Local Reasoning for Robust Observational Equivalence

Dan R. Ghica\textsuperscript{1}
Koko Muroya\textsuperscript{1,2}
Todd Waugh Ambridge\textsuperscript{1}

\textsuperscript{1}SoCS, University of Birmingham
\textsuperscript{2}RIMS, Kyoto University

PERR 2019 (Prague), 6 April 2019
Outline

1. Motivation
2. The SPARTAN Calculus
3. SPARTAN Semantics – Focussed Graph Rewriting
4. The Characterisation Theorem
5. An Application of The Characterisation Theorem
6. Conclusion & Further Work
Motivation

In a language, terms $t_1, t_2$ are equivalent if they are observationally equivalent:

$$t_1 \equiv t_2 \iff \forall C. C[t_1] \simeq C[t_2]$$

(for given notions of “terms”, “all contexts” and “$\simeq$”)
Motivation

In a language, terms $t_1, t_2$ are equivalent if they are observationally equivalent:

$$t_1 \equiv t_2 \iff \forall C. C[t_1] \simeq C[t_2]$$

(for given notions of “terms”, “all contexts” and “≃”)

Equivalence reasoning in a language with effects is fragile
Motivation

In a language, terms $t_1, t_2$ are equivalent if they are observationally equivalent:

$$t_1 \equiv t_2 \iff \forall C. C[t_1] \simeq C[t_2]$$

(for given notions of "terms", "all contexts" and "$\simeq$")

Equivalence reasoning in a language with effects is fragile

How to make a robust way of reasoning about languages with effects?

Characterise effects by their consequences on the equational properties of the language

An equational law that is not broken by any semantic feature is robust
Motivation

In a language, terms $t_1, t_2$ are equivalent if they are observationally equivalent:

$$t_1 \equiv t_2 \iff \forall C. C[t_1] \simeq C[t_2]$$

(for given notions of “terms”, “all contexts” and “≃”)

Equivalence reasoning in a language with effects is fragile

How to make a robust way of reasoning about languages with effects?

Characterise effects by their consequences on the equational properties of the language

An equational law that is not broken by any semantic feature is robust

We need a common framework for equational reasoning
The SPARTAN Calculus

Three key intrinsic elements:
- **Variables** that manage copying
- **Names** that manage sharing
- **Thunks** that manage evaluation

Operations are defined extrinsically

\[
\begin{align*}
t & ::= \\
& \vdash x \mid \text{bind } x \rightarrow t' \text{ in } t'' \\
& \vdash a \mid \text{new } a \rightarrow t' \text{ in } t'' \\
& \vdash \overline{y}. t' \\
& \vdash \phi(t'; t'')
\end{align*}
\]

These model all interesting semantic features – datatypes, functions, effects
They may take eager arguments and deferred arguments (thunks)
They are partitioned into passive operations and active operations
i.e. Values and redexes
The SPARTAN Calculus

Three key intrinsic elements:

- **Variables** that manage copying
- **Names** that manage sharing
- **Thunks** that manage evaluation

Operations are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects
They may take eager arguments and deferred arguments (thunks)
They are partitioned into *passive operations* and *active operations*
i.e. Values and redexes

\[ n \Rightarrow n(-; -) \]
The **SPARTAN Calculus**

Three key intrinsic elements:

- **Variables** that manage copying
- **Names** that manage sharing
- **Thunks** that manage evaluation

**Operations** are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects
They may take eager arguments and deferred arguments (thunks)
They are partitioned into passive operations and active operations
i.e. Values and redexes

\[
\begin{align*}
t & ::= \\
& x \mid \text{bind } x \rightarrow t' \text{ in } t'' \\
& a \mid \text{new } a \rightarrow t' \text{ in } t'' \\
& \bar{y}. t' \\
& \phi(t'; t'')
\end{align*}
\]

\[
t + u \mapsto +(t, u; -)
\]
The SPARTAN Calculus

Three key intrinsic elements:

- **Variables** that manage copying
- **Names** that manage sharing
- **Thunks** that manage evaluation

**Operations** are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects
They may take eager arguments and deferred arguments (thunks)
They are partitioned into passive operations and active operations
i.e. Values and redexes

\[ \lambda x. t \mapsto \lambda (\_ ; x. t) \quad u \; v \mapsto @ (u, v; \_) \]
The SPARTAN Calculus

Three key intrinsic elements:

- **Variables** that manage copying
- **Names** that manage sharing
- **Thunks** that manage evaluation

**Operations** are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects
They may take eager arguments and deferred arguments (thunks)
They are partitioned into passive operations and active operations
i.e. Values and redexes

\[
t; u \mapsto \text{seq}(t; u)
\]
The SPARTAN Calculus

Three key intrinsic elements:

- **Variables** that manage copying
- **Names** that manage sharing
- **Thunks** that manage evaluation

**Operations** are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects
They may take eager arguments and deferred arguments (thunks)
They are partitioned into **passive operations** and **active operations**
  i.e. Values and redexes

\[ t := u \mapsto \text{assign}(t, u; \neg) \]
The SPARTAN Calculus

Three key intrinsic elements:
- **Variables** that manage copying
- **Names** that manage sharing
- **Thunks** that manage evaluation

**Operations** are defined extrinsically

These model all interesting semantic features – datatypes, functions, effects
They may take eager arguments and deferred arguments (thunks)
They are partitioned into **passive operations** and **active operations**
  i.e. Values and redexes

\[ t ::= \]
\[ x \mid \text{bind } x \to t' \text{ in } t'' \]
\[ | a \mid \text{new } a \to t' \text{ in } t'' \]
\[ | \tilde{y}. t' \]
\[ | \phi(t'; t'') \]

\[ ! t \mapsto \text{deref}(t; -) \]
The SPARTAN Calculus

Three key intrinsic elements:

- Variables that manage copying
- Names that manage sharing
- Thunks that manage evaluation

Operations are defined extrinsically

\[
 t ::= \\
 x \mid \text{bind } x \rightarrow t' \text{ in } t'' \\
 a \mid \text{new } a \rightarrow t' \text{ in } t'' \\
 \overline{y}. t' \\
 \varphi(t'; t'') \\
\]

An Alternative Intuition:

Universal Algebra

+ Sharing, Copying, Thunking

= A Programming Language with Effects
SPARTAN Semantics

bind \( z \rightarrow 1 + 2 \) in

\((if \ (z == 3) \ then \ (1 + z) \ else \ (2 + z))\)
SPARTAN Semantics

\[ \text{bind } z \rightarrow 1 + 2 \text{ in} \]
\[ (\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z)) \]
SPARTAN Semantics

\[ \text{bind } z \rightarrow 1 + 2 \text{ in} \]
\[ (\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z)) \]
PARTAN Semantics

bind $z \rightarrow 1 + 2$ in
(if ($z == 3$) then ($1 + z$) else ($2 + z$))
**SPARTAN Semantics**

bind $z \rightarrow 1 + 2$ in

(if ($z == 3$) then $(1 + z)$ else $(2 + z)$)

**Hypernets**
(Nested labelled hypergraphs)

**Token:**
- ? – Search
- ✓ – Value
- ✂ – Rewrite
**SPARTAN Semantics**

`bind z → 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))`
SPARTAN Semantics

bind \( z \to 1 + 2 \) in
\((if \ (z == 3) \ then \ (1 + z) \ else \ (2 + z))\)

Token:
- \( ? \) – Search
- \( ✓ \) – Value
- \( \_\_\_ \) – Rewrite

Hypernets (Nested labelled hypergraphs)

Intrinsic Copying Rewrite:
SPARTAN Semantics

\[ \text{bind } z \rightarrow 1 + 2 \text{ in} \]
\[ (if \ (z == 3) \ then \ (1 + z) \ else \ (2 + z)) \]

Intrinsic Copying Rewrite:

Token:
- ? – Search
- ✓ – Value
- ✎ – Rewrite
**SPARTAN Semantics**

\[ \text{bind } z \to 1 + 2 \text{ in } \]
\[ (\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z)) \]

**Intrinsic Copying Rewrite:**

**Token:**
- `?` - Search
- `✓` - Value
- `‡` - Rewrite
SPARTAN Semantics

\( \text{bind } z \rightarrow 1 + 2 \text{ in} \)

\((if \ (z == 3) \text{ then } (1 + z) \text{ else } (2 + z))\)

**Intrinsic Copying Rewrite:**

Token:
- ? – Search
- ✓ – Value
- ⬩ – Rewrite
SPARTAN Semantics

\( \text{bind } z \rightarrow 1 + 2 \text{ in } \)

\((if (z == 3) \text{ then } (1 + z) \text{ else } (2 + z)) \)

**Token:**

- \(\times\) – Search
- \(\checkmark\) – Value
- \(\Rightarrow\) – Rewrite

**Intrinsic Copying Rewrite:**
bind z → 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))
SPARTAN Semantics

bind z → 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))
SPARTAN Semantics

\( bind \ z \rightarrow 1 + 2 \ in \)
\( (if \ (z == 3) \ then \ (1 + z) \ else \ (2 + z)) \)
SPARTAN Semantics

(bind z → 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z)))
SPARTAN Semantics

bind $z \rightarrow 1 + 2$ in
(if ($z == 3$) then $(1 + z)$ else $(2 + z)$)
SPARTAN Semantics

bind $z \rightarrow 1 + 2$ in
(if ($z == 3$) then $(1 + z)$ else $(2 + z)$)

Extrinsic Addition Op. Rewrite:
**SPARTAN Semantics**

bind $z \to 1 + 2$ in

$(if \ (z == 3) \ then \ (1 + z) \ else \ (2 + z))$

*Extrinsic Addition Op. Rewrite:*
SPARTAN Semantics

bind z → 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))
SPARTAN Semantics

bind \( z \rightarrow 1 + 2 \) in

\((\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z))\)
SPARTAN Semantics

bind z → 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))
**SPARTAN Semantics**

\[ \text{bind } z \rightarrow 1 + 2 \text{ in} \]
\[(\text{if } z == 3 \text{ then } (1 + z) \text{ else } (2 + z))\]
**Spartan Semantics**

_bind z → 1 + 2 in_

(if (z == 3) then (1 + z) else (2 + z))
**SPARTAN Semantics**

bind $z \rightarrow 1 + 2$ in

$(\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z))$
**SPARTAN Semantics**

*bind z → 1 + 2 in*

*(if (z == 3) then (1 + z) else (2 + z))*

Extrinsic If-then-else Op. Rewrite:
**SPARTAN Semantics**

\[
\text{bind } z \rightarrow 1 + 2 \text{ in } \\
(\text{if } (z \equiv 3) \text{ then } (1 + z) \text{ else } (2 + z))
\]
SPARTAN Semantics

\[ \text{bind } z \to 1 + 2 \text{ in} \]
\[ (if \ (z == 3) \ then \ (1 + z) \ else \ (2 + z)) \]
**SPARTAN Semantics**

\[ \text{bind } z \rightarrow 1 + 2 \text{ in } \\
(\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z)) \]
**SPARTAN Semantics**

\[
\text{bind } z \rightarrow 1 + 2\text{ in } \\
(\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z))
\]
SPARTAN Semantics

\[ \text{bind } z \rightarrow 1 + 2 \text{ in} \]
\[ (\text{if } (z \equiv 3) \text{ then } (1 + z) \text{ else } (2 + z)) \]
SPARTAN Semantics

bind \( z \rightarrow 1 + 2 \) in

\((\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z))\)
SPARTAN Semantics

bind z \to 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))
Spartan Semantics

bind $z \rightarrow 1 + 2 \ in$

$(if \ (z \ == \ 3) \ then \ (1 + z) \ else \ (2 + z))$
SPARTAN Semantics

bind $z \rightarrow 1 + 2$ in
(if ($z == 3$) then $(1 + z)$ else $(2 + z)$)
SPARTAN Semantics

bind z → 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))
SPARTAN Semantics

bind z \rightarrow 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))
SPARTAN Semantics

\[ \text{bind } z \rightarrow 1 + 2 \ in \ (if \ (z \ == \ 3) \ then \ (1 + z) \ else \ (2 + z)) \]
**SPARTAN Semantics**

bind $z \rightarrow 1 + 2$ in 
(if $(z == 3)$ then $(1 + z)$ else $(2 + z)$)

```
1 + 1
  +
  +
  +
  +
PLUS(2,z)
```
bind \( z \to 1 + 2 \) in

\((\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z))\)
SPARTAN Semantics

bind z → 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))

PLUS(2, z)
SPARTAN Semantics

bind z \rightarrow 1 + 2 in
(if (z == 3) then (1 + z) else (2 + z))
**SPARTAN Semantics**

\[\text{bind } z \rightarrow 1 + 2 \text{ in} \]

\[(if \ (z \equiv 3) \ then \ (1 + z) \ else \ (2 + z))\]
bind $z \rightarrow 1 + 2$ in

$(\text{if } (z == 3) \text{ then } (1 + z) \text{ else } (2 + z))$
**SPARTAN Semantics**

`bind z \rightarrow 1 + 2\ in`

`(if (z == 3) then (1 + z) else (2 + z))`
SPARTAN Semantics

$$(\lambda x. x + x) 2$$
new $a \rightarrow 5$ in

$a := 6;$

!$a$
The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

* Rewrite rules of operations are deterministic and refocusing
The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

*Rewrite rules of operations are deterministic and refocusing

Recall: Terms $t_1, t_2$ are equivalent if they are observationally equivalent:

$$t_1 \equiv t_2 \iff \forall C^{bf}. \text{Init}(C^{bf}[t_1]) \simeq \text{Init}(C^{bf}[t_2])$$
The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

*Rewrite rules of operations are deterministic and refocusing

Recall: Terms $t_1, t_2$ are equivalent if they are observationally equivalent:

$$t_1 \equiv t_2 \iff \forall C^{bf}.\text{Init}(C^{bf}[t_1]) \simeq \text{Init}(C^{bf}[t_2])$$

Terms $(t_1, t_2)$:

$$t := x \; | \; \text{bind } x \to t' \; \text{in } t'' \; | \; a \; | \; \text{new } a \to t' \; \text{in } t'' \; | \; \tilde{y}.t' \; | \; \phi(t'; t'')$$
The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

*Rewrite rules of operations are deterministic and refocusing

Recall: Terms $t_1, t_2$ are equivalent if they are observationally equivalent:

$$t_1 \equiv t_2 \iff \forall C^{bf}. \text{Init}(C^{bf}[t_1]) \simeq \text{Init}(C^{bf}[t_2])$$

Terms $(t_1, t_2)$:

$$t := x | \text{bind } x \rightarrow t' \text{ in } t'' | a | \text{new } a \rightarrow t' \text{ in } t'' | \checkmark . t' | \phi(t'; t'')$$

All contexts $(\forall C)$:

$$C^{bf} := \square | \text{bind } x \rightarrow t \text{ in } C' | \text{new } a \rightarrow t \text{ in } C' | \checkmark . C' |$$

$$\phi(t, C', t'; t'') | \phi(t; t', C', t'')$$
The Characterisation Theorem

For any* given set of operations, when are two SPARTAN terms equivalent?

*Rewrite rules of operations are deterministic and refocusing

Recall: Terms \( t_1, t_2 \) are equivalent if they are observationally equivalent:

\[
\forall C^{bf}. \text{Init}(C^{bf}[t_1]) \approx \text{Init}(C^{bf}[t_2])
\]

Terms \( (t_1, t_2): \)

\[
t := x \mid \text{bind } x \rightarrow t' \text{ in } t'' \mid a \mid \text{new } a \rightarrow t' \text{ in } t'' \mid \tilde{y}.t' \mid \phi(t';t'')
\]

All contexts \( (\forall C): \)

\[
C^{bf} := \emptyset \mid \text{bind } x \rightarrow t \text{ in } C' \mid \text{new } a \rightarrow t \text{ in } C' \mid \tilde{y}.C' \mid 
\phi(t, C', t'; t'') \mid \phi(t; t', C', t'')
\]

Equivalence of output behaviour \( (\simeq): \) Untyped setting!

Terms have the same output behaviour if their initial states are state equivalent:

\[
\dot{G}_1 \simeq \dot{G}_2 \iff \dot{G}_1 \rightarrow^* \dot{N}_1 \text{ (final state)} \text{ if and only if } \dot{G}_2 \rightarrow^* \dot{N}_2 \text{ (final state)}
\]
The Characterisation Theorem

SPARTAN graphs $g_1, g_2$ are equivalent if their initial states are state equivalent:

$$g_1 \equiv g_2 \iff \forall C^{bf}. \text{Init}(C^{bf}[g_1]) \simeq \text{Init}(C^{bf}[g_2])$$

Contexts in the hypernet model are more expressive than those in the term model:
- Can immediately identify relevant direct interactions arising in a term
- Thus, there are not hidden interactions between the program and context

This leads to a notion of local reasoning about programs
The Characterisation Theorem

\[ g_1 \equiv g_2 \iff \forall C^{bf}. \text{Init}(C^{bf}[g_1]) \simeq \text{Init}(C^{bf}[g_2]) \]
The Characterisation Theorem

*Pre-template:* A family of binary relations on hypernets with the same interface

\[ g_1 \equiv g_2 \iff \forall C^{bf}. \text{Init}(C^{bf}[g_1]) \simeq \text{Init}(C^{bf}[g_2]) \]
The Characterisation Theorem

\[ g_1 \equiv g_2 \iff \forall C^{bf}. \text{Init}(C^{bf}[g_1]) \simeq \text{Init}(C^{bf}[g_2]) \]
\[ \forall C^{bf}. \]

\[ g_1 \equiv g_2 \iff \forall C^{bf}. \text{Init}(C^{bf}[g_1]) \simeq \text{Init}(C^{bf}[g_2]) \]
The Characterisation Theorem

$\equiv$

$g_1$

$C^{bf}$

$g_2$

$C^{bf}$
The Characterisation Theorem

\[
\begin{align*}
\tilde{g}_1 & \sim \tilde{g}_2 \\
C^{bf} & \sim C^{bf}
\end{align*}
\]
The Characterisation Theorem

\[ g_1 \sim g_2 \]
The Characterisation Theorem

Robust: A pre-template that is preserved by any rewrite in the context
The Characterisation Theorem

\[ C^{bf} \approx ? \]

\[ g_1 \]

\[ \sim \]

\[ C^{bf} \]

\[ g_2 \]
The Characterisation Theorem

**Input-safe:** A pre-template that is preserved by any input search token
The Characterisation Theorem

\[ g_1 \sim g_2 \]

\[ C^{bf} \]
The Characterisation Theorem

**Output-closed:** A pre-template where the token cannot reach an output from the initial state
A *pre-template* is a family of binary relations on graphs with the same interface.

A *template* is a pre-template that is both *input-safe* and *output-closed*.

*Input-safe:* The pre-template is preserved by any input search token.

*Output-closed:* The pre-template prevents any output search token.

A (pre-)template is *robust* if it is preserved by any rewrite in any context.

**Characterisation Theorem.**
Robust templates induce observational equivalences.
The Characterisation Theorem

**Definition.** A pre-template $g_1 \triangleleft g_2$ is *input-safe* if, for any valid focussed context $\hat{C}$ with an input search token, one of the following holds:

1. $\hat{C}[g_1] \rightarrow^* \hat{N}_1$ and $\hat{C}[g_2] \rightarrow^* \hat{N}_2$
   for two stuck states $\hat{N}_1, \hat{N}_2$
2. $\hat{C}[g_1] \rightarrow^* \hat{C}'[g_1']$ and $\hat{C}[g_2] \rightarrow^* \hat{C}'[g_2']$
   for two hypernets $g_1', g_2'$ such that $g_1' \triangleleft g_2'$
   such that the token is not in rewrite status

**Definition.** A pre-template $g_1 \triangleleft g_2$ is *robust* if, for any valid focussed context $\hat{C}$ with a rewrite token, one of the following holds:

1. $\hat{C}[g_1] \rightarrow^+ \hat{N}_1$ and $\hat{C}[g_2] \rightarrow^* \hat{N}_2$
   for two stuck states $\hat{N}_1, \hat{N}_2$
2. $\hat{C}[g_1] \rightarrow^+ \hat{C}'[g_1']$ and $\hat{C}[g_2] \rightarrow^* \hat{C}'[g_2']$
   for two hypernets $g_1', g_2'$ such that $g_1' \triangleleft g_2'$
   such that the token is not in rewrite status
We want to show that the $\beta$-Law pre-template is a robust template.
The Characterisation Theorem

We want to show that the \( \beta \)-Law pre-template is a robust template.

The \( \beta \)-Law pre-template is a family of binary relations.
We want to show that the $\beta$-Law pre-template is a robust template.

The $\beta$-Law pre-template is a family of binary relations.
The Characterisation Theorem

\[ \mathcal{C}[g_1] \]

\[ \mathcal{C}[g_2] \]

Input-safe

Output-closed

Robust
The Characterisation Theorem

**Def.** The pre-template is *input-safe* if *input-safe* if for any valid focussed context \( \hat{C} \) with an input search token one of the following holds:

1. \( \hat{C}[g_1] \rightarrow^* \hat{N} \) and \( \hat{C}[g_2] \rightarrow^* \hat{N}_2 \)

   for two stuck states \( \hat{N}_1, \hat{N}_2 \)

2. \( \hat{C}[g_1] \rightarrow^* \hat{C}'[g'_1] \) and \( \hat{C}[g_2] \rightarrow^* \hat{C}'[g'_2] \)

   for two hypernets \( g'_1, g'_2 \) s.t. \( g'_1 \triangleleft g'_2 \)

   and a valid focussed context \( \hat{C}' \) such that the token is not in rewrite status

---

**Input-safe**

- ✓
- ?

**Output-closed**

- ✓
- ?

**Robust**

- ?
- ?
The Characterisation Theorem

**Def.** The pre-template is *input-safe* if _input-safe_ if for any valid focussed context $\check{C}$ with an input search token one of the following holds:

1. $\check{C}[g_1] \rightarrow^* \check{N}$ and $\check{C}[g_2] \rightarrow^* \check{N}_2$ for two stuck states $\check{N}_1, \check{N}_2$

2. $\check{C}[g_1] \rightarrow^* \check{C}'[g_1]$ and $\check{C}[g_2] \rightarrow^* \check{C}'[g_2]$ for two hypernets $g_1', g_2'$ s.t. $g_1' \triangleleft g_2'$ and a valid focussed context $\check{C}'$ such that the token is not in rewrite status

**Input-safe**  
Output-closed ✓  
Robust ?
**The Characterisation Theorem**

**Def.** The pre-template is *input-safe* if *input-safe* if for any valid focussed context $\hat{\mathcal{C}}$ with an input search token one of the following holds:

1. $\hat{\mathcal{C}}[g_1] \rightarrow^* \hat{\mathcal{N}}$ and $\hat{\mathcal{C}}[g_2] \rightarrow^* \hat{\mathcal{N}}_2$ for two stuck states $\hat{\mathcal{N}}_1, \hat{\mathcal{N}}_2$

2. $\hat{\mathcal{C}}[g_1] \rightarrow^* \hat{\mathcal{C}}'[g_1]'$ and $\hat{\mathcal{C}}[g_2] \rightarrow^* \hat{\mathcal{C}}'[g_2]'$ for two hypernets $g_1', g_2'$ s.t. $g_1' \prec g_2'$ and a valid focussed context $\hat{\mathcal{C}}'$ such that the token is not in rewrite status.

**Input-safe**

- Output-closed ✓
- Robust ?

---

Token bounces off and goes to other argument

---

PERR (2019)  
Local Reasoning for Robust Observational Equivalence

4 / 6
The Characterisation Theorem

**Def.** The pre-template is *input-safe* if *input-safe* if for any valid focussed context \( \hat{C} \) with an input search token one of the following holds:

1. \( \hat{C}[g_1] \rightarrow^* \hat{N} \) and \( \hat{C}[g_2] \rightarrow^* \hat{N}_2 \) for two stuck states \( \hat{N}_1, \hat{N}_2 \)

2. \( \hat{C}[g_1] \rightarrow^* \hat{C}'[g'_1] \) and \( \hat{C}[g_2] \rightarrow^* \hat{C}'[g'_2] \) for two hypernets \( g'_1, g'_2 \) s.t. \( g'_1 \triangleleft g'_2 \) and a valid focussed context \( \hat{C}' \) such that the token is not in rewrite status

**Input-safe**

* Output-closed ✓
* Robust ?

Token will come back (stable hypernet)

Token bounces off and goes to other argument

The Characterisation Theorem

PERR (2019) Local Reasoning for Robust Observational Equivalence
Def. The pre-template is **input-safe** if **input-safe** if for any valid focussed context $\hat{C}$ with an input search token one of the following holds:

1. $\hat{C}[g_1] \rightarrow^* \hat{N}_1$ and $\hat{C}[g_2] \rightarrow^* \hat{N}_2$ for two stuck states $\hat{N}_1, \hat{N}_2$

2. $\hat{C}[g_1] \rightarrow^* \hat{C}'[g_1']$ and $\hat{C}[g_2] \rightarrow^* \hat{C}'[g_2']$ for two hypernets $g_1', g_2'$ s.t. $g_1' \triangleleft g_2'$ and a valid focussed context $\hat{C}'$ such that the token is not in rewrite status

**Input-safe**

- Output-closed ✓
- Robust ?

---

The Characterisation Theorem

**PERR (2019)**

Local Reasoning for Robust Observational Equivalence

4 / 6
The Characterisation Theorem

**Def.** The pre-template is *input-safe* if *input-safe* if for any valid focussed context $\hat{C}$ with an input search token one of the following holds:

1. $\hat{C}[g_1] \rightarrow^* \hat{N}_1$ and $\hat{C}[g_2] \rightarrow^* \hat{N}_2$ for two stuck states $\hat{N}_1, \hat{N}_2$

2. $\hat{C}[g_1] \rightarrow^* \hat{C'}[g_1]$ and $\hat{C}[g_2] \rightarrow^* \hat{C'}[g_2]$ for two hypernets $g_1', g_2'$ s.t. $g_1' \triangleleft g_2'$ and a valid focussed context $\hat{C'}$ such that the token is not in rewrite status

**Input-safe** ?

**Output-closed** ✓

**Robust** ?

---

PERR (2019)  Local Reasoning for Robust Observational Equivalence  4 / 6
The Characterisation Theorem

**Def.** The pre-template is *input-safe* if *input-safe* if for any valid focussed context $\hat{C}$ with an input search token one of the following holds:

1. $\hat{C}[g_1] \rightarrow^* \hat{N}$ and $\hat{C}[g_2] \rightarrow^* \hat{N}_2$ for two stuck states $\hat{N}_1, \hat{N}_2$

2. $\hat{C}[g_1] \rightarrow^* \hat{C}'[g'_1]$ and $\hat{C}[g_2] \rightarrow^* \hat{C}'[g'_2]$ for two hypernets $g'_1, g'_2$ s.t. $g'_1 \triangleleft g'_2$ and a valid focussed context $\hat{C}'$ such that the token is not in rewrite status

**Input-safe**
- ✓

**Output-closed**
- ✓

**Robust**
- ?
The Characterisation Theorem

**Def.** The pre-template is *robust* if, for any valid focussed context \( \hat{\mathcal{C}} \) with a rewrite token, one of the following holds:

1. \( \hat{\mathcal{C}}[g_1] \xrightarrow{+} \hat{N} \) and \( \hat{\mathcal{C}}[g_2] \xrightarrow{*} \hat{N}_2 \)
   for two stuck states \( \hat{N}_1, \hat{N}_2 \)

2. \( \hat{\mathcal{C}}[g_1] \xrightarrow{+} \hat{\mathcal{C}}'[g_1'] \) and \( \hat{\mathcal{C}}[g_2] \xrightarrow{*} \hat{\mathcal{C}}'[g_2'] \)
   for two hypernets \( g_1', g_2' \) s.t. \( g_1' \triangleright g_2' \)
   and a valid focussed context \( \hat{\mathcal{C}}' \)
   such that the token is not in rewrite status

---

**Input-safe** ✓  
**Output-closed** ✓  
**Robust (⊗)**  

---

PERR (2019)  
Local Reasoning for Robust Observational Equivalence  
4 / 6
**The Characterisation Theorem**

**Def.** The pre-template is *robust* if, for any valid focussed context $\hat{C}$ with a rewrite token, one of the following holds:

1. $\hat{C}[g_1] \Rightarrow^+ \hat{N}$ and $\hat{C}[g_2] \Rightarrow^* \hat{N}_2$ for two stuck states $\hat{N}_1, \hat{N}_2$

2. $\hat{C}[g_1] \Rightarrow^+ \hat{C}'[g_1']$ and $\hat{C}[g_2] \Rightarrow^* \hat{C}'[g_2']$ for two hypernets $g_1', g_2'$ s.t. $g_1' \downarrow g_2'$ and a valid focussed context $\hat{C}'$ such that the token is not in rewrite status

**Input-safe ✓**
**Output-closed ✓**
**Robust (⊗) ✓**
The Characterisation Theorem

**Def.** The pre-template is *robust* if, for any valid focussed context \( \hat{C} \) with a rewrite token, one of the following holds:

1. \( \hat{C}[g_1] \to^+ \hat{N} \) and \( \hat{C}[g_2] \to^* \hat{N}_2 \)
   for two stuck states \( \hat{N}_1, \hat{N}_2 \)

2. \( \hat{C}[g_1] \to^+ \hat{C}'[g_1'] \) and \( \hat{C}[g_2] \to^* \hat{C}'[g_2'] \)
   for two hypernets \( g_1', g_2' \) s.t. \( g_1' \triangleleft g_2' \)
   and a valid focussed context \( \hat{C}' \)
   such that the token is not in rewrite status

---

**Input-safe** ✓
**Output-closed** ✓
**Robust (;)** ?
The Characterisation Theorem

**Def.** The pre-template is *robust* if, for any valid focussed context $\hat{C}$ with a rewrite token, one of the following holds:

1. $\hat{C}[g_1] \rightarrow^+ \hat{N}$ and $\hat{C}[g_2] \rightarrow^* \hat{N}_2$ for two stuck states $\hat{N}_1, \hat{N}_2$

2. $\hat{C}[g_1] \rightarrow^+ \hat{C}'[g_1']$ and $\hat{C}[g_2] \rightarrow^* \hat{C}'[g_2']$ for two hypernets $g_1', g_2'$ s.t. $g_1' \triangleleft g_2'$ and a valid focussed context $\hat{C}'$ such that the token is not in rewrite status

**Input-safe ✓**

**Output-closed ✓**

**Robust (;) ✓**
Def. The pre-template is *robust* if, for any valid *focussed context* \( \hat{C} \) with a rewrite token, one of the following holds:

1. \( \hat{C}[g_1] \rightarrow^+ \hat{N} \) and \( \hat{C}[g_2] \rightarrow^* \hat{N}_2 \)
   for two stuck states \( \hat{N}_1, \hat{N}_2 \)

2. \( \hat{C}[g_1] \rightarrow^+ \hat{C}'[g_1'] \) and \( \hat{C}[g_2] \rightarrow^* \hat{C}'[g_2'] \)
   for two hypernets \( g_1', g_2' \) s.t. \( g_1' \triangleright g_2' \)
   and a valid *focussed context* \( \hat{C}' \)
   such that the token is not in rewrite status

Input-safe ✓
Output-closed ✓
Robust (count) ?
The Characterisation Theorem

**Def.** The pre-template is *robust* if, for any valid focussed context $\hat{C}$ with a rewrite token, one of the following holds:

1. $\hat{C}[g_1] \rightarrow^+ \hat{N}$ and $\hat{C}[g_2] \rightarrow^* \hat{N}_2$ for two stuck states $\hat{N}_1, \hat{N}_2$

2. $\hat{C}[g_1] \rightarrow^+ \hat{C}'[g_1']$ and $\hat{C}[g_2] \rightarrow^* \hat{C}''[g_2']$ for two hypernets $g_1', g_2'$ s.t. $g_1' \ll g_2'$ and a valid focussed context $\hat{C}'$ such that the token is not in rewrite status

**Input-safe** ✓
**Output-closed** ✓
**Robust (count)** ❌

PERR (2019)
Local Reasoning for Robust Observational Equivalence
An Application of The Characterisation Theorem

\[(\lambda x\, \lambda f\, (\lambda z\, !x)(f \, () ))(\text{ref 1}) \equiv? \lambda g\, (\lambda y\, 1)(f \, ())\]
∀C. C[(\lambda x. \lambda f. (\lambda z. ! x)(f ())) (ref 1)] \simeq? C[\lambda g. (\lambda y. 1)(g ())]
An Application of The Characterisation Theorem

∀C. C[(\lambda x. \lambda f. (\lambda z. !x)(f ())(ref 1))] \equiv? C[\lambda g. (\lambda y. 1)(g ())]
An Application of The Characterisation Theorem

∀C. C[(\lambda x. \lambda f. (\lambda z. !x)(f () ))(\text{ref 1})] \equiv? C[\lambda g. (\lambda y. 1)(g () )]

Characterisation Theorem.
Robust templates induce observational equivalences

Robust Template 1: $\beta$-Law
An Application of The Characterisation Theorem

∀C. C[(\lambda x. \lambda f. (\lambda z. x)(f ())) (\text{ref } 1)] \simeq ? C[\lambda g. (\lambda y. 1)(g ())] 

Characterisation Theorem.
Robust templates induce observational equivalences

Robust Template 1: \(\beta\)-Law
Robust Template 2: ‘Ref’ Rewrite

\[ \triangleleft \text{R} \]
An Application of The Characterisation Theorem

∀C. C[((λx.λf.(λz.x)(f () ))(ref 1)) ≃? C[λg.(λy.1)(g () )]]

Characterisation Theorem.
Robust templates induce observational equivalences

Robust Template 1: β-Law
Robust Template 2: ‘Ref’ Rewrite
Robust Template 3: ‘!’ Rewrite
An Application of The Characterisation Theorem

∀C. C[(λx. λf. (λz. !x)(f ())) (ref 1)] ≃? C[λg. (λy. 1)(g ())]  

**Characterisation Theorem.**
Robust templates induce observational equivalences

*Robust Template 1: β-Law*
*Robust Template 2: ‘Ref’ Rewrite*
*Robust Template 3: ‘!’ Rewrite*
*Robust Template 4: Extend Thunk*
An Application of The Characterisation Theorem

\[ \forall C. C[(\lambda x. \lambda f. (\lambda z. ! x)(f \langle x \rangle))(\text{ref} 1)] \approx? C[\lambda g. (\lambda y. 1)(g \langle y \rangle)] \]
An Application of The Characterisation Theorem

\[ \forall C. C[(\lambda x. \lambda f. (\lambda z. !x)(f (\cdot))(\text{ref 1})] \approx? C[\lambda g. (\lambda y. 1)(g (\cdot))] \]
An Application of The Characterisation Theorem

\[ \forall C. C[(\lambda x. \lambda f. (\lambda z. ! x)(f () ))(\text{ref 1})] \simeq? C[\lambda g. (\lambda y. 1)(g () )] \]
An Application of The Characterisation Theorem

\[ \forall C. C[(\lambda x. \lambda f. (\lambda z. !x)(f (@))(\text{ref } 1)] \approx? C[\lambda g. (\lambda y. 1)(g (@))] \]
An Application of The Characterisation Theorem

\[ \forall C. C[(\lambda x. \lambda f. (\lambda z. !x)(f (\_)))(\text{ref } 1)] \sim? C[\lambda g. (\lambda y. 1)(g (\_))] \]
An Application of The Characterisation Theorem

\[ \forall C.C[(\lambda x. \lambda f. (\lambda z. !x)(f (\_)))(\text{ref } 1)] \simeq ? C[\lambda g. (\lambda y. 1)(g (\_))] \]
An Application of The Characterisation Theorem

\[ \forall C. C[(\lambda x. \lambda f. (\lambda z. !x)(f (\_))) \text{ref 1}] \simeq? C[\lambda g. (\lambda y. 1)(g \_)] \]
An Application of The Characterisation Theorem

\[ \forall C. C[(\lambda x. \lambda f. (\lambda z. x)(f (\_)))(\text{ref 1})] \approx? C[\lambda g. (\lambda y. 1)(g (\_))] \]
∀C. C[(\lambda x. \lambda f. (\lambda z. !x)(f (\_))(ref 1)) \cong? C[\lambda g. (\lambda y. 1)(g (\_))]]
An Application of The Characterisation Theorem

$$\forall C. C[(\lambda x. \lambda f. (\lambda z. !x)(f (\_))))(\text{ref } 1)] \simeq? \ C[\lambda g. (\lambda y. 1)(g (\_))$$
∀C. C[((λx. λf. (λz. !x)(f ())) (ref 1))] ≃？ C[λg. (λy. 1) (g ())]

An Application of The Characterisation Theorem
An Application of The Characterisation Theorem

\[ \forall C. C[(\lambda x. \lambda f. (\lambda z. ! x)(f \, (\, )))(\text{ref} \, 1)] \simeq C[\lambda g. (\lambda y. 1)(g \, (\, ))] \]
An Application of The Characterisation Theorem

\[(\lambda x. \lambda f. (\lambda z. !x)(f () ))(\text{ref 1}) \equiv \checkmark \lambda g. (\lambda y. 1)(f () )\]
Closing Remarks & Further Work
Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓

SPARTAN
Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓
   \textit{SPARTAN}

Parameterisation of contexts for observational equivalence ✓
   \textit{Binding-free & Robustness}
Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓

SPARTAN

Parameterisation of contexts for observational equivalence ✓

Binding-free & Robustness

Characterisation of effects from POV of equational properties of the language ✓

Characterisation Theorem
Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓
  \texttt{SPARTAN}

Parameterisation of contexts for observational equivalence ✓
  \textit{Binding-free & Robustness}

Characterisation of effects from POV of equational properties of the language ✓
  \textit{Characterisation Theorem}

Control operations ？

Non-deterministic operations ？

Concurrency ？

Type system ？
Closing Remarks & Further Work

Common framework for reasoning with programming languages with effects ✓

Spartan

Parameterisation of contexts for observational equivalence ✓

Binding-free & Robustness

Characterisation of effects from POV of equational properties of the language ✓

Characterisation Theorem

Control operations ？

Non-deterministic operations ？

Concurrency ？

Type system ？

Spartan Visualiser & Paper:

tnttodda.github.io/
Spartan-Visualiser