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

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

let x = 100 in
50 + 50
```

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

let x = 100 in
50 + 50
```

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

let x = 100 in
50 + 50
```
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

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

\[(\lambda x. 0) 100 \neq 0\]
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

\[
t, u ::= \\
\quad \mid x \mid \text{bind } x \rightarrow u \text{ in } t \\
\quad \mid a \mid \text{new } a \rightarrow u \text{ in } t \\
\quad \mid x \cdot t \\
\quad \mid \phi(t, ..., t; \overline{x} \cdot u, ..., \overline{x} \cdot u)
\]
1. the Spartan calculus

programming

= copying via variables

+ sharing via atoms/names

+ thunking

+ algebra

\[ t, u ::= \]

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

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

\[ |x.t| \]

\[ |\phi(t, \ldots, t; \overrightarrow{x}.u, \ldots, \overrightarrow{x}.u)| \]
1. the Spartan calculus

programming

= copying via variables

+ sharing via atoms/names

+ thunking

+ algebra

\[ t, u ::= \]
\[ | x | \text{bind } x \to u \text{ in } t \]
\[ | a | \text{new } a \to u \text{ in } t \]
\[ | x \cdot t \]
\[ | \phi(t, \ldots, t; \overrightarrow{x}.u, \ldots, \overrightarrow{x}.u) \]
1. The Spartan calculus

Programming
= copying via variables
+ sharing via atoms/names
+ thunking
+ algebra

Language features as (extrinsic) operations

Reference to (a copy of) computation

Location of (shared) computation

Delaying computation with bound variable(s)

\[
t, u ::= \\
| x | \text{bind } x \rightarrow u \text{ in } t \\
| a | \text{new } a \rightarrow u \text{ in } t \\
| x . t \\
| \phi(t, \ldots, t; \overrightarrow{x}.u, \ldots, \overrightarrow{x}.u)\
\]

Muoya (RIMS, Kyoto U. & U. B’ham.)
1. the **Spartan** calculus

programming

= copying via variables

+ sharing via atoms/names

+ thunking

+ algebra

language features as (extrinsic) operations
1. the **Spartan** calculus

programming

= copying via variables

+ sharing via atoms/names

+ thunking

+ algebra

language features as (extrinsic) operations

t, u ::= 

| x | bind x → u in t

da | new a → u in t

| ϕ(t, ..., t; \overrightarrow{x}.u, ..., \overrightarrow{x}.u)

eager arguments

defered arguments (thunks)
1. the Spartan calculus

- Language features as (extrinsic) operations
  - 0, 1, 2, 3, ...
  - PLUS\( (t, u) \)
  - IF\( (t; u_1, u_2) \)
  - LAMBDA\( (; x . t) \)
  - APP\( (t, u) \)
  - LOOKUP\( (t; x . u) \)
  - DEREIF\( (t) \)
  - ASSIGN\( (t, u) \)

- Algebra

- eager arguments
  - | \( \phi(t, ..., t; \overline{x}.u, ..., \overline{x}.u) \) \)

- deferred arguments (thunks)

Muroya (RIMS, Kyoto U. & U. B’ham.)
A general framework
to analyse robustness/fragility of observational equivalence:

1. the Spartan calculus

   \textit{programming} = \textit{copying} + \textit{sharing} + \textit{thunking} + \textit{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

<table>
<thead>
<tr>
<th>(Spartan) Term</th>
<th>Hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1 + 2) \times 3)</td>
<td><img src="" alt="Diagram" /></td>
</tr>
<tr>
<td>(\text{TIMES}(\text{PLUS}(1,2),3))</td>
<td></td>
</tr>
</tbody>
</table>
2. A universal abstract machine

computation = focussed “hyernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(SPARTAN) term</th>
<th>hyernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1 + 2) \times 3)</td>
<td>![Diagram of nodes and hyperedges]</td>
</tr>
<tr>
<td>(\text{TIMES}({\text{PLUS}(1,2)},3))</td>
<td>![Diagram of nodes and hyperedges]</td>
</tr>
</tbody>
</table>

Muroya (RIMS, Kyoto U. & U. B’ham.)
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(SPARTAN) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 + 2) × 3</td>
<td></td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td></td>
</tr>
</tbody>
</table>
2. A universal abstract machine

computation = focussed “hypermnet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(SPARTAN) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1 + 2) \times 3$</td>
<td>![Hypernet Diagram]</td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td>![Hypernet Diagram]</td>
</tr>
</tbody>
</table>

“focus”
- bring query $?$ up
- bring answer $✓$/Ụ down
- trigger rewrite $✓$
2. A universal abstract machine

computation = focussed “hyernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>hyernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1 + 2) \times 3)</td>
<td><img src="image" alt="Hyernet Diagram" /></td>
</tr>
<tr>
<td><code>TIMES(PLUS(1,2),3)</code></td>
<td></td>
</tr>
</tbody>
</table>

“focus”
- bring query ? up
- bring answer ✓/✗ down
- trigger rewrite ✗
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th><strong>(SPARTAN) term</strong></th>
<th><strong>hypernet</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>((1 + 2) \times 3)</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

“focus”
- bring query ? up
- bring answer ✓ / ✎ down
- trigger rewrite ↘
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1 + 2) \times 3)</td>
<td>![Hypernet Diagram]</td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td>“focus”</td>
</tr>
<tr>
<td>“focus”</td>
<td>• bring query (\uparrow) up</td>
</tr>
<tr>
<td></td>
<td>• bring answer (\checkmark/\downarrow) down</td>
</tr>
<tr>
<td></td>
<td>• trigger rewrite (\downarrow)</td>
</tr>
</tbody>
</table>

Muroya (RIMS, Kyoto U. & U. B’ham.)
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(SPARTAN) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 + 2) × 3</td>
<td>![hypernet diagram]</td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td>![hypernet diagram]</td>
</tr>
</tbody>
</table>

“focus”
- bring query ? up
- bring answer ✓/.downcase
- trigger rewrite ↩
2. A universal abstract machine

computation = focussed “hykernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>Hykernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 + 2) × 3</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

“focus”
- bring query ? up
- bring answer ✓ / ✗ down
- trigger rewrite ↘
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1 + 2) \times 3$</td>
<td><img src="image" alt="Hypernet Diagram" /></td>
</tr>
<tr>
<td>\text{TIMES}(\text{PLUS}(1,2),3)</td>
<td><img src="image" alt="Hypernet Diagram" /></td>
</tr>
</tbody>
</table>

“focus”
- bring query ? up
- bring answer ✓/↯ down
- trigger rewrite ↯
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1 + 2) \times 3$</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td>“focus”</td>
</tr>
</tbody>
</table>
  - bring query ? up
  - bring answer √/✔ down
  - trigger rewrite ❌
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(SPARTAN) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1 + 2) \times 3$</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

“focus”
- bring query ? up
- bring answer ✓/✓ down
- trigger rewrite ⇪
2. A universal abstract machine

computation = focussed “hyernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>hyernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1 + 2) \times 3$</td>
<td><img src="image" alt="hyernet diagram" /></td>
</tr>
<tr>
<td><code>TIMES(PLUS(1,2),3)</code></td>
<td><img src="image" alt="hyernet diagram" /></td>
</tr>
</tbody>
</table>

“focus”
- bring query `?` up
- bring answer ✓/✓ down
- trigger rewrite ↘
2. A universal abstract machine

computation = focussed “hyoernet” rewriting

• higher-order hypergraph
• hierarchical hypergraph

<table>
<thead>
<tr>
<th>(SPARTAN) term</th>
<th>hyoernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1 + 2) \times 3)</td>
<td>TIMES(PLUS(1,2),3)</td>
</tr>
</tbody>
</table>

“focus”
• bring query \(?\) up
• bring answer \(✓\) down
• trigger rewrite \(✓\)
2. A universal abstract machine

Computation = focussed “hypernet” rewriting

- Higher-order hypergraph
- Hierarchical hypergraph

<table>
<thead>
<tr>
<th>(SPARTAN) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1 + 2) \times 3)</td>
<td>![Diagram of hypernet]</td>
</tr>
</tbody>
</table>

“Focus”
- Bring query \(?\) up
- Bring answer \(✓/✗\) down
- Trigger rewrite \(✓\)
2. A universal abstract machine

computation = focussed “hypernet” rewriting

• higher-order hypergraph
• hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 + 2) × 3</td>
<td></td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td></td>
</tr>
</tbody>
</table>

“focus”
• bring query ? up
• bring answer ✓ / ✅ down
• trigger rewrite ↘
2. A universal abstract machine

computation = focussed “hypernet” rewriting

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 + 2) × 3</td>
<td></td>
</tr>
<tr>
<td>TIMES(PLUS(1,2),3)</td>
<td></td>
</tr>
</tbody>
</table>

“focus”
- bring query ? up
- bring answer √/��态 down
- trigger rewrite ⇣
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>Hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda x. x + x) 3)</td>
<td></td>
</tr>
<tr>
<td>APP(LAMBDA((; x \cdot \text{PLUS}(x, x))), 3)</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Muroya (RIMS, Kyoto U. & U. B’ham.)
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\lambda x.x + x)\ 3$</td>
<td></td>
</tr>
<tr>
<td>APP(LAMBDA(; \ x.PLUS(x, x)), 3)</td>
<td></td>
</tr>
</tbody>
</table>

Muroya (RIMS, Kyoto U. & U. B’ham.)
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(SPARTAN) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda x \cdot x + x)\ 3)</td>
<td>(\text{hyperedge labelled with hypernet})</td>
</tr>
<tr>
<td>(\text{APP(LAMBDA(;} \cdot x . \text{PLUS}(x, x)), \ 3))</td>
<td></td>
</tr>
</tbody>
</table>
$(\lambda x . x + x) 3$
$(\lambda x. x + x) 3$

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

Muroya (RIMS, Kyoto U. & U. B’ham.)
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

(SparTan) term | hypernet
---|---
new \(a \rightarrow 1\) in \((a := 2; !a)\) | ![Diagram](image)

new \(a \rightarrow 1\) in \(\text{SEC} (\text{ASSIGN}(a,2); \text{DEREF}(a))\)

Hyperedge labelled with hypernet
2. A universal abstract machine

computation = focussed “hypermnet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(Spartan) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>new $a \leftarrow 1 \text{ in } (a := 2; !a)$</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>new $a \leftarrow 1 \text{ in SEC(ASSIGN(a,2); DEREF(a))}$</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Muroya (RIMS, Kyoto U. & U. B’ham.)
2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>(SPARTAN) term</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>new $a \to 1$ in ($a := 2; !a$)</td>
<td><img src="1" alt="Diagram" /></td>
</tr>
<tr>
<td>new $a \to 1$ in SEC(ASSIGN($a, 2$); DEREF($a$))</td>
<td><img src="2" alt="Diagram" /></td>
</tr>
</tbody>
</table>

name occurrences

Muroya (RIMS, Kyoto U. & U. B’ham.)
### 2. A universal abstract machine

computation = focussed “hypernet” rewriting

- higher-order hypergraph
- hierarchical hypergraph

<table>
<thead>
<tr>
<th>program</th>
<th>hypernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>program fragment</td>
<td>sub-hypernet (sub-graph)</td>
</tr>
<tr>
<td>program execution</td>
<td>moves of focus (query / answer) &amp; focussed rewrites</td>
</tr>
</tbody>
</table>
A general framework
to analyse robustness/fragility of observational equivalence:

1. the Spartan calculus

\[ \text{programming} = \text{copying} + \text{sharing} + \text{thunking} + \text{algebra} \]

2. a universal abstract machine

\[ \text{computation} = \text{focussed “hypernet” rewriting} \]

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 \( \text{new } a \to 1 \text{ in } \lambda x. (a := 2; !a) \) behave the same as \( \lambda x. 2 \)?”
3. Locality & equational reasoning

**Locality** of graph syntax

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

with linear syntax:

\[
\begin{array}{cccccc}
\cdots & \textbf{new} & a & \rightarrow & 1 & \text{in} & \cdots \\
\cdots & \lambda x. (a := 2; !a) & \cdots & \lambda x. (a := 2; !a) & \cdots \\
\cdots & \lambda x. 2 & \cdots & \lambda x. 2 & \cdots \\
\end{array}
\]
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

```
... new a ← 1 in ... λx.(a := 2; !a) ... λx.(a := 2; !a) ...

... λx.2 ... λx.2 ...
```
3. Locality & equational reasoning

**Locality** of graph syntax

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

with linear syntax: \textit{comparison between sub-terms}

\[
\begin{array}{cccc}
\cdots & \text{new } a \rightarrow 1 \text{ in } & \cdots & \lambda x.(a := 2; !a) \cdots \\
\cdots & \lambda x.2 & \cdots & \lambda x.2 \cdots \\
\end{array}
\]

with graph syntax: \textit{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)
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

   \[ \text{programming} = \text{copying} + \text{sharing} + \text{thunking} + \text{algebra} \]

2. a universal abstract machine

   \[ \text{computation} = \text{focussed “hypernet” rewriting} \]

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/](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