Hypernet semantics and robust observational equivalence

Koko Muroya
(RIMS, Kyoto University)

joint work with
Dan R. Ghica & Todd Waugh Ambridge
(University of Birmingham)

CMCS 2020 (online), 12 October 2020

Overview

1. Motivation: robustness of observational equivalence
2. Hypernet semantics
3. Locality & step-wise reasoning
4. Discussion: complication of simulation notion

Overview

1. Motivation: robustness of observational equivalence
2. Hypernet semantics
3. Locality & step-wise reasoning
4. Discussion: complication of simulation notion

Observational equivalence on program fragments

“Do two program fragments behave the same?”

“Is it safe to replace a program fragment with another?”

```
let x = 100 in
let y = 50 in
y + y
```

Observational equivalence on program fragments

“Do two program fragments behave the same?”

“Is it safe to replace a program fragment with another?”

```
let x = 100 in
let y = 50 in
y + y
```

Observational equivalence on program fragments

“Do two program fragments behave the same?”

“Is it safe to replace a program fragment with another?”

```
let x = 100 in
let y = 50 in
y + y
```

```
let y = 50 in
y + y
```

Observational equivalence on program fragments

“Do two program fragments behave the same?”

“Is it safe to replace a program fragment with another?”

```
let x = 100 in
let y = 50 in
y + y
```

→

```
let y = 50 in
y + y
```

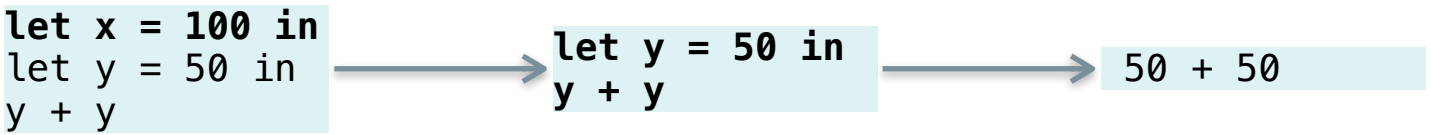
→

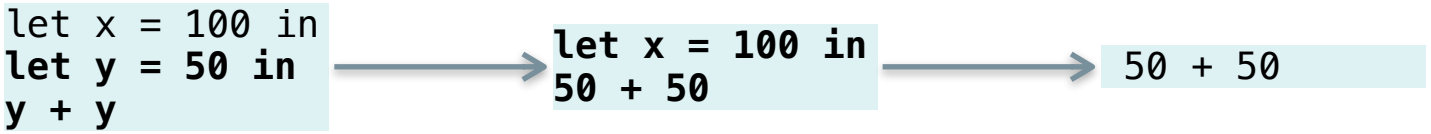
```
50 + 50
```

Observational equivalence on program fragments

"Do two program fragments behave the same?"

"Is it safe to replace a program fragment with another?"

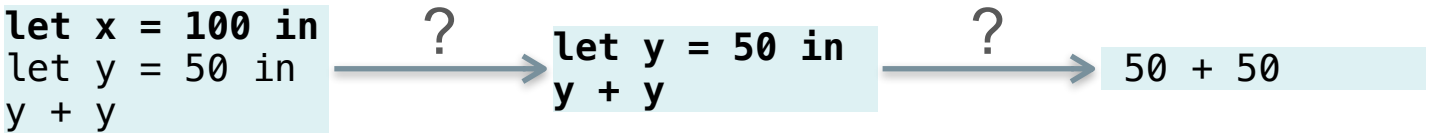


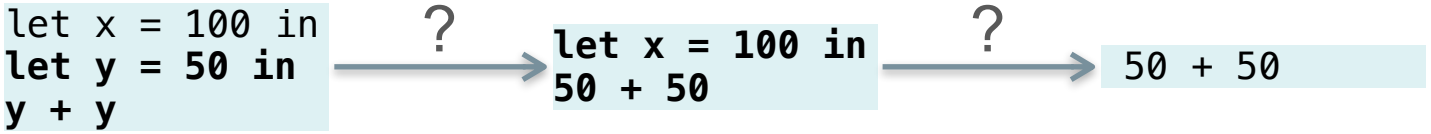


Observational equivalence on program fragments

"Do two program fragments behave the same?"

"Is it safe to replace a program fragment with another?"

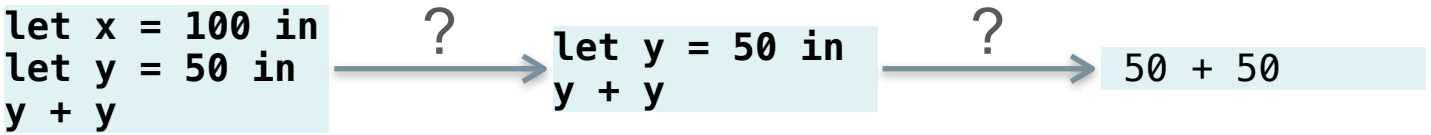


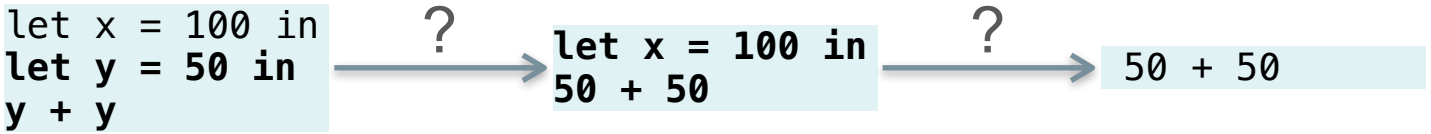


Observational equivalence on program fragments

"Do two program fragments behave the same?"

"Is it safe to replace a program fragment with another?"





If YES (*"Two program fragments are observationally equal."*):

- justification of compiler optimisation
- program verification

Observational equivalence on program fragments

“Do two program fragments behave the same?”

Observational equivalence on program fragments

"~~Do two~~ program fragments behave the same?"

*"**What** program fragments behave the same?"*

the beta-law

$$(\lambda x\,.\,M)\,N \;\simeq\; M[x := N]$$

a parametricity law

$$\texttt{let}\ a = \texttt{ref}\ 1\ \texttt{in}\ \lambda x\,.\,(a := 2; !a) \;\simeq\; \lambda x.2$$

Robustness of observational equivalence

"~~Do~~ two program fragments behave the same?"

*"**When do** program fragments behave the same?"*

the beta-law

$$(\lambda x\,.\,M)\,N \;\simeq\; M[x := N]$$

Does the beta-law always hold?

Robustness of observational equivalence

"~~Do~~ two program fragments behave the same?"

*"**When do** program fragments behave the same?"*

the beta-law

$$(\lambda x\,.\,M)\,N \;\simeq\; M[x := N]$$

Does the beta-law always hold?

No, it's violated if program contexts use OCaml's Gc module:

$$(\lambda x.0)\,100 \;\not\simeq\; 0$$

for memory management

Robustness of observational equivalence

"~~Do~~ two program fragments behave the same?"

*"**When do** program fragments behave the same?"*

the beta-law

$$(\lambda x\,.\,M)\,N \;\simeq\; M[x := N]$$

Does the beta-law always hold?

No, it's violated if program contexts use OCaml's Gc module:

$$(\lambda x.0)\,100 \;\not\simeq\; 0$$

for memory management

How **robust** is the beta-law then?

Robustness of observational equivalence

~~*"Do two program fragments behave the same?"*~~

*"What fragments, **in which contexts,** behave the same?"*

Robustness of observational equivalence

~~*"Do two program fragments behave the same?"*~~

*"What fragments, **in which contexts,** behave the same?"*

… in the presence of (arbitrary) language features:

pure vs. effectful (e.g. `50 + 50` vs. `ref 1`)

encoded vs. native (e.g. `State` vs. `ref`)

extrinsics (e.g. `Gc.stat`)

foreign language calls

Robustness of observational equivalence

~~*"Do two program fragments behave the same?"*~~

*"What fragments, **in which contexts,** behave the same?"*

… in the presence of (arbitrary) language features

Our (big) goal:

analysing robustness/fragility of observational equivalence,

using a general framework

Robustness of observational equivalence

~~*"Do two program fragments behave the same?"*~~

*"What fragments, **in which contexts,** behave the same?"*

… in the presence of (arbitrary) language features

Our result:

analysing robustness/fragility of observational equivalence,

using a *graphical* framework

- hypernet semantics: a *graphical* abstract machine
- *local & step-wise* reasoning to prove observational equivalence, with the concept of *robustness*

Overview

1. Motivation: robustness of observational equivalence

2. Hypernet semantics

3. Locality & step-wise reasoning

4. Discussion: complication of simulation notion

Hypernet semantics

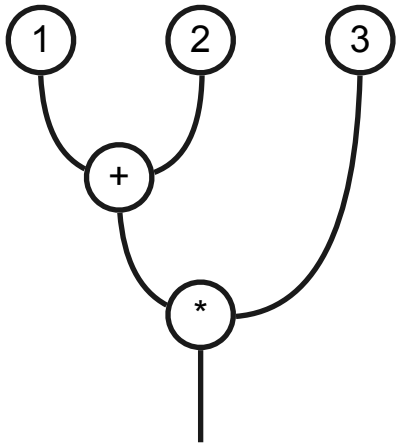
- program execution by a *graphical* abstract machine
  - programs as

    certain hierarchical hypergraphs (*“hypernets”*)
  - execution as

    step-by-step strategical update of hypernets

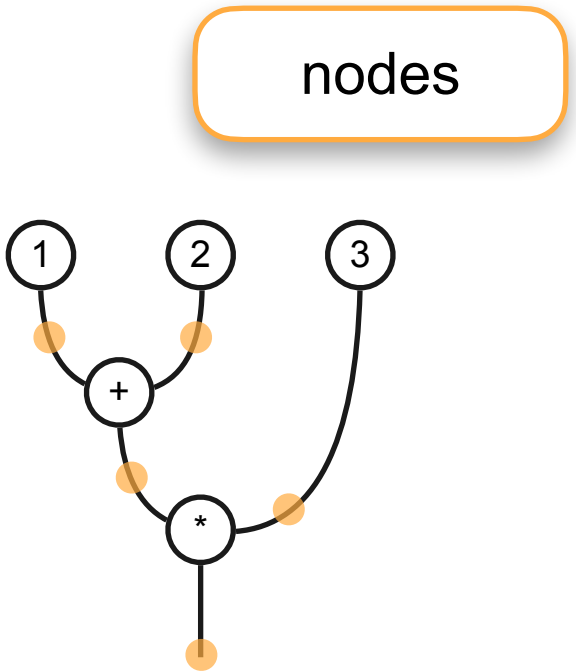
Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)
`(1 + 2) * 3`	

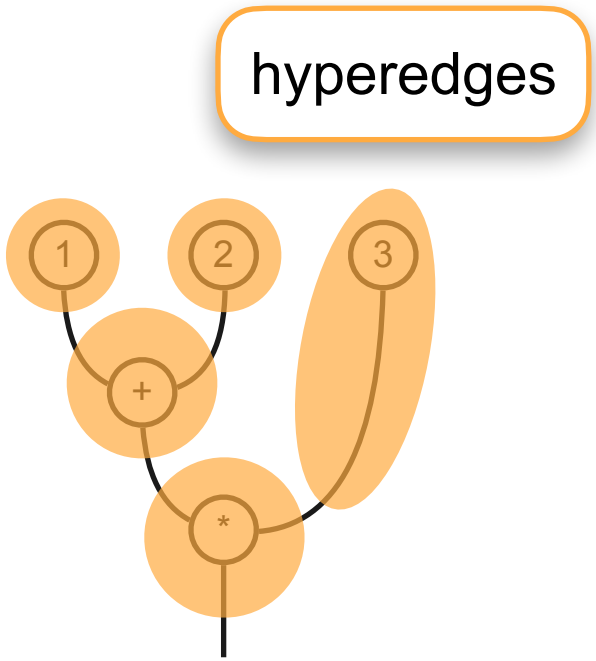
Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)
`(1 + 2) * 3`	

Programs, graphically as *hypernets*

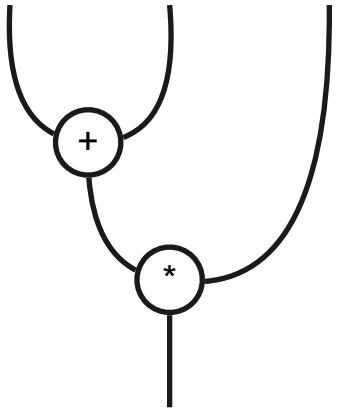
Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)
`(1 + 2) * 3`	

Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

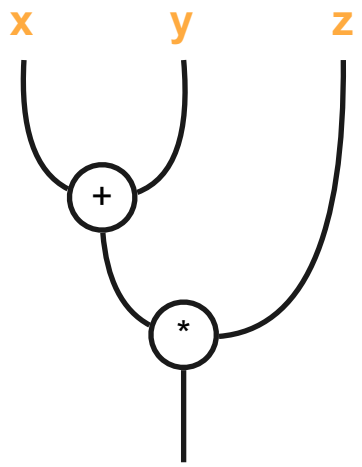
program	hypernet (hierarchical hypergraph)

`(x + y) * z`
 	
`(i + j) * k`
	

Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

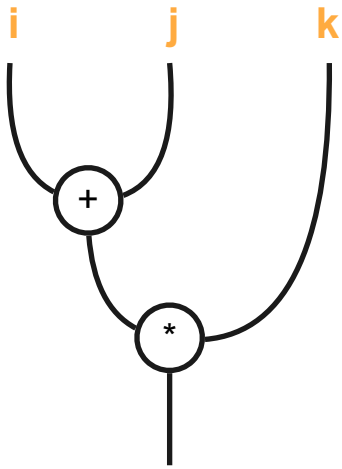
program	hypernet (hierarchical hypergraph)

`(x + y) * z`
 	
`(i + j) * k`
	

Programs, graphically as *hypernets*

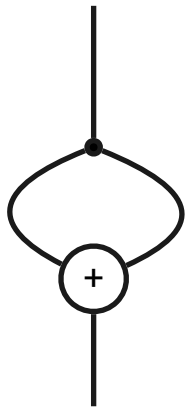
Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)

`(x + y) * z`
 	
`(i + j) * k`
	

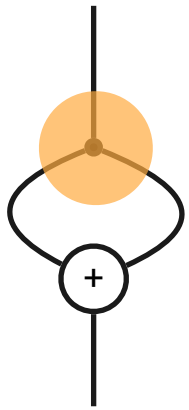
Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)
`x + x`	

Programs, graphically as *hypernets*

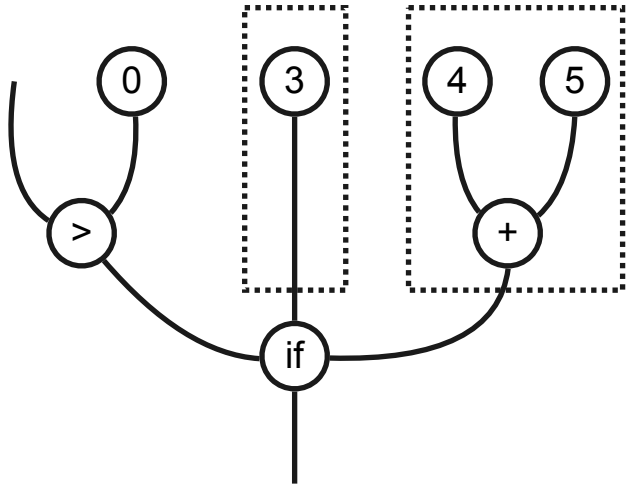
Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)
`x + x`	sharing 

Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

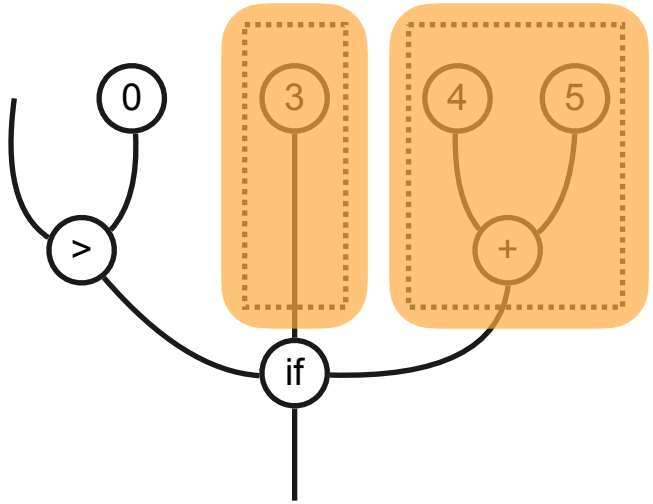
program	hypernet (hierarchical hypergraph)

`if x > 0`
`then 3`
`else 4 + 5`
	

Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)

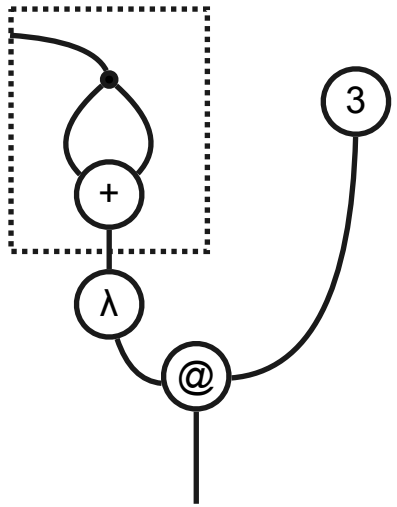
`if x > 0`
`then 3`
`else 4 + 5`
	

hierarchical hyperedge
(hyperedge labelled with hypergraph)

…representing *deferred* computation

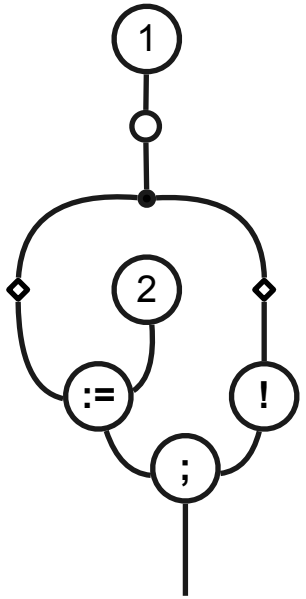
Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)
`(λx. x + x) 3`	

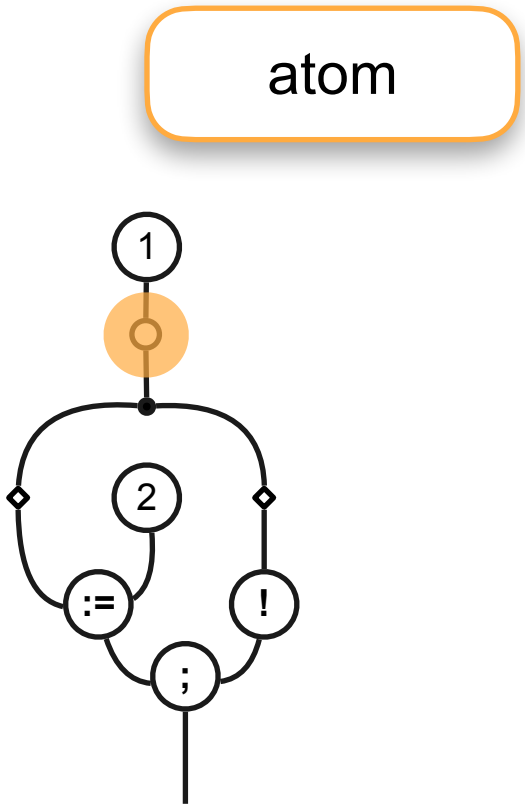
Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)
`new a = 1 in` `a := 2; !a`	

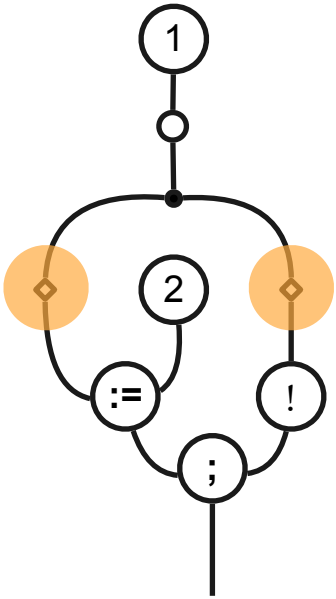
Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)
`new a = 1 in` `a := 2; !a`	

Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

program	hypernet (hierarchical hypergraph)
`new a = 1 in` `a := 2; !a`	atom occurences 

Programs, graphically as *hypernets*

Idea: abstracting away variable names, and more…

Programs, graphically as *hypernets*

Idea: abstracting away variable names, and ~~more…~~

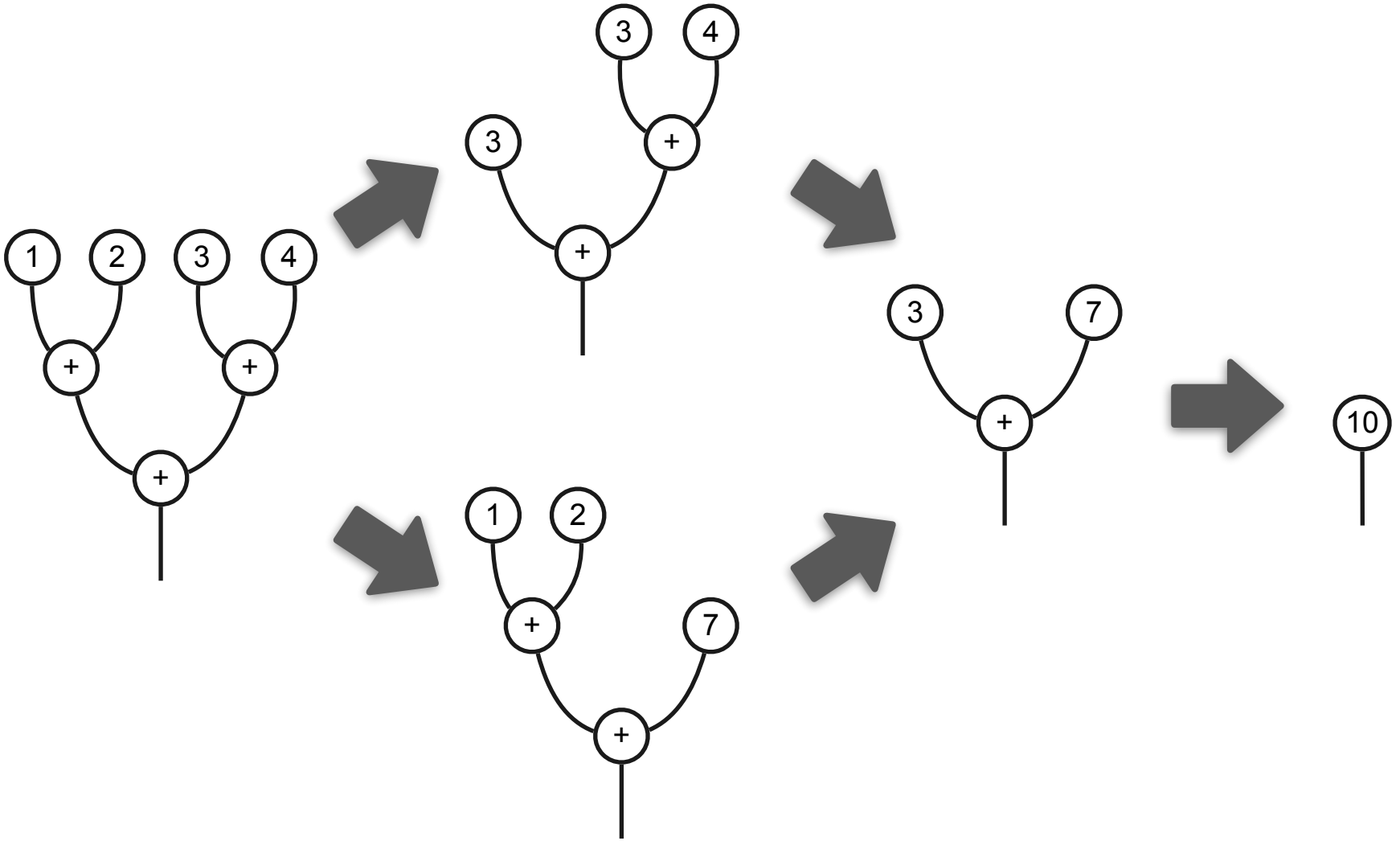
- making blocks of deferred computation explicit
- accommodating atoms (reference names/locations)

Program execution, graphically

Idea: updating hypernets step-by-step

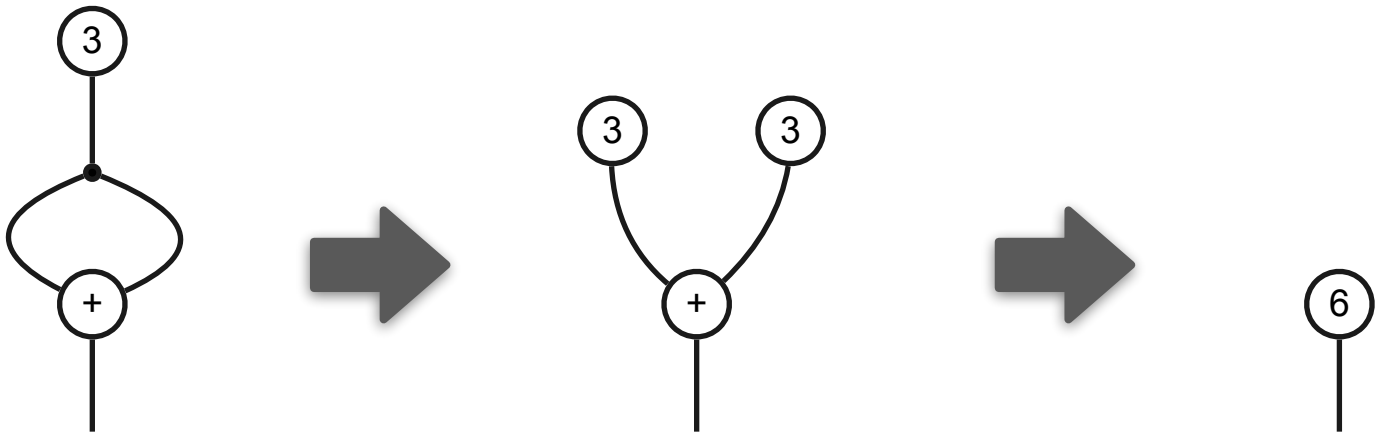
Program execution, graphically

Idea: updating hypernets step-by-step



Program execution, graphically

Idea: updating hypernets step-by-step

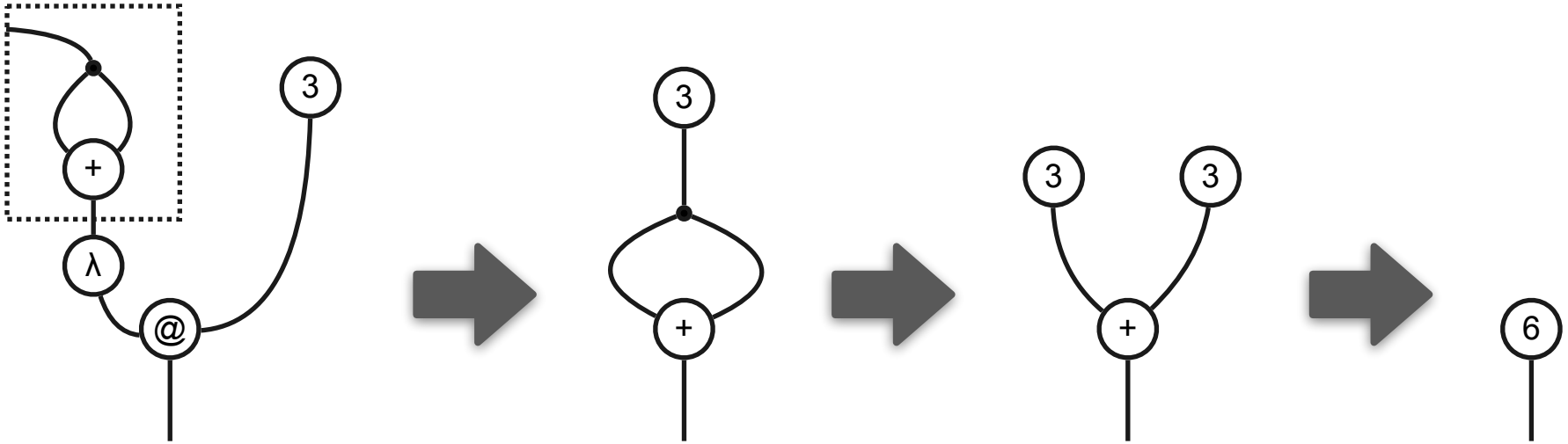



```
let x = 3 in
x + x
```

```
3 + 3
```

Program execution, graphically

Idea: updating hypernets step-by-step




```
(λx. x + x) 3
```

```
let x = 3 in
x + x
```

```
3 + 3
```

Program execution, graphically

Idea: updating hypernets step-by-step

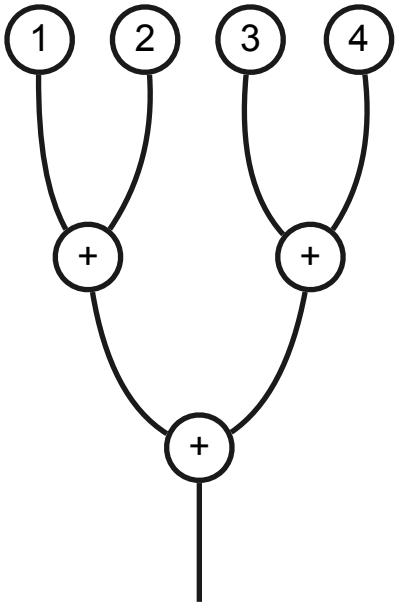
… and strategically, using *focus* with three modes:

- ? depth-first redex search
- ✓ backtracking
- ↯ triggering update of hypernet

Program execution, graphically

Idea: updating hypernets step-by-step

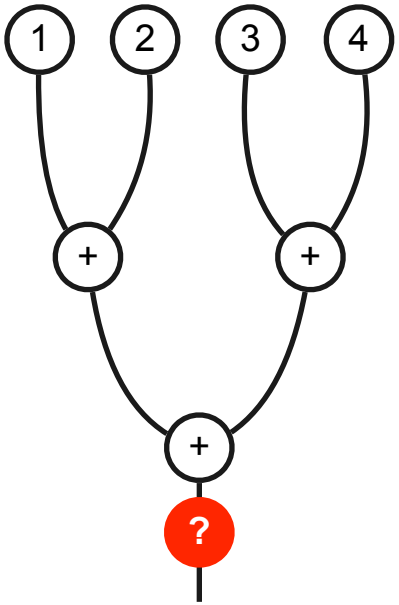
… and strategically, using *focus*



Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

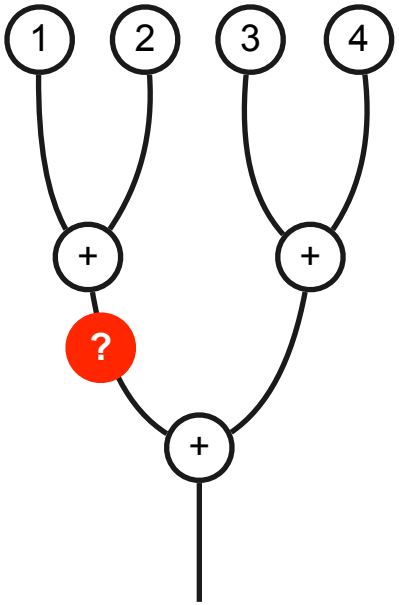


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

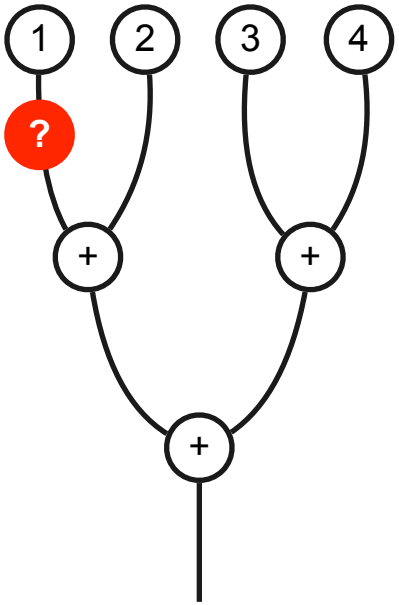


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

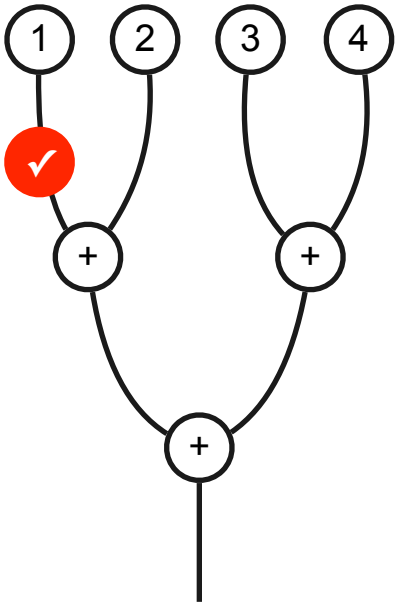


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

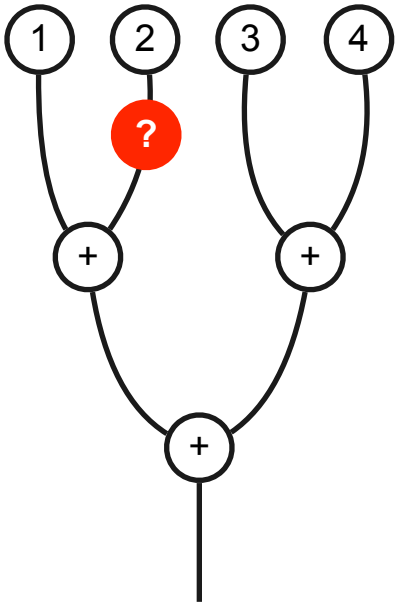


backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

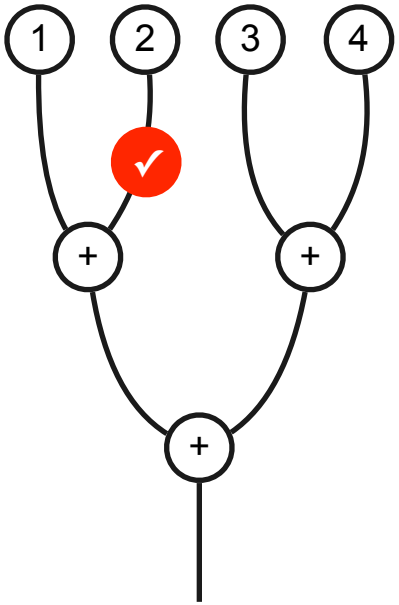


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

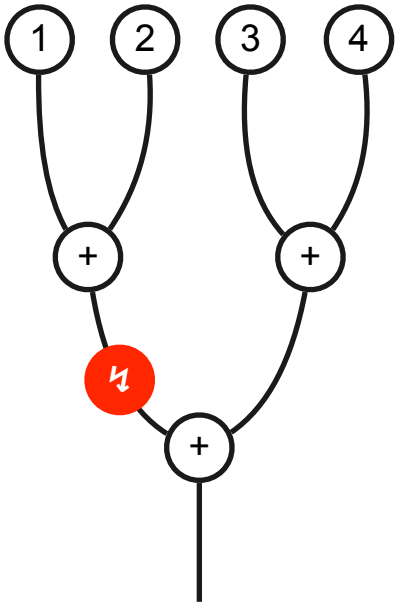


backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

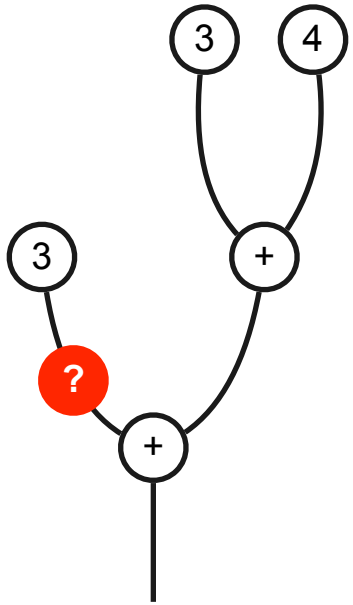


triggering update of hypernet

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

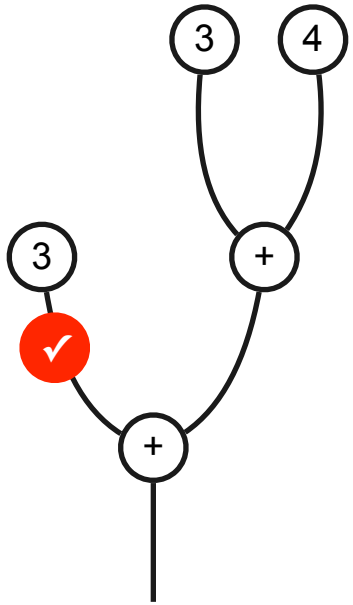


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

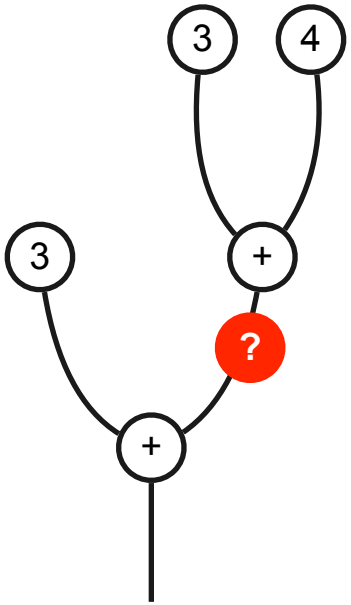


backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

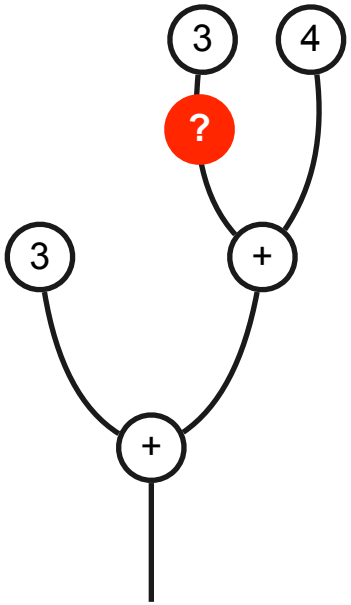


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

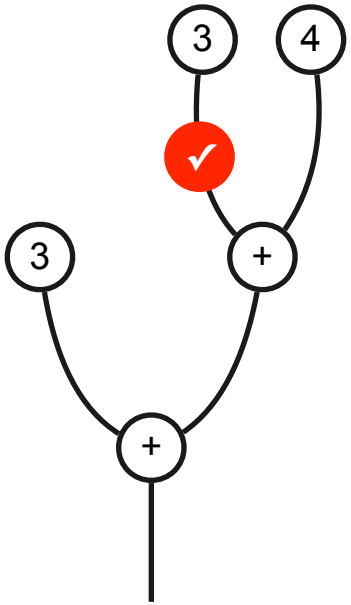


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

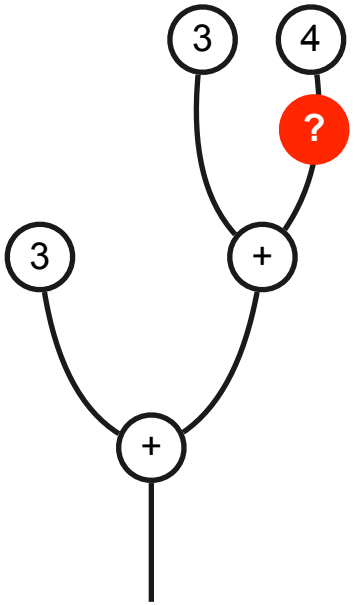


backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

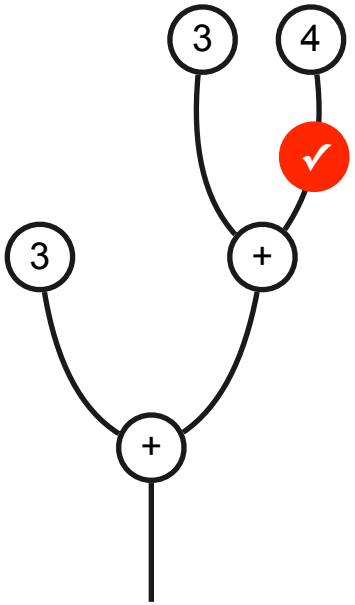


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

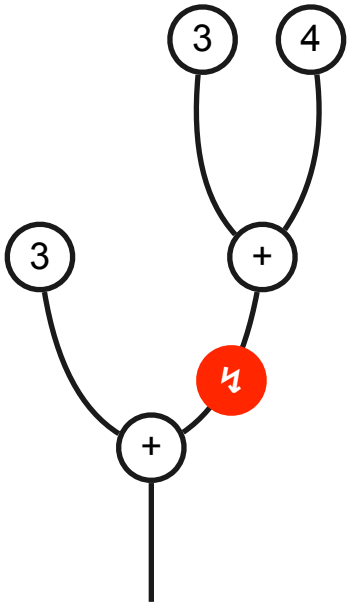


backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*



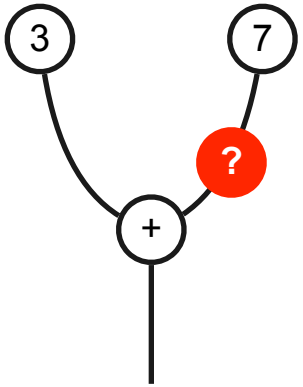
triggering update of hypernet

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

depth-first redex search

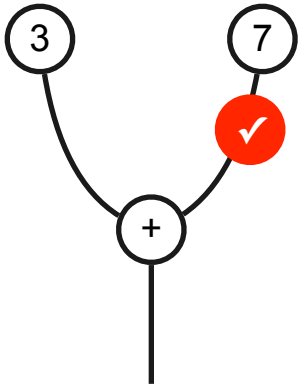


Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

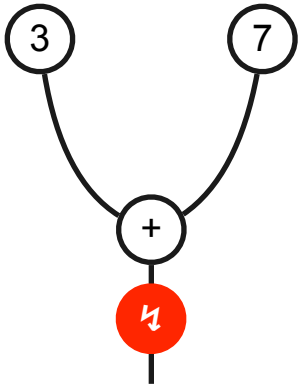
backtracking



Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*



triggering update of hypernet

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

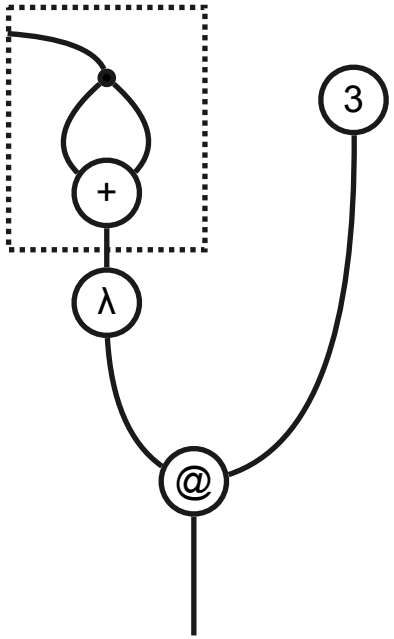
… and strategically, using *focus*

backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

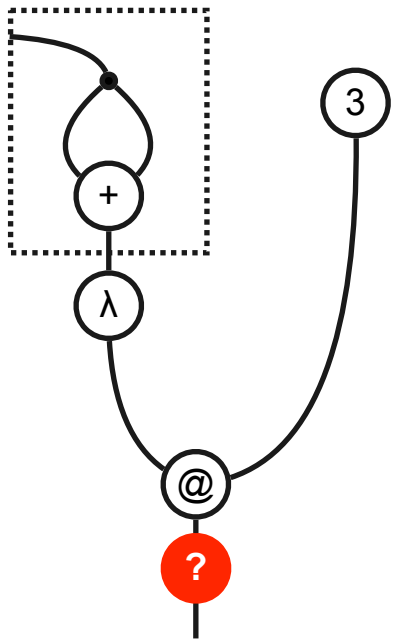
… and strategically, using *focus*



Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

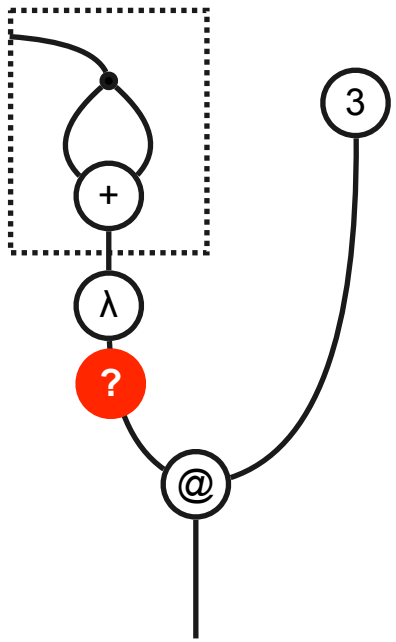


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

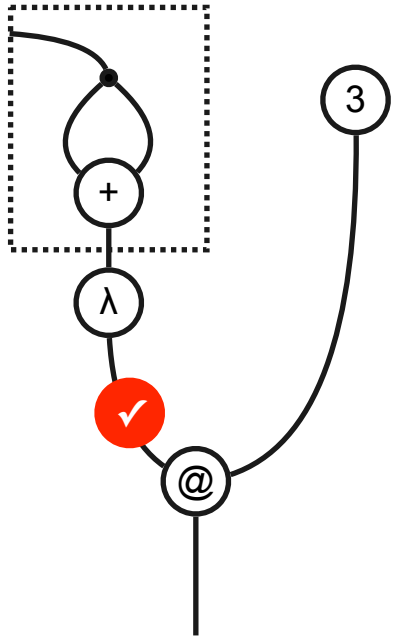


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

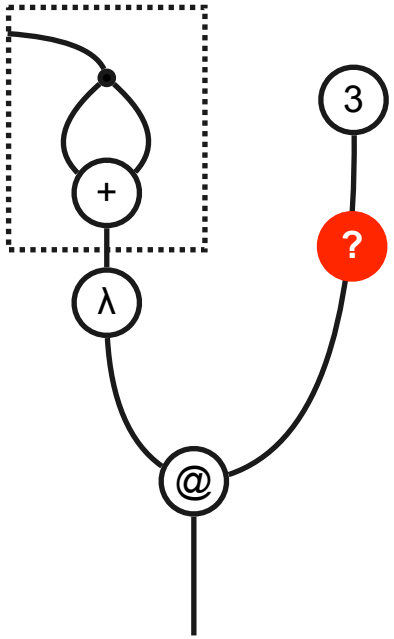


backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

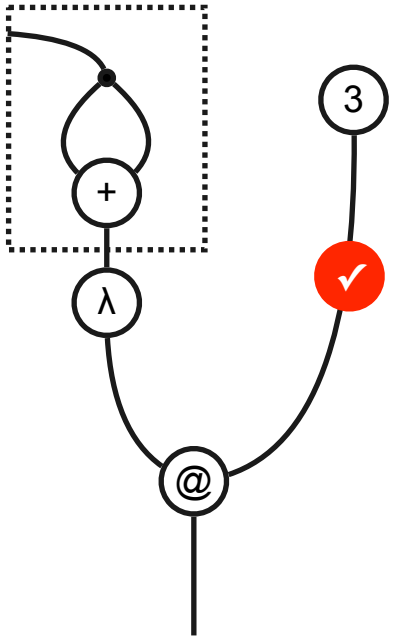


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

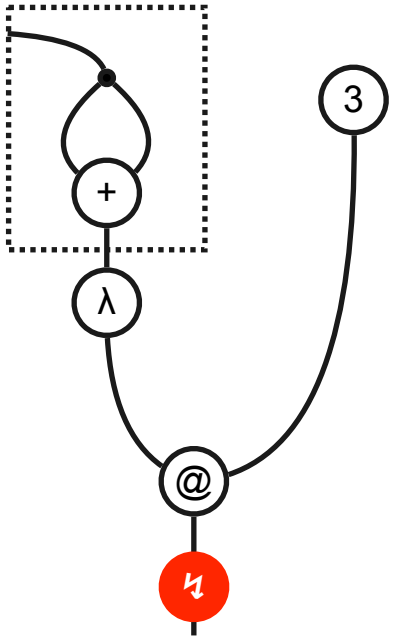


backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

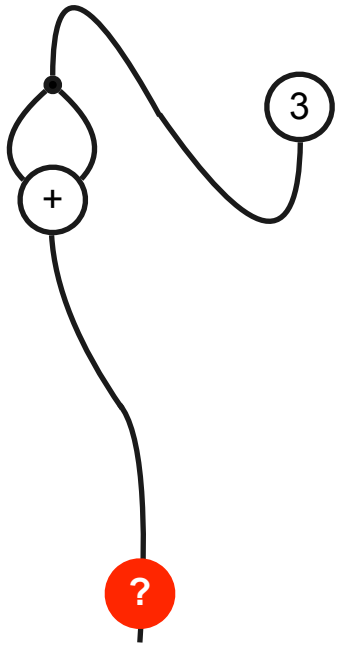


triggering update of hypernet

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

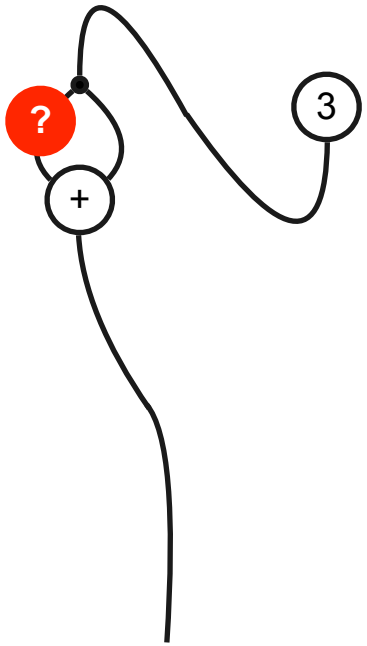


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

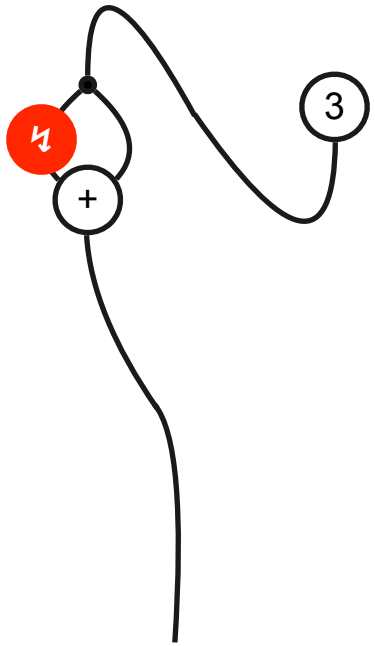


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

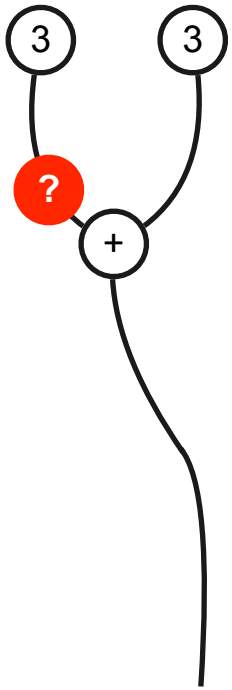


triggering update of hypernet

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

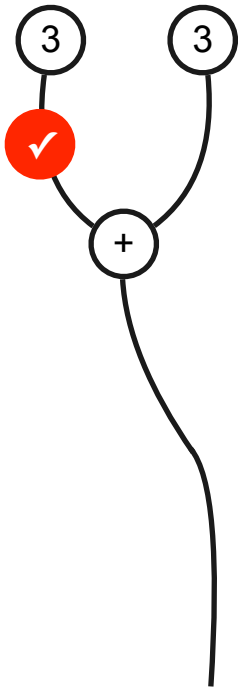


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

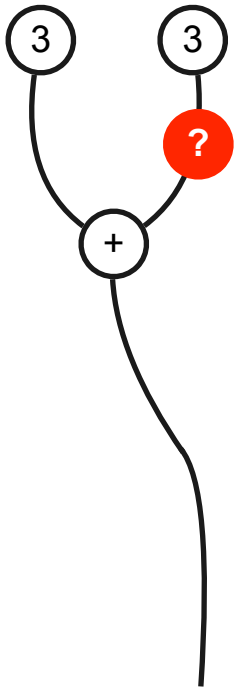


backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

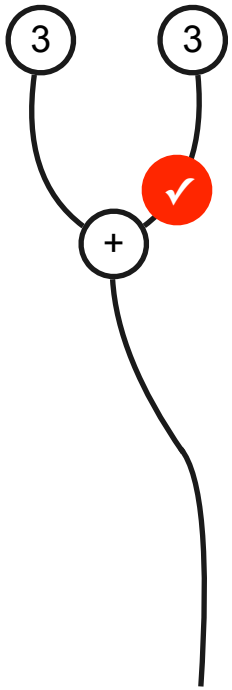


depth-first redex search

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

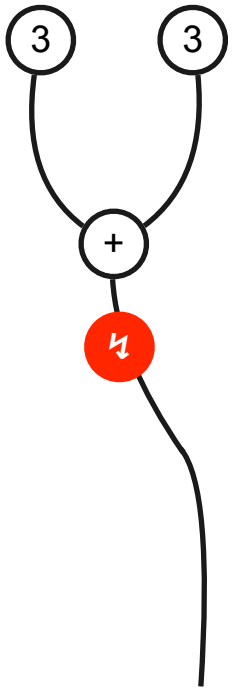


backtracking

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*



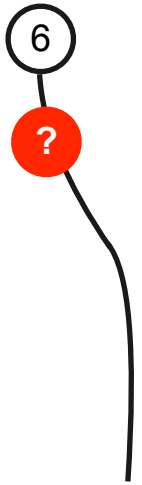
triggering update of hypernet

Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

depth-first redex search

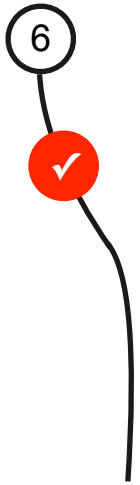


Program execution, graphically

Idea: updating hypernets step-by-step

… and strategically, using *focus*

backtracking



Hypernet semantics

- program execution by a *graphical* abstract machine
  - programs as

    certain hierarchical hypergraphs (*“hypernets”*)
  - execution as

    step-by-step strategical update of hypernets

Hypernet semantics

- program execution by a *graphical* abstract machine
  - programs as

    certain hierarchical hypergraphs (*"hypernets"*)
  - execution as

    step-by-step strategical update of hypernets
  - state = hypernet with focus 
  - transition = move of focus, or update of hypernet

Overview

1. Motivation: robustness of observational equivalence
2. Hypernet semantics
3. Locality & step-wise reasoning
4. Discussion: complication of simulation notion

Proof of observational equivalence, using *locality*

“Do two program fragments behave the same?”

Proof of observational equivalence, using *locality*

~~*"Do two program fragments behave the same?"*~~

"Do two sub-graphs behave the same in hypernet semantics?"

Proof of observational equivalence, using *locality*

~~*"Do two program fragments behave the same?"*~~

"Do two sub-graphs behave the same in hypernet semantics?"

★ Sub-graphs can represent parts of a program that are not necessarily well-formed,

e.g. parts relevant to a certain reference:

```
… new a = 1 in … (λx. a := 2; !a) … (λx. a := 2; !a) …
```

Proof of observational equivalence, using *locality*

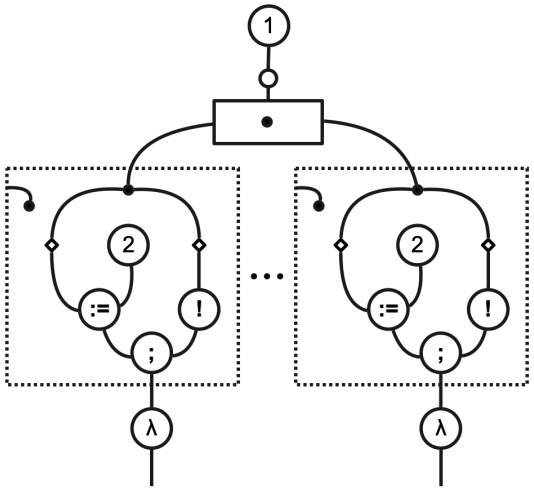
~~*"Do two program fragments behave the same?"*~~

"Do two sub-graphs behave the same in hypernet semantics?"

★ Sub-graphs can represent parts of a program that are not necessarily well-formed,

e.g. parts relevant to a certain reference:

```
… new a = 1 in … (λx. a := 2; !a) … (λx. a := 2; !a) …
```



Proof of observational equivalence, using *locality*

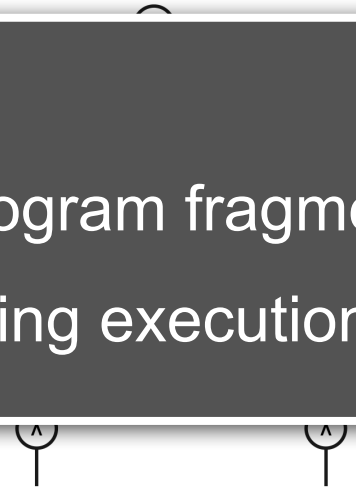
~~*"Do two program fragments behave the same?"*~~

"Do two sub-graphs behave the same in hypernet semantics?"

★ Sub-graphs can represent parts of a program that are not necessarily well-formed,
e.g. parts relevant to a certain reference:

```
… new a = 1 in … (λx. a := 2; !a) … (λx. a := 2; !a) …
```

Idea of *locality*:
analysing behaviour of program fragments,
by tracing sub-graphs during execution



Proof of observational equivalence, using *locality*

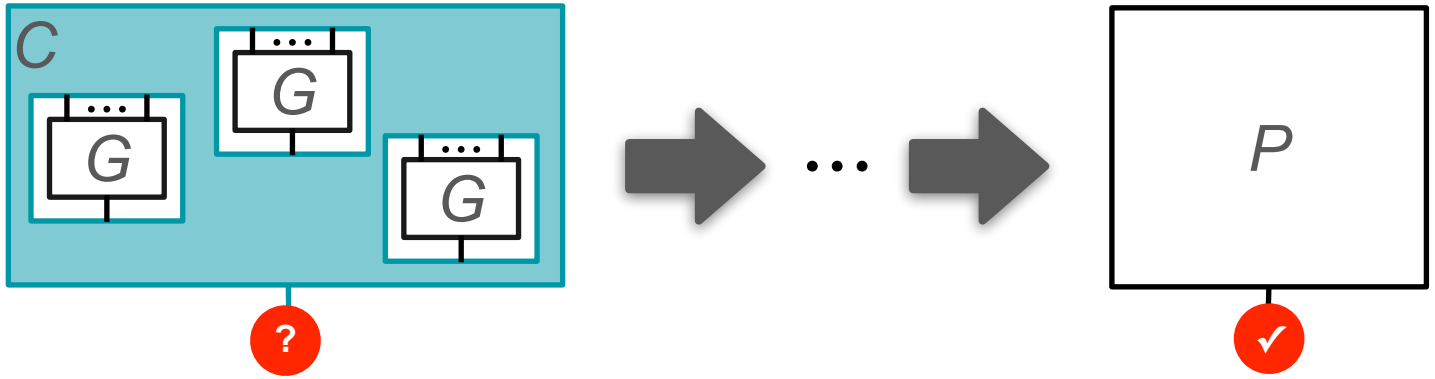
Claim: "Behaviour of a sub-graph G can be matched by behaviour of a sub-graph H."

Proof of observational equivalence, using *locality*

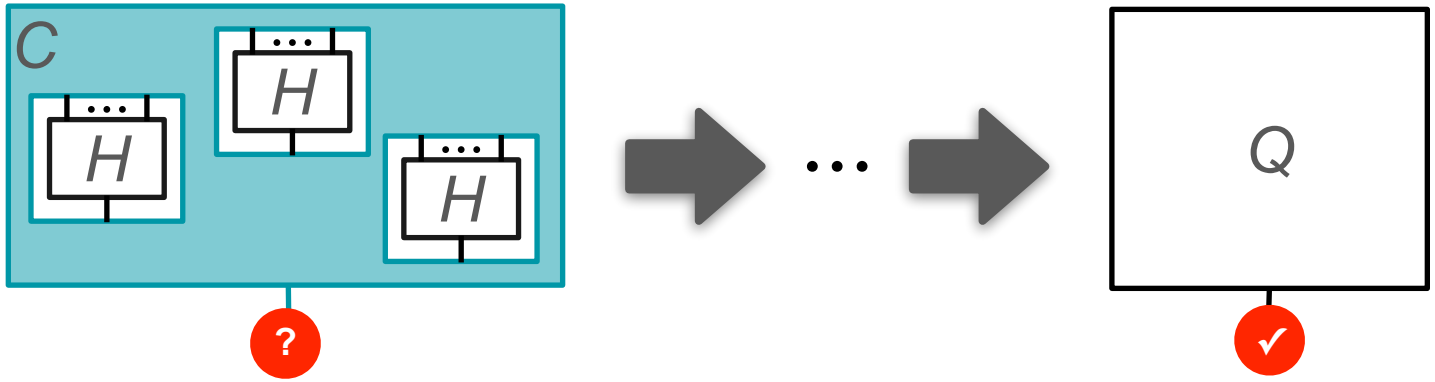
Claim: "Behaviour of a sub-graph G can be matched by behaviour of a sub-graph H."

For any context *C*,

if



then



Proof of observational equivalence, using *locality*

Claim: "Behaviour of a sub-graph G can be matched by behaviour of a sub-graph H."

Proof idea (simplified):

1. take **contextual closure** *R* of *(G,H)*

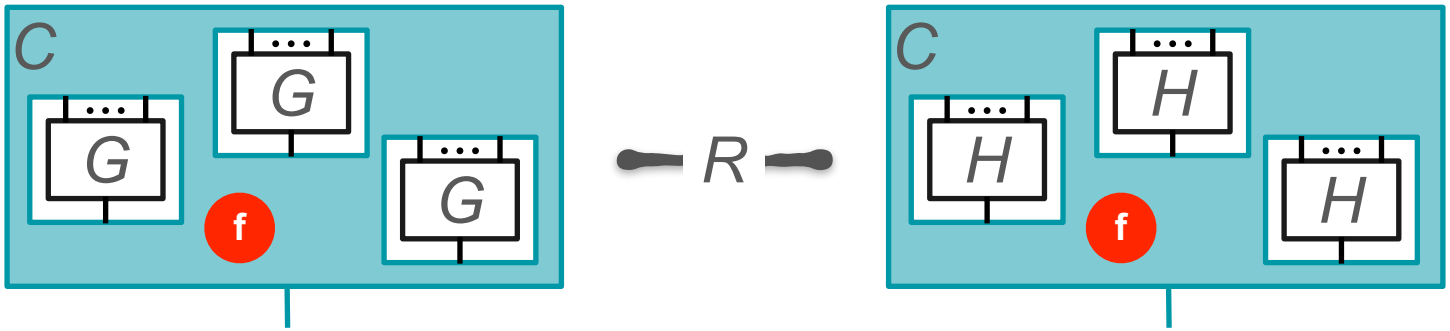
2. prove that the contextual closure *R* is a **simulation**

Proof of observational equivalence, using *locality*

Claim: "Behaviour of a sub-graph G can be matched by behaviour of a sub-graph H."

Proof idea (simplified):

1. take **contextual closure** *R* of *(G,H)*



for any context *C* with focus

2. prove that the contextual closure *R* is a **simulation**

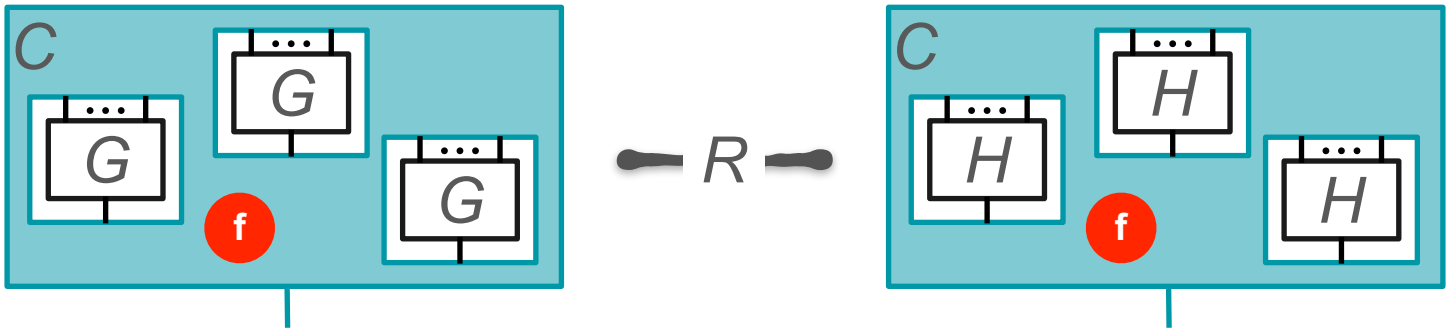
Proof of observational equivalence, using *locality*

Claim: "Behaviour of a sub-graph G can be matched by behaviour of a sub-graph H."

Proof idea (simplified):

R is closed under contexts, by definition

1. take **contextual closure** *R* of *(G,H)*



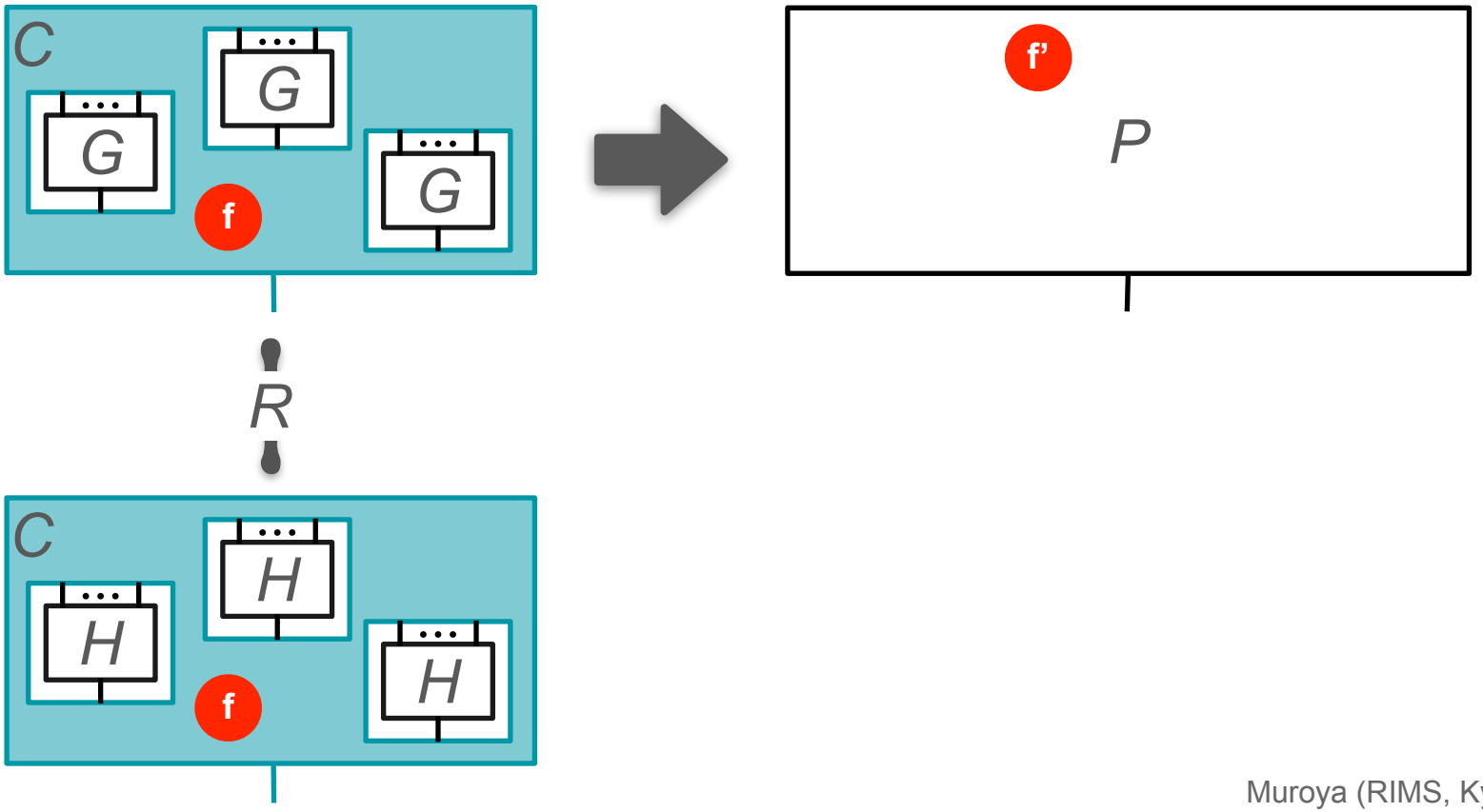
for any context *C* with focus

2. prove that the contextual closure *R* is a **simulation**

Proof of observational equivalence, using *locality*

Proof idea (simplified):

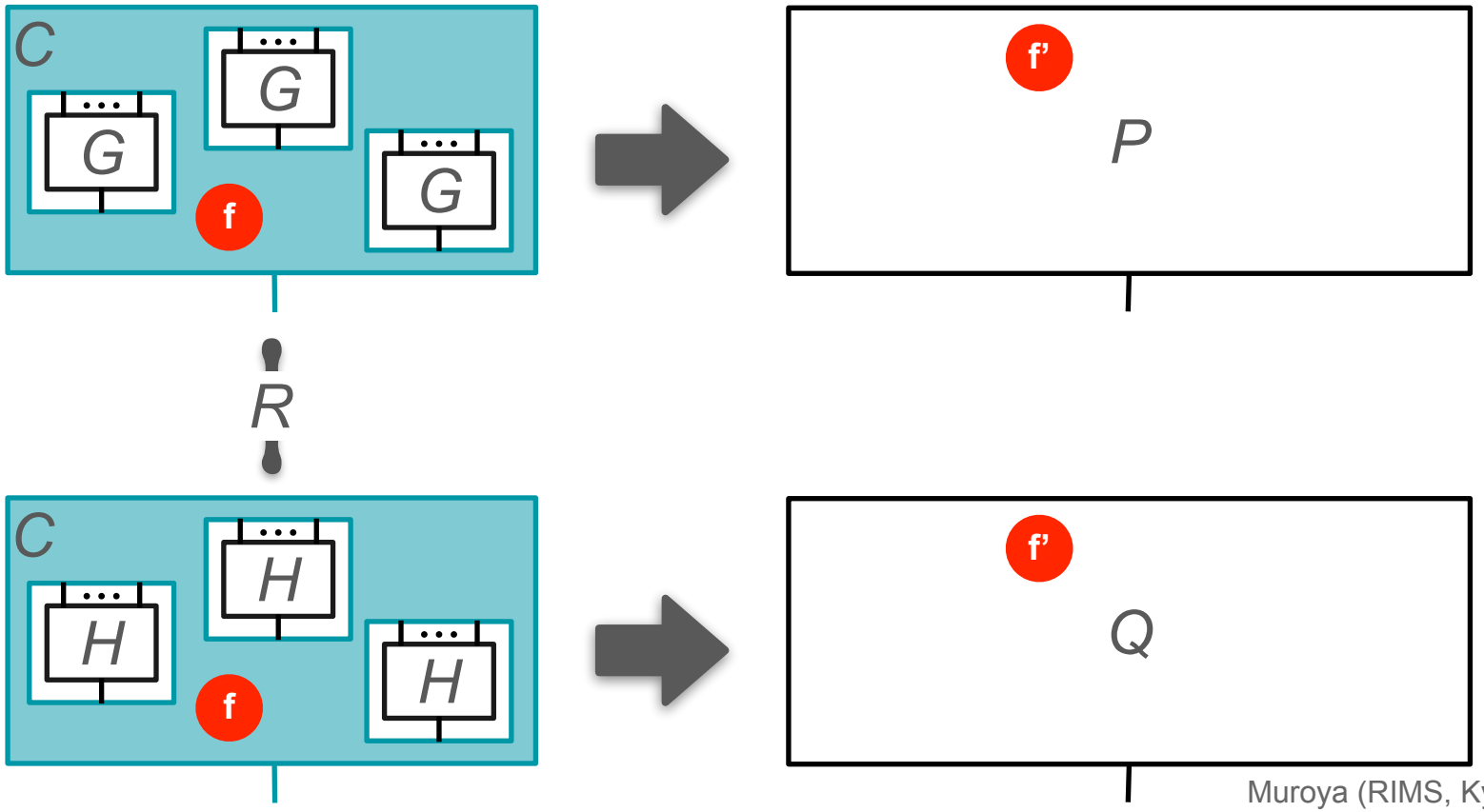
2. prove that the contextual closure R is a **simulation**



Proof of observational equivalence, using *locality*

Proof idea (simplified):

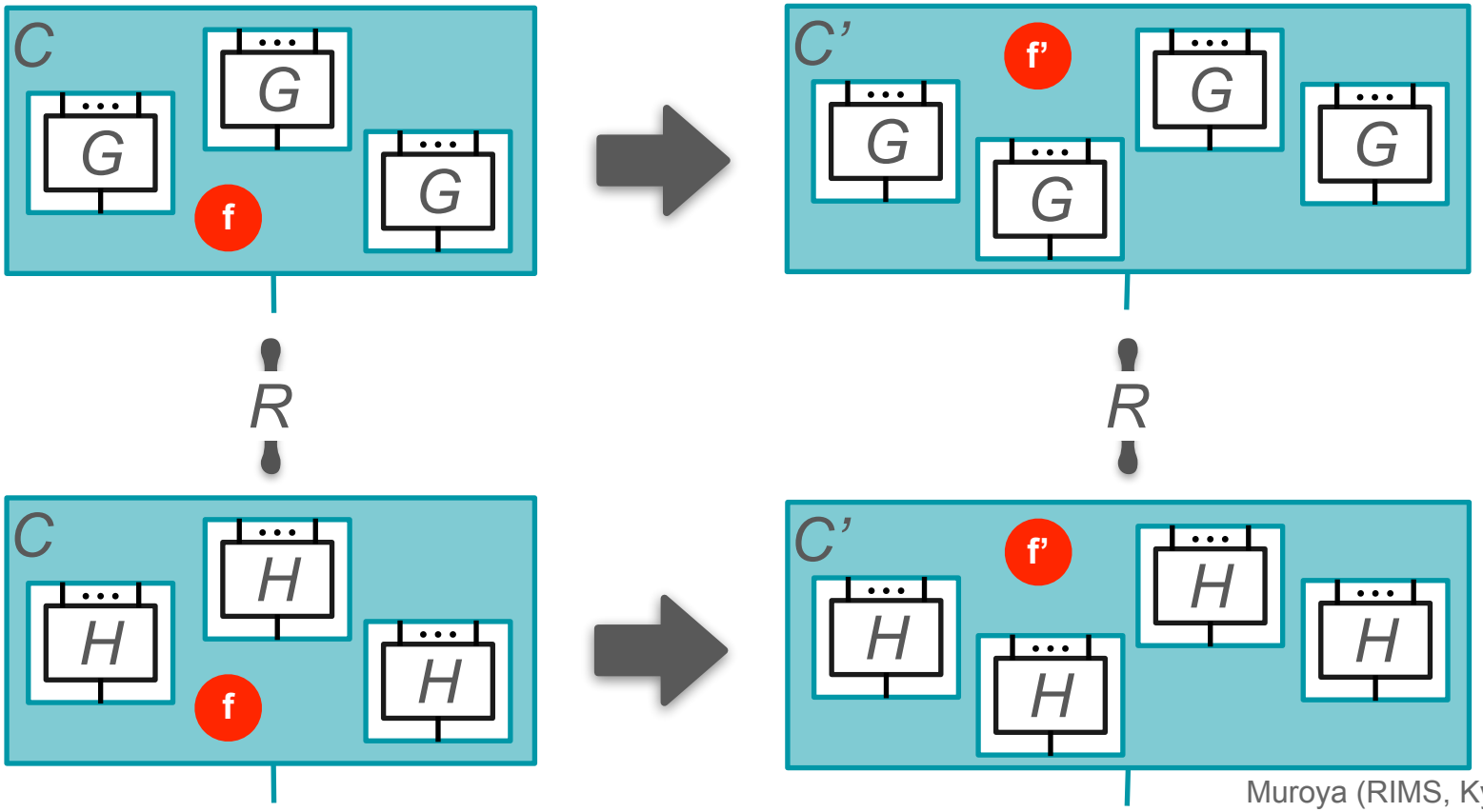
2. prove that the contextual closure R is a **simulation**



Proof of observational equivalence, using *locality*

Proof idea (simplified):

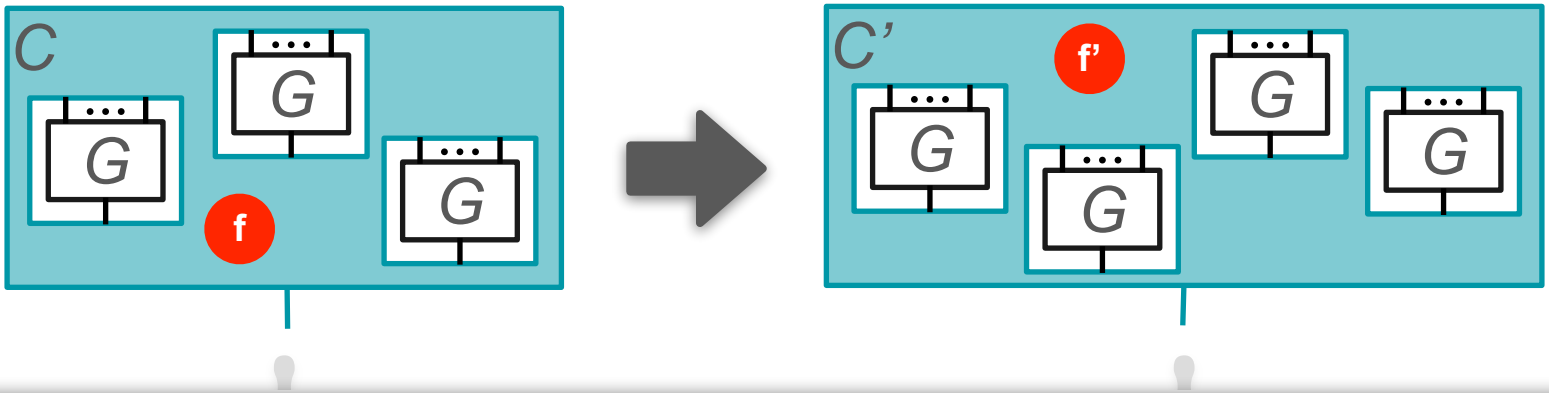
2. prove that the contextual closure *R* is a **simulation**



Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure *R* is a **simulation**



Idea of *locality*:

tracing sub-graphs during transition,

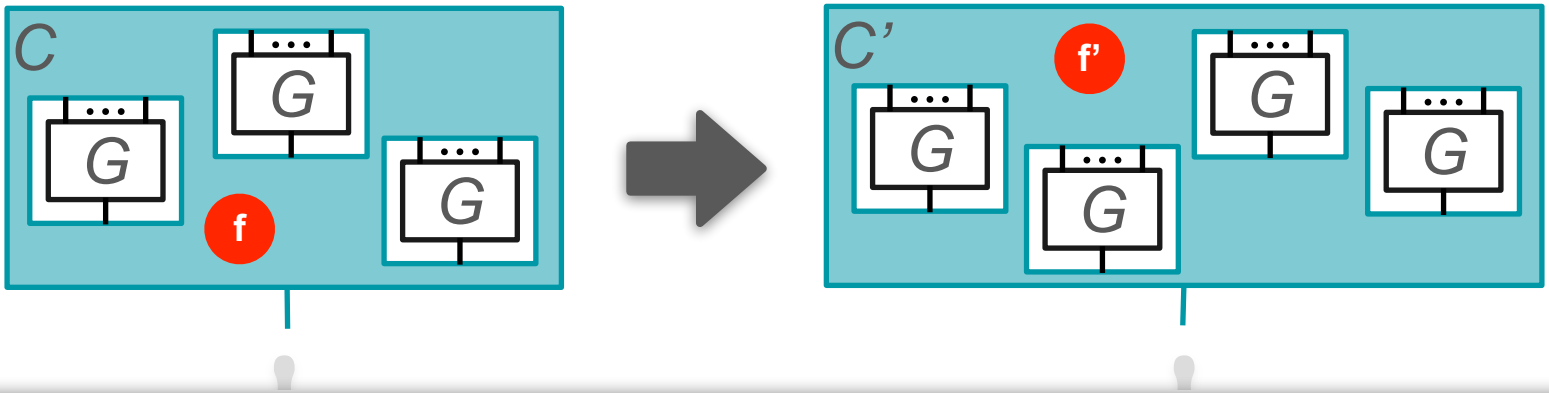
by analysing what happens around the focus during transition

Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure *R* is a **simulation**

   … by case analysis of transition



Idea of *locality*:

tracing sub-graphs during transition,

by analysing what happens around the focus during transition

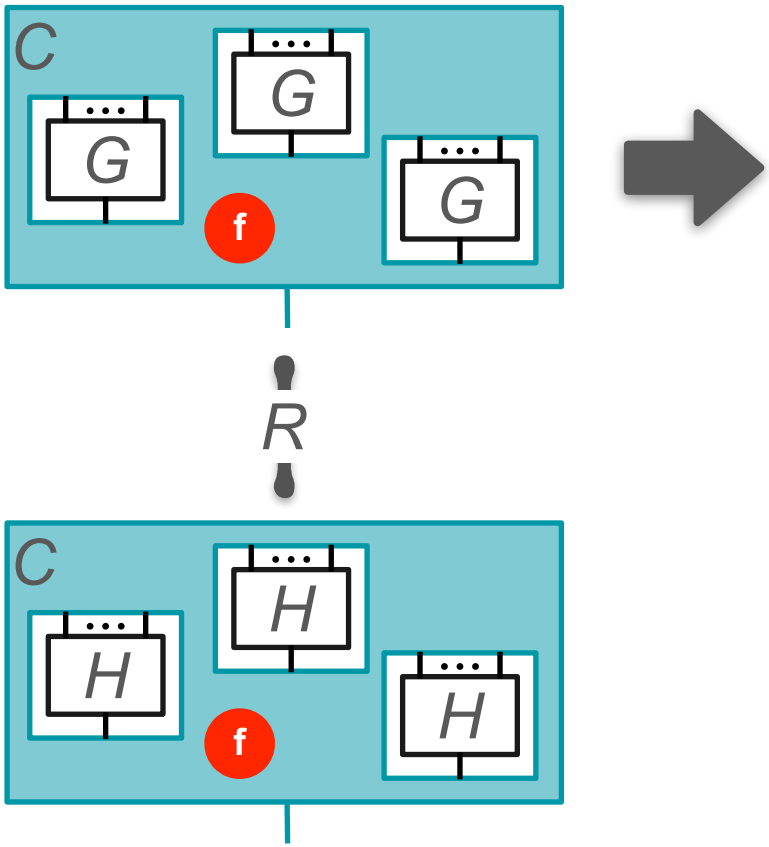
move, or trigger update

Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure R is a **simulation**

Case (1) move of focus ? or ✓ inside context

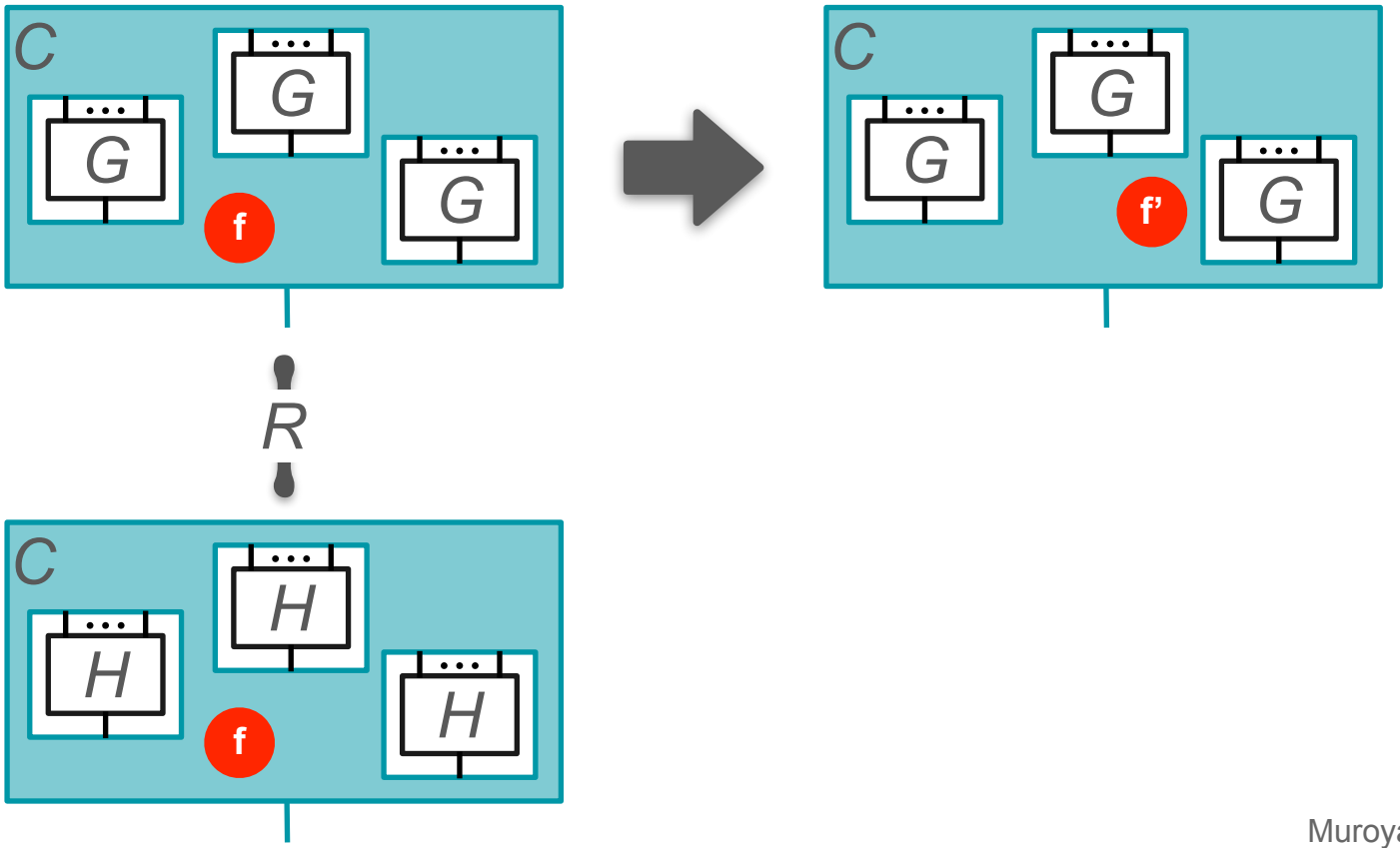


Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure *R* is a **simulation**

Case (1) move of focus ? or ✓ inside context

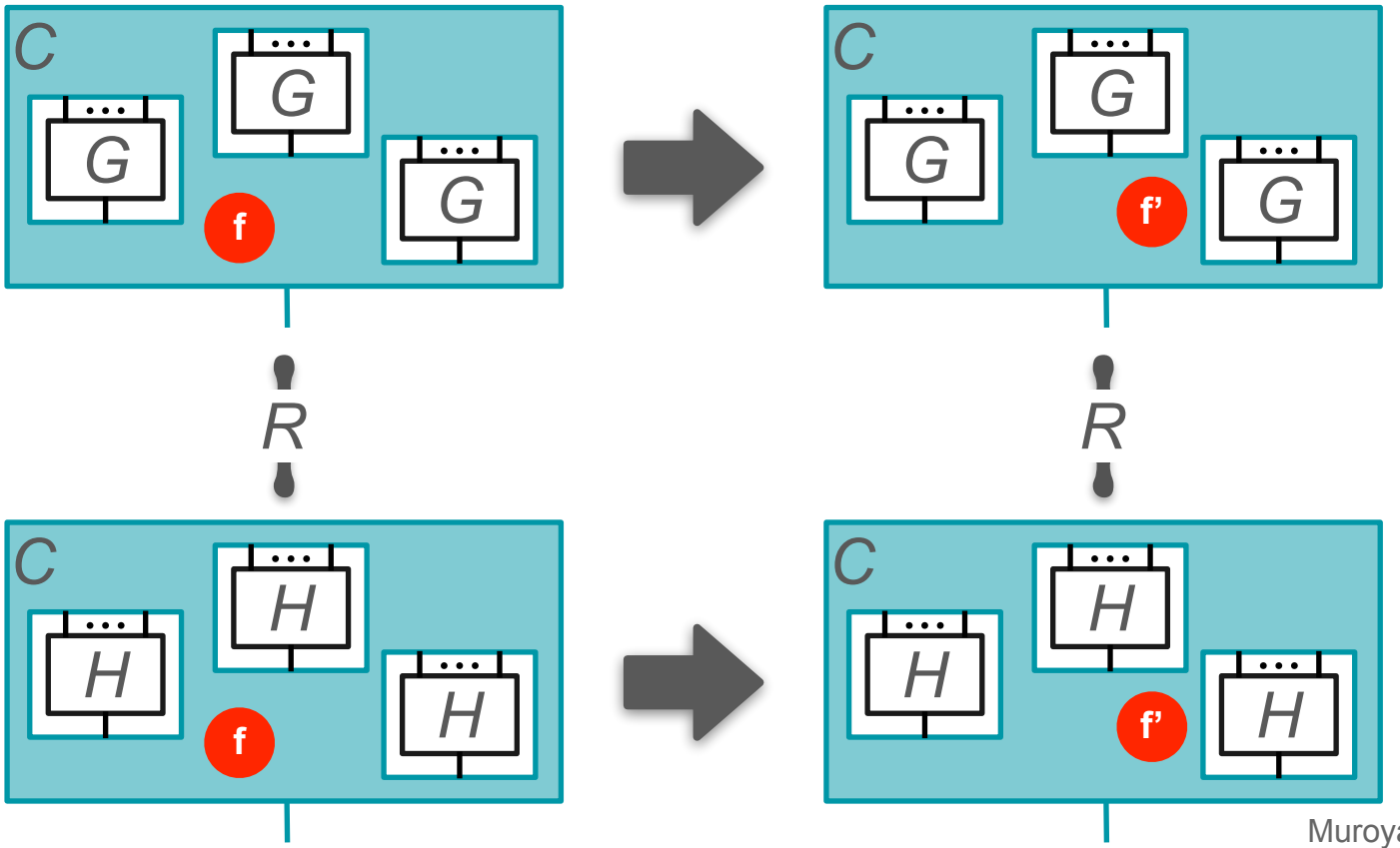


Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure *R* is a **simulation**

Case (1) move of focus ? or ✓ inside context

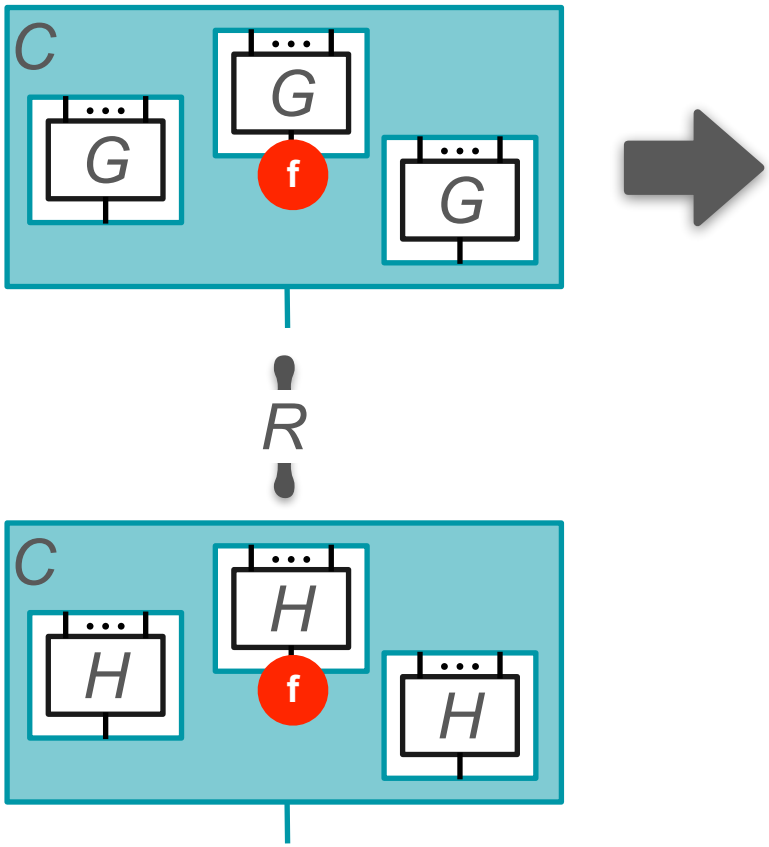


Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure *R* is a **simulation**

Case (2) move of focus ? or ✓ , entering *G*

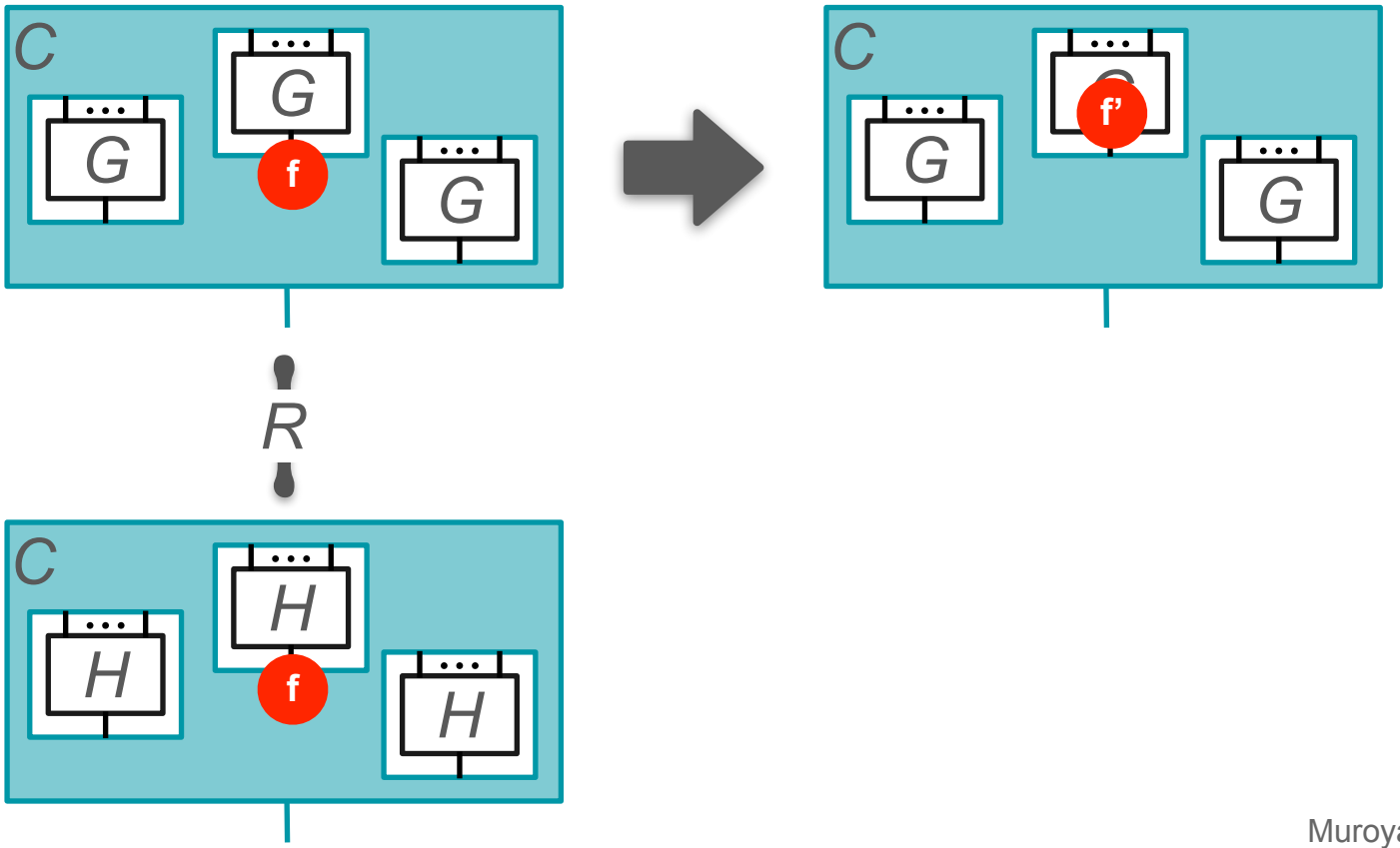


Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure *R* is a **simulation**

Case (2) move of focus ? or ✓ , entering *G*

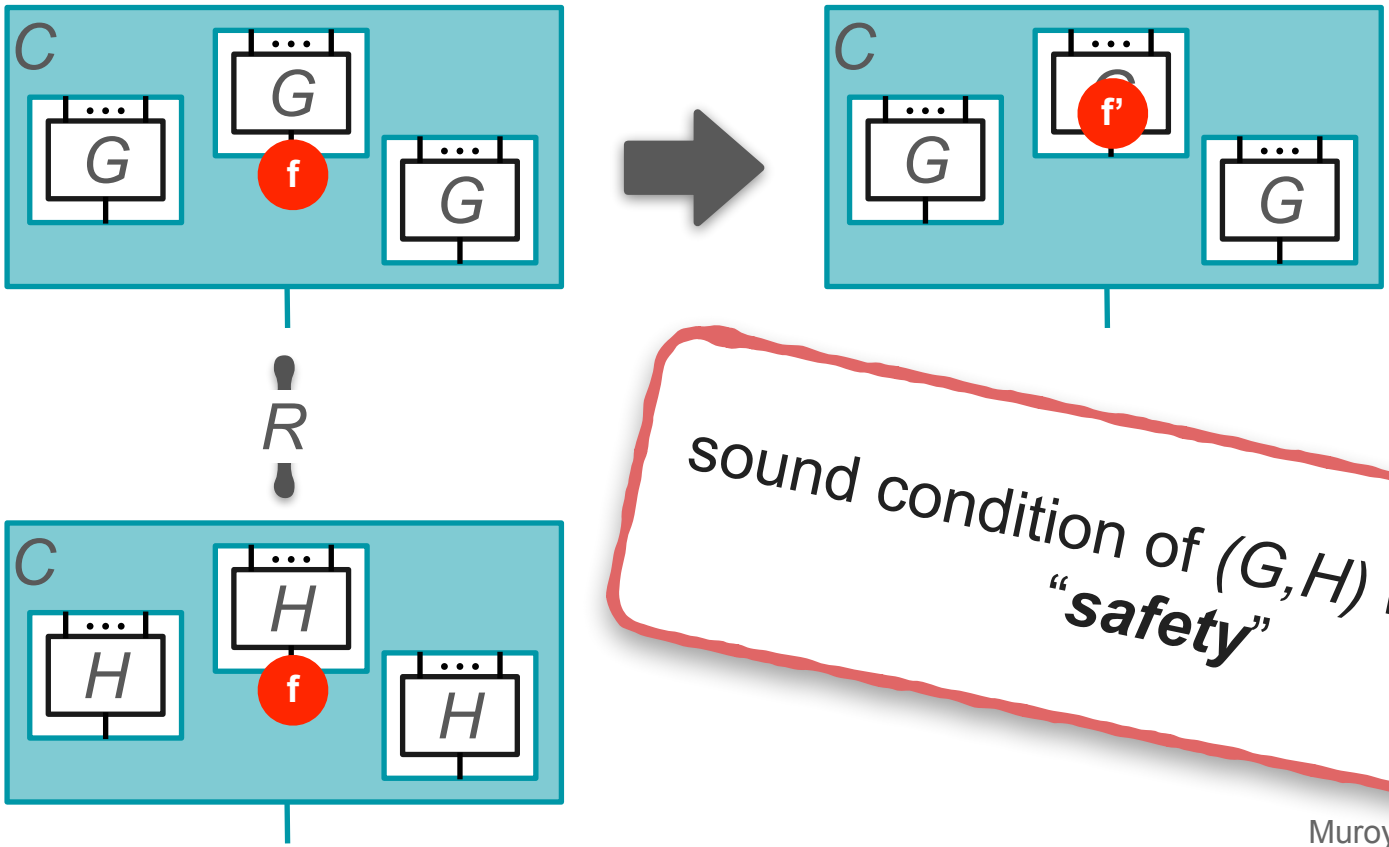


Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure *R* is a **simulation**

Case (2) move of focus ? or ✓ , entering *G*



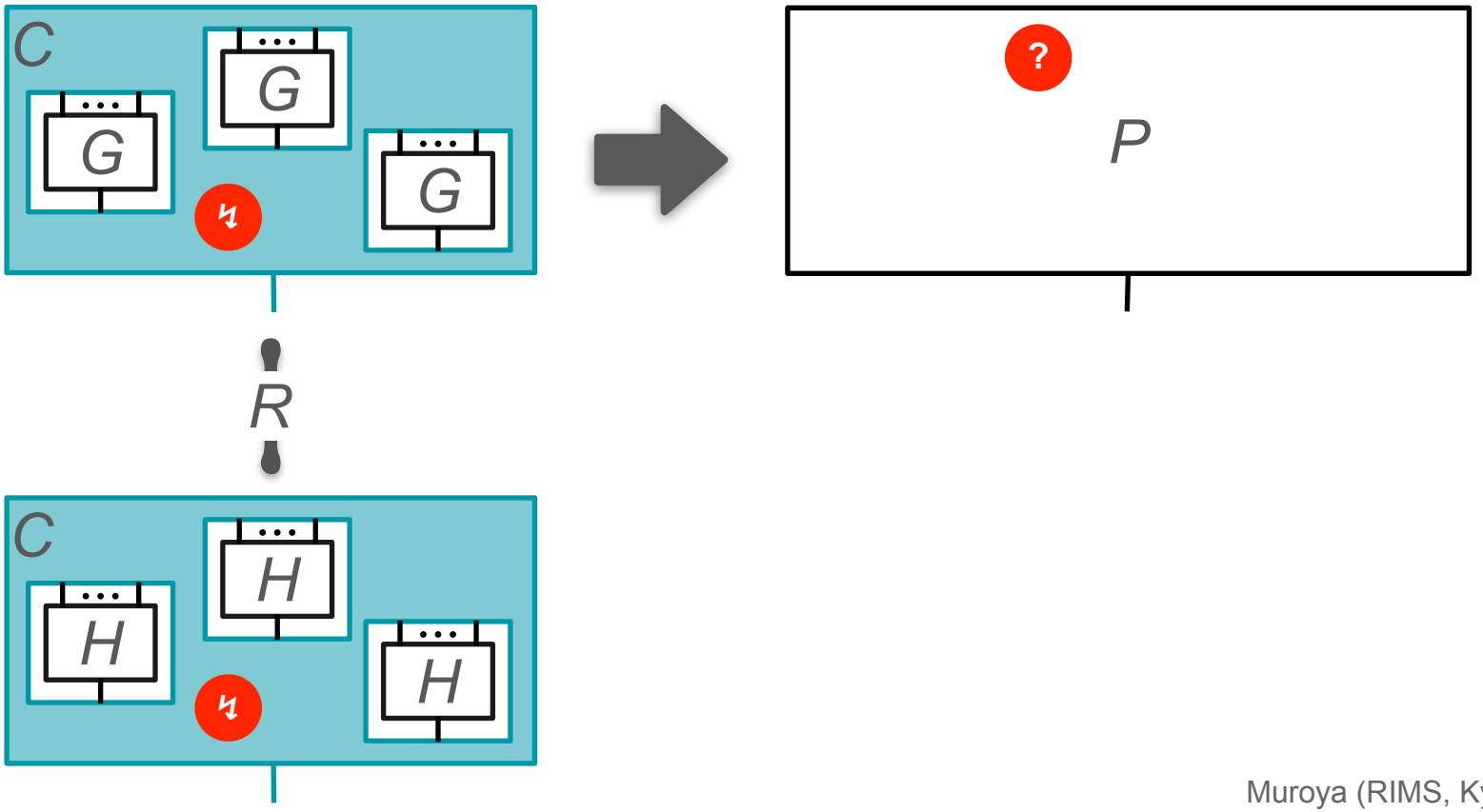
sound condition of (G,H) identified:
"**safety**"

Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure R is a **simulation**

Case (3) update of hypernet

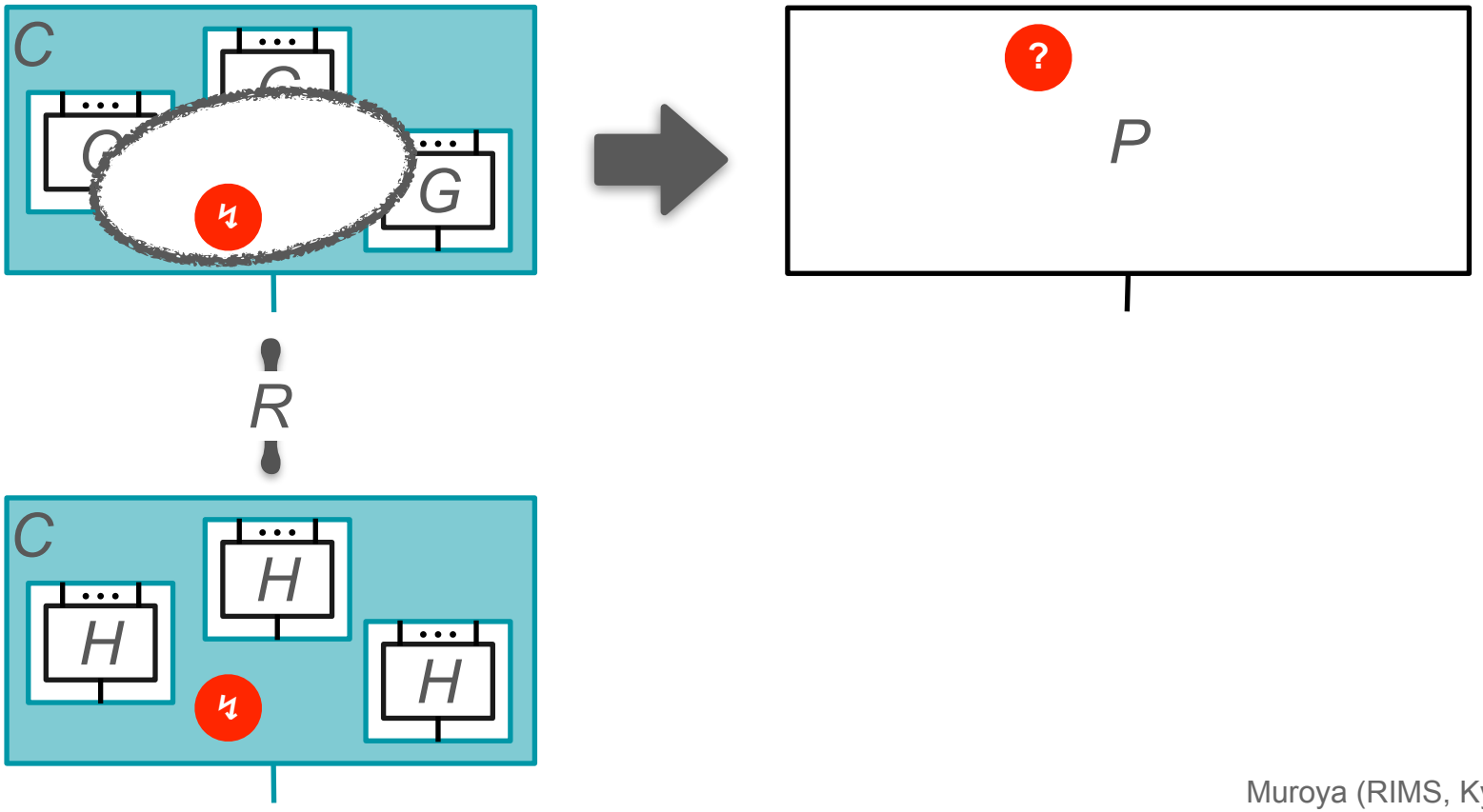


Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure R is a **simulation**

Case (3) update of hypernet

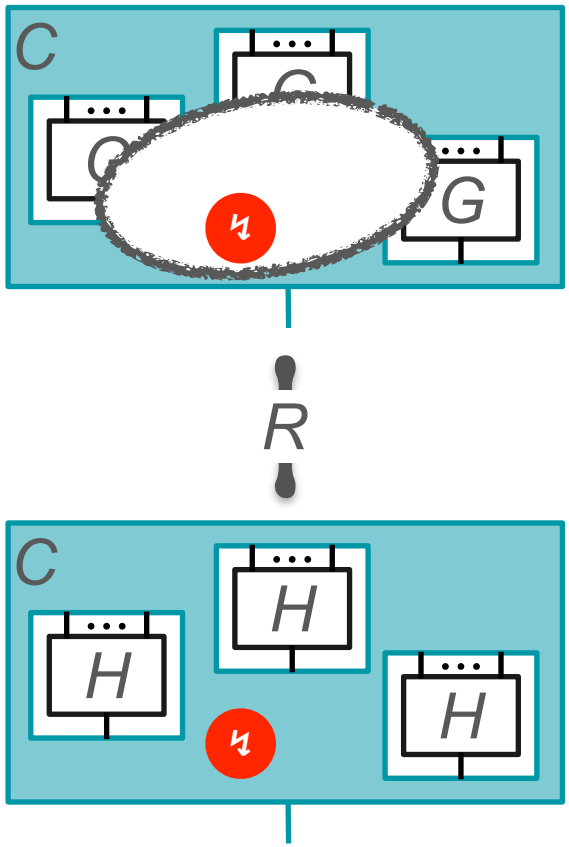


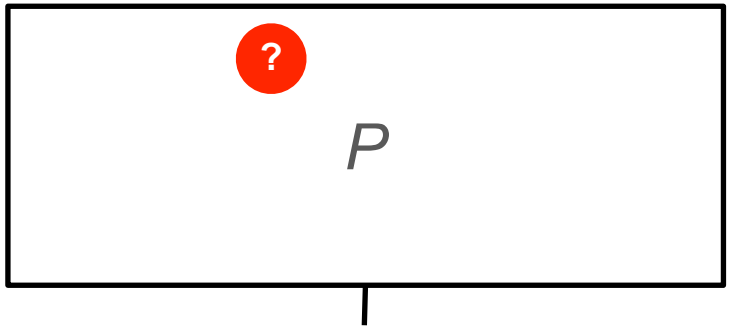
Proof of observational equivalence, using *locality*

Proof idea (simplified):

2. prove that the contextual closure *R* is a **simulation**

Case (3) update of hypernet





sound condition of *(G,H)* identified:
"***robustness***"
relative to all possible rewrites

Proof of observational equivalence, using *locality*

Claim: "Behaviour of a sub-graph G can be matched by behaviour of a sub-graph H."

Proof idea (simplified):

1. take **contextual closure** *R* of *(G,H)*
2. prove that the contextual closure *R* is a **simulation**

   by case analysis

Proof of observational equivalence, using *locality*

Claim: "Behaviour of a sub-graph G can be matched by behaviour of a sub-graph H."

Proof idea (simplified):

1. take **contextual closure** *R* of *(G,H)*
2. prove that the contextual closure *R* is a **simulation**

   by case analysis

Characterisation Theorem

***Robust** and **safe** template induce observational equivalences.*

(for deterministic & "reasonable" languages)

Overview

1. Motivation: robustness of observational equivalence
2. Hypernet semantics
3. Locality & step-wise reasoning
4. Discussion: complication of simulation notion

Complication of simulation notion

Proof idea (simplified):

1. take **contextual closure** *R* of *(G,H)*

2. prove that the contextual closure *R* is a **simulation**

   by case analysis

Characterisation Theorem
Robust *and* ***safe*** *template induce observational equivalences.*
(for deterministic & “reasonable” languages)

Complication of simulation notion

Proof idea (simplified):

1. take **contextual closure** *R* of *(G,H)*
2. prove that the contextual closure *R* is a **simulation**

   by case analysis

Characterisation Theorem

Robust *and* ***safe*** *template induce observational equivalences.*

(for deterministic & “reasonable” languages)

Observation:

ordinary simulations do not always suffice…

Complication of simulation notion

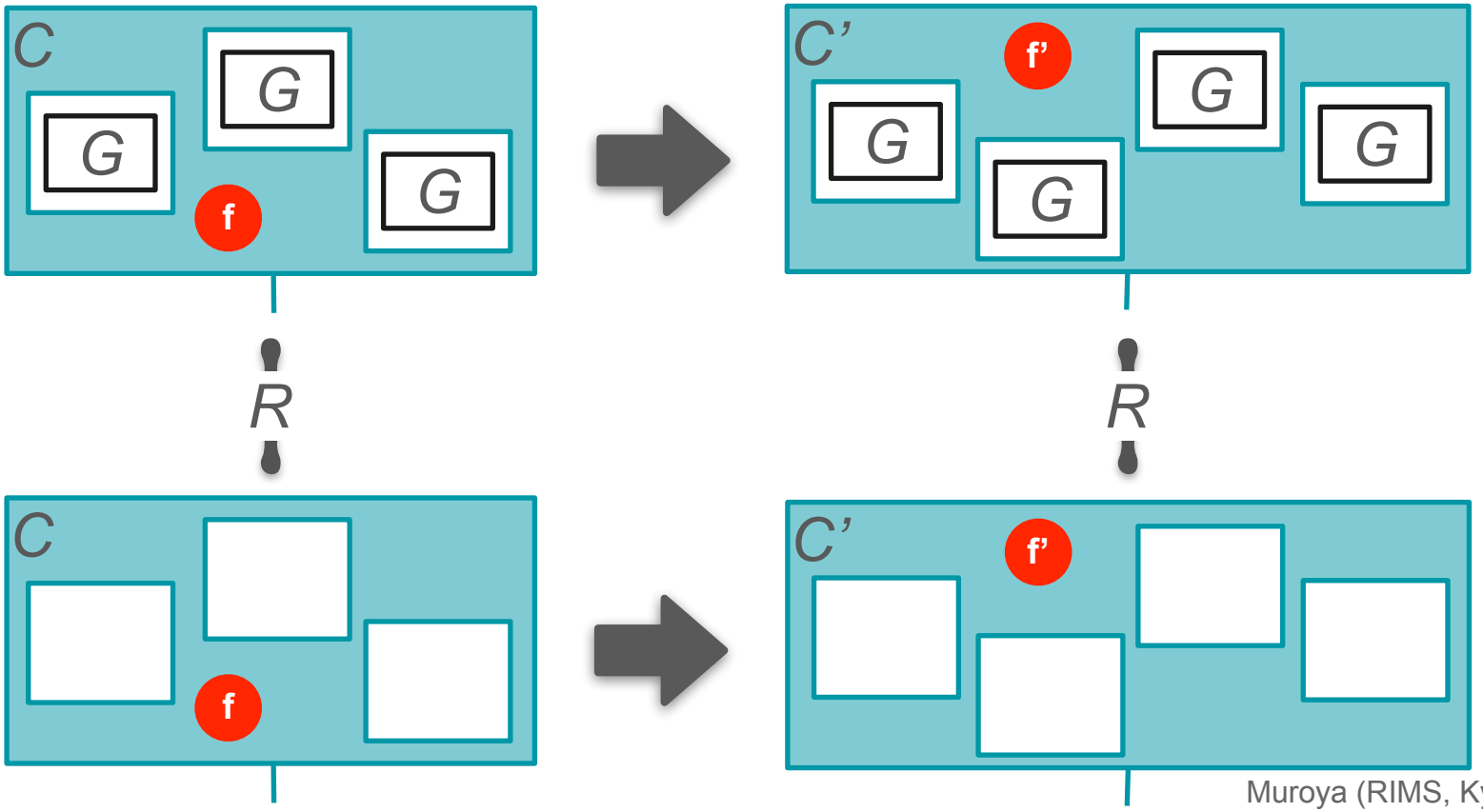
2. prove that the contextual closure R is a **simulation**

   - The standard simulation suffices for GC:

Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

- The standard simulation suffices for GC:



Complication of simulation notion

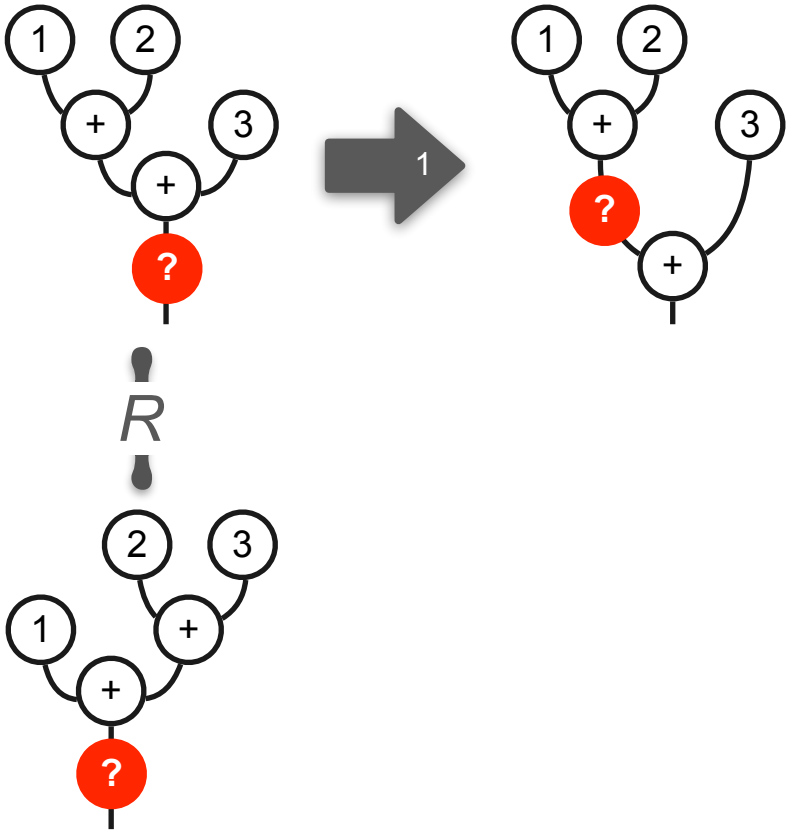
2. prove that the contextual closure *R* is a **simulation**

   - A *weak* simulation is desired for some arithmetic laws:

Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

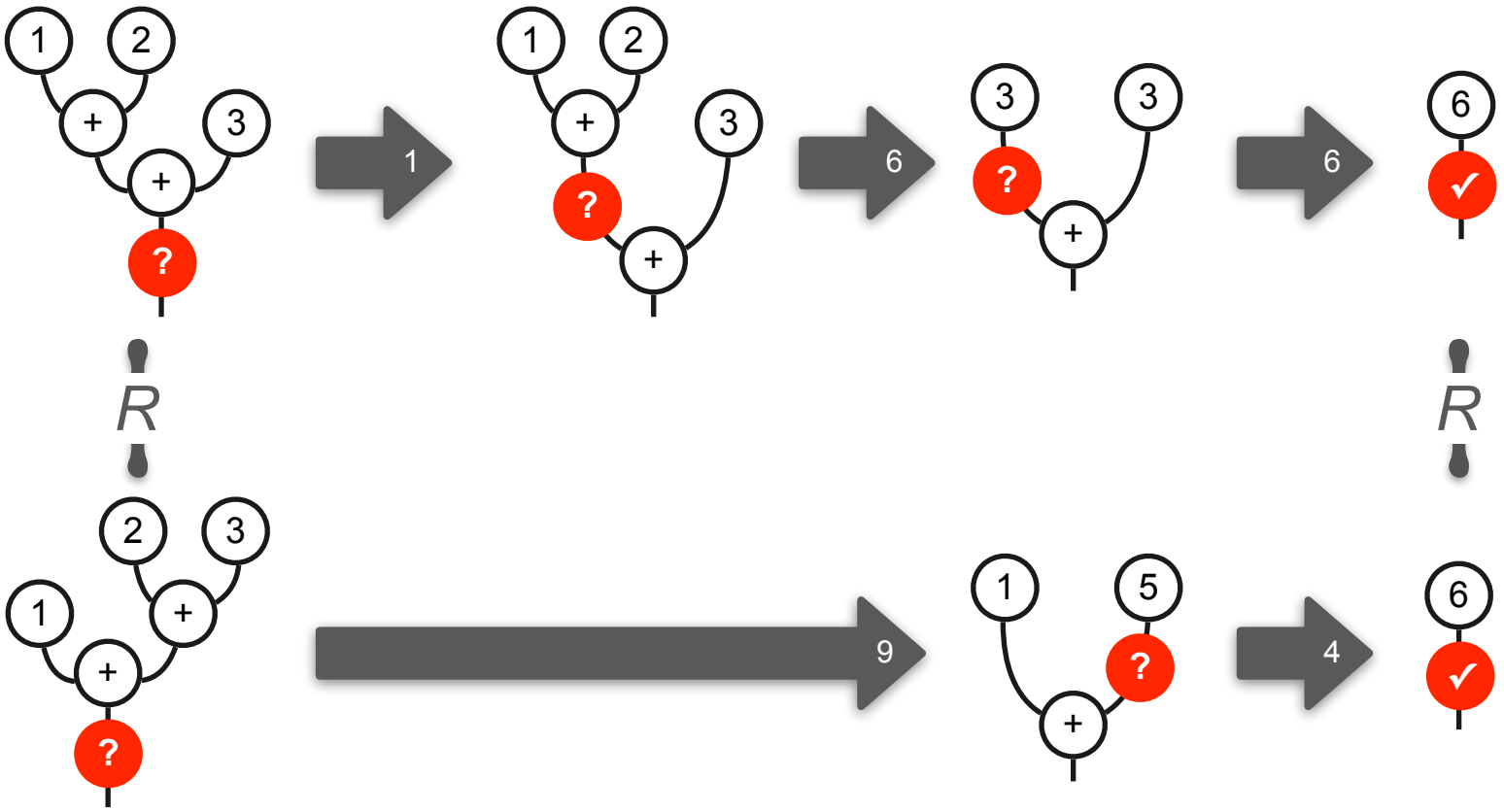
- A *weak* simulation is desired for some arithmetic laws:



Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

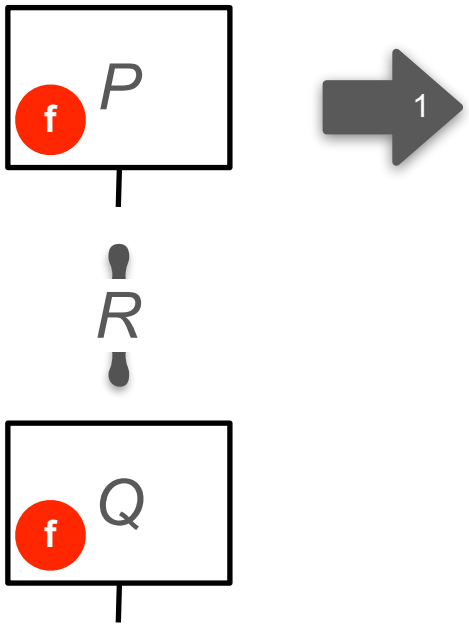
- A *weak* simulation is desired for some arithmetic laws:



Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

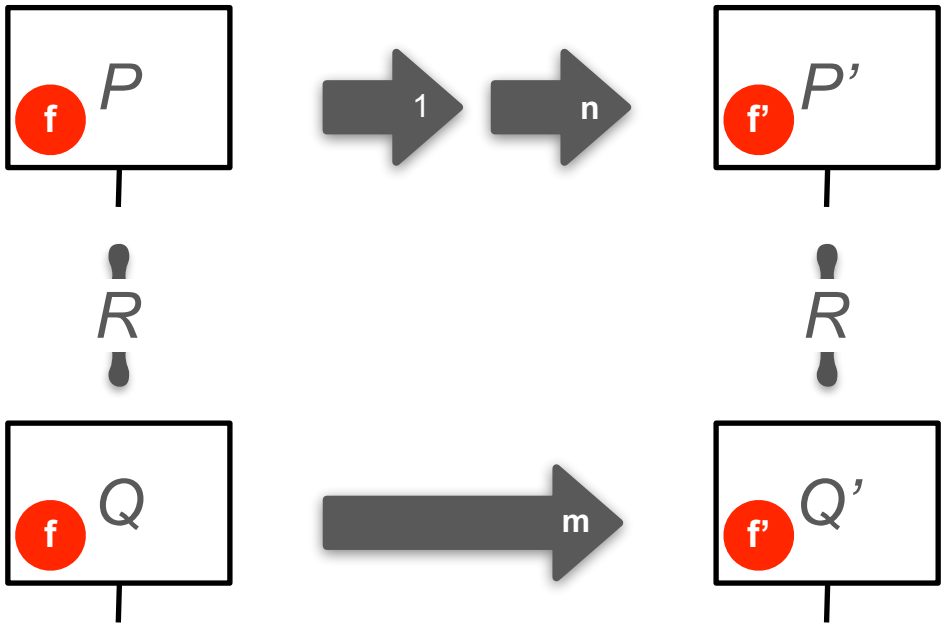
- A *weak* simulation is desired:



Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

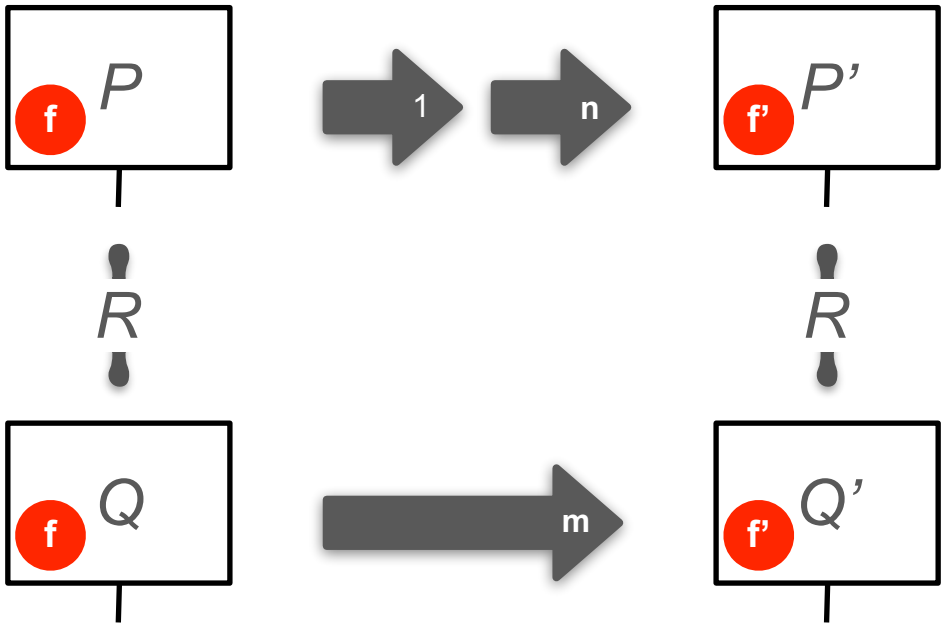
- A *weak* simulation is desired:



Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

- A *weak* simulation is desired:

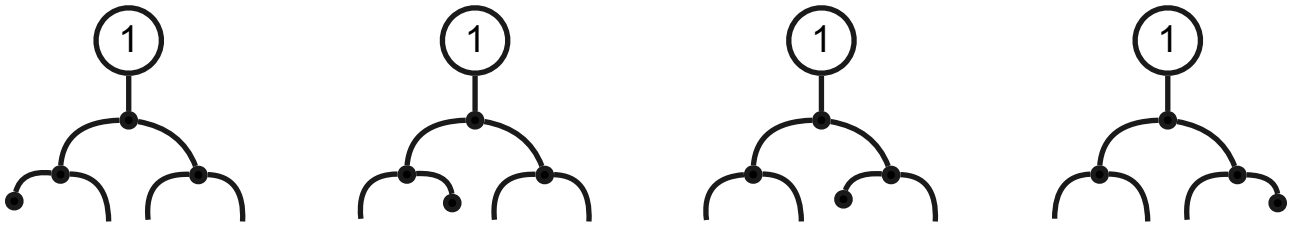


- Soundness fails, in the presence of nondeterminism.

Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

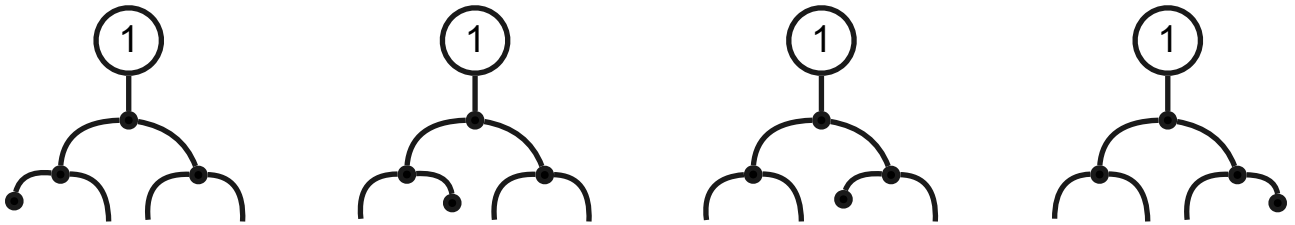
- A weak simulation *up to observational equivalence* is desired, to identify different ways of sharing:



Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

- A weak simulation *up to observational equivalence* is desired, to identify different ways of sharing:



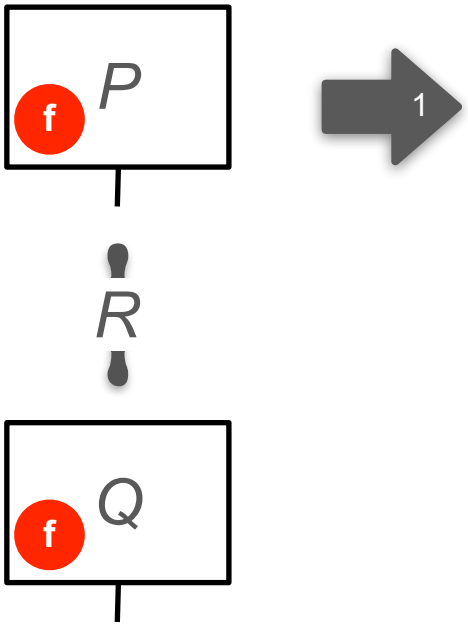
✗ working on graphs modulo structural equivalence

✓ using up-to technique with structural equivalence

Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

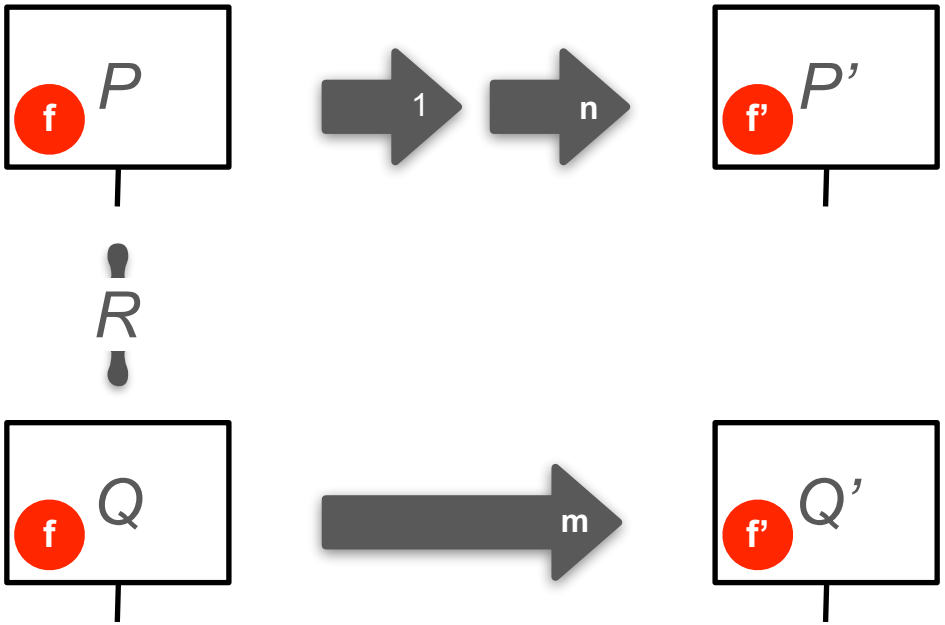
- A weak simulation *up to observational equivalence* is desired:



Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

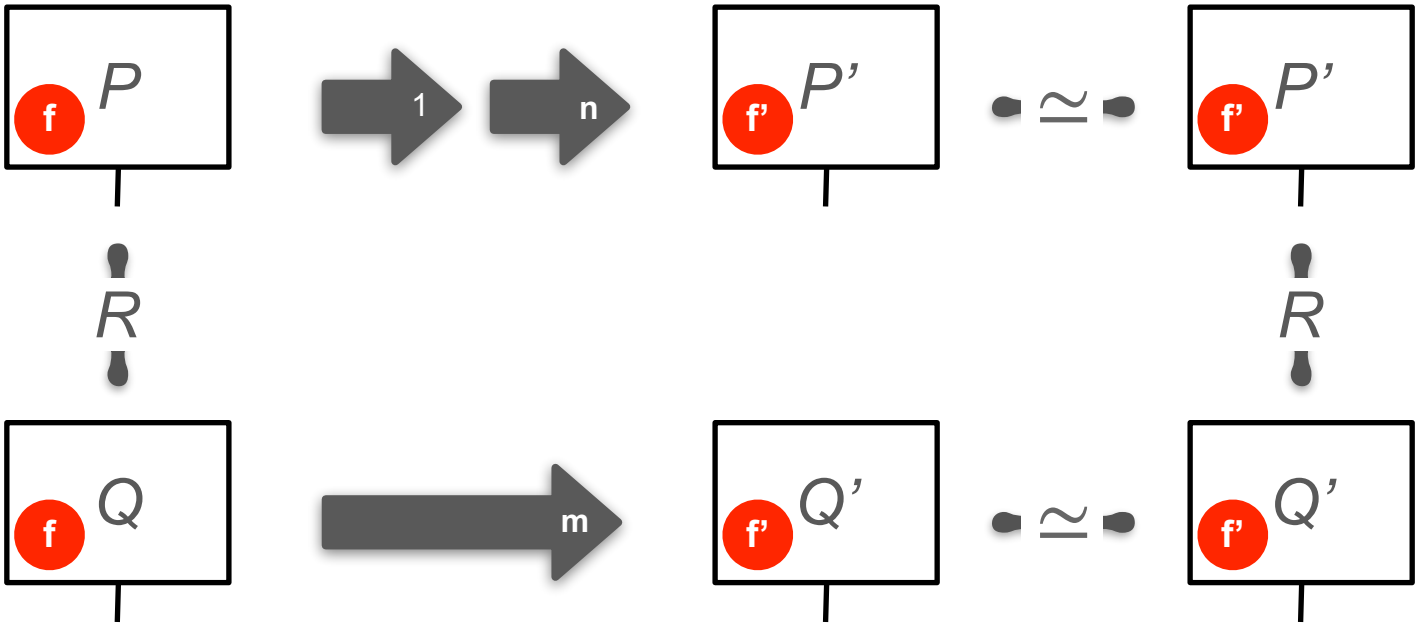
- A weak simulation *up to observational equivalence* is desired:



Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

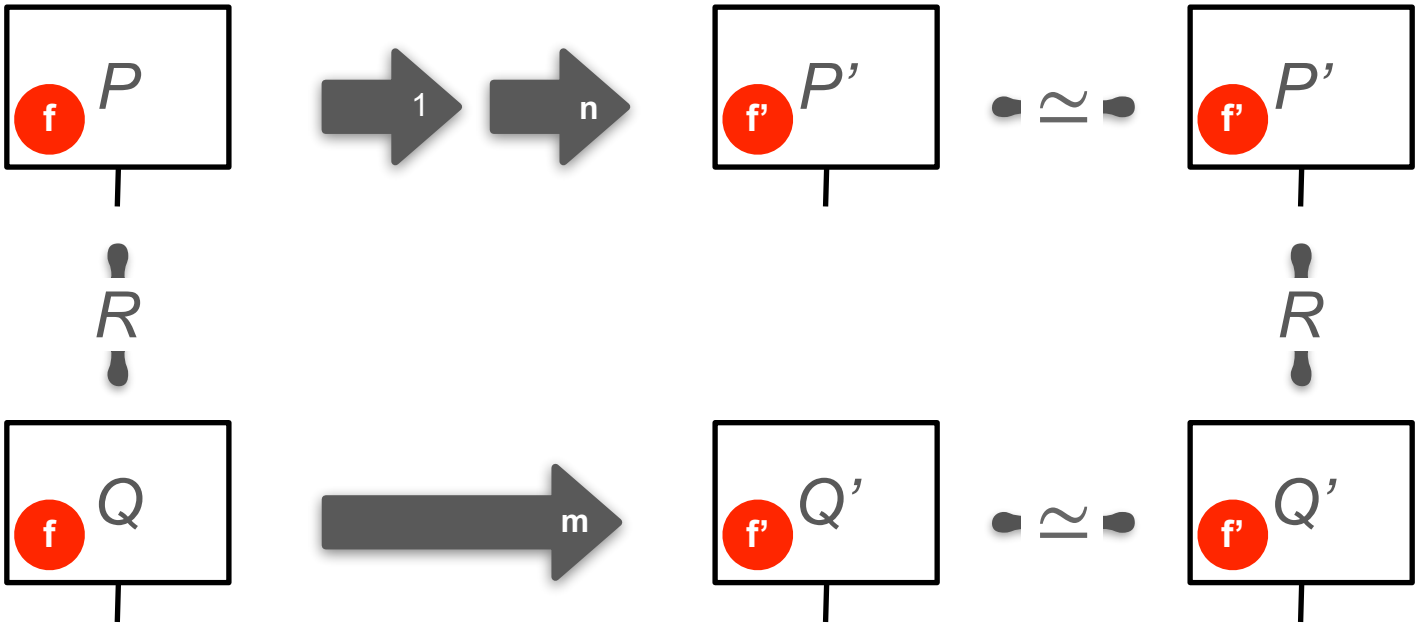
- A weak simulation *up to observational equivalence* is desired:



Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

- A weak simulation *up to observational equivalence* is desired:

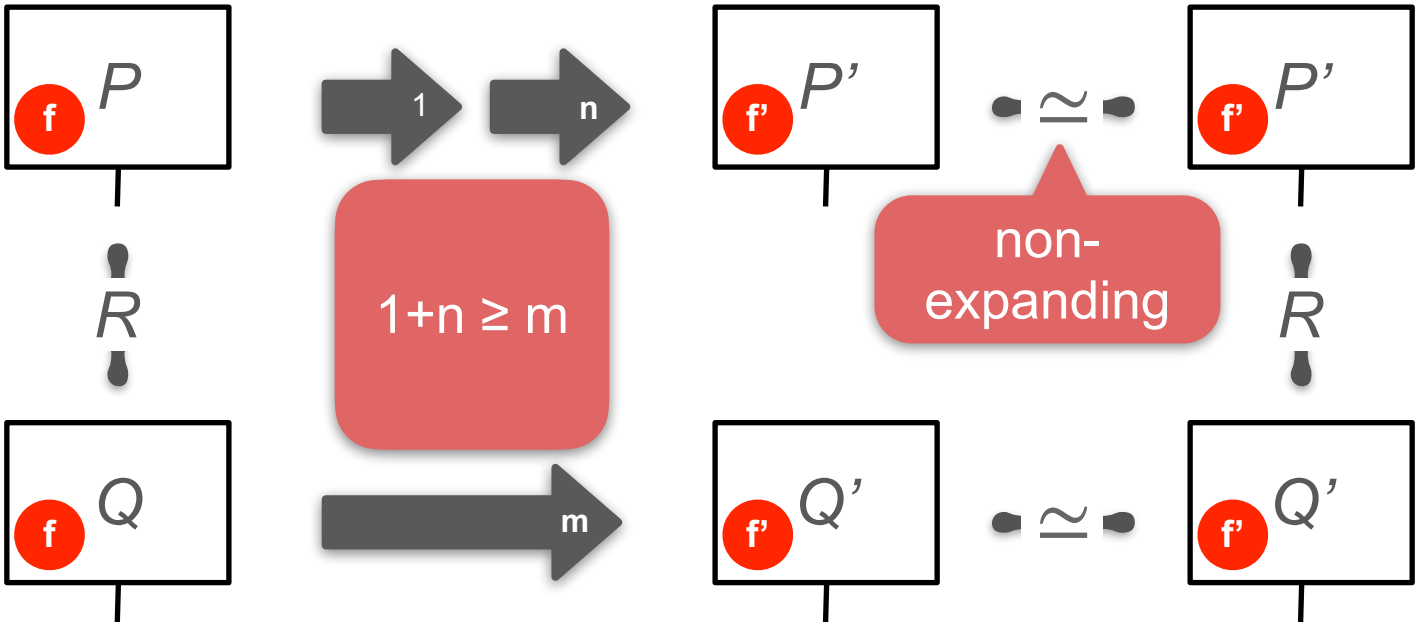


- Soundness fails, without some quantitative restrictions.

Complication of simulation notion

2. prove that the contextual closure *R* is a **simulation**

- A weak simulation *up to observational equivalence* is desired:



- Soundness fails, without some quantitative restrictions.

Overview

1. Motivation: robustness of observational equivalence
2. Hypernet semantics
3. Locality & step-wise reasoning
4. Discussion: complication of simulation notion

Conclusion

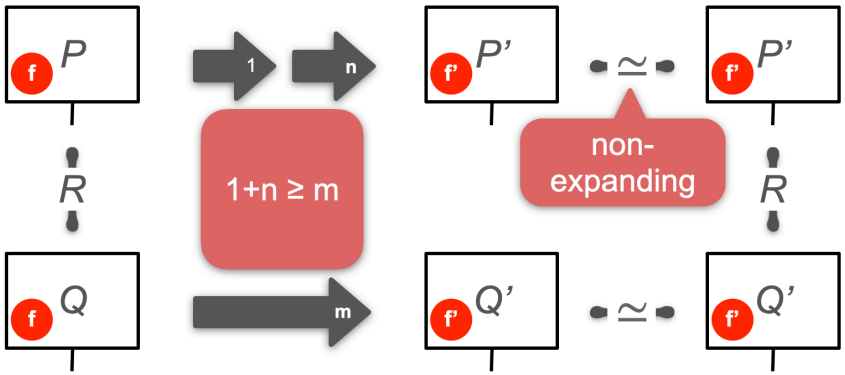
- a (general) framework for analysing and proving robustness of observational equivalence

  - hypernet semantics: a *graphical* abstract machine
  - *local & step-wise* reasoning to prove observational equivalence, with the concept of *robustness*

- current key limitation: determinism

Future directions

- complication of simulation notion



  - What causes complication of simulation notion?
  - How can this complication be justified?
  - Are there relevant simulation notions?
  - How can we deal with nondeterminism?

Future directions

- Sand's improvement theory

  (incorporating cost reduction in observational equivalence)

  - The number of steps can already be dealt with,

    by the quantitative restrictions on the weak up-to simulation.