— Overview

Spelling out exactly what it means to “encode a type” in a purely operational way is non-obvious.

Today we'll define Reynold's relational parametricity. We'll also cover contextual equivalence, the standard operational way of dealing with such encodings.

We want to show things like

$$\text{bool} \approx \forall\alpha.\alpha\to\alpha\to\alpha.$$
$$\tau_1\times\tau_2 \approx \forall\alpha.(\tau_1 \to \tau_2 \to \alpha) \to \alpha.$$
$$\tau_1+\tau_2 \approx \forall\alpha.(\tau_1 \to \alpha) \to (\tau_2 \to \alpha) \to \alpha.$$

but what is this $\approx$?

(Aside: We'll go through more interesting types next time.)

For each of these types, we have canonical forms.

Consider Bool:

$\text{true} = \Lambda\alpha.\lambda x.\lambda y.x$
$\text{false} = \Lambda\alpha.\lambda x.\lambda y.y$

We want to show

If $\vdash v : \text{bool}$,
then $v \equiv_{ctx} \text{true} \vee v \equiv_{ctx} \text{false}$.

The relation $\equiv_{ctx}$, called contextual equivalence, is the notion of equivalence we get from looking at the operational semantics. Informally, if $v \equiv_{ctx} \text{true}$, then putting v in any program is “the same” as putting true in that program.

Informally, we want to show that such Church encodings are /full/ and /faithful/. They are full in the sense that we can encode all the canonical forms we expect (as elements of the type) and faithful in the sense that there are no extra inhabitants of the type that don't behave like one of the canonical forms; that is, no junk.

Today we'll define contextual equivalence, talk about why Derek wanted to avoid it, talk about Reynold's idea of relational parametricity, and how Reynold's method compares to Girard's method.

(Aside: The hard part, for Derek, is finding a way to motivate "Reynold's method".)

— What "encoding a type" means

Here's the general pattern. For some type τ defined via a Church encoding, we want to prove

$$\forall v{:}\tau.\ v \equiv_{ctx} \eta\text{-expansion}(v) \downarrow \text{canonical}(\tau).$$

Last time, we saw that the second conjunct—that the η-expansion of v evaluates to one of the canonical forms of type τ—can be proven with the unary model. We saw we need something else for the fact that v is contextually equivalent to its η-expansion.

For bool, we'll show

$$\forall v : \text{bool}.\ \ v \equiv_{ctx} v\ [\text{bool}]\ \text{true}\ \text{false} \downarrow v' \in \{\text{true},\text{false}\}.$$

It turns out that the second conjunct is also a contextual equivalence (since all $\equiv_{ctx}$ can do is evaluate stuff):

$$\forall v : \text{bool}.\ \ v \equiv_{ctx} v\ [\text{bool}]\ \text{true}\ \text{false} \equiv_{ctx} v' \in \{\text{true},\text{false}\}.$$

— Observations without base types

Derek didn't want to introduce $\equiv_{ctx}$ because you have to talk about "distinguishing different results" (observations) and client programs (contexts). In many languages, you can use termination as observation. In System F, all programs terminate. So what do you observe? The usual thing people do is work with a language with some kind of base type and build observations up from that. We didn't bake in a built in, say, unit or boolean type in System F.

Derek thinks you can define contextual equivalence perfectly well, just with System F. We'll use our Church encoding of bool and the fact that η-expansion reduces to true or false (provable with Girard's method) to define our observations.

— Motivating relational parametricity

Back to our story. Let's spell out what contextual equivalence means for bool.

$\forall f{:}bool.$
$(\forall \sigma, v_1, v_2.\ f\ \sigma\ v_1\ v_2 \downarrow v_1) \vee$
$(\forall \sigma, v_1, v_2.\ f\ \sigma\ v_1\ v_2 \downarrow v_2).$

That's roughly $\forall f{:}bool.\ f \equiv ctx\ true \vee f \equiv ctx\ false$.

Question: Wouldn't it be asounding (given our notion of observations) if this didn't work for bool?

> Answer: Derek offered a counterexample based on an extension to the langauge. The idea is to break parametricity. Add, say, a typecase operation and build a value around it. Here's our really stupid extension:
>
> > $iftypeisbool(\tau)\ then\ e_1\ else\ e_2$
>
> We have more things of type bool than true and false. We also have:
>
> > $v := \Lambda\alpha.\lambda x_1.\lambda x_2.iftypeisbool(\alpha)\ then\ x_1\ else\ x_2.$
>
> So $v\ [bool]\ true\ false \downarrow true$.
> But if $\tau \neq bool$, then $v\ [\tau]\ v_1\ v_2 \downarrow v_2$.
>
> We have not broken the η-expansion property. But we have broken contextual equivalence to true/false.

Idea: We know $f\ [\sigma]\ v_1\ v_2 \downarrow v' \in \{v_1, v_2\}$.
Suppose we get v_1.
TS: $\forall \sigma', v'_1, v'_2.\ f[\sigma']\ v'_1\ v'_2 \downarrow v'_1$.

To clear this up, we'll set up a relation r s.t. if $(v_1, v'_1) \in r$ and $(v_2, v'_2) \in r$, then $(v', v'_1) \in r$; that is, we'll show that if f is given related arguments, then it produces related results.

That's the motivation/intuition for Reynold's relational parametricity. We'll return to this example once we define the relational model.

— Relational parametricity

Recall Girard's model:

> Cand := Sub(CVal)

$E[\tau]\rho := \{ e \mid \exists v.\ e \downarrow v \wedge v \in V[\tau]\rho \}$
$V[\alpha]\rho := \rho(\alpha)$
$V[\sigma \to \tau]\rho := \{ v \mid \forall v' \in V[\sigma]\rho.\ v\ v' \in E[\tau]\rho \}$
$V[\forall\alpha.\tau]\rho := \{ v \mid \forall \sigma \in CTyp.\ \forall S \in Cand.\ v\ \sigma \in E[\tau](\rho, \alpha \mapsto S) \}$
$D[\Delta] := \{ \rho \in \Delta \to Cand \}$
Minor tweak: Our earlier definition was $\{ \rho \in Tyvar \rightharpoonup Cand \mid \Delta \subseteq dom(\rho) \}$.
$G[\Gamma]\rho := \{ \gamma \in Var \rightharpoonup CVal \mid \forall (x{:}\tau) \in \Gamma.\ \gamma(x) \in V[\tau]\rho \}$
(Aside from Dave: We should tweak both D[] and G[] or neither.)

Let's adjust it, a line at a time. Everywhere you see one thing, you turn it into two things.

$Cand := Sub(CVal \times CVal)$
$E[\tau]\rho := \{ (e_1,e_2) \mid \exists v_1,v_2.\ e_1 \downarrow v_1 \wedge e_2 \downarrow v_2 \wedge (v_1,v_2) \in V[\tau]\rho \}$
$V[\alpha]\rho := \rho(\alpha)$
This case doesn't change: We'll adjust ρ.
$V[\sigma \to \tau]\rho := \{ (v_1,v_2) \mid \forall (v'_1,v'_2) \in V[\sigma]\rho.\ (v_1\ v'_1,\ v_2\ v'_2) \in E[\tau]\rho \}$
$V[\forall\alpha.\tau]\rho := \{ (v_1,v_2) \mid \forall \sigma_1, \sigma_2 \in CTyp.\ \forall R \in Cand.\ (v_1\ \sigma_1,\ v_2\ \sigma_2) \in E[\tau](\rho,\alpha\mapsto R) \}$
$D[\Delta] := \{ \rho \in \Delta \to Cand \}$
$G[\Gamma]\rho := \{ (\gamma_1,\gamma_2) \mid \forall (x{:}\tau) \in \Gamma.\ (\gamma_1 x,\ \gamma_2 x) \in V[\tau]\rho \}$

This generalization from unary to binary was *non-obvious* and the key to Reynold's work.

Reynold's worked in a denotational model and proved a lovely result. Aside: He worked in a model that didn't actually exist, but the result mattered.

FTLR (Reynolds, “Abstraction Theorem”, 1983):
If $\Delta; \Gamma \vdash e : \tau$,
then $\forall \rho \in D[\Delta].\ \forall (\gamma_1,\gamma_2) \in G[\Gamma]\rho.\ \forall \delta_1,\delta_2 : \Delta \to CTyp.$
$(\delta_1\gamma_1 e,\ \delta_2,\gamma_2 e) \in E[\tau]\rho$.

Compare this to the FTLR with Girard's method. We've just doubled things.

(We'll prove this in a slightly more general form later on.)

Corollary:
If $\vdash e : \tau$,
then $(e,e) \in E[\tau]$.

We'll use the corollary to prove our result about bool.

$\vdash$ f : bool
Suppose f [σ] v_1 v_2 $\downarrow$ hat{v}.
TS: $\forall \sigma', v'_1, v'_2$. f [$\sigma'$] v'_1 v'_2 $\downarrow$ hat{v'}.
(f,f) $\in$ V[bool].
R := $\{(v_1,v'_1), (v_2,v'_2)\}$.
(f σ_1 v_1 v_2, f σ_2 v'_1 v'_2) $\in$ E[α]$\alpha \mapsto$ R.
(hat{v},hat{v'}) $\in$ R.
Thus, either hat{v} = $v_1 \wedge$ hat{v'} = v'_1
or hat{v} = $v_2 \wedge$ hat{v'} = v'_2.

— Logical relation on open terms, FTLR, and soundness

(Aside from Dave: Derek started this section using the phrase “logical equivalence” but later noted the relation isn't transitive.)

Let's restate the FTLR in terms of the following type-respecting binary relation.

Definition (Logical relation on open terms):

$\Delta; \Gamma \vdash e_1 \approx e_2 : \tau :\Longleftrightarrow$
$\Delta; \Gamma \vdash e_1 : \tau \wedge$
$\Delta; \Gamma \vdash e_2 : \tau \wedge$
$\forall \rho \in D[\Delta]. \forall (\gamma_1,\gamma_2) \in G[\Gamma]\rho. \forall \delta_1,\delta_2 : \Delta \longrightarrow CTyp.$
$(\delta_1\gamma_1 e_1, \delta_2,\gamma_2 e_2) \in E[\tau]\rho.$

This is a standard approach: We quantify over all closing substitutions and use the logical relation for closed terms.

FTLR (Abstraction Theorem aka Fundamental Property aka Reflexivity):
If $\Delta; \Gamma \vdash e : \tau$,
then $\Delta; \Gamma \vdash e \approx e : \tau$.

(In its generalized form, the FTLR for binary logical relations boils down to “it's reflexive on well-typed terms”.)

Our $\approx$ is generally useful. We'll show $\approx \subseteq \equiv$ctx. (That's soundness of the logical relation.) Thus, we can use $\approx$ as a proof technique for $\equiv$ctx.

We haven't established that $\approx$ is an equivalence relation. (It isn't an

equivalence relation. It's probably symmetric. It's not transitive.)

Question: Do these techniques let us talk about more general things than contextual equivalence?

> Answer: We focus on contextual equivalence and contextual approximation (or refinement).

(Aside from Dave: Derek started writing ≡ for contextual equivalence at this point. In future notes, I will do so. For now, let's stick with ≡ctx, leaving ≡ available for arbitrary "type-respecting binary relations".)

We'll aim for the following

Soundness theorem:
> If $\Delta; \Gamma \vdash e_1 \approx e_2 : \tau$
> then $\Delta; \Gamma \vdash e_1 \equiv_{ctx} e_2 : \tau$.

Aside: We'll define ≡ctx so that the fundamental property falls out from soundness.

— Defining contextual equivalence via contexts

How do we define ≡ctx?

There are several ways.

Traditional way: Define well-formed contexts C (ie, a term with a hole). To do this properly, you have to define a context typing judgement

> $\vdash C : (\Delta; \Gamma; \tau) \longrightarrow (\Delta'; \Gamma'; \tau')$
>
> "typing context for hole" ⟶ "typing for C[−]"

It's tedious. There are opportunities for errors. The whole point is you want variable capture, so this is hard to mechanize.

Informally, we use contextual equivalence because its something we can all agree on. But it's technical and annoying to do. You end up having to prove a lemma that amounts to the following (slightly more direct) approach.

Once you have this notion of contexts with holes, you say something like

$\Delta; \Gamma \vdash e_1 \equiv_{ctx} e_2 : \tau \iff$(vague)
$\forall C : (\Delta;\Gamma;\tau) \longrightarrow (\cdot;\cdot;\text{bool})$.
$C[e_1]\downarrow$true iff $C[e_2]\downarrow$true.

— Defining contextual equivalence via congruence relations

(We're following Pitts' chapter on “Typed Operational Reasoning” in Pierce's ATTPL. Pitts said he followed Andrew Gordon and Søren Lassen.)

Our goal is to define $\equiv_{ctx}$ as the largest, adequate congruence on well-typed terms.

What is adequate? What is a congruence? Together, they tease out the two things we sought informally: “quantification over the client context” and “closed programs produce true/false together”.

(Aside from Dave: Derek didn't bother with the following definition in class. I think it helps to have the universe from $\equiv_{ctx}$ is drawn.)

Definition (Type respecting binary relations):
RAtom := $\{ (\Delta,\Gamma,e_1,e_2,\tau) \mid \Delta; \Gamma \vdash e_1 : \tau \wedge \Delta; \Gamma \vdash e_2 : \tau \}$
TRBR := Sub(Ratom).

Informally, a type-respecting binary relation $\equiv \in$ TRBR is a relation on terms e_1 and e_2 that are well-typed wrt a common context and type.

We write $\Delta; \Gamma \vdash e_1 \equiv e_2 : \tau$ when $(\Delta,\Gamma,e_1,e_2,\tau) \in \equiv$.

Idea: We care about relations $\equiv \in$ TRBR that are “adequate for observing termination”.

Definition (Adequacy):
Let $\equiv \in$ TRBR.
$\equiv$ is /adequate/ if
$\cdot \vdash e_1 \equiv e_2 : \text{bool} \Longrightarrow e_1 \downarrow v_1 \wedge e_2 \downarrow v_2 \wedge v_1 \equiv_{KL} v_2$.

Definition (Kleene equivalence at type bool):
Let $v_1, v_2 \in$ CVal.
$v_1 \equiv_{KL} v_2$ if $(v_1 \downarrow \top \iff v_2 \downarrow \top)$

where $v \downarrow \top$ if v [bool] true false $\downarrow$ true
(and $v \downarrow \bot$ if v [bool] true false $\downarrow$ false).

Idea: The η-expansion approach spits out the canonical form.

Idea: Kleene equivalence and adequacy are well-defined because of Girard's method. We use Kleene equivalence to canonize v_1 and v_2 into comparable forms.

Definition (Congruence):
Let $\equiv \in$ TRBR.
$\equiv$ is a /congruence/ if $\equiv$ is an equivalence relation
(reflexive, symmetric, and transitive) that is compatible.

(Aside from Dave: We might spell out how to lift reflexivity, symmetry, and transitivity from Exp×Exp to RAtom. Not even I want to be that pedantic.)

Idea: A compatible relation $\equiv \in$ TRBR is “closed under the term-formation rules of the language”.

To define compatibility, we'll follow the typing rules and double them up, much as we did for the logical relations.

Definition (Compatibility):
Let $\equiv \in$ TRBR.
$\equiv$ is /compatible/ if it is closed under the following axioms and rules.

$$\frac{x{:}\tau \in \Gamma}{\Delta; \Gamma \vdash x \equiv x : \tau}$$

$$\frac{\Delta; \Gamma, x{:}\sigma \vdash e_1 \equiv e_2 : \tau}{\Delta; \Gamma \vdash \lambda x.e_1 \equiv \lambda x.e_2 : \sigma \longrightarrow \tau}$$

$$\frac{\Delta; \Gamma \vdash e_1 \equiv e'_1 : \sigma \longrightarrow \tau \quad \Delta; \Gamma \vdash e_2 \equiv e'_2 : \sigma}{\Delta; \Gamma \vdash e_1\ e_2 \equiv e'_1\ e'_2 : \tau}$$

$$\frac{\Delta, \alpha; \Gamma \vdash e_1 \equiv e_2 : \tau}{}$$

$\Delta; \Gamma \vdash \Lambda\alpha.e_1 \equiv \Lambda\alpha.e_2 : \forall\alpha.\tau$

$\Delta, \Gamma \vdash e_1 \equiv e_2 : \forall\alpha.\tau$
$\text{ftv}(\sigma) \subseteq \Delta$
—
$\Delta; \Gamma \vdash e_1\ \sigma \equiv e_2\ \sigma : \tau[\sigma/\alpha]$

We pick the same σ on both sides. You could imagine picking σ_1 and σ_2 if the language had a notion of type equivalence. (But it doesn't.)

Idea: To prove a relation $\equiv \in$ TRBR compatible, you prove a lemma for each compatibility rule.

Question from Deepak: Why does "the" largest adequate congruence exist? That's not obvious at all.

Answer: We'll claim it for now. At some point, Derek will explain. (The proof in Pitts' chapter is kinda fiddley. Derek wants a more direct presentation.)

Idea: Rather than define ≡ctx, we'll (eventually) prove it exists.

Claim (≡ctx, channeling Pitts):
There exists a largest type-respecting binary relation between System F terms that is a congruence and adequate.

We call it contextual equivalence and write it ≡ctx.

Proof: Sorry. Derek will prove this later.

If you want to use contextual equivalence with C[–], you have to cough up a context. Here, we essentially build up that context using these compatibility rules.

Somehow the traditional definition is clearly well-defined. Whereas here, we have to argue there's a "largest".

Our notion of compatibility corresponds to multi-holed contextual equivalence. Consider the application rule: There are two subterms and we may have equivalent things in either the function or argument position.

The work we have to do is to show that multi-hole contextual

equivalence boils down to single-hole contextual equivalence. (That can fail to be true.) Of course, you can define multi-hole contextual equivalence, but it's ugly.

Once you prove a largest, adequate congruence exists, you obtain a more direct proof method.

> (Aside from Dave: One example “multi-holed contextual equivalence” in the wild: Birkedal and Harper, I&C'99. Relational interpretations of recursive types in an operational setting.
>
> If you read that paper, don't worry too much about the details: Derek can teach us a better way to deal with recursive types.)

— Soundness of the logical relation on open terms

Theorem (Soundness):
$\approx \subseteq \equiv_{ctx}$.

It suffices to show that $\approx$ is adequate and compatible. (Recall, $\approx$ is not an equivalence hence not a congruence.)

The idea is to take the symmetric, transitive closure of $\approx$ and show it's adequate and a congruence. We've just closed up over the missing rules, so its clearly a congruence. But we have to show that we've preserved adequacy and compatiblity.

In the traditional approach, you have to prove “$\approx$ is adequate and compatible”, anyway.

We'll prove $\approx$ is adequate and compatible.

To show compatibility for $\approx$, we'll prove a lemma for each of the compatibility rules.

Once we've done those things, we'll know soundness for the LR. But then the FTLR follows from the easy lemma (see Pitts):

Lemma (see Pitts):
Let $\equiv \in$ TRBR.
If $\equiv$ is compatibile,
then $\equiv$ is reflexive.

Proof: Easy induction on $D :: \Delta; \Gamma \vdash e : \tau$. In defining the "term formation rules", we doubled up the typing rules.

— Compatibility of $\approx$

Our proof of compatiblity is just the "binarification" of our proof of Girard's method.

HW: Prove the compatiblity lemmas:

$$\frac{x{:}\tau \in \Gamma}{\Delta; \Gamma \vdash x \approx x : \tau}$$

$$\frac{\Delta; \Gamma, x{:}\sigma \vdash e_1 \approx e_2 : \tau}{\Delta; \Gamma \vdash \lambda x.e_1 \approx \lambda x.e_2 : \sigma \longrightarrow \tau}$$

$$\frac{\Delta; \Gamma \vdash e_1 \approx e'_1 : \sigma \longrightarrow \tau \quad \Delta; \Gamma \vdash e_2 \approx e'_2 : \sigma}{\Delta; \Gamma \vdash e_1\ e_2 \approx e'_1\ e'_2 : \tau}$$

$$\frac{\Delta, \alpha; \Gamma \vdash e_1 \approx e_2 : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.e_1 \approx \Lambda\alpha.e_2 : \forall\alpha.\tau}$$

$$\frac{\Delta, \Gamma \vdash e_1 \approx e_2 : \forall\alpha.\tau \quad ftv(\sigma) \subseteq \Delta}{\Delta; \Gamma \vdash e_1\ \sigma \approx e_2\ \sigma : \tau[\sigma/\alpha]}$$

— Things we've yet to prove

Claim ($\equiv$ctx, channeling Pitts):

> There exists a largest type-respecting binary relation between System F terms that is a congruence and adequate.

Claim (Adequacy of $\approx$):

> $\approx$ is adequate.