

Local reasoning for robust observational equivalence

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

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

Local reasoning for robust observational equivalence

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

A Graph-Rewriting
Perspective of the Beta-Law

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

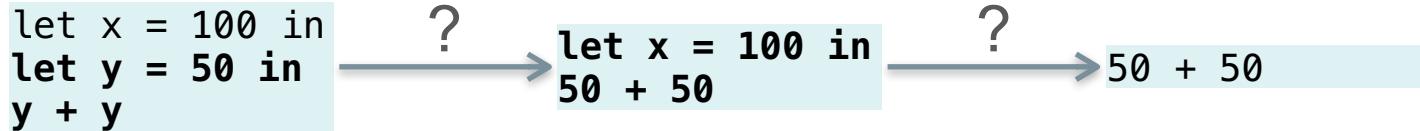
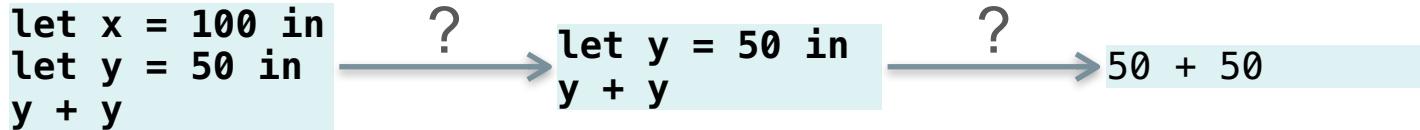
Koko Muroya
(University of Birmingham
& RIMS, Kyoto University)

work in
progress

Reasoning about observational equivalence

“Do two program fragments behave the same?”

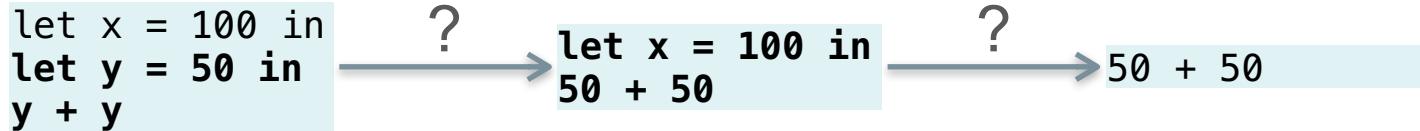
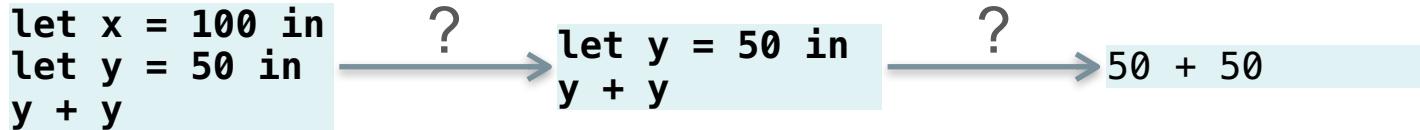
“Is it safe to replace a program fragment with another?”



Reasoning about observational equivalence

“Do two program fragments behave the same?”

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



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

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

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

$t, u ::=$

$| x | \text{bind } x \rightarrow u \text{ in } t$

$| a | \text{new } a \multimap u \text{ in } t$

$| x . t$

$| \phi(t, \dots, t; \vec{x} . u, \dots, \vec{x} . u)$

1. the SPARTAN calculus

programming

= copying via variables

reference to
(a copy of) computation

+ sharing via atoms/names

location of
(shared) computation

+ thunking

+ algebra

$t, u ::=$

$| x | \text{bind } x \rightarrow u \text{ in } t$

$| a | \text{new } a \multimap u \text{ in } t$

$| x . t$

$| \phi(t, \dots, t; \vec{x} . u, \dots, \vec{x} . u)$

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

$t, u ::=$

$| x | \text{bind } x \rightarrow u \text{ in } t$

$| a | \text{new } a \multimap u \text{ in } t$

$| x . t$

$| \phi(t, \dots, t; \vec{x} . u, \dots, \vec{x} . u)$

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

language features as
(extrinsic) operations

$t, u ::=$

$| x | \text{bind } x \rightarrow u \text{ in } t$

$| a | \text{new } a \multimap u \text{ in } t$

$| x . t$

$| \phi(t, \dots, t; \vec{x} . u, \dots, \vec{x} . u)$

1. the SPARTAN calculus

programming

= copying via variables

+ sharing via atoms/names

+ thunking

+ algebra

language features as
(extrinsic) operations

$t, u ::=$

$| x | \text{bind } x \rightarrow u \text{ in } t$

$| a | \text{new } a \multimap u \text{ in } t$

$| x.t$

$| \phi(t, \dots, t; \vec{x}.u, \dots, \vec{x}.u)$

1. the SPARTAN calculus

programming

= copying via variables

+ sharing via atoms/names

+ thunking

+ algebra

language features as
(extrinsic) operations

$t, u ::=$

$| x | \text{bind } x \rightarrow u \text{ in } t$

$| a | \text{new } a \multimap u \text{ in } t$

eager
arguments

deferred
arguments
(thunks)

$| \phi(t, \dots, t; \overrightarrow{x} . u, \dots, \overrightarrow{x} . u)$

1. the SPARTAN calculus

language features as
(extrinsic) operations

0,1,2,3,...

PLUS(t, u)

LAMBDA(; $x . t$)

IF($t; u_1, u_2$)

APP(t, u)

LOOKUP($t; x . u$)

DEREF(t)

ASSIGN(t, u)

+ algebra

$t, u ::=$

$| x | \text{bind } x \rightarrow u \text{ in } t$

$| a | \text{new } a \multimap u \text{ in } t$

eager
arguments

deferred
arguments
(thunks)

$| \phi(\underline{t}, \dots, \underline{t}; \ \overrightarrow{x} . u, \dots, \overrightarrow{x} . u)$

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

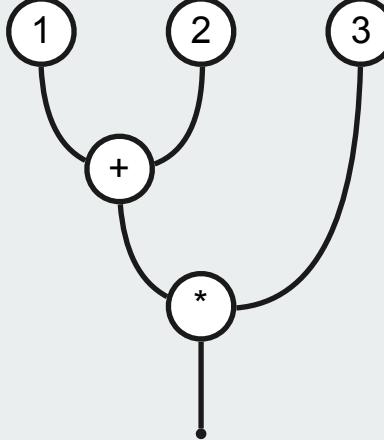
computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

2. A universal abstract machine

computation = focussed “hypernet” rewriting

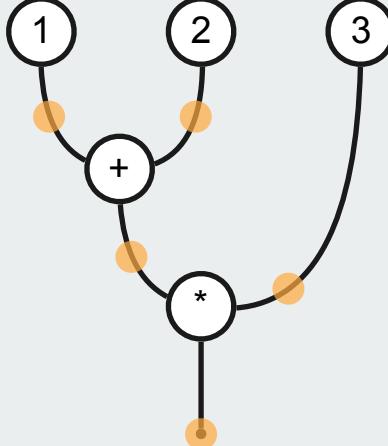
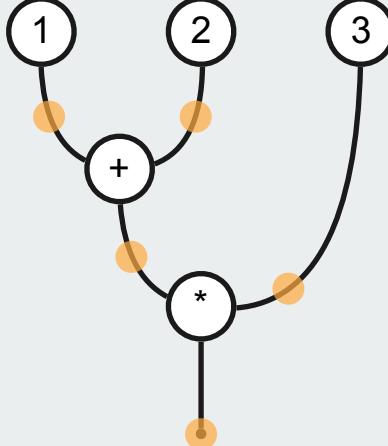
- higher-order hypergraph
- hierarchical hypergraph

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

2. A universal abstract machine

computation = focussed “hypernet” rewriting

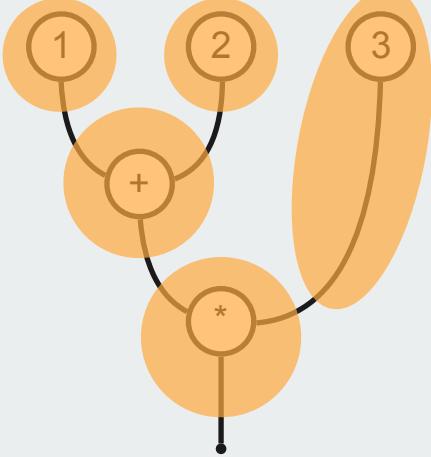
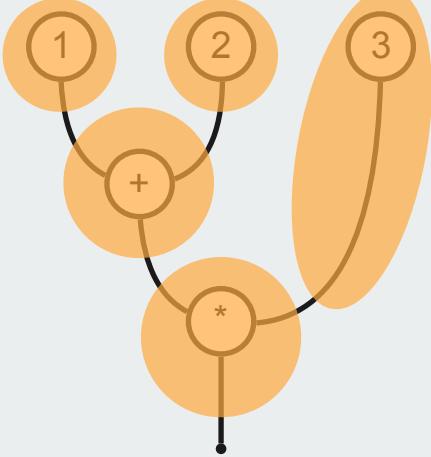
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$	
$\text{TIMES}(\text{PLUS}(1,2),3)$	<p>nodes</p> 

2. A universal abstract machine

computation = focussed “hypernet” rewriting

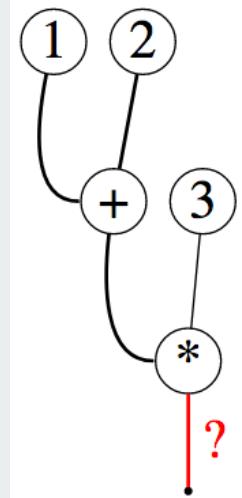
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$	 <p>hyperedges</p>
$\text{TIMES}(\text{PLUS}(1,2),3)$	 <p>hyperedges</p>

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)	<p>“focus”</p> <ul style="list-style-type: none">• bring query ? up• bring answer ✓/↖ down• trigger rewrite ↖

2. A universal abstract machine

computation = focussed “hypernet” rewriting

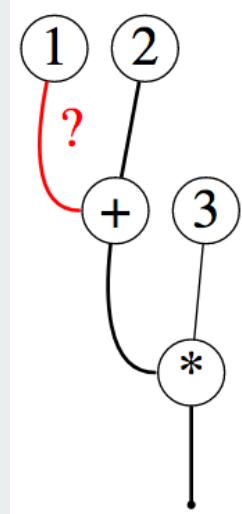
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$	
TIMES(PLUS(1,2),3)	<p>“focus”</p> <ul style="list-style-type: none">• bring query ? up• bring answer ✓/↖ down• trigger rewrite ↖

2. A universal abstract machine

computation = focussed “hypernet” rewriting

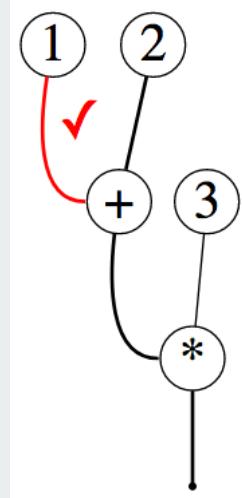
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$	
TIMES(PLUS(1,2),3)	<p>“focus”</p> <ul style="list-style-type: none">• bring query ? up• bring answer $\checkmark/\text{↯}$ down• trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

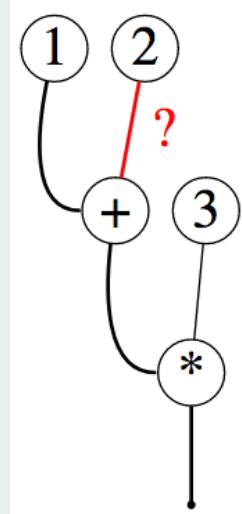
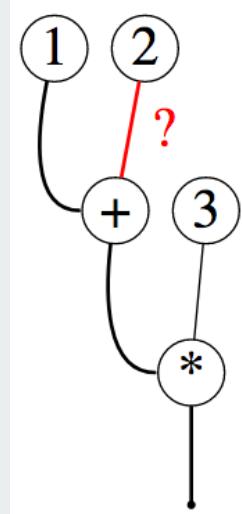
(SPARTAN) term	hypernet
$(1 + 2) \times 3$	
TIMES(PLUS(1,2),3)	 <p>The hypergraph consists of five nodes arranged in two rows. The top row contains nodes 1 and 2. The bottom row contains nodes + and 3 on the left, and * on the right. Edges connect 1 to + (red with checkmark), 2 to +, and 3 to *. A dashed red rectangle encloses nodes 1, 2, and +, with the word "focus" written inside.</p>

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

2. A universal abstract machine

computation = focussed “hypernet” rewriting

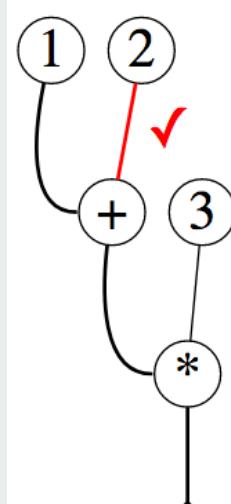
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$	
TIMES(PLUS(1,2),3)	 <p>“focus”</p> <ul style="list-style-type: none">• bring query ? up• bring answer $\checkmark/\text{↯}$ down• trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

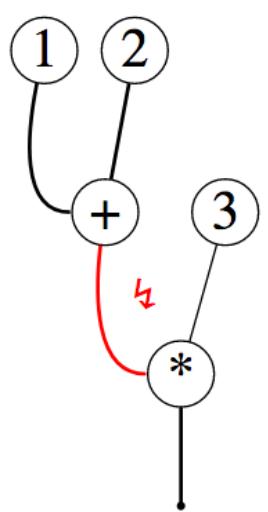
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$	
TIMES(PLUS(1,2),3)	<p>“focus”</p> <ul style="list-style-type: none">• bring query ? up• bring answer $\checkmark/\text{↯}$ 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)	

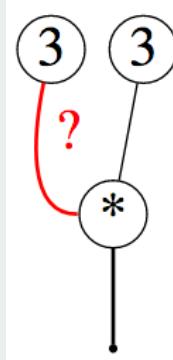
“focus”

- bring query ? up
- bring answer $\checkmark / \text{↯}$ down
- trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

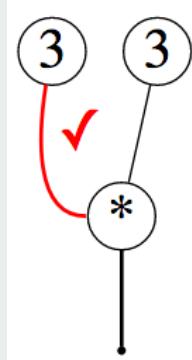
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$	
TIMES(PLUS(1,2),3)	 <p>A hypergraph diagram illustrating the rewriting process. It shows a query node containing a question mark (?), which is connected by a red curved arrow to a multiplication node (*). The multiplication node is connected by a black line to an answer node.</p> <p>The diagram is enclosed in a dashed red box labeled "focus".</p> <ul style="list-style-type: none">• 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)	 <p>The diagram illustrates a hypernet corresponding to the expression $(1 + 2) \times 3$. It consists of four nodes: two nodes labeled '3' and one node labeled '*' (representing multiplication). The nodes are interconnected: one '3' node is connected to the '*' node, and both nodes are connected to the other '3' node. A red curved arrow highlights the path from the first '3' node through the '*' node back to the second '3' node, indicating a specific focus or rewrite step.</p>

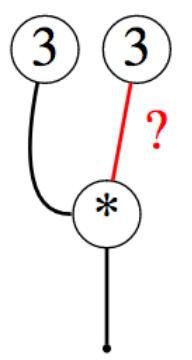
“focus”

- bring query $\textcolor{red}{?}$ up
- bring answer $\checkmark / \textcolor{red}{\downarrow}$ down
- trigger rewrite $\textcolor{red}{\downarrow}$

2. A universal abstract machine

computation = focussed “hypernet” rewriting

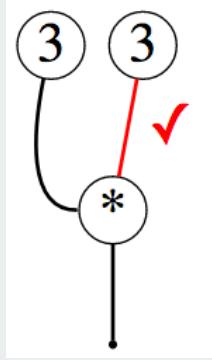
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$	
TIMES(PLUS(1,2),3)	 <p>A hypergraph diagram illustrating the rewriting of the term $\text{TIMES}(\text{PLUS}(1,2),3)$. It consists of four nodes: two nodes labeled '3' at the top left and top right, a central node labeled '*' at the bottom, and a red question mark node '?' positioned above the '*' node. A black curved edge connects the left '3' node to the '*' node. A red curved edge connects the right '3' node to the '?' node. A black vertical edge connects the '*' node to the bottom.</p> <p>“focus”</p> <ul style="list-style-type: none">• bring query ? up• bring answer $\checkmark/\text{↯}$ 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)	 <p>The diagram shows a hypergraph with three nodes. Two nodes are labeled '3' and one is labeled '*'. There are two edges: one connecting the left '3' node to the '*' node, and another connecting the right '3' node to the '*' node. A red checkmark is placed on the edge between the right '3' node and the '*' node.</p>

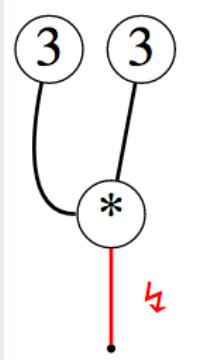
“focus”

- bring query ? up
- bring answer $\checkmark/\text{↯}$ 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)	 A hypergraph with three nodes. Two nodes are labeled with the number 3, and one node is labeled with an asterisk (*). There are two edges connecting the two '3' nodes to the '*' node. A red arrow points downwards from the '*' node towards the bottom right corner of the slide.

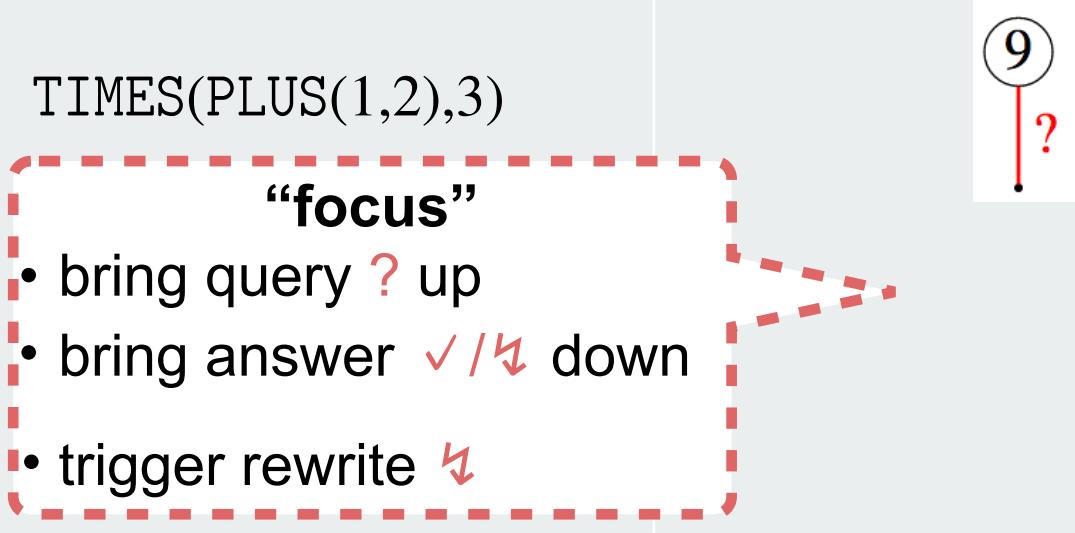
“focus”

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

2. A universal abstract machine

computation = focussed “hypernet” rewriting

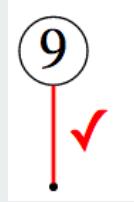
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$	
TIMES(PLUS(1,2),3)	 <p>“focus”</p> <ul style="list-style-type: none">• bring query ? up• bring answer $\checkmark / \text{↯}$ down• trigger rewrite ↯

2. A universal abstract machine

computation = focussed “hypernet” rewriting

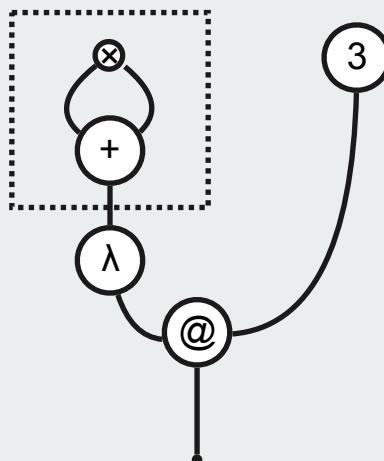
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(1 + 2) \times 3$ TIMES(PLUS(1,2),3) “focus” <ul style="list-style-type: none">• bring query ? up• bring answer ✓ / ↴ down• trigger rewrite ↴	

2. A universal abstract machine

computation = focussed “hypernet” rewriting

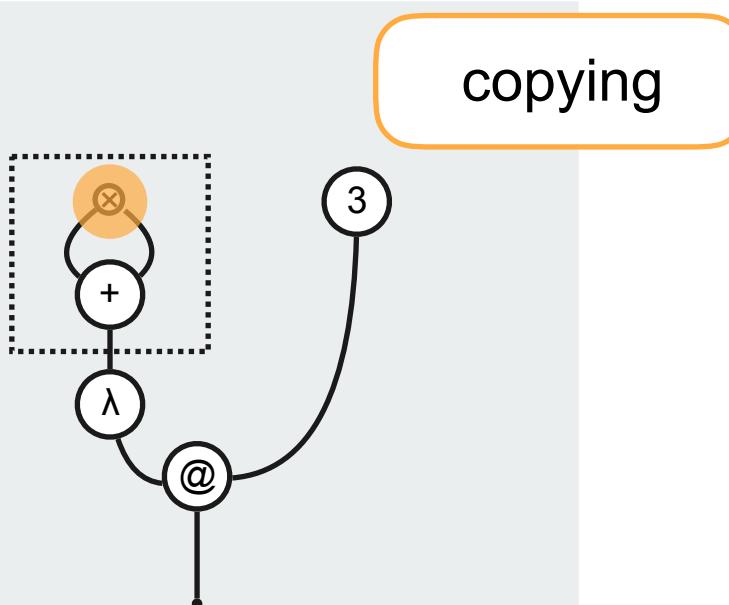
- higher-order hypergraph
- hierarchical hypergraph

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

2. A universal abstract machine

computation = focussed “hypernet” rewriting

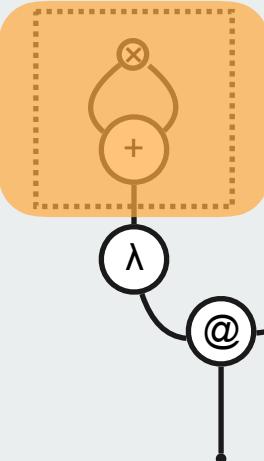
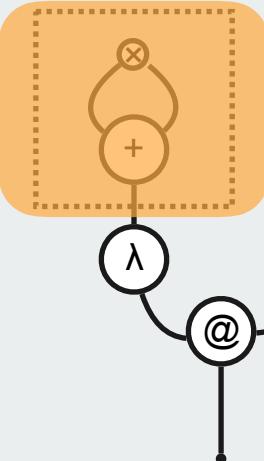
- higher-order hypergraph
- hierarchical hypergraph

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

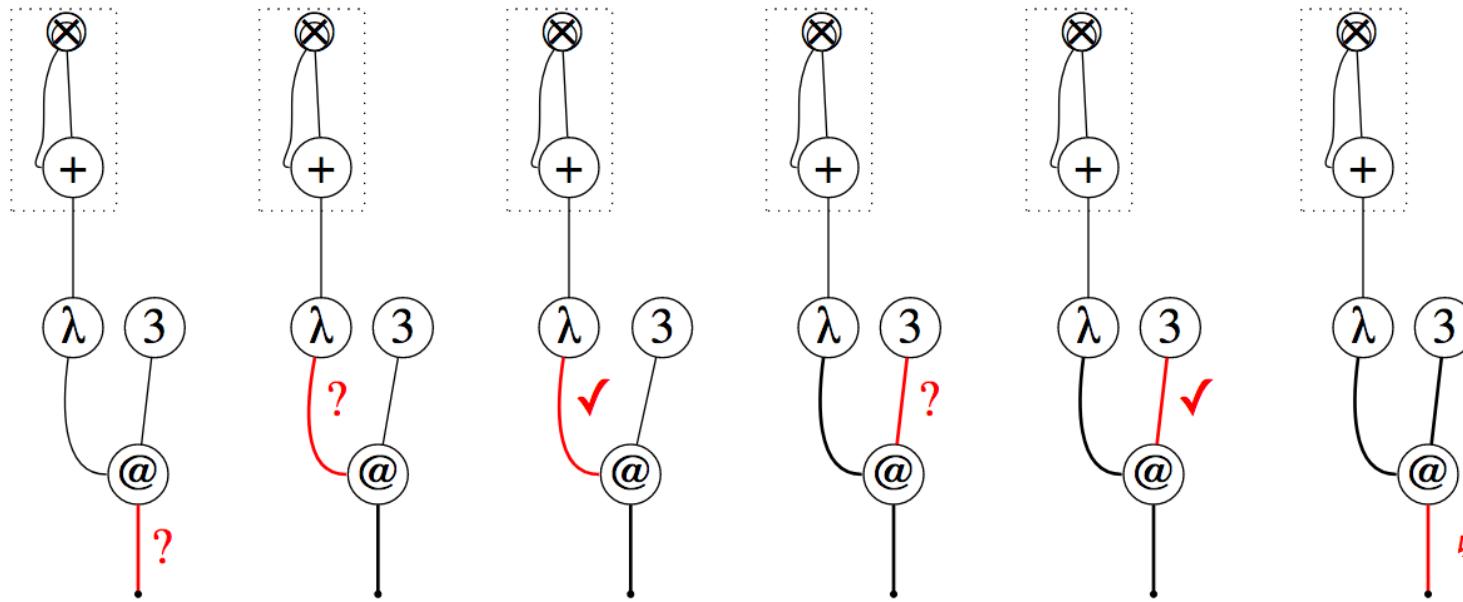
2. A universal abstract machine

computation = focussed “hypernet” rewriting

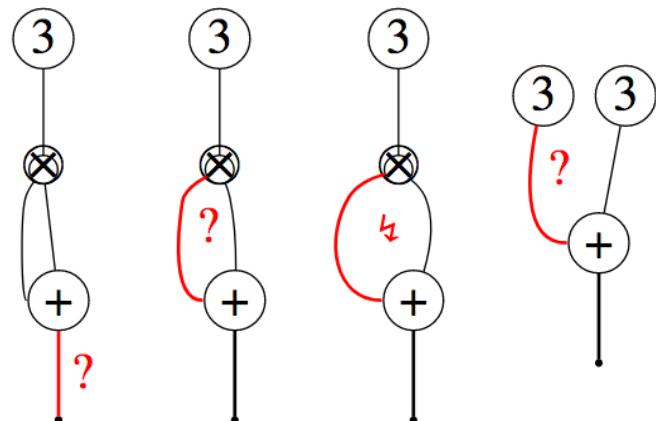
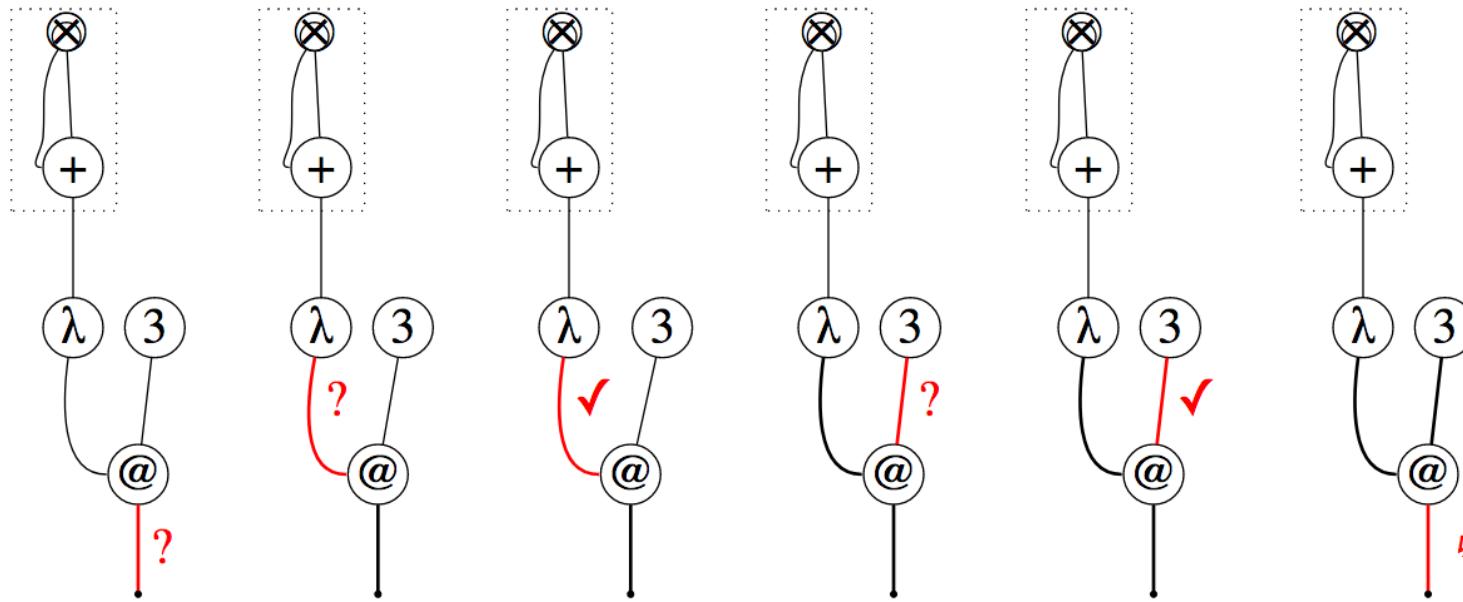
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$(\lambda x . x + x) 3$	
APP(LAMBDA(; x . PLUS(x, x)), 3)	 <p>hyperedge labelled with hypernet</p>

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



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



“focus”

- bring query ? up
- bring answer $\checkmark / \text{\textcolor{red}{\text{?}}}$ down
- trigger rewrite $\text{\textcolor{red}{\text{?}}}$



2. A universal abstract machine

computation = focussed “hypernet” rewriting

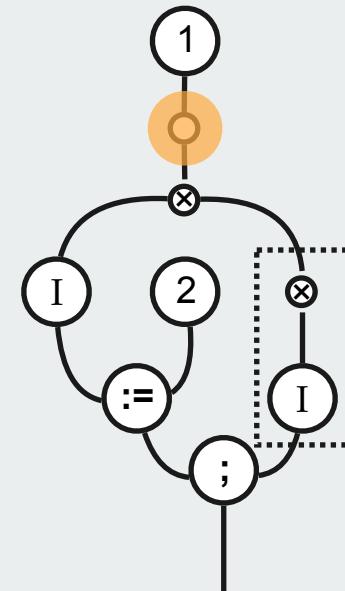
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet	
$\text{new } a \multimap 1 \text{ in } (a := 2; !a)$		hyperedge labelled with hypernet
$\text{new } a \multimap 1 \text{ in }$ $\text{SEC}(\text{ASSIGN}(a,2); \text{DEREF}(a))$		

2. A universal abstract machine

computation = focussed “hypernet” rewriting

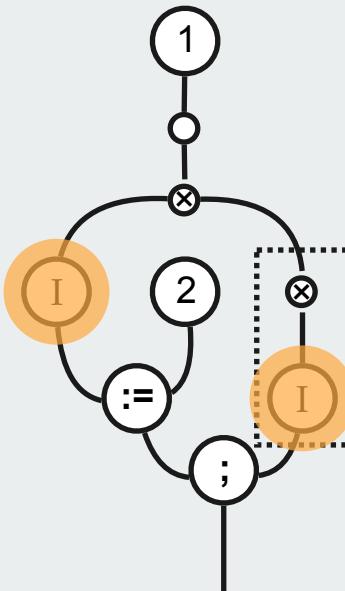
- higher-order hypergraph
- hierarchical hypergraph

(SPARTAN) term	hypernet
$\text{new } a \multimap 1 \text{ in } (a := 2; !a)$	 <p>sharing</p>
$\text{new } a \multimap 1 \text{ in }$ $\text{SEC}(\text{ASSIGN}(a,2); \text{DEREF}(a))$	

2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

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

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 sub-graphs behave the same in focussed rewriting?”

3. Locality & equational reasoning

Locality of graph syntax

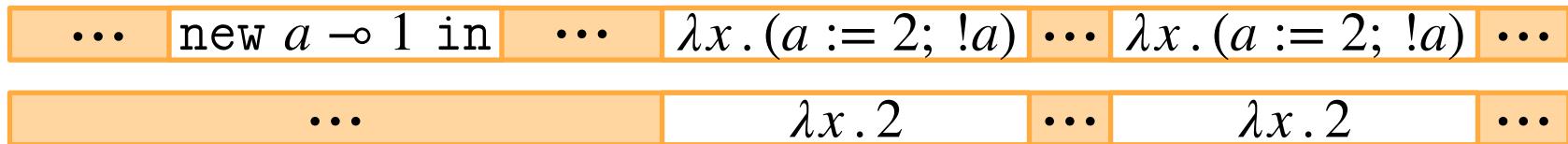
“Does `new a -> 1 in λx.(a := 2; !a)` behave the same as `λx.2`? ”

3. Locality & equational reasoning

Locality of graph syntax

“Does `new a → 1 in λx.(a := 2; !a)` behave the same as `λx.2`?”

with linear syntax:

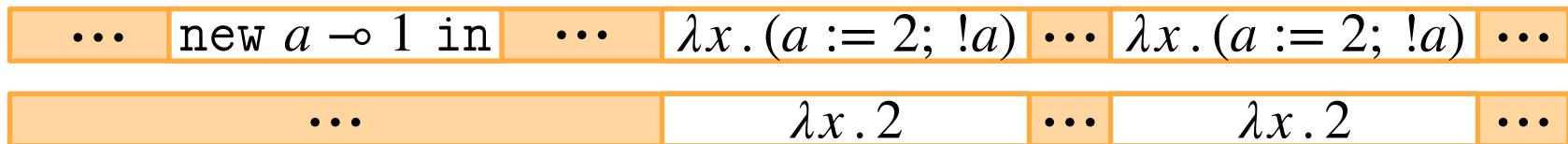


3. Locality & equational reasoning

Locality of graph syntax

“Does `new a → 1 in λx.(a := 2; !a)` behave the same as `λx.2`?”

with linear syntax: comparison between sub-terms

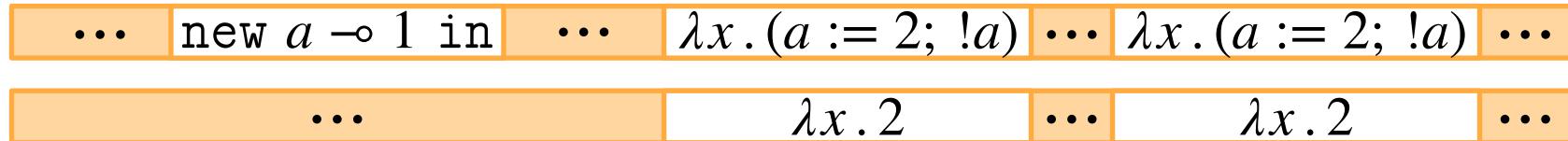


3. Locality & equational reasoning

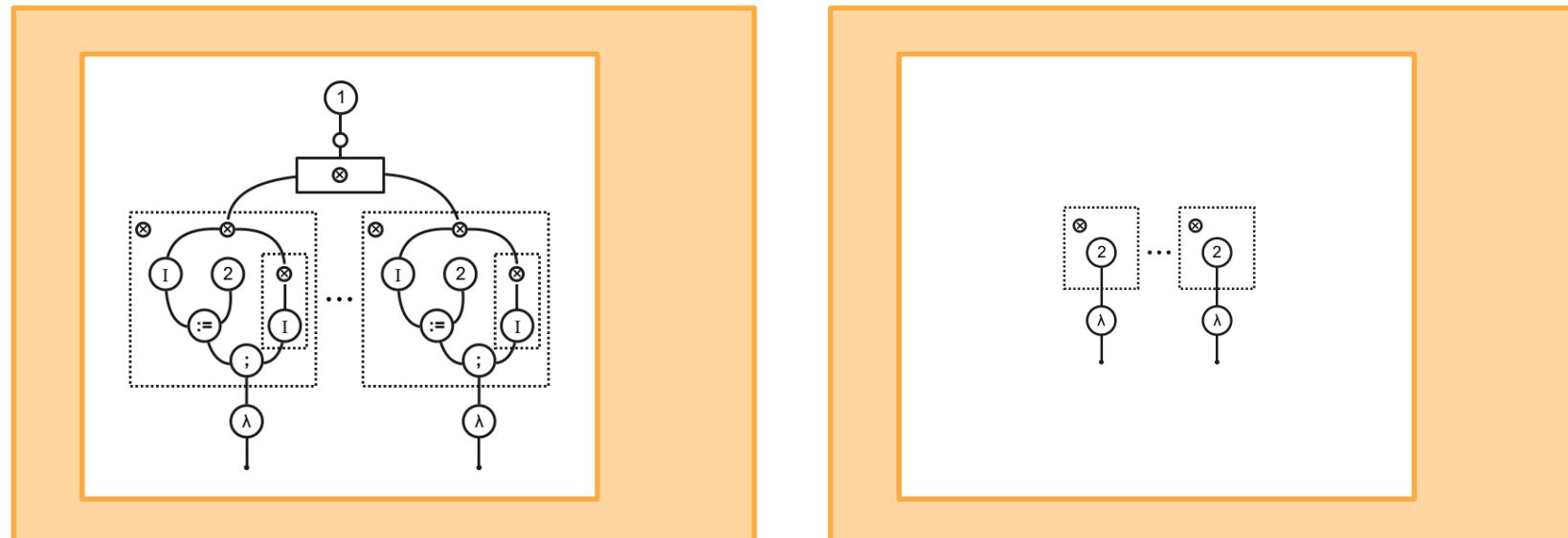
Locality of graph syntax

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

with linear syntax: comparison between sub-terms



with graph syntax: comparison between sub-graphs



3. Locality & equational reasoning

~~“Do two program fragments behave the same?”~~

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

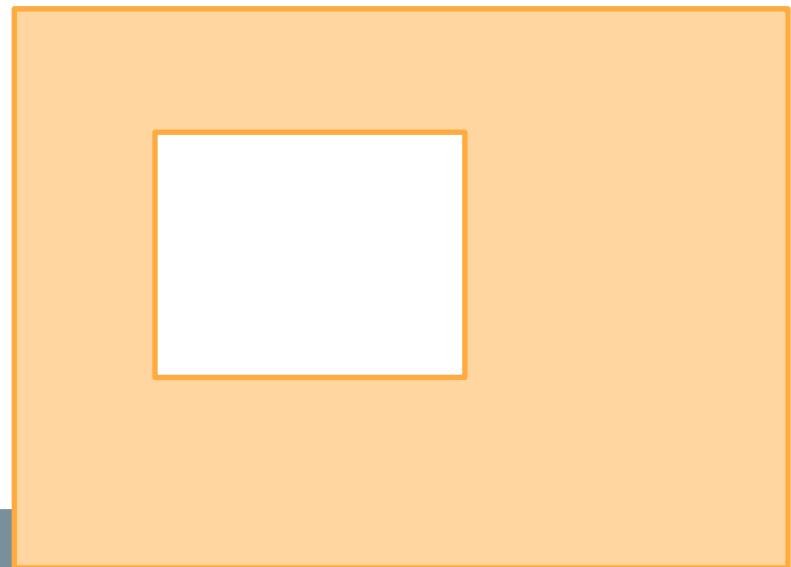
3. Locality & equational reasoning

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



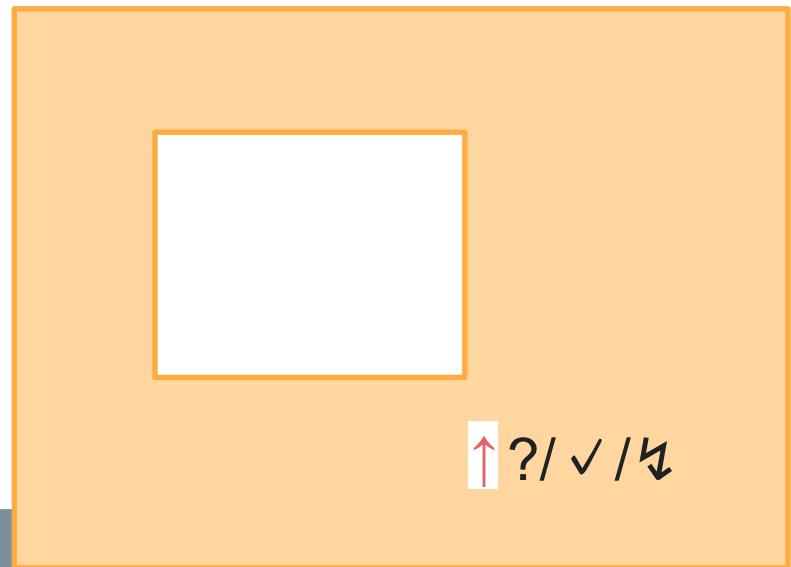
3. Locality & equational reasoning

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)

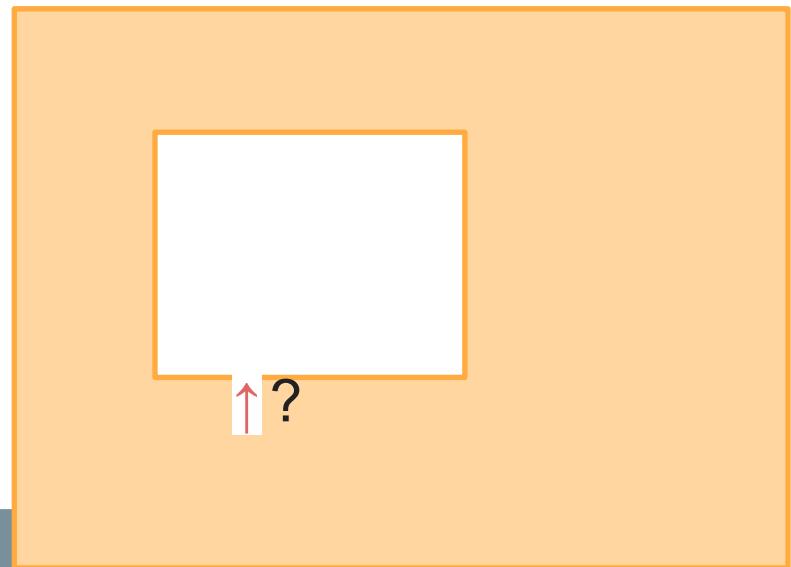
3. Locality & equational reasoning

Locality of focussed rewriting

“How does a sub-graph behave?”

Case analysis

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



(no interference)

interference

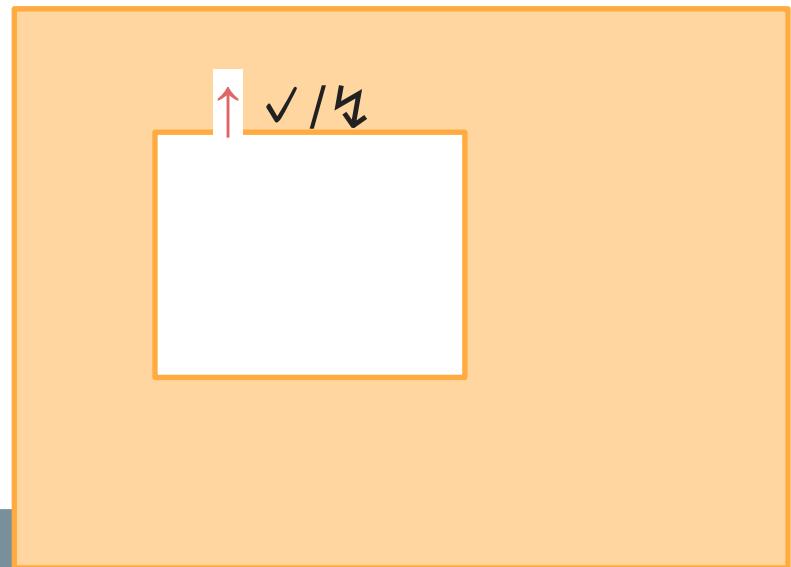
3. Locality & equational reasoning

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)

interference

interference

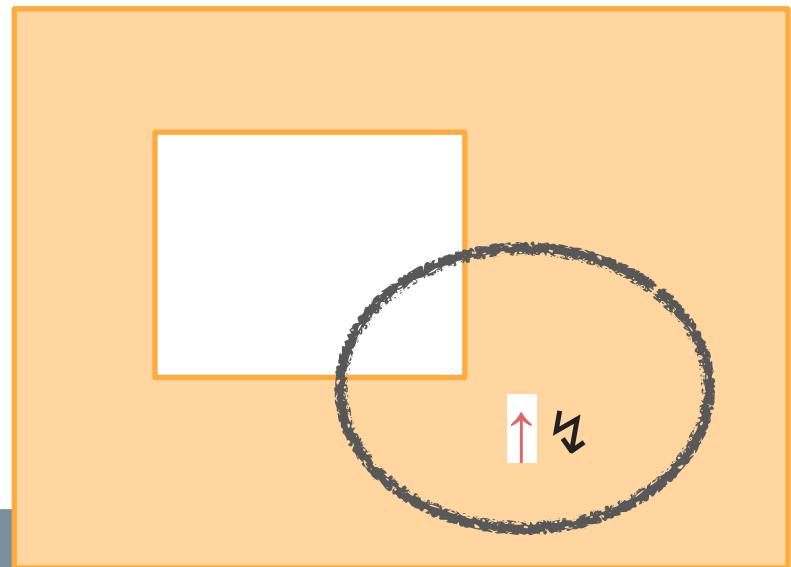
3. Locality & equational reasoning

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)

interference

interference

possible interference

3. Locality & equational reasoning

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

Case analysis

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

with the sub-graphs:

(no interference)

interference

interference

possible interference

3. Locality & equational reasoning

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

Case analysis

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

the sub-graphs are
interfered the same:

(always)

if *input-safe*

if *output-closed*

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

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

the sub-graphs are
interfered the same:

(always)

if *input-safe*

if *output-closed*

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

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

the sub-graphs are
interfered the same:

(always)

if *input-safe*

if *output-closed*

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 G0l)
- types
 - weak notion of safety
- tooling
 - <https://tnttodd.github.io/Spartan-Visualiser/>
- cost analysis
 - observational equivalence with cost improvement (*improvement theory*)
 - cost model of focussed hypernet rewriting
(cf. dynamic G0l)
- strengthening mathematics
 - hypernets = trees + hierarchy + copying + sharing
 - focussed DPO rewriting
 - reasoning with graph contexts