Techniques for proving contextual equivalence

Contextual equivalence

Two phrases of a programming language are ("Morris style") contextually equivalent ($\approx_{ctx}$) 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): two mathematical objects are equal if there is no test to distinguish them.
The semantics of programs only depends on their abstract syntax (parse trees) modulo renaming of bound identifiers ($\alpha$-equivalence, $=_{\alpha}$).

E.g. definition & properties of OCaml typing relation $\Gamma \vdash M : \tau$ are simpler if we identify $M$ up to $=_{\alpha}$.
The semantics of programs only depends on their abstract syntax (parse trees) modulo renaming of bound identifiers ($\alpha$-equivalence, $\equiv_{\alpha}$).

So it pays to formulate program equivalences using mathematical notions that respect $\alpha$-equivalence.

But filling holes in contexts does not respect $\equiv_{\alpha}$:

$$\text{fun } x \rightarrow (\_ ) \equiv_{\alpha} \text{fun } y \rightarrow (\_ )$$

and

$$x \equiv_{\alpha} x$$

but

$$\text{fun } x \rightarrow x \not\equiv_{\alpha} \text{fun } y \rightarrow x$$
Expression relations

Language’s typing relation

\[ \Gamma \vdash M : \tau \]

dictates the form of relations like contextual equivalence:

Define an expression relation to be any set \( \mathcal{E} \) of tuples \((\Gamma, M, M', \tau)\) satisfying:

\[ (\Gamma \vdash M \in \mathcal{E} M' : \tau) \implies (\Gamma \vdash M : \tau) \& (\Gamma \vdash M' : \tau) \]
Operations on expression relations

Composition $\mathcal{E}_1, \mathcal{E}_2 \mapsto \mathcal{E}_1; \mathcal{E}_2$:

\[
\Gamma \vdash M \mathcal{E}_1 M' : \tau \quad \Gamma \vdash M' \mathcal{E}_2 M'' : \tau \\
\Gamma \vdash M (\mathcal{E}_1; \mathcal{E}_2) M'' : \tau
\]

Reciprocation $\mathcal{E} \mapsto \mathcal{E}^\circ$:

\[
\Gamma \vdash M \mathcal{E} M' : \tau \\
\Gamma \vdash M' \mathcal{E}^\circ M : \tau
\]

Identity $\text{Id}$:

\[
\Gamma \vdash M : \tau \\
\Gamma \vdash M \text{Id} M : \tau
\]

Compatible refinement $\mathcal{E} \mapsto \hat{\mathcal{E}}$:

\[
\Gamma \vdash M_1 : \tau \rightarrow \tau' \quad M_2 : \tau \\
\Gamma \vdash M_1 M_2 : \tau'
\]
Compatible refinement $\mathcal{E} \leftrightarrow \hat{\mathcal{E}}$:

\[
\frac{\Gamma \vdash M_1 \mathcal{E} M'_1 : \tau \rightarrow \tau'}{\Gamma \vdash M_1 \mathcal{E} M'_1 : \tau \rightarrow \tau'} \quad \frac{\Gamma \vdash M_2 \mathcal{E} M'_2 : \tau}{\Gamma \vdash M_2 \mathcal{E} M'_2 : \tau'} \quad \frac{\Gamma \vdash M_1 M_2 \hat{\mathcal{E}} M'_1 M'_2 : \tau'}{\Gamma \vdash M_1 M_2 \hat{\mathcal{E}} M'_1 M'_2 : \tau'}
\]

\[
\Gamma, x : \tau \vdash M : \tau' \\
\frac{\Gamma \vdash (\text{fun } x \rightarrow M) : \tau \rightarrow \tau'}{\Gamma \vdash (\text{fun } x \rightarrow M) : \tau \rightarrow \tau'}
\]
Operations on expression relations

Compatible refinement $\mathcal{E} \mapsto \hat{\mathcal{E}}$:

\[
\begin{align*}
\Gamma & \vdash M_1 \mathcal{E} M'_1 : \tau \rightarrow \tau' & \Gamma & \vdash M_2 \mathcal{E} M'_2 : \tau \\
\Gamma & \vdash M_1 M_2 \hat{\mathcal{E}} M'_1M'_2 : \tau' \\
\Gamma, x : \tau & \vdash M \mathcal{E} M' : \tau' \\
\Gamma & \vdash (\text{fun } x \rightarrow M) \hat{\mathcal{E}} (\text{fun } x \rightarrow M') : \tau \rightarrow \tau' \\
\end{align*}
\]
### Operations on expression relations

**Compatible refinement** $\mathcal{E} \mapsto \widehat{\mathcal{E}}$: 

\[
\begin{align*}
\Gamma \vdash M_1 \mathcal{E} M'_1 : \tau \to \tau' & \quad \Gamma \vdash M_2 \mathcal{E} M'_2 : \tau \\
\Gamma \vdash M_1 M_2 \widehat{\mathcal{E}} M'_1 M'_2 : \tau' \\
\Gamma, x : \tau \vdash M \mathcal{E} M' : \tau' \\
\Gamma \vdash \text{fun} x \to M \widehat{\mathcal{E}} \text{fun} x \to M' : \tau \to \tau' \\
\Gamma \vdash M \mathcal{E} M' : \tau \\
\Gamma \vdash \text{ref} M \widehat{\mathcal{E}} \text{ref} M' : \tau \text{ref}
\end{align*}
\]

etc, etc (one rule for each typing rule)

---

### Contextual equiv. without contexts

**Theorem** [Gordon, Lassen (1998)]

$\approx_{\text{ctx}}$ (defined conventionally, using contexts) is the greatest compatible & adequate expression relation.

where an expression relation $\mathcal{E}$ is

- compatible if $\widehat{\mathcal{E}} \subseteq \mathcal{E}$
Theorem [Gordon, Lassen (1998)]

$\simeq_{ctx}$ (defined conventionally, using contexts) is the greatest compatible & adequate expression relation.

where an expression relation $\mathcal{E}$ is

- compatible if $\hat{\mathcal{E}} \subseteq \mathcal{E}$
- adequate if $\emptyset \vdash M \mathcal{E} M' : \text{bool} \Rightarrow \forall s.
  (\exists s'.\langle s, M \rangle \rightarrow^* \langle s', \text{true} \rangle) \iff
  (\exists s''.\langle s, M' \rangle \rightarrow^* \langle s'', \text{true} \rangle)

Precise definition varies according to the observational scenario. E.g. use “bisimulation” rather than “trace” based adequacy in presence of concurrency features.

Contextual equiv. without contexts

Definition

$\simeq_{ctx}$ is the union of all expression relations that are compatible and adequate.

where an expression relation $\mathcal{E}$ is

- compatible if $\hat{\mathcal{E}} \subseteq \mathcal{E}$
- adequate if $\emptyset \vdash M \mathcal{E} M' : \text{bool} \Rightarrow \forall s.
  (\exists s'.\langle s, M \rangle \rightarrow^* \langle s', \text{true} \rangle) \iff
  (\exists s''.\langle s, M' \rangle \rightarrow^* \langle s'', \text{true} \rangle)

So defined, $\simeq_{ctx}$ is also reflexive ($\mathcal{I}d \subseteq \mathcal{E}$), symmetric ($\mathcal{E}^\circ \subseteq \mathcal{E}$) and transitive ($\mathcal{E} ; \mathcal{E} \subseteq \mathcal{E}$).
<table>
<thead>
<tr>
<th></th>
<th>sound</th>
<th>complete</th>
<th>useful</th>
<th>general</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>(+)</td>
</tr>
<tr>
<td>CIU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domains</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Games</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical relns</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bisimulations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program logics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

sound: determines a compatible and adequate expression relation
complete: characterises $\cong_{\text{ctx}}$
useful: for proving programming “laws” & PL correctness properties
general: what PL features can be dealt with?

Brute force: sometimes compatible closure of $\{(M, M')\}$ is adequate, and hence $M \cong_{\text{ctx}} M'$.
(E.g. [AMP + Shinnwell, LMCS 4(1:4) 2008].)
<table>
<thead>
<tr>
<th></th>
<th>sound</th>
<th>complete</th>
<th>useful</th>
<th>general</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brute force</strong></td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>(+)</td>
</tr>
<tr>
<td><strong>CIU</strong></td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td><strong>Domains</strong></td>
<td>+</td>
<td>−</td>
<td>±</td>
<td>+</td>
</tr>
<tr>
<td><strong>Games</strong></td>
<td>+</td>
<td>+</td>
<td>±</td>
<td>−</td>
</tr>
<tr>
<td><strong>Logical relns</strong></td>
<td>+</td>
<td>+</td>
<td>±</td>
<td>−</td>
</tr>
<tr>
<td><strong>Bisimulations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Program logics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CIU: “Uses of Closed Instantiations”** [Mason-Talcott et al].

Equates open expressions if their closures w.r.t. substitutions have same reduction behaviour w.r.t. any frame stack.

**Domains:** traditional denotational semantics.

**Games:** game semantics [Abramsky, Malacaria, Hyland, Ong,...]
<table>
<thead>
<tr>
<th>Method</th>
<th>sound</th>
<th>complete</th>
<th>useful</th>
<th>general</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>(+)</td>
</tr>
<tr>
<td>CIU</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Domains</td>
<td>+</td>
<td>−</td>
<td>±</td>
<td>+</td>
</tr>
<tr>
<td>Games</td>
<td>+</td>
<td>+</td>
<td>±</td>
<td>−</td>
</tr>
<tr>
<td>Logical relns</td>
<td>+</td>
<td>±</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Bisimulations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program logics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Logical relations:** type-directed analysis of $\simeq_{ctx}$. At function types: relate functions if they send related arguments to related results.

Initially denotational [Plotkin,...], but now also operational [AMP, Birkedal-Harper-Crary, Ahmed, Johann-Voigtlaender,...].

**Bisimulations**—the legacy of concurrency theory:

$$M_1 \sim M_2$$

$$T \downarrow M_1'$$
### Bisimulations—the legacy of concurrency theory:

\[
M_1 \sim \sim M_2 \quad \text{(and symmetrically)}
\]

\[
\begin{array}{c}
M_1' \\
\sim \\
M_1' \sim \sim M_2'
\end{array}
\]
<table>
<thead>
<tr>
<th></th>
<th>sound</th>
<th>complete</th>
<th>useful</th>
<th>general</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>(+)</td>
</tr>
<tr>
<td>CIU</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Domains</td>
<td>+</td>
<td>−</td>
<td>±</td>
<td>+</td>
</tr>
<tr>
<td>Games</td>
<td>+</td>
<td>+</td>
<td>±</td>
<td>−</td>
</tr>
<tr>
<td>Logical relns</td>
<td>+</td>
<td>±</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Bisimulations</td>
<td>+</td>
<td>±</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Program logics</td>
<td>+</td>
<td>+</td>
<td>±</td>
<td>−</td>
</tr>
</tbody>
</table>

Program logics—e.g. higher-order Hoare logic
[Berger-Honda-Yoshida]

Beyond universal identities.

Q: How do we make sense of all these techniques and results?
Q: How do we make sense of all these techniques and results?
A: Category Theory can help!
For example...

### Example

Relational parametricity is a tool for proving contextual equivalences between polymorphic programs:

\[ \emptyset \vdash \Lambda \alpha \cdot e_1 \cong_{\text{ctx}} \Lambda \alpha \cdot e_2 : \forall \alpha. \tau \]

if and only if

for all \( \tau_1, \tau_2 \) and all “good” relations \( \tau_1 \leftrightarrow \tau_2 \),

\[ e_1[\tau_1/\alpha] \] and \( e_2[\tau_2/\alpha] \) are related by \( \tau[\tau_1/\alpha] \leftrightarrow \tau[\tau_2/\alpha] \)

Category theory guides us to

- “free theorems” via natural transformations [Wadler];
- universal properties of recursive datatypes: initial algebras / final coalgebras / Freyd’s free dialgebras [Hasagawa et al].
Wise words

“But once feasibility has been checked by an operational model, operational reasoning should be immediately abandoned; it is essential that all subsequent reasoning, calculation and design should be conducted in each case at the highest possible level of abstraction.”


Conclusion

- Operational models can support “reasoning, calculation and design” at a high level of abstraction—especially if we let Category Theory be our guide.
Research opportunities

The development of programming language theory based on contextual equivalences lags far behind the development of programming language design.

Type soundness results are two a penny, but correctness properties up to $\equiv_{\text{ctx}}$ are scarce (because they are hard!).

E.g. FP community is enthusiastically designing languages combining (higher rank) polymorphic types/kinds with recursively defined functions, datatypes, local state, subtyping, . . .

In many cases the relational parametricity properties of $\equiv_{\text{ctx}}$ are unknown.

Operationally-based work on programming language theory badly needs better tools for computer-aided proof.

“You want proof? I’ll give you proof!”