Step-indexed Biorthogonality

Andrew Pitts

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Workshop on 'parametricity *or* logical relations'

Question: What are

logical relations parameterised by relations

good for?

Workshop on ‘parametricity *or* logical relations’

Question: What are

logical relations parameterised by relations

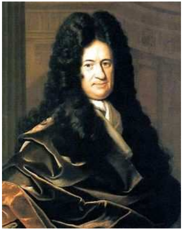
good for?

One answer: proving properties of

contextual equivalence of programs.

Contextual equivalence

Two phrases of a programming language are ("Morris style") contextually equivalent ($\cong_{\mathbf{ctx}}$) if occurrences of the first phrase in any program can be replaced by the second phrase without affecting the observable results of executing the program.



Gottfried Wilhelm Leibniz (1646–1716),
identity of indiscernibles:
duo quaedam communes proprietates eorum nequaquam possit
(two distinct things cannot have all their properties in common).

Contextual equiv. without contexts

Principle [*cf.* Gordon, Lassen (1998)]
$\cong_{\mathbf{ctx}}$ (defined conventionally, using contexts) is the greatest compatible & adequate relation.

Contextual equiv. without contexts

Principle [*cf.* Gordon, Lassen (1998)]
$\cong_{\mathbf{ctx}}$ (defined conventionally, using contexts) is the greatest compatible & adequate relation.

Impredicative characterization ☹

Important nonetheless, because:

- 'program-with-hole' too concrete for languages with
  - binders
  - complicated judgement forms where we have to be careful about compatibility/substitutivity properties.

Contextual equiv. without contexts

Principle [*cf.* Gordon, Lassen (1998)]
$\cong_{\mathbf{ctx}}$ (defined conventionally, using contexts) is the greatest compatible & adequate relation.

Impredicative characterization

Important nonetheless, because:

- 'program-with-hole' too concrete for languages with
  - binders
  - complicated judgement forms where we have to be careful about compatibility/substitutivity properties.
- emphasises the coinductive nature of $\cong_{\mathbf{ctx}}$—expect it to be rich in properties, e.g. relational parametricity...

$\cong_{ctx}$ is relationally parametric

For example: $\emptyset \vdash \Lambda\alpha.e_1 \cong_{ctx} \Lambda\alpha.e_2 : \forall\alpha.\tau$
if and only if
for all τ_1, τ_2 and all 'good' relations $\tau_1 \overset{r}{\leftrightarrow} \tau_2$,
$e_1[\tau_1/\alpha]$ and $e_2[\tau_2/\alpha]$ are related by $\tau[\tau_1/\alpha] \overset{\tau[r/\alpha]}{\leftrightarrow} \tau[\tau_2/\alpha]$

$\cong_{\mathbf{ctx}}$ is relationally parametric

Consequences:

- type-directed extensionality properties of $\cong_{\mathbf{ctx}}$
- functoriality/naturality w.r.t. indexes
- universal properties of recursive datatypes

Functional programming is category theory without equations—working up to $\cong_{\mathbf{ctx}}$, we get the equations 'for free'.

Template for syntactic relational parametricity

1. Define a (paramaterised) binary logical relation $\sim$ between programming language expressions. Make sure $\sim$ is compatible and adequate by construction; so $\sim \subseteq \cong_{\mathbf{ctx}}$.

Template for syntactic relational parametricity

1. Define a (paramaterised) binary logical relation $\sim$ between programming language expressions. Make sure $\sim$ is compatible and adequate by construction; so $\sim \subseteq \cong_{\mathbf{ctx}}$.

2. Prove $\sim$ is reflexive (Fundamental Property of LRs).

Template for syntactic relational parametricity

1. Define a (paramaterised) binary logical relation $\sim$ between programming language expressions. Make sure $\sim$ is compatible and adequate by construction; so $\sim \subseteq \cong_{\mathbf{ctx}}$.

2. Prove $\sim$ is reflexive (Fundamental Property of LRs).

3. Prove that $\sim$ respects $\cong_{\mathbf{ctx}}$. Hence by 2, $\cong_{\mathbf{ctx}} = (\cong_{\mathbf{ctx}} \circ id) \subseteq (\cong_{\mathbf{ctx}} \circ \sim) \subseteq \sim$.

Template for syntactic relational parametricity

1. Define a (paramaterised) binary logical relation $\sim$ between programming language expressions. Make sure $\sim$ is compatible and adequate by construction; so $\sim \subseteq \cong_{\mathbf{ctx}}$.

2. Prove $\sim$ is reflexive (Fundamental Property of LRs).

3. Prove that $\sim$ respects $\cong_{\mathbf{ctx}}$. Hence by 2, $\cong_{\mathbf{ctx}} = (\cong_{\mathbf{ctx}} \circ id) \subseteq (\cong_{\mathbf{ctx}} \circ \sim) \subseteq \sim$.

4. Deduce properties of $\cong_{\mathbf{ctx}}$ from 1+3.

   Become rich and famous.

Template for syntactic relational parametricity

1. Define a (paramaterised) binary logical relation $\sim$ between programming language expressions. Make sure $\sim$ is compatible and adequate by construction; so $\sim \subseteq \cong_{\mathbf{ctx}}$.

2. Prove $\sim$ is reflexive (Fundamental Property of LRs).

3. Prove that $\sim$ respects $\cong_{\mathbf{ctx}}$. Hence by 2, $\cong_{\mathbf{ctx}} = (\cong_{\mathbf{ctx}} \circ \mathit{id}) \subseteq (\cong_{\mathbf{ctx}} \circ \sim) \subseteq \sim$.

Difficulties:

- How do we ensure relations are 'respectful' (3)?
- Term-level recursion makes 2 moderately hard.
- Type-level recursion (with —ve occurrences) makes 1 very hard.

Template for syntactic relational parametricity

1. Define a (paramaterised) binary logical relation $\sim$ between programming language expressions. Make sure $\sim$ is compatible and adequate by construction; so $\sim \subseteq \cong_{\mathbf{ctx}}$.

2. Prove $\sim$ is reflexive (Fundamental Property of LRs).

3. Prove that $\sim$ respects $\cong_{\mathbf{ctx}}$. Hence by 2,
   $\cong_{\mathbf{ctx}} = (\cong_{\mathbf{ctx}} \circ id) \subseteq (\cong_{\mathbf{ctx}} \circ \sim) \subseteq \sim$.

Difficulties:

- How do we ensure relations are 'respectful' (3)? Ans: $(\_)^{\perp\perp}$
- Term-level recursion makes 2 moderately hard. Ans: $(\_)^{\perp\perp}$
- Type-level recursion makes 1 very hard.

Biorthogonal-closed relations

Given sets T, T^* and relation $\langle \_ | \_ \rangle \subseteq T^* \times T$, we get

a Galois connection
$$\begin{tikzcd} \mathcal{P}(T \times T)^{op} \arrow[rr, bend left, "(\_)^{\perp}"] & & \mathcal{P}(T^* \times T^*) \arrow[ll, bend left, "(\_)^{\perp}"] \end{tikzcd} :$$

$$(E, E') \in r^{\perp} \triangleq (\forall (e, e') \in r)\ \langle E | e \rangle \Leftrightarrow \langle E' | e' \rangle$$
$$(e, e') \in R^{\perp} \triangleq (\forall (E, E') \in R)\ \langle E | e \rangle \Leftrightarrow \langle E' | e' \rangle$$

Biorthogonal-closed relations

Given sets $\boldsymbol{T}$, $\boldsymbol{T^*}$ and relation $\langle \_ | \_ \rangle \subseteq T^* \times T$, we get

a Galois connection
$$\begin{tikzcd} \mathcal{P}(T \times T)^{op} \arrow[rr, bend left, "(\_)^{\perp}"] & & \mathcal{P}(T^* \times T^*) \arrow[ll, bend left, "(\_)^{\perp}"] \end{tikzcd} :$$

$$\boxed{\begin{array}{c} r_1 \subseteq r_2 \Rightarrow r_2^{\perp} \subseteq r_1^{\perp} \\ R \subseteq r^{\perp} \Leftrightarrow r \subseteq R^{\perp} \end{array}}$$

Biorthogonal-closed relations

Given sets T, T^* and relation $\langle \_ | \_ \rangle \subseteq T^* \times T$, we get

a Galois connection
$$
\begin{tikzcd}
\mathcal{P}(T \times T)^{op} \arrow[rr, bend left, "(\_)^{\perp}"] & & \mathcal{P}(T^* \times T^*) \arrow[ll, bend left, "(\_)^{\perp}"]
\end{tikzcd}
$$
:

$\perp\perp$-closed relations: $r = r^{\perp\perp}$

Apply with T = terms, T^* = evaluation contexts, $\langle \_ | \_ \rangle$ = termination (+observation).

For call-by-value, we have $V \subseteq T$ and use

'valuable' relations: $r = (r|_V)^{\perp\perp}$

Biorthogonal-closed relations

Given sets T, T^* and relation $\langle \_ | \_ \rangle \subseteq T^* \times T$, we get

a Galois connection
$$\begin{tikzcd} \mathcal{P}(T \times T)^{op} \arrow[rr, bend left, "(\_)^{\perp}"] & & \mathcal{P}(T^* \times T^*) \arrow[ll, bend left, "(\_)^{\perp}"] \end{tikzcd}$$
:

$\perp\perp$-closed relations: $r = r^{\perp\perp}$

Apply with T = terms, T^* = evaluation contexts, $\langle \_ | \_ \rangle$ = termination (+observation).

Using $\perp\perp$-closed relations to define $\sim$, magically (?) we get respectfulness (for step 3) and admissibility properties for term-level recursions (for step 2).

Template for syntactic relational parametricity

1. Define a (paramaterised) binary logical relation $\sim$ between programming language expressions. Make sure $\sim$ is compatible and adequate by construction; so $\sim \subseteq \cong_{\mathbf{ctx}}$.

2. Prove $\sim$ is reflexive (Fundamental Property of LRs).

3. Prove that $\sim$ respects $\cong_{\mathbf{ctx}}$. Hence by 2,
$\cong_{\mathbf{ctx}} = (\cong_{\mathbf{ctx}} \circ \mathit{id}) \subseteq (\cong_{\mathbf{ctx}} \circ \sim) \subseteq \sim$.

Difficulties:

- How do we ensure relations are 'respectful' (3)? Ans: $(\_)^{\perp\perp}$
- Term-level recursion makes 2 moderately hard. Ans: $(\_)^{\perp\perp}$
- Type-level recursion makes 1 very hard.

Template for syntactic relational parametricity

1. Define a (paramaterised) binary logical relation $\sim$ between programming language expressions. Make sure $\sim$ is

$\cong_{\mathbf{ctx}}$.

Biorthogonal-closed relations were first used for relational parametricity results in [AMP+Stark, HOOTS book, pp 227–273 (CUP 1998)] & [AMP, MSCS 10(2000) 321-359]; but the technique goes back (at least) to Girard's normalization proof for Linear Logic, to Krivine, to ...

2.

3.

$=_{\mathbf{ctx}} = (=_{\mathbf{ctx}} \circ id) \subseteq (=_{\mathbf{ctx}} \circ \sim) \subseteq \sim.$

Difficulties:

- How do we ensure relations are 'respectful' (3)? Ans: $(\_)^{\perp\perp}$
- Term-level recursion makes 2 moderately hard. Ans: $(\_)^{\perp\perp}$
- Type-level recursion makes 1 very hard.

Template for syntactic relational parametricity

1. Define a (paramaterised) binary logical relation $\sim$ between programming language expressions. Make sure $\sim$ is compatible and adequate by construction; so $\sim \subseteq \cong_{\mathbf{ctx}}$.

Difficulties:

- Type-level recursion makes 1 very hard.

Simple example of mixed variance, type-level recursion—no types!

Untyped cbv λ-calculus

$$
\begin{array}{lrcl}
\text{Values} & v \in V & ::= & x, f \\
 & & \mid & \mathbf{fun}(f\,x = e) \\
\text{Expressions} & e \in \Lambda & ::= & v \\
 & & \mid & v\,v \\
 & & \mid & \mathbf{let}\,x = e\,\mathbf{in}\,e \\
\text{Frame stacks} & E \in \Lambda^* & ::= & \mathbf{Id} \\
 & & \mid & E \circ (x \to e)
\end{array}
$$

$$
\text{Termination} \quad \frac{}{\langle \mathbf{Id} | v \rangle} \quad \frac{\langle E | e[v/x] \rangle}{\langle E \circ (x \to e) | v \rangle} \quad \frac{\langle E \circ (x \to e) | e' \rangle}{\langle E |\, \mathbf{let}\,x = e\,\mathbf{in}\,e' \rangle} \quad \text{etc.}
$$

Simple example of mixed variance, type-level recursion—no types!

Want relation $\lhd$ on closed values/expressions/frame stacks satisfying:

- For $v = \mathbf{fun}(f\,x = e)$ & $v' = \mathbf{fun}(f\,x = e')$:
$$v \lhd v' \;\Leftrightarrow\; (\forall v_1, v_1')\; v_1 \lhd v_1' \Rightarrow e[v/f, v_1/x] \lhd e'[v'/f, v_1'/x]$$
- $e \lhd e' \;\Leftrightarrow\; (\forall E, E')\; E \lhd E' \Rightarrow \langle E|e\rangle \Rightarrow \langle E'|e'\rangle$
- $E \lhd E' \;\Leftrightarrow\; (\forall v, v')\; v \lhd v' \Rightarrow \langle E|v\rangle \Rightarrow \langle E'|v'\rangle$

For then after some work we get $\lhd \cap \lhd^{op} = \cong_{\mathbf{ctx}}$ and become rich & famous.

Simple example of mixed variance, type-level recursion—no types!

Want relation $\lhd$ on closed values/expressions/frame stacks satisfying:

- For $v = \mathbf{fun}(f\,x = e)$ & $v' = \mathbf{fun}(f\,x = e')$:
  $$v \lhd v' \;\Leftrightarrow\; (\forall v_1, v_1')\; v_1 \lhd v_1' \Rightarrow e[v/f, v_1/x] \lhd e'[v'/f, v_1'/x]$$
- $e \lhd e' \;\Leftrightarrow\; (\forall E, E')\; E \lhd E' \Rightarrow \langle E | e \rangle \Rightarrow \langle E' | e' \rangle$
- $E \lhd E' \;\Leftrightarrow\; (\forall v, v')\; v \lhd v' \Rightarrow \langle E | v \rangle \Rightarrow \langle E' | v' \rangle$

But how to define such a relation $\lhd$?
(Note the occurrence of $\lhd \Rightarrow \lhd$.)

Simple example of mixed variance, type-level recursion—no types!

Step-indexed version the logical relation:

- For $v = \mathbf{fun}(f\,x = e)$ & $v' = \mathbf{fun}(f\,x = e')$:
  $$v \lhd_n v' \triangleq (\forall m < n)(\forall v_1, v_1')\ v_1 \lhd_m v_1' \Rightarrow e[v/f, v_1/x] \lhd_m e'[v'/f, v_1'/x]$$
- $$e \lhd_n e' \triangleq (\forall m \leq n)(\forall E, E')\ E \lhd_m E' \Rightarrow \langle E|e\rangle_m \Rightarrow \langle E'|e'\rangle$$
- $$E \lhd_n E' \triangleq (\forall m \leq n)(\forall v, v')\ v \lhd_m v' \Rightarrow \langle E|v\rangle_m \Rightarrow \langle E'|v'\rangle$$

where $\langle E|e\rangle_n$ means termination in at most n steps.

Simple example of mixed variance, type-level recursion—no types!

Step-indexed version the logical relation:

- For $v = \mathbf{fun}(f\,x = e)$ & $v' = \mathbf{fun}(f\,x = e')$:
  $$v \lhd_n v' \triangleq (\forall m < n)(\forall v_1, v_1')\; v_1 \lhd_m v_1' \Rightarrow e[v/f, v_1/x] \lhd_m e'[v'/f, v_1'/x]$$
- $$e \lhd_n e' \triangleq (\forall m \leq n)(\forall E, E')\; E \lhd_m E' \Rightarrow \langle E | e \rangle_m \Rightarrow \langle E' | e' \rangle$$
- $$E \lhd_n E' \triangleq (\forall m \leq n)(\forall v, v')\; v \lhd_m v' \Rightarrow \langle E | v \rangle_m \Rightarrow \langle E' | v' \rangle$$

defined by well-founded recursion for $(\omega, <)$.

Simple example of mixed variance, type-level recursion—no types!

Step-indexed version the logical relation:

Theorem. $\cong_{\mathbf{ctx}} = \lhd \cap \lhd^{op}$, where $\lhd = \bigcup_{n<\omega} \lhd_n$.

[For details, see tutorial by AMP in Ahmed, Benton, Birkedal and Hofmann (eds), *Modelling, Controlling and Reasoning About State*, Dagstuhl Seminar Proceedings 10351 (2010).]

Step-indexed relations

- When Appel & McAllester first introduced the technique, it seemed too intensional to be useful for proving (extensional) properties of $\cong_{\mathbf{ctx}}$.
  Ahmed [ESOP 2006] proved otherwise.

Step-indexed relations

- When Appel & McAllester first introduced the technique, it seemed too intensional to be useful for proving (extensional) properties of $\cong_{\mathbf{ctx}}$.

  Ahmed [ESOP 2006] proved otherwise.
- Benton-Hur [ICFP 2009], Dreyer-Neis-Birkedal [ICFP 2010] combined step-indexing with use of biorthogonal closure.

Step-indexed relations

- When Appel & McAllester first introduced the technique, it seemed too intensional to be useful for proving (extensional) properties of $\cong_{\mathbf{ctx}}$.

  Ahmed [ESOP 2006] proved otherwise.
- Benton-Hur [ICFP 2009], Dreyer-Neis-Birkedal [ICFP 2010] combined step-indexing with use of biorthogonal closure.
- Big win over previous methods (e.g. do not have to change the language by adding syntactic projections).

  I consider Exercise 7.8.1 in my chapter in ATTAPL (marked [$\ast\ast\ast\ast\ldots$]) to be answered!

7.8.1 EXERCISE [★★★★..., ↛]: Extend F_{ML} with *isorecursive types*, $\mu\mathsf{X}.\mathsf{T}$, as in Figure 20-1 of *TAPL*, Chapter 20. By finding an operationally based logical relation as in §7.6 or otherwise, try to prove the kind of properties of contextual equivalence for this extended language that we developed for F_{ML} in this chapter. (For the special case of iso-recursive types $\mu\mathsf{X}.\mathsf{T}$ for which T contains no negative occurrences of X, albeit for a non-strict functional language, see Johann (2002). The generalized ideal model of recursive polymorphic in Vouillon and Melliès (2004) uses the same kind of Galois connection as we used in §7.6 and may well shed light on this exercise. Recent work by Sumii and Pierce [2005] is also relevant.) □

Take-home messages

- Relational parametricity is a powerful tool for proving contextual equivalence of programs.

Take-home messages

- Relational parametricity is a powerful tool for proving contextual equivalence of programs.
- Biorthogonal-closed, step-indexed, syntactical relations provide an approach to relationally parametric characterisations of contextual equivalence which is simple and widely applicable.

Simple?

- $+$ Mathematically elementary.
- $+$ Works with the syntax as-is.

  (Do not have to define, or add, syntactic projections.)
- $+$ Uniform method that delvers a wide range of *general* properties of $\cong_{\mathbf{ctx}}$.

  (Extensionality, 'free theorems', unfolding recursion,. . .)
- $-$ Does not, in itself, help us to understand feature-specific properties of $\cong_{\mathbf{ctx}}$.

  (E.g. higher-order local stores.)
- $-$ Index manipulation! How to place 'guards'?

  (Denotational understanding is still important: see the work of Nakano, Birkedal *et al*, . . . , on type theory & logic for guarded recursion.)

Widely applicable?

Two examples of the robustness of the approach:

1. Correctness of representation of nominal algebraic data mod α by FreshML $\cong_{\mathrm{ctx}}$, via extensionality property of name-abstractions $\langle a \rangle e$:

$$\langle a \rangle e \cong_{\mathrm{ctx}} \langle a' \rangle e' \;\Leftrightarrow\; (\text{И} a'')\; (a\ a'') \cdot e \cong_{\mathrm{ctx}} (a'\ a'') \cdot e'$$

Can apply the SIBCLR template, but instead of sets and relations, use nominal sets and finitely supported relations.

See [AMP, chapter 10 of *Nominal Sets* book, to appear].

Widely applicable?

Two examples of the robustness of the approach:

2. *Step-indexed relational reasoning for countable nondeterminism* [Schwinghammer & Birkedal, CSL 2011]. Previous approaches fall over because

recursion + choice $\Rightarrow$ unbounded non-determinism $\Rightarrow$ non-continuous denotations.

SIBCLR template works beautifully to prove properties of must-$\cong_{\mathbf{ctx}}$, using $(\omega_1, <)$ for the steps instead of $(\omega, <)$.

Widely applicable?

More examples needed:

- Properties of $\cong_{\mathbf{ctx}}$ for lazy functional programming—the SIBCLR template should cope well with recursive nature of heaps.
  - call-by-need $\cong_{\mathbf{ctx}}$ = call-by-name $\cong_{\mathbf{ctx}}$
  - monadic encapsulation of effects
  - ?
- HOT languages with concurrency features.
- ? (over to you)

Development/support within interactive theorem-provers?