Equational Reasoning

“When are two expressions equal?” - Harper, PPPL

Syntactic Equality

\[ \lambda x. x + T = \lambda x. x + T \]

\[ \lambda y. y + T \neq \lambda x. x + T \]

\[ \lambda x. x + T \neq \lambda x. x + T \]

Equivalence

\[ \lambda x. x \equiv \lambda y. y \]

Kleene Equality

(for “observable types”, e.g. int, bool)

\[ e \equiv e' \text{ if } \exists v. e \rightarrow^* v \text{ and } e' \rightarrow^* v \]

e.g.

\[ 1 + 3 \approx 3 \approx 2 + 1 \]

What about \( \lambda x. \lambda y. x + y \) and \( \lambda x. \lambda y. y + x \)?

Observational Equivalence

Can’t do an “experiment” that can tell them apart

Only looking at results! e.g.

\[ 1 + 2 + 3 + 4 + 5 \approx 15 \]

\( \text{Quicksort} \approx \text{Bubble sort} \)

Expression context

\[ C ::= 0 | \lambda x. C | C \mid e \mid e \mid C | (C) | (e) | (e, C) \]

| \[ | C + C | \text{and} | C \mid \text{in} | C \mid \text{in} | C \]

Like evaluation

\[ \text{case } C \text{ of } \lambda x. e; y. e' \]

Case contexts but the hole can be anywhere!

\[ \text{case } e \text{ of } \lambda x. C; y. e' \]

\[ \lambda x. e \mid e(C) \]
Program Context: Exp. context that has type int and no free vars at outer level.

i.e., closed exp. of type int w/one hole

need to be able
to tell the result of the experiment!

Types for contexts: \((\alpha \triangleright P \triangleright D)\)

exp to fill hole has outer exp has type
type \(\tau\) under \(P, D\) \(\tau'\) under \(P, D\).

If \(\xi\) is a program context, \(C \cdot (\xi \triangleright P \triangleright D) \sim (\xi \triangleright \text{PD})\)

\[\vdash \xi \vdash e : \tau \quad \text{then} \quad \xi \vdash C(e) : \text{int}.\]

Observational equivalence

If \(\xi \vdash e : \tau\) and \(\xi \vdash e' : \tau\), then \(e \equiv e'\) if for all program contexts \(C : (\alpha \triangleright P \triangleright D) \sim (\alpha \triangleright \text{PD})\), \(C(e) \equiv C(e')\).

So is \(\lambda x. \lambda y. x + y \equiv \lambda x. \lambda y. y + x\)?

Reasoning about all program contexts is hard!

Logical Equivalence 

\(\sim\text{-derived inductively on the type}\)

\[\begin{align*}
e \sim \text{int e' } & \quad \text{if} \quad e = e' \\
e \sim \text{int e' } & \quad \text{if} \quad e = e' \\
e \sim \tau, \tau' e' & \quad \text{if} \quad \forall \xi, \xi' \text{ s.t. } e \equiv e', \text{we have } e \sim e', e' \sim_{\tau} \tau, \tau' e'. \\
e \sim \text{var, e' } & \quad \text{if} \quad \forall 
\text{ and admissible } R : \exists x \Rightarrow \text{bool}; \\
& \quad \text{use } R \text{ to compare at type } \alpha \\
e \sim \alpha \text{ e' } & \quad \text{if} \quad R(e, e').
\end{align*}\]
Definition. A relation \( R : \text{p} \times \text{p} \rightarrow \text{Bool} \) is admissible if

1. It respects observational eq. If \( R(e, e') \) and \( d \equiv e \) and \( d' \equiv e' \) then \( R(d, d') \).
2. "Closure under converse evaluation": If \( R(e, e') \), then:
   a. If \( d \rightarrow e \), then \( R(d, e') \)
   b. If \( d' \rightarrow e' \), then \( R(e, d') \)

\( R \) "can't tell apart things it shouldn't be able to."

Theorem 1. If \( \bullet \). \( e : \text{c} \) and \( \bullet \). \( e' : \text{c} \) then \( e \equiv e' \Leftrightarrow e = e' \).

(Oct also generalize loyal eq. to non-empty contexts)

Theorem 2. (Parametricity) If \( \bullet \). \( e : \text{c} \), then \( e \equiv e. \)

Theorem 3. Let \( \bullet \). \( e : \text{a} \rightarrow \text{a} \) and let \( \text{id} \equiv \lambda x. x \).

Then \( e \equiv \text{id} \) and by Thm 1, \( e = \text{id} \).

Proof. Let \( p, p' \) be types and \( R : \text{p} \times \text{p} \rightarrow \text{Bool} \) admissible. Suppose \( R(e, e') \).

WTS (by def of -) \( R(e[p], e_0, \text{id}[p])(e_0') \).

Because \( \text{id}[p']e_0' \rightarrow e_0' \), suffices to show \( R(e[p]e_0, e_0') \)

by Def 1.2

Because \( R(e_0, e_0') \) and \( e_0 \equiv e_0' \), by Def 1.1,

suffices to show \( e[p]e_0 = e_0. \)

By Thm 2, \( e \equiv \text{a} \rightarrow \text{a} \rightarrow e \). So, for any admissible \( S : \text{p} \times \text{p} \rightarrow \text{Bool} \),

if \( S(e_0, e_0) \), then \( S(e[p]e_0, e[p]e_0) \).

Let \( S(d, d') \) if \( d \equiv e_0 \equiv d' \).

Clearly \( S(e_0, e_0) \), so \( S(e[p]e_0, e[p]e_0) \Rightarrow e[p]e_0 = e_0. \)