Local reasoning for robust observational equivalence

Tako Muroya
(RIMS, Kyoto U. & U. Birmingham)

Dan R. Ghica
Todd Waugh Ambridge
(U. Birmingham)

Logic and Semantics Seminar
(Computer Laboratory, Cambridge), 12 September 2019
SYED & STRINGS III (Birmingham), 6 September 2019
Local reasoning for robust observational equivalence

Toko 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 and STRINGS III (Birmingham), 6 September 2019
Diagrammatic modelling of program execution
2D representation of programs

\[(1 + 2) \times 3\]

\[3 \times 3\]

9

expected axioms

\[\begin{align*}
(1 + 2) &= 3 \\
3 \times 3 &= 9
\end{align*}\]
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

\[(\lambda x.x) (\lambda y.y) \]
\[= \alpha (\lambda x.x) (\lambda z.z) \]

expected axiom

\[= (\beta) \]
2D representation of programs

\[ (\lambda x. x + x) \ 3 \]

\[ \text{let } x = 3 \ \text{in} \]

\[ x + x \]

\[ 3 + 3 \]

multiple occurrences of a variable

expected axioms

\[ \psi \]

\[ (\psi) \]

\[ 3 \]

\[ = \]

\[ 3 \]

\[ 3 \]

\[ \text{for copying} \]
2D representation of programs

$$(\lambda x. 1) 3$$

let $x = 3$ in $1$

zero occurrence of a variable

expected axioms

\[ \psi \vdash 3 \]

\[ \psi \]

\[ \psi \]

\[ \psi \]

\[ \psi \]

! for discarding
2D representation of programs

new a = 1 in \!a

reference/location creation
dereference/read

expected axiom

\![1] = 1
2D representation of programs

new a = 1 in !a + !a    new a = 1 in 1 + 1

expected axioms
2D representation of programs

new $a = 1 \text{ in } !a + !a$

new $a = 1 \text{ in } 1 + 1$

indicates

multiple occurrences

 undesired axiom

(let $x = 1 \text{ in }$

(new $a = x \text{ in } !a) + (new a = x \text{ in } !a)$

location indicator

blocks copying
2D representation of programs

- name-free (α-equivalence built in)
- more refined & less structured than 1D syntax

desired feature of a diagrammatic language
- copying vs. sharing

\[ \lambda x. x =_\alpha \lambda y.y \]

\[ \lambda x.1 + x \quad \text{let } w = 1 \text{ in } \lambda x.w + x \]

\[ = \]

\[ \neq \]
2D modelling of program execution

modelling dynamic (operational) behaviour
with strategic diagram-rewriting

\[(1 + 2) \times 3 \rightarrow 3 \times 3 \rightarrow 9\]

rewrite rules

\[
\begin{align*}
\text{1} + \text{2} & \rightarrow \text{3} \\
\text{3} \times \text{3} & \rightarrow 9
\end{align*}
\]
2D modelling of program execution

modelling dynamic (operational) behaviour

with strategical diagram-rewriting

\[(\lambda x. 1 + x) \ 2 \rightarrow 1 + 2 \rightarrow 3\]
2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

strategy of redex search

\[ (1+2) \times (3+4) \longrightarrow 3 \times (3+4) \longrightarrow 3 \times 7 \longrightarrow 21 \]
2D modelling of program execution

modelling dynamic (operational) behaviour
with strategic diagram-rewriting

> strategy of redex search specified by `take`

\[(1+2) \times (3+4) \rightarrow 3 \times (3+4) \rightarrow 3 \times 7 \rightarrow 21\]
2D modelling of program execution

modelling dynamic (operational) behaviour
with strategical diagram-rewriting

- strategy of redex search specified by *taken*

\[(1+2) \ast (3+4) \rightarrow 3 \ast (3+4) \rightarrow 3 \ast 7 \rightarrow 21\]
2D modelling of program execution
modelling dynamic (operational) behaviour
with strategic diagram-rewriting

> strategy of redex search specified by taken

redex search is also rewriting

(found a redex)
2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

→ strategy of redex search specified by taken

redex search is also rewriting

\[(\lambda x.1) \ (2+3) \rightarrow (\lambda x.1) \ 5 \rightarrow 1\]

(call-by-value)
2D modelling of program execution

modelling dynamic (operational) behaviour with **strategical diagram-rewriting**

- strategy of redex search specified by tensor

redex search is also rewriting

\[(\lambda x.1) (2+3) \rightarrow 1\]

(call-by-name)
2D modelling of program execution

modelling dynamic (operational) behaviour with strategic diagram-rewriting

- strategy of duplication

let \( u = \lambda x.1+x \) in \( u(u\ 2) \rightarrow (\lambda x.1+x)((\lambda x.1+x)\ 2) \)
2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of duplication

let \( u = \text{let } w = 1 \text{ in } \lambda x. w + x \text{ in } u \ u2 \)
2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

> strategy of duplication

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

\[
\begin{align*}
\text{let } w = 1 \text{ in } \\
\text{let } u = \lambda x. w + x \text{ in } u \ (u \ 2)
\end{align*}
\]

\[
(\lambda x. w + x) \ (\lambda x. w + x) \ 2
\]
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 black/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

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

[Diagram of box structures]

desired axiom

[Not a functorial box (Mellies)]
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
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 \lambda x. !a' and
'\lambda x. 1' have the same (dynamic) behaviour
in any possible programs

trial with diagrams

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

(let u = \lambda x. 1 in (u 0) + (u 0))
Proving observational equivalence

(\textit{trial with diagrams})

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

(let u = \lambda x. 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

t ::= 
  | x | bind x → u in t
  | a | new a → u in t
  | x . t
  | η (t; t)
SPARTAN, the target calculus

programming
= copying
+ sharing
+ thunking
+ algebra

t ::= 

| x | bind x -> u in t

| a | new a -> u in t

| x . t |

| φ(t; t) |
SPARTAN, the target calculus

programming

= copying

+ sharing

+ thunking

+ algebra

t ::= 

| x | bind x → u in t

| a | new a → u in t

defered computation with a bound variable

| x. t

| y (\overline{t}; \overline{t})
SPARTAN, the target calculus

programming

\[ t ::= \]

| x | bind \( x \to u \) in \( t \)

| a | new \( a \to u \) in \( t \)

| x.t | extrinsic operation with eager arguments & deferred arguments

| \( \varphi(t^2; t^3) \) |
SPARTAN, the target calculus

programming

= copying

+ sharing

+ thanking

+ algebra

examples

extrinsic operation with eager arguments & deferred arguments

\[ \psi(t; \overline{t}) \]
SPARTAN, the target calculus

programming

= copying

+ sharing

+ thunking

+ algebra

\[ t ::= \]

\[  | x \mid \text{bind } x \to u \text{ in } t \]

\[  | a \mid \text{new } a \to u \text{ in } t \]

\[  | x \cdot t \]

extrinsic operation

with eager arguments

& deferred arguments

\[  | \psi(t^2 ; t^2) \]
Proving observational/contextual equivalence
Proving observational/contextual equivalence

recall...

modelling dynamic (operational) behaviour with **strategical diagram-rewriting**

▷ strategy of redex search **specified by** taken

\[(1+2) \times (3+4) \rightarrow 3 \times (3+4) \rightarrow 8 \times 7 \rightarrow 21\]
Proving observational/contextual equivalence

recall...

**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 diagrams**

(let \( u = (\text{new } a=1 \text{ in } \lambda x. !a) \text{ in } (u0)+(u0) \))

(let \( u = \lambda x. 1 \text{ in } (u0)+(u0) \))

tracing sub-diagrams
Proving observational/contextual equivalence

goal prove (generalised) contextual refinement $G \leq^c \circ H$
on diagrams $G, H$

\[ G \leq^c_\circ H \iff \forall C \in \mathcal{C}. \quad \forall k \in \mathbb{N}. \]

\[ C[G] \Downarrow_k \Rightarrow \exists l \in \mathbb{N}. \quad \left( C[H] \Downarrow_l \land k \leq l \right) \]

a preorder on nat. numbers
\[ \mathbb{N} \times \mathbb{N}, =, \geq, \ldots \]
Proving observational/contextual equivalence

goal prove (generalised) contextual refinement $G \leq^e H$

on diagrams $G$, $H$

$G = \begin{array}{c}
\text{1} \\
\text{2} \\
\text{3} \\
\end{array}
$, $H = \begin{array}{c}
\text{1} \\
\text{2} \\
\text{3} \\
\end{array}$
Proving observational/contextual equivalence

**Goal**
prove (generalised) contextual refinement $G \preceq H$

on diagrams $G, H$

**Step 1**
construct a binary relation $R$

on diagrams, s.t. $G \mathrel{R} H$

$G = \begin{array}{c} 1 \end{array} + \begin{array}{c} 2 \end{array}$, $H = \begin{array}{c} 3 \end{array}$

$R := (\begin{array}{c} 1 \end{array}, \begin{array}{c} 2 \end{array}, \begin{array}{c} 3 \end{array})$
Proving observational/contextual equivalence

goal prove (generalised) contextual refinement $G \leq_h H$

on diagrams $G$, $H$

step 1 construct a binary relation $R$

on diagrams, s.t. $G \mathcal{R} H$

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

namely: $\overline{R}^c$

where $\emptyset \in \{1, 2, 3\}$
Proving observational/contextual equivalence

goal prove (generalised) contextual refinement $G \leq^c_H$
on diagrams $G$, $H$

step prove that $R^c$ is a "$c$-weak" simulation

possible cases:
1. $\bigcirc$ or $\bigotimes$ moves within $C$
2. $\bigcirc$ or $\bigotimes$ enters $G$ and $H$
3. $\bigodot$ triggers a rewrite
Proving observational/contextual equivalence

Goal: prove (generalised) contextual refinement $G \preceq H$

on diagrams $G, H$

Step 3: prove that $R^e$ is a "$Q$-weak" simulation

$G = \{1, 2\}$, $H = \{3\}$

$R^e$
Proving observational/contextual equivalence

**Goal**: prove (generalised) contextual refinement $G \leq^c_\Theta H$
on diagrams $G, H$

**Step 3**: prove that $R^c_e$ is a "$\Theta$-weak" simulation

$$G = \begin{array}{c}
\circ \\
\circ \\
\circ \\
\circ
\end{array} + \\
\begin{array}{c}
\circ \\
\circ \\
\circ \\
\circ
\end{array}, \quad H = \begin{array}{c}
\circ \\
\circ
\end{array}$$

$$AND \ (1 + k) Q \ lambda$$

(plus some up-to technique...)
Proving observational/contextual equivalence

- **Goal**: Prove (generalised) contextual refinement $G \leq^e H$
- **Diagram**: $G = \{1, 2\}, \quad H = \{3\}$

**Step 3**: Prove that $R^e$ is a "Q-weak" simulation

**Theorem**: $R^e$ is a "Q-weak" simulation

\[ \Rightarrow R \text{ implies } \leq^e \]
Proving observational/contextual equivalence

Goal: prove (generalised) contextual refinement $G \preceq \rho H$
on diagrams $G, H$

Step 3: prove that $R^e$ is a "$\rho$-weak" simulation

Theorem

$R^e$ is a "$\rho$-weak" simulation

$\Rightarrow R$ implies $\preceq \rho$
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 calculus
modelling dynamic (operational) behaviour

with strategic diagram-rewriting

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/