Local reasoning for robust observational equivalence

Tate Muraya
(RIMS, Kyoto U. & U. Birmingham)

Dan R. Ghica
Todd Waugh Ambridge
(U. Birmingham)
In search of a diagrammatic language for...

Local reasoning for robust observational equivalence
In search of a diagrammatic language for...

modelling program execution
2D representation of programs

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

\[3 \times 3\]

\[9\]
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

\[(\lambda x.x) (\lambda y.y)\]

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

\[\lambda y.y\]
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

\[ \Psi \quad 3 \quad \Rightarrow \quad 3\ 3 \quad \text{for copying} \]
2D representation of programs

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

\(\text{let } x = 3 \text{ in } 1\)

expected axioms

\[ u = \psi(\beta) \]

\(\uparrow \text{ for discarding}\)
2D representation of programs

new a = 1 in !a

reference/location creation
dereference/read

1

reference/location indicator

expected axiom
2D representation of programs

new a = 1 in ![a] + ![a]  new a = 1 in 1 + 1

1

\[ \text{indicator} \]

\[ \text{multiple occurrences} \]

1

\[ \text{expected axioms} \]

\[
\begin{array}{c}
\text{1} \\
\text{!} \\
\hline
\text{1} \\
\text{!} \\
\text{1} \\
\end{array}
\]

=  

\[
\begin{array}{c}
\text{1} \\
\text{1} \\
\text{1} \\
\text{1} \\
\hline
\text{1} \\
\text{!} \\
\end{array}
\]

=  

\[
\begin{array}{c}
\text{1} \\
\text{1} \\
\text{1} \\
\text{1} \\
\hline
\text{1} \\
\text{!} \\
\end{array}
\]

=  

\[
\begin{array}{c}
\text{1} \\
\text{1} \\
\text{1} \\
\end{array}
\]
2D representation of programs

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

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

Let x = 1 \text{ in }

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

Undesired axiom
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

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

\[ \lambda x.1 + x \]

\[ \text{let } w = 1 \text{ in } \lambda x.w + x \]
2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

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

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

modelling dynamic (operational) behaviour with strategical diagram-rewriting

Strategy of redex search specified by taken

\[(1+2) \times (3+4) \rightarrow 3 \times (3+4) \rightarrow 3 \times 7 \rightarrow 21\]

(narrowing scope)

(found a value)
2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

strategy of redex search specified by taken

\[(1+2) \times (3+4) \rightarrow 3 \times (3+4) \rightarrow 3 \times 7 \rightarrow 21\]
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 strategic 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 taken

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 strategical diagram-rewriting

> strategy of duplication

let \( u = \lambda x.1+x \) in \( u(u\ 2) \rightarrow (\lambda x.9+x)((\lambda x.9+x)\ 2) \)
2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

> strategy of duplication

let \( u = (\text{let } w = 1 \text{ in } \lambda x. w + x) \text{ in } u(u2) \)

\[ \text{cf.} \]

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

modelling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of duplication

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

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

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

modeling dynamic (operational) behaviour with strategical diagram-rewriting

- strategy of duplication specified by unit blocks of deferral

- unit of duplication

refinement

\[ \lambda x. 1 + x \]

\[ \text{if } 3 = 1 \text{ then } 1+2 \text{ else } 3+4 \]

- unit of deferral
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 strategic diagram-rewriting

desired feature of a diagrammatic language
- block/box structure

\[
\text{desired axiom} = \quad \not\text{a functorial box} \quad \text{[Mellies]}
\]
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
2D modelling of program execution

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

tryal with terms

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

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

(let u = λx. 1 in (u 0) + (u 0)

→ ((λx. 1) 0) + ((λx. 1) 0)

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

tryal with diagrams

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

(let u = λx. 1 in (u 0) + (u 0))
2D modelling of program execution

tryal 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
2D modelling of program execution

modelling dynamic (operational) behaviour
with strategic diagram-rewriting

... but, modelling for what?

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

proof possible by tracing sub-diagrams
2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

2D language is...

more refined & less structured than 1D syntax

- We can analyse dynamic behaviour by tracing sub-diagrams that may not be sub-terms.

- We need to keep some useful aspects of the (inductive) structure of 1D syntax.
2D modelling of program execution

modelling dynamic (operational) behaviour with strategical diagram-rewriting

What is the right 2D diagrammatical language?

- copying vs. sharing

- black/box structure

- tracing sub-diagrams

   (zoom-in / zoom-out ability?)

   We exploit locality more than compositionality.