d) + 1)$$
$$\mathscr{L}(d) = \min(0, \mathscr{L}(v) + 1)$$

Can every router reach the destination d in under 10 hops?

[Beckett et al., SIGCOMM 2018]
[Griffin and Sobrinho, SIGCOMM 2005]
[Sobrinho, IEANEP 2005]
[Griffin et al., IEANEP 2002]

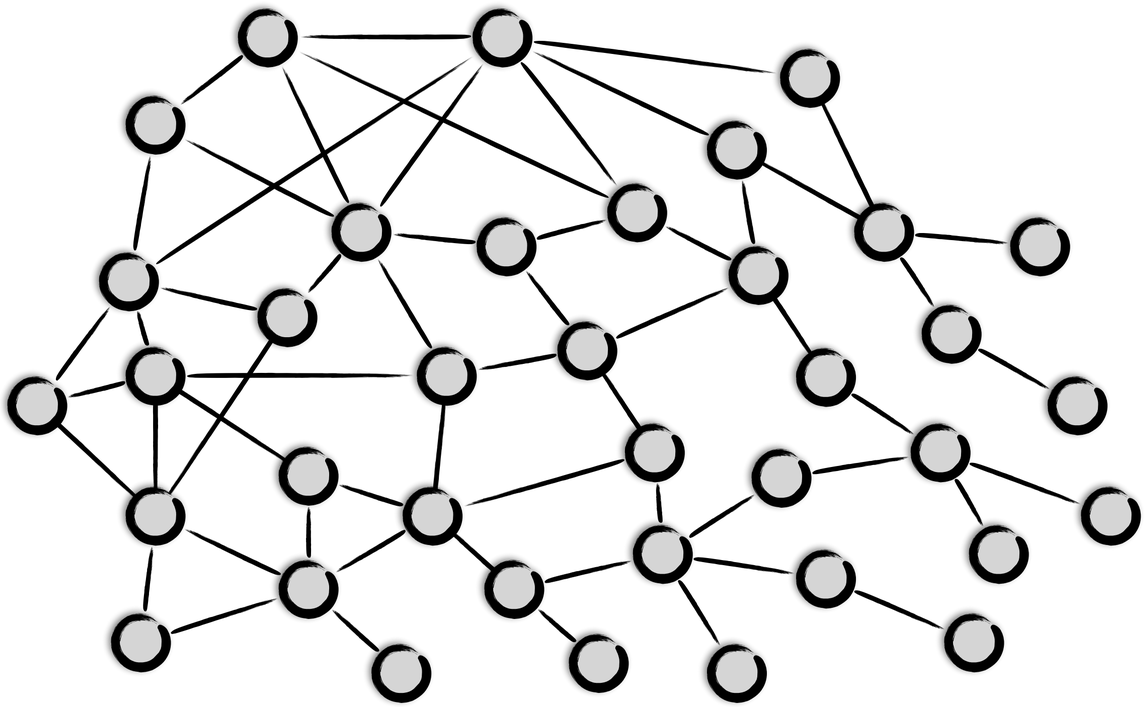
Verifying SRPs Compositionally

Verifying SRPs Compositionally



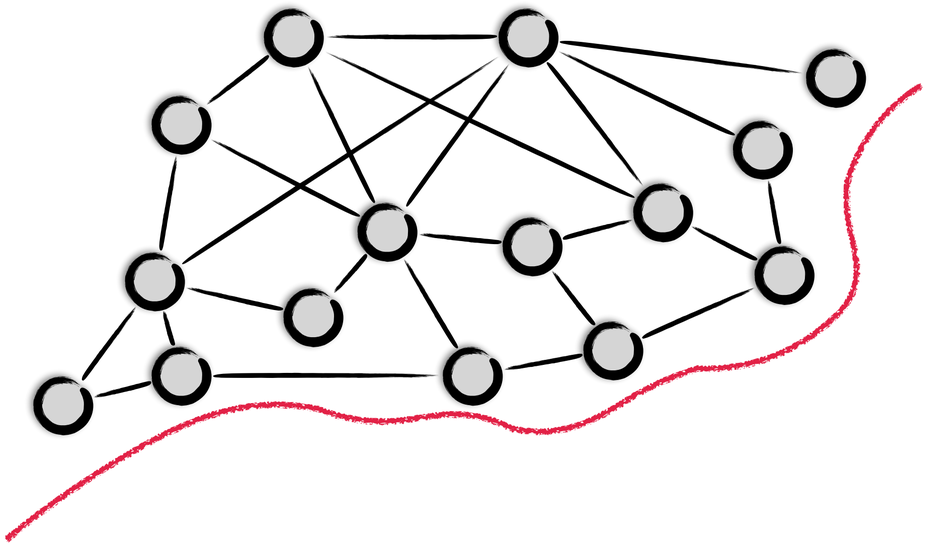
Verifying SRPs Compositionally

- Identify a cut-set of edges as forming an *interface*



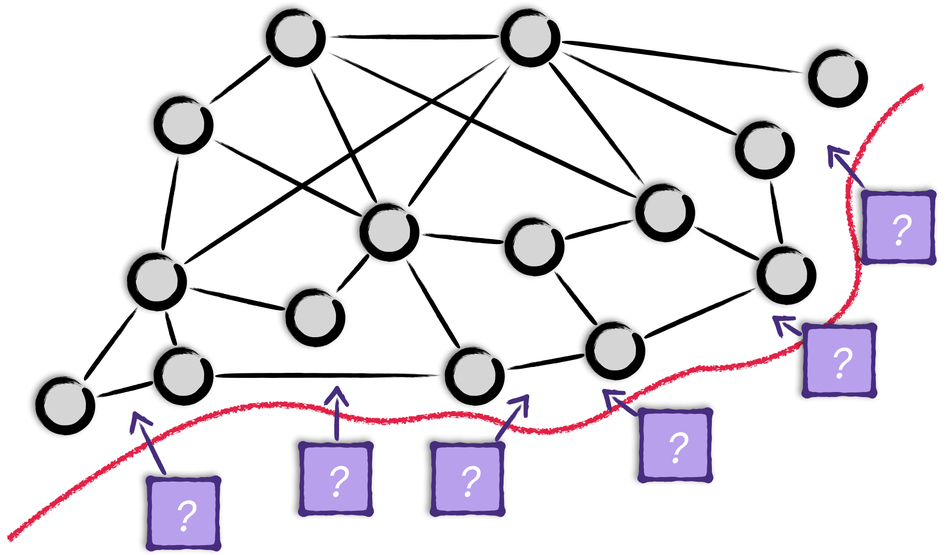
Verifying SRPs Compositionally

- Identify a cut-set of edges as forming an *interface*
- *Partition* the network along those edges



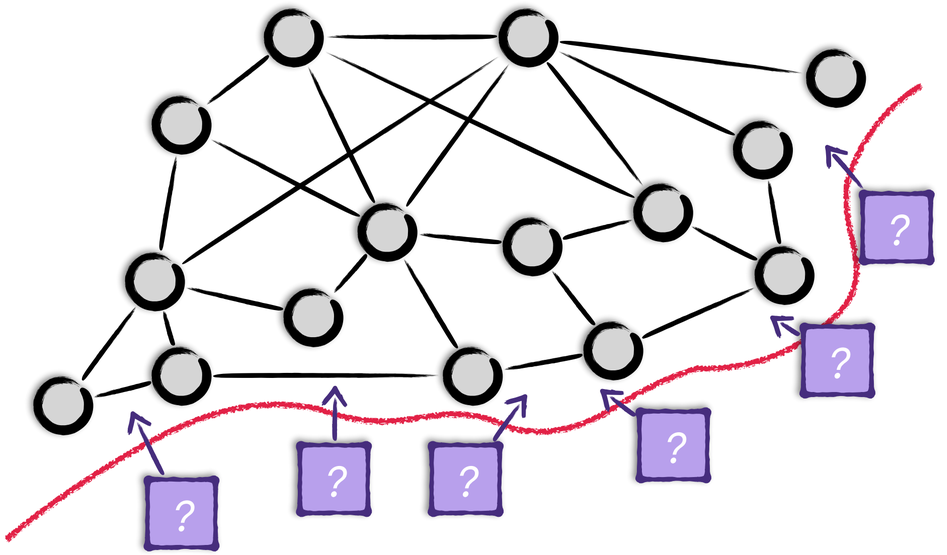
Verifying SRPs Compositionally

- Identify a cut-set of edges as forming an *interface*
- *Partition* the network along those edges
- Provide *hypotheses* for the interface edges
  - (We'll assume that the user provides these)

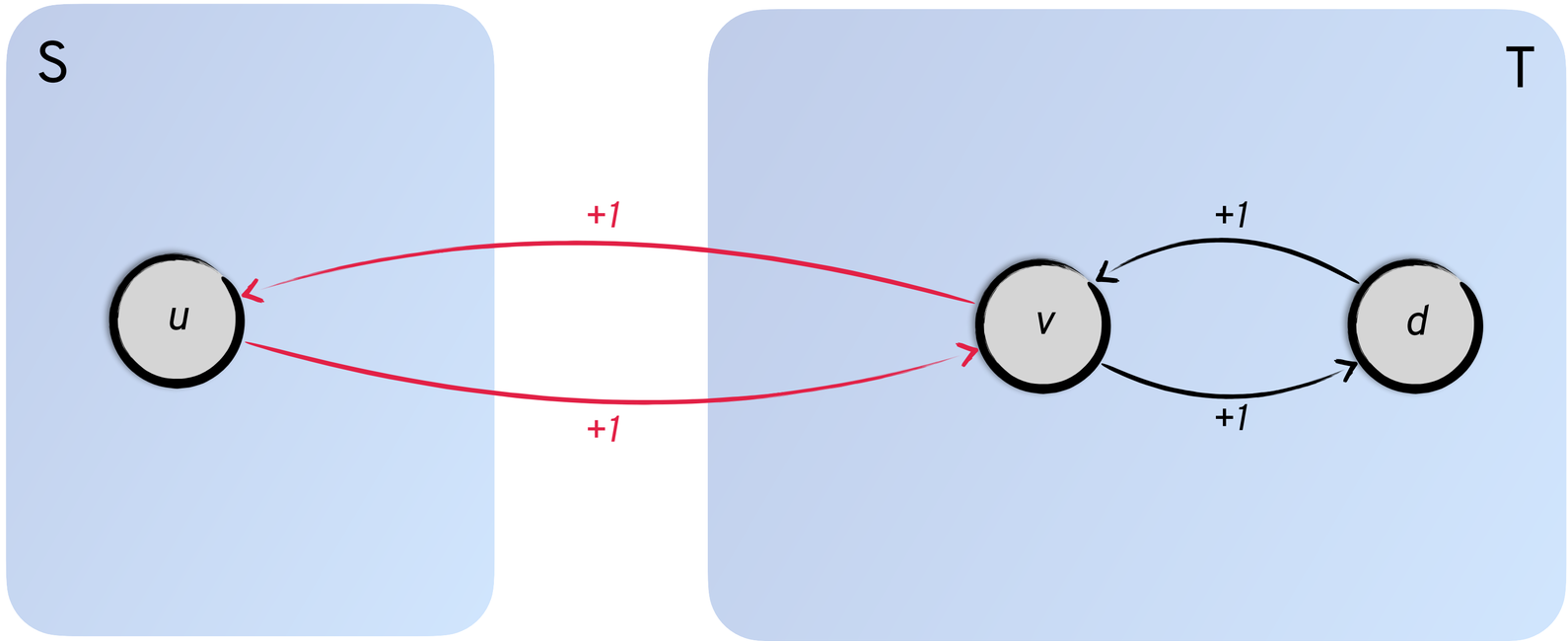


Verifying SRPs Compositionally

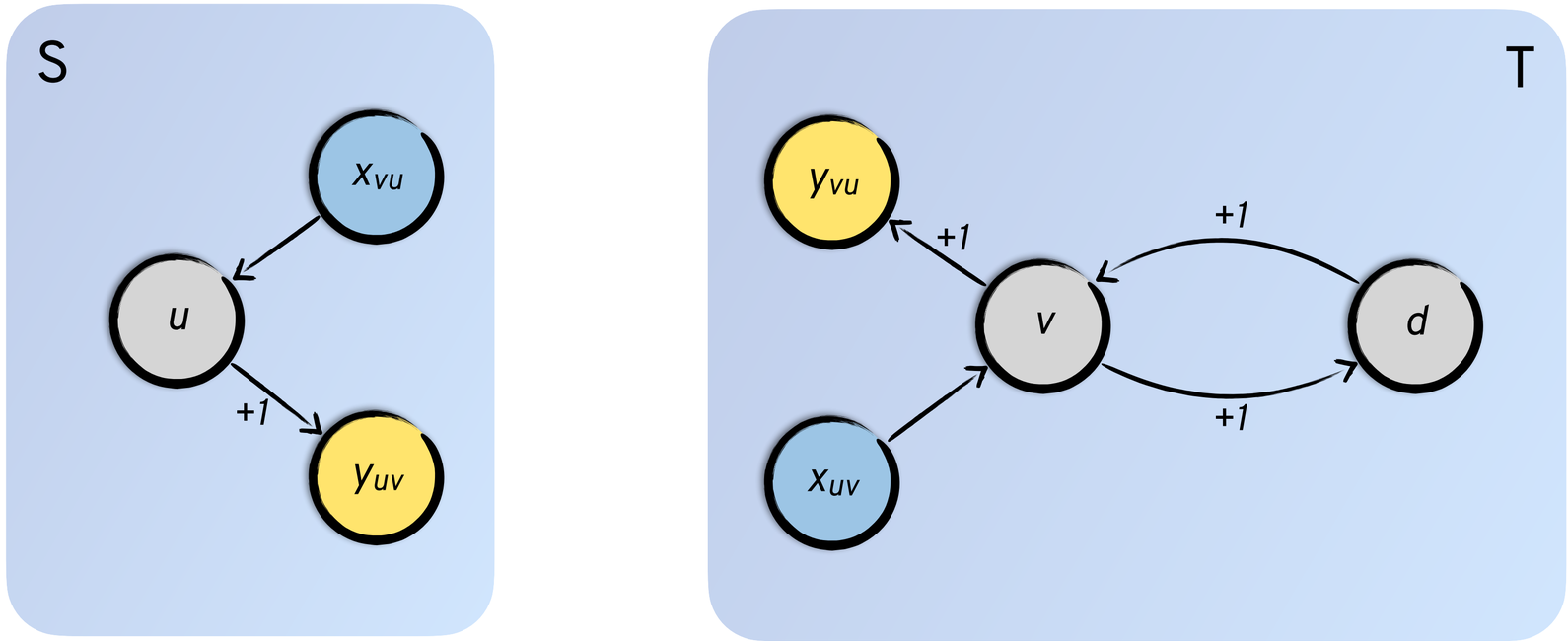
- Identify a cut-set of edges as forming an *interface*
- *Partition* the network along those edges
- Provide *hypotheses* for the interface edges
  - (We'll assume that the user provides these)
- Verify the components using hypotheses, and then demonstrate that this suffices to prove the monolithic network correct



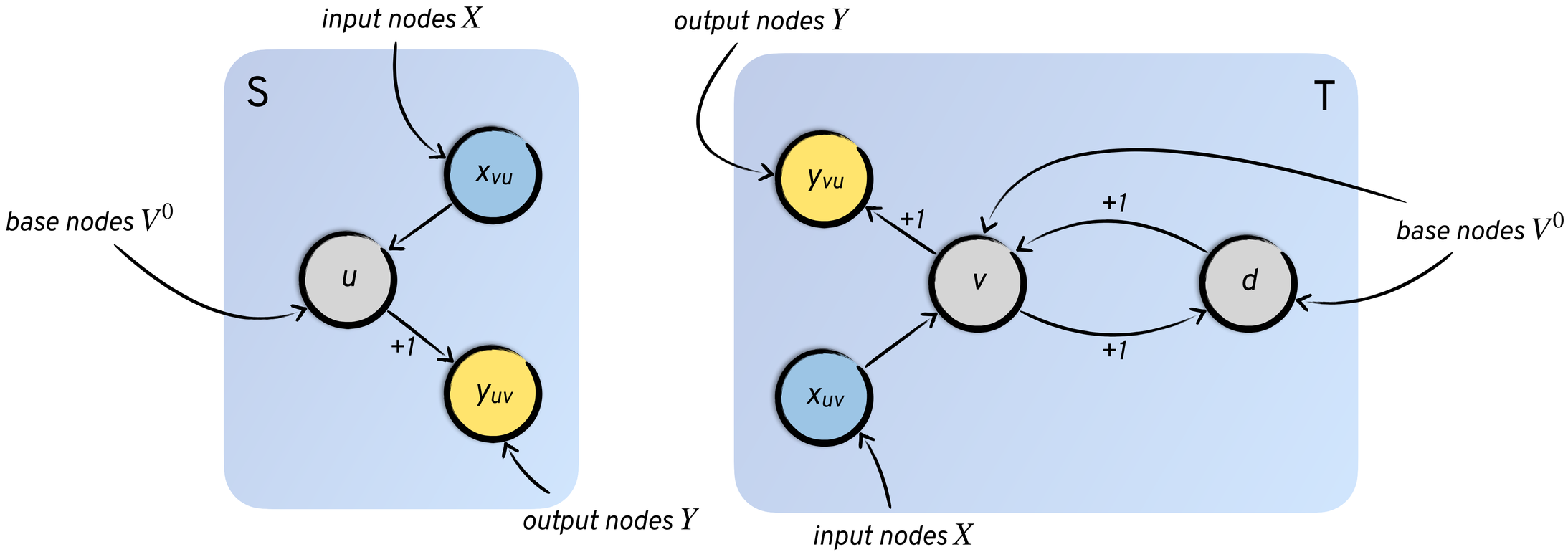
Open SRPs



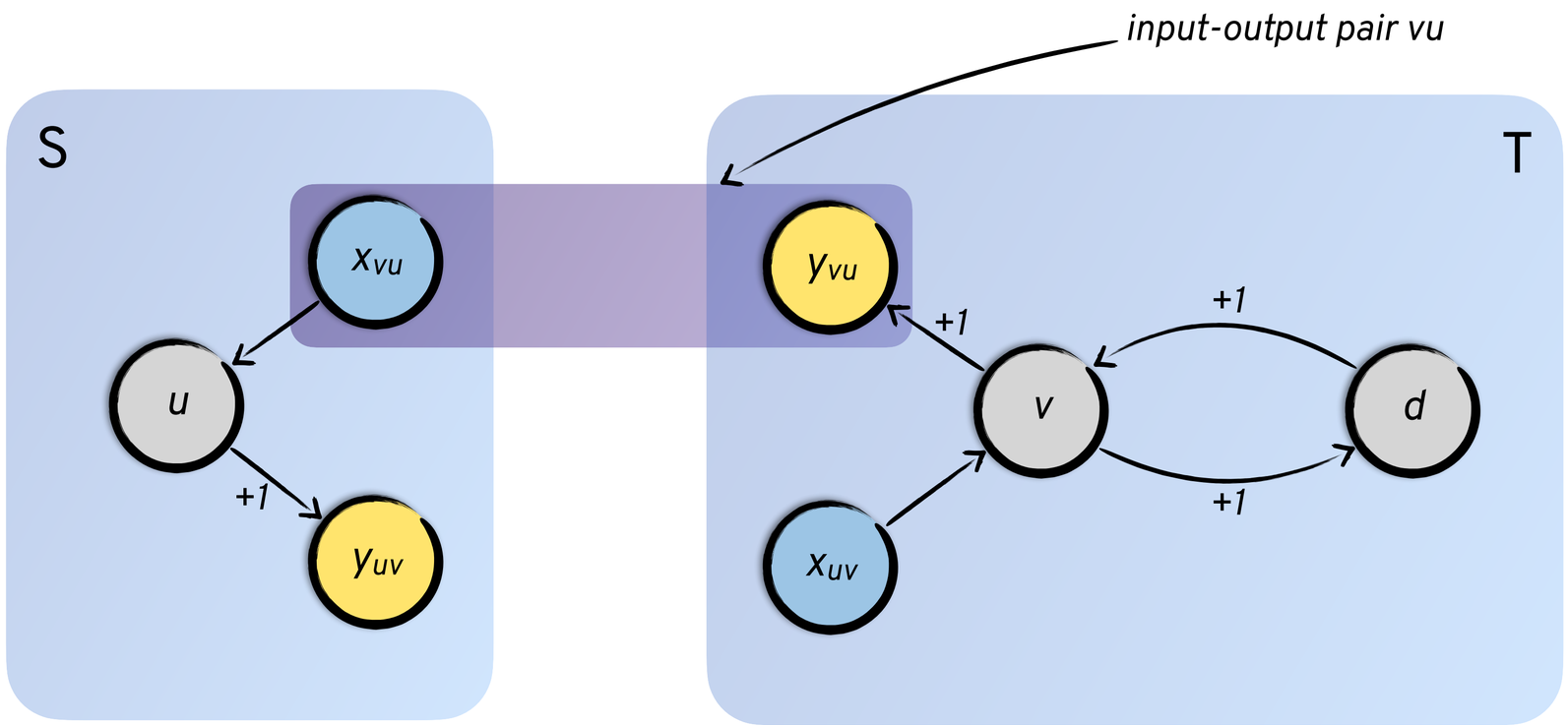
Open SRPs



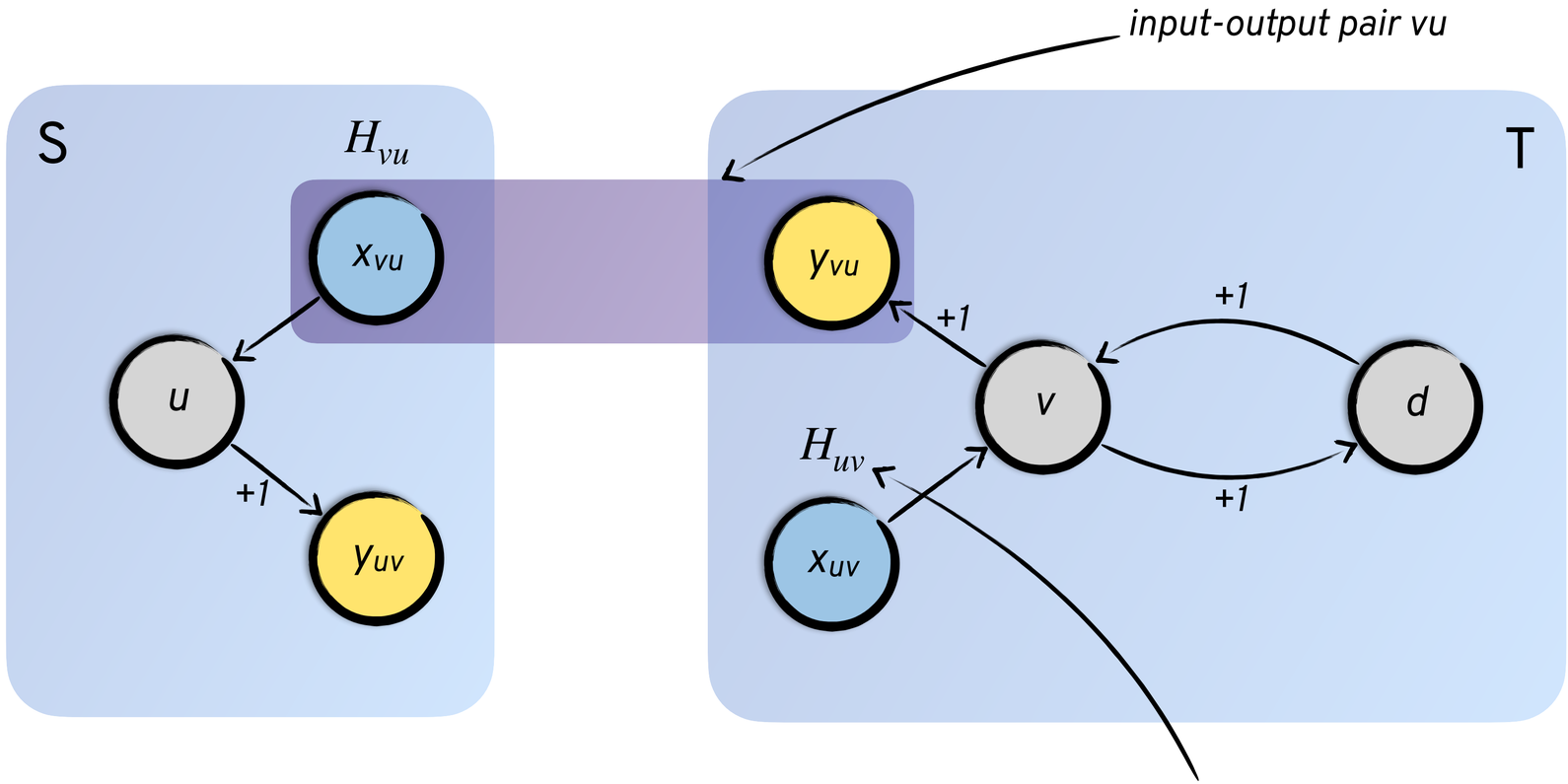
Open SRPs


input nodes X
S
x_{vu}
base nodes V^0
u
+1
y_{uv}
output nodes Y
output nodes Y
T
y_{vu}
+1
+1
base nodes V^0
v
d
+1
x_{uv}
input nodes X

Open SRPs

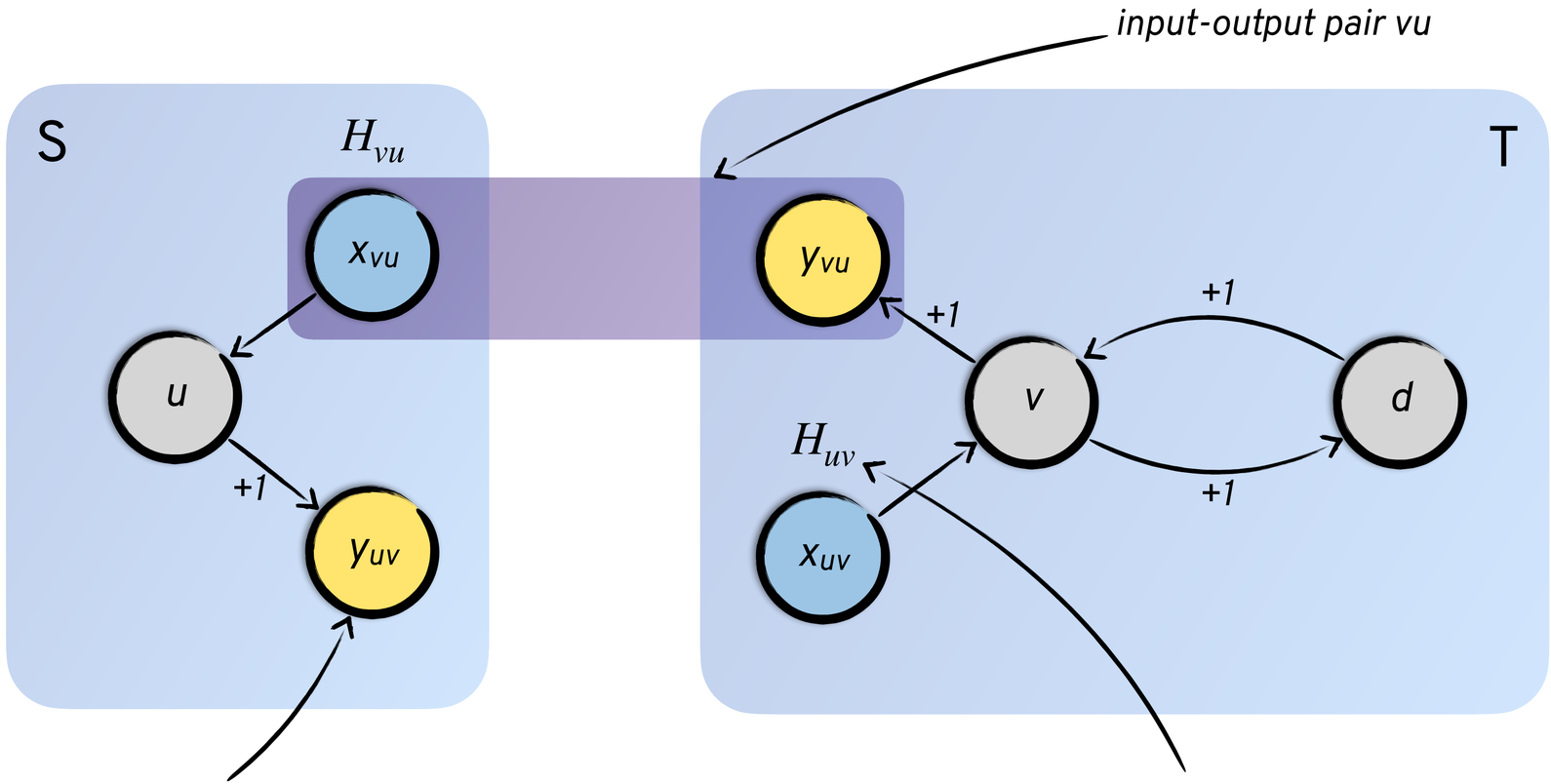


Open SRPs



hypotheses $\mathbf{H}$ are formulae over the solutions of inputs:
they provide assumptions about the other partition

Open SRPs

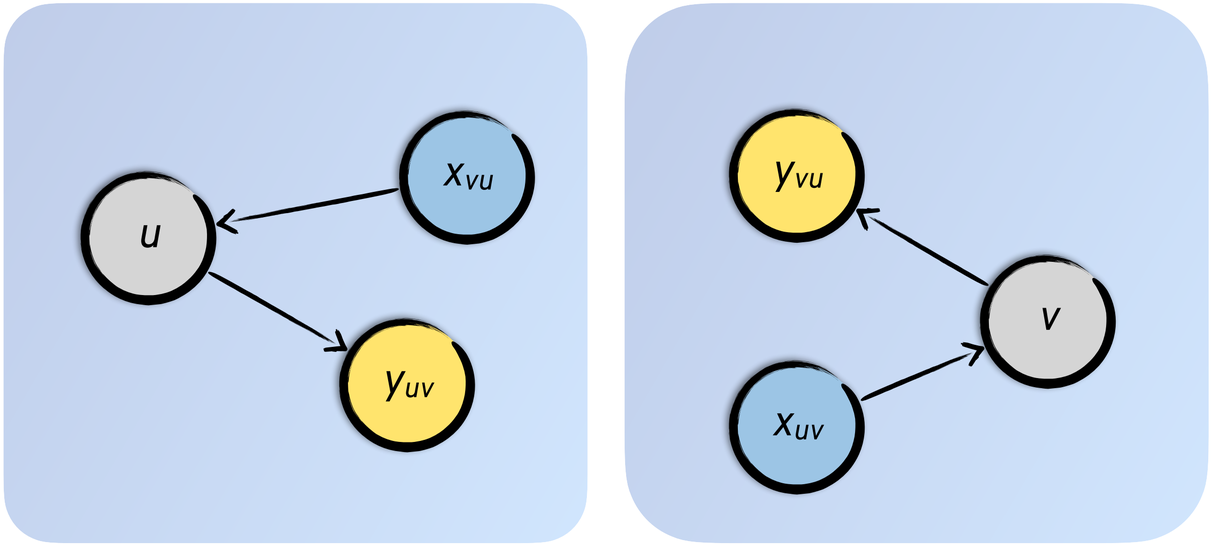


formulae over output solutions provide guarantees for the opposite partition

hypotheses $\mathbf{H}$ are formulae over the solutions of inputs: they provide assumptions about the other partition

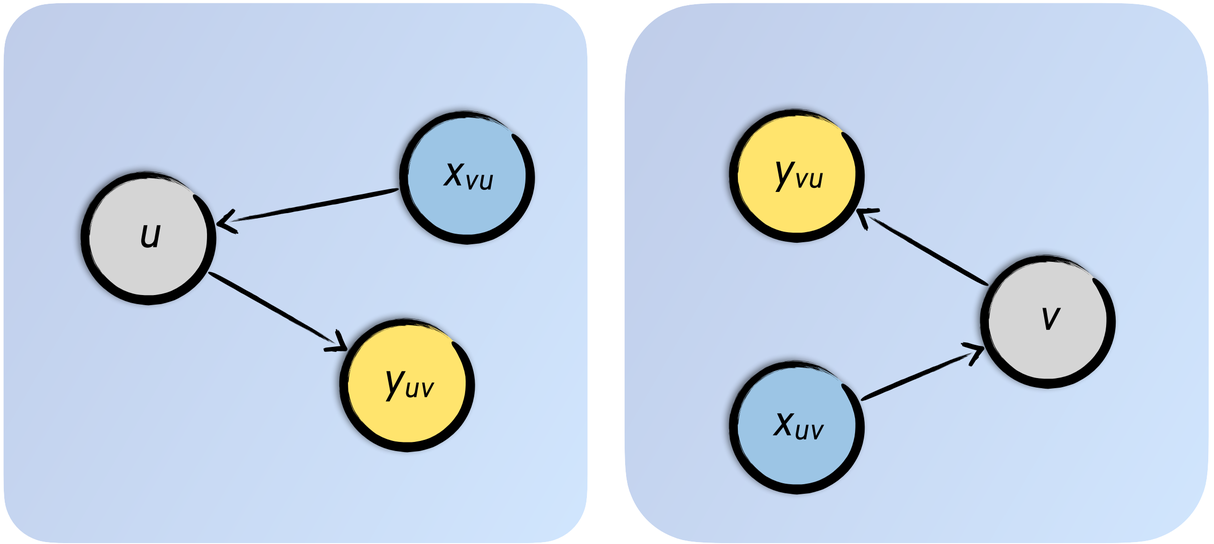
The Kirigami Algorithm

Interfaces



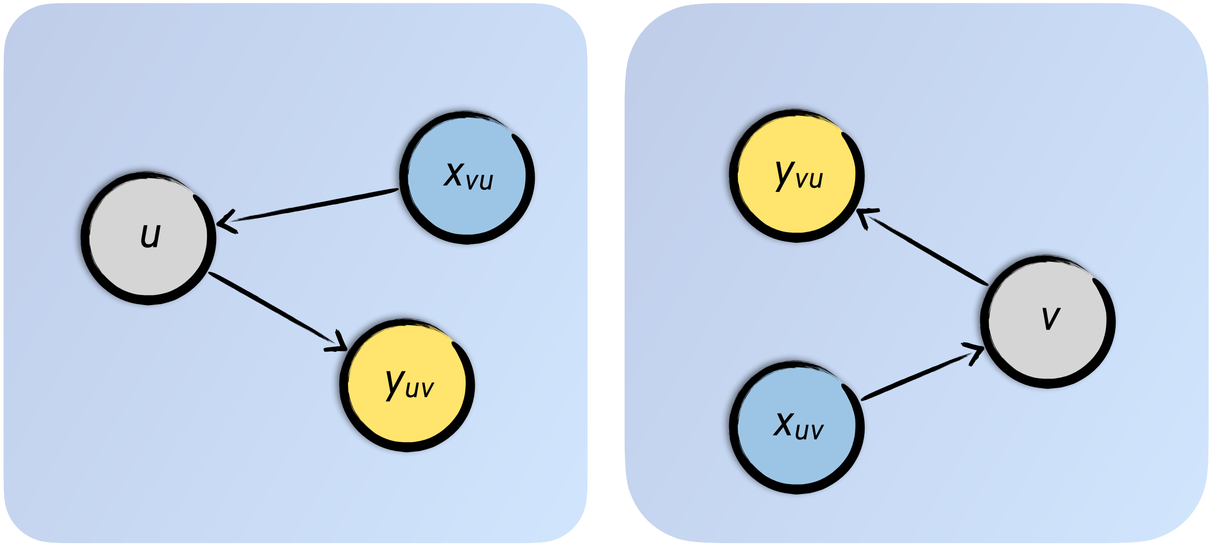
Interfaces

- To verify an SRP compositionally, let's define an *interface*!



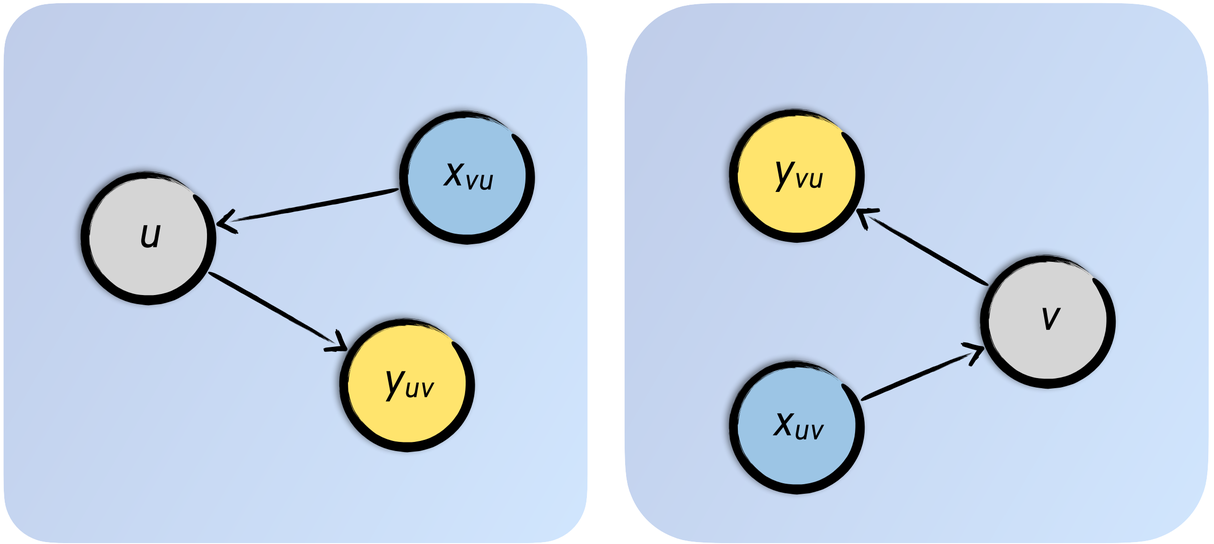
Interfaces

- To verify an SRP compositionally, let's define an *interface*!
  - An *interface* is a set of cut-set edges, each with an associated hypothesis



Interfaces

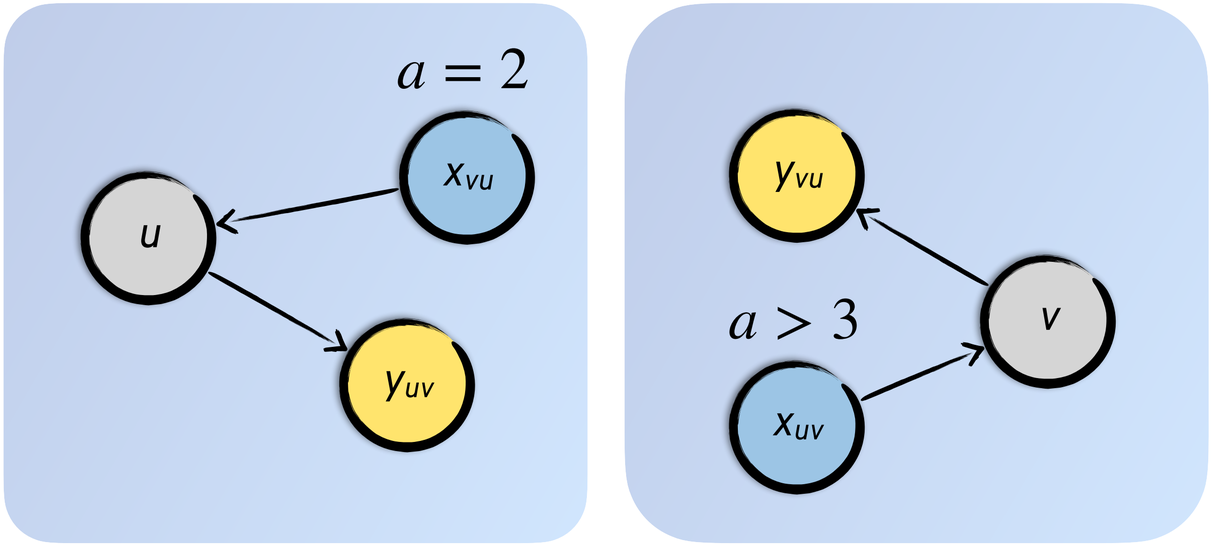
- To verify an SRP compositionally, let's define an *interface*!
  - An *interface* is a set of cut-set edges, each with an associated hypothesis
  - $\mathcal{I} = \left\{ \left(u_j, v_j, H_{u_j v_j} \right) \right\}$



Interfaces

- To verify an SRP compositionally, let's define an *interface*!
  - An *interface* is a set of cut-set edges, each with an associated hypothesis
  - $\mathcal{I} = \left\{ \left(u_j, v_j, H_{u_j v_j} \right) \right\}$

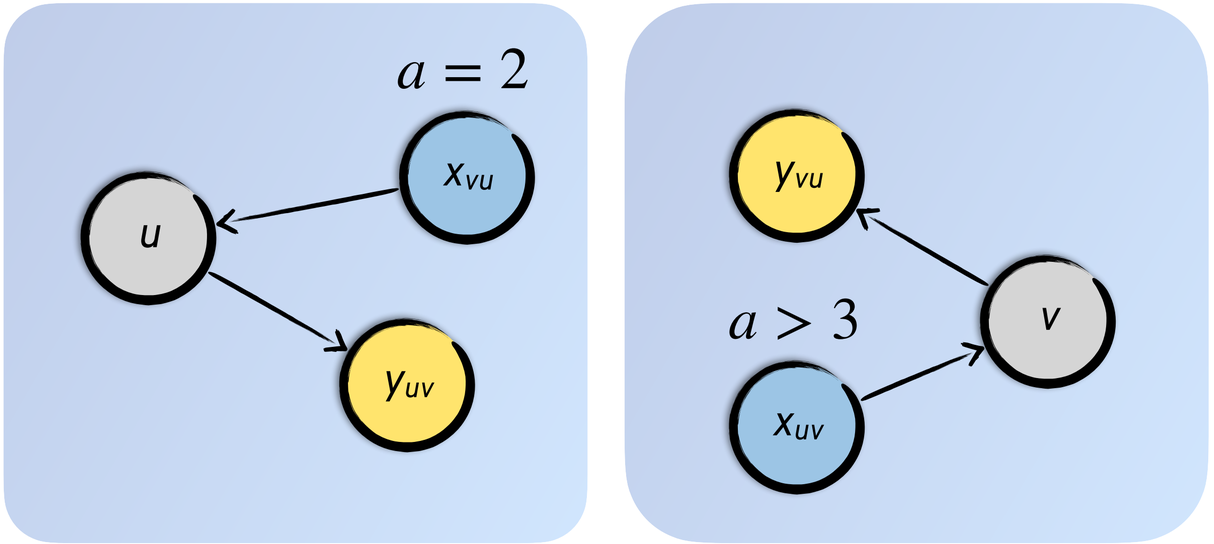
$(v, u, a = 2)$
$(u, v, a > 3)$



Interfaces

- To verify an SRP compositionally, let's define an *interface*!
  - An *interface* is a set of cut-set edges, each with an associated hypothesis
  - $\mathcal{I} = \left\{ \left(u_j, v_j, H_{u_j v_j} \right) \right\}$
- Hypotheses needs to be *weak enough* to capture all behaviours of the original SRP, while *strong enough* to still allow us to prove our property

$(v, u, a = 2)$
$(u, v, a > 3)$



The Kirigami Algorithm

An algorithm for checking *interfaces*

[Beckett et al., SIGCOMM 2017]

The Kirigami Algorithm

An algorithm for checking *interfaces*

Input: a closed SRP R, a property P and an interface $\mathcal{I}$

Output: “TRUE” if the partitioned SRPs *soundly over-approximate* R and P holds; else “FALSE” with counterexample(s)

[Beckett et al., SIGCOMM 2017]

The Kirigami Algorithm

An algorithm for checking *interfaces*

Input: a closed SRP R, a property P and an interface $\mathcal{I}$

Output: “TRUE” if the partitioned SRPs *soundly over-approximate* R and P holds; else “FALSE” with counterexample(s)

1. Partition R into disconnected open SRPs S and T

$$\mathit{partition}(R, P, \mathcal{I}) = ((S, P_S, \mathbf{H}_S), (T, P_T, \mathbf{H}_T))$$

[Beckett et al., SIGCOMM 2017]

The Kirigami Algorithm

An algorithm for checking *interfaces*

Input: a closed SRP R, a property P and an interface $\mathcal{I}$

Output: “TRUE” if the partitioned SRPs *soundly over-approximate* R and P holds; else “FALSE” with counterexample(s)

1. Partition R into disconnected open SRPs S and T

$$\mathit{partition}(R, P, \mathcal{I}) = ((S, P_S, \mathbf{H}_S), (T, P_T, \mathbf{H}_T))$$

2. Encode S and T as *transition relations* $\mathcal{M}_S$ and $\mathcal{M}_T$ using the Minesweeper encoding

[Beckett et al., SIGCOMM 2017]

The Kirigami Algorithm

An algorithm for checking *interfaces*

Input: a closed SRP R, a property P and an interface $\mathscr{I}$

Output: “TRUE” if the partitioned SRPs *soundly over-approximate* R and P holds; else “FALSE” with counterexample(s)

1. Partition R into disconnected open SRPs S and T

$$\mathit{partition}(R, P, \mathscr{I}) = ((S, P_S, \mathbf{H}_S), (T, P_T, \mathbf{H}_T))$$

2. Encode S and T as *transition relations* $\mathscr{M}_S$ and $\mathscr{M}_T$ using the Minesweeper encoding

   - Solutions of outputs = *strongest postconditions* over transition relations and the input hypotheses

[Beckett et al., SIGCOMM 2017]

The Kirigami Algorithm

An algorithm for checking *interfaces*

Input: a closed SRP R, a property P and an interface $\mathcal{I}$

Output: “TRUE” if the partitioned SRPs *soundly over-approximate* R and P holds; else “FALSE” with counterexample(s)

1. Partition R into disconnected open SRPs S and T

$$\textit{partition}(R, P, \mathcal{I}) = ((S, P_S, \mathbf{H}_S), (T, P_T, \mathbf{H}_T))$$

2. Encode S and T as *transition relations* $\mathscr{M}_S$ and $\mathscr{M}_T$ using the Minesweeper encoding
   - Solutions of outputs = *strongest postconditions* over transition relations and the input hypotheses
3. For partition S (resp. T), check validity of the following *verification conditions* (VCs) on $\mathscr{M}_S$ (resp. $\mathscr{M}_T$):

[Beckett et al., SIGCOMM 2017]

The Kirigami Algorithm

An algorithm for checking *interfaces*

Input: a closed SRP R, a property P and an interface $\mathcal{I}$

Output: "TRUE" if the partitioned SRPs *soundly over-approximate* R and P holds; else "FALSE" with counterexample(s)

1. Partition R into disconnected open SRPs S and T

$$\mathit{partition}(R, P, \mathcal{I}) = ((S, P_S, \mathbf{H}_S), (T, P_T, \mathbf{H}_T))$$

2. Encode S and T as *transition relations* $\mathscr{M}_S$ and $\mathscr{M}_T$ using the Minesweeper encoding
   - Solutions of outputs = *strongest postconditions* over transition relations and the input hypotheses
3. For partition S (resp. T), check validity of the following *verification conditions* (VCs) on $\mathscr{M}_S$ (resp. $\mathscr{M}_T$):
   A. Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$

$\langle A \rangle M \langle G \rangle$ notation:
assuming A about M's inputs,
M must guarantee G (cf. Hoare triples)

[Beckett et al., SIGCOMM 2017]

The Kirigami Algorithm

An algorithm for checking *interfaces*

Input: a closed SRP R, a property P and an interface $\mathcal{I}$

Output: "TRUE" if the partitioned SRPs *soundly over-approximate* R and P holds; else "FALSE" with counterexample(s)

1. Partition R into disconnected open SRPs S and T

$$\mathit{partition}(R, P, \mathcal{I}) = ((S, P_S, \mathbf{H}_S), (T, P_T, \mathbf{H}_T))$$

2. Encode S and T as *transition relations* $\mathscr{M}_S$ and $\mathscr{M}_T$ using the Minesweeper encoding
   - Solutions of outputs = *strongest postconditions* over transition relations and the input hypotheses
3. For partition S (resp. T), check validity of the following *verification conditions* (VCs) on $\mathscr{M}_S$ (resp. $\mathscr{M}_T$):
   A. Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$
   B. Safety: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle P_S \rangle$

$\langle A \rangle M \langle G \rangle$ *notation:*
assuming A about M's inputs,
M must guarantee G (cf. Hoare triples)

[Beckett et al., SIGCOMM 2017]

The Kirigami Algorithm

An algorithm for checking *interfaces*

Input: a closed SRP R, a property P and an interface $\mathcal{I}$

Output: "TRUE" if the partitioned SRPs *soundly over-approximate* R and P holds; else "FALSE" with counterexample(s)

1. Partition R into disconnected open SRPs S and T

$$\mathit{partition}(R, P, \mathcal{I}) = ((S, P_S, \mathbf{H}_S), (T, P_T, \mathbf{H}_T))$$

2. Encode S and T as *transition relations* $\mathscr{M}_S$ and $\mathscr{M}_T$ using the Minesweeper encoding
   - Solutions of outputs = *strongest postconditions* over transition relations and the input hypotheses
3. For partition S (resp. T), check validity of the following *verification conditions* (VCs) on $\mathscr{M}_S$ (resp. $\mathscr{M}_T$):
   A. Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$
   B. Safety: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle P_S \rangle$
   C. Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$

$\langle A \rangle M \langle G \rangle$ *notation:*
assuming A about M's inputs,
M must guarantee G (cf. Hoare triples)

[Beckett et al., SIGCOMM 2017]

The Kirigami Algorithm

An algorithm for checking *interfaces*

Input: a closed SRP R, a property P and an interface $\mathcal{I}$

Output: "TRUE" if the partitioned SRPs *soundly over-approximate* R and P holds; else "FALSE" with counterexample(s)

1. Partition R into disconnected open SRPs S and T

$$\mathit{partition}(R, P, \mathcal{I}) = ((S, P_S, \mathbf{H}_S), (T, P_T, \mathbf{H}_T))$$

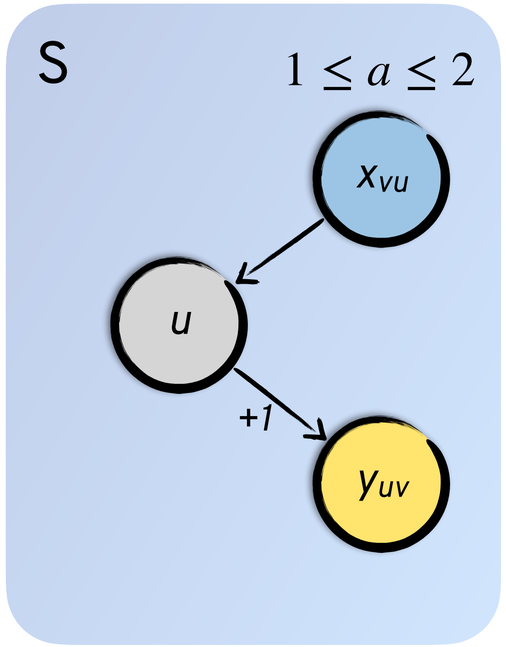
2. Encode S and T as *transition relations* $\mathscr{M}_S$ and $\mathscr{M}_T$ using the Minesweeper encoding
   - Solutions of outputs = *strongest postconditions* over transition relations and the input hypotheses
3. For partition S (resp. T), check validity of the following *verification conditions* (VCs) on $\mathscr{M}_S$ (resp. $\mathscr{M}_T$):
   A. Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$
   B. Safety: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle P_S \rangle$
   C. Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$
4. Return "TRUE" if the VCs hold, or "FALSE" with counterexample(s)

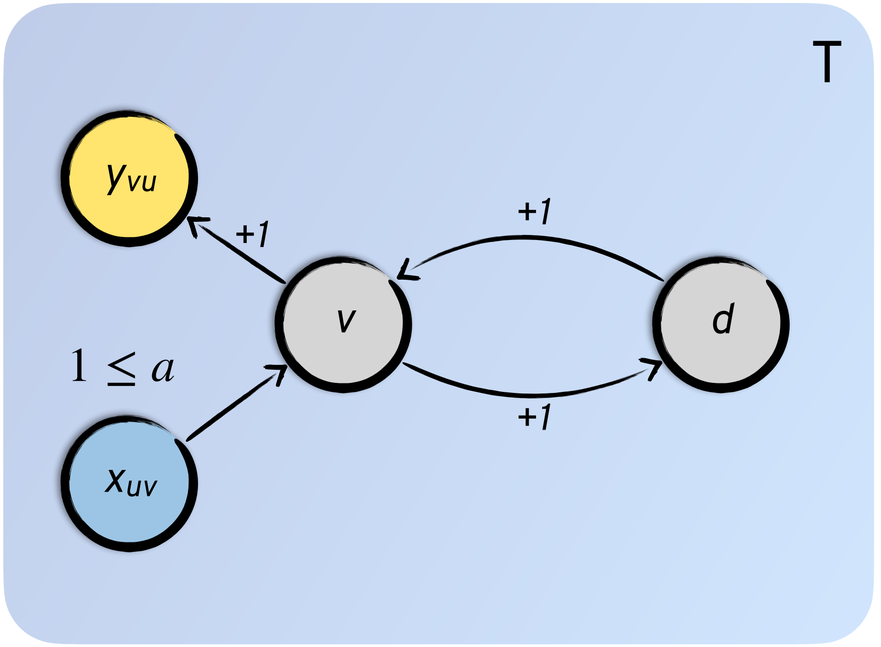
$\langle A \rangle M \langle G \rangle$ *notation:*
assuming A about M's inputs,
M must guarantee G (cf. Hoare triples)

[Beckett et al., SIGCOMM 2017]

Checking an Interface

A: Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$



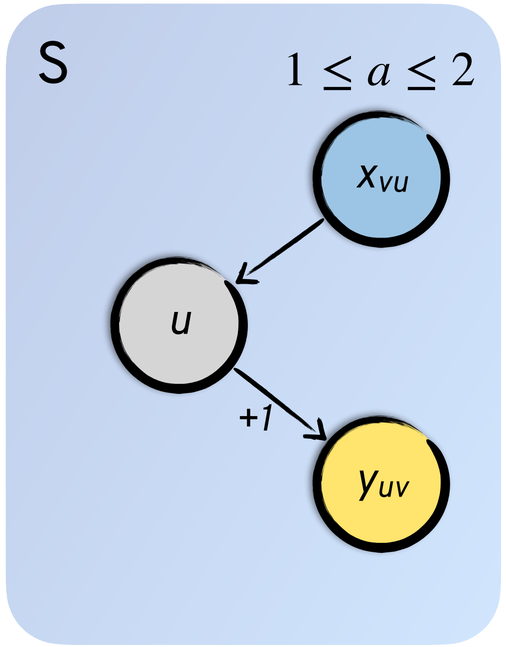


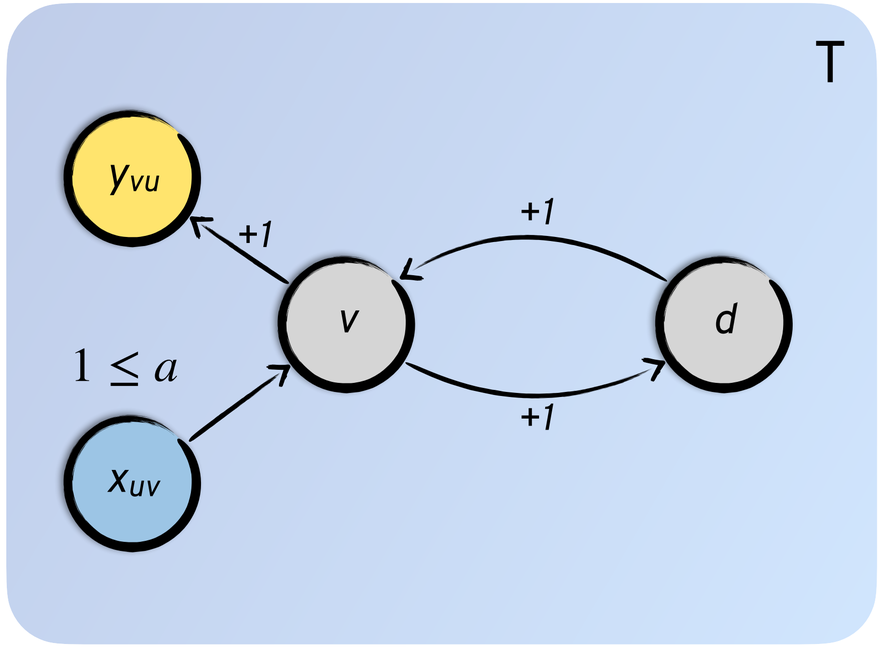
Checking an Interface

A: Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

$(v, u, 1 \leq a \leq 2)$

$(u, v, 1 \leq a)$

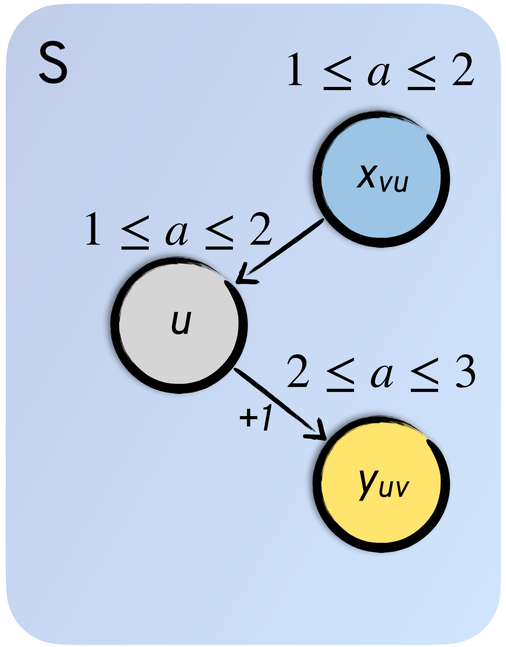


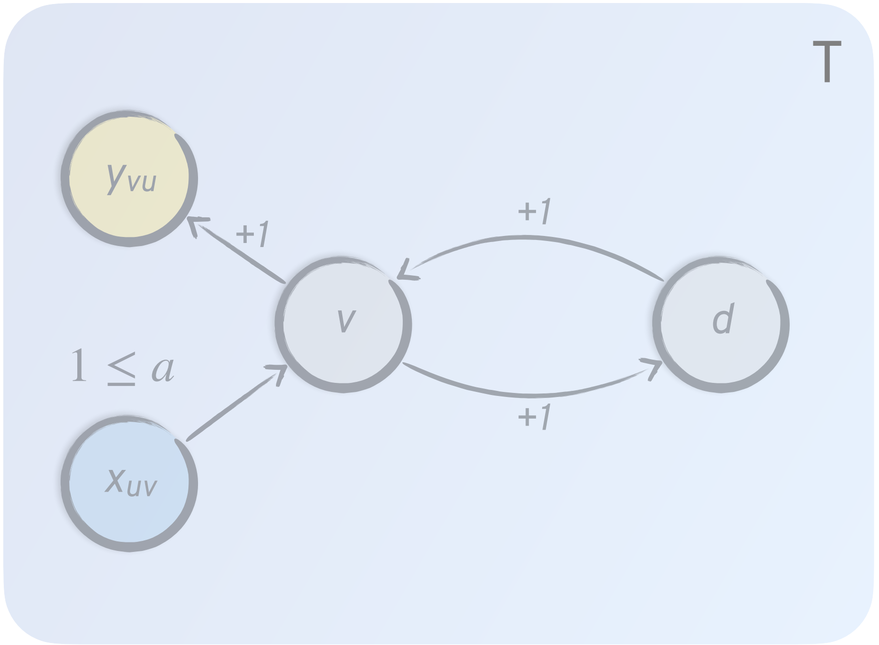


Checking an Interface

A: Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$

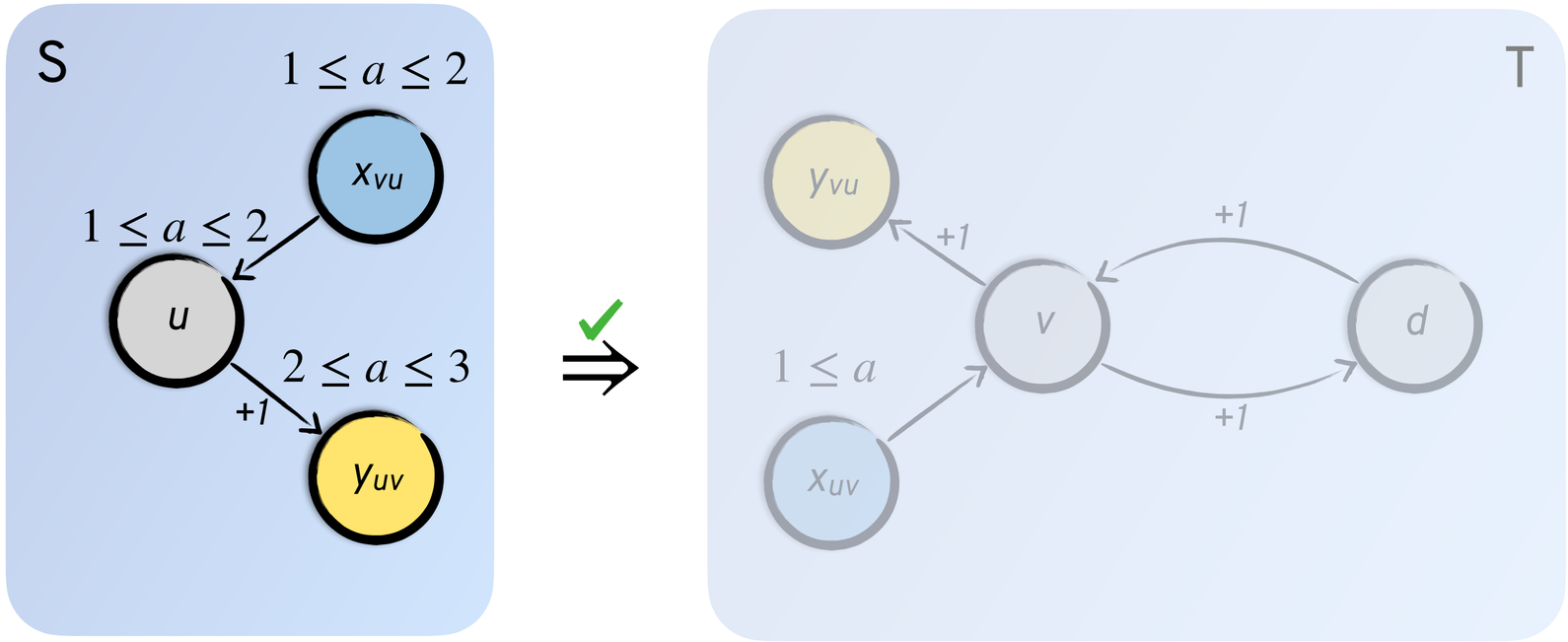




Checking an Interface

A: Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

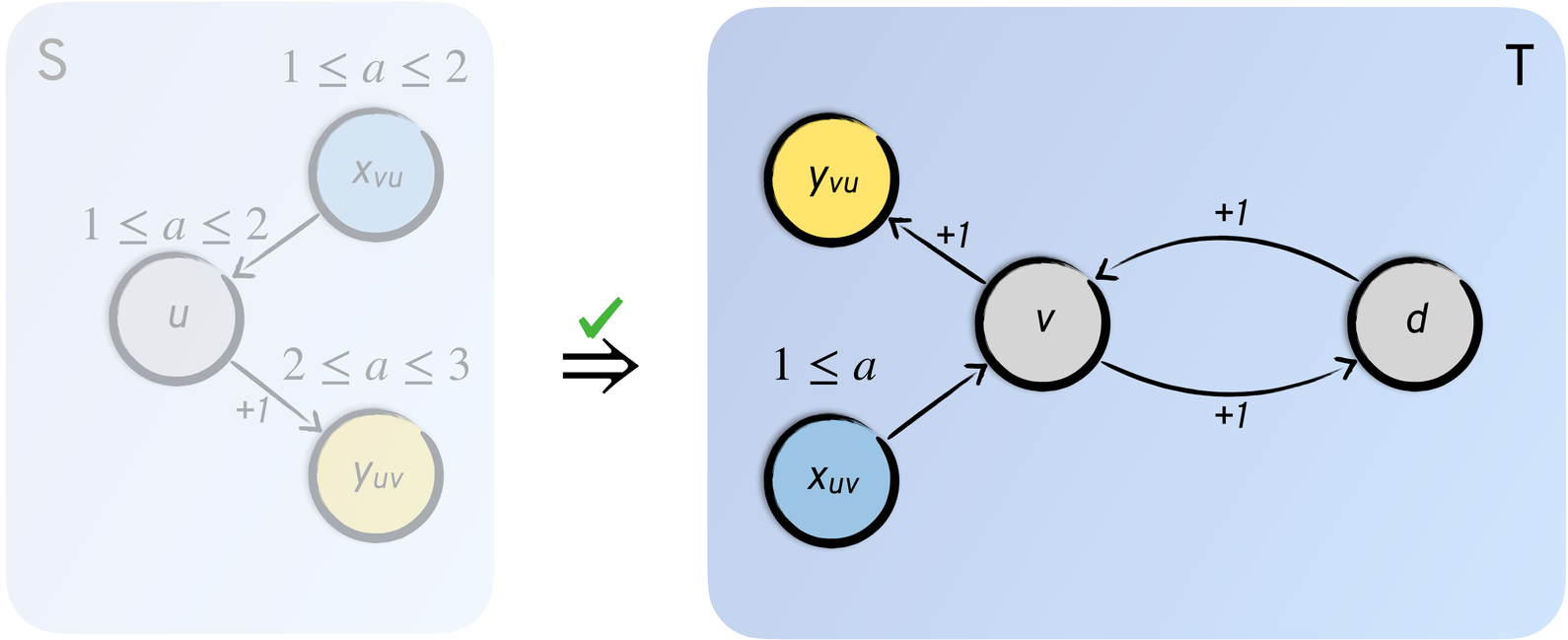
$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$



Checking an Interface

A: Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

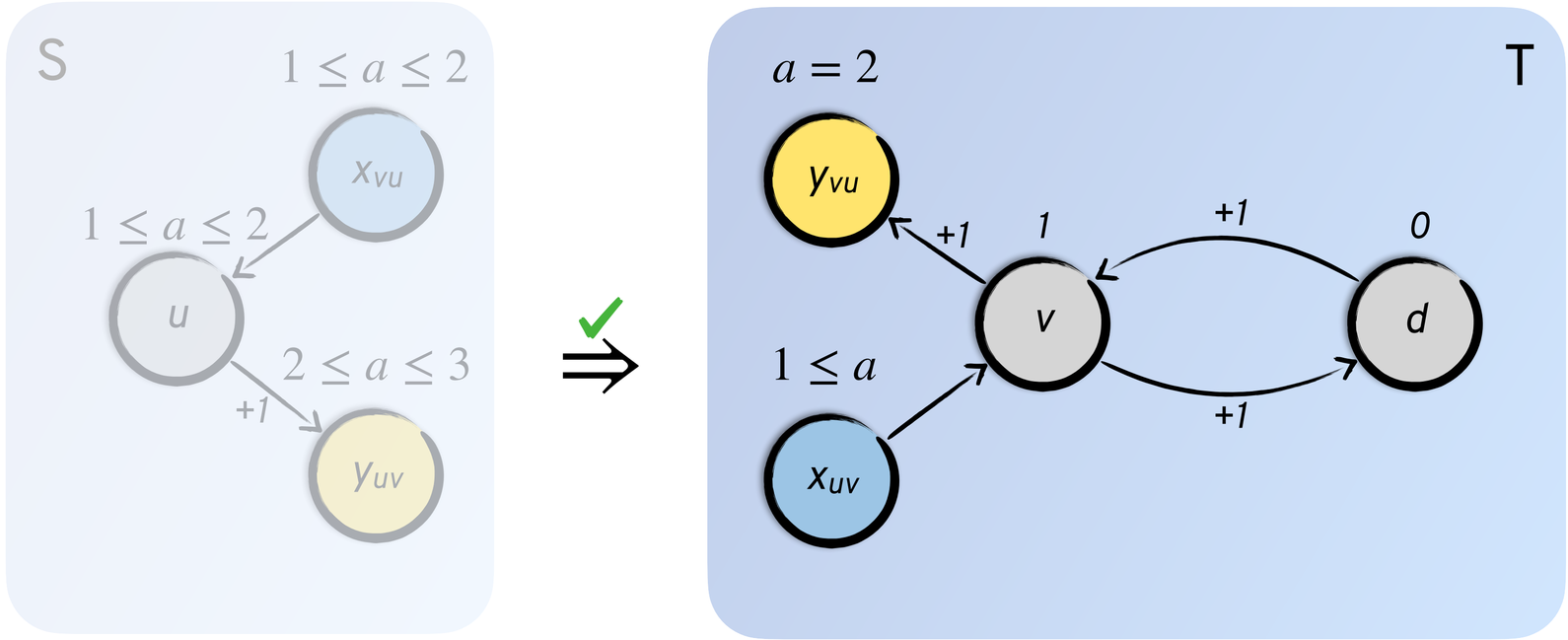
$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$



Checking an Interface

A: Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

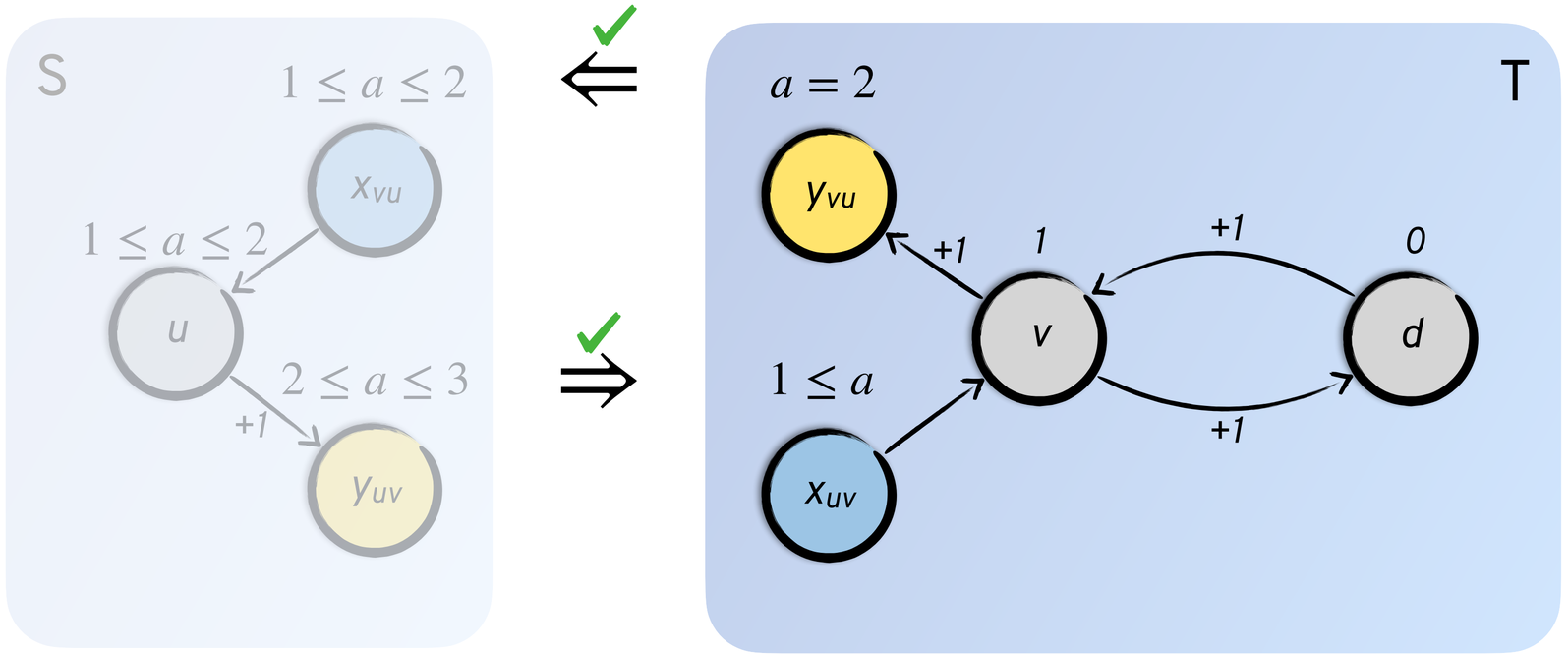
$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$



Checking an Interface

A: Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$

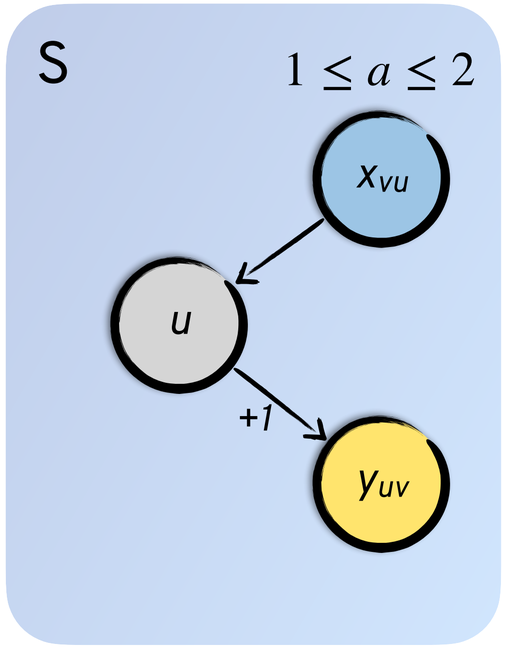


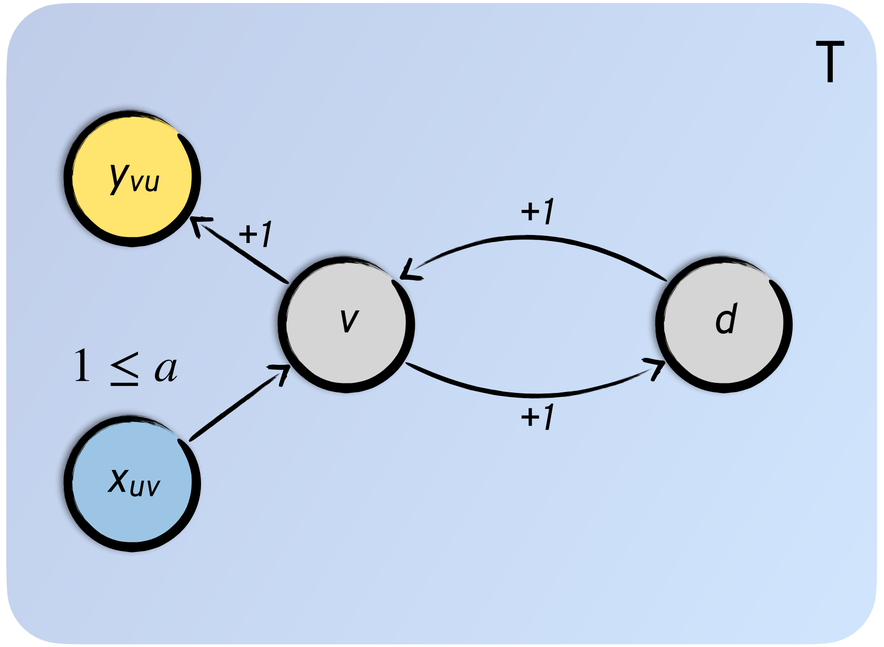
Checking an Interface

B: Safety: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle P_S \rangle$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle P_T \rangle$

$(v, u, 1 \leq a \leq 2)$

$(u, v, 1 \leq a)$





$P_S = \mathscr{L}(u) < 10$

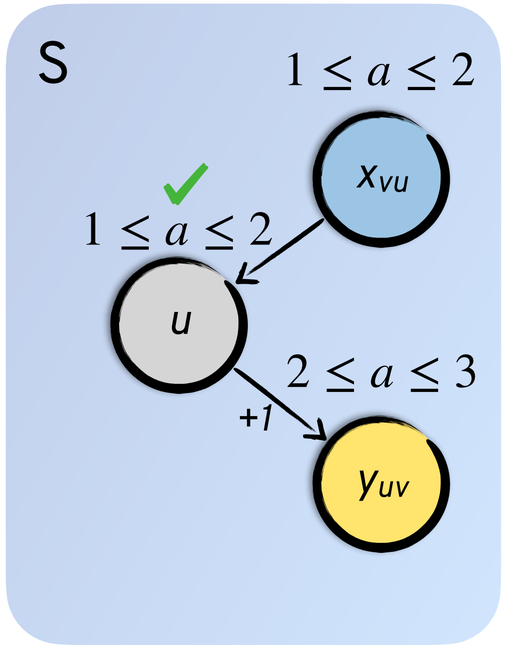
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

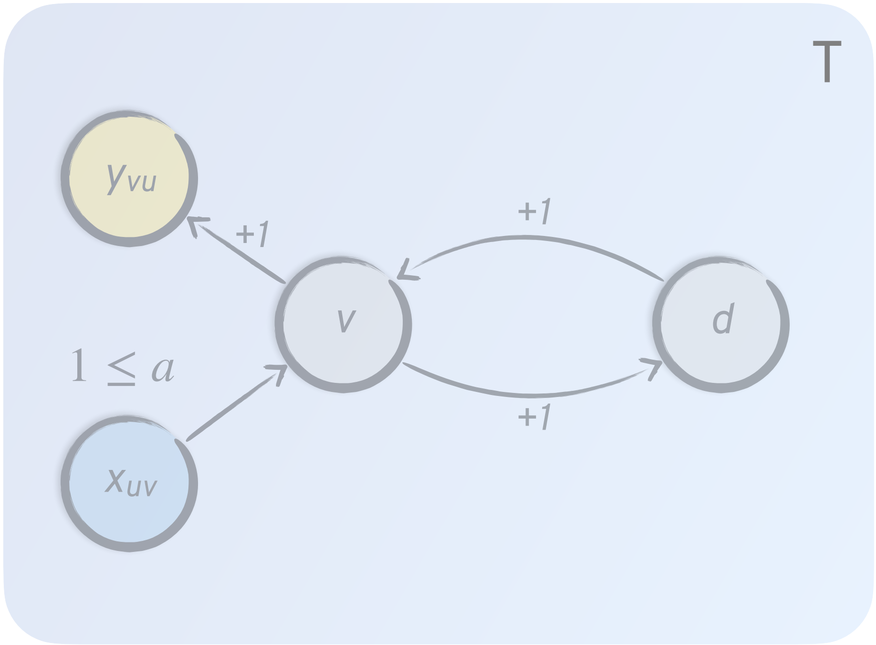
Checking an Interface

B: Safety: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle P_S \rangle$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle P_T \rangle$

$(v, u, 1 \leq a \leq 2)$

$(u, v, 1 \leq a)$





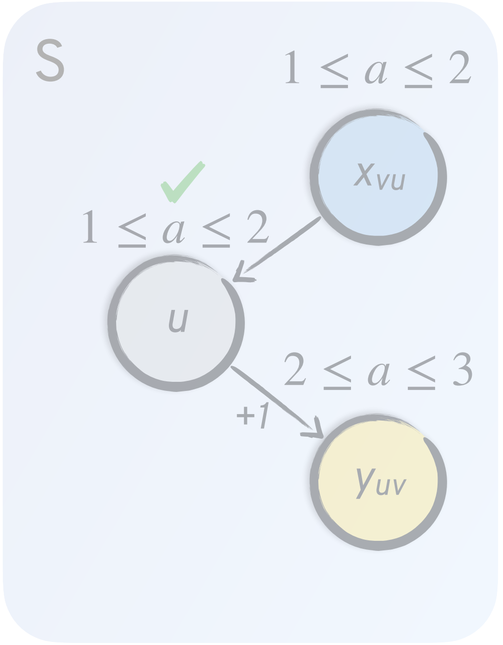
$P_S = \mathscr{L}(u) < 10$

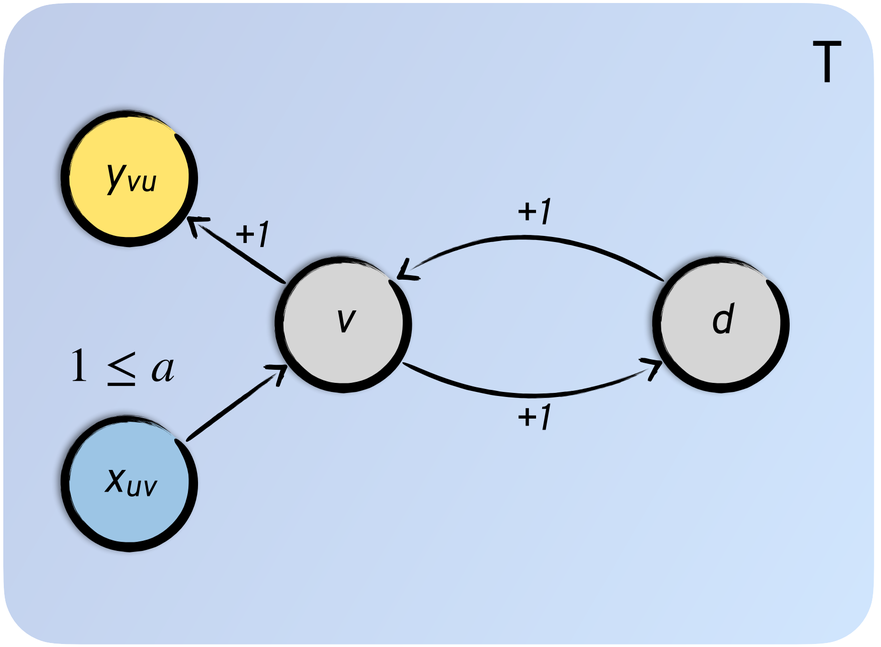
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Checking an Interface

B: Safety: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle P_S \rangle$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle P_T \rangle$

$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$





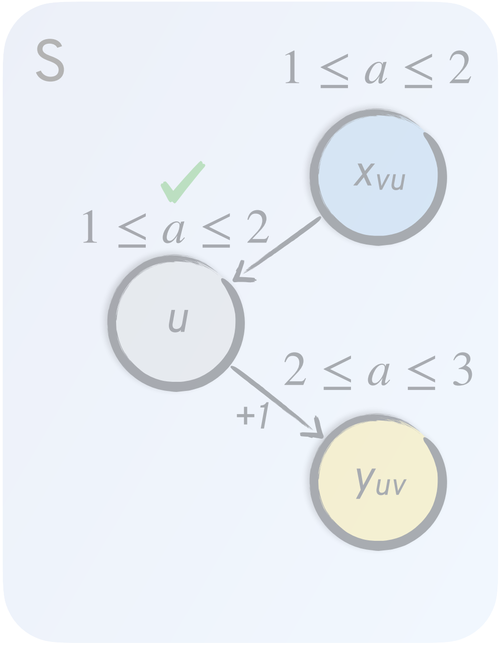
$P_S = \mathscr{L}(u) < 10$

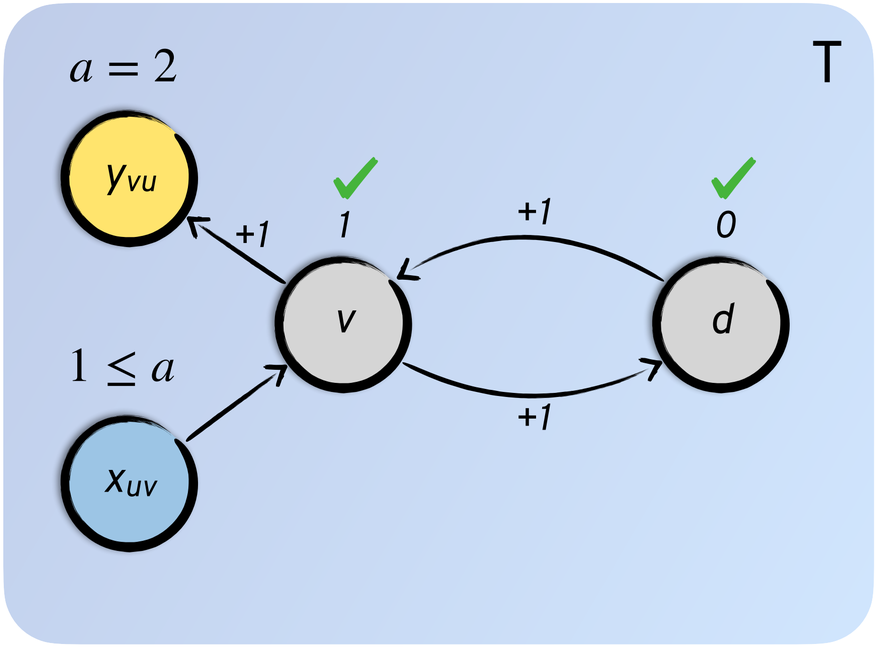
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Checking an Interface

B: Safety: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle P_S \rangle$, $\langle \mathbf{H}_T \rangle \mathscr{M}_T \langle P_T \rangle$

$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$





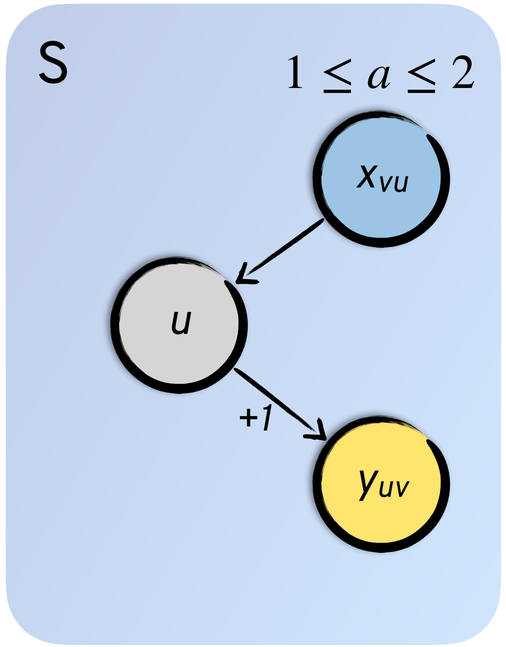
$P_S = \mathscr{L}(u) < 10$

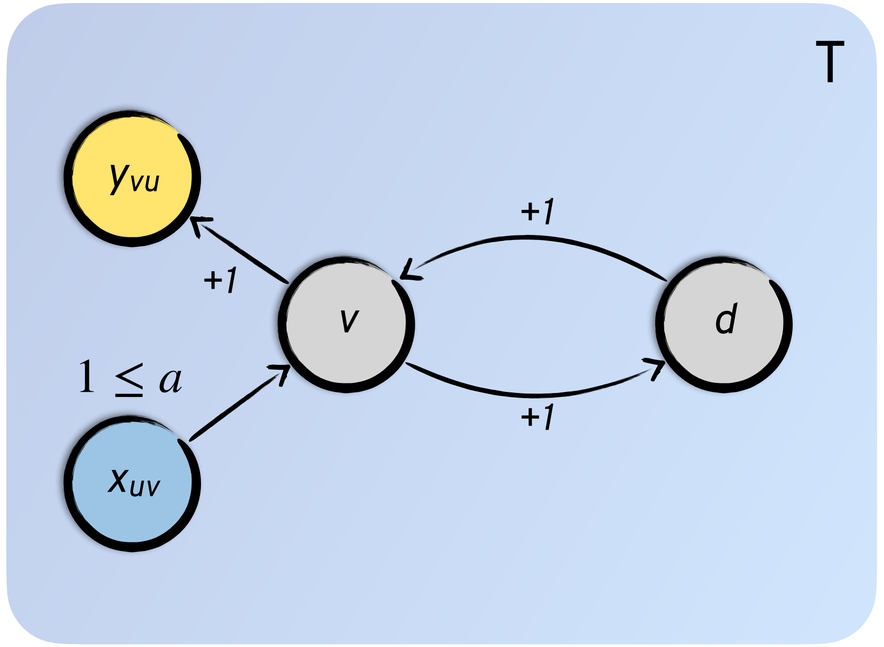
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Checking an Interface

C: Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \text{true} \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$



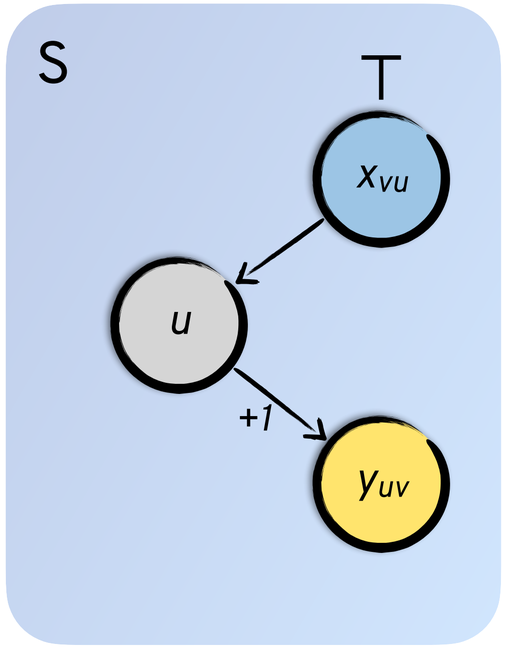


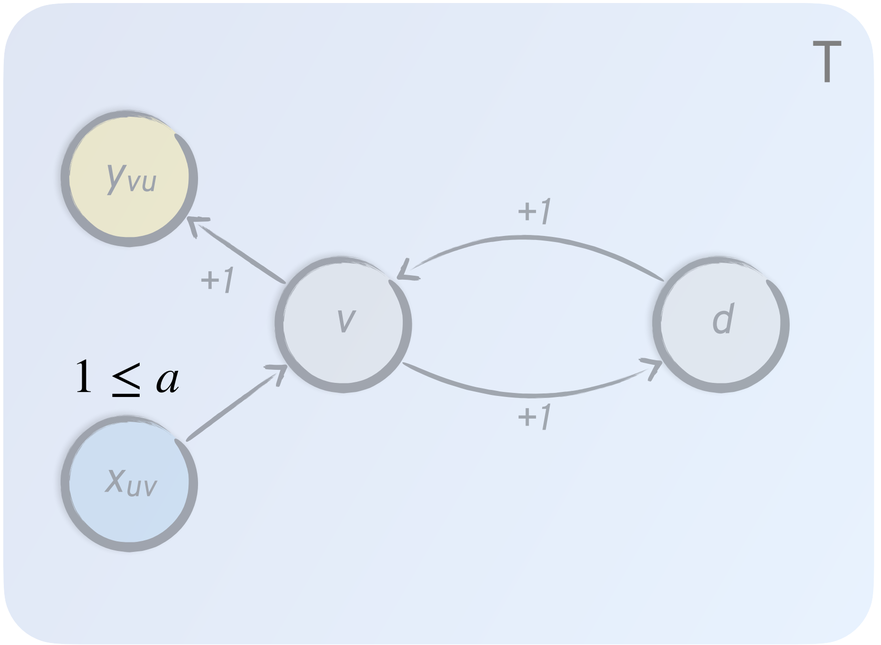
Checking an Interface

C: Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \text{true} \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

$(v, u, 1 \leq a \leq 2)$

$(u, v, 1 \leq a)$



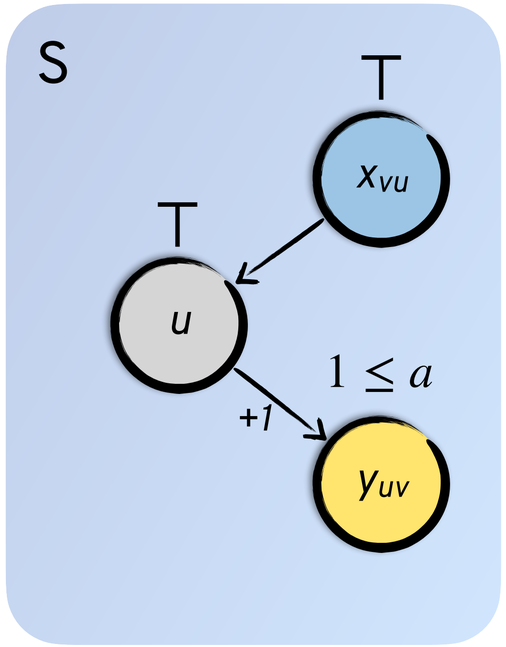


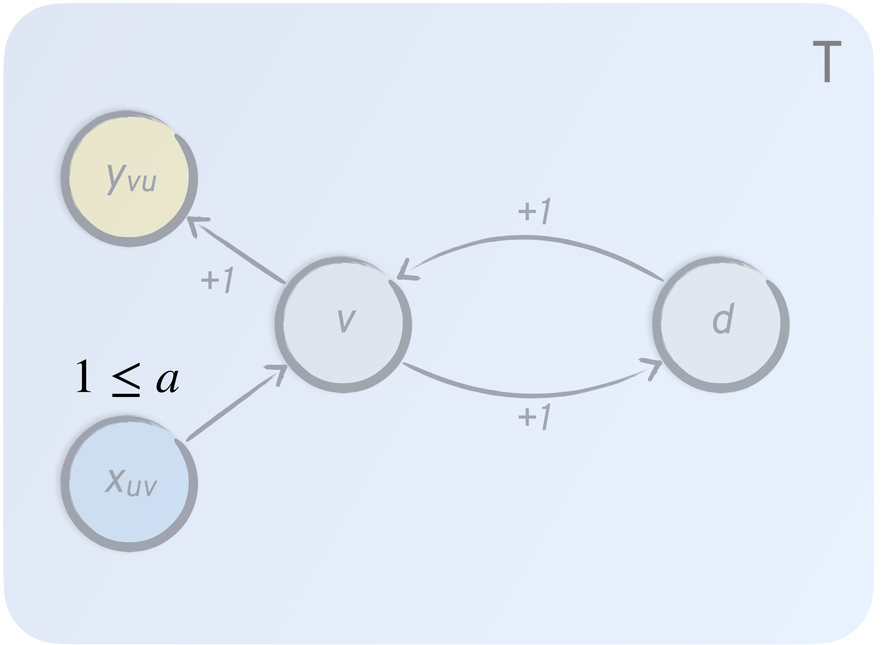
Checking an Interface

C: Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \text{true} \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

$(v, u, 1 \leq a \leq 2)$

$(u, v, 1 \leq a)$



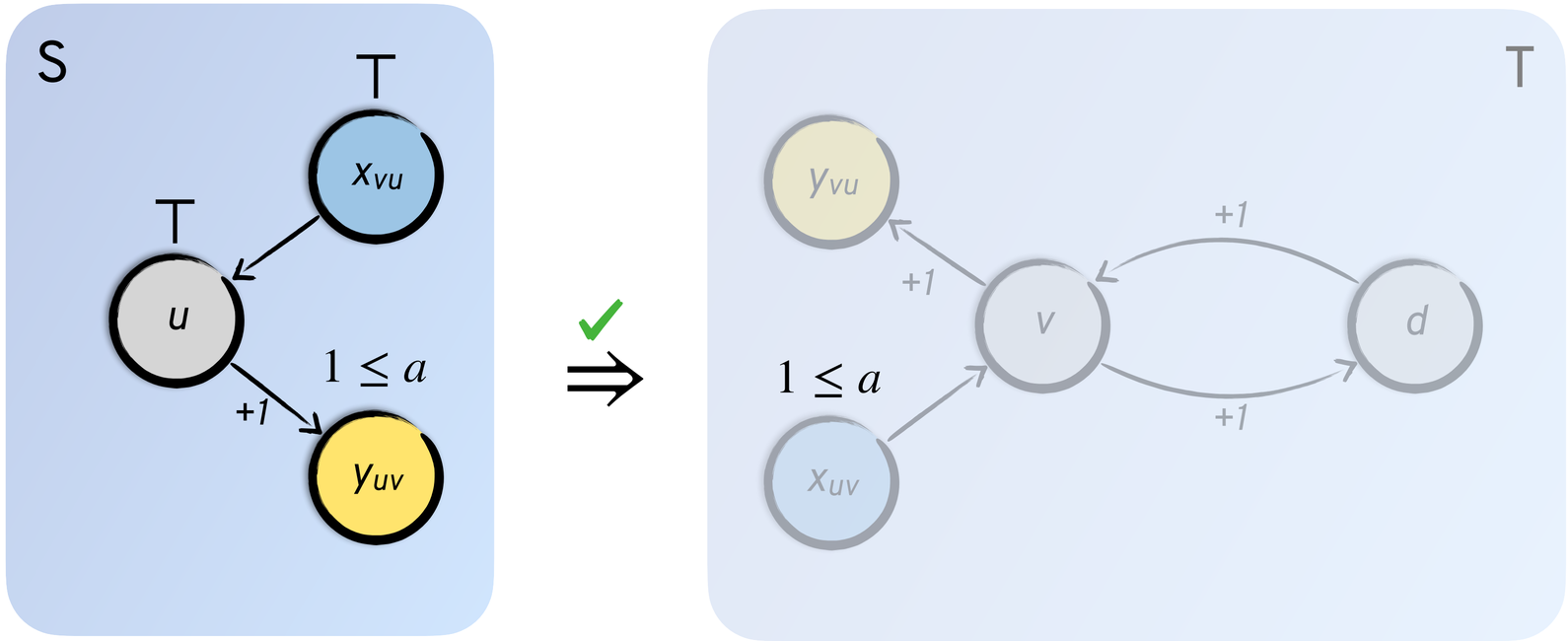


Checking an Interface

C: Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \text{true} \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

$(v, u, 1 \leq a \leq 2)$

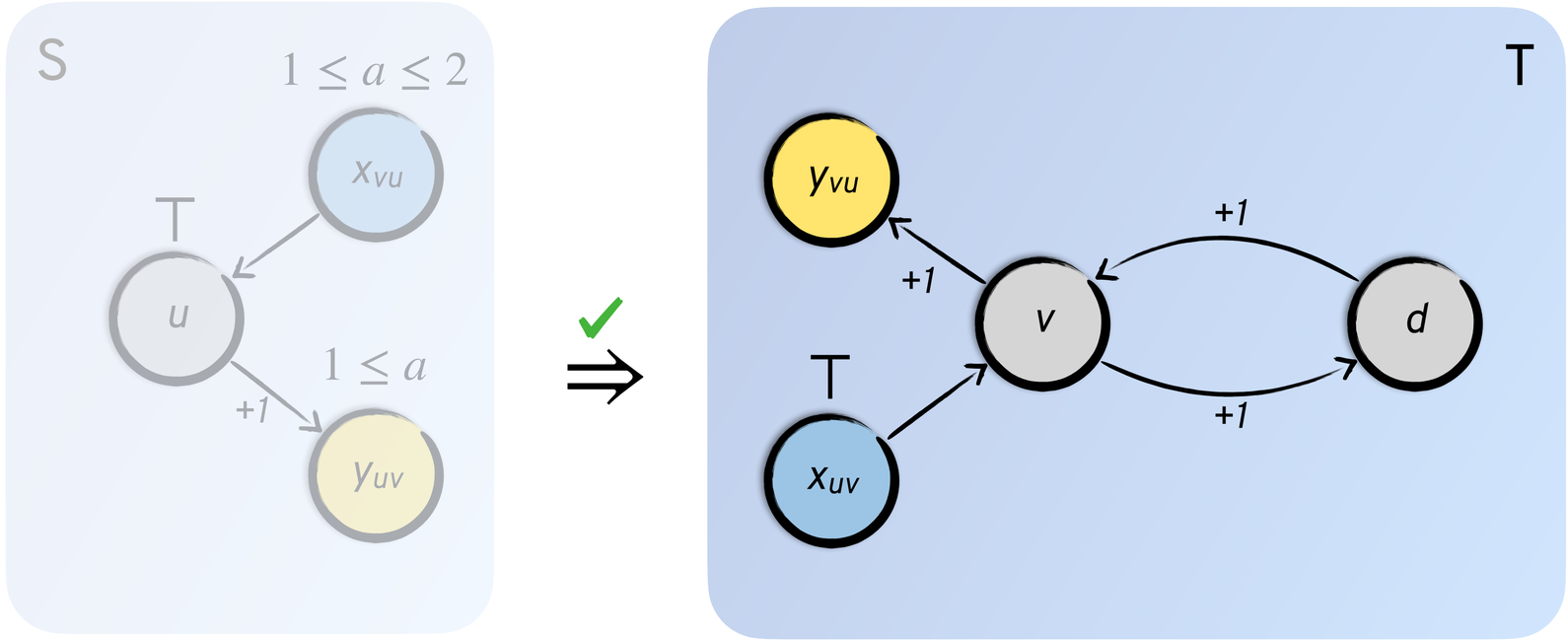
$(u, v, 1 \leq a)$



Checking an Interface

C: Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \text{true} \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$

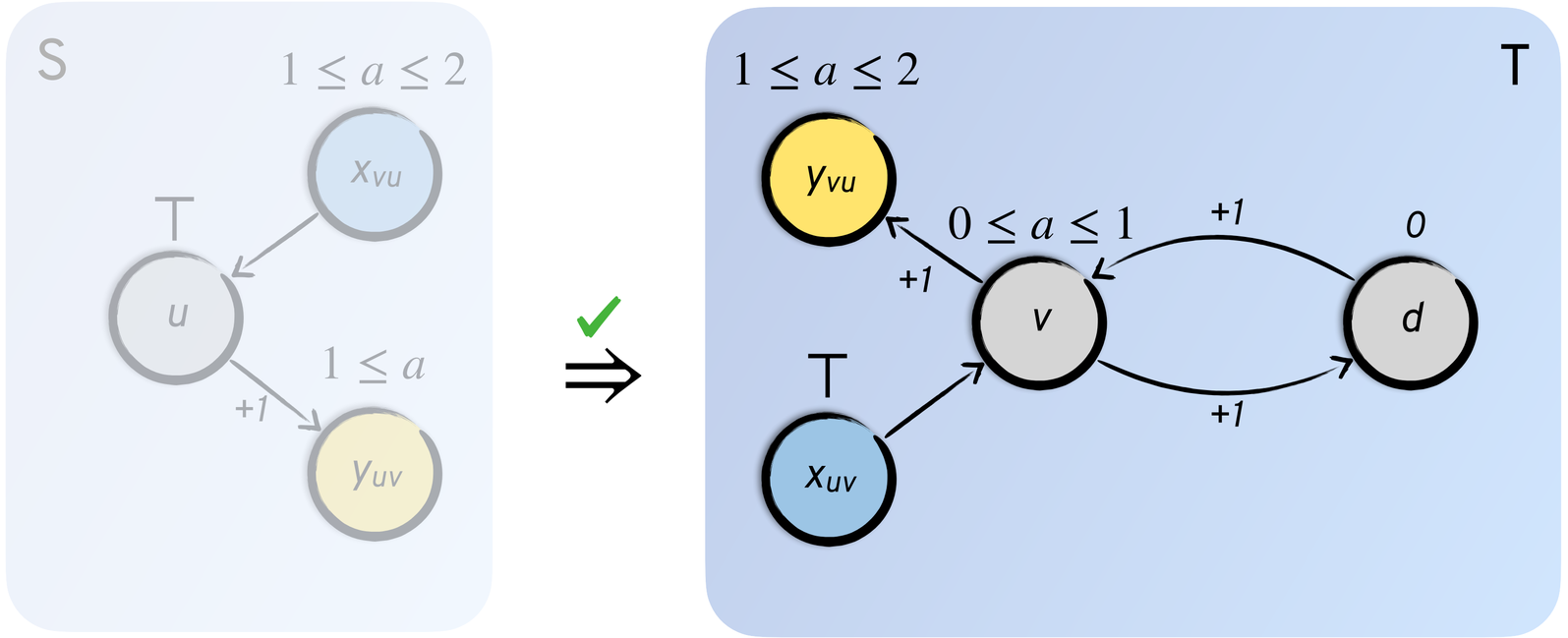


Checking an Interface

C: Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \text{true} \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

$(v, u, 1 \leq a \leq 2)$

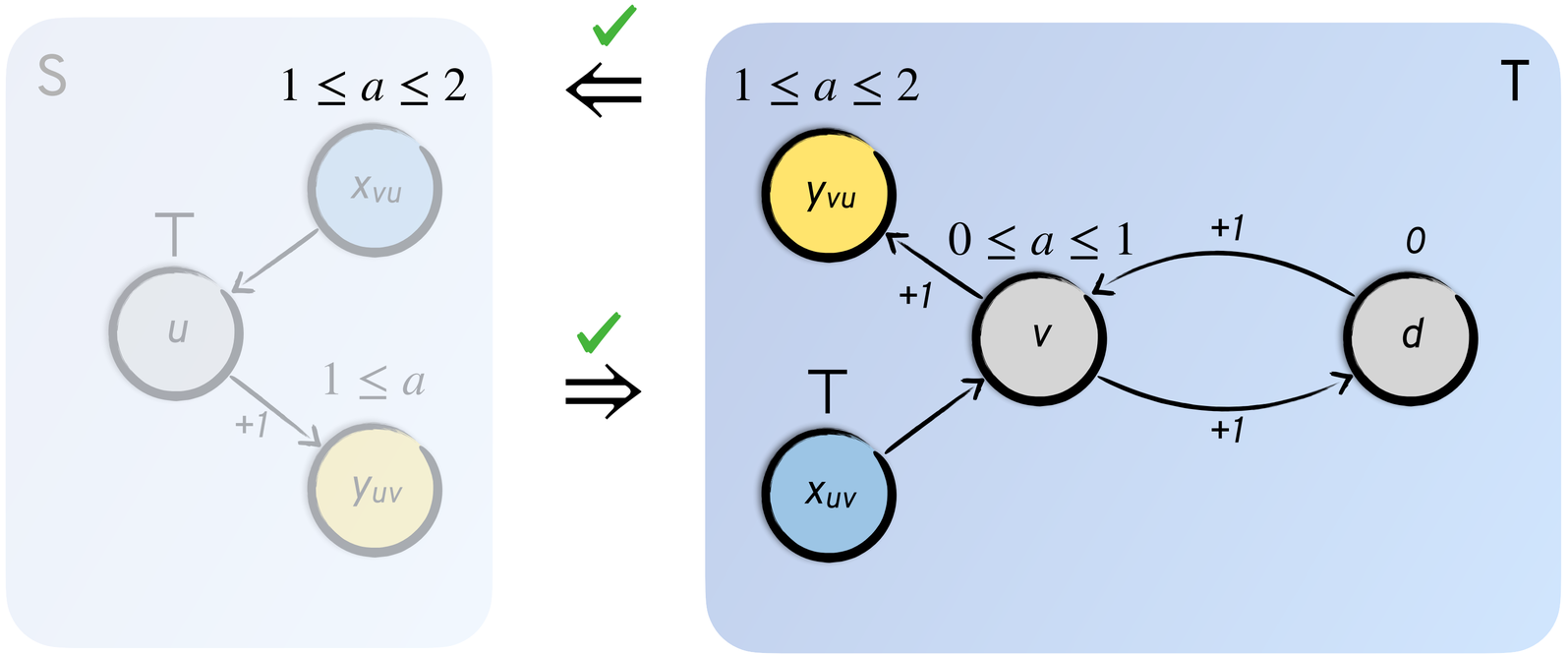
$(u, v, 1 \leq a)$



Checking an Interface

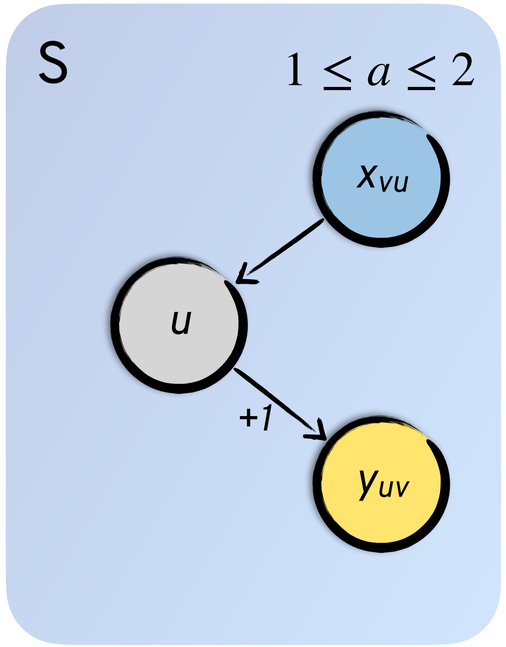
C: Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$, $\langle \text{true} \rangle \mathscr{M}_T \langle G_T \rangle \wedge G_T \Rightarrow \mathbf{H}_S$

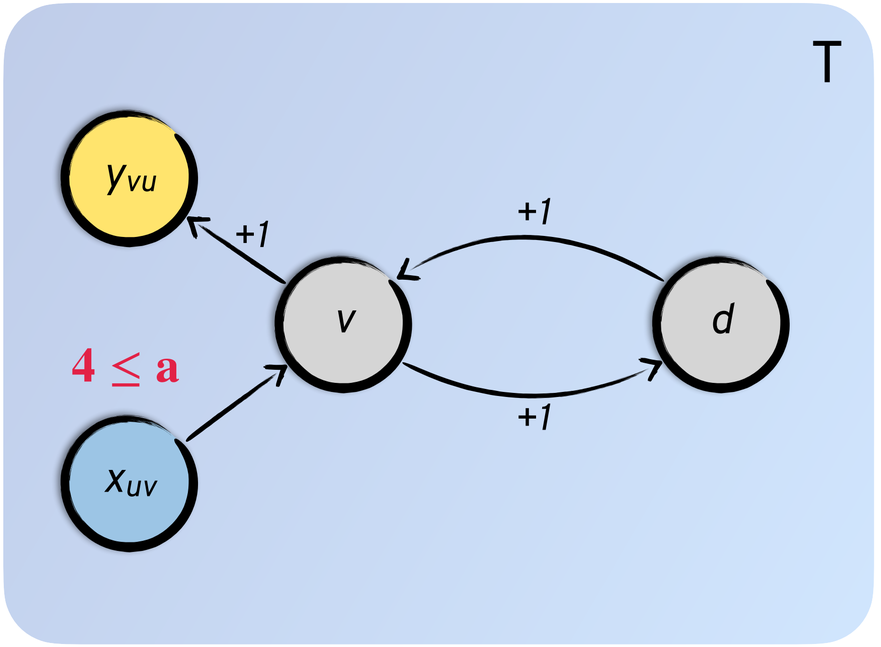
$(v, u, 1 \leq a \leq 2)$
$(u, v, 1 \leq a)$



Checking an Interface

Inductiveness VC?



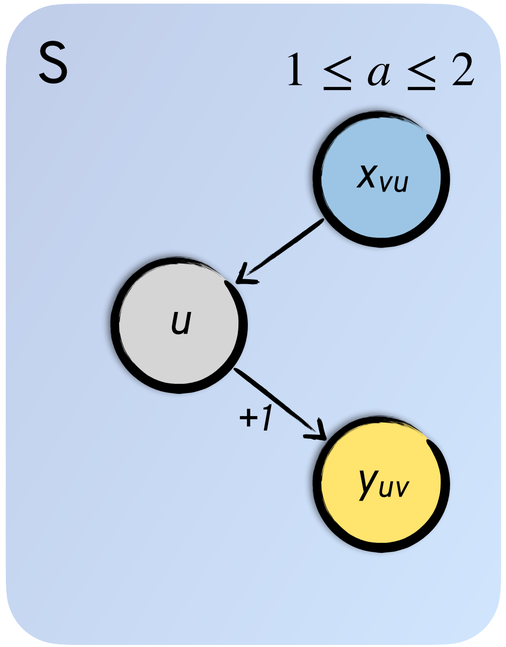


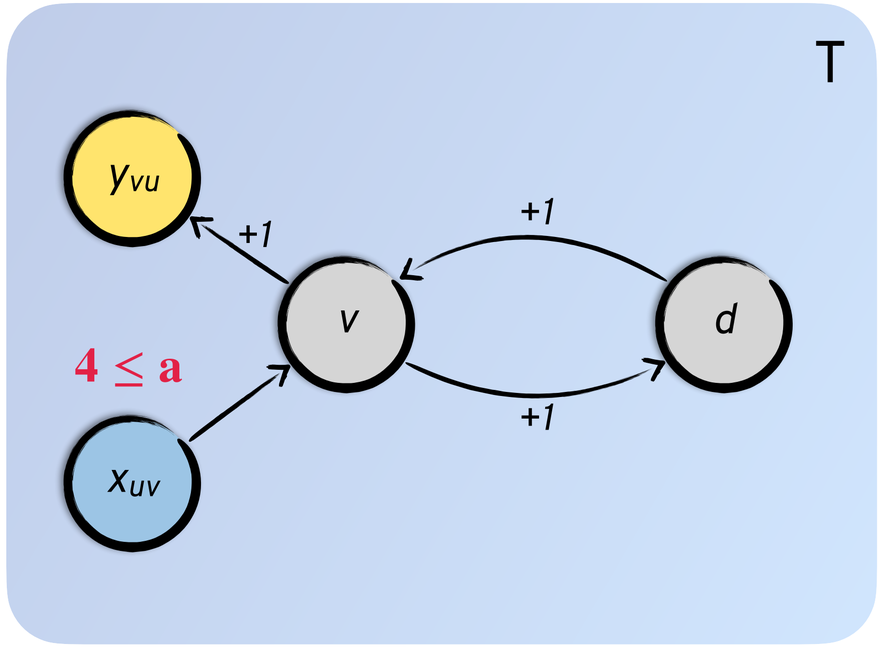
Checking an Interface

Inductiveness VC?

$(v, u, 1 \leq a \leq 2)$

$(u, v, \mathbf{4 \leq a})$





$P_S = \mathscr{L}(u) < 10$

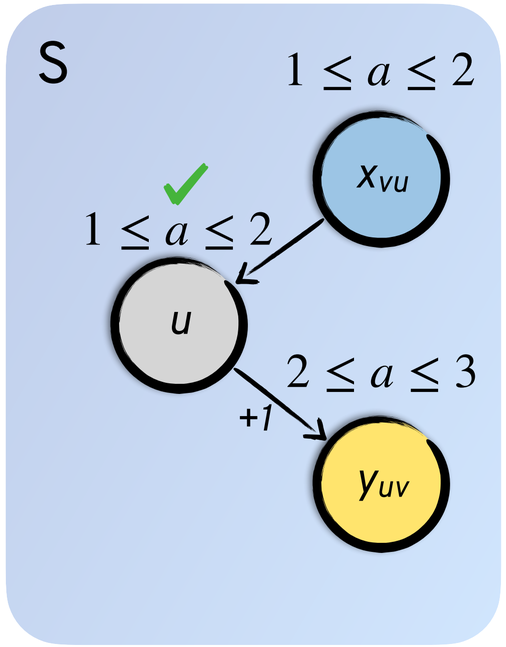
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

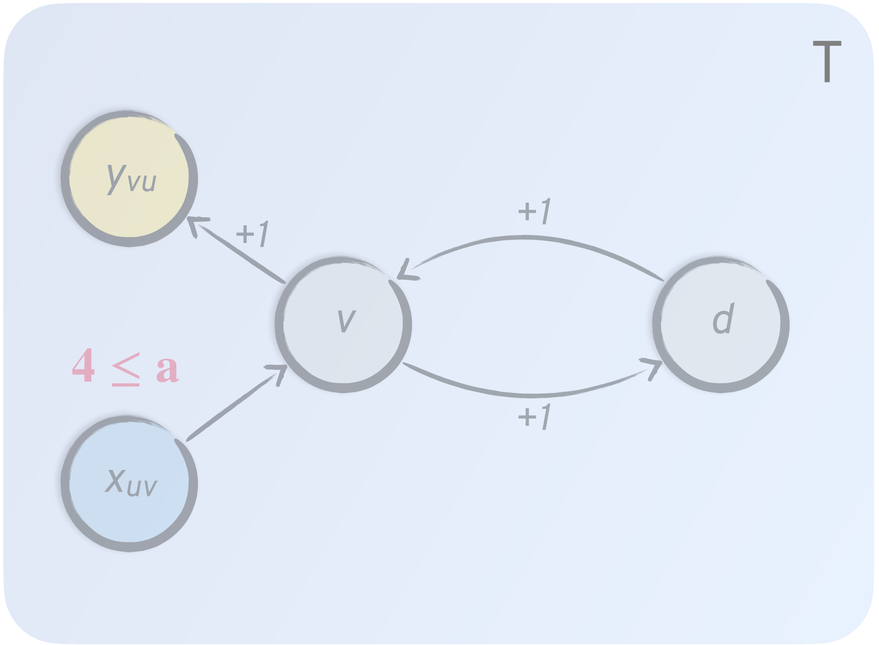
Checking an Interface

Inductiveness VC?

$(v, u, 1 \leq a \leq 2)$

$(u, v, \mathbf{4 \leq a})$





$P_S = \mathscr{L}(u) < 10$

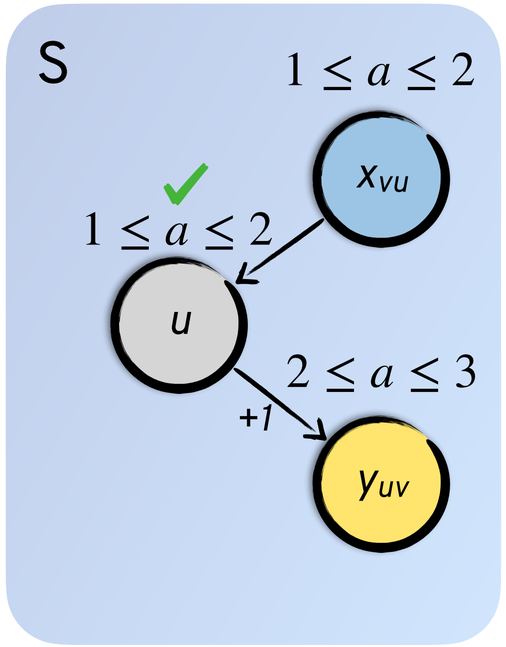
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

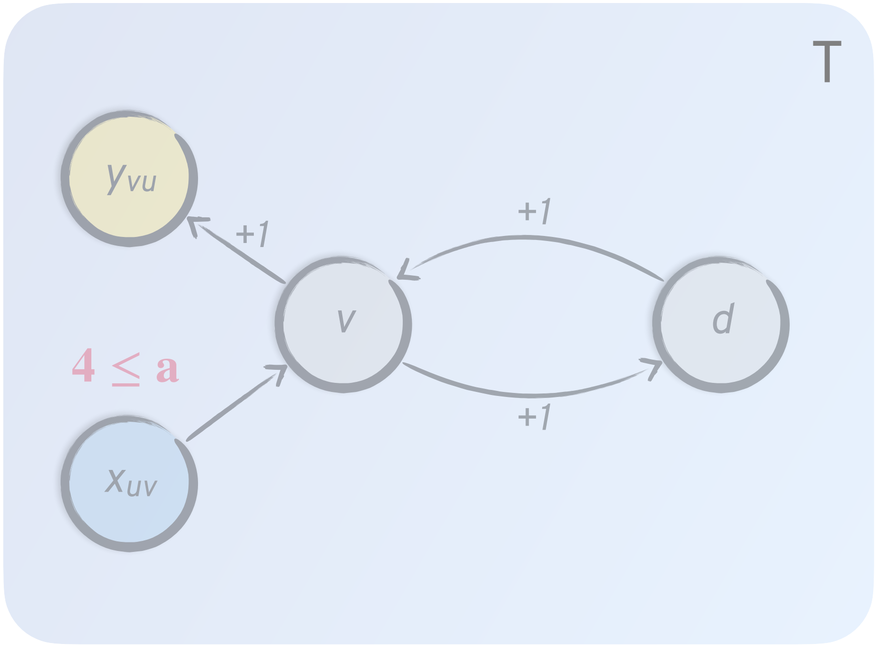
Checking an Interface

Inductiveness VC?

$(v, u, 1 \leq a \leq 2)$

$(u, v, \mathbf{4 \leq a})$





$P_S = \mathscr{L}(u) < 10$

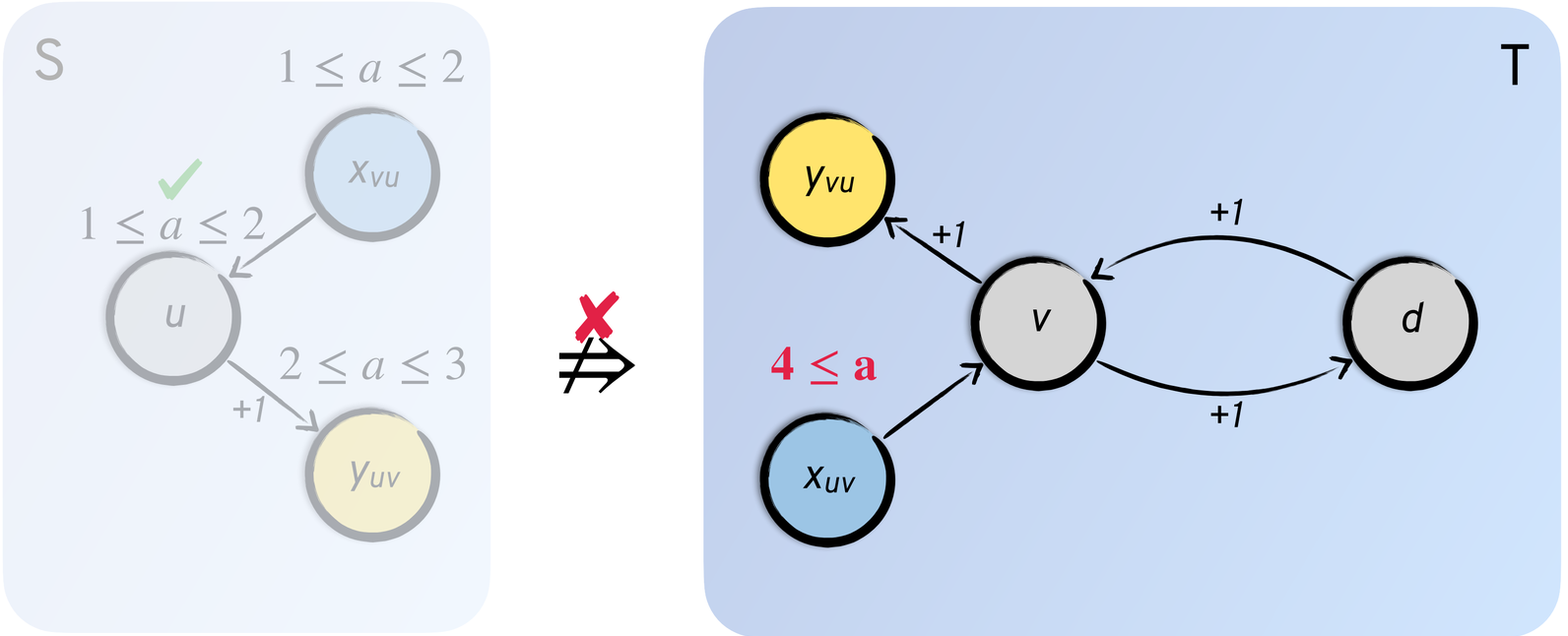
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Checking an Interface

Inductiveness VC?

$(v, u, 1 \leq a \leq 2)$

$(u, v, \mathbf{4 \leq a})$



$P_S = \mathscr{L}(u) < 10$

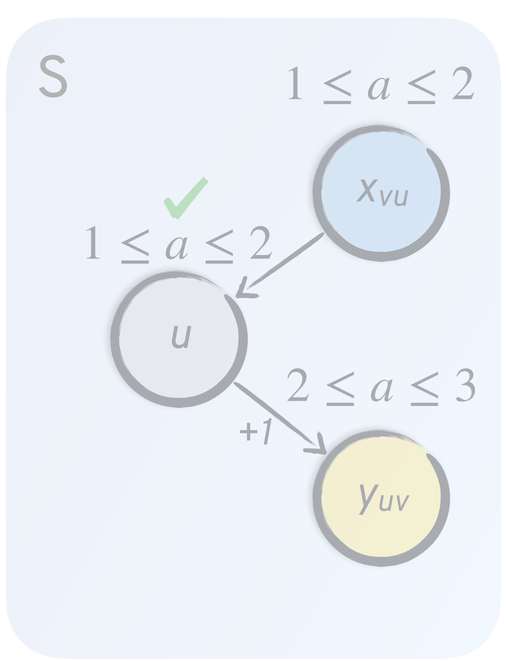
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

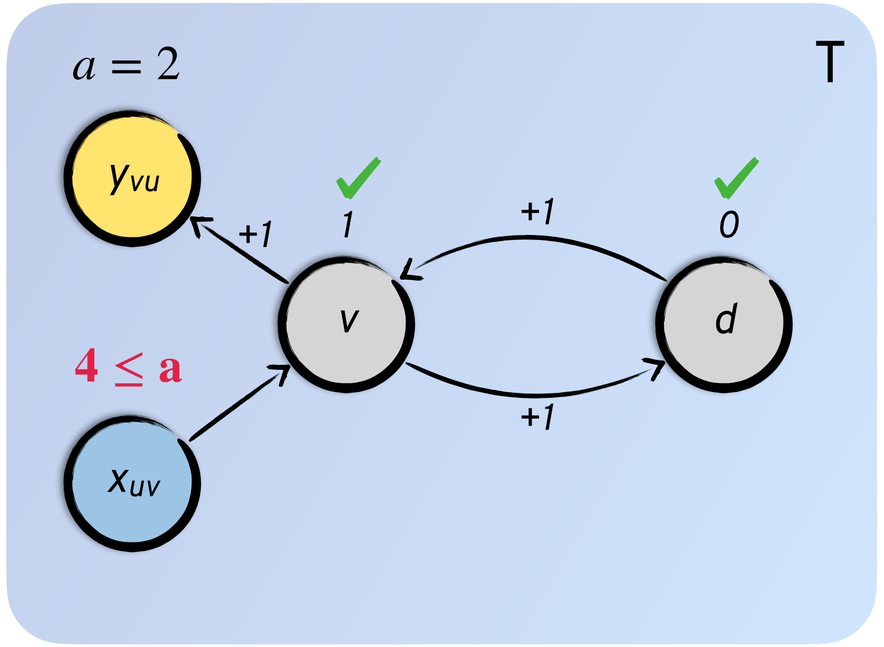
Checking an Interface

Inductiveness VC?

$(v, u, 1 \leq a \leq 2)$

$(u, v, \mathbf{4 \leq a})$





$P_S = \mathscr{L}(u) < 10$

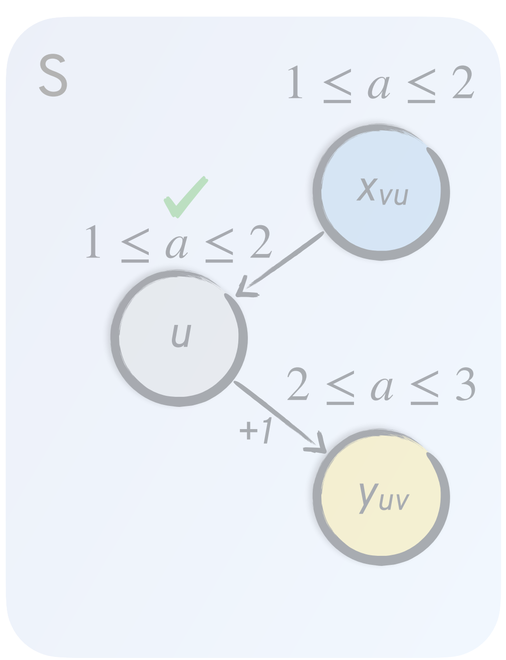
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

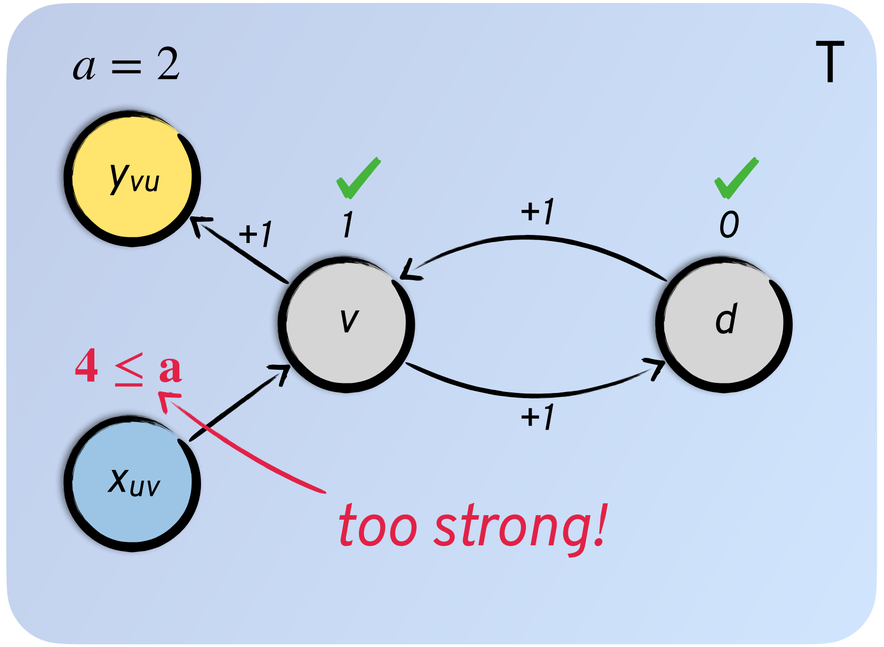
Checking an Interface

Inductiveness VC?

$(v, u, 1 \leq a \leq 2)$

$(u, v, \mathbf{4 \leq a})$





$P_S = \mathscr{L}(u) < 10$

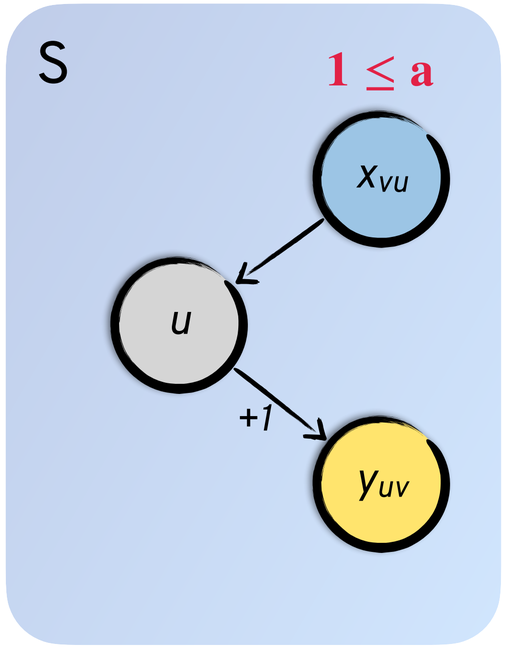
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

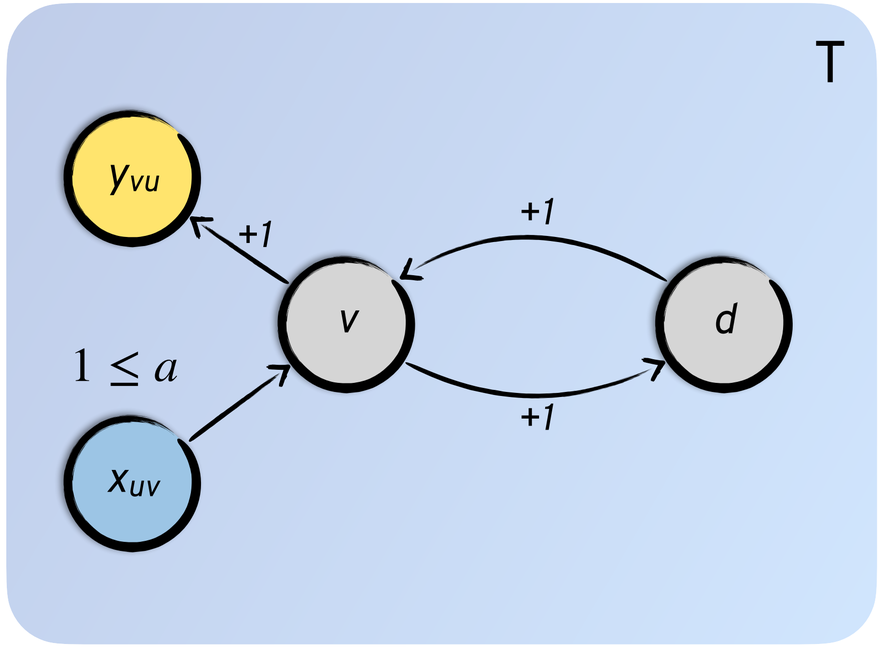
Checking an Interface

Safety VC?

$(v, u, \mathbf{1 \leq a})$

$(u, v, 1 \leq a)$





$P_S = \mathscr{L}(u) < 10$

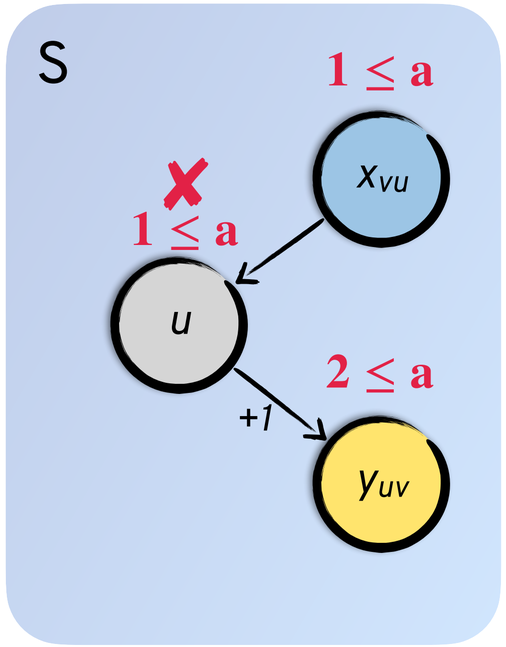
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

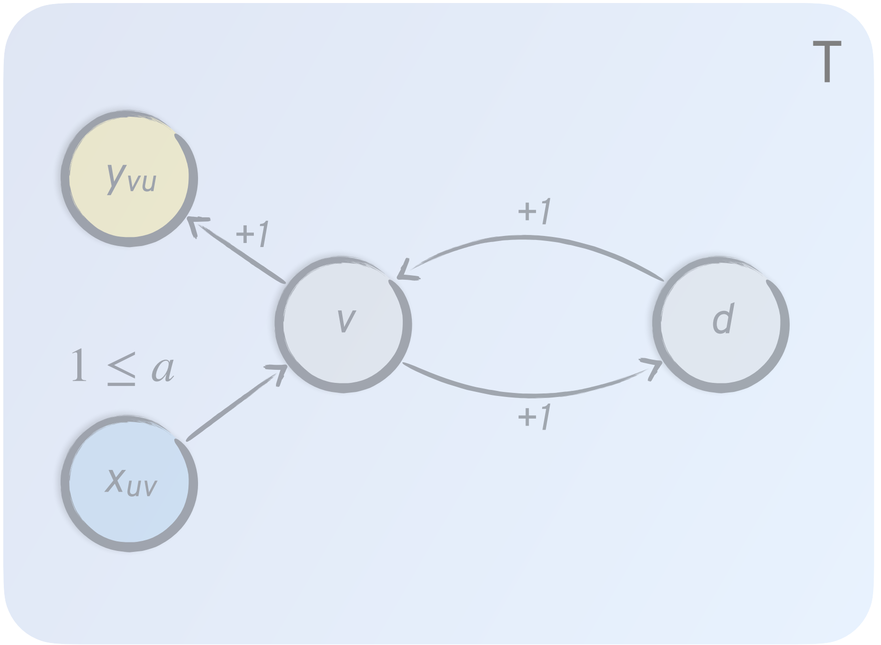
Checking an Interface

Safety VC?

$(v, u, \mathbf{1 \leq a})$

$(u, v, 1 \leq a)$





$P_S = \mathscr{L}(u) < 10$

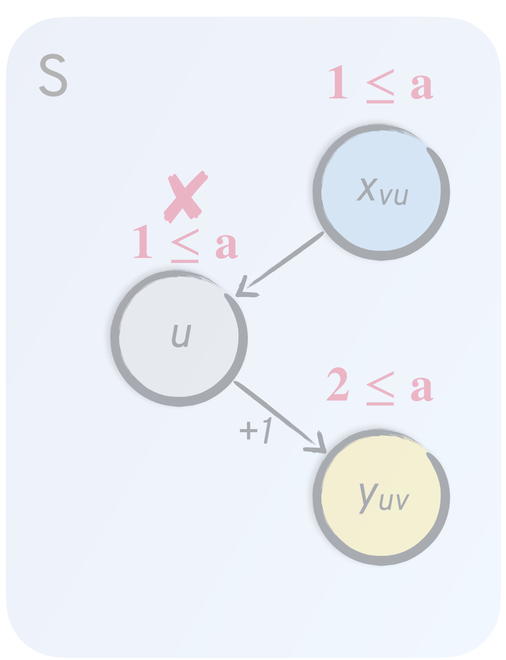
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

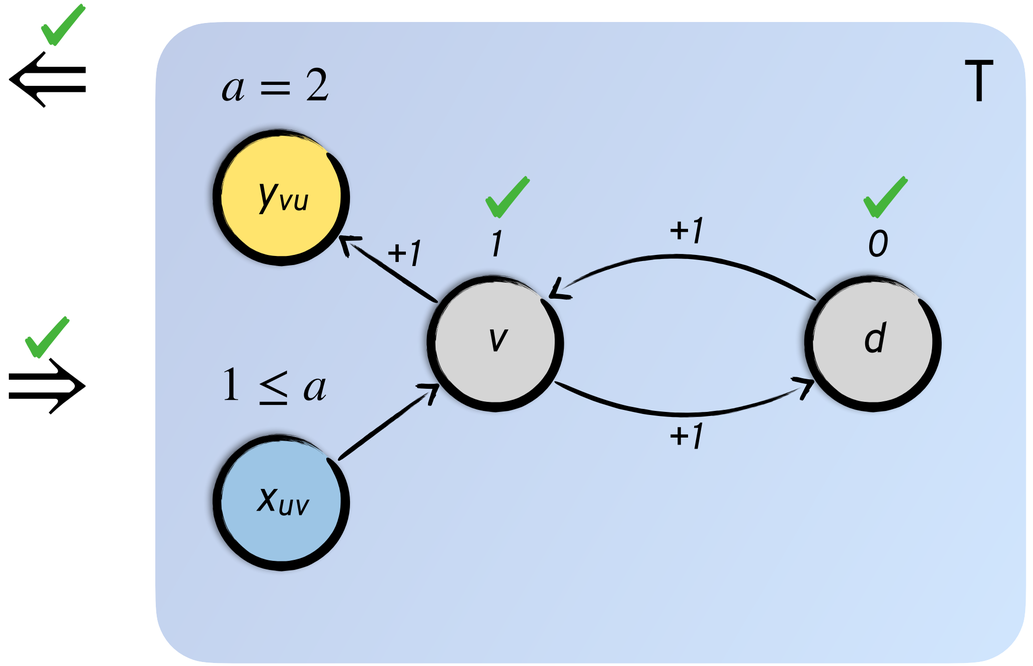
Checking an Interface

Safety VC?

$(v, u, \mathbf{1 \leq a})$

$(u, v, 1 \leq a)$





$P_S = \mathscr{L}(u) < 10$

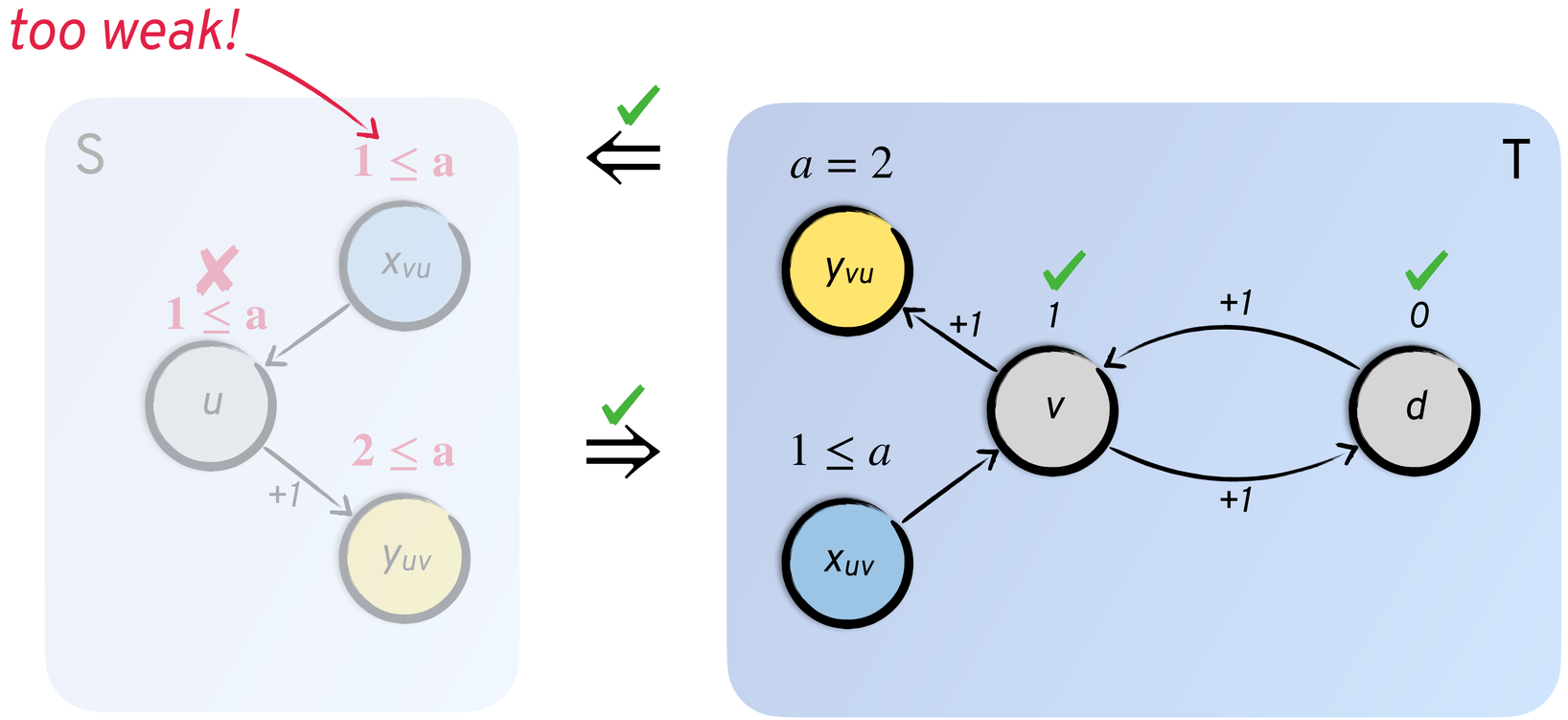
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Checking an Interface

Safety VC?

$(v, u, \mathbf{1 \leq a})$

$(u, v, 1 \leq a)$



$P_S = \mathscr{L}(u) < 10$

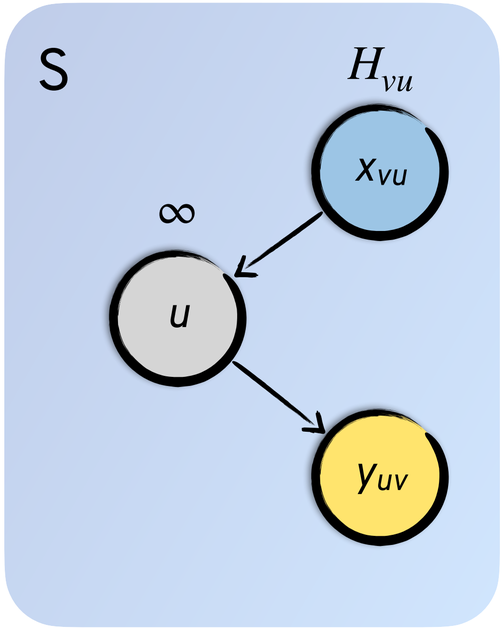
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Checking an Interface

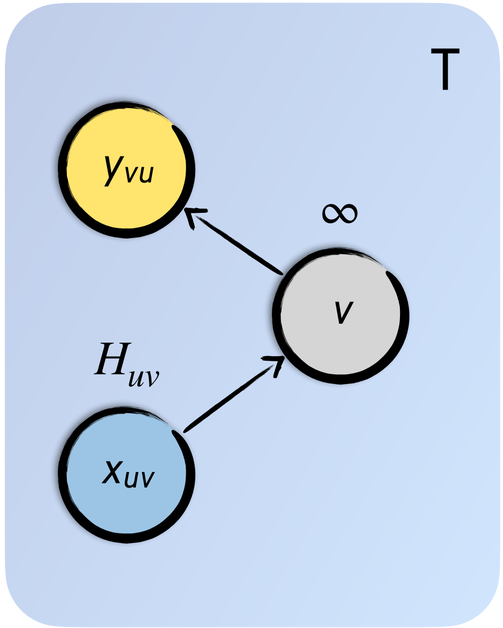
Networks with Multiple Solutions

$(v, u, a = \infty)$

$(u, v, a = \infty)$



$P_S = true$



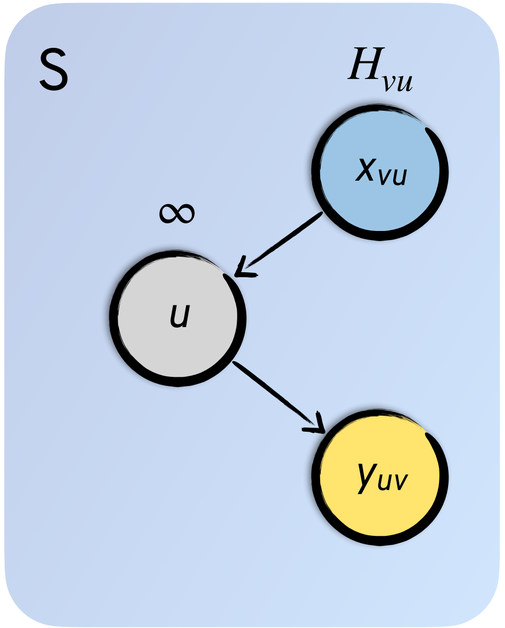
$P_T = true$

Checking an Interface

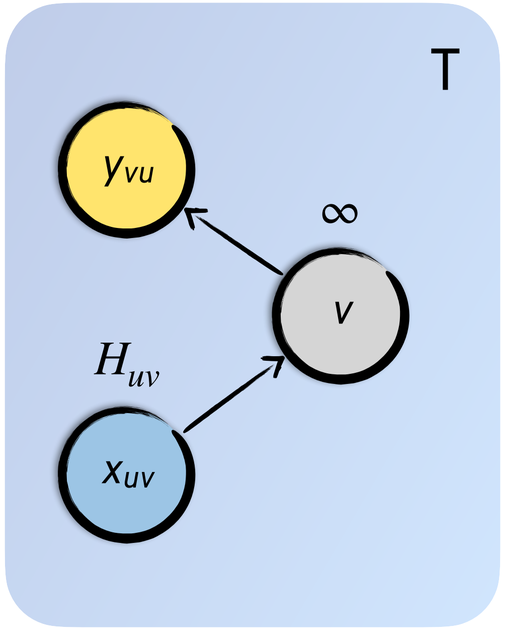
Networks with Multiple Solutions

$(v, u, a = \infty)$

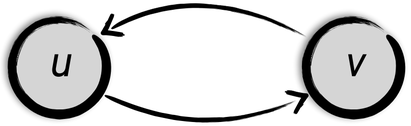
$(u, v, a = \infty)$



$P_S = true$



$P_T = true$

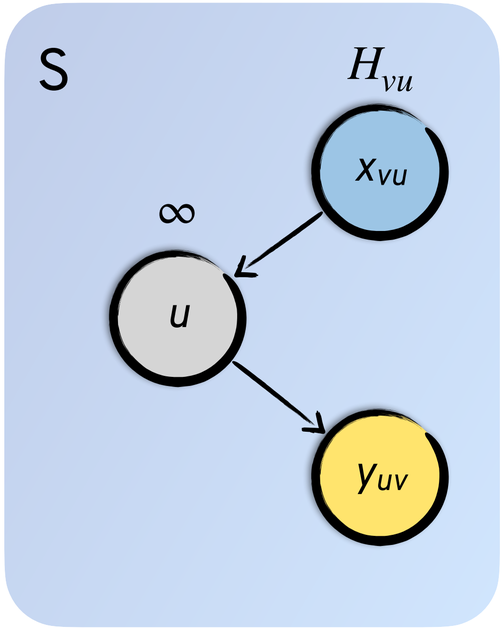


Checking an Interface

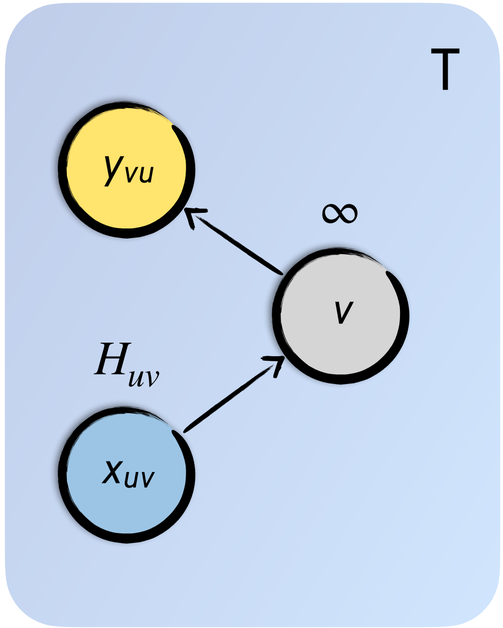
Networks with Multiple Solutions

$(v, u, a = \infty)$

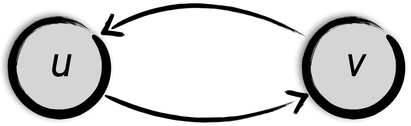
$(u, v, a = \infty)$



$P_S = true$



$P_T = true$



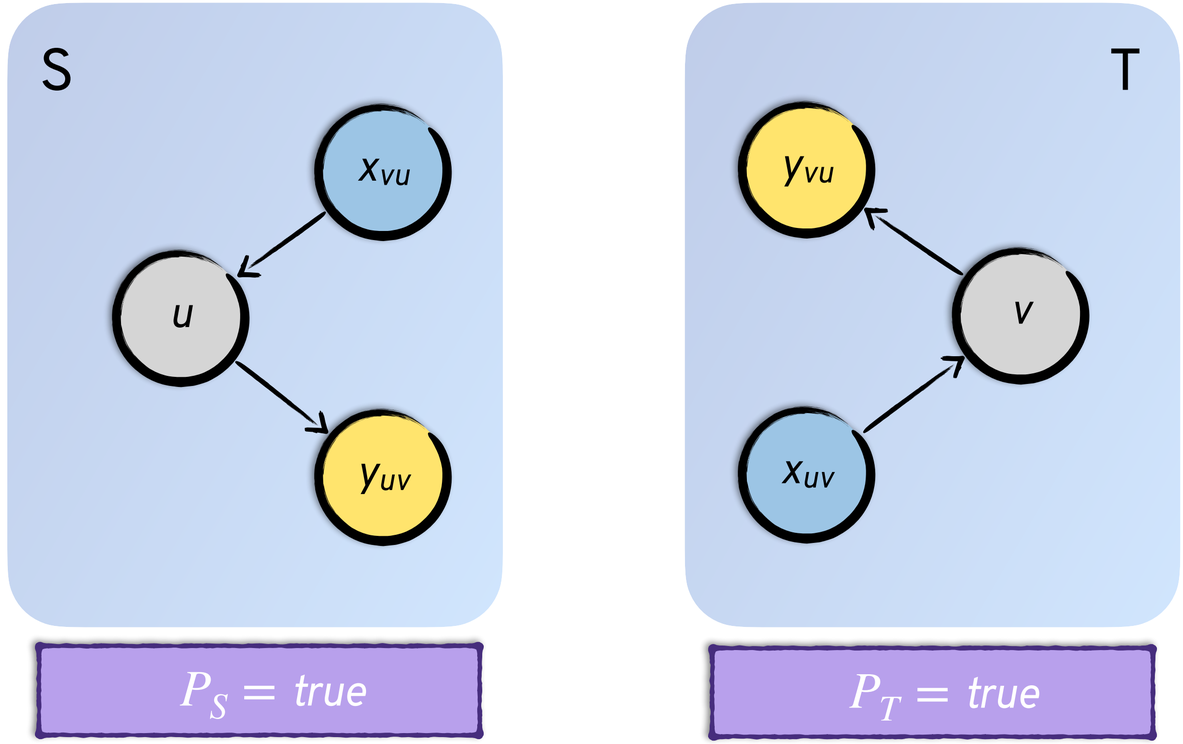
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\cdots$$
$$= \infty$$

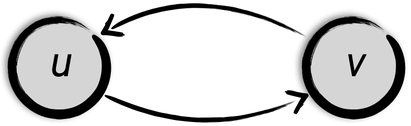
Checking an Interface

Networks with Multiple Solutions

$(v, u, a = \infty)$

$(u, v, a = \infty)$





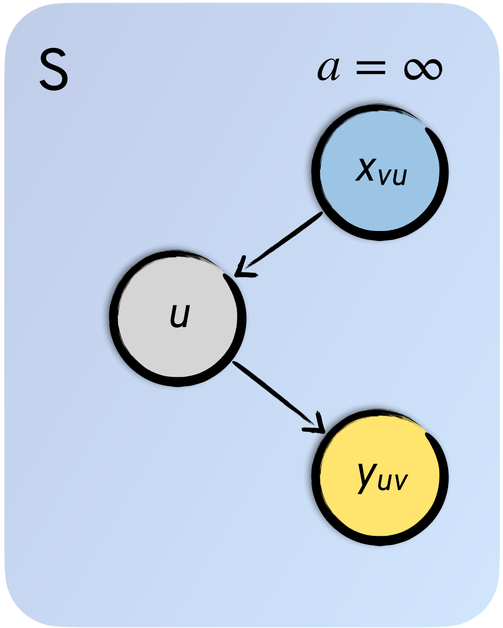
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\cdots$$
$$= \infty$$

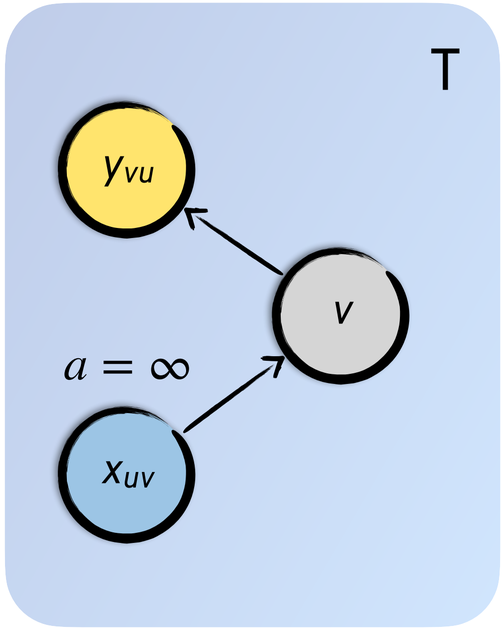
Checking an Interface

Networks with Multiple Solutions

$(v, u, a = \infty)$

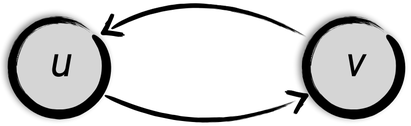
$(u, v, a = \infty)$





$P_S = true$

$P_T = true$



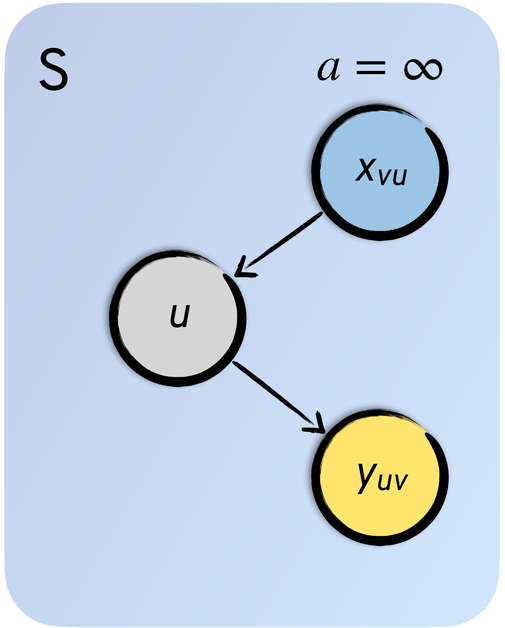
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\bullet \bullet \bullet$$
$$= \infty$$

Checking an Interface

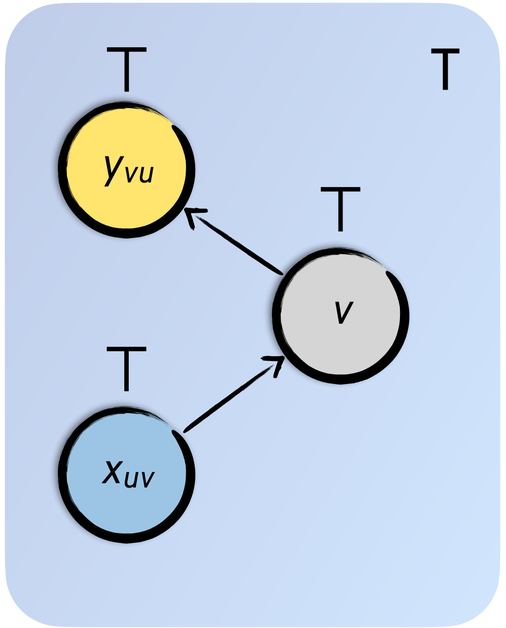
Networks with Multiple Solutions

$(v, u, a = \infty)$

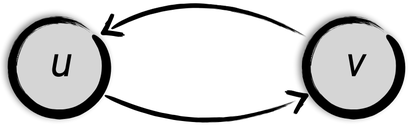
$(u, v, a = \infty)$



$P_S = true$



$P_T = true$



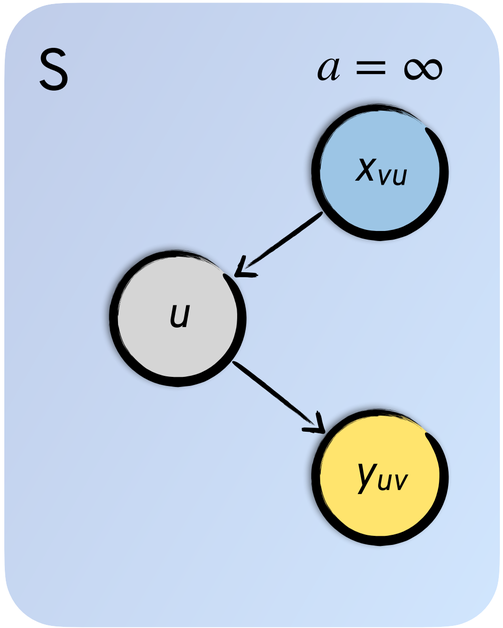
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\cdots$$
$$= \infty$$

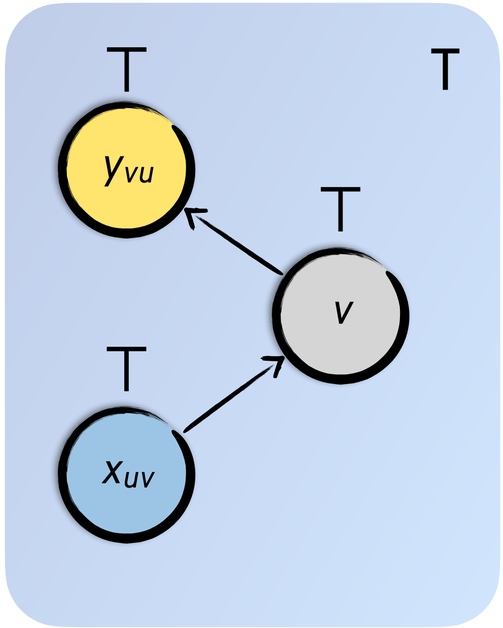
Checking an Interface

Networks with Multiple Solutions

$(v, u, a = \infty)$

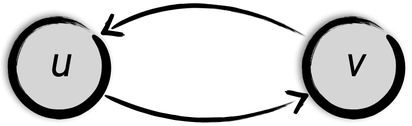
$(u, v, a = \infty)$





$P_S = true$

$P_T = true$



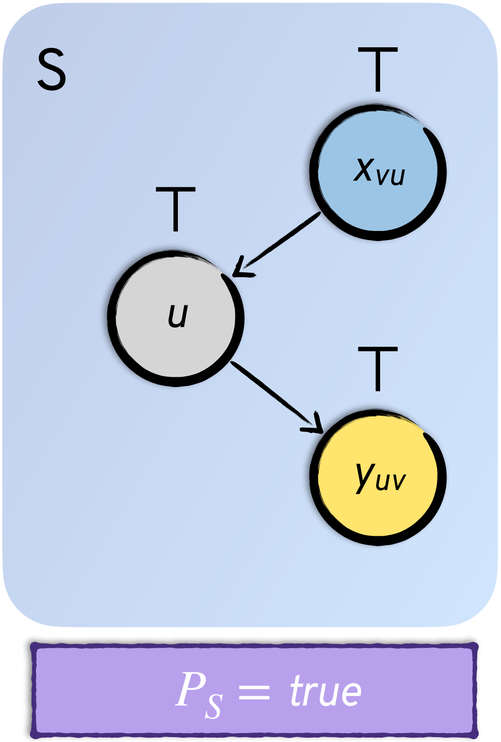
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\cdots$$
$$= \infty$$

Checking an Interface

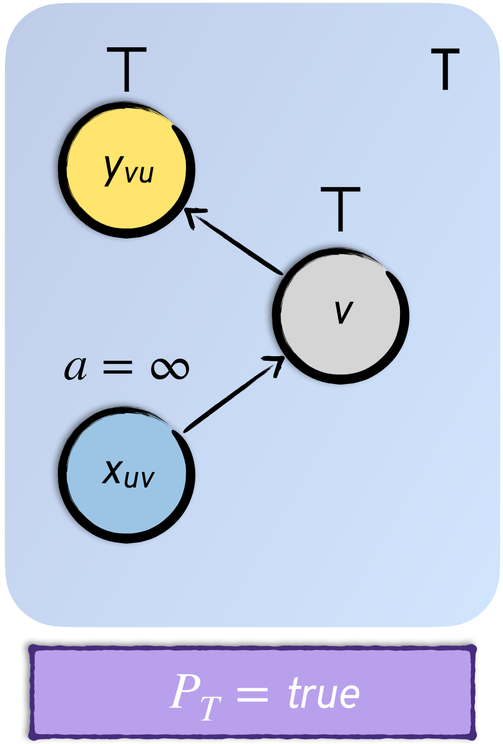
Networks with Multiple Solutions

$(v, u, a = \infty)$

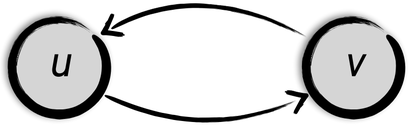
$(u, v, a = \infty)$



$P_S = true$



$P_T = true$



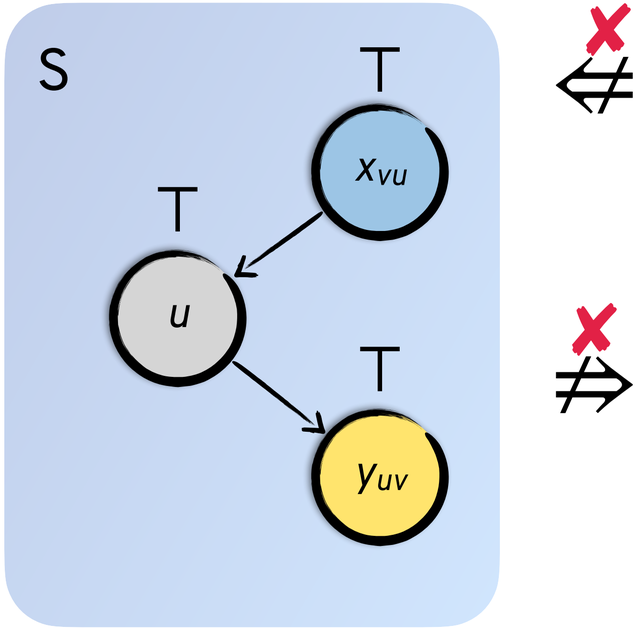
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\bullet \bullet \bullet$$
$$= \infty$$

Checking an Interface

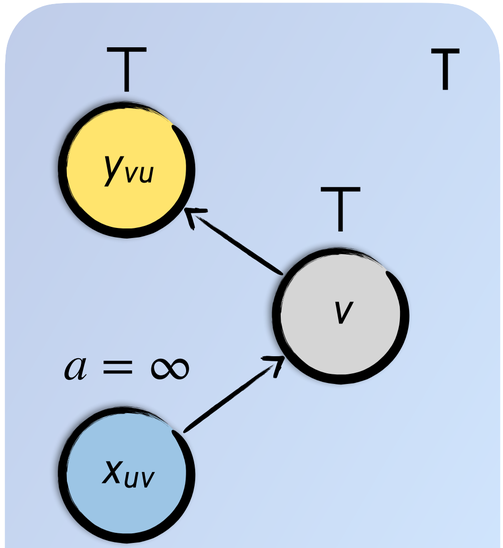
Networks with Multiple Solutions

$(v, u, a = \infty)$

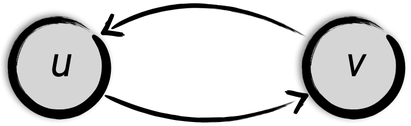
$(u, v, a = \infty)$



$P_S = true$



$P_T = true$



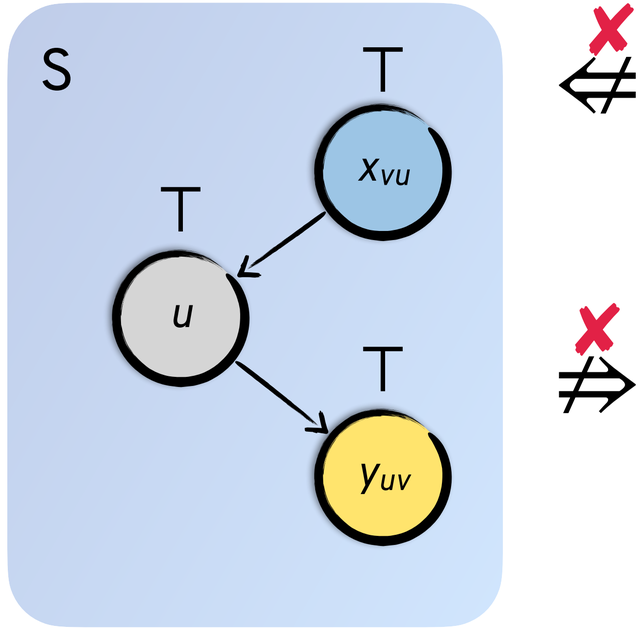
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\cdots$$
$$= \infty$$

Checking an Interface

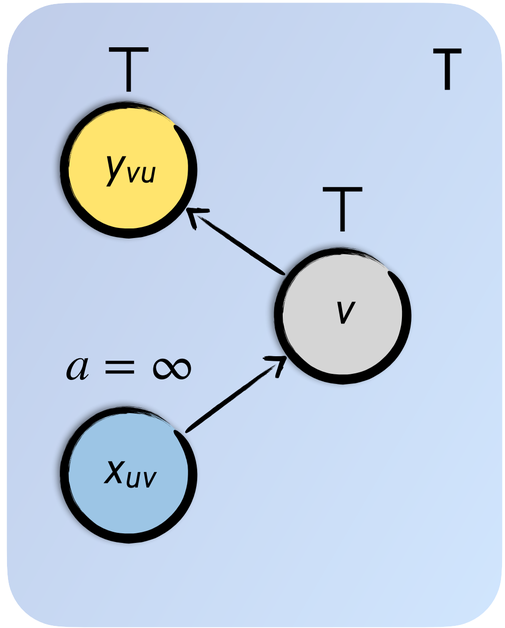
Networks with Multiple Solutions

$(v, u, true)$

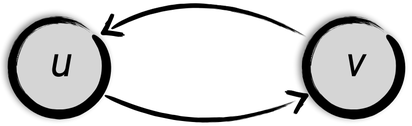
$(u, v, true)$



$P_S = true$



$P_T = true$



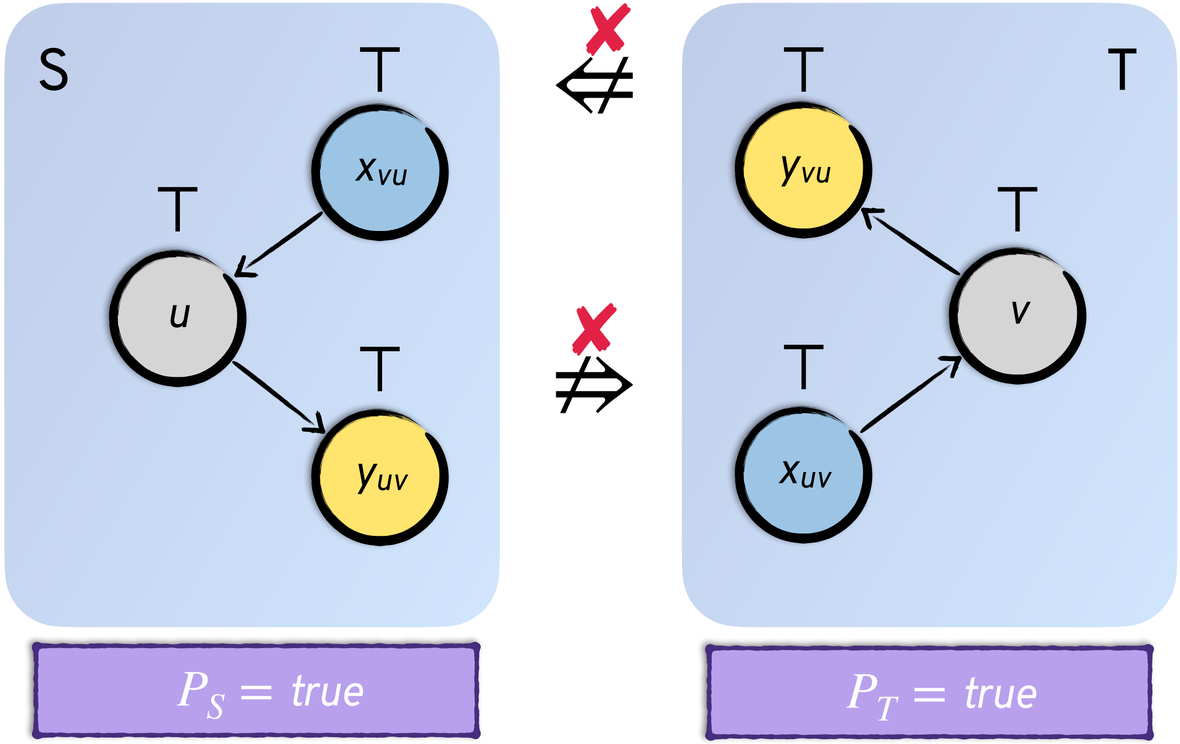
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\cdots$$
$$= \infty$$

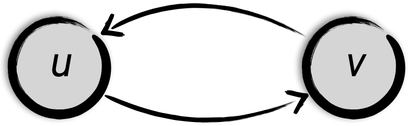
Checking an Interface

Networks with Multiple Solutions

$(v, u, true)$

$(u, v, true)$





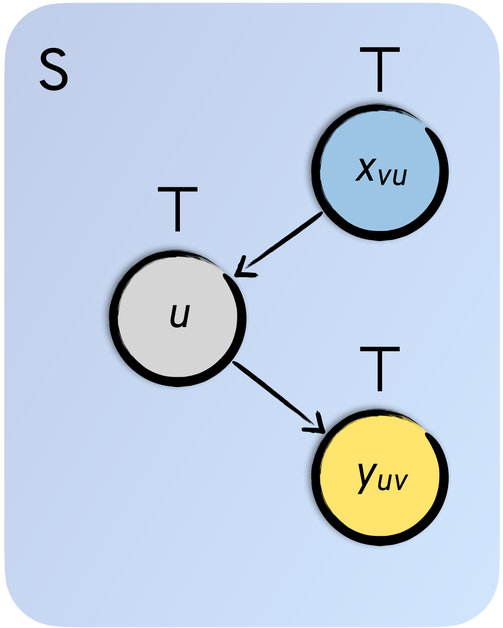
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\cdots$$
$$= \infty$$

Checking an Interface

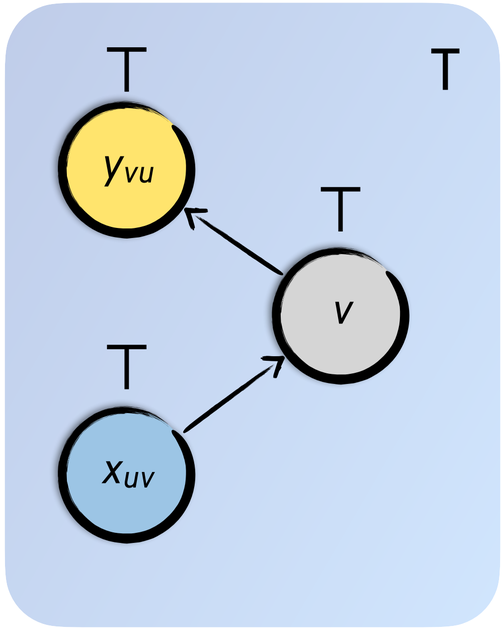
Networks with Multiple Solutions

$(v, u, true)$

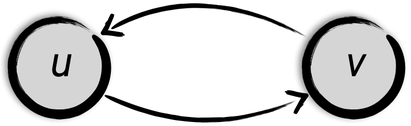
$(u, v, true)$



$P_S = true$



$P_T = true$



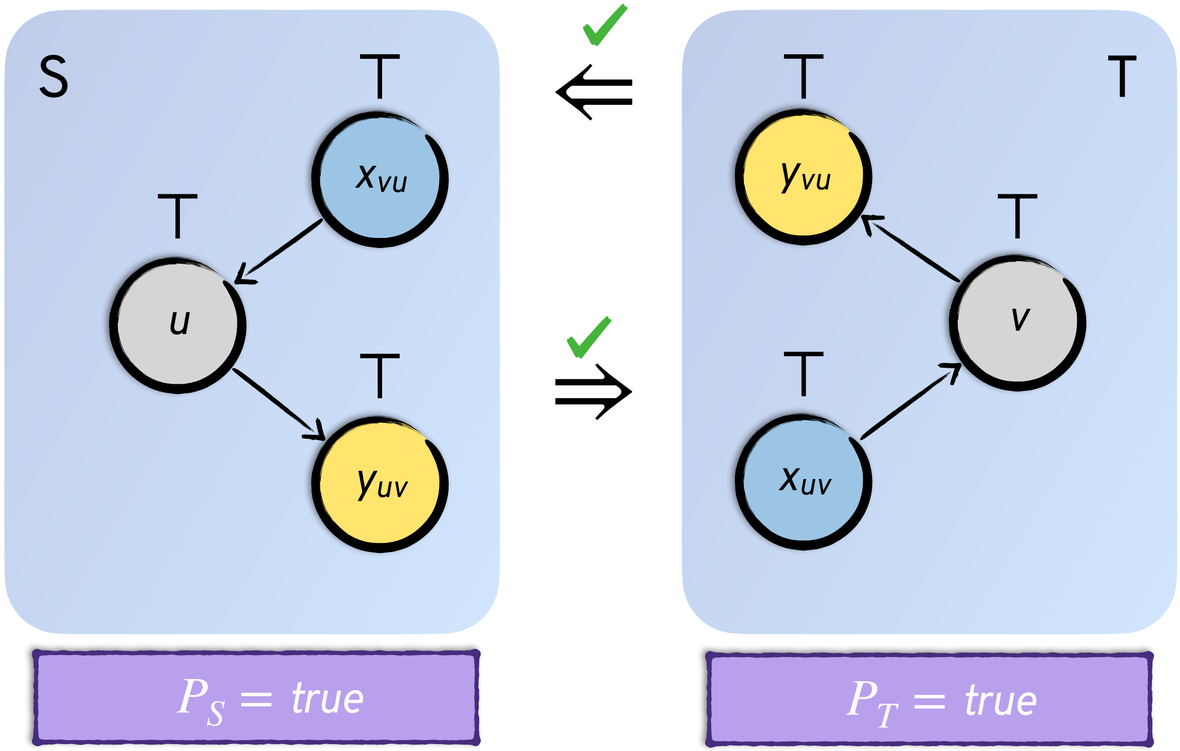
$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\cdots$$
$$= \infty$$

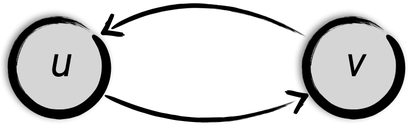
Checking an Interface

Networks with Multiple Solutions

$(v, u, true)$

$(u, v, true)$





$$\mathscr{L}(u) = \mathscr{L}(v) = 0$$
$$= 1$$
$$= 2$$
$$\cdots$$
$$= \infty$$

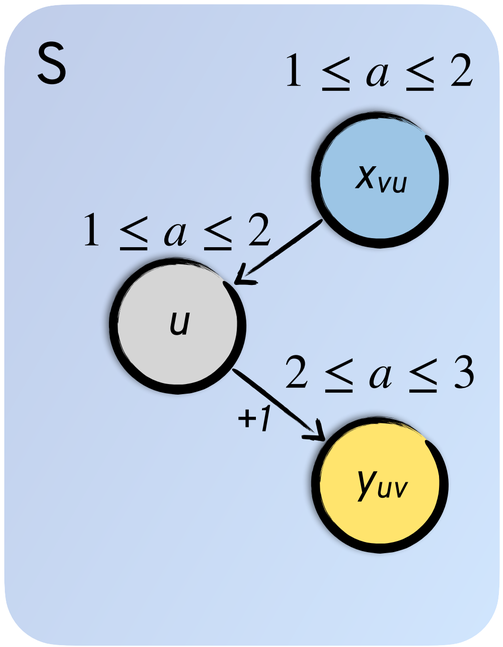
Kirigami Is Sound!

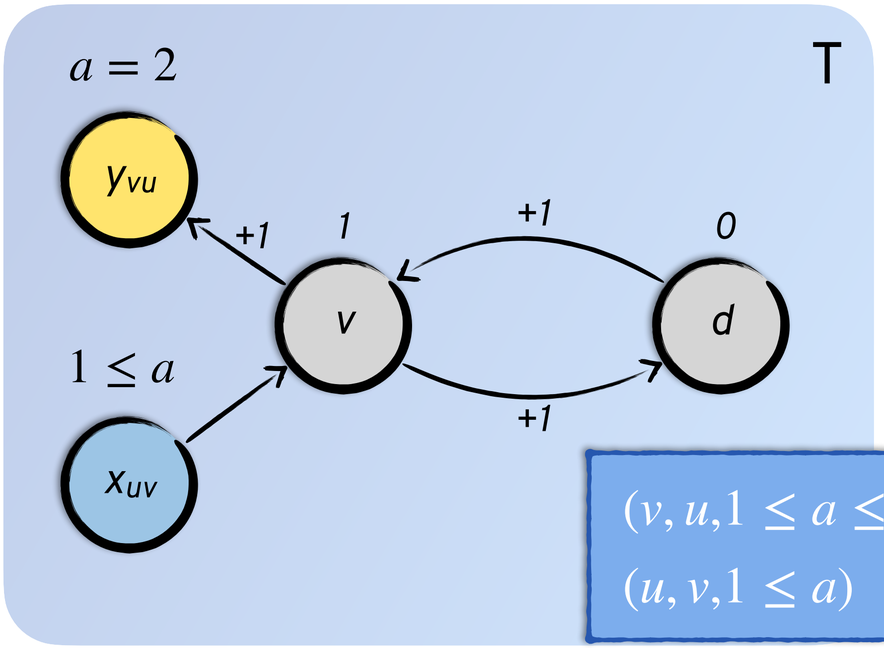
Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R





$(v, u, 1 \le a \le 2)$

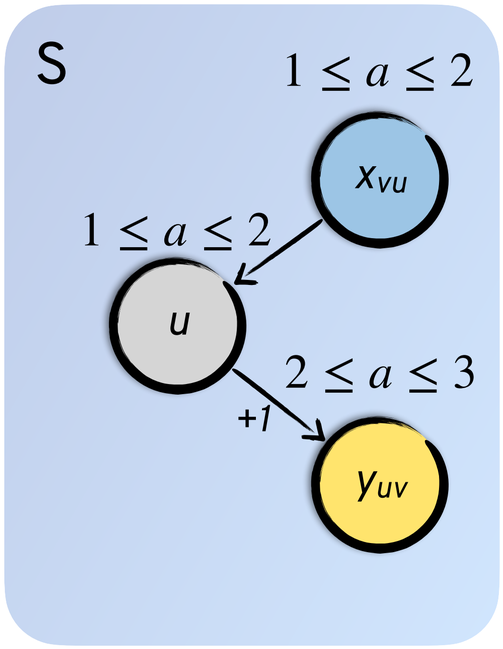
$(u, v, 1 \le a)$

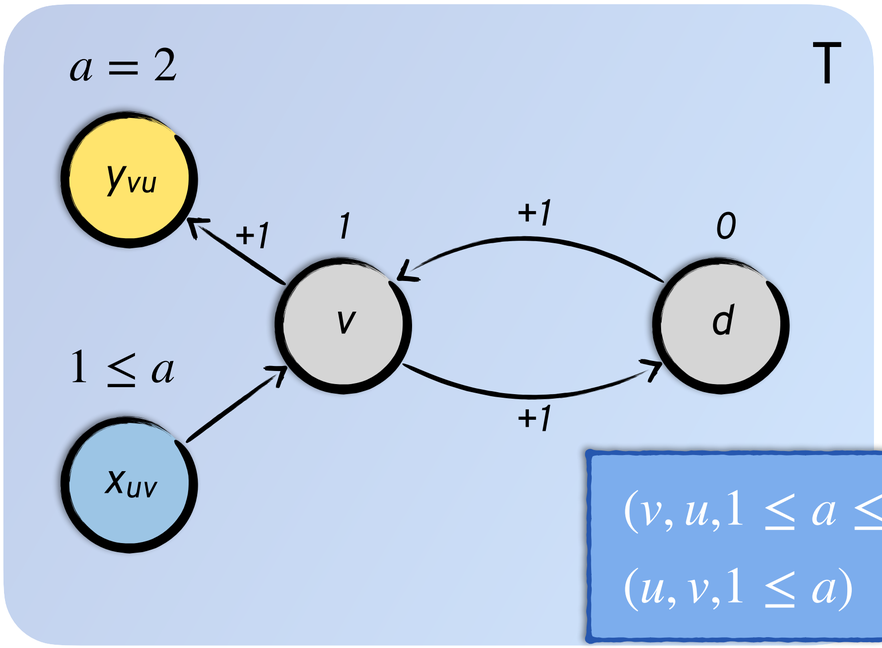
$P_S = \mathscr{L}(u) < 10$

$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R





$(v, u, 1 \le a \le 2)$
$(u, v, 1 \le a)$

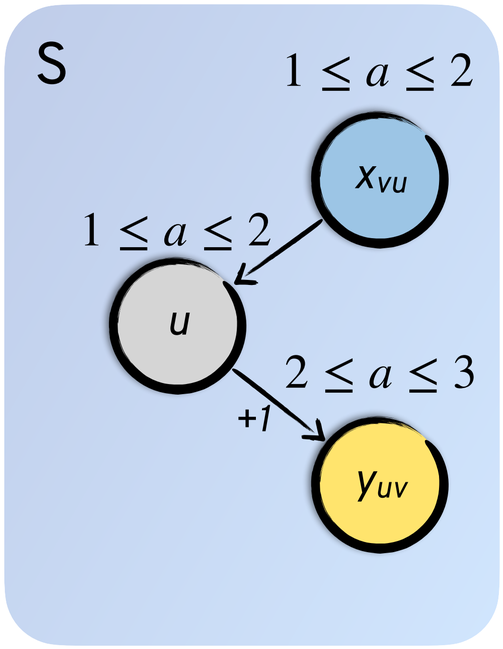
$P_S = \mathscr{L}(u) < 10$

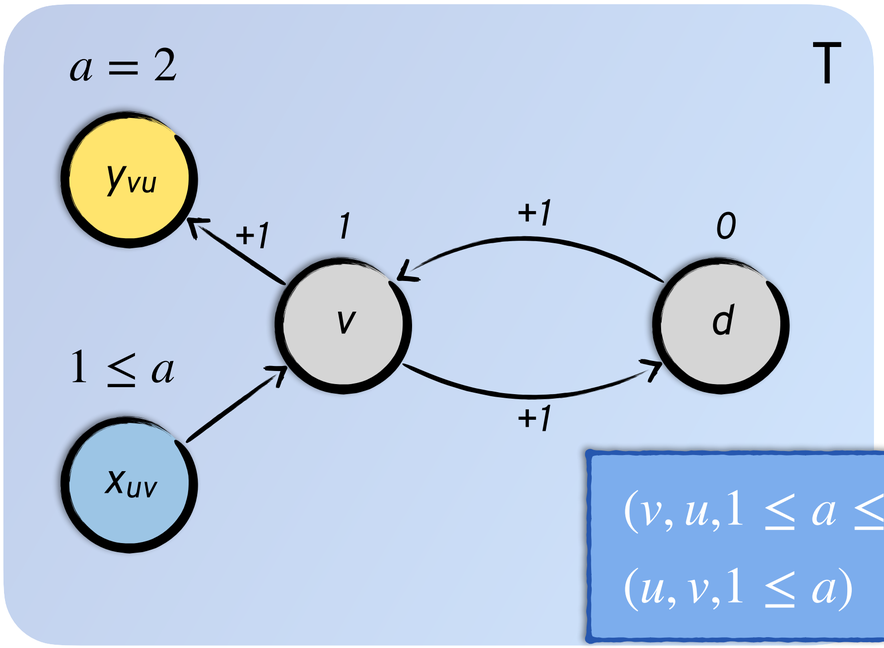
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

- To start, show that for all nodes v, $\mathscr{L}_R(v) \subseteq \mathscr{L}_T(v)$ (or S)

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R





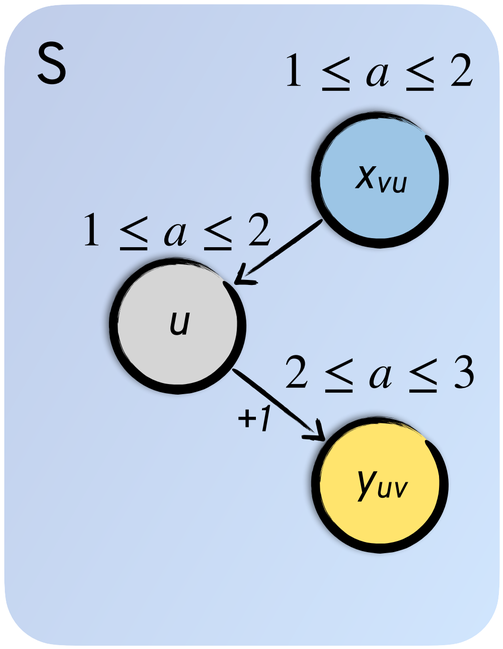
$P_S = \mathscr{L}(u) < 10$

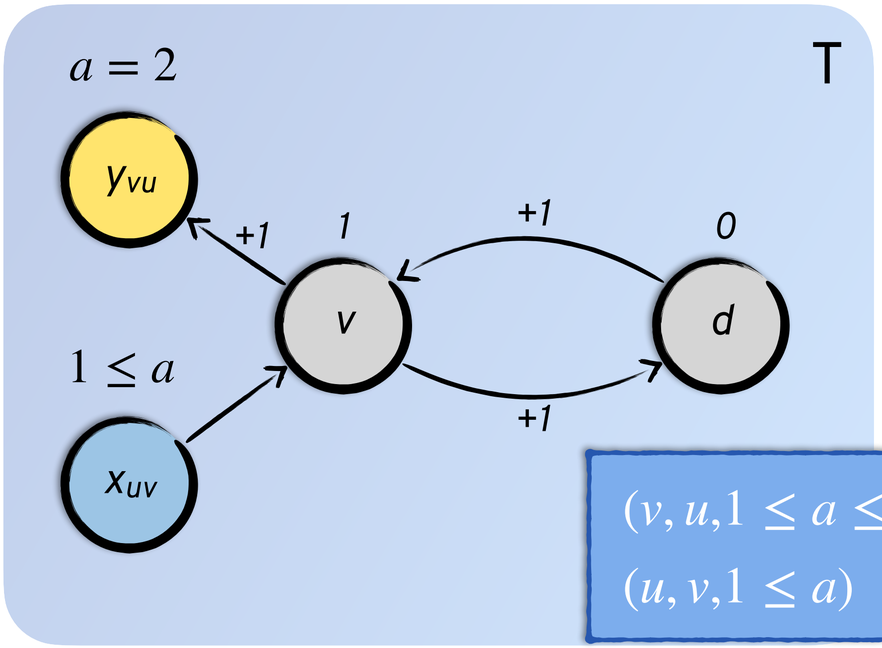
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

- To start, show that for all nodes v, $\mathscr{L}_R(v) \subseteq \mathscr{L}_T(v)$ (or S)
- Node's solution in terms of neighbors' transferred solutions trans$((u, v), \mathscr{L}(u))$

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R





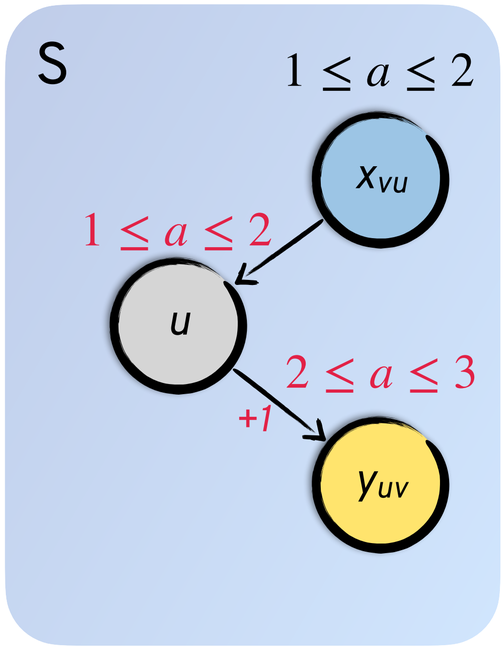
$P_S = \mathscr{L}(u) < 10$

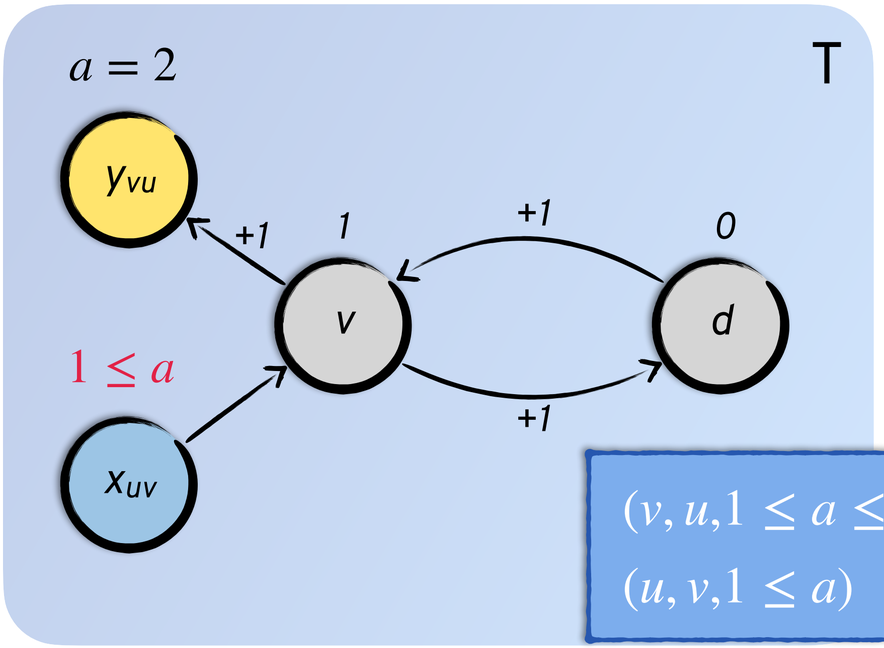
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

- To start, show that for all nodes v, $\mathscr{L}_R(v) \subseteq \mathscr{L}_T(v)$ (or S)
- Node's solution in terms of neighbors' transferred solutions $\text{trans}((u, v), \mathscr{L}(u))$
- Case analysis on the neighbors of v in T (or S)

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R





$(v, u, 1 \le a \le 2)$

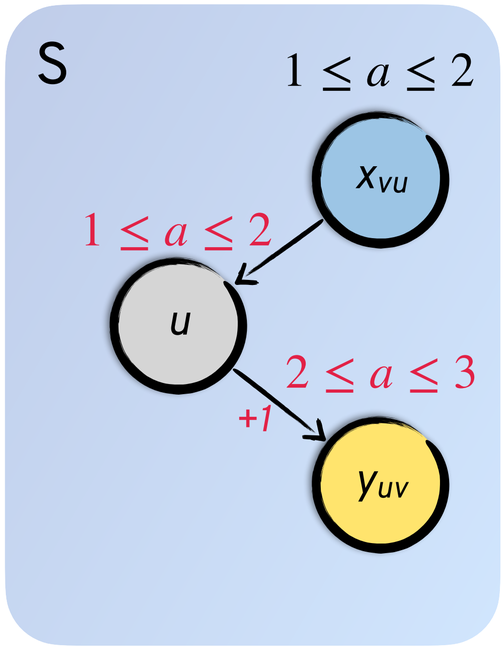
$(u, v, 1 \le a)$

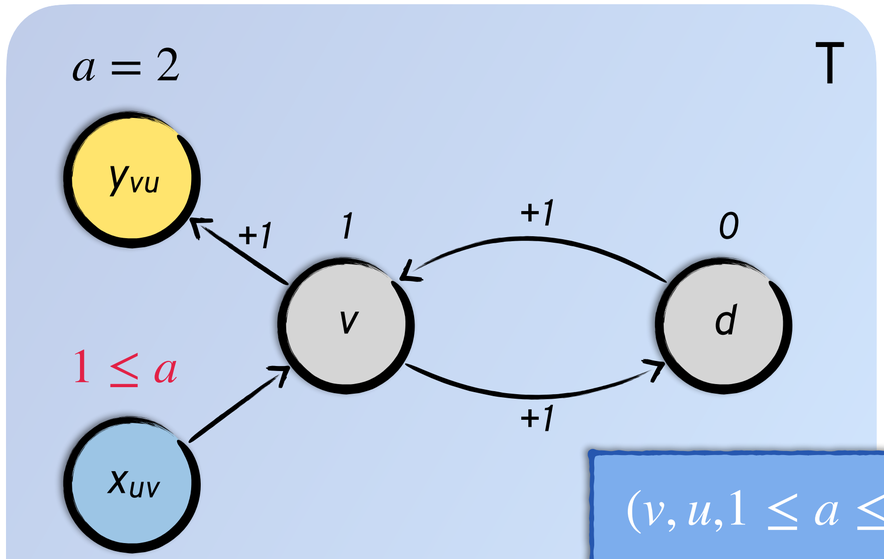
$P_S = \mathscr{L}(u) < 10$

$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R





$(v, u, 1 \le a \le 2)$
$(u, v, 1 \le a)$

$P_S = \mathscr{L}(u) < 10$

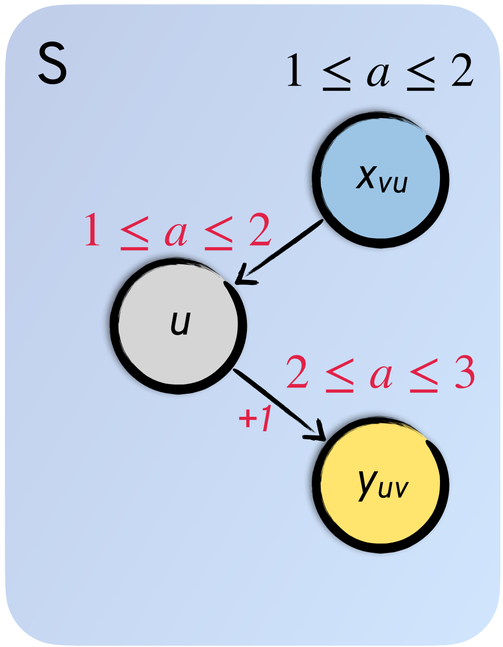
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Case 1: neighbor is input node

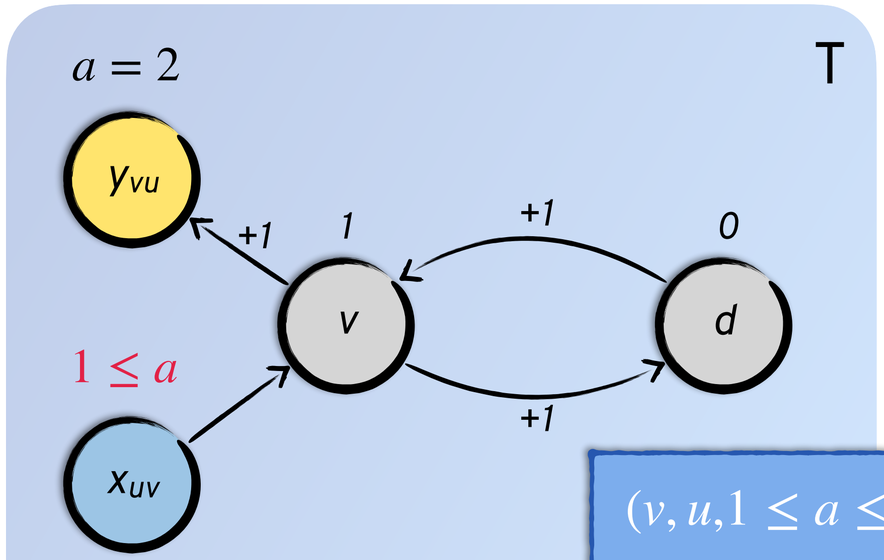
x_{uv}:

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R



$P_S = \mathscr{L}(u) < 10$



$(v, u, 1 \le a \le 2)$
$(u, v, 1 \le a)$

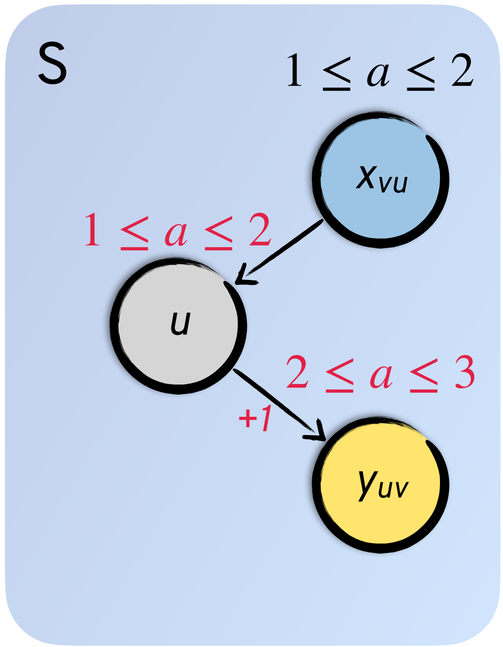
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Case 1: neighbor is input node x_{uv}:

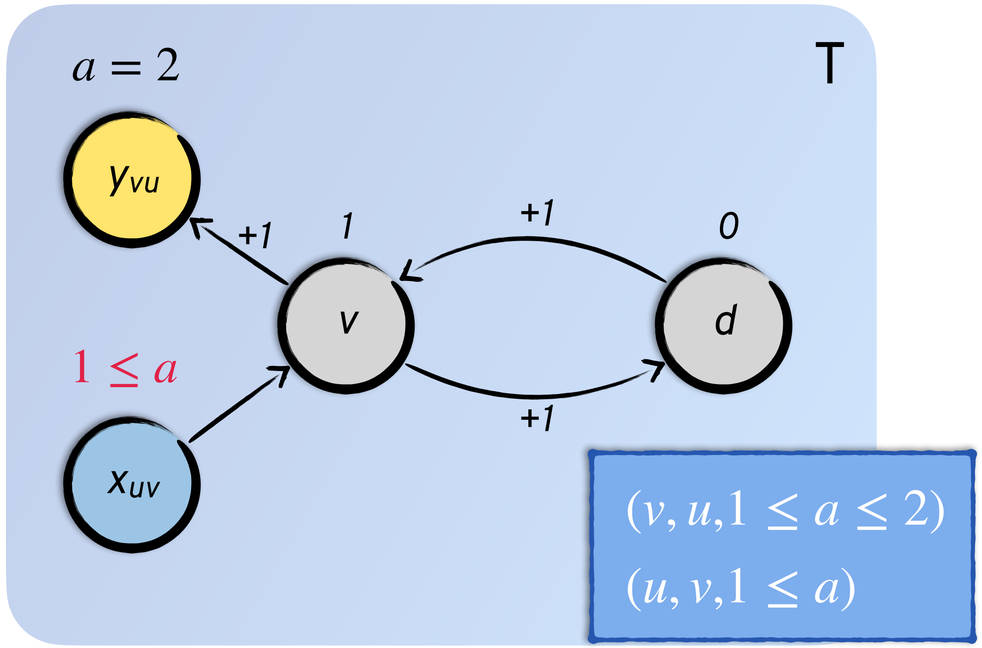
- By inductiveness check, $\text{trans}((u, y_{uv}), \mathscr{L}_S(u)) \subseteq \text{trans}((x_{uv}, v), \mathscr{L}_T(x_{uv}))$

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R



$P_S = \mathscr{L}(u) < 10$



$(v, u, 1 \le a \le 2)$
$(u, v, 1 \le a)$

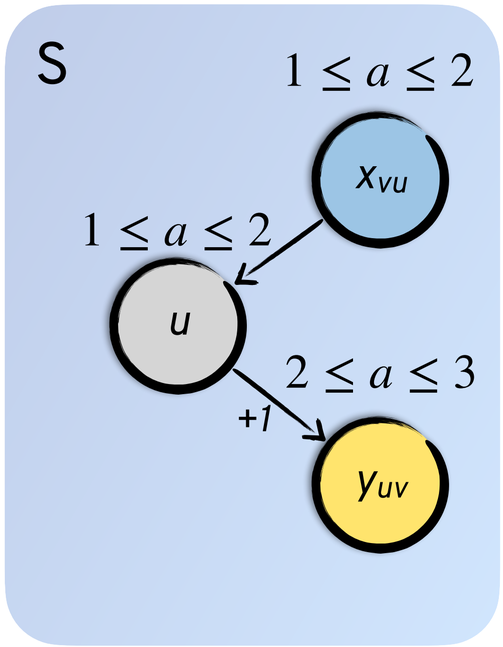
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

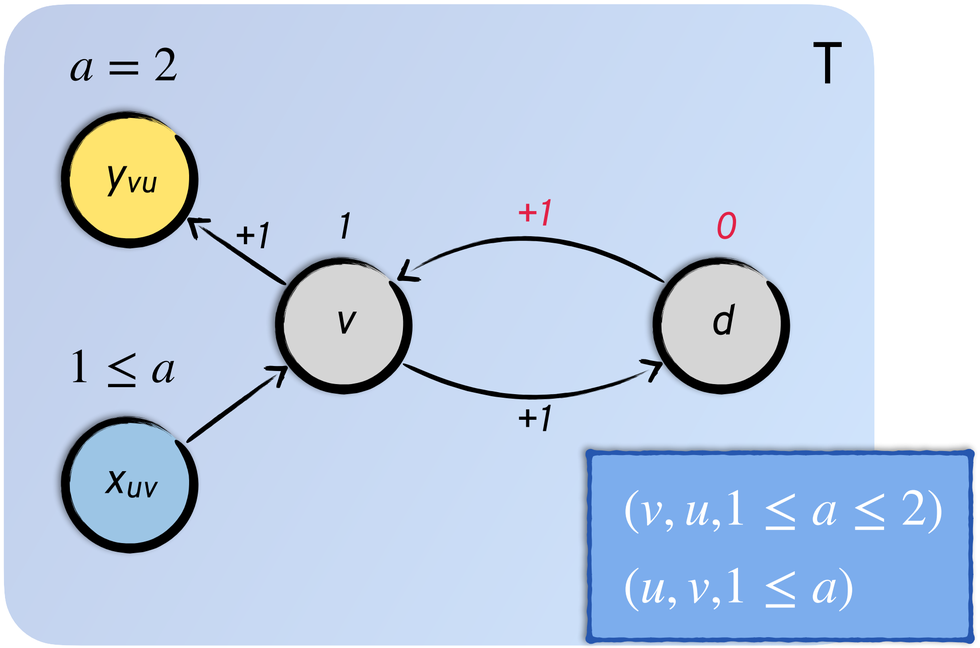
Case 1: neighbor is input node x_{uv}:

- By inductiveness check, $\text{trans}((u, y_{uv}), \mathscr{L}_S(u)) \subseteq \text{trans}((x_{uv}, v), \mathscr{L}_T(x_{uv}))$
- By co-induction, $\mathscr{L}_R(u) \subseteq \mathscr{L}_S(u)$

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R





$(v, u, 1 \le a \le 2)$

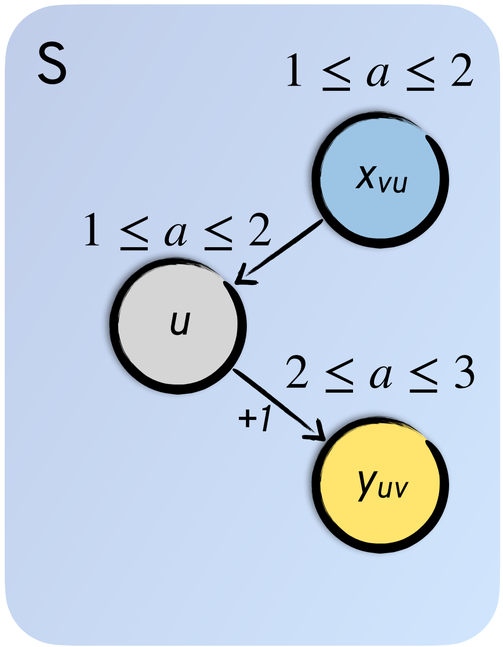
$(u, v, 1 \le a)$

$P_S = \mathscr{L}(u) < 10$

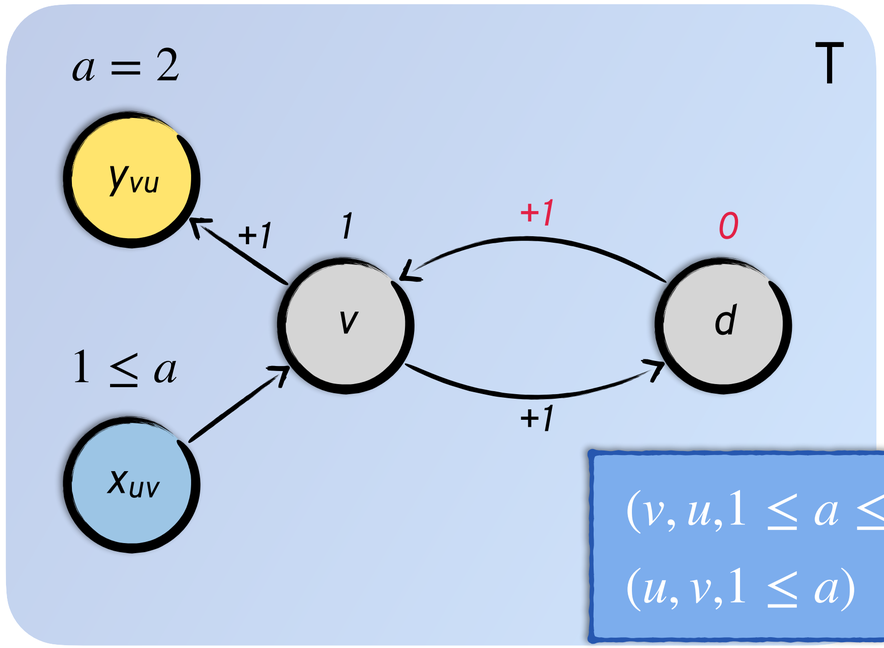
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R



$P_S = \mathscr{L}(u) < 10$



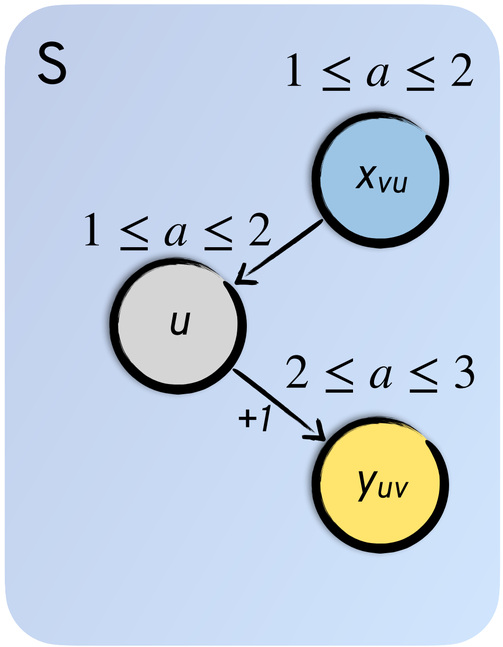
$(v, u, 1 \le a \le 2)$
$(u, v, 1 \le a)$

$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

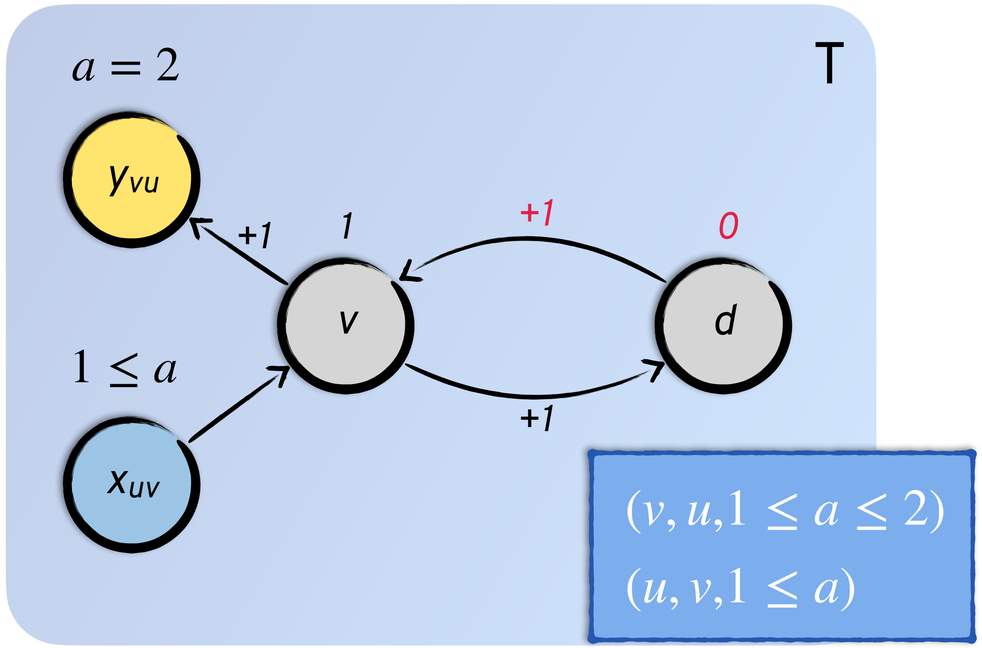
Case 2: neighbor is a base node d:

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R



$P_S = \mathscr{L}(u) < 10$



$(v, u, 1 \le a \le 2)$
$(u, v, 1 \le a)$

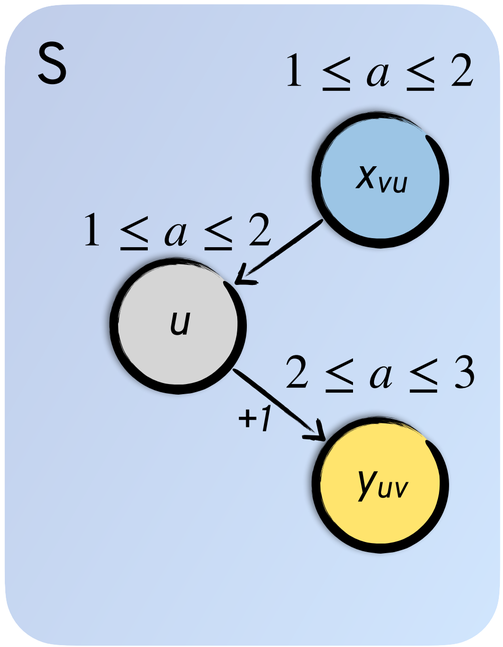
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Case 2: neighbor is a base node d:

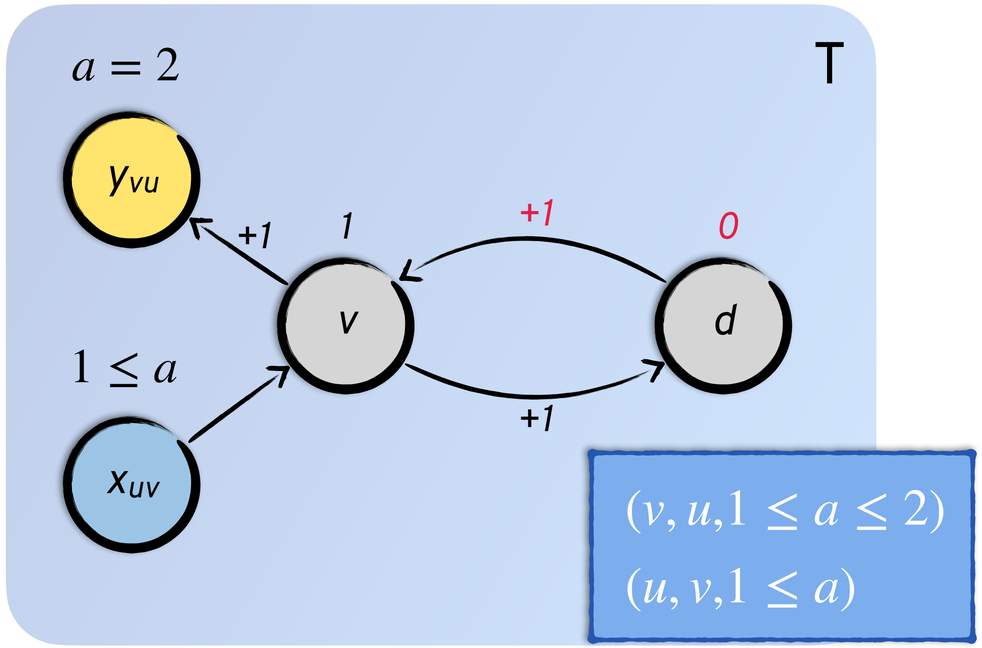
- By co-induction, $\mathscr{L}_R(d) \subseteq \mathscr{L}_T(d)$

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R



$P_S = \mathscr{L}(u) < 10$



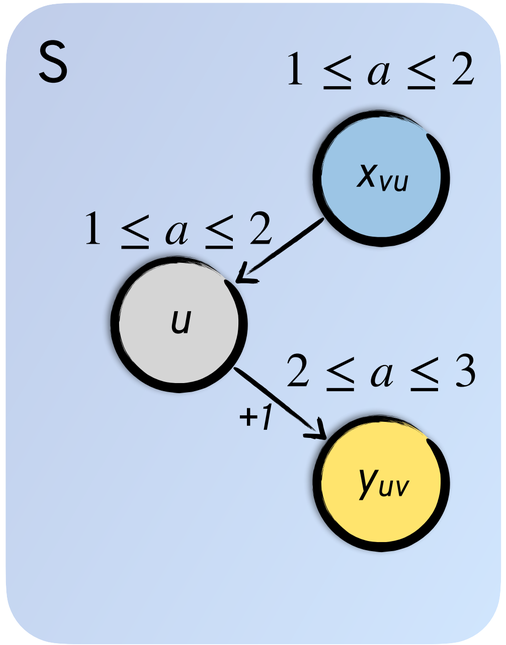
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

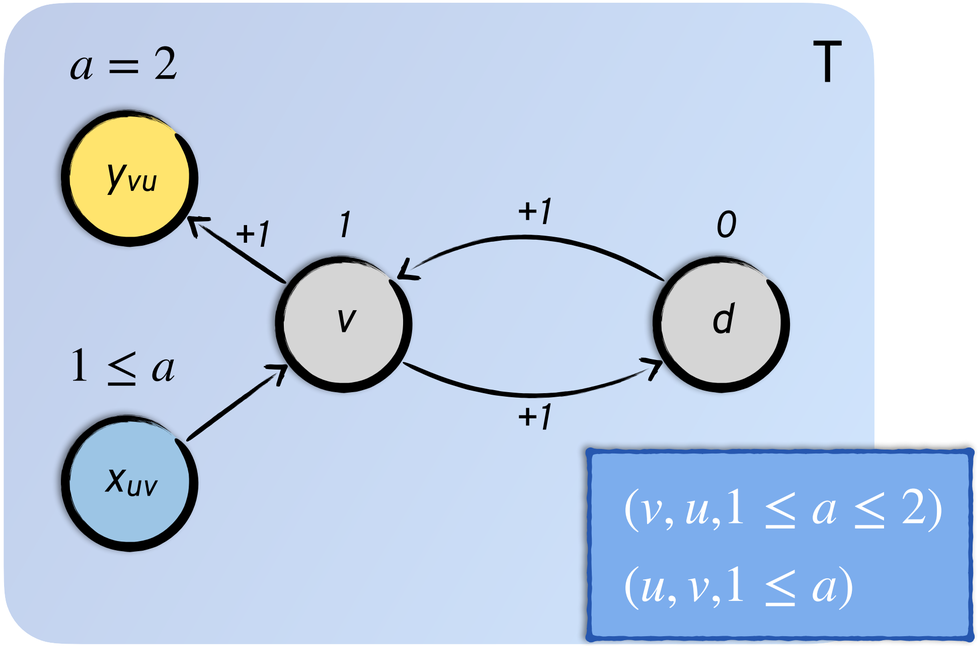
Case 2: neighbor is a base node d:

- By co-induction, $\mathscr{L}_R(d) \subseteq \mathscr{L}_T(d)$
- trans in R is the same as trans in T

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R



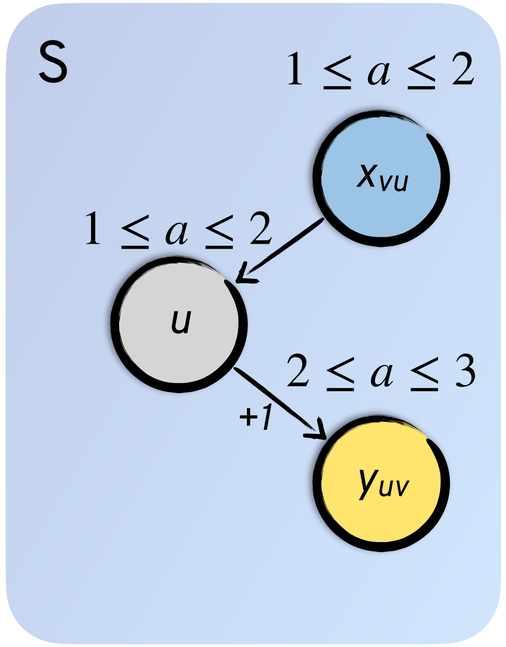


$P_S = \mathscr{L}(u) < 10$

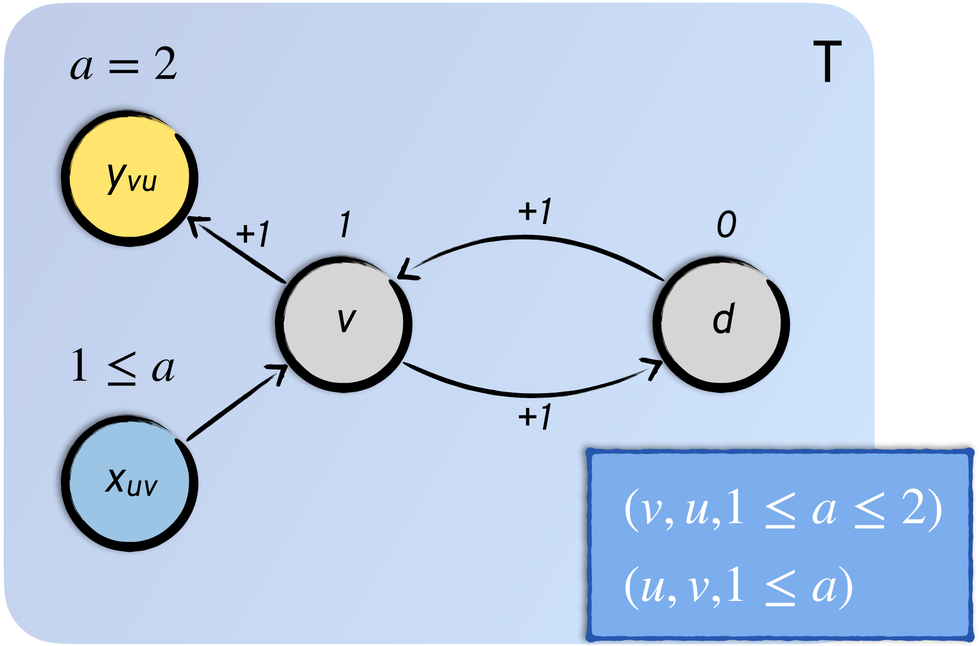
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R



$P_S = \mathscr{L}(u) < 10$

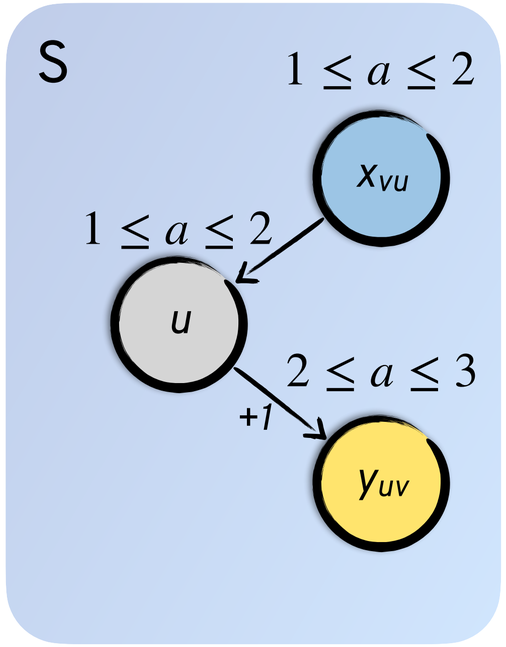


$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

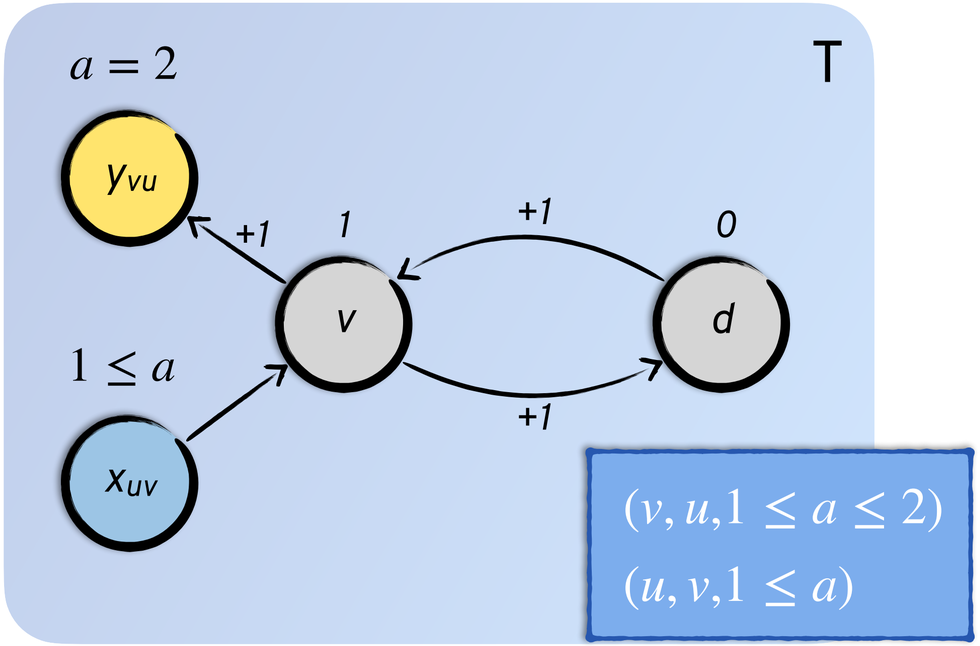
- By the two cases, for all nodes v,
  $\mathscr{L}_R(v) \subseteq \mathscr{L}_T(v)$ (or S)

Kirigami Is Sound!

Theorem: if Kirigami returns true, then property P holds for monolithic network R



$P_S = \mathscr{L}(u) < 10$



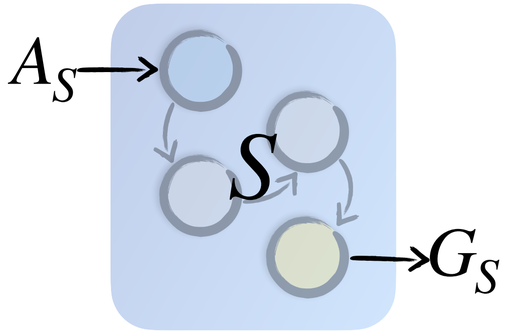
$P_T = \mathscr{L}(v) < 10 \wedge \mathscr{L}(d) < 10$

- By the two cases, for all nodes v,
  $\mathscr{L}_R(v) \subseteq \mathscr{L}_T(v)$ (or S)
- Then, since property P holds for S and T (by the safety check), it must also hold for R

An Assume-Guarantee Proof Rule

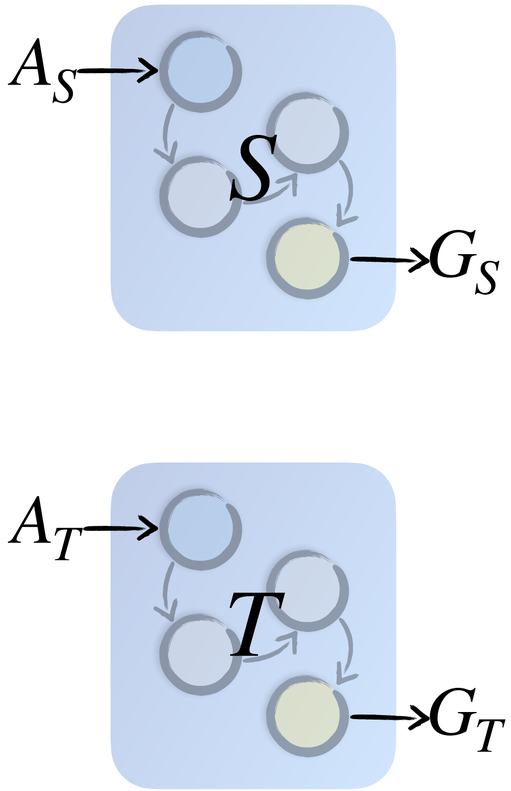
[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

An Assume-Guarantee Proof Rule



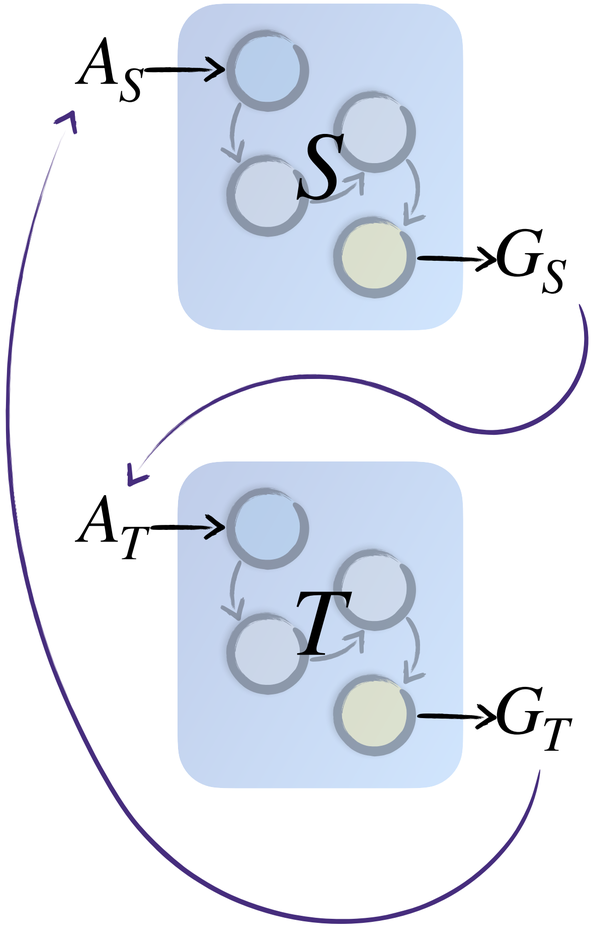
[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

An Assume-Guarantee Proof Rule



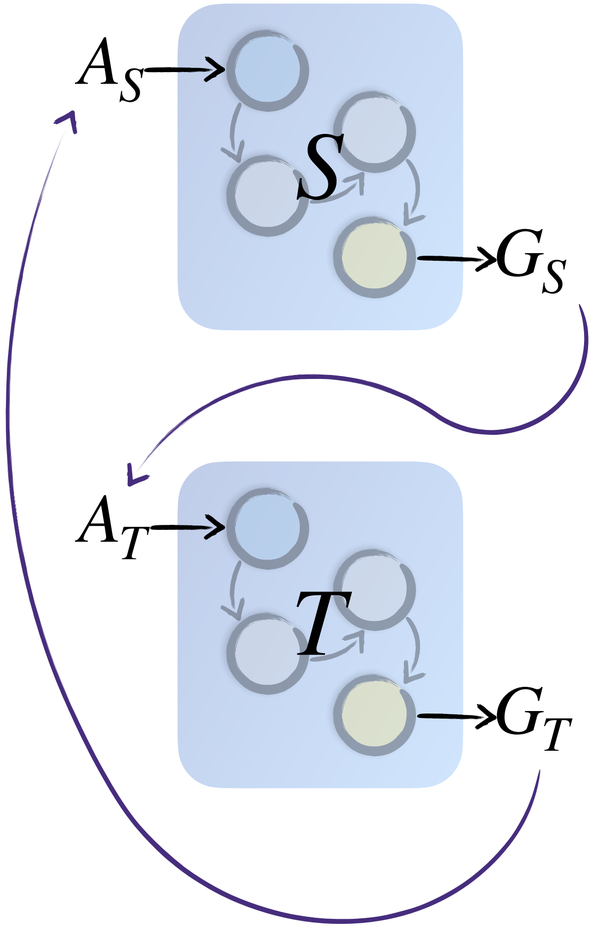
[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

An Assume-Guarantee Proof Rule



[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

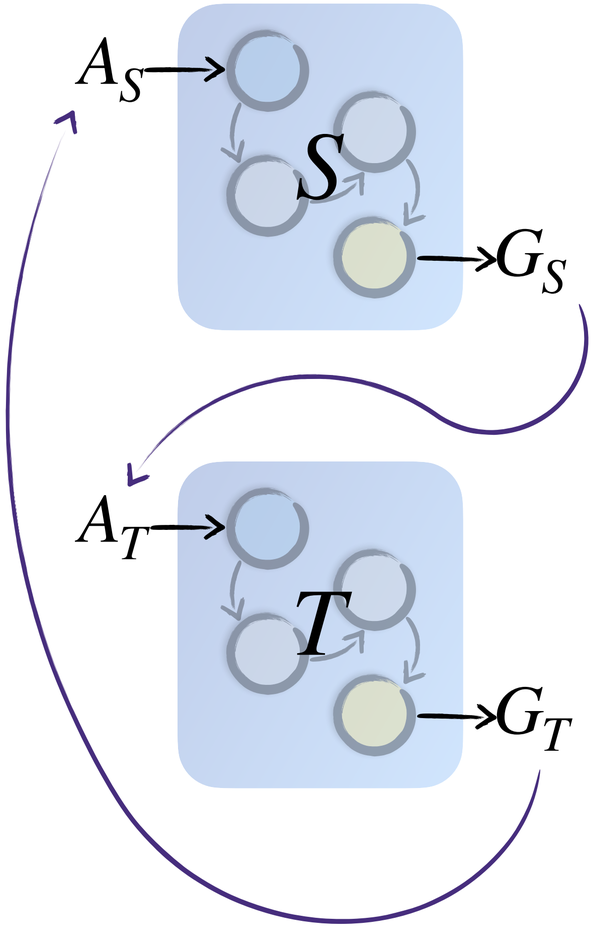
An Assume-Guarantee Proof Rule



$$\frac{\begin{array}{c}\langle A_S\rangle\ S\ \langle G_S\rangle \\ \langle A_T\rangle\ T\ \langle G_T\rangle \\ G_T \Rightarrow A_S \\ G_S \Rightarrow A_T \\ \langle A_S\rangle\ S\ \langle P_S\rangle \\ \langle A_T\rangle\ T\ \langle P_T\rangle \\ \langle \mathsf{true}\rangle\ S\ \langle G_S\rangle \\ \langle \mathsf{true}\rangle\ T\ \langle G_T\rangle\end{array}}{\langle \mathsf{true}\rangle\ S \parallel T\ \langle P_S \wedge P_T\rangle}$$

[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

An Assume-Guarantee Proof Rule

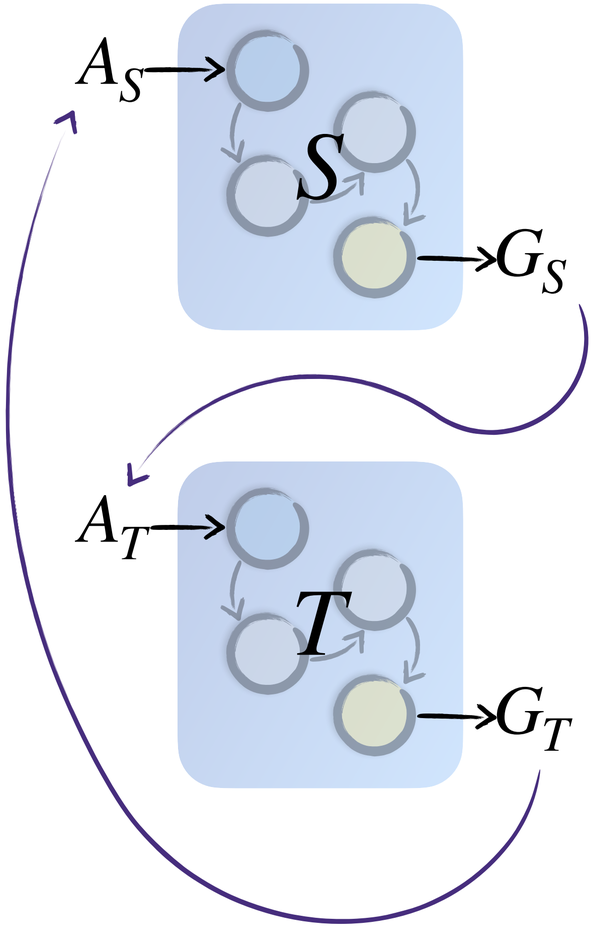


$$\frac{\begin{array}{r}\langle A_S\rangle\ S\ \langle G_S\rangle \\ \langle A_T\rangle\ T\ \langle G_T\rangle \\ G_T \Rightarrow A_S \\ G_S \Rightarrow A_T \\ \langle A_S\rangle\ S\ \langle P_S\rangle \\ \langle A_T\rangle\ T\ \langle P_T\rangle \\ \langle \text{true}\rangle\ S\ \langle G_S\rangle \\ \langle \text{true}\rangle\ T\ \langle G_T\rangle\end{array}}{\langle \text{true}\rangle\ S \parallel T\ \langle P_S \wedge P_T\rangle}$$

Inductiveness check

[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

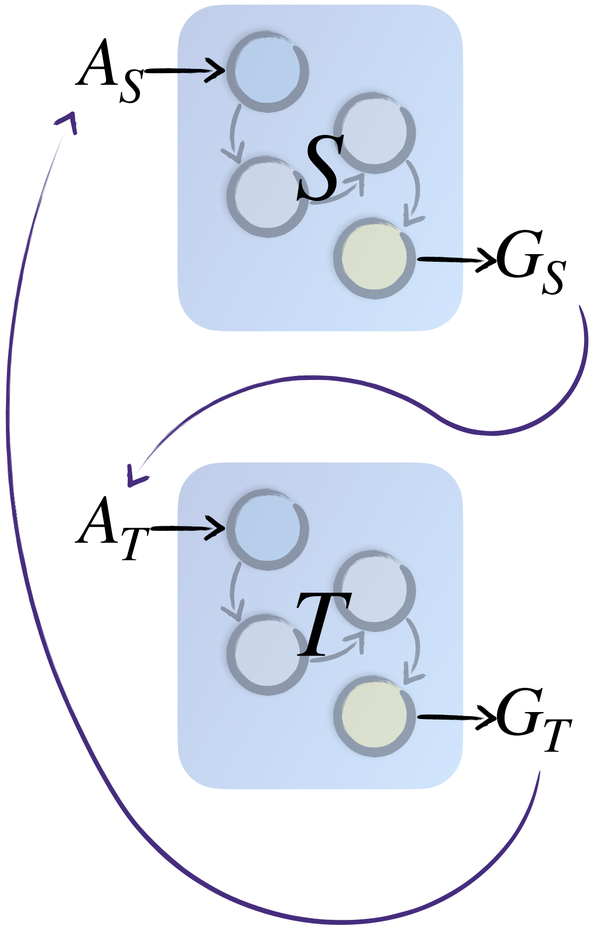
An Assume-Guarantee Proof Rule



$$\frac{\begin{array}{rl} \langle A_S \rangle\ S\ \langle G_S \rangle & \left.\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right\} \text{Inductiveness check} \\ \langle A_T \rangle\ T\ \langle G_T \rangle & \\ G_T \Rightarrow A_S & \\ G_S \Rightarrow A_T & \\ \langle A_S \rangle\ S\ \langle P_S \rangle & \left.\vphantom{\begin{array}{l}a\\a\end{array}}\right\} \text{Safety check} \\ \langle A_T \rangle\ T\ \langle P_T \rangle & \\ \langle \text{true} \rangle\ S\ \langle G_S \rangle & \\ \langle \text{true} \rangle\ T\ \langle G_T \rangle & \end{array}}{\langle \text{true} \rangle\ S \parallel T\ \langle P_S \wedge P_T \rangle}$$

[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

An Assume-Guarantee Proof Rule



$$\frac{\begin{array}{rl} \langle A_S \rangle\ S\ \langle G_S \rangle & \left.\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right\} \text{Inductiveness check} \\ \langle A_T \rangle\ T\ \langle G_T \rangle & \\ G_T \Rightarrow A_S & \\ G_S \Rightarrow A_T & \\ \langle A_S \rangle\ S\ \langle P_S \rangle & \left.\vphantom{\begin{array}{l}a\\a\end{array}}\right\} \text{Safety check} \\ \langle A_T \rangle\ T\ \langle P_T \rangle & \\ \langle \text{true} \rangle\ S\ \langle G_S \rangle & \left.\vphantom{\begin{array}{l}a\\a\end{array}}\right\} \text{Initial check} \\ \langle \text{true} \rangle\ T\ \langle G_T \rangle & \end{array}}{\langle \text{true} \rangle\ S \parallel T\ \langle P_S \wedge P_T \rangle}$$

[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

Implementing Kirigami in NV

Implementing Kirigami in NV

[Beckett et al., NetPL 2019]

[Giannarakis et al., PLDI 2020]

Implementing Kirigami in NV

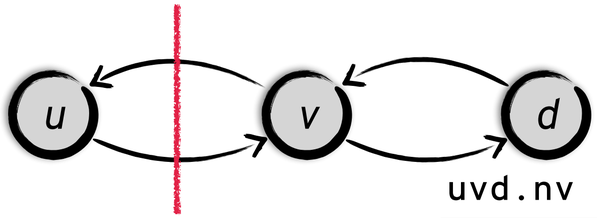
uvd.nv

```
type attribute = int

(* Number of nodes in network topology *)
(* 0 = d; 1 = v; 2 = u *)
let nodes = 3

(* List of edges in network topology *)
let edges = { 0=1; 1=2; }

(* The merge function for receiving attributes *)
let merge node x y =
  if x < y then x else y

(* The trans function for sending attributes *)
let trans edge x = x + 1

(* The initial state of the network *)
let init node =
  match node with
  | 0n -> 0
  | _ -> 10

(* The assertion on each node's solution *)
let assert node x = x < 10
```

[Beckett et al., NetPL 2019]

[Giannarakis et al., PLDI 2020]

Implementing Kirigami in NV



uvd.nv

```
type attribute = int

(* Number of nodes in network topology *)
(* 0 = d; 1 = v; 2 = u *)
let nodes = 3

(* List of edges in network topology *)
let edges = { 0=1; 1=2; }

(* The merge function for receiving attributes *)
let merge node x y =
  if x < y then x else y

(* The trans function for sending attributes *)
let trans edge x = x + 1

(* The initial state of the network *)
let init node =
  match node with
  | 0n -> 0
  | _ -> 10

(* The assertion on each node's solution *)
let assert node x = x < 10
```

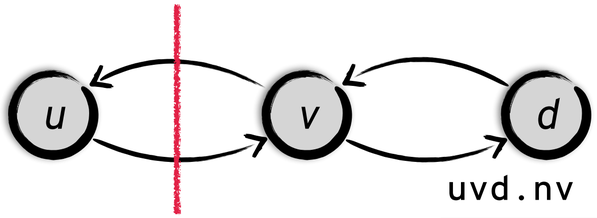
uvd-kirigami.nv

```
include "uvd.nv"
```

[Beckett et al., NetPL 2019]

[Giannarakis et al., PLDI 2020]

Implementing Kirigami in NV



uvd.nv

```
type attribute = int

(* Number of nodes in network topology *)
(* 0 = d; 1 = v; 2 = u *)
let nodes = 3

(* List of edges in network topology *)
let edges = { 0=1; 1=2; }

(* The merge function for receiving attributes *)
let merge node x y =
  if x < y then x else y

(* The trans function for sending attributes *)
let trans edge x = x + 1

(* The initial state of the network *)
let init node =
  match node with
  | 0n -> 0
  | _ -> 10

(* The assertion on each node's solution *)
let assert node x = x < 10
```

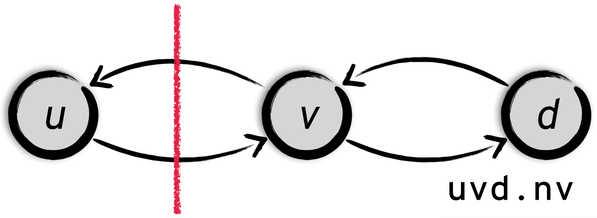
uvd-kirigami.nv

```
include "uvd.nv"

(* Associate each node with a partition *)
let partition node = match node with
  | 0n -> 0
  | 1n -> 0
  | 2n -> 1
```

[Beckett et al., NetPL 2019]

[Giannarakis et al., PLDI 2020]

Implementing Kirigami in NV



uvd.nv

```
type attribute = int

(* Number of nodes in network topology *)
(* 0 = d; 1 = v; 2 = u *)
let nodes = 3

(* List of edges in network topology *)
let edges = { 0=1; 1=2; }

(* The merge function for receiving attributes *)
let merge node x y =
  if x < y then x else y

(* The trans function for sending attributes *)
let trans edge x = x + 1

(* The initial state of the network *)
let init node =
  match node with
  | 0n -> 0
  | _ -> 10

(* The assertion on each node's solution *)
let assert node x = x < 10
```

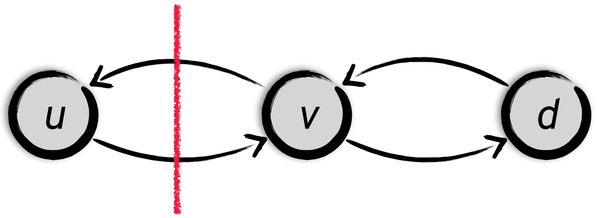
uvd-kirigami.nv

```
include "uvd.nv"

(* Associate each node with a partition *)
let partition node = match node with
  | 0n -> 0
  | 1n -> 0
  | 2n -> 1

(* Associate each edge with a hypothesis *)
let interface edge = match edge with
  | 1~2 -> Some (fun a -> a = 1 || a = 2)
  | 2~1 -> Some (fun a -> a >= 1)
  | _ -> None
```

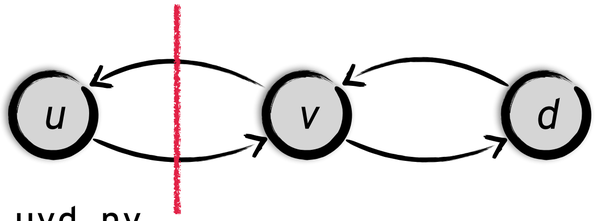
[Beckett et al., NetPL 2019]

[Giannarakis et al., PLDI 2020]

Implementing Kirigami in NV



Implementing Kirigami in NV



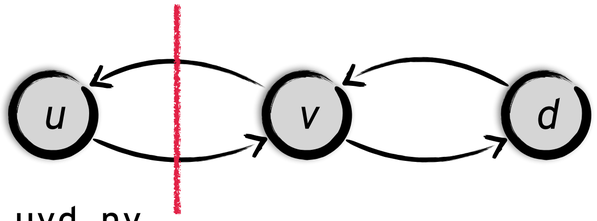
uvd.nv

```
type attribute = int

(* Number of nodes in network topology *)
(* 0 = d; 1 = v; 2 = u *)
let nodes = 3

(* List of edges in network topology *)
let edges = { 0=1; 1=2; }

(* The merge function for receiving attributes *)
let merge node x y =
  if x < y then x else y

(* The trans function for sending attributes *)
let trans edge x = x + 1

(* The initial state of the network *)
let init node =
  match node with
  | 0n -> 0
  | _ -> 10

(* The assertion on each node's solution *)
let assert node x = x < 10
```

uvd-part.nv

```
include "uvd.nv"

(* Associate each node with a partition *)
let partition node = match node with
  | 0n -> 0
  | 1n -> 0
  | 2n -> 1

(* Associate each edge with a hypothesis *)
let interface edge = match edge with
  | 1~2 -> Some (fun a -> a = 1 || a = 2)
  | 2~1 -> Some (fun a -> a >= 1)
  | _ -> None
```

Implementing Kirigami in NV

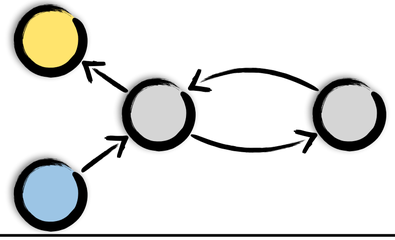


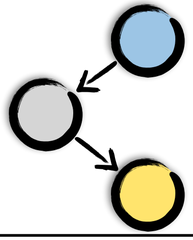
uvd.nv

```
type attribute = int

(* Number of nodes in network topology *)
(* 0 = d; 1 = v; 2 = u *)
let nodes = 3

(* List of edges in network topology *)
let edges = { 0=1; 1=2; }

(* The merge function for receiving attributes *)
let merge node x y =
  if x < y then x else y

(* The trans function for sending attributes *)
let trans edge x = x + 1

(* The initial state of the network *)
let init node =
  match node with
  | 0n -> 0
  | _ -> 10

(* The assertion on each node's solution *)
let assert node x = x < 10
```

uvd-part.nv

```
include "uvd.nv"

(* Associate each node with a partition *)
let partition node = match node with
  | 0n -> 0
  | 1n -> 0
  | 2n -> 1

(* Associate each edge with a hypothesis *)
let interface edge = match edge with
  | 1~2 -> Some (fun a -> a = 1 || a = 2)
  | 2~1 -> Some (fun a -> a >= 1)
  | _ -> None
```





```
(* Partition 0 *)
(* Constraints over inputs *)
symbolic h_2_1 : int
require (fun a -> a >= 1) h_2_1

let nodes = (* updated nodes *)
let edges = (* updated edges *)

let merge node x y =
  if x < y then x else y

let trans edge x =
  match edge with
  | (* input edge *) -> x
  | _ -> x + 1

(* The initial state of the network *)
let init node =
  match node with
  | (* input of 2 *) -> h_2_1
  | (* ...as before... *)

(* The assertion on each node's solution *)
let assert node x =
   match node with
  | (* output of 1 *) -> (fun a -> a = 1 || a = 2) x
  | _ -> x < 10
```

```
(* Partition 1 *)
(* Constraints over inputs *)
symbolic h_1_2 : int
require (fun a -> a = 1 || a = 2) h_1_2

let nodes = (* updated nodes *)
let edges = (* updated edges *)

let merge node x y =
  if x < y then x else y

let trans edge x =
  match edge with
  | (* input edge *) -> x
  | _ -> x + 1

(* The initial state of the network *)
let init node =
  match node with
  | (* input of 1 *) -> h_1_2
  | (* ...as before... *)

(* The assertion on each node's solution *)
let assert node x =
   match node with
  | (* output of 2 *) -> (fun a -> a >= 1) x
  | _ -> x < 10
```

Internal representation

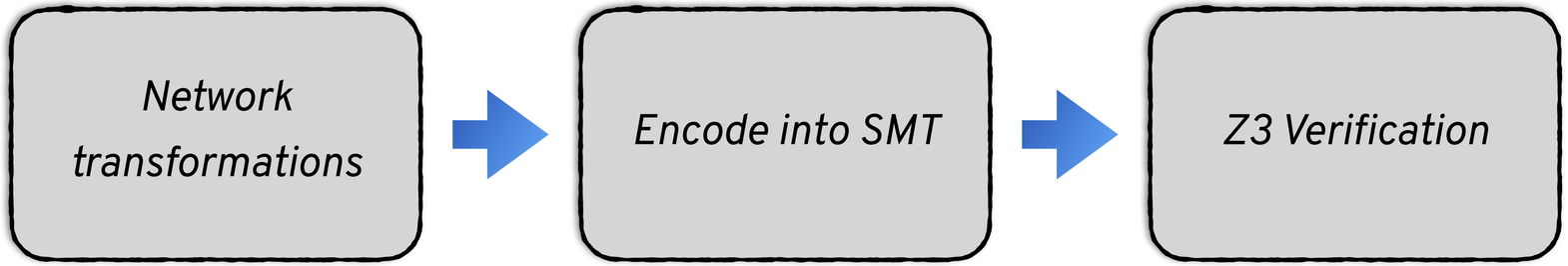
Running a Query in NV

[Beckett et al., NetPL 2019]

[Giannarakis et al., PLDI 2020]

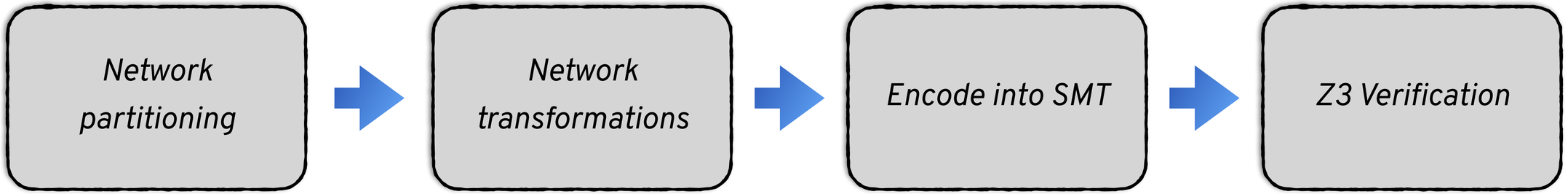
[De Moura and Bjørner, TACAS 2008]

Running a Query in NV



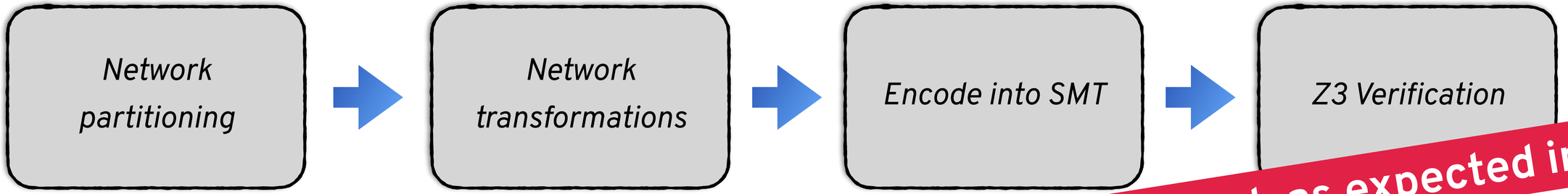
[Beckett et al., NetPL 2019]
[Giannarakis et al., PLDI 2020]
[De Moura and Bjørner, TACAS 2008]

Running a Query in NV



[Beckett et al., NetPL 2019]
[Giannarakis et al., PLDI 2020]
[De Moura and Bjørner, TACAS 2008]

Running a Query in NV



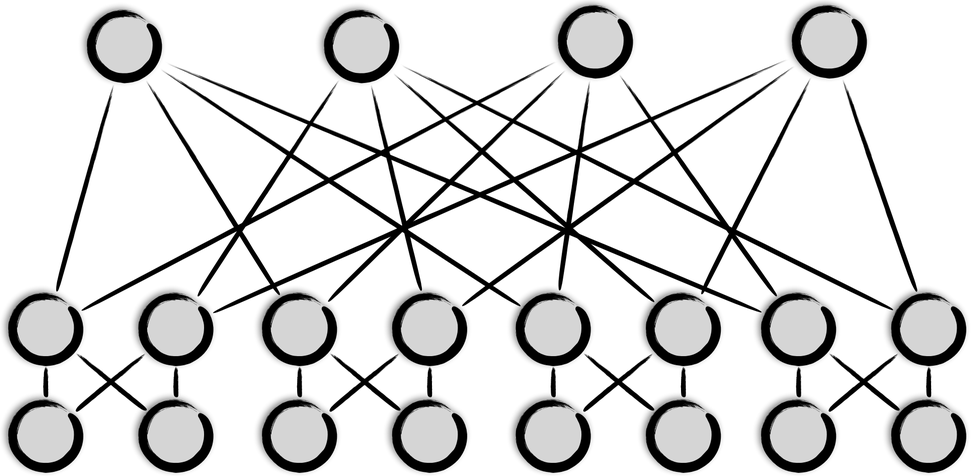
All the examples we've seen thus far have been verified and work as expected in NV!

[Beckett et al., NetPL 2019]
[Giannarakis et al., PLDI 2020]
[De Moura and Bjørner, TACAS 2008]

Results

Fattree Reachability

Case Study



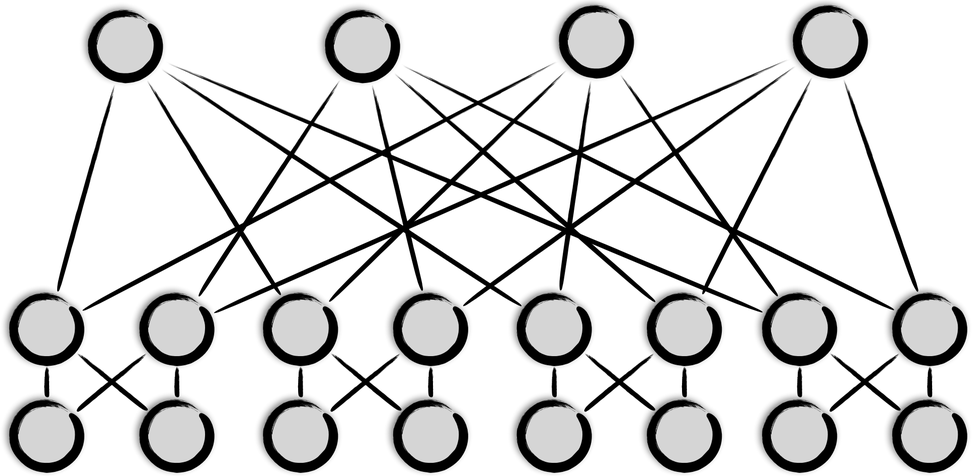
[Fogel et al., NSDI 2015]

[Leiserson, IEEE TC 1985]

Fattree Reachability

Case Study

- Common data centre network



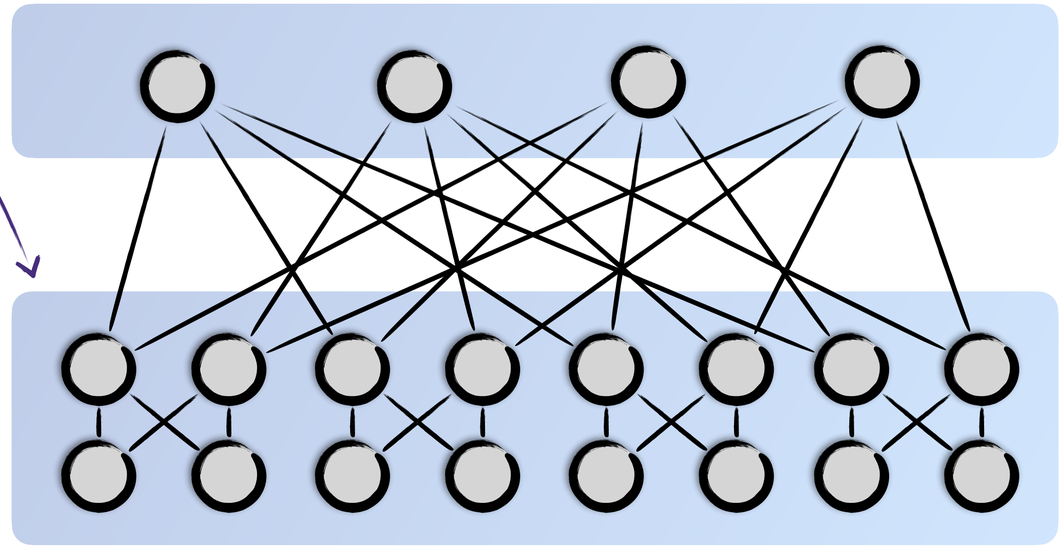
[Fogel et al., NSDI 2015]

[Leiserson, IEEE TC 1985]

Fattree Reachability

Case Study

- Common data centre network
- Hierarchical design using *spines* and *pods*



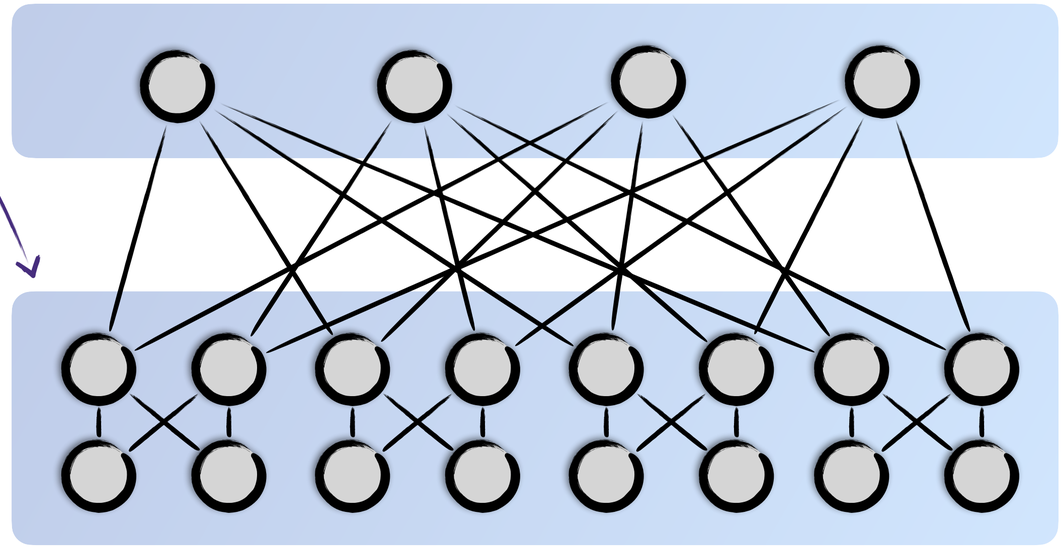
[Fogel et al., NSDI 2015]

[Leiserson, IEEE TC 1985]

Fattree Reachability

Case Study

- Common data centre network
- Hierarchical design using *spines* and *pods*
- Real-world infrastructures can have over 10,000 nodes



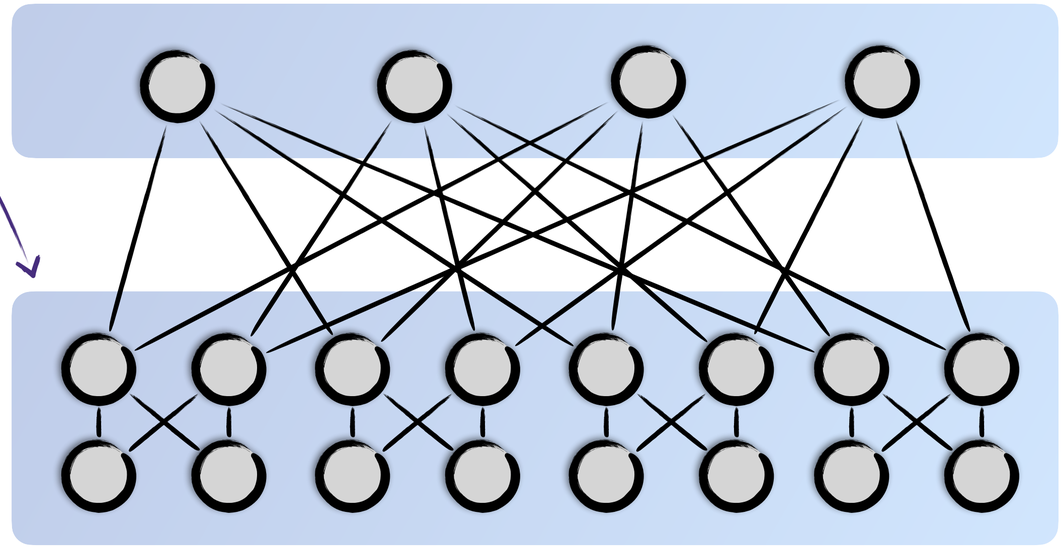
[Fogel et al., NSDI 2015]

[Leiserson, IEEE TC 1985]

Fattree Reachability

Case Study

- Common data centre network
- Hierarchical design using *spines* and *pods*
- Real-world infrastructures can have over 10,000 nodes
  - Synthetic networks ranged from 20 to 500 nodes



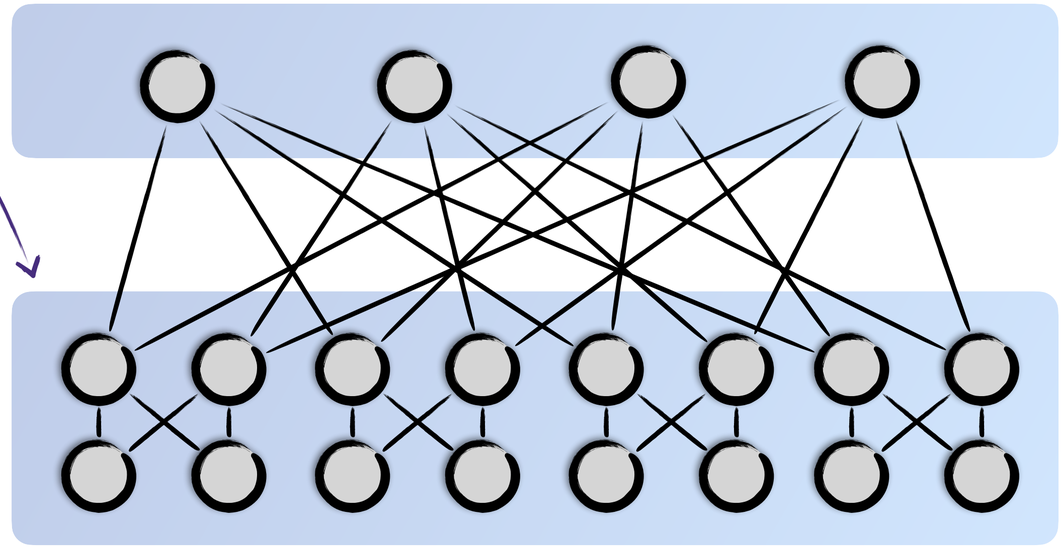
[Fogel et al., NSDI 2015]

[Leiserson, IEEE TC 1985]

Fattree Reachability

Case Study

- Common data centre network
- Hierarchical design using *spines* and *pods*
- Real-world infrastructures can have over 10,000 nodes
  - Synthetic networks ranged from 20 to 500 nodes
- Modelled shortest-path routing



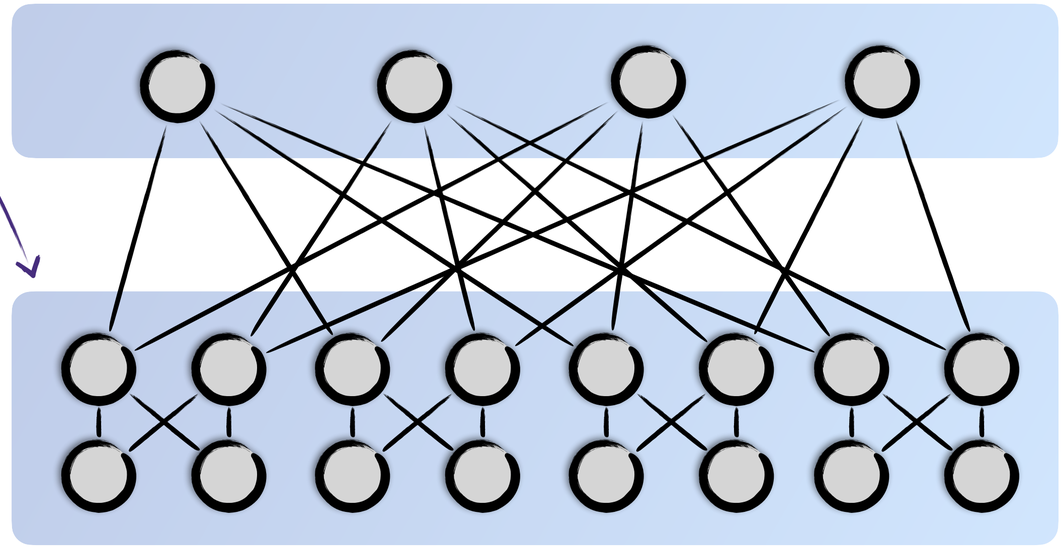
[Fogel et al., NSDI 2015]

[Leiserson, IEEE TC 1985]

Fattree Reachability

Case Study

- Common data centre network
- Hierarchical design using *spines* and *pods*
- Real-world infrastructures can have over 10,000 nodes
  - Synthetic networks ranged from 20 to 500 nodes
- Modelled shortest-path routing
- Property: *reachability of a single destination*

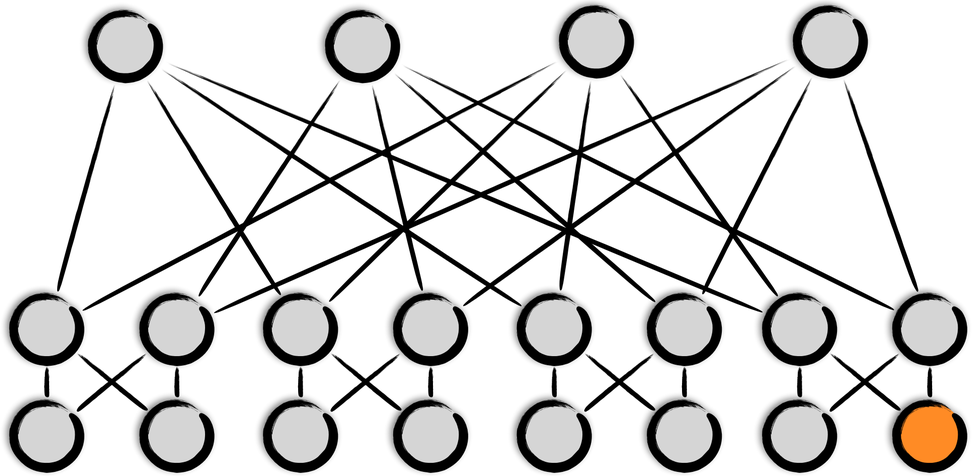


[Fogel et al., NSDI 2015]

[Leiserson, IEEE TC 1985]

Fattree Reachability

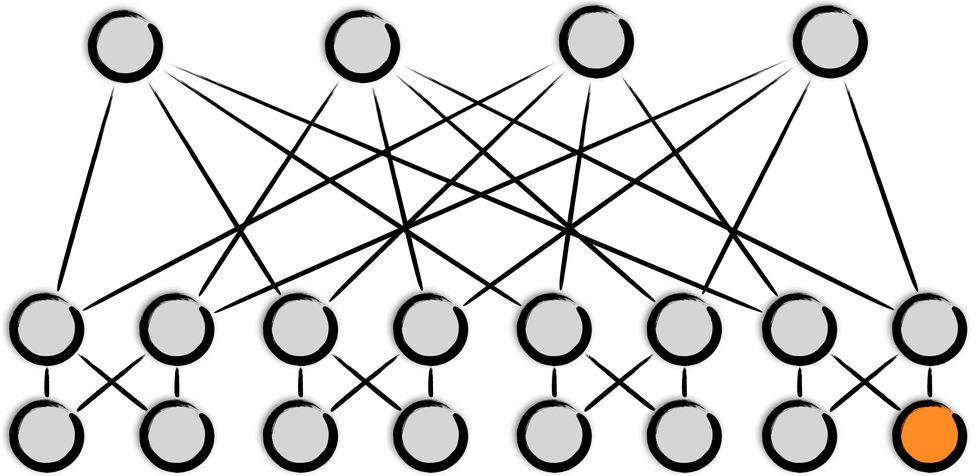
Case Study



Fattree Reachability

Case Study

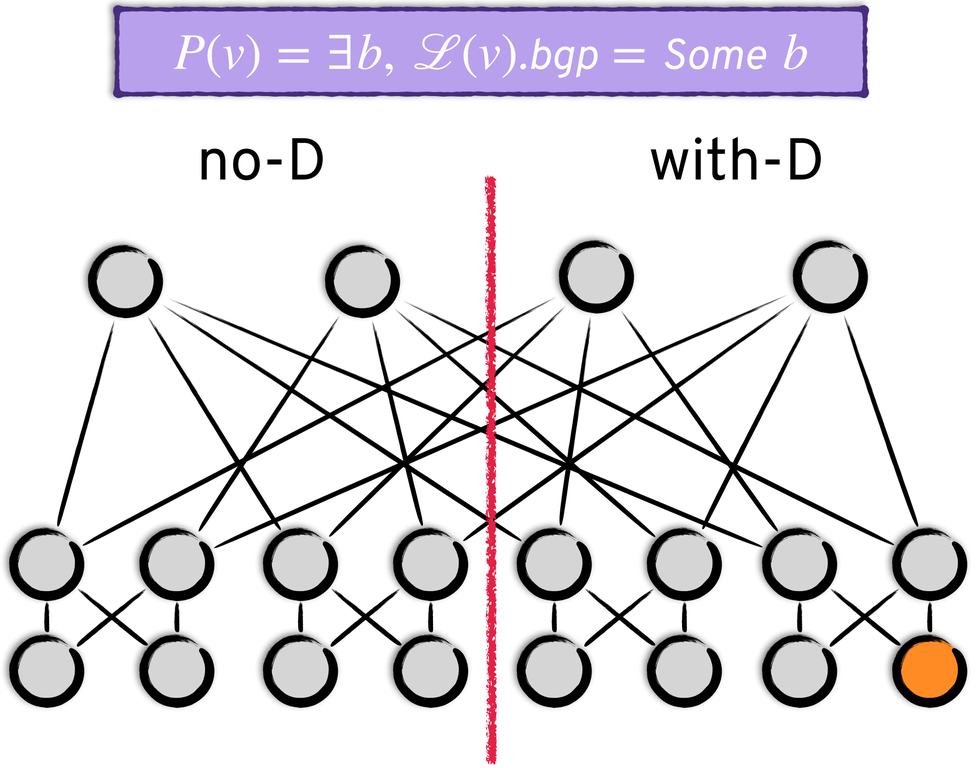
$$P(v) = \exists b,\ \mathscr{L}(v).bgp = Some\ b$$



Fattree Reachability

Case Study

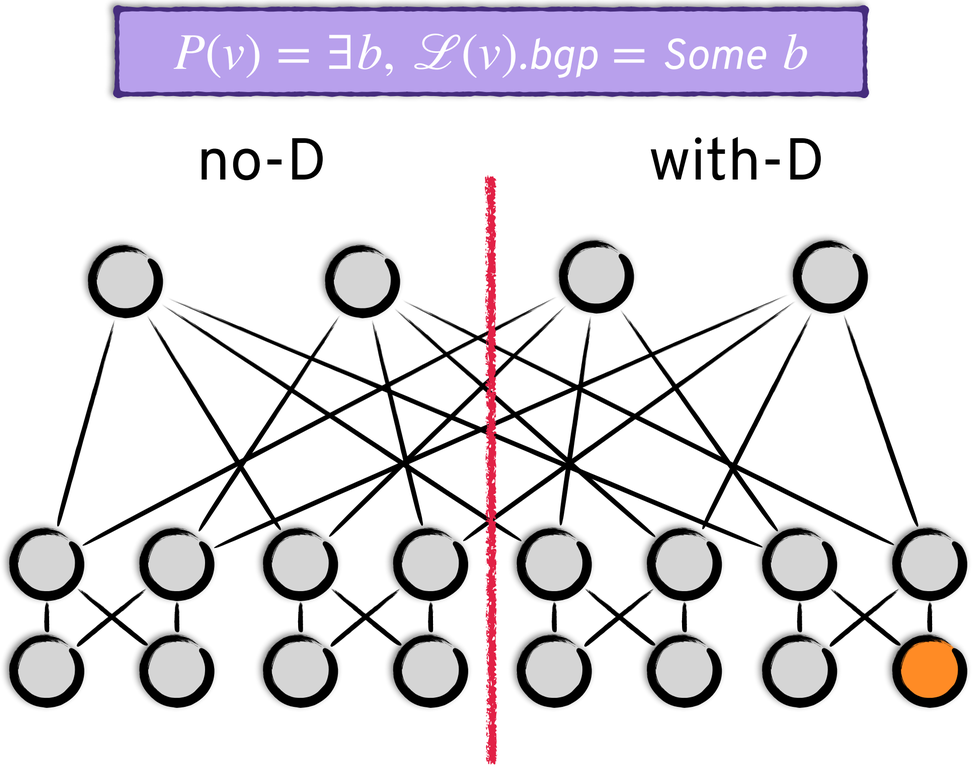
- Partition vertically in half
  - Destination side (with-D)
  - Non-destination side (no-D)



Fattree Reachability

Case Study

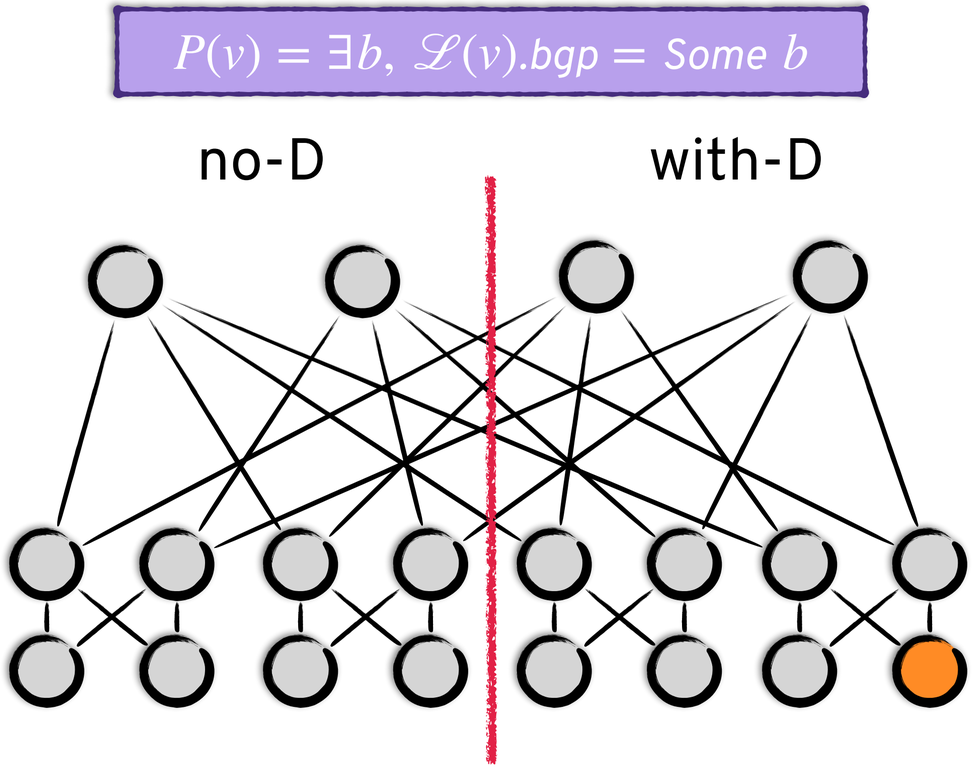
- Partition vertically in half
  - Destination side (with-D)
  - Non-destination side (no-D)
- Hypotheses needed are quite simple!



Fattree Reachability

Case Study

- Partition vertically in half
  - Destination side (with-D)
  - Non-destination side (no-D)
- Hypotheses needed are quite simple!
  - Only need to be specific enough that the property holds

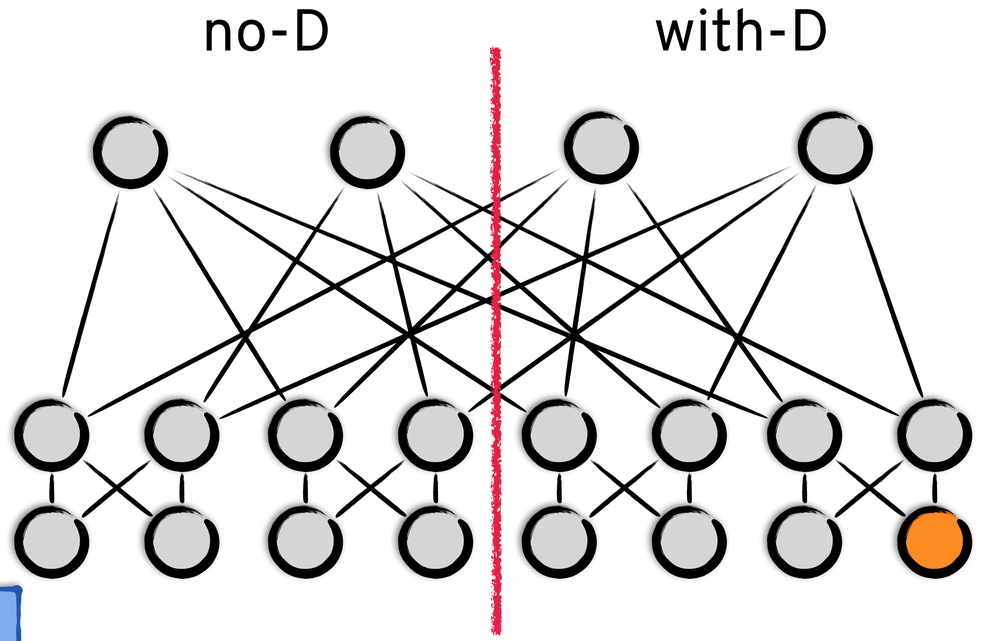


Fattree Reachability

Case Study

- Partition vertically in half
  - Destination side (with-D)
  - Non-destination side (no-D)
- Hypotheses needed are quite simple!
  - Only need to be specific enough that the property holds

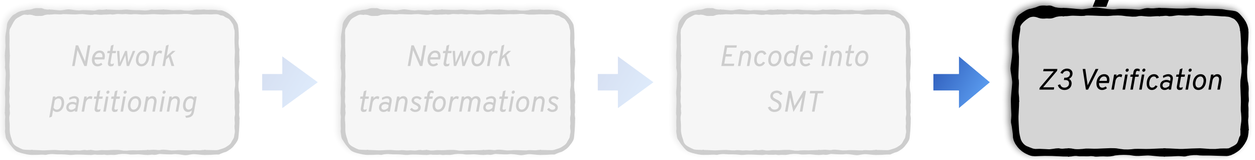
$P(v) = \exists b,\ \mathscr{L}(v).bgp = Some\ b$



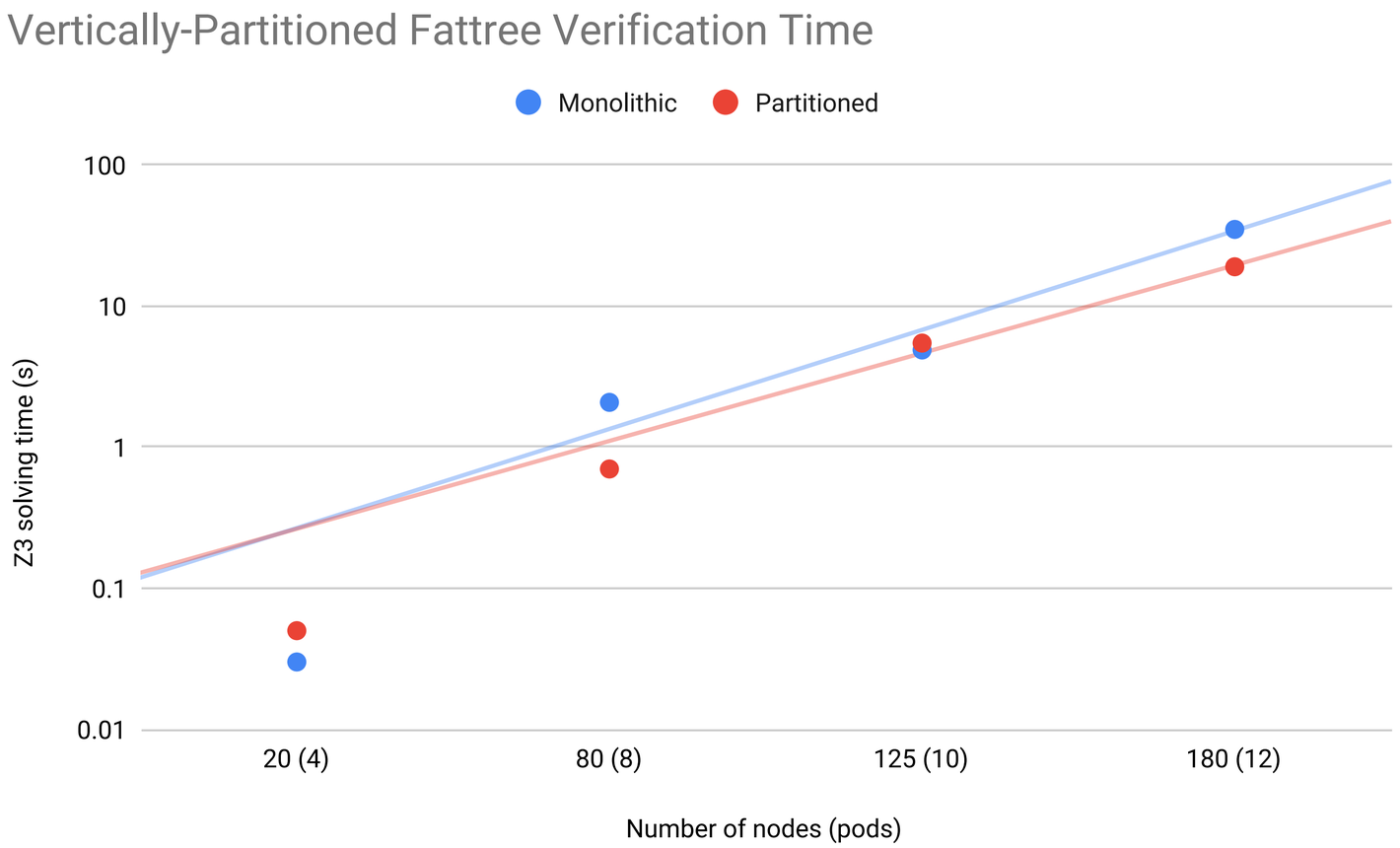
from no-D to with-D: $(\ldots, true)$

from with-D to no-D: $(\ldots, a = (\exists b, a.bgp = Some\ b))$

Fattree Reachability



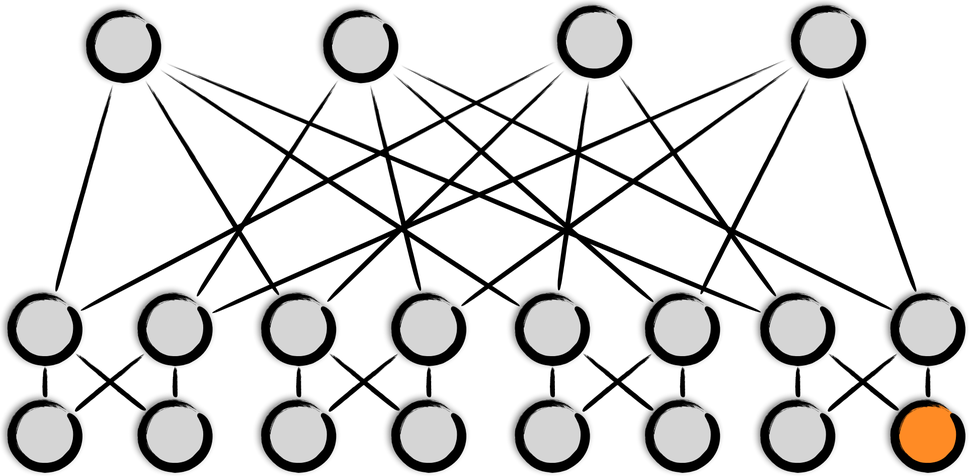
(lower is better)



Fattree Reachability

An alternative cut

$P(v) = \exists b,\ \mathscr{L}(v).bgp = Some\ b$

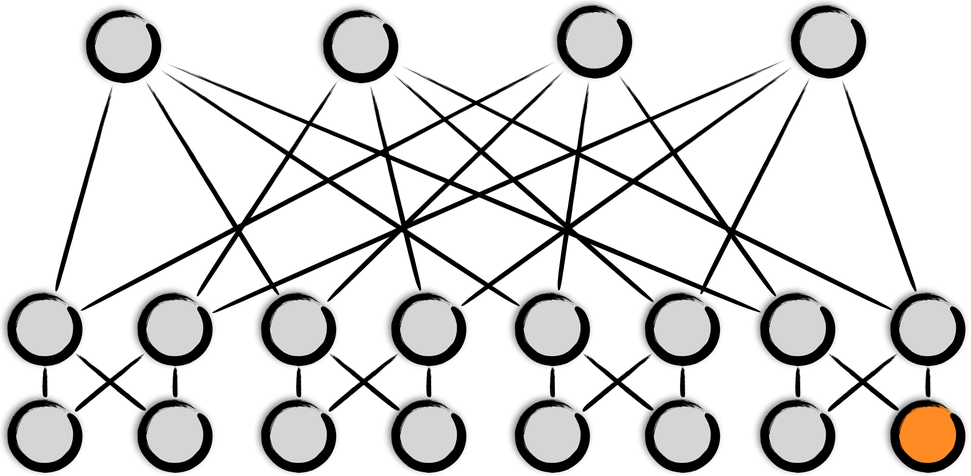


Fattree Reachability

An alternative cut

- Consider an alternative partition scheme

$$P(v) = \exists b,\ \mathscr{L}(v).bgp = Some\ b$$

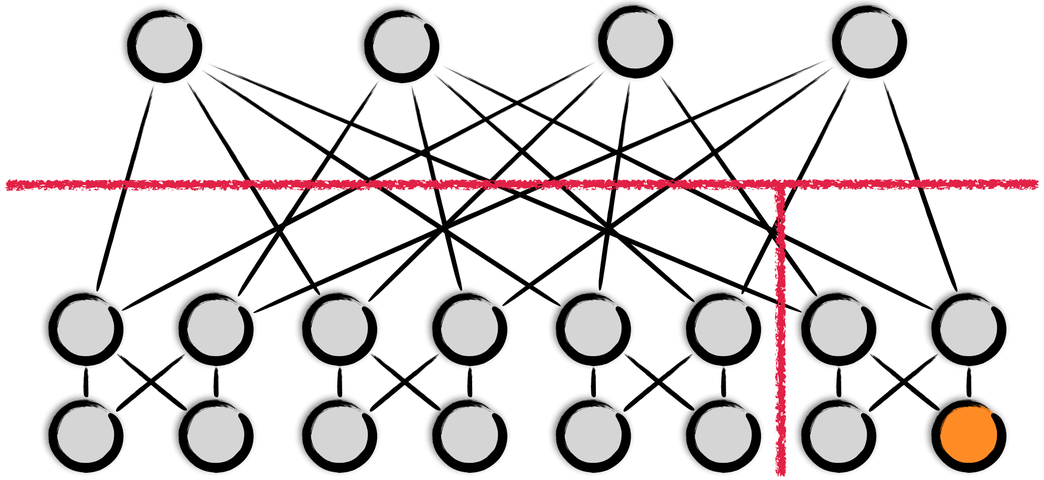


Fattree Reachability

An alternative cut

- Consider an alternative partition scheme
  - Partition horizontally into spines and pods: pod with destination separate from others

$P(v) = \exists b,\ \mathscr{L}(v).bgp = Some\ b$

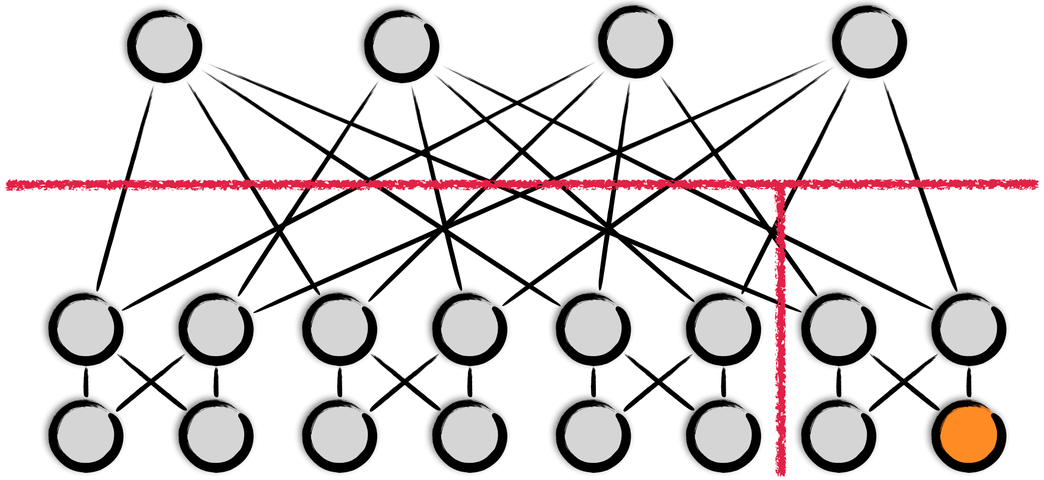


Fattree Reachability

An alternative cut

- Consider an alternative partition scheme
  - Partition horizontally into spines and pods: pod with destination separate from others
  - Should work just as before!

$P(v) = \exists b,\ \mathscr{L}(v).bgp = \text{Some } b$



from pods-no-D to spines: $(\ldots, true)$; *from spines to pods-no-D:* $(\ldots, a = (\exists b, a.bgp = \text{Some } b))$

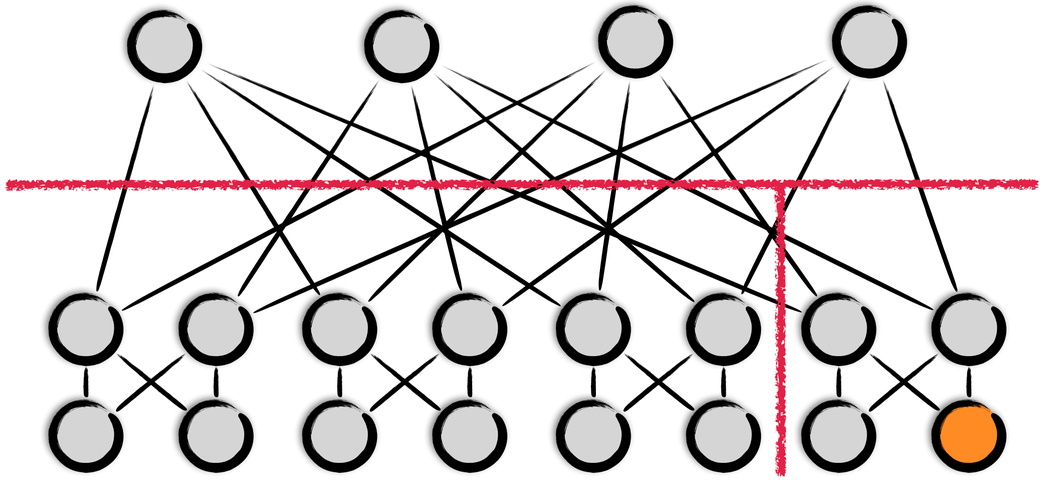
from spines to pod-with-D: $(\ldots, true)$; *from pod-with-D to spines:* $(\ldots, a = (\exists b, a.bgp = \text{Some } b))$

Fattree Reachability

An alternative cut

- Consider an alternative partition scheme
  - Partition horizontally into spines and pods: pod with destination separate from others
  - Should work just as before!
    - Unfortunately, initial check won't hold for spines to pods-no-D

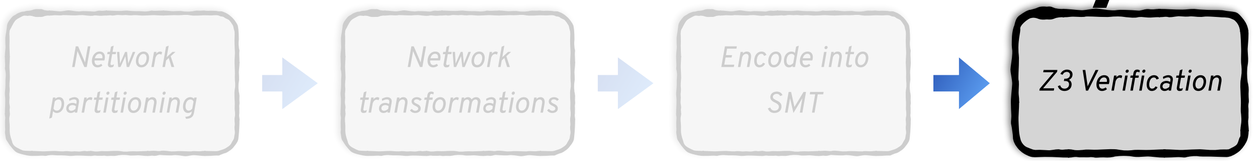
$P(v) = \exists b,\ \mathscr{L}(v).bgp = \text{Some } b$



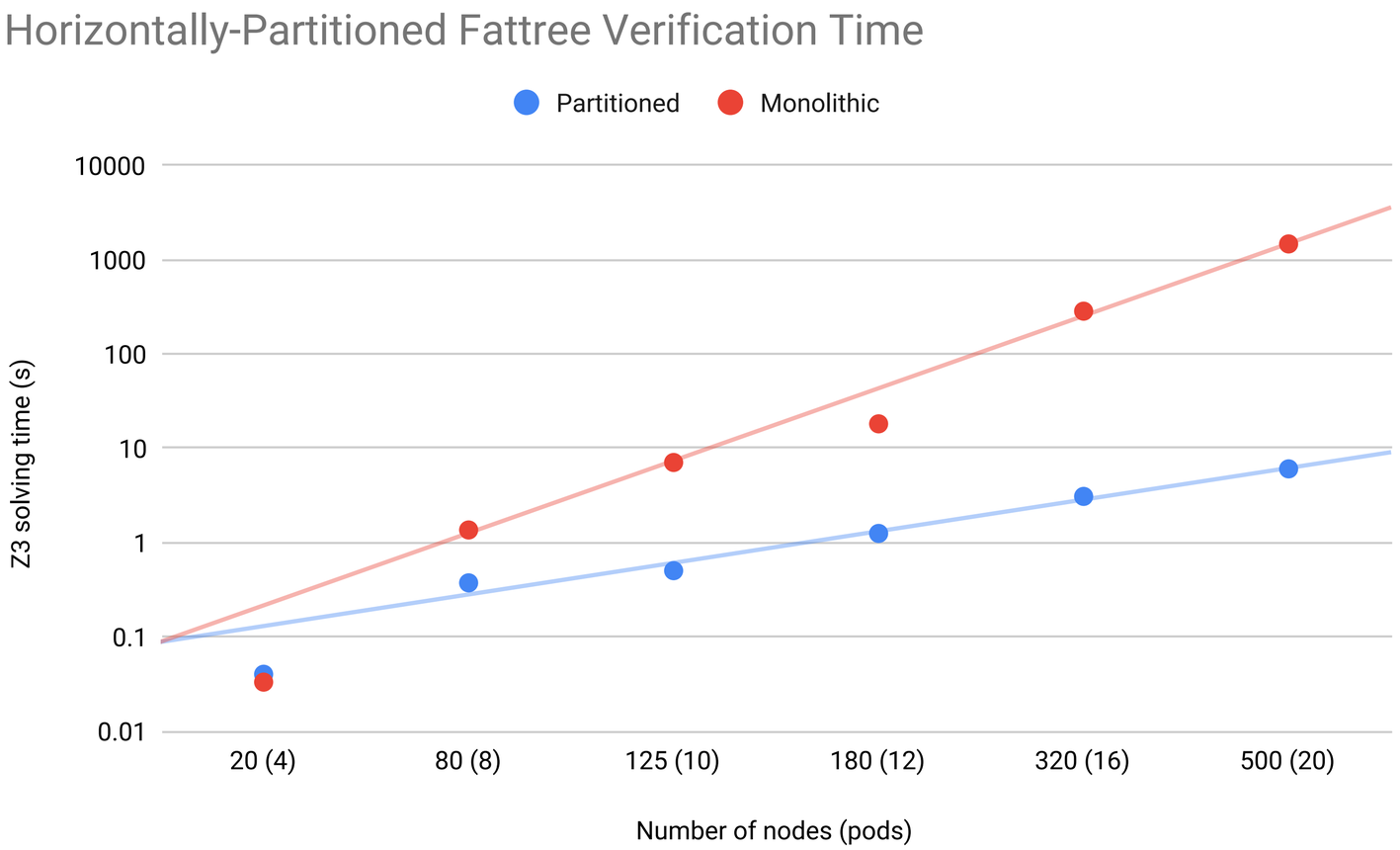
from pods-no-D to spines: $(\ldots, true)$; *from spines to pods-no-D:* $(\ldots, a = (\exists b, a.bgp = \text{Some } b))$

from spines to pod-with-D: $(\ldots, true)$; *from pod-with-D to spines:* $(\ldots, a = (\exists b, a.bgp = \text{Some } b))$

Fattree Reachability



(lower is better)



A New Assume-Guarantee Proof Rule?

[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

A New Assume-Guarantee Proof Rule?

New Rule

$$\frac{\begin{array}{c}\langle A_S\rangle\ S\ \langle G_S\rangle \\ \langle A_T\rangle\ T\ \langle G_T\rangle \\ G_T \Rightarrow A_S \\ G_S \Rightarrow A_T \\ \langle A_S\rangle\ S\ \langle P_S\rangle \\ \langle A_T\rangle\ T\ \langle P_T\rangle \\ \langle \text{true}\rangle\ S\ \langle G_S\rangle\end{array}}{\langle \text{true}\rangle\ S \parallel T\ \langle P_S \wedge P_T\rangle}$$

[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

A New Assume-Guarantee Proof Rule?

New Rule

$$\frac{\begin{array}{c}\langle A_S\rangle\ S\ \langle G_S\rangle \\ \langle A_T\rangle\ T\ \langle G_T\rangle \\ G_T \Rightarrow A_S \\ G_S \Rightarrow A_T \\ \langle A_S\rangle\ S\ \langle P_S\rangle \\ \langle A_T\rangle\ T\ \langle P_T\rangle \\ \langle \text{true}\rangle\ S\ \langle G_S\rangle\end{array}}{\langle \text{true}\rangle\ S \parallel T\ \langle P_S \wedge P_T\rangle}$$

Rule CIRC

$$\frac{\begin{array}{c}\langle A_2\rangle\ M_2\ \langle A_1\rangle \\ \langle A_1\rangle\ M_1\ \langle P\rangle \\ \langle \text{true}\rangle\ M_1\ \langle A_2\rangle\end{array}}{\langle \text{true}\rangle\ M_1 \parallel M_2\ \langle P\rangle}$$

[Giannakopoulou et al., *Handbook of Model Checking.* 2018]

Related Work and Future Directions

Related Work

Related Work

- Network verification

Related Work

- Network verification
  - Control plane verification

Related Work

- Network verification
  - Control plane verification
    - **Minesweeper** [Beckett et al., SIGCOMM 2017]: encodes a network into SMT and verifies it *monolithically*

Related Work

- Network verification
  - Control plane verification
    - **Minesweeper** [Beckett et al., SIGCOMM 2017]: encodes a network into SMT and verifies it *monolithically*
    - **Bonsai** [Beckett et al., SIGCOMM 2018]: compresses *symmetric* networks

Related Work

- Network verification
  - Control plane verification
    - **Minesweeper** [Beckett et al., SIGCOMM 2017]: encodes a network into SMT and verifies it *monolithically*
    - **Bonsai** [Beckett et al., SIGCOMM 2018]: compresses *symmetric* networks
    - **Shapeshifter** [Beckett et al., POPL 2020]: analyzes properties using *abstract interpretation*

Related Work

- Network verification
  - Control plane verification
    - **Minesweeper** [Beckett et al., SIGCOMM 2017]: encodes a network into SMT and verifies it *monolithically*
    - **Bonsai** [Beckett et al., SIGCOMM 2018]: compresses *symmetric* networks
    - **Shapeshifter** [Beckett et al., POPL 2020]: analyzes properties using *abstract interpretation*
  - Data plane verification

Related Work

- Network verification
  - Control plane verification
    - **Minesweeper** [Beckett et al., SIGCOMM 2017]: encodes a network into SMT and verifies it *monolithically*
    - **Bonsai** [Beckett et al., SIGCOMM 2018]: compresses *symmetric* networks
    - **Shapeshifter** [Beckett et al., POPL 2020]: analyzes properties using *abstract interpretation*
  - Data plane verification
    - **Symmetry and Surgery** [Plotkin et al., SIGPLAN 2016]: *slices* network topology and abstracts packet headers

Related Work

- Network verification
  - Control plane verification
    - **Minesweeper** [Beckett et al., SIGCOMM 2017]: encodes a network into SMT and verifies it *monolithically*
    - **Bonsai** [Beckett et al., SIGCOMM 2018]: compresses *symmetric* networks
    - **Shapeshifter** [Beckett et al., POPL 2020]: analyzes properties using *abstract interpretation*
  - Data plane verification
    - **Symmetry and Surgery** [Plotkin et al., SIGPLAN 2016]: *slices* network topology and abstracts packet headers
    - **SecGuru** [Jayaraman et al., SIGCOMM 2019]: checks local behavior using *operator-provided specifications*

Related Work

- Network verification
  - Control plane verification
    - **Minesweeper** [Beckett et al., SIGCOMM 2017]: encodes a network into SMT and verifies it *monolithically*
    - **Bonsai** [Beckett et al., SIGCOMM 2018]: compresses *symmetric* networks
    - **Shapeshifter** [Beckett et al., POPL 2020]: analyzes properties using *abstract interpretation*
  - Data plane verification
    - **Symmetry and Surgery** [Plotkin et al., SIGPLAN 2016]: *slices* network topology and abstracts packet headers
    - **SecGuru** [Jayaraman et al., SIGCOMM 2019]: checks local behavior using *operator-provided specifications*
- Compositional verification of programs

Related Work

- Network verification
  - Control plane verification
    - **Minesweeper** [Beckett et al., SIGCOMM 2017]: encodes a network into SMT and verifies it *monolithically*
    - **Bonsai** [Beckett et al., SIGCOMM 2018]: compresses *symmetric* networks
    - **Shapeshifter** [Beckett et al., POPL 2020]: analyzes properties using *abstract interpretation*
  - Data plane verification
    - **Symmetry and Surgery** [Plotkin et al., SIGPLAN 2016]: *slices* network topology and abstracts packet headers
    - **SecGuru** [Jayaraman et al., SIGCOMM 2019]: checks local behavior using *operator-provided specifications*
- Compositional verification of programs
  - **Assume-guarantee reasoning** [Owicki and Gries, ACM 1976]: reasons about *non-interference* of shared variables

Related Work

- Network verification
  - Control plane verification
    - **Minesweeper** [Beckett et al., SIGCOMM 2017]: encodes a network into SMT and verifies it *monolithically*
    - **Bonsai** [Beckett et al., SIGCOMM 2018]: compresses *symmetric* networks
    - **Shapeshifter** [Beckett et al., POPL 2020]: analyzes properties using *abstract interpretation*
  - Data plane verification
    - **Symmetry and Surgery** [Plotkin et al., SIGPLAN 2016]: *slices* network topology and abstracts packet headers
    - **SecGuru** [Jayaraman et al., SIGCOMM 2019]: checks local behavior using *operator-provided specifications*
- Compositional verification of programs
  - **Assume-guarantee reasoning** [Owicki and Gries, ACM 1976]: reasons about *non-interference* of shared variables
  - **Split invariants** [Cohen et al., CAV 2010]: uses *vector of assertions* defined over local state variables

Future Work

Future Work

- Improving compilation time to SMT

Future Work

- Improving compilation time to SMT
- Proving a less restrictive initial check

Future Work

- Improving compilation time to SMT
- Proving a less restrictive initial check
- Inferring interface hypotheses

Future Work

- Improving compilation time to SMT
- Proving a less restrictive initial check
- Inferring interface hypotheses
  - Given a network and a cut, what should the hypotheses be?

Future Work

- Improving compilation time to SMT
- Proving a less restrictive initial check
- Inferring interface hypotheses
  - Given a network and a cut, what should the hypotheses be?
  - Possible opportunity for CEGAR-style refinement

Future Work

- Improving compilation time to SMT
- Proving a less restrictive initial check
- Inferring interface hypotheses
  - Given a network and a cut, what should the hypotheses be?
  - Possible opportunity for CEGAR-style refinement
- Designing where to cut the network

Future Work

- Improving compilation time to SMT
- Proving a less restrictive initial check
- Inferring interface hypotheses
  - Given a network and a cut, what should the hypotheses be?
  - Possible opportunity for CEGAR-style refinement
- Designing where to cut the network
  - Consider graph properties, natural structure of topology


Thank you!

Appendix

Kirigami with an Alternate Initial Check

For partition S (resp. T), check validity of the following *verification conditions* (VCs) on $\mathscr{M}_S$ (resp. $\mathscr{M}_T$, swapping S and T):

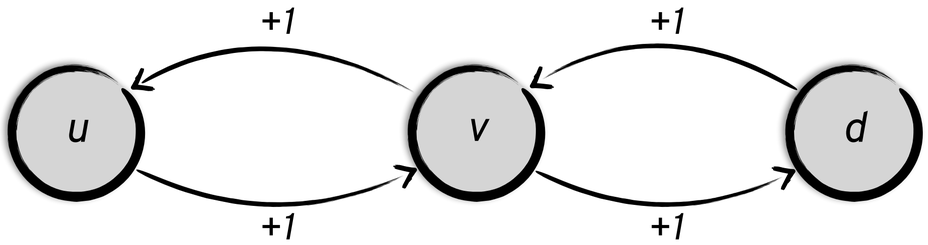
A. Inductiveness: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$

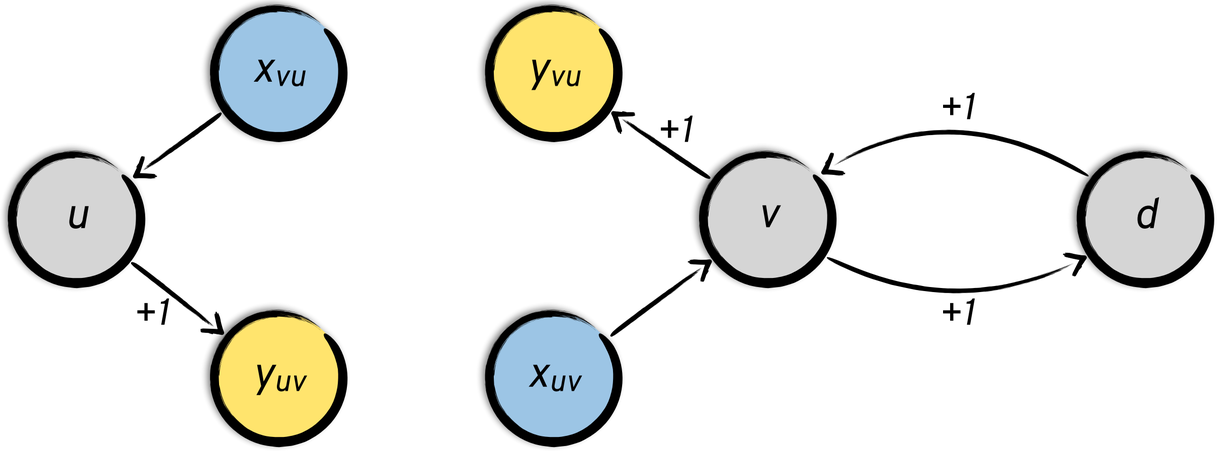
B. Safety: $\langle \mathbf{H}_S \rangle \mathscr{M}_S \langle P_S \rangle$

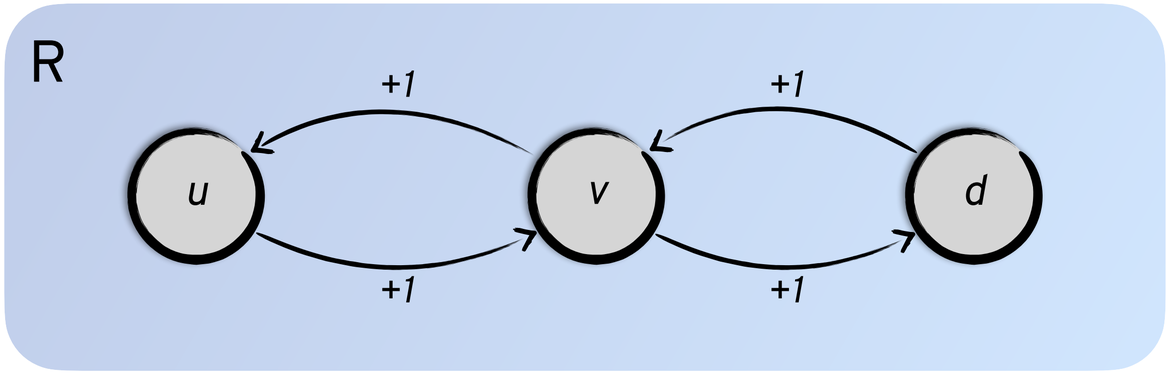
C. Initial: $\langle \text{true} \rangle \mathscr{M}_S \langle G_S \rangle \wedge G_S \Rightarrow \mathbf{H}_T$

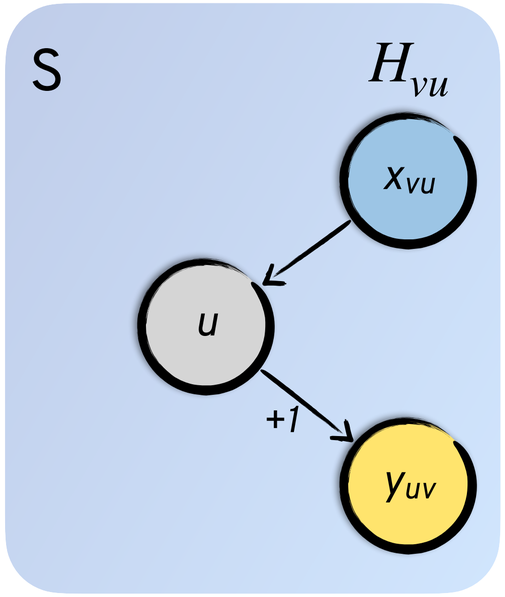
If A and B hold for both S and T, and *C holds for* S *or* T, then return true

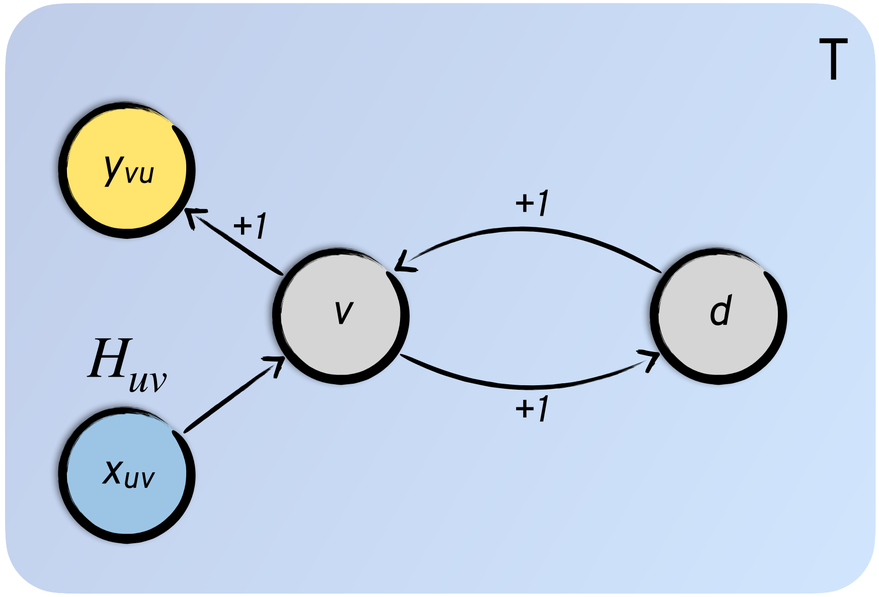
Drafts


+1
+1
u
v
d
+1
+1


x_{vu}
u
+1
y_{uv}
y_{vu}
+1
+1
v
d
+1
x_{uv}


R
+1
+1
u
v
d
+1
+1


S
H_{vu}
x_{vu}
u
+1
y_{uv}


T
y_{vu}
+1
+1
v
d
H_{uv}
+1
x_{uv}