Local reasoning for robust observational equivalence

<u>Koko Muroya</u>

(RIMS, Kyoto U. & U. Birmingham)

Dan R. Ghica

Todd Waugh Ambridge

(U. Birmingham)

Logic and Semantics Seminar (Computer Laboratory, Cambridge), 12 September 2019

~~SYCO V & STRINGS III (Birmingham), 6 September 2019~~

Local reasoning for robust observational equivalence

Koko Muroya

(RIMS, Kyoto U. & U. Birmingham)

Dan R. Ghica

Todd Waugh Ambridge

(U. Birmingham)

MRG meeting (Imperial, London), 19 September 2019

~~Logic and Semantics Seminar (Computer Laboratory, Cambridge), 12 September 2019~~

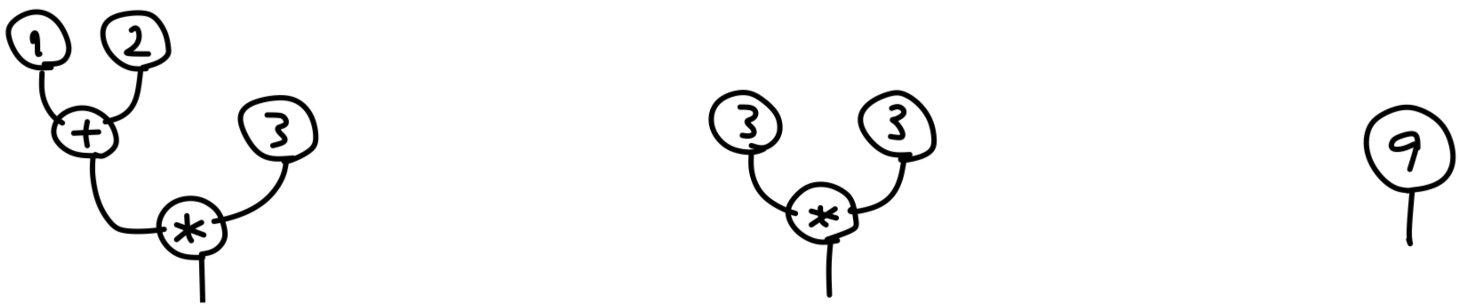
~~SYCO V & STRINGS III (Birmingham), 6 September 2019~~

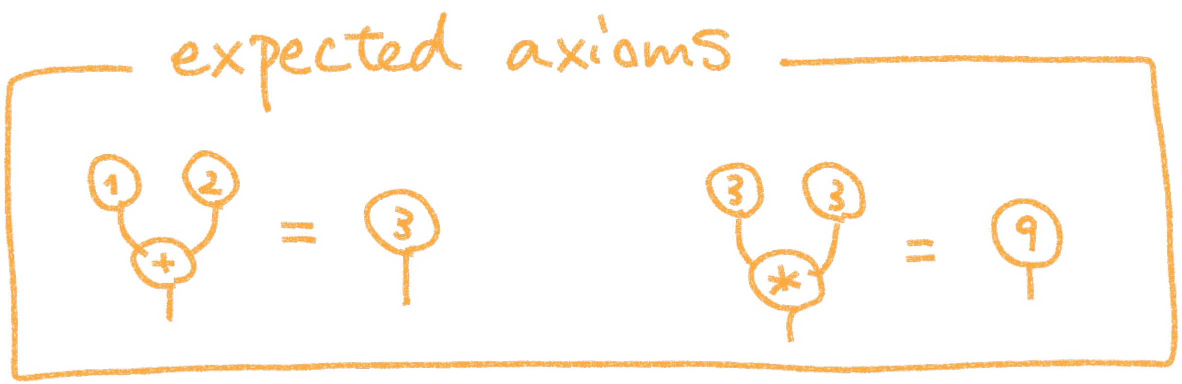
PART I

Diagrammatic modelling of program execution

2D representation of programs

(1 + 2) * 3   3 * 3   9



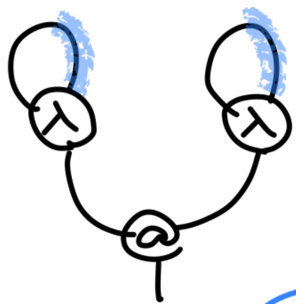


2D representation of programs

$(\lambda x.x)\ (\lambda y.y)$

$=_\alpha (\lambda z.z)\ (\lambda z.z)$

$\lambda y.y$

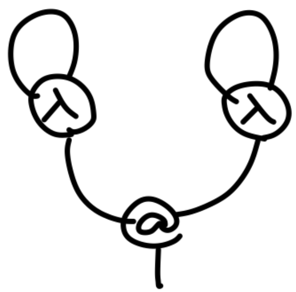


variables as wires

2D representation of programs

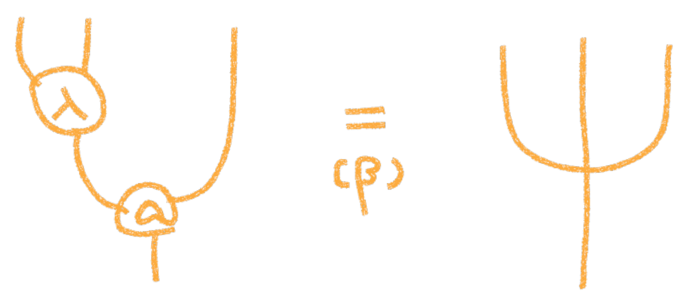
$(\lambda x.x)\ (\lambda y.y)$

$=_\alpha (\lambda z.z)\ (\lambda z.z)$



$\lambda y.y$

expected axiom

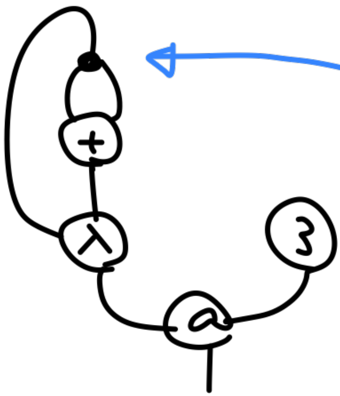


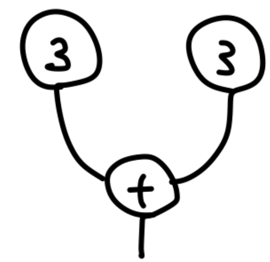
2D representation of programs

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

let $x = 3$ in $x + x$

$3 + 3$





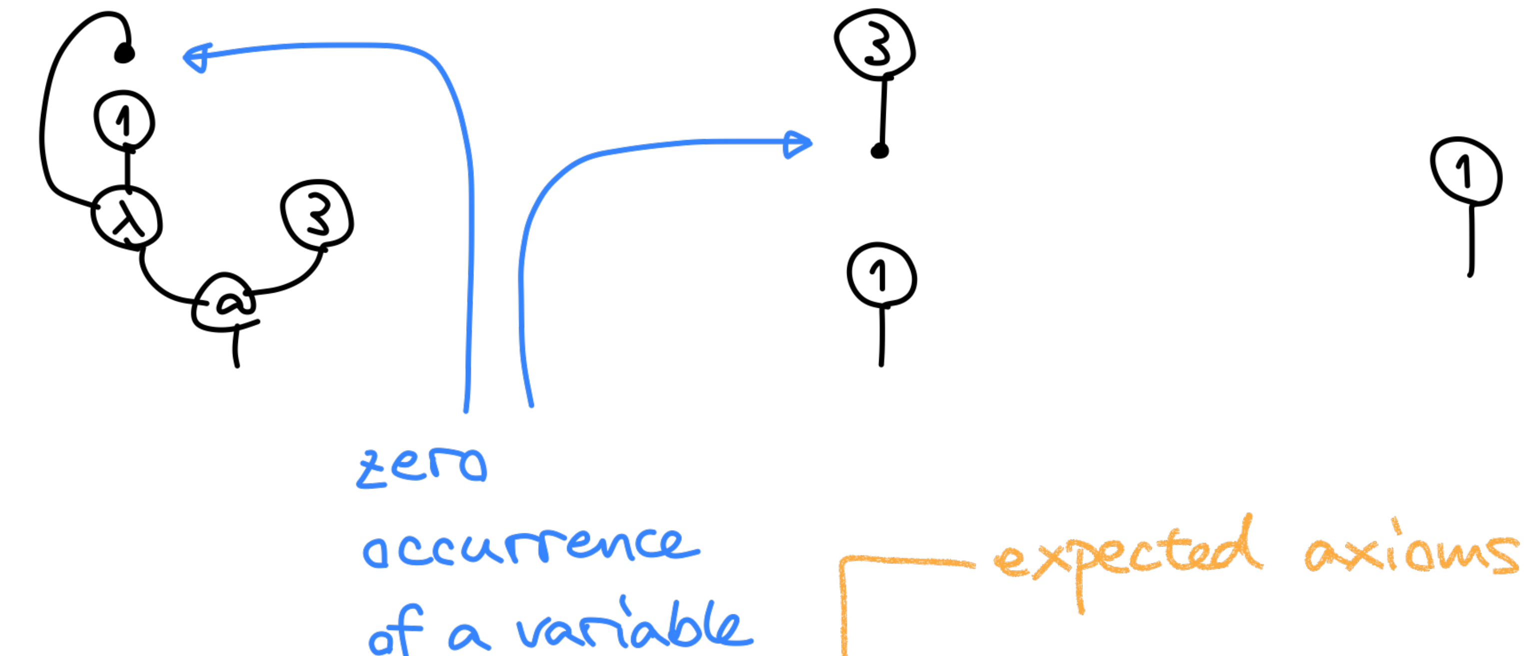
multiple occurrences of a variable

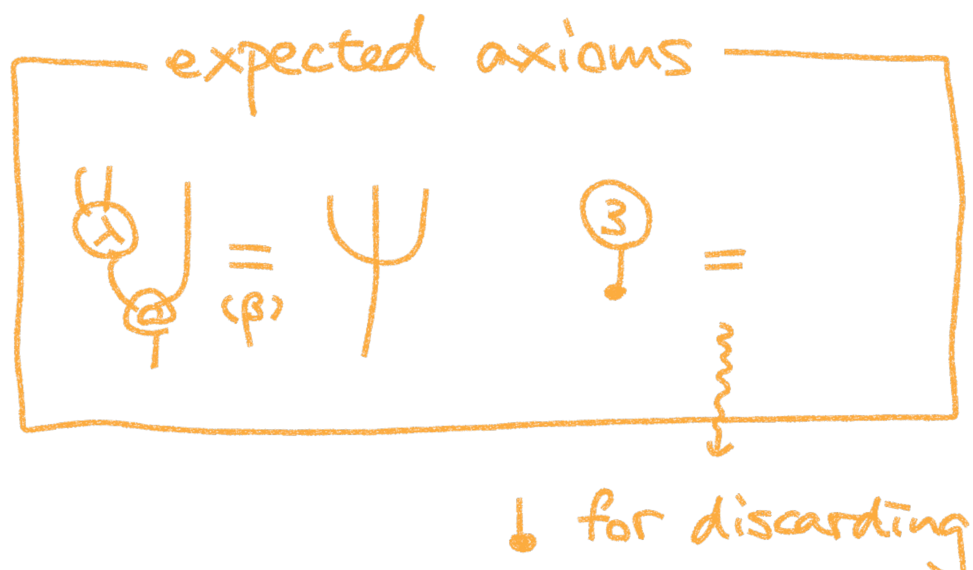
expected axioms

for copying

2D representation of programs

$(\lambda x.\ 1)\ 3$ | let $x = 3$ in 1 | 1



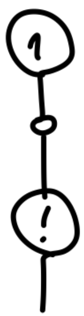


2D representation of programs

new a = 1 in !a   1

reference / location creation

dereference / read

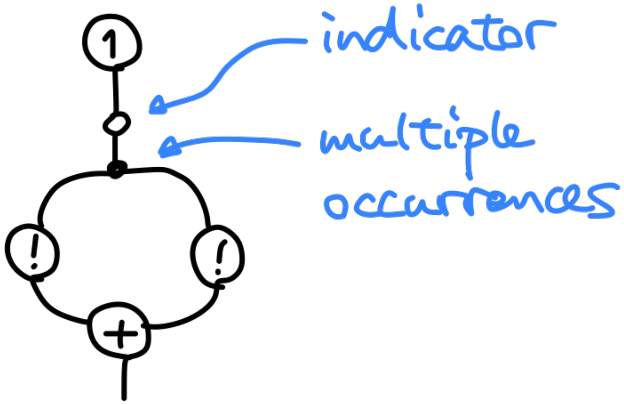


← reference/location indicator

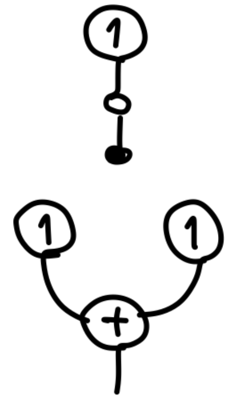
expected axiom

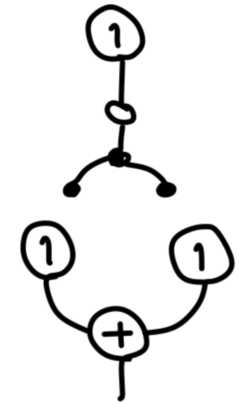
2D representation of programs

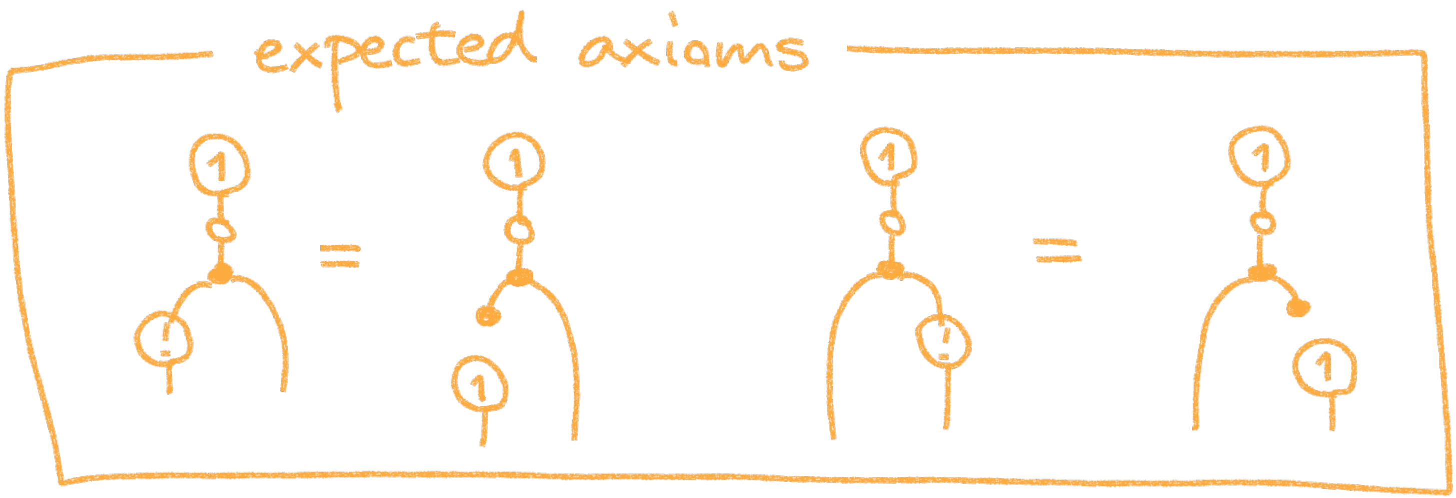
new a = 1 in !a + !a



new a=1 in 1+1

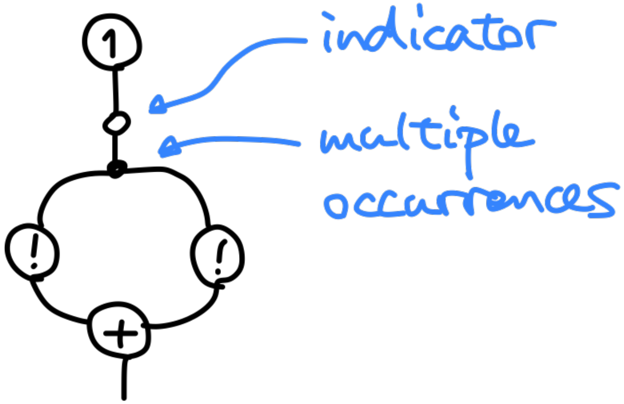




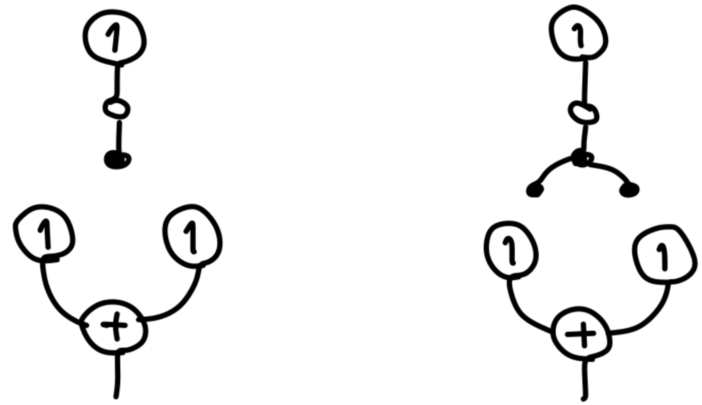


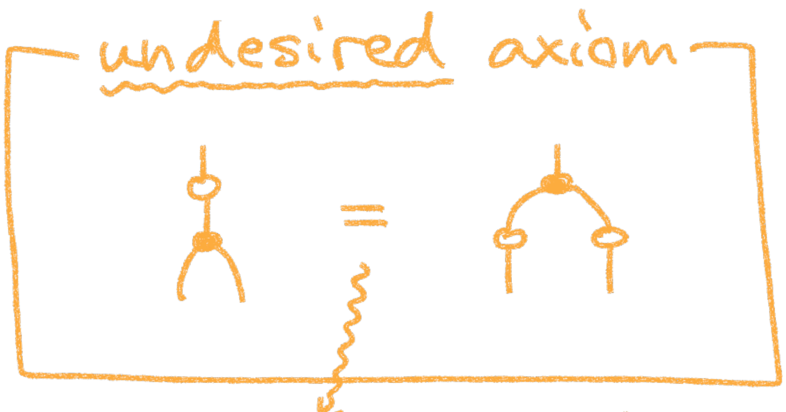
2D representation of programs

new a = 1 in !a + !a



new a = 1 in 1 + 1



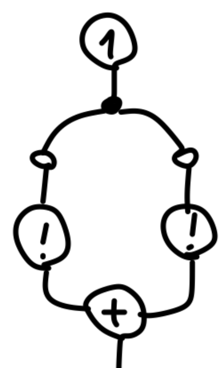


location indicator 

blocks copying 

let $x = 1$ in

(new a = x in !a) + (new a = x in !a)



2D representation of programs

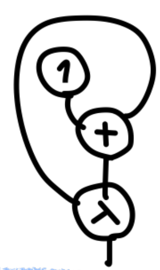
- name-free (α-equivalence built in)

$\lambda x.x$
$=_\alpha \lambda y.y$

- more refined & less structured than 1D syntax

diagrams with no term counterpart e.g. 

$\lambda x. 1 + x$



$\text{let } w = 1 \text{ in } \lambda x. w + x$

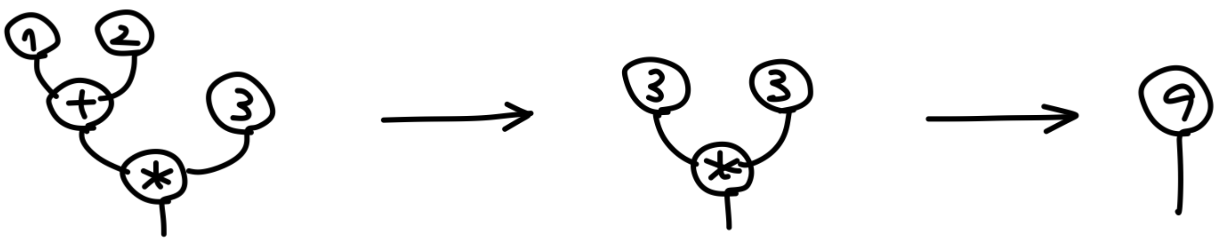
desired feature of a diagrammatic language

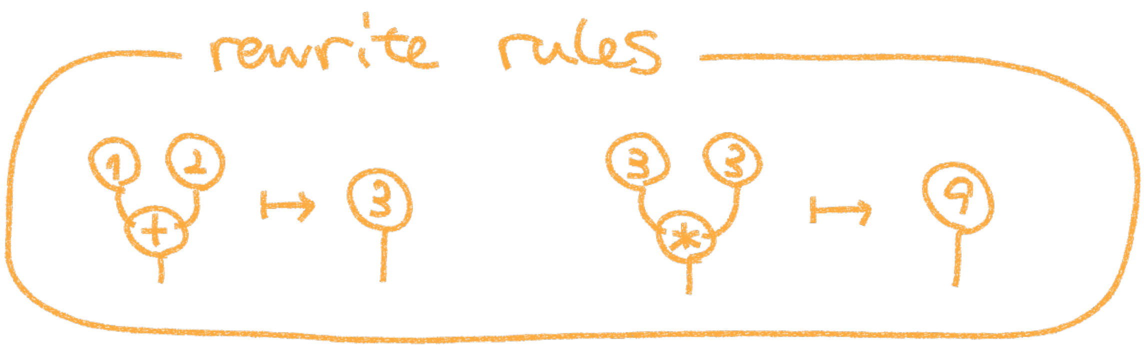
- copying vs. sharing

2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

$(1+2)*3 \longrightarrow 3*3 \longrightarrow 9$

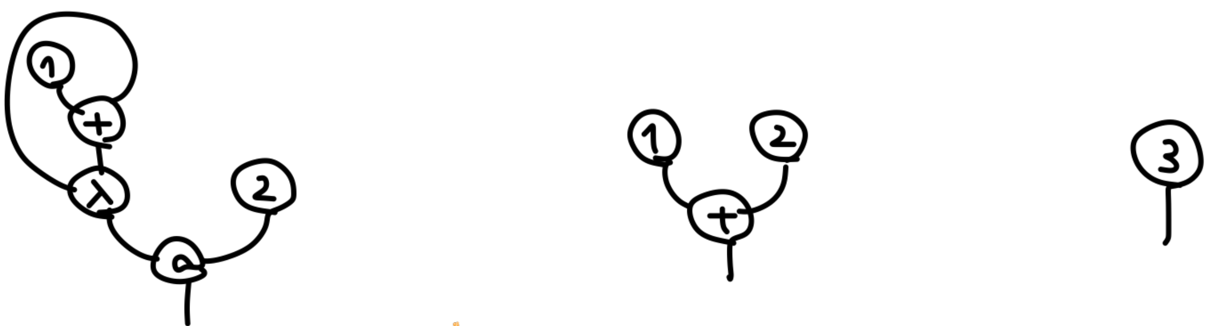


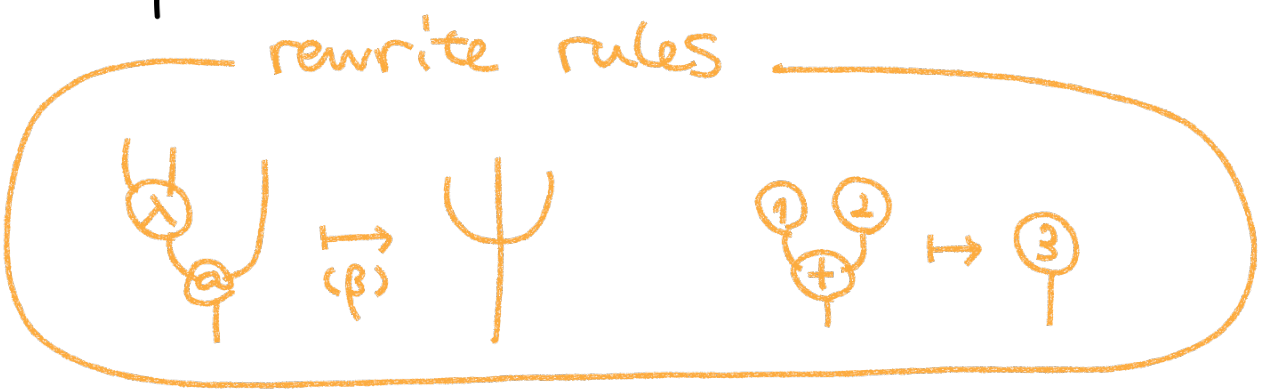


2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

$$(\lambda x.\ 1 + x)\ 2 \longrightarrow 1 + 2 \longrightarrow 3$$

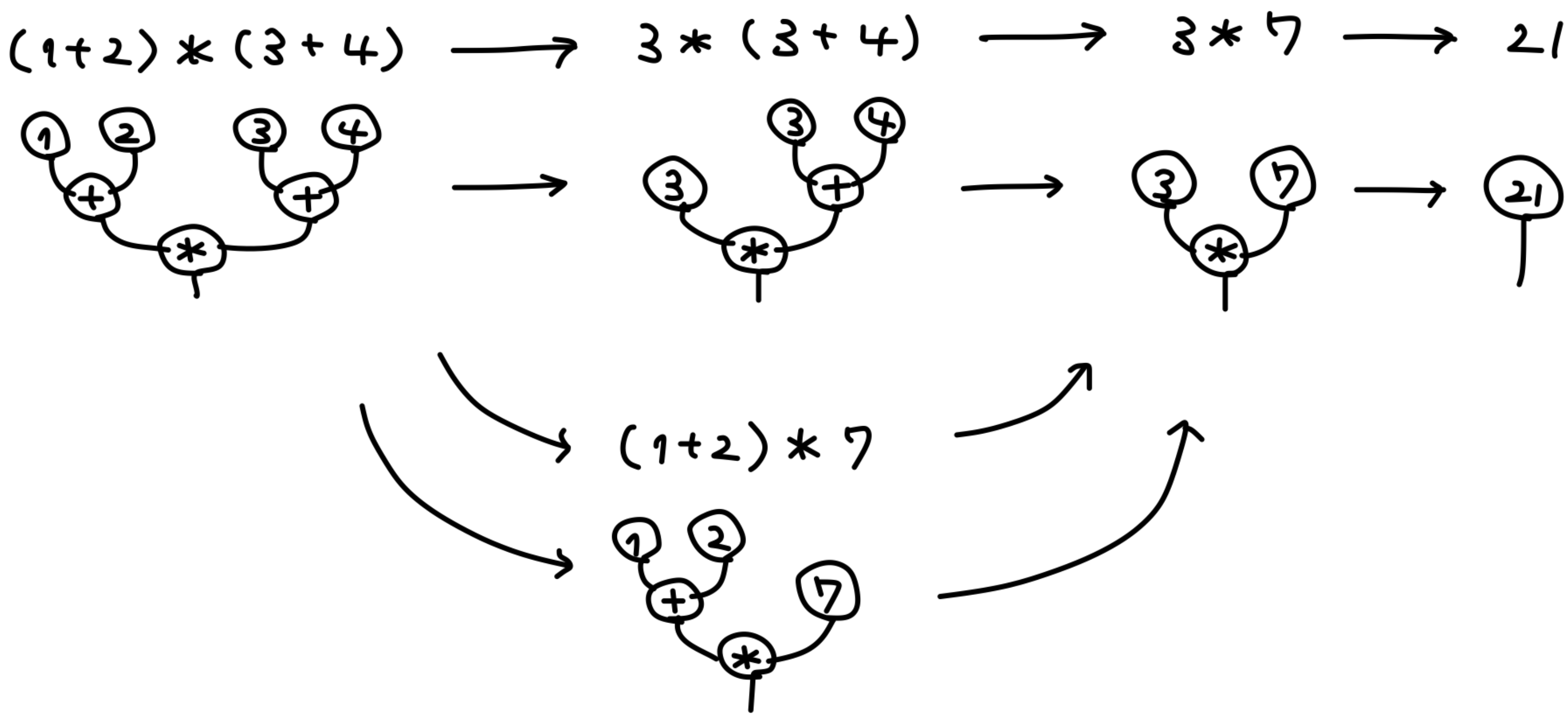




2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of redex search

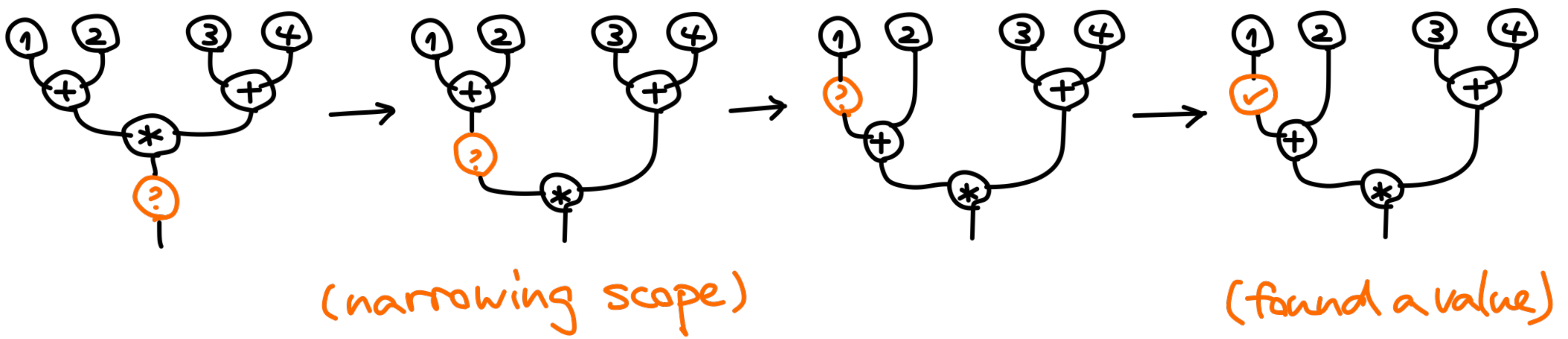


2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of redex search specified by token

$(1+2)*(3+4) \longrightarrow 3*(3+4) \longrightarrow 3*7 \longrightarrow 21$

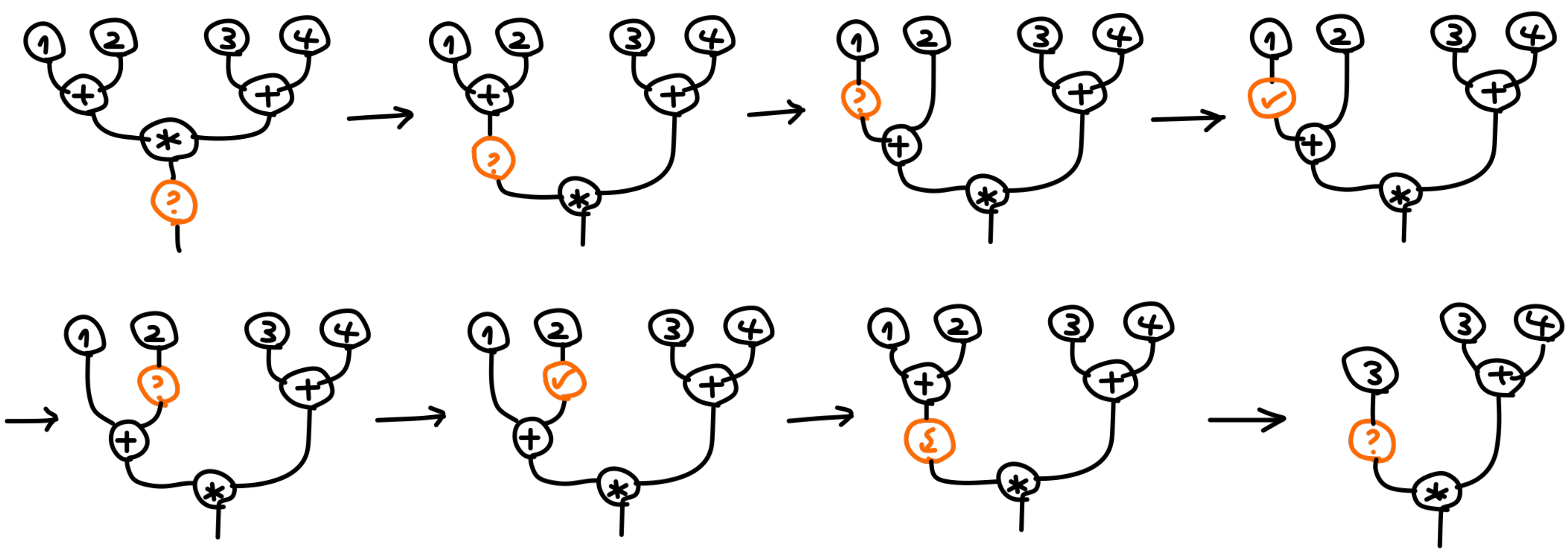


2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of redex search specified by token

$$(1+2)*(3+4) \longrightarrow 3*(3+4) \longrightarrow 3*7 \longrightarrow 21$$



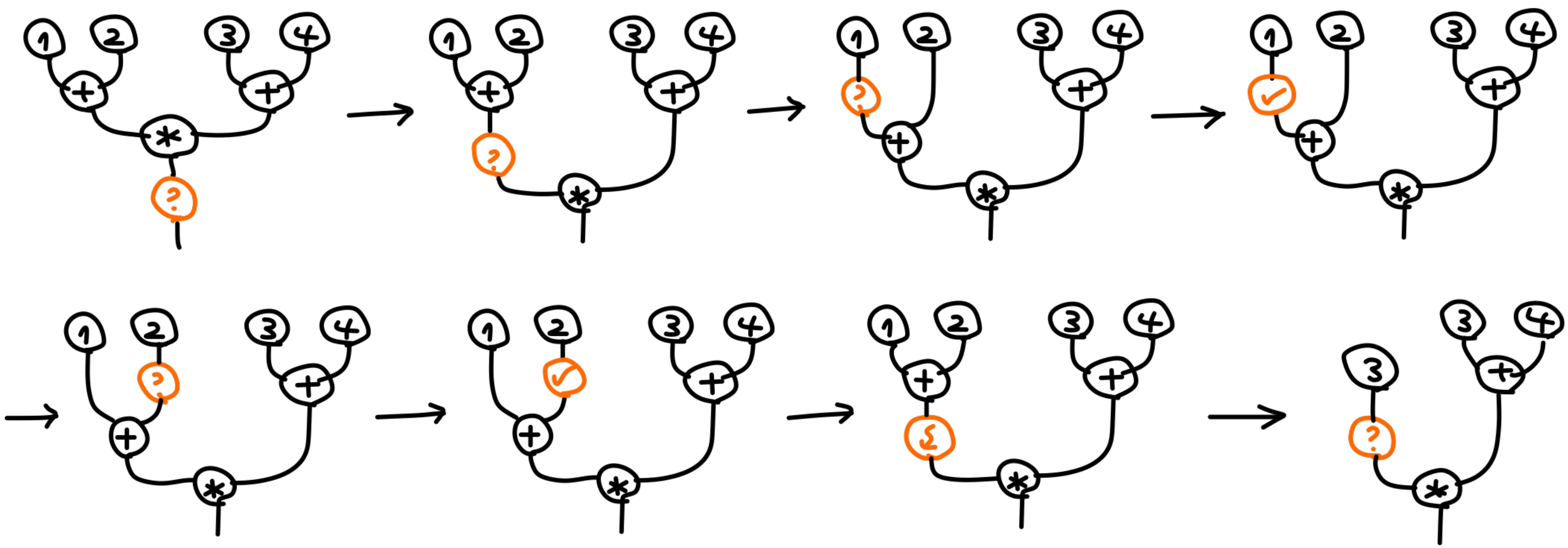
(found a redex)

2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of redex search specified by token

redex search is also rewriting



(found a redex)

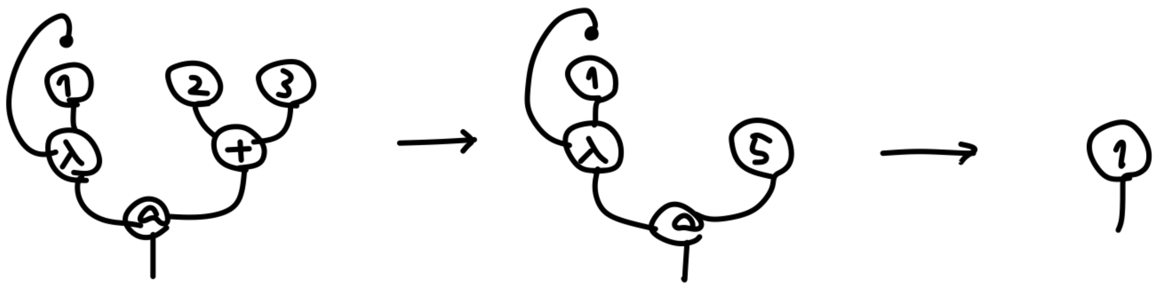
2D modelling of program execution

modelling dynamic (operational) behaviour with <u>strategical diagram-rewriting</u>

- ▷ strategy of redex search specified by <u>token</u>

  redex search is also rewriting

$$(\lambda x.1)\ (2+3) \longrightarrow (\lambda x.1)\ 5 \longrightarrow 1$$



(call-by-value)

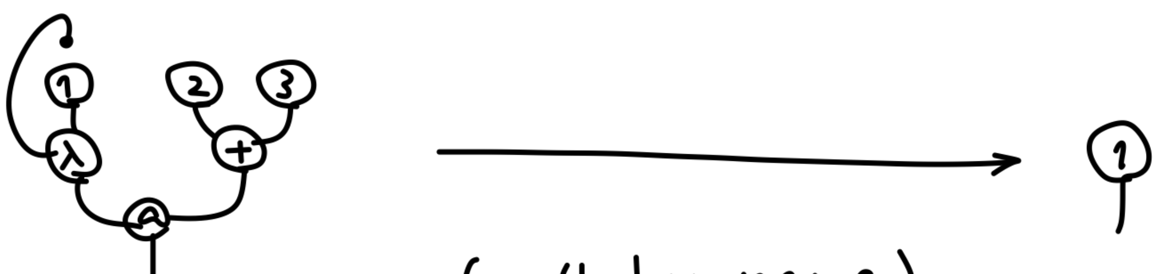
2D modelling of program execution

modelling dynamic (operational) behaviour
with strategical diagram-rewriting

- strategy of redex search specified by token

  (redex search is also rewriting)

$(\lambda x. 1)(2+3) \longrightarrow 1$



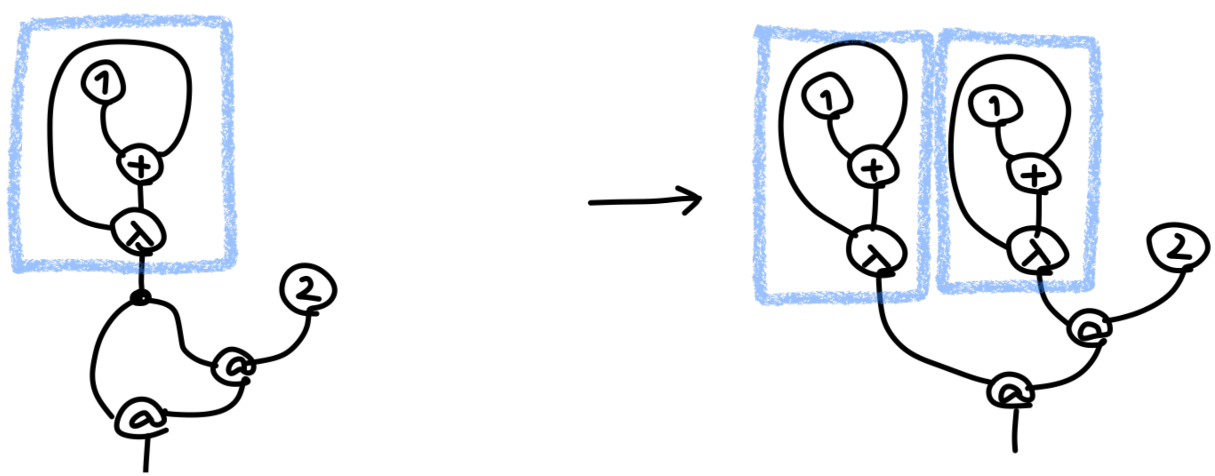
(call-by-name)

2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of duplication

$\text{let } u = \lambda x. 1 + x \text{ in } u\,(u\;2) \longrightarrow (\lambda x. 1 + x)\,((\lambda x. 1 + x)\;2)$

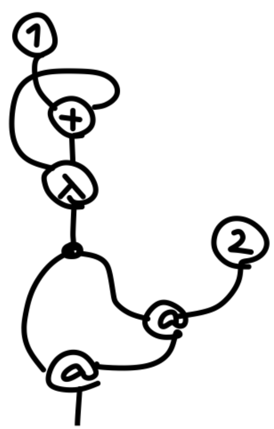


2D modelling of program execution

modelling dynamic (operational) behaviour with <u>strategical diagram-rewriting</u>

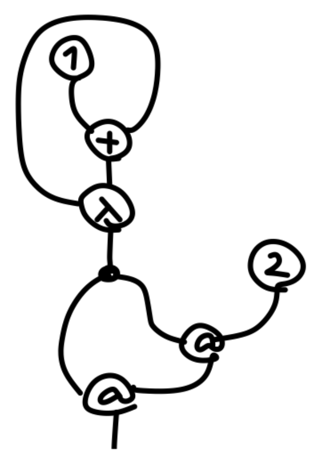
- ▹ strategy of duplication

$\text{let } u = (\text{let } w = 1 \text{ in } \lambda x.\, w + x) \text{ in } u\ (u\ 2)$



[cf.

$\text{let } u = \lambda x.\, 1 + x \text{ in } u\ (u\ 2)$



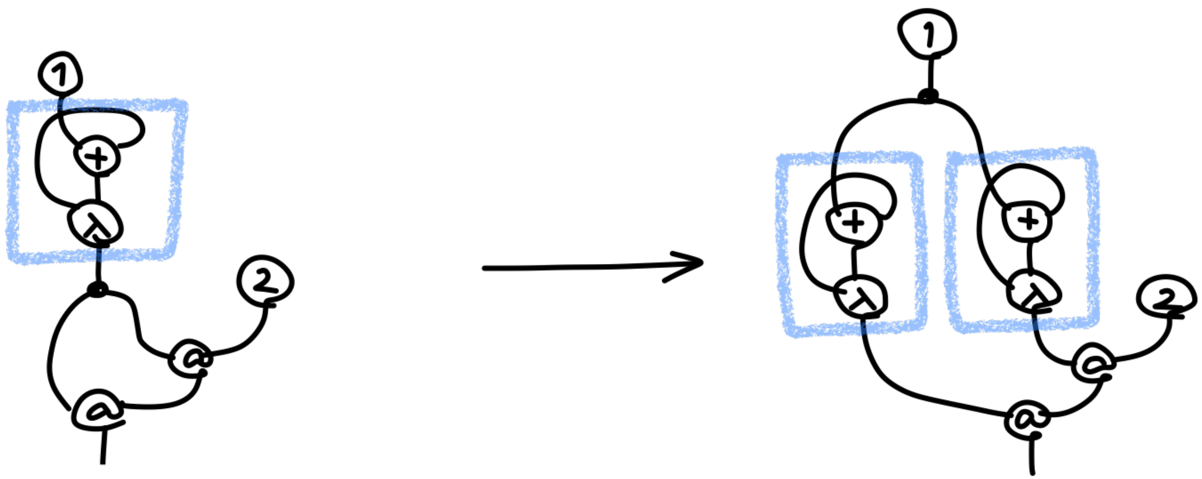
]

2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of duplication

let u = (let w = 1 in λx. w + x) in u (u 2)



let w = 1 in
let u = λx. w + x in u (u 2) → let w = 1 in
(λx. w + x) ((λx. w + x) 2)

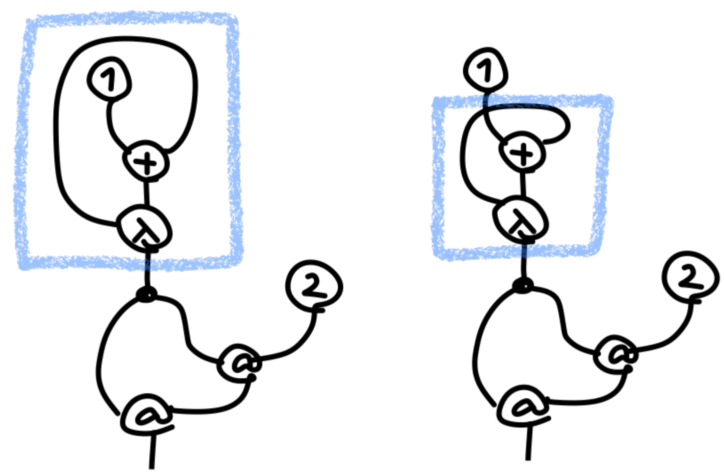
2D modelling of program execution

modelling dynamic (operational) behaviour
with strategical diagram-rewriting

- ▹ strategy of duplication
  specified by unit blocks of duplication

equip diagrams with
a block / box structure

(graph-theoretically:
nodes labelled with
a graph)

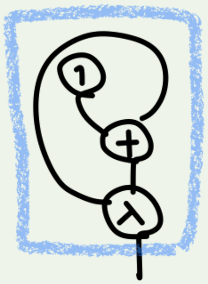


2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of duplication specified by unit blocks of deferral

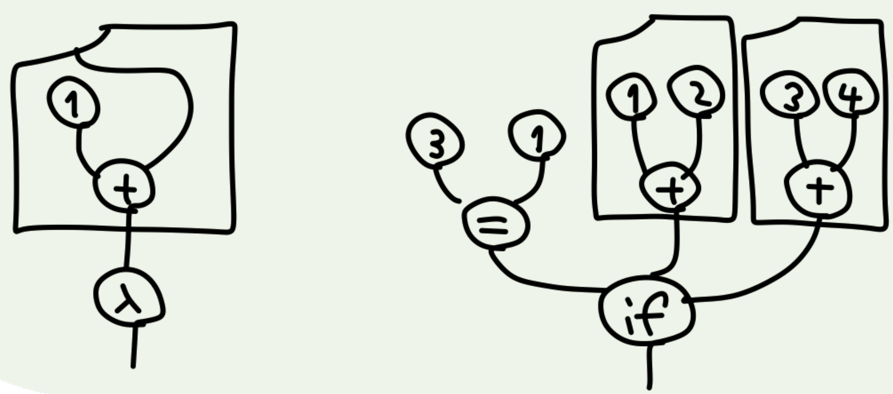
unit of duplication



⇝ refinement

unit of deferral

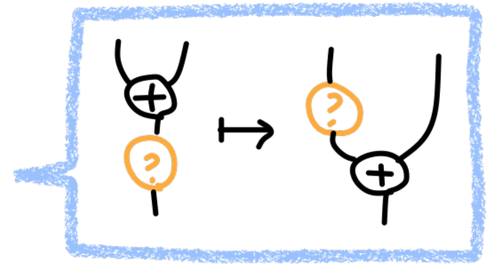
$\lambda x.\ 1 + x$ if $3 = 1$ then $1 + 2$ else $3 + 4$



2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of redex search: specified by rewriting with token



- strategy of duplication: specified by unit blocks of duplication/deferral

desired feature of a diagrammatic language

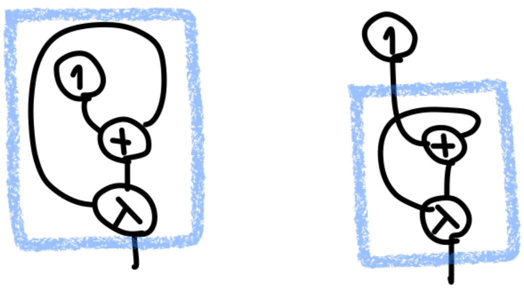
- block/box structure

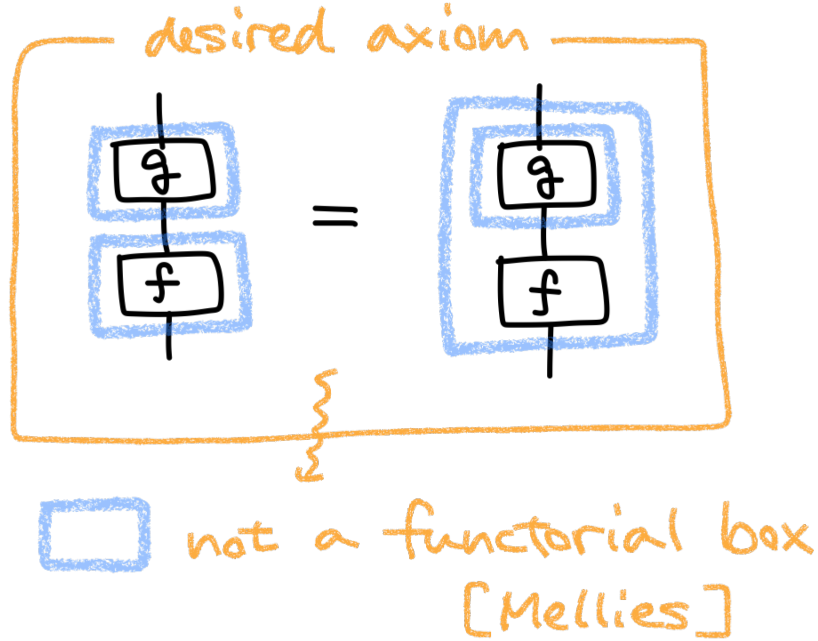
2D modelling of program execution

modelling dynamic (operational) behaviour
with strategical diagram-rewriting

desired feature of a diagrammatic language

- block/box structure





2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

... but, modelling for what?

an answer: proving that two program fragments have the same behaviour

observational equivalence

PART II

Local reasoning for robust observational equivalence

Proving observational equivalence

exercise prove that 'new $a = 1$ in $\lambda x. !a$' and '$\lambda x. 1$' have the same (dynamic) behaviour in any possible programs

trial with terms

$$\text{let } u = (\text{new } a = 1 \text{ in } \lambda x. !a) \text{ in } (u\ 0) + (u\ 0)$$

$$\rightarrow \text{new } a = 1 \text{ in } ((\lambda x. !a)\ 0) + ((\lambda x. !a)\ 0)$$

$$\text{let } u = \lambda x. 1 \text{ in } (u\ 0) + (u\ 0)$$

$$\rightarrow ((\lambda x. 1)\ 0) + ((\lambda x. 1)\ 0)$$

tracing non sub-terms

Proving observational equivalence

exercise prove that 'new a=1 in λx. !a' and 'λx. 1' have the same (dynamic) behaviour in any possible programs

trial with diagrams

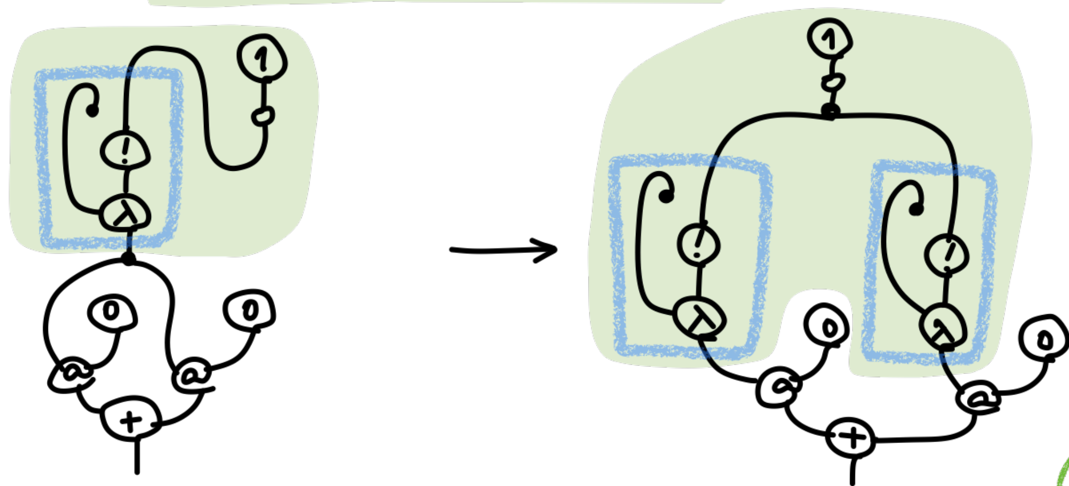
let u = (new a=1 in λx. !a) in (u 0) + (u 0)

let u = λx. 1 in (u 0) + (u 0)

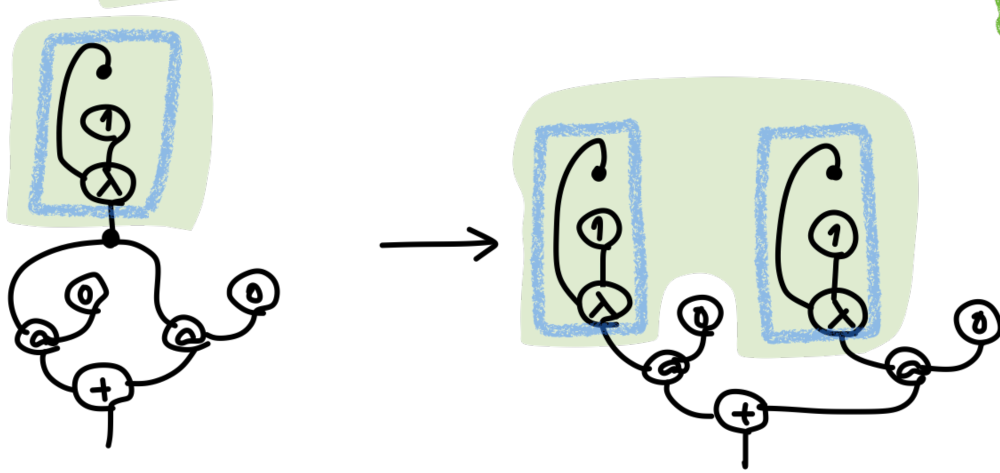
Proving observational equivalence

trial with diagrams

let u = (new a=1 in λx.!a) in (u 0) + (u 0)



let u = λx. 1 in (u 0) + (u 0)



tracing sub-diagrams

modelling dynamic (operational) behaviour with strategical diagram-rewriting

⇝

proving observational equivalence

observations

- proof possible by tracing sub-diagrams
- apparent generality of the proof methodology

modelling dynamic (operational) behaviour with strategical diagram-rewriting

⇝

proving observational equivalence

observations

- proof possible by tracing sub-diagrams
- apparent generality of the proof methodology

⇧

case study: deterministic & sequential computation

SPARTAN, the target calculus

should accommodate
as much language features as possible
in a uniform way

as extrinsics!

SPARTAN, the target calculus

programming
= copying
+ sharing
+ thunking
+ algebra

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

SPARTAN, the target calculus

programming
= copying
+ sharing
+ thunking
+ algebra

$t ::=$

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

variables — referenced computation

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

names/atoms/locations — stored computation

$\mid x.\, t$

$\mid \varphi(\vec{t}\,;\vec{t}\,)$

SPARTAN, the target calculus

programming
= copying
+ sharing
+ thunking
+ algebra

$t ::=$

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

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

$\mid x.\, t$ — deferred computation with a bound variable

$\mid \varphi(\vec{t}\,;\, \vec{t}\,)$

SPARTAN, the target calculus

programming
= copying
+ sharing
+ thunking
+ algebra

$t ::=$

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

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

$\mid x.\, t$

$\mid \varphi(\vec{t}\,;\vec{t}\,)$

extrinsic operation with eager arguments & deferred arguments

SPARTAN, the target calculus

programming
= copying
+ sharing
+ thunking
+ algebra

$0(-;-)$, $1(-;-)$, …
$\mathrm{PLUS}(t,u;-)$
$\mathrm{IF}(t;u_1,u_2)$
$\mathrm{LAMBDA}(-;x.t)$
$\mathrm{APP}(t,u;-)$
$\mathrm{LOOKUP}(t;x.u)$
$\mathrm{DEREF}(t)$
$\mathrm{ASSIGN}(t,u)$

examples

extrinsic operation with eager arguments & deferred arguments

$\mid \varphi(\vec{t};\vec{t})$

SPARTAN, the target calculus

programming
= copying
+ sharing
+ thunking
+ algebra

$t ::=$

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

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

$\mid x.\, t$

$\mid \varphi(\vec{t}\,;\vec{t}\,)$

extrinsic operation with eager arguments & deferred arguments

Proving observational / contextual equivalence

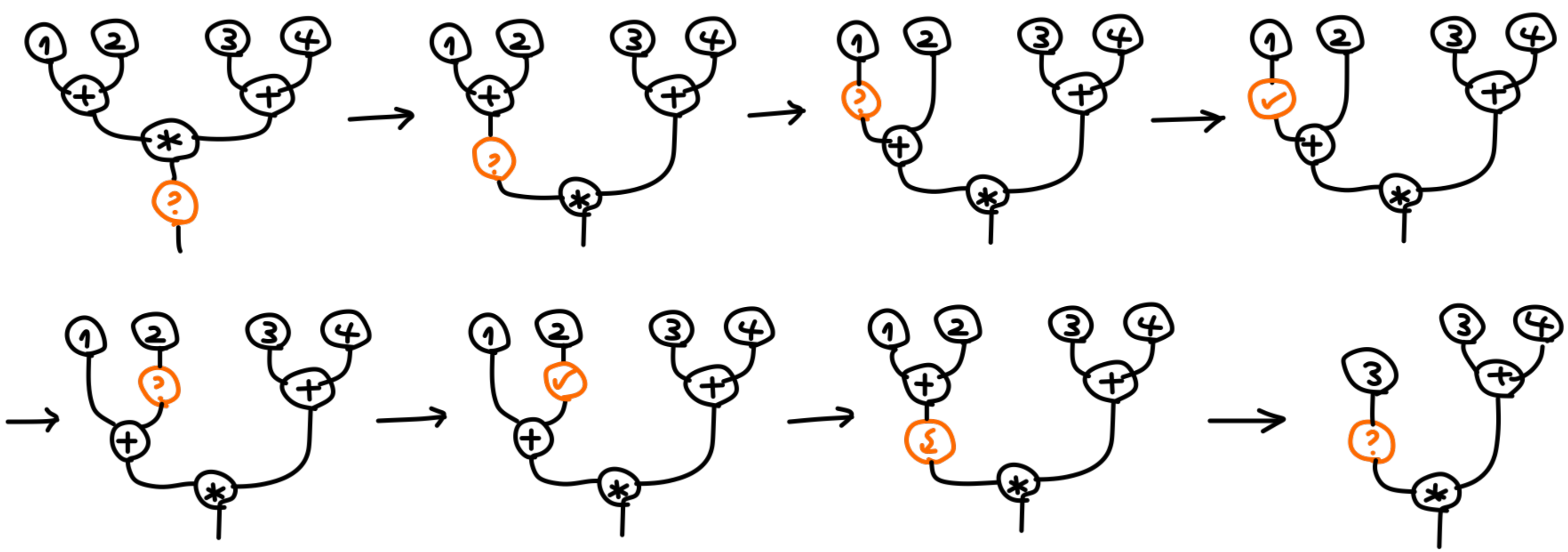
Proving observational / contextual equivalence

recall...

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of redex search specified by token

$(1+2) * (3+4) \longrightarrow 3 * (3+4) \longrightarrow 3 * 7 \longrightarrow 21$



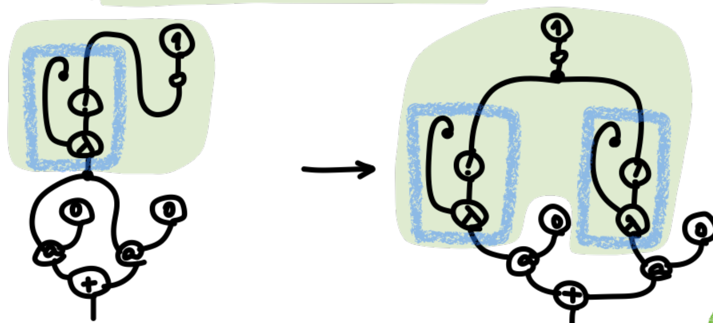
Proving observational / contextual equivalence

recall...

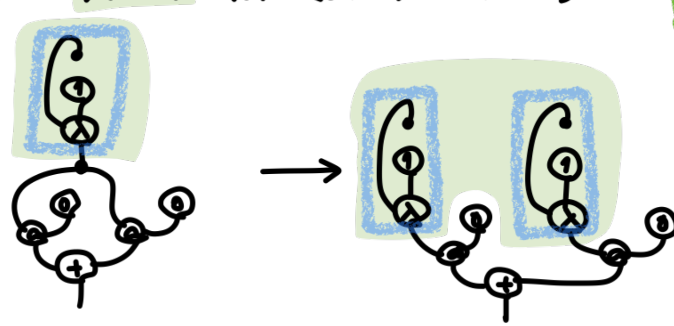
exercise prove that 'new a=1 in λx. !a' and 'λx. 1' have the same (dynamic) behaviour in any possible programs

trial with diagrams

let u = (new a=1 in λx. !a) in (u 0) + (u 0)



let u = λx. 1 in (u 0) + (u 0)



tracing sub-diagrams

Proving observational / contextual equivalence

goal: prove (generalised) contextual refinement $G \preceq^{\mathcal{C}}_{Q} H$ on diagrams G, H

a class of (diagrammatic) contexts

$$G \preceq^{\mathcal{C}}_{Q} H \overset{\Delta}{\iff} \forall C \in \mathcal{C}.\ \forall k \in \mathbb{N}.\ C[G] \Downarrow_k \Rightarrow \left(\exists \ell \in \mathbb{N}.\ C[H] \Downarrow_\ell \wedge k \, Q \, \ell \right)$$

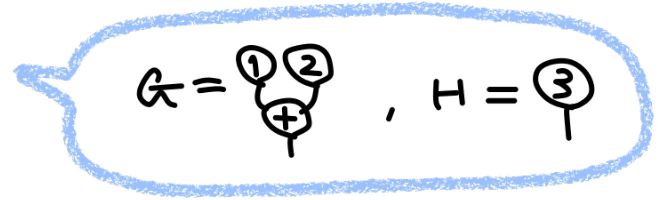
a preorder on nat. numbers

$\mathbb{N} \times \mathbb{N}$, $=$, $\geq$, ...

Proving observational / contextual equivalence

<u>goal</u> prove (generalised) contextual refinement $G \leq^{\mathcal{C}}_{Q} H$

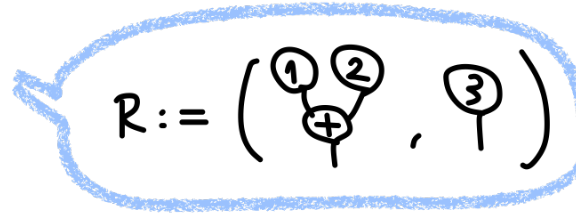
on diagrams G, H



Proving observational / contextual equivalence

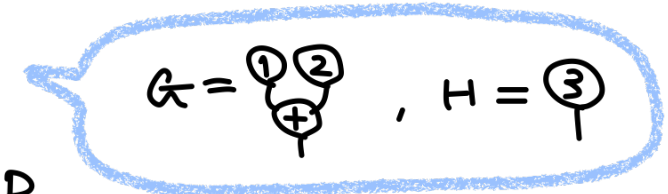
goal prove (generalised) contextual refinement $G \leq^{\varepsilon}_{Q} H$ on diagrams G, H

step 1 construct a binary relation R on diagrams, s.t. G R H

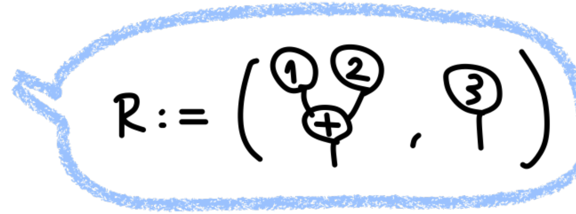


Proving observational / contextual equivalence

goal prove (generalised) contextual refinement $G \leq^{\mathcal{C}}_{Q} H$ on diagrams G, H

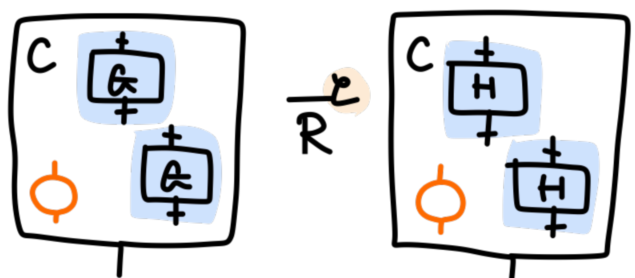


step 1 construct a binary relation R on diagrams, s.t. $G \; R \; H$

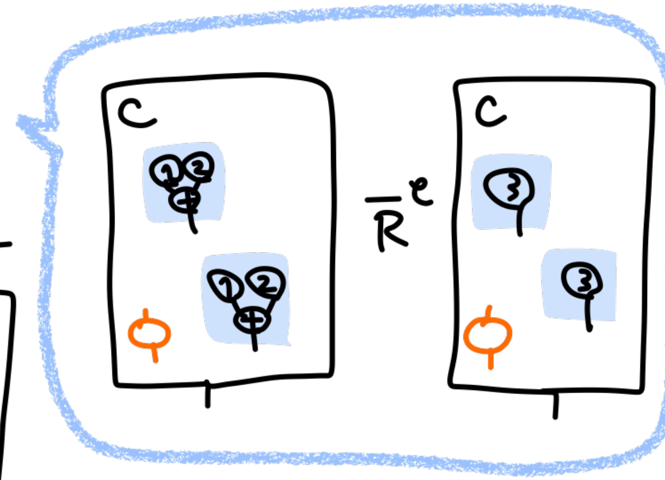


step 2 take the contextual & focussed closure $\overline{R}^{\mathcal{C}}$ of R,

namely: $\dfrac{G \; R \; H \qquad C \in \mathcal{C}}{}$

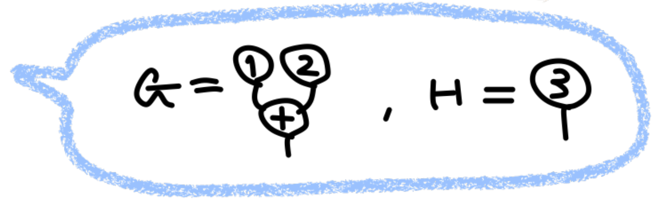


where

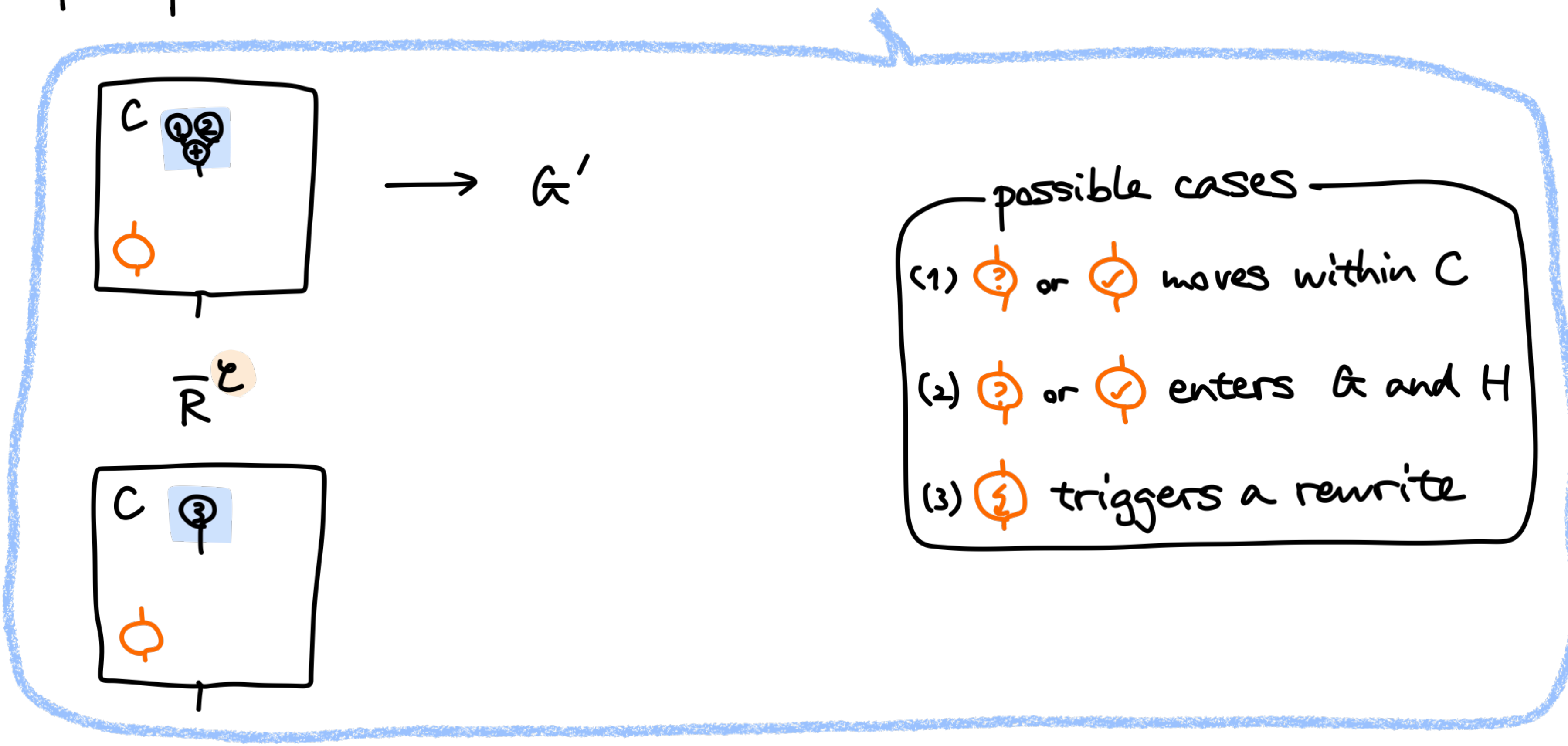


Proving observational / contextual equivalence

goal prove (generalised) contextual refinement $G \leq^{\mathcal{C}}_{Q} H$ on diagrams G, H



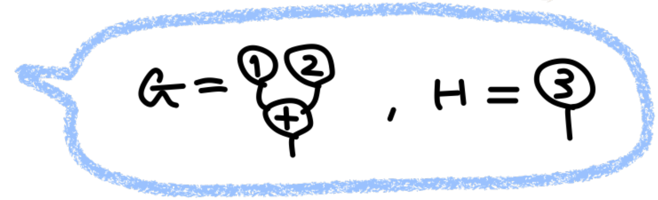
step 3 prove that $\overline{R}^{\mathcal{C}}$ is a "Q-weak" simulation



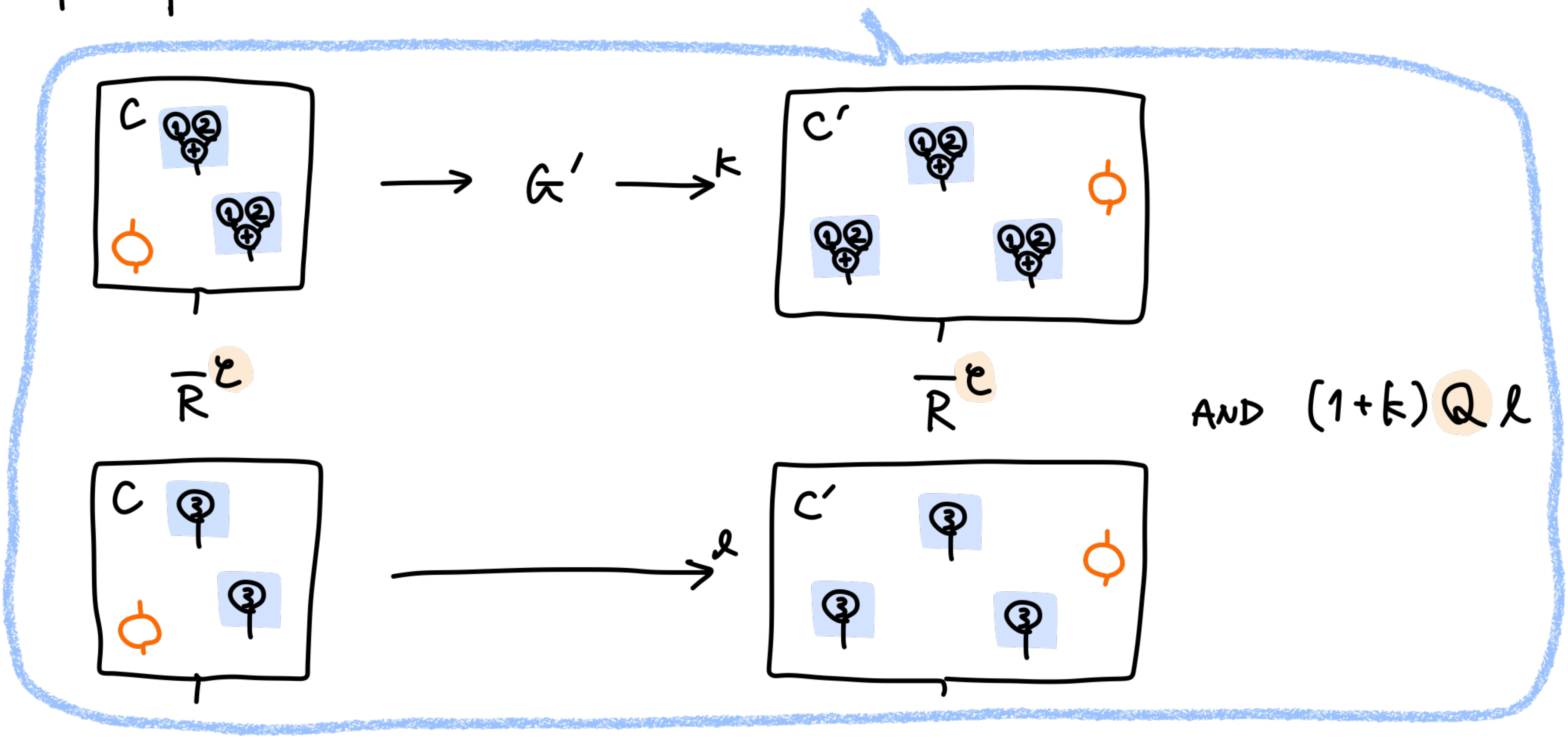
Proving observational / contextual equivalence

goal prove (generalised) contextual refinement $G \leq^{\mathcal{C}}_{Q} H$

on diagrams G, H



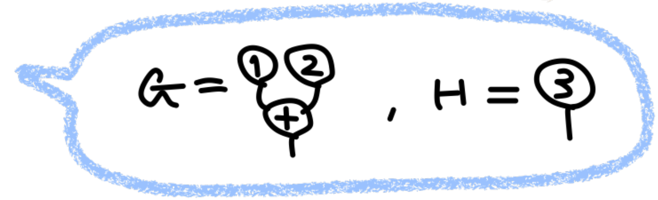
step 3 prove that $\overline{R}^{\mathcal{C}}$ is a "Q-weak" simulation



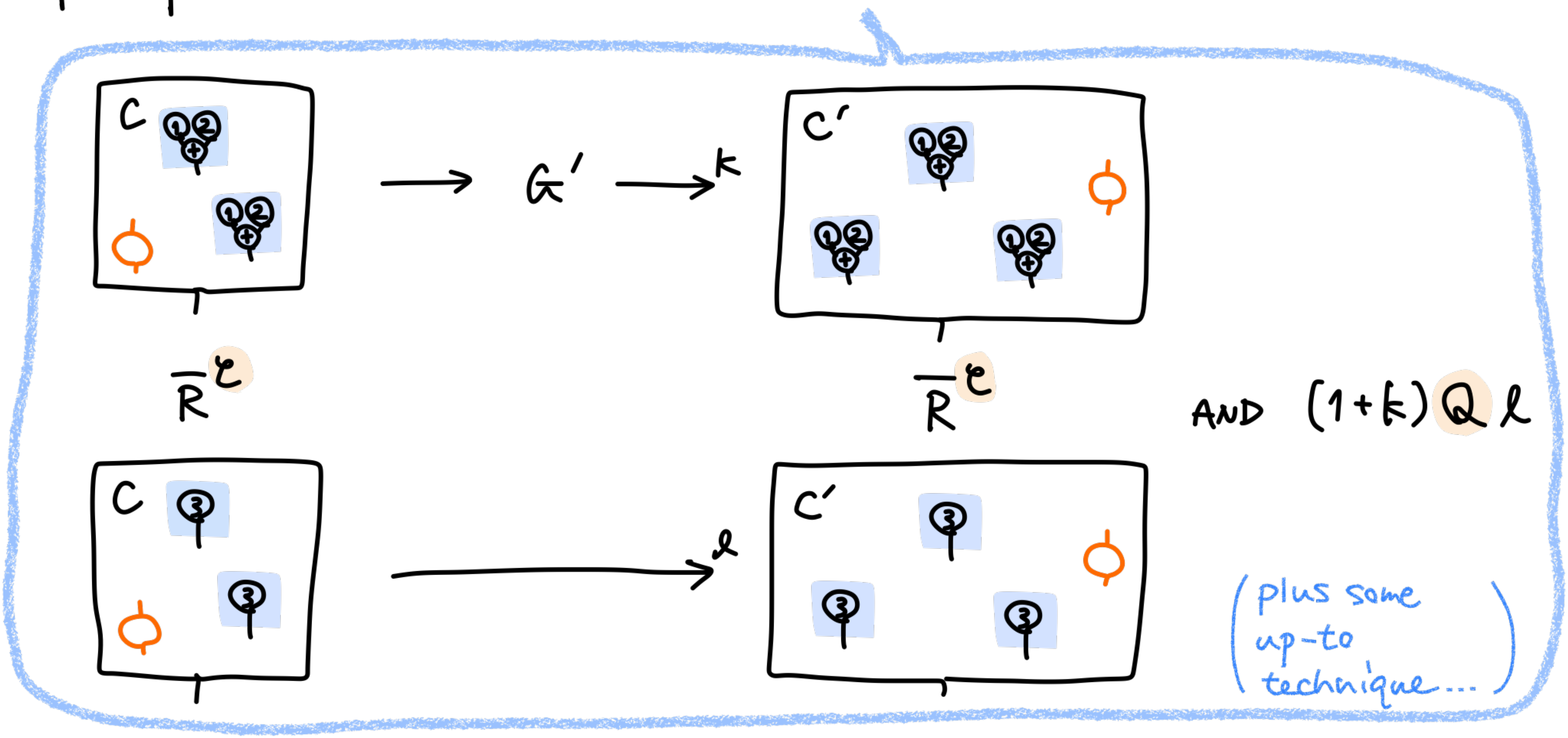
Proving observational / contextual equivalence

<u>goal</u> prove (generalised) contextual refinement $G \leq^{\varepsilon}_{Q} H$

on diagrams G, H



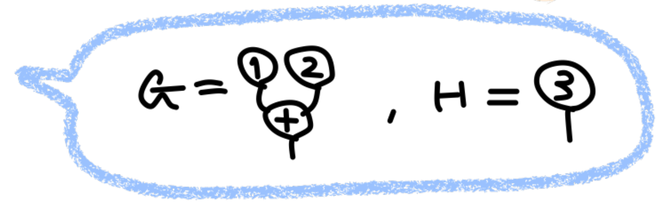
<u>step 3</u> prove that $\overline{R}^{\varepsilon}$ is a "Q-weak" simulation



Proving observational / contextual equivalence

<u>goal</u> prove (generalised) contextual refinement $G \leq^{\mathcal{C}}_{Q} H$

on diagrams G, H

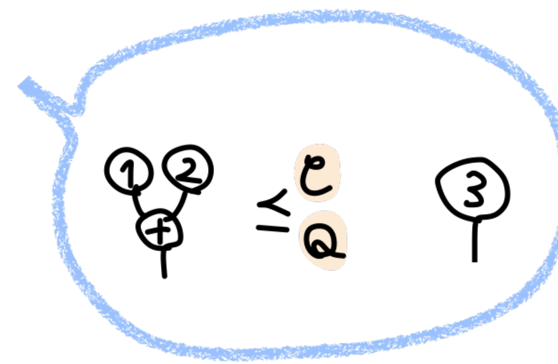


<u>step 3</u> prove that $\overline{R}^{\mathcal{C}}$ is a "Q-weak" simulation

Theorem

$\overline{R}^{\mathcal{C}}$ is a "Q-weak" simulation

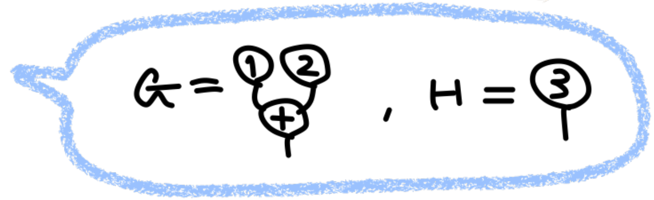
$\Rightarrow$ R implies $\leq^{\mathcal{C}}_{Q}$.



Proving observational / contextual equivalence

<u>goal</u> prove (generalised) contextual refinement $G \leq^{\mathcal{C}}_{Q} H$

on diagrams G, H

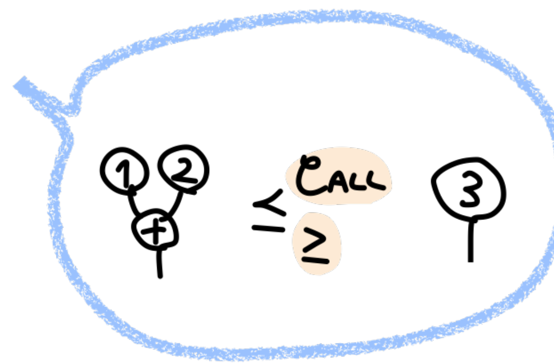


<u>step 3</u> prove that $\overline{R}^{\mathcal{C}}$ is a "Q-weak" simulation

Theorem

$\overline{R}^{\mathcal{C}}$ is a "Q-weak" simulation

$\Rightarrow$ R implies $\leq^{\mathcal{C}}_{Q}$.



modelling dynamic (operational) behaviour with strategical diagram-rewriting

⇝

proving observational equivalence ← (generalised) contextual equivalence

observations

- proof possible by tracing sub-diagrams
- apparent generality of the proof methodology

↑

case study: deterministic & sequential computation ← SPARTAN calculus

modelling dynamic (operational) behaviour
with strategical diagram-rewriting

$\rightsquigarrow$

proving observational equivalence

observations

- proof possible by tracing sub-diagrams
- apparent generality of the proof methodology

  ⇒ analysis of
  robustness of observational equivalences ?

Materials

working draft

https://arxiv.org/abs/1907.01257

on-line visualiser of diagrammatic execution

https://tnttodda.github.io/Spartan-Visualiser/