Logical Relations for Program Equivalence

Filip Sieczkowski

Heriot-Watt University
+ University of Wroclaw

Program Equivalence / Approximation: Challenges + Applications

- Why do we care?
  - ↳ Program correctness (equational spec., e.g., $x + 0 \cong x$)
  - ↳ Compiler optimisations (shouldn't introduce new behaviours)
  - ↳ Representation independence
    (can we replace a simple implementation of a data struct. with an efficient one?)

- What do we need?
  - ↳ (Equivalence) Relation on program fragments
  - ↳ In a complex language, often w/o well-studied reduction theory

Defining Program Equivalence

Assumption: has values $\underline{n}$ for $n \in \mathbb{N}$

$p_1 \cong p_2$ iff $p_1 \to^* \underline{n}$ for all $n \in \mathbb{N}$ $\iff$ $p_2 \to^* \underline{n}$ for all $n \in \mathbb{N}$

print 3; return 1 $\not\cong$ print 5; return 1

Is this a good definition?

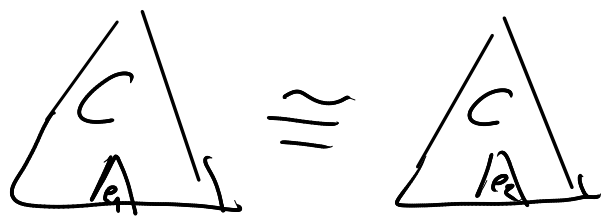
Maybe! We need to consider both the language & expected properties

Defining Program Equivalence

$p_1 \cong p_2$ iff $\forall n \in \mathbb{N}.\ p_1 \to^* \underline{n} \iff p_2 \to^* \underline{n}$

$e_1 \cong e_2$ iff $\forall C.\ C[e_1] \cong C[e_2]$ [Morris '68]

context



The Language: System T [Gödel, 1940s]

$$\sigma, \tau ::= \mathsf{nat} \mid \sigma \to \tau \qquad \text{(types)}$$

$$u, v ::= \mathsf{O} \mid \mathsf{S}\, v \mid \lambda x.\, e \mid x \qquad \text{(values)}$$

$$e ::= v \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid u\ v \mid \qquad \text{(expressions)}$$

$$\mathsf{rec}(u, v_1, v_2)$$

The Language: System T [Gödel, 1940s]

$$\sigma, \tau ::= \mathsf{nat} \mid \sigma \to \tau \qquad \text{(types)}$$

$$u, v ::= \mathsf{O} \mid \mathsf{S}\, v \mid \lambda x.\, e \mid x \qquad \text{(values)}$$

$$e ::= v \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid u\ v \mid \mathsf{rec}(u, v_1, v_2) \qquad \text{(expressions)}$$

$$\mathsf{let}\ x = v\ \mathsf{in}\ e \longmapsto e[{}^{v}/_{x}]$$

$$(\lambda x.\, e)\ v \longmapsto e[{}^{v}/_{x}]$$

$$\mathsf{rec}(\mathsf{O}, v_1, v_2) \longmapsto v_1$$

$$\mathsf{rec}(\mathsf{S}\, u, v_1, v_2) \longmapsto \mathsf{let}\ x = \mathsf{rec}(u, v_1, v_2)\ \mathsf{in}\ v_2\ x$$

$$\frac{e_1 \longmapsto e_2}{\mathsf{let}\ x = e_1\ \mathsf{in}\ e \longmapsto \mathsf{let}\ x = e_2\ \mathsf{in}\ e}$$

Typing System T

(values)

$$\Gamma \vdash 0 : \mathsf{nat}$$

$$\frac{\Gamma \vdash u : \mathsf{nat}}{\Gamma \vdash \mathsf{S}\, u : \mathsf{nat}}$$

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.\, e : \sigma \to \tau}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

(computations)

$$\frac{\Gamma \vdash v : \tau}{\Gamma \vdash v : \tau}$$

$$\frac{\Gamma \vdash e_1 : \sigma \qquad \Gamma, x : \sigma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau}$$

$$\frac{\Gamma \vdash u : \sigma \to \tau \qquad \Gamma \vdash v : \sigma}{\Gamma \vdash u\, v : \tau}$$

$$\frac{\Gamma \vdash u : \mathsf{nat} \qquad \Gamma \vdash v_1 : \sigma \qquad \Gamma \vdash v_2 : \sigma \to \sigma}{\Gamma \vdash \mathsf{rec}(u, v_1, v_2) : \sigma}$$

Example: Addition

$add \triangleq \lambda x, y.\ rec(x, y, S)$

$add : nat \to nat \to nat$

$add\ (S\ O)\ (S\ O) \longmapsto rec(S\ O, S\ O, S) \mapsto$

$\quad \mapsto S(rec(O, S\ O, S)) \longmapsto S(S\ O)$

$\underline{n} \triangleq S^n\ O$ for $n \in \mathbb{N}$

$add\ O\ x$ — can't reduce since open

$add\ x\ O$

Lemma: $x : \mathsf{nat} \vdash x + 0 \cong x$

Take $C : x : \mathsf{nat} \vdash \mathsf{nat} \rightsquigarrow \cdot \vdash \mathsf{nat}$

$$C[\overset{\text{add0}}{x + 0}] \overset{?}{\cong} C[x]$$

And we get stuck! Only way to proceed directly is induction over C, but inductive hypotheses are way too weak!

We need a stronger technique...

Logical Relations: the Plan

↳ Notion of contextual equivalence is nice, but difficult to use

↳ We need an alternative characterisation
to eliminate quantification over contexts

↳ To do this, find another way to deal with free variables...

↳ ... and with computations (almost) ...

↳ ... and start by only dealing with all closed ~~variables~~ values.

↳ To deal with these, leverage the structure of types

Logical Relation: the Definition

alt. def. for nat

$R\ 0\ 0$

$$\frac{R\ u\ v}{R\ (S\ u)\ (S\ v)}$$

$\mathcal{V}[\![-]\!]$ — type-indexed family of relations on closed values

$\mathcal{E}[\![-]\!]$ — type-indexed family of relations on closed expressions

$$\mathcal{V}[\![\mathsf{nat}]\!] \triangleq \{(\underline{n}, \underline{n}) \mid n \in \mathbb{N}\}$$

negative occurrence

$$\mathcal{V}[\![\sigma \to \tau]\!] \triangleq \{(u_1, u_2) \mid \forall (v_1, v_2) \in \mathcal{V}[\![\sigma]\!].\ (u_1\, v_1,\ u_2\, v_2) \in \mathcal{E}[\![\tau]\!]\}$$

$$\mathcal{E}[\![\tau]\!] \triangleq \{(e_1, e_2) \mid \exists v_1, v_2.\ e_1 \mapsto^* v_1,\ e_2 \mapsto^* v_2,\ (v_1, v_2) \in \mathcal{V}[\![\tau]\!]\}$$

$$[\![\Gamma]\!] \triangleq \{(\gamma_1, \gamma_2) \mid \forall x \in \mathrm{dom}\,\Gamma.\ (\gamma_1 x, \gamma_2 x) \in \mathcal{V}[\![\Gamma\ x]\!]\}$$

$$\Gamma \vDash_v v_1 \approx v_2 : \tau \triangleq \forall (\gamma_1, \gamma_2) \in [\![\Gamma]\!].\ (\gamma_1(v_1), \gamma_2(v_2)) \in \mathcal{V}[\![\tau]\!]$$

$$\Gamma \vDash_e e_1 \approx e_2 : \tau \triangleq \forall (\gamma_1, \gamma_2) \in [\![\Gamma]\!].\ (\gamma_1(e_1), \gamma_2(e_2)) \in \mathcal{E}[\![\tau]\!]$$

Addition, revisited

$x : \mathsf{nat} \vdash_e \mathsf{add}\ x\ 0 \approx x : \mathsf{nat}$

$\mathsf{add} = \lambda x, y.\ \mathsf{rec}(x, y, S)$

Take $(\gamma_1, \gamma_2) \in [\![x : \underline{\mathsf{nat}}]\!]$. Need to show $(\mathsf{add}\ \gamma_1(x)\ 0,\ \gamma_2(x)) \in \mathcal{E}[\![\mathsf{nat}]\!]$

$(\gamma_1(x), \gamma_2(x)) \in \mathcal{V}[\![\mathsf{nat}]\!] = \{(\underline{n}, \underline{n}) \mid n \in \mathbb{N}\}$

Therefore $\gamma_1(x) = \gamma_2(x) = \underline{n}$ for some n.

$(\mathsf{add}\ \underline{n}\ 0,\ \underline{n}) \in \mathcal{E}[\![\mathsf{nat}]\!]$

$?\ \downarrow_*\quad \downarrow_*$

$(\underline{n},\ \underline{n}) \in \mathcal{V}[\![\mathsf{nat}]\!]$

We have just 1 obligation:

$\mathsf{add}\ \underline{n}\ 0 \longmapsto^* \underline{n}$

easy!

Logical Relations: The Properties

↳ Adequacy: If $\Gamma \vdash e_1 \approx e_2 : \tau$ then $\Gamma \vdash e_1 \cong e_2 : \tau$

Ind. over closing cxt. $C : \Gamma \vdash \tau \rightsquigarrow \cdot \vdash \text{nat}$

1) if $C = \square$, then $(e_1, e_2) \in \mathcal{E}[\![\text{nat}]\!]$

$\Downarrow^*$ $\Downarrow^*$ so, by def. of eval. $e_1 \cong e_2$

$\Gamma \vdash u_1 \approx u_2 : \sigma \to \tau$

$C = C'[\square\ v]$ $\Gamma \vdash v \approx v : \sigma$

$C' : \Gamma \vdash \tau \rightsquigarrow \cdot \vdash \text{nat}$

↳ Reflexivity (Fundamental thm. of log. rel's)

If $\Gamma \vdash e : \tau$ then $\Gamma \vdash e \approx e : \tau$

Proof. Ind. str. typing derivation.

$$\frac{\Gamma \vdash u : \sigma \to \tau \qquad \Gamma \vdash v : \sigma}{\Gamma \vdash u\ v : \tau}$$

$\forall \sigma. \ \Gamma \vdash uv \approx uv : \tau$

By appropriate compat. lemma and IH.

↳ Compatibility ✓

$$\frac{\Gamma \vdash e_1 \approx e_2 : \tau \qquad \Gamma, x : \tau \vdash e_1' \approx e_2' : \sigma}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_1' \approx \text{let } x = e_2 \text{ in } e_2' : \sigma}$$

Compatibility Lemmas (selection)

$$\frac{\Gamma \vdash e_1 \approx e_2 : \tau \quad (*) \qquad \Gamma, x:\tau \vdash e_1' \approx e_2' : \sigma \quad (**)}{\Gamma \vdash \mathtt{let}\, x = e_1 \,\mathtt{in}\, e_1' \approx \mathtt{let}\, x = e_2 \,\mathtt{in}\, e_2' : \sigma}$$

a_1 a_2

Take $(\gamma_1, \gamma_2) \in [\![\Gamma]\!]$. Need to show $\gamma_1(a_1), \gamma_2(a_2) \in \mathcal{E}[\![\sigma]\!]$.

From $(*)$ $(\gamma_1(e_1), \gamma_2(e_2)) \in \mathcal{E}[\![\tau]\!]$

$(v_1, v_2) \in \mathcal{V}[\![\tau]\!]$

$N_{\text{ow}}, (\gamma_1[x \mapsto v_1], \gamma_2[x \mapsto v_2]) \in [\![\Gamma, x:\tau]\!]$, so from $(**)$ $(\gamma_1'(e_1'), \gamma_2'(e_2')) \in \mathcal{E}[\![\sigma]\!]$

γ_1' γ_2'

$(u_1, u_2) \in \mathcal{V}[\![\sigma]\!]$

$\gamma_1(a) = \mathtt{let}\, x = \gamma_1(e_1) \,\mathtt{in}\, \gamma_1^x(e_1')$

$\mathtt{let}\, x = v_1 \,\mathtt{in}\, \gamma_1^x(e_1')$

$\gamma_1^x(e_1')[v_1/x] = \gamma_1'(e_1') \mapsto^* u_1$

$(u_1, u_2) \in \mathcal{V}[\![\sigma]\!]$

$(u_1, u_2) \in \mathcal{V}[\![\text{nat}]\!] \longrightarrow u_1 = u_2 = \underline{n}$

$(v_1, v_2) \in \mathcal{V}[\![\sigma]\!]$ (*)

$(w_1, w_2) \in \mathcal{V}[\![\sigma \to \sigma]\!]$ (**)

$(\text{rec}(u_1, v_1, w_1) \approx \text{rec}(u_2, v_2, w_2)) \in \mathcal{E}[\![\sigma]\!]$

$u_1 = \underline{n}$, $u_2 = \underline{n}$

Proof: Induction over n.

1. $n = 0$

T.S. $(\text{rec}(0, v_1, w_1), \text{rec}(0, v_2, w_2)) \in \mathcal{E}[\![\sigma]\!]$

$\downarrow$ $\downarrow$

$(v_1 , v_2) \in \mathcal{V}[\![\sigma]\!]$ by (*)

2. Have hyp. $(\text{rec}(\underline{n}, v_1, w_1), \text{rec}(\underline{n}, v_2, w_2)) \in \mathcal{E}[\![\sigma]\!]$

T.S. $(\text{rec}(S\,\underline{n}, v_1, w_1), \text{rec}(S\,\underline{n}, v_2, w_2)) \in \mathcal{E}[\![\sigma]\!]$

$\downarrow$ $\downarrow$

$(\text{let } x = \text{rec}(\underline{n}, v_1, w_1) \text{ in } w_1\, x \quad , \quad \ldots) \in \mathcal{E}[\![\sigma]\!]$

S.T.S. that for any $(k_1, k_2) \in \mathcal{V}[\![\sigma]\!]$, $(w_1 k_1, w_2 k_2) \in \mathcal{E}[\![\sigma]\!]$, which follows from (**).

Consequences — Termination

↳ Termination follows directly from fundamental theorem

↳ This makes logical relations a powerful technique of proving (various kinds of) normalisation results, esp. since the method scales well

↳ However, this should also raise concerns:
if the technique enforces termination, can it be used to handle realistic (non-terminating!) languages?

↳ This tends to require using some form of coinduction-like reasoning
↳ Most common modern technique is step-indexing (see Robert's lecture)
↳ Step-indexing comes at a price: binary relations usually aren't transitive...

(Impredicative) Polymorphism (System F)

$\tau ::= \ldots \mid \forall \alpha.\, \tau \mid \exists \alpha.\, \tau \mid \alpha$

$v ::= \ldots \mid \Lambda e \mid \mathsf{pack}\, v$

$e ::= \ldots \mid v\, * \mid \mathsf{unpack}\, v\, \mathsf{as}\, x\, \mathsf{in}\, e$

$\tau_{id} = \forall \alpha.\, \alpha \to \alpha$

$id \equiv \Lambda.\, \lambda x.\, x$

$(\Lambda \alpha.\, \lambda x : \alpha.\, x)$

tyvar ctxt

$$\frac{\Delta, \alpha \mid \Gamma \vdash e : \tau}{\Delta \mid \Gamma \vdash \Lambda e : \forall \alpha.\, \tau}$$

$$\frac{\Delta \mid \Gamma \vdash v : \forall \alpha.\, \tau \qquad \Delta \vdash \sigma\ \mathsf{wf}}{\Delta \mid \Gamma \vdash v * : \tau[\sigma/\alpha]}$$

$$(\Lambda e) * \mapsto e$$

$$\frac{\Delta \mid \Gamma \vdash v : \tau[\sigma/\alpha]}{\Delta \mid \Gamma \vdash \mathsf{pack}\, v : \exists \alpha.\, \tau}$$

$$\frac{\Delta \mid \Gamma \vdash v : \exists \alpha.\, \tau \qquad \Delta, \alpha \mid \Gamma, x : \tau \vdash e : \sigma}{\Delta \mid \Gamma \vdash \mathsf{unpack}\, v\, \mathsf{as}\, x\, \mathsf{in}\, e : \sigma}$$

$$\mathsf{unpack}\ (\mathsf{pack}\, v)\, \mathsf{as}\, x\, \mathsf{in}\, e \mapsto \mathsf{let}\, x = v\, \mathsf{in}\, e$$

$$\frac{id : \forall \alpha.\, \alpha \to \alpha}{id * : \tau_{id} \to \tau_{id} \qquad id : \forall \alpha.\, \alpha \to \alpha} \qquad = (\alpha \to \alpha)[\tau_{id}/\alpha]$$

$$id * id : \forall \alpha.\, \alpha \to \alpha$$

Interpreting Polymorphism

Idea: Interpret $\Delta \vdash \tau$ wf $\quad \dfrac{\alpha \in \Delta}{\Delta \vdash \alpha \text{ wf}}$ How do I interpret this?

Have env. w/ ints of open tyvars
Where do elts. of env live? —— semantic int. of types

$$\underline{Typ} \triangleq \{R \mid R \subseteq Val \times Val\}$$

$$\mathcal{V}[\![\Delta \vdash \alpha \text{ wf}]\!] : \underline{Typ}^{\text{dom}\,\Delta} \to \underline{Typ}$$
$$\mathcal{E}[\![\Delta \vdash \alpha \text{ wf}]\!] : \underline{Typ}^{\text{dom}\,\Delta} \to \underline{Comp}$$

$$\mathcal{V}[\![\Delta \vdash \alpha]\!]\rho \triangleq \rho(\alpha)$$
$$\mathcal{V}[\![\Delta \vdash \forall\alpha.\tau]\!]\rho \triangleq \{(u_1,u_2) \mid \forall \mu \in \underline{Typ}.\ (u_1 *, u_2 *) \in \mathcal{E}[\![\Delta,\alpha \vdash \tau]\!]\rho[\alpha \mapsto \mu]\}$$
$$\mathcal{V}[\![\Delta \vdash \exists\alpha.\tau]\!]\rho \triangleq \{(\text{pack } v_1, \text{pack } v_2) \mid \exists \mu \in \underline{Typ}.\ (v_1,v_2) \in \mathcal{V}[\![\Delta,\alpha \vdash \tau]\!]\rho[\alpha \mapsto \mu]\}$$

Parametricity by example

$$x : \forall\alpha.\alpha\to\alpha \vdash x \approx id : \forall\alpha.\alpha\to\alpha.$$

Take $(v_1, v_2) \in \mathcal{V}[\![\forall\alpha.\alpha\to\alpha]\!]$. (1) ; Show $(v_1, id) \in \mathcal{V}[\![\forall\alpha.\alpha\to\alpha]\!]$.

$v_i * \mapsto^* u_i$

Take $R \in \underline{\text{Type}}$. Since $v_1 * \mapsto^* u_1$, S.T.S. $(u_1, \lambda x.x) \in \mathcal{V}[\![\alpha \mapsto \alpha]\!]_{[\alpha\mapsto R]}$

Take $(w_1, w_2) \in R$. Need to show $(u_1 w_1, (\lambda x.x)\, w_2) \in \mathcal{E}[\![\alpha]\!]_{\alpha\mapsto R}$

How can we proceed? precise value or values don't matter!

Construct relation $S = \{(w_1, w_2)\}$, and feed it to (1).

By determinism of reduction, we get that

$$(u_1, u_2) \in \mathcal{V}[\![\alpha \mapsto \alpha]\!]_{\alpha\mapsto S}$$

Since $(w_1, w_2) \in S$, $(u_1 w_1, u_2 w_2) \in \mathcal{E}[\![\alpha]\!]_{\alpha\mapsto S}$

Thus $u_i w_i \mapsto^* r_i$, and $(r_1, r_2) \in S$ — but this means $r_1 = w_1$!

Thus, $u_1 w_1 \mapsto^* w_1$, $(\lambda x.x)\, w_2 \mapsto^* w_2$, — and of course $(w_1, w_2) \in R$, which ends the proof.

Exercises

1. System T is very simple, but quite expressive. Define Ackermann function (which is famously not primitive recursive) that satisfies the following equations:

$$A\ 0\ n = S\ n$$
$$A\ (S\ m)\ 0 = A\ m\ (S\ 0)$$
$$A\ (S\ m)\ (S\ n) = A\ m\ (A\ (S\ m)\ n)$$

2. Our definition of System T was quite minimal — but we can extend it. Add product types $(\sigma \times \tau)$ and sum types $(\sigma + \tau)$ to our language, with the following (typed) syntax. Extend the semantics — and the logical relation! — appropriately.

$$\frac{\Gamma \vdash v_1 : \tau_1 \qquad \Gamma \vdash v_2 : \tau_2}{\Gamma \vdash (v_1, v_2) : \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash v : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{fst}\ v : \tau_1 \qquad \Gamma \vdash \mathsf{snd}\ v : \tau_2}$$

$$\frac{\Gamma \vdash v : \tau_1}{\Gamma \vdash \mathsf{in}_1\ v : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash v : \tau_2}{\Gamma \vdash \mathsf{in}_2\ v : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash u : \sigma_1 + \sigma_2 \qquad \Gamma, x : \sigma_1 \vdash e_1 : \tau \qquad \Gamma, x : \sigma_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{match}\ u\ \mathsf{with}\ \begin{array}{l} \mathsf{in}_1\ x \to e_1 \\ \mathsf{in}_2\ x \to e_2 \end{array} : \tau}$$

3. Primitive recursion as it is usually defined also gives access to the predecessor, as defined by the following equations:

$$\mathrm{recp}(0, v, w) = v$$

$$\mathrm{recp}(S\,u, v, w) = w\ u\ (\mathrm{recp}(u, v, w))$$

In the context of System T, we can actually express recp! Show how to define it, formulate lemmas that will justify the equations above, and prove them using the logical relation. (Hint: it is relatively easy in a language with products – the particular use of products can then be coded away)

4. We do not need to restrict ourselves to simple and inductive types in extending System T.
Add the type of infinite <u>streams</u> with the following typing and reduction rules.

$$\frac{\Gamma \vdash u : \sigma \quad \Gamma \vdash v : \sigma \to \tau \quad \Gamma \vdash w : \sigma \to \sigma}{\Gamma \vdash str(u,v,w) : stream\ \tau} \qquad \frac{\Gamma \vdash v : stream\ \tau}{\Gamma \vdash hd\ v : \tau} \qquad \frac{\Gamma \vdash v : stream\ \tau}{\Gamma \vdash tl\ v : stream\ \tau}$$

$$hd\,(str(u,v,w)) \longmapsto v\,u$$

$$tl\,(str(u,v,w)) \longmapsto \text{let } x = w\,u \text{ in } str(x,v,w)$$

Extend the definition of the logical relation to handle streams!

Hint: The relation needs to be <u>coinductive</u>;
it is <u>much</u> easier to define using eliminators (hd & tl)