Local reasoning for robust observational equivalence

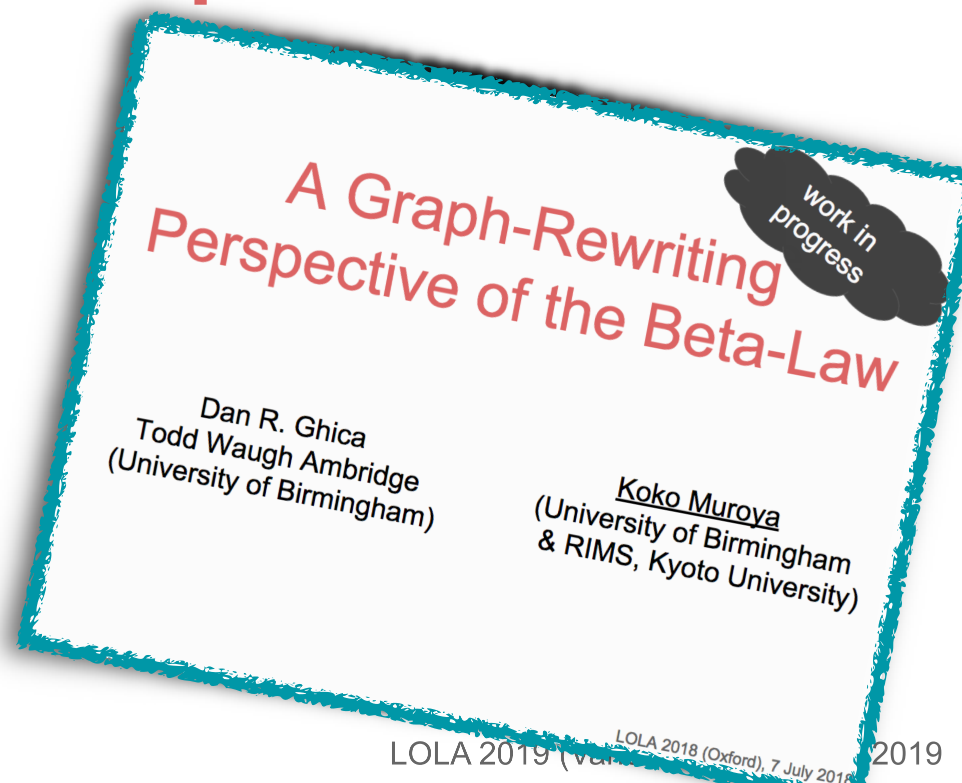
Koko Muroya
(RIMS, Kyoto University
& University of Birmingham)

Dan R. Ghica
Todd Waugh Ambridge
(University of Birmingham)

LOLA 2019 (Vancouver), 23 June 2019

Local reasoning for robust observational equivalence

Koko Muroya
(RIMS, Kyoto University
& University of Birmingham)



Reasoning about observational equivalence

“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
```

$\xrightarrow{?}$

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

$\xrightarrow{?}$

```
50 + 50
```

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

$\xrightarrow{?}$

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

$\xrightarrow{?}$

```
50 + 50
```

Reasoning about observational equivalence

"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`

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

If YES:

- justification of compiler optimisation
- program verification

Reasoning about observational equivalence

"Do two program fragments behave the same?"

Reasoning about observational equivalence

"~~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

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

Reasoning about 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?

Reasoning about 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

Reasoning about observational equivalence

“Do two program fragments behave the same?”

What fragments, in which contexts?

Reasoning about observational equivalence

"Do two program fragments behave the same?"

What fragments, in which contexts?

… 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

Reasoning about observational equivalence

"Do two program fragments behave the same?"

What fragments, in which contexts?

… in the presence of (arbitrary) language features

Analysing robustness/fragility of observational equivalence,

with a general framework

A general framework

to analyse robustness/fragility of observational equivalence:

1. the Spartan calculus

2. a universal abstract machine

3. locality & equational reasoning

1. the Spartan calculus

programming

= copying via variables

+ sharing via atoms/names

+ thunking

+ algebra

$$t, u ::=$$

$$\mid x \mid \texttt{bind}\ x \to u\ \texttt{in}\ t$$

$$\mid a \mid \texttt{new}\ a \multimap u\ \texttt{in}\ t$$

$$\mid x\,.\,t$$

$$\mid \phi(t, \ldots, t;\ \overrightarrow{x}\,.\,u,\ \ldots,\ \overrightarrow{x}\,.\,u)$$

1. the Spartan calculus

programming

= copying via variables

reference to
(a copy of) computation

+ sharing via atoms/names

location of
(shared) computation

+ thunking

+ algebra

$$t, u ::=$$

$$\mid x \mid \texttt{bind}\ x \to u\ \texttt{in}\ t$$

$$\mid a \mid \texttt{new}\ a \multimap u\ \texttt{in}\ t$$

$$\mid x\,.\,t$$

$$\mid \phi(t, \ldots, t;\ \vec{x}\,.\,u, \ \ldots, \vec{x}\,.\,u)$$

1. the SPARTAN calculus

programming

= copying via variables

reference to
(a copy of) computation

+ sharing via atoms/names

location of
(shared) computation

+ thunking

delaying
computation with bound
variable(s)

+ algebra

$$
\begin{aligned}
t, u ::= &\\
& \mid x \mid \mathtt{bind}\ x \to u\ \mathtt{in}\ t\\
& \mid a \mid \mathtt{new}\ a \multimap u\ \mathtt{in}\ t\\
& \mid x\,.\,t\\
& \mid \phi(t, \ldots, t;\ \vec{x}\,.\,u,\ \ldots,\ \vec{x}\,.\,u)
\end{aligned}
$$

1. the SPARTAN calculus

programming

= copying via variables

reference to
(a copy of) computation

+ sharing via atoms/names

location of
(shared) computation

+ thunking

delaying
computation with bound
variable(s)

+ algebra

language features as
(extrinsic) operations

$$t, u ::=$$

$$\mid x \mid \mathtt{bind}\ x \to u\ \mathtt{in}\ t$$

$$\mid a \mid \mathtt{new}\ a \multimap u\ \mathtt{in}\ t$$

$$\mid x\,.\,t$$

$$\mid \phi(t, \ldots, t;\ \vec{x}\,.\,u,\ \ldots,\ \vec{x}\,.\,u)$$

1. the Spartan calculus

programming

= copying via variables

+ sharing via atoms/names

+ thunking

+ algebra

$$
\begin{aligned}
t, u ::=\ & \\
& \mid x \mid \mathtt{bind}\ x \to u\ \mathtt{in}\ t \\
& \mid a \mid \mathtt{new}\ a \multimap u\ \mathtt{in}\ t \\
& \mid x\,.\,t \\
& \mid \phi(t, \ldots, t;\ \vec{x}\,.\,u,\ \ldots,\ \vec{x}\,.\,u)
\end{aligned}
$$

language features as (extrinsic) operations

1. the SPARTAN calculus

programming

= copying via variables

+ sharing via atoms/names

+ thunking

+ algebra

$$
\begin{array}{l}
t, u ::= \\
\quad \mid x \mid \mathtt{bind}\ x \to u\ \mathtt{in}\ t \\
\quad \mid a \mid \mathtt{new}\ a \multimap u\ \mathtt{in}\ t \\
\quad \mid \phi(t, \ldots, t;\ \vec{x}\,.\,u, \ldots, \vec{x}\,.\,u)
\end{array}
$$

eager arguments

deferred arguments (thunks)

language features as (extrinsic) operations

1. the SPARTAN calculus

language features as (extrinsic) operations

$0, 1, 2, 3, \ldots$

$\mathtt{PLUS}(t, u)$

$\mathtt{LAMBDA}(;\ x\,.\,t)$

$\mathtt{APP}(t, u)$

$\mathtt{IF}(t;\ u_1, u_2)$

$\mathtt{LOOKUP}(t;\ x\,.\,u)$

$\mathtt{DEREF}(t)$

$\mathtt{ASSIGN}(t, u)$

+ algebra

$t, u ::=$

$\mid x \mid \mathtt{bind}\ x \to u\ \mathtt{in}\ t$

$\mid a \mid \mathtt{new}\ a \multimap u\ \mathtt{in}\ t$

$\mid \phi(\underline{t}, \ldots, \underline{t};\ \underline{\vec{x}\,.\,u}, \ldots, \underline{\vec{x}\,.\,u})$

eager arguments

deferred arguments (thunks)

A general framework

to analyse robustness/fragility of observational equivalence:

1. the Spartan calculus

   programming = copying + sharing + thunking + algebra

2. a universal abstract machine

3. locality & equational reasoning

2. A universal abstract machine

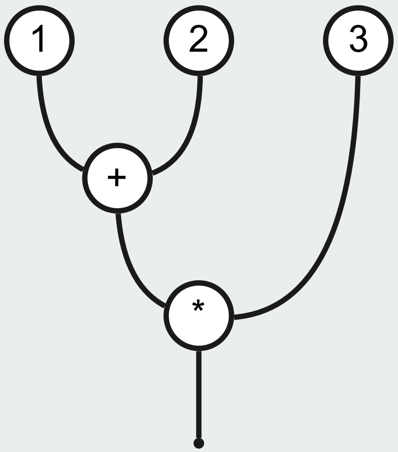
computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

2. A universal abstract machine

computation = focussed “hypernet” rewriting

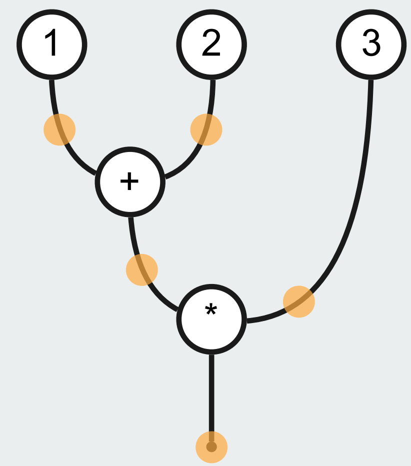
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ $\mathtt{TIMES(PLUS(1,2),3)}$	

2. A universal abstract machine

computation = focussed "hypernet" rewriting

- higher-order hypergraph
- hierarchical hypergraph

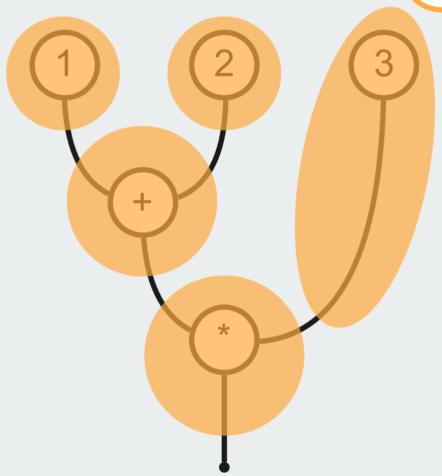
(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

nodes

2. A universal abstract machine

computation = focussed “hypernet” rewriting

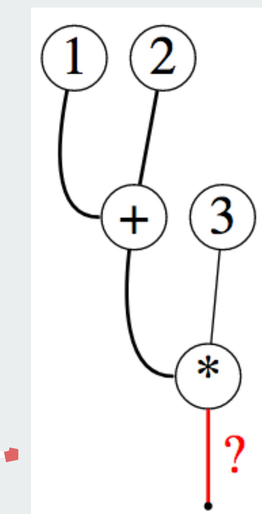
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	hyperedges 

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

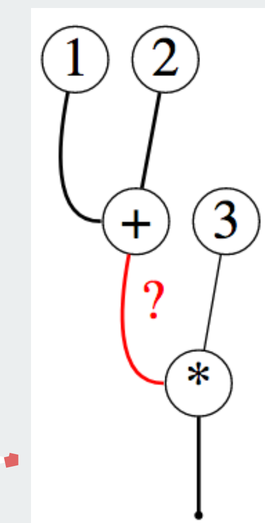
“focus”

- bring query ? up
- bring answer ✓/↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

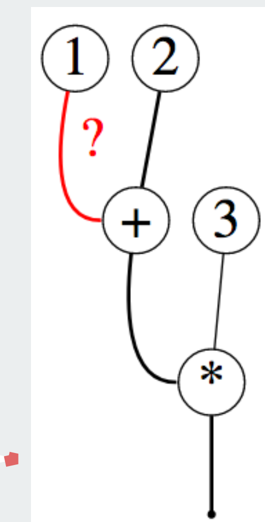
“focus”

- bring query ? up
- bring answer ✓ / ↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed "hypernet" rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

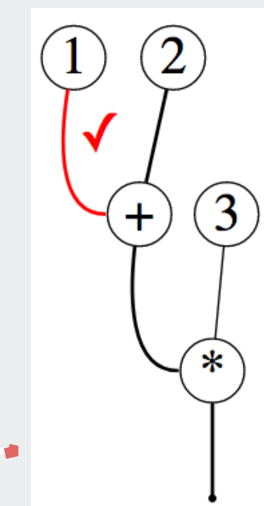
"focus"

- bring query ? up
- bring answer ✓/↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

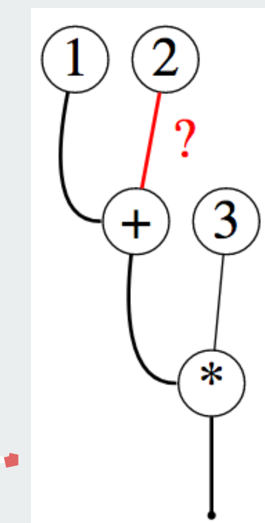
“focus”

- bring query ? up
- bring answer ✓/↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$ TIMES(PLUS(1,2),3)	

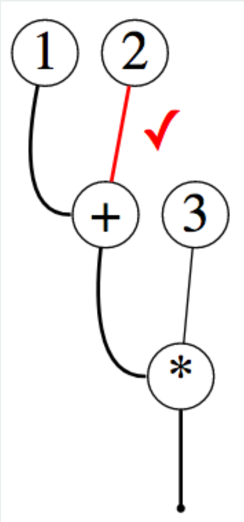
“focus”

- bring query ? up
- bring answer ✓ / ↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed "hypernet" rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

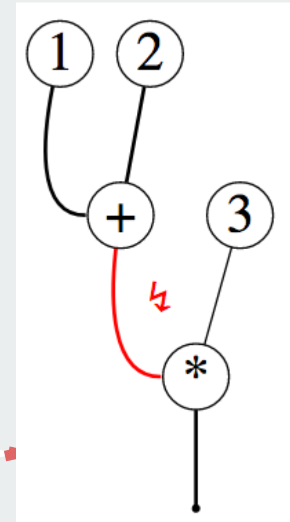
"focus"

- bring query ? up
- bring answer ✓ / ↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

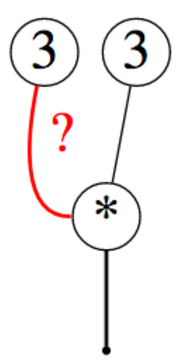
“focus”

- bring query ? up
- bring answer ✓/↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed "hypernet" rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$	
TIMES(PLUS(1,2),3)	

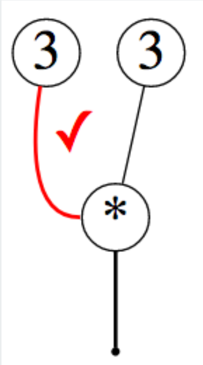
"focus"

- bring query ? up
- bring answer ✓ / ↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2) \times 3$ TIMES(PLUS(1,2),3)	

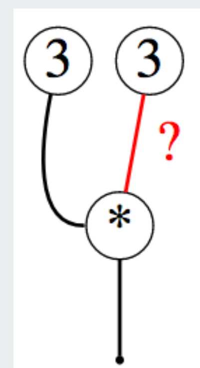
“focus”

- bring query ? up
- bring answer ✓/↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed "hypernet" rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

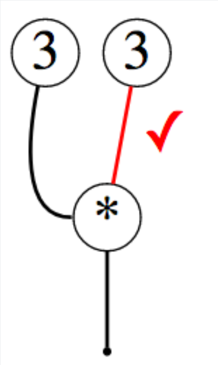
"focus"

- bring query ? up
- bring answer ✓ / ↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

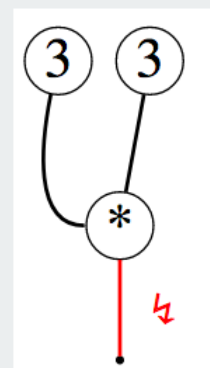
“focus”

- bring query ? up
- bring answer ✓ / ↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

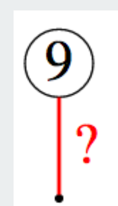
“focus”

- bring query ? up
- bring answer ✓/↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed "hypernet" rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1+2)\times 3$ TIMES(PLUS(1,2),3)	

"focus"

- bring query ? up
- bring answer ✓ / ↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$ TIMES(PLUS(1,2),3)	

“focus”

- bring query ? up
- bring answer ✓ / ↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

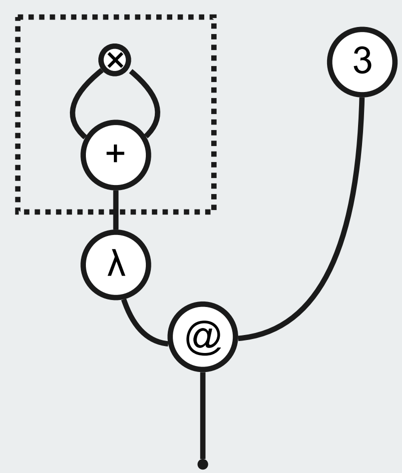
- higher-order hypergraph
- hierarchical hypergraph

<table>
<tr><th>(SPARTAN) term</th><th>hypernet</th></tr>
<tr><td>

$(\lambda x\,.\,x + x)\,3$

$\mathrm{APP}(\mathrm{LAMBDA}(;\ x\,.\,\mathrm{PLUS}(x, x)),\ 3)$

</td><td>

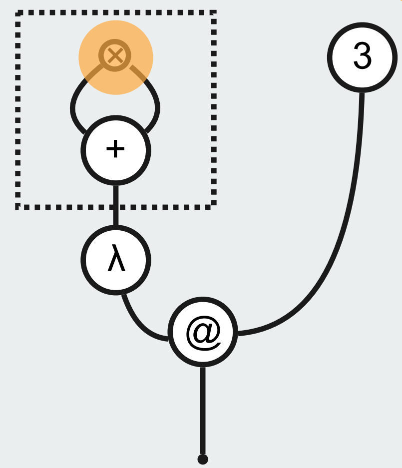


</td></tr>
</table>

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

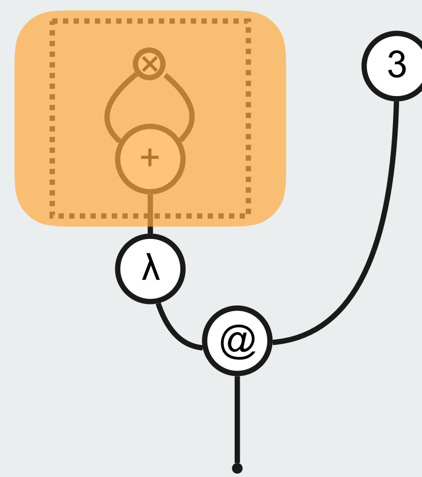
(SPARTAN) term	hypernet
$(\lambda x\,.\,x + x)\,3$ $\mathtt{APP}(\mathtt{LAMBDA}(;\ x\,.\,\mathtt{PLUS}(x, x)),\ 3)$	

copying

2. A universal abstract machine

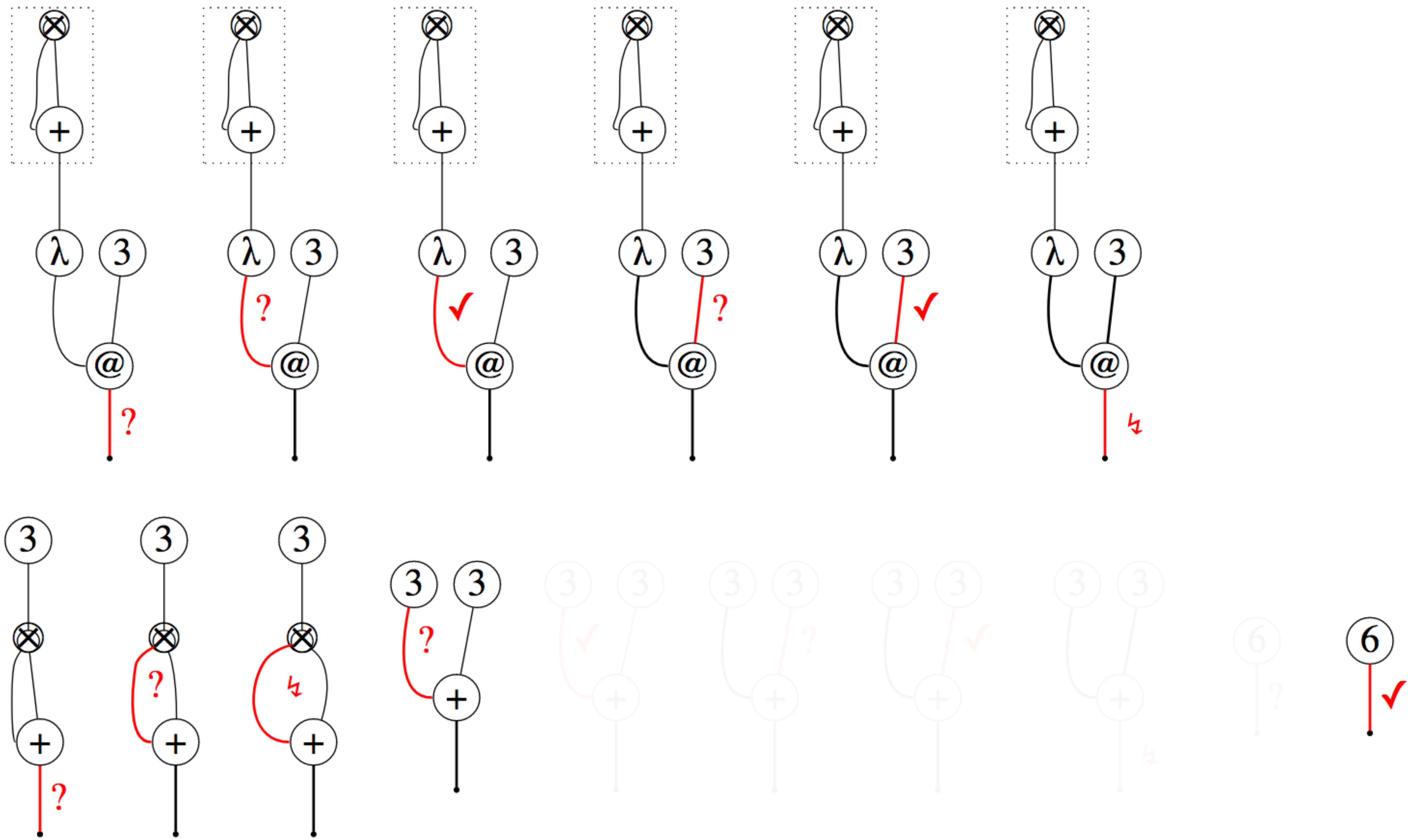
computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

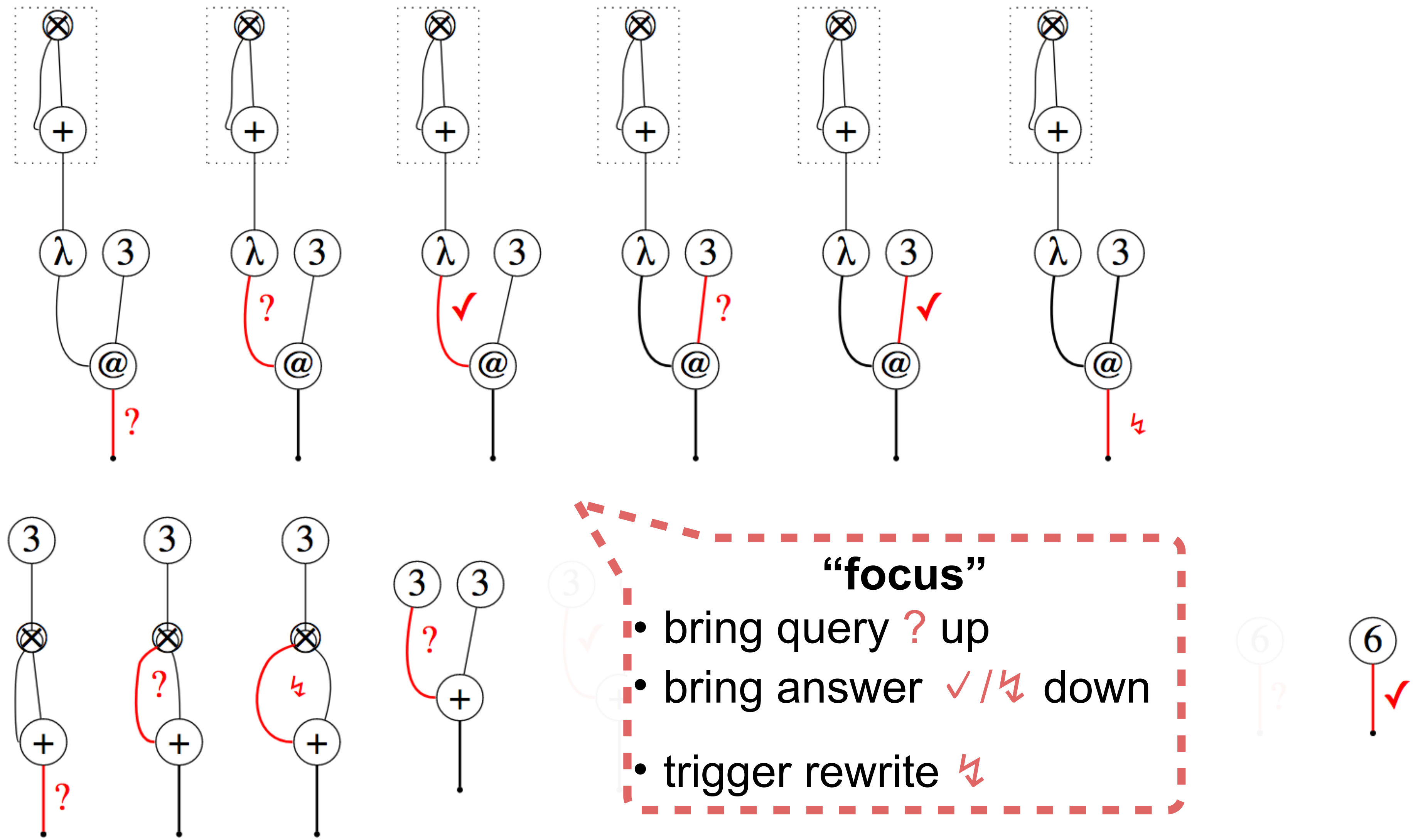
(SPARTAN) term	hypernet
$(\lambda x\,.\,x + x)\,3$ $\mathtt{APP}(\mathtt{LAMBDA}(;\, x\,.\,\mathtt{PLUS}(x, x)),\ 3)$	

hyperedge labelled with hypernet

$(\lambda x\,.\,x + x)\,3$



$(\lambda x \,.\, x + x)\, 3$



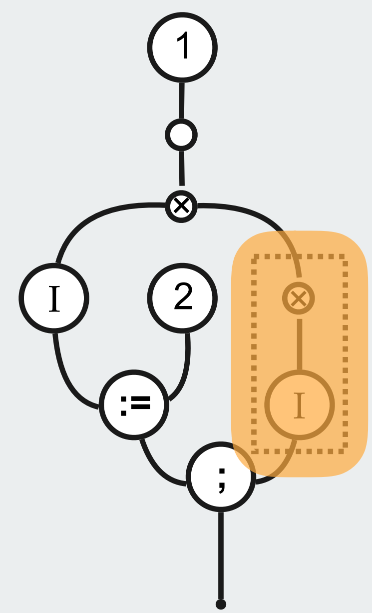
“focus”

- bring query ? up
- bring answer ✓/↯ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed "hypernet" rewriting

- higher-order hypergraph
- hierarchical hypergraph

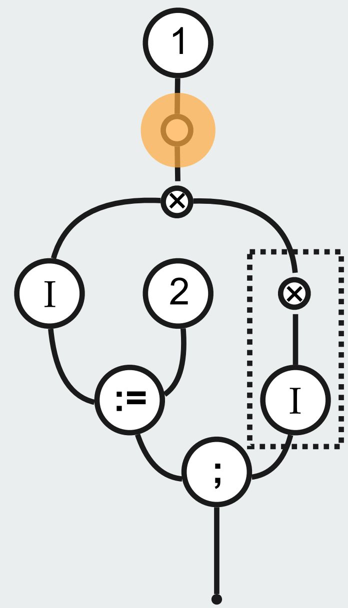
(SPARTAN) term	hypernet
$\mathtt{new}\ a \multimap 1\ \mathtt{in}\ (a := 2;\ !a)$ $\mathtt{new}\ a \multimap 1\ \mathtt{in}$ $\mathrm{SEC}(\mathrm{ASSIGN}(a,2);\ \mathrm{DEREF}(a))$	

hyperedge labelled with hypernet

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

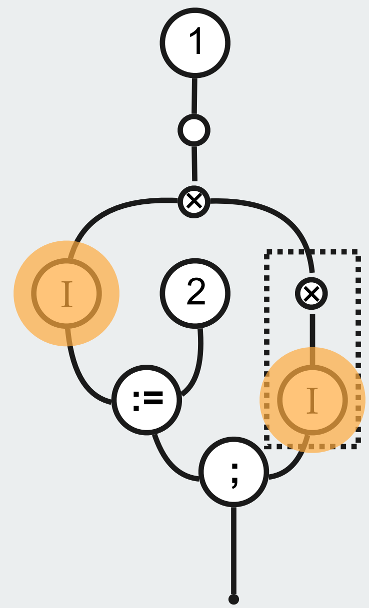
(SPARTAN) term	hypernet
new $a \multimap 1$ in $(a := 2;\ !a)$ new $a \multimap 1$ in SEC(ASSIGN(a,2); DEREF(a))	

sharing

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(Spartan) term	hypernet
$\mathtt{new}\ a \multimap 1\ \mathtt{in}\ (a := 2;\ !a)$ $\mathtt{new}\ a \multimap 1\ \mathtt{in}$ $\mathtt{SEC}(\mathtt{ASSIGN}(a,2);\ \mathtt{DEREF}(a))$	

name occurrences

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

program	hypernet
program fragment	sub-hypernet (sub-graph)
program execution	moves of focus (query / answer) & focussed rewrites

A general framework

to analyse robustness/fragility of observational equivalence:

1. the Spartan calculus

   programming = copying + sharing + thunking + algebra

2. a universal abstract machine

   computation = focussed "hypernet" rewriting

3. locality & equational reasoning

3. Locality & equational reasoning

“Do two program fragments behave the same?”

3. Locality & equational reasoning

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

"Do two <u>sub-graphs</u> behave the same in focussed rewriting?"

3. Locality & equational reasoning

Locality of graph syntax

"Does $\texttt{new}\ a \multimap 1\ \texttt{in}\ \lambda x\,.\,(a := 2;\ !a)$ behave the same as $\lambda x\,.\,2$?"

3. Locality & equational reasoning

Locality of graph syntax

“Does $\mathtt{new}\ a \multimap 1\ \mathtt{in}\ \lambda x.(a := 2;\ !a)$ behave the same as $\lambda x.2$?”

with linear syntax:

$\cdots$	$\mathtt{new}\ a \multimap 1\ \mathtt{in}$	$\cdots$	$\lambda x.(a := 2;\ !a)$	$\cdots$	$\lambda x.(a := 2;\ !a)$	$\cdots$

$\cdots$	$\lambda x.2$	$\cdots$	$\lambda x.2$	$\cdots$

3. Locality & equational reasoning

Locality of graph syntax

“Does $\mathtt{new}\ a \multimap 1\ \mathtt{in}\ \lambda x.(a := 2;\ !a)$ behave the same as $\lambda x.2$?”

with linear syntax: ~~comparison between sub-terms~~

$\cdots$	$\mathtt{new}\ a \multimap 1\ \mathtt{in}$	$\cdots$	$\lambda x.(a := 2;\ !a)$	$\cdots$	$\lambda x.(a := 2;\ !a)$	$\cdots$
$\cdots$			$\lambda x.2$	$\cdots$	$\lambda x.2$	$\cdots$

3. Locality & equational reasoning

Locality of graph syntax

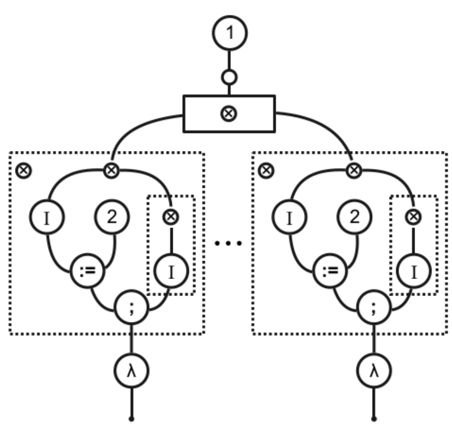
“Does $\texttt{new}\ a \multimap 1\ \texttt{in}\ \lambda x.(a := 2;\ !a)$ behave the same as $\lambda x.2$?”

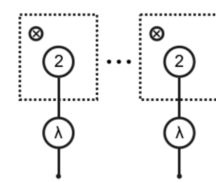
with linear syntax: ~~comparison between sub-terms~~

$\cdots$	$\texttt{new}\ a \multimap 1\ \texttt{in}$	$\cdots$	$\lambda x.(a := 2;\ !a)$	$\cdots$	$\lambda x.(a := 2;\ !a)$	$\cdots$

$\cdots$	$\lambda x.2$	$\cdots$	$\lambda x.2$	$\cdots$

with graph syntax: comparison between sub-graphs





3. Locality & equational reasoning

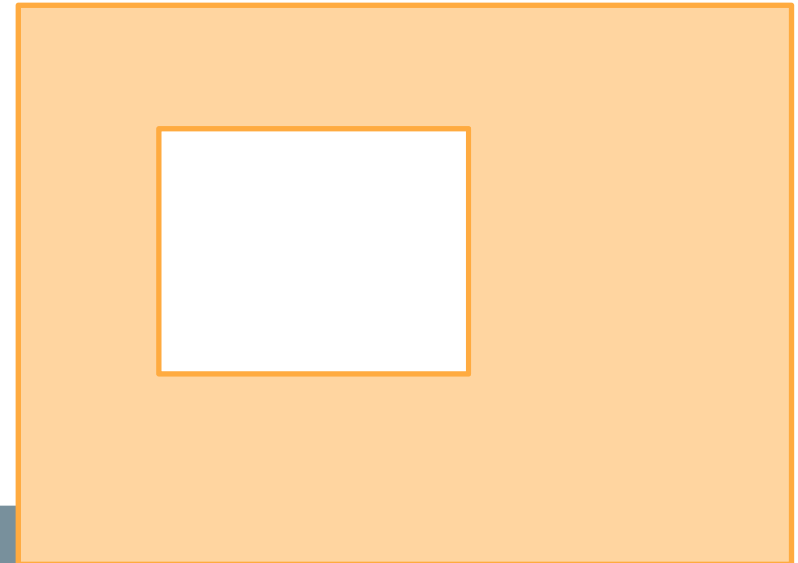
~~*"Do two program fragments behave the same?"*~~

"Do two sub-graphs <u>behave</u> the same in focussed rewriting?"

3. Locality & equational reasoning

Locality of focussed rewriting

“How does a sub-graph behave?”



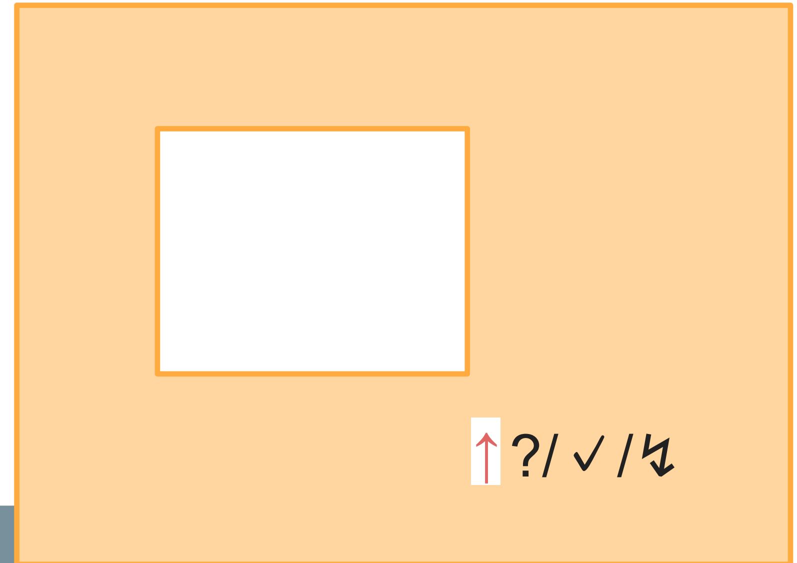
Case analysis

1. query/answer in context
2. query to the sub-graph
3. answer to the sub-graph
4. focussed rewrite in context

3. Locality & equational reasoning

Locality of focussed rewriting

“How does a sub-graph behave?”



Case analysis

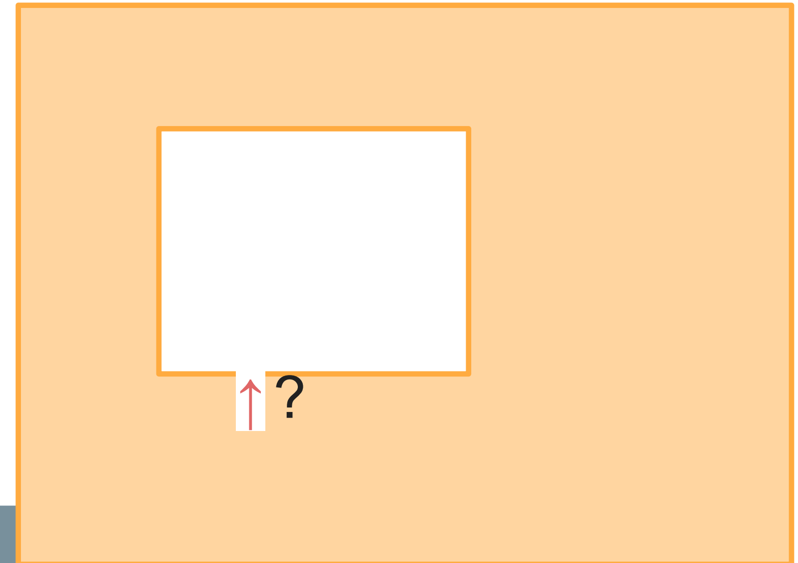
1. query/answer in context
2. query to the sub-graph
3. answer to the sub-graph
4. focussed rewrite in context

(no interference)

3. Locality & equational reasoning

Locality of focussed rewriting

“How does a sub-graph behave?”



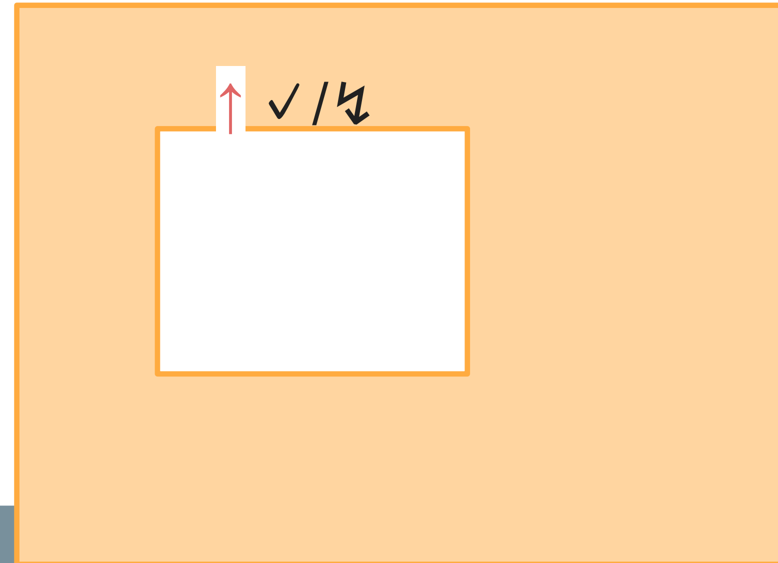
Case analysis

1. query/answer in context	(no interference)
2. query to the sub-graph	interference
3. answer to the sub-graph	
4. focussed rewrite in context	

3. Locality & equational reasoning

Locality of focussed rewriting

"How does a sub-graph behave?"



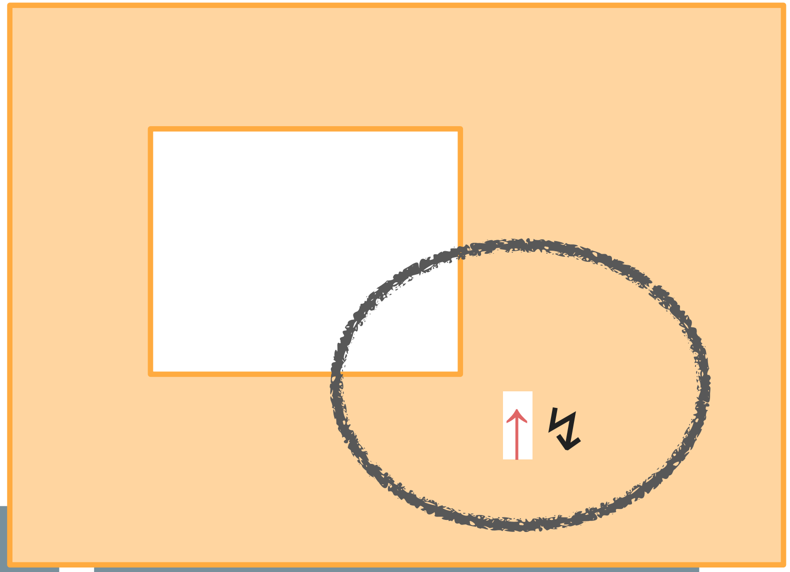
Case analysis

1. query/answer in context	(no interference)
2. query to the sub-graph	interference
3. answer to the sub-graph	interference
4. focussed rewrite in context	

3. Locality & equational reasoning

Locality of focussed rewriting

“How does a sub-graph behave?”



Case analysis

1. query/answer in context	(no interference)
2. query to the sub-graph	interference
3. answer to the sub-graph	interference
4. focussed rewrite in context	possible interference

3. Locality & equational reasoning

"Do two sub-graphs behave the same in focussed rewriting?"

Case analysis	with the sub-graphs:
1. query/answer in context	(no interference)
2. query to the sub-graphs	interference
3. answer to the sub-graphs	interference
4. focussed rewrite in context	possible interference

3. Locality & equational reasoning

"Do two sub-graphs behave the same in focussed rewriting?"

Case analysis	the sub-graphs are interfered the same:
1. query/answer in context	(always)
2. query to the sub-graphs	if *input-safe*
3. answer to the sub-graphs	if *output-closed*
4. focussed rewrite in context	if *robust*

Characterisation Theorem
Robust templates induce observational equivalences.

3. Locality & equational reasoning

"Do two sub-graphs behave the same in focussed rewriting?"

Case analysis	the sub-graphs are interfered the same:
1. query/answer in context	(always)
2. query to the sub-graphs	if ***input-safe***
3. answer to the sub-graphs	if ***output-closed***
4. focussed rewrite in context	if *robust*

Characterisation Theorem
*Robust **templates** induce observational equivalences.*

3. Locality & equational reasoning

"Do two sub-graphs behave the same in focussed rewriting?"

Case analysis	the sub-graphs are interfered the same:
1. query/answer in context	(always)
2. query to the sub-graphs	if *input-safe*
3. answer to the sub-graphs	if *output-closed*
4. focussed rewrite in context	if ***robust***

Characterisation Theorem
Robust *templates induce observational equivalences.*

A general framework

to analyse robustness/fragility of observational equivalence:

1. the SPARTAN calculus

   programming = copying + sharing + thunking + algebra

2. a universal abstract machine

   computation = focussed "hypernet" rewriting

3. locality & equational reasoning

Characterisation Theorem
Robust templates induce observational equivalences.

Directions

- beyond determinism
  - nondeterminism, probability, I/O
  - refined notion of observational equivalence
- concurrency
  - rewriting with multiple focusses (cf. multi-token GoI)
- types
  - weak notion of safety
- tooling
  - https://tnttodda.github.io/Spartan-Visualiser/
- cost analysis
  - observational equivalence with cost improvement (*improvement theory*)
  - cost model of focussed hypernet rewriting (cf. dynamic GoI)
- strengthening mathematics
  - hypernets = trees + hierarchy + copying + sharing
  - *focussed* DPO rewriting
  - reasoning with graph contexts