Let's begin by fixing the bug in the proof from last time.

Last time, we defined CTerm and CVal to be "mostly closed" in order to simplify the statement of the fundamental theorem. It didn't work. We wanted to avoid adding δ, the closing syntactic substitutions.

There are two ways to fix the setup/proof.

• The simple solution: Make sure everything is closed. We'll take this approach for now.

• Unexplored: We might define E and V modulo changes in the syntactic types. We may take this approach with later models.

Note in the statement of the fundamental theorem that δ and ρ have nothing to do with each other: The syntactic types play no role dynamically.

The fundamental theorem is fundamental because it says that syntactic terms inhabit the model.

(We went over my proof.)

— Variation: Avoid canonical forms in $V[\sigma\to\tau]\rho$ and $V[\forall\alpha.\tau]\rho$.

Let's discuss some variations on the model we've defined. (Pay particular attention to this first one.)

Idea: We might avoid making explicit the canonical forms of $\to$ and $\forall$ types in the definition of V.

Before, we had

$$V[\sigma\to\tau]\rho := \{\ \lambda x.e \mid \forall v \in V[\sigma]\rho.\ e[v/x] \in E[\tau]\rho\ \}.$$

This isn't necessary. We could have written

$$V[\sigma\to\tau]\rho := \{\ v \mid \forall v' \in V[\sigma]\rho.\ v\ v' \in E[\tau]\rho\ \}.$$

That's a little closer to what Girard did in his original proof.

Similarly, we could change

$$V[\forall\alpha.\tau]\rho := \{\ \Lambda\alpha.e \mid \forall\sigma \in CTyp.\ \forall S \in Cand.\ e[\sigma/\alpha] \in E[\tau](\rho, \alpha \mapsto S)\ \}$$

to

$$V[\forall\alpha.\tau]\rho := \{\ v \mid \forall\sigma \in CTyp.\ \forall S \in Cand.\ v\ \sigma \in E[\tau](\rho, \alpha \mapsto S)\ \}.$$

In this setting, we're fine because if v v' is in the model, then we know after one step of β-reduction, we're in the model.

We'll have to be much more careful about this kind of thing when we start using step-indexing: Such models are sensitive to the number of β-reduction steps.

For some of the results we'll seek, this variant is a little slicker.

(Aside: Neel pointed out that if you have constants inhabiting your arrow type that aren't necessarily lambdas, this formulation is more useful.)

— Variation: Use type-erased terms.

We'll simply sketch the other variation.

Idea: Define the model using type-erased terms. (To avoid this δ nonsense.)

Let's define

Erased Terms

$\text{hat}\{e\} ::= x \mid \lambda x.\text{hat}\{e\} \mid \text{hat}\{e_1\}\ \text{hat}\{e_2\} \mid \Lambda.\text{hat}\{e\} \mid \text{hat}\{e\}[]$

We indicate where type abstractions and applications occur, but don't bother with the types.

We have to define reduction $\text{hat}\{\mapsto\}$ on erased terms. It's defined exactly as before, except

$(\Lambda.\text{hat}\{e\})[]\ \text{hat}\{\mapsto\}\ \text{hat}\{e\}$.

We define a function

$|-| : \text{Term} \longrightarrow \text{ErasedTerm}$

The interesting cases of this function:

$|\Lambda\alpha.e| = \Lambda.|e|$

$|e\ \sigma| = |e|[]$.

The idea: Build the model over erased terms and erased values. The interesting case:

$V[\forall\alpha.\tau]\rho := \{ v \mid \forall S \in \text{Cand}.\ v[] \in E[\tau](\rho, \alpha \mapsto S) \}$

(We don't need $\forall\sigma \in \text{CTyp}$.)

Of course, we'll have to rephrase the fundamental theorem:

If $\Delta; \Gamma \vdash e : \tau$,
then $\forall \rho \in D[\Delta].\ \forall \gamma \in G[\Gamma]\rho.\ \gamma|e| \in E[\tau]\rho$.

(We managed to drop the syntactic substitutions δ.)

To reason about the original language, we'll have to somehow connect up the reduction behavior of terms to that for erased terms. Establish a simulation between the reduction of terms and the reduction of erased terms using the following handy lemma.

i. $e \mapsto e' \implies |e|\ \mathrm{hat}\{\mapsto\}\ |e'|$
ii. $|e|\ \mathrm{hat}\{\mapsto\}\ \mathrm{hat}\{e\} \implies \exists e'.\ e \mapsto e' \wedge |e'| = \mathrm{hat}\{e\}$.

Now, when we apply the fundamental theorem, the model will give us

$\vdash e : \tau \implies |e| \downarrow \mathrm{hat}\{v\}$.

By iteratively applying (ii), we get $e \downarrow v$.

(Aside: If we rely on progress and preservation—which we know—then we could probably get away with just (i) in the lemma.)

Discussion: Which treatment of type variables is better?

1. In our original model, δ's get pushed around through our proofs (to no real purpose).

2. With this variant, we define some (otherwise useless) shadow semantics to avoid needless clutter in the definition of the model and its metatheory.

3. You could avoid proving the bisimulation by taking the reduction on erased terms *as* the operational semantics. (Neel and Deepak ask why we care about the semantics that mentions types but doesn't use them?)

On the one hand is your program, the thing you type-check. On the other hand is a computation (derived from the program), the thing with a dynamics.

4. Yet another approach (taken by Ahmed but thought distasteful by Derek): Just work with terms like $\Lambda.e$ and give typing rules at that level. This becomes problematic; for example, if you have modules.

Derek's preferred approach is to use the langauge with types for statics and the langauge without for dynamics.

— Some theorems we can prove with Girard's method

Some of this material doesn't seem to exist in the literature. We're doing things with parametricity in a strictly operational setting. Most early work used denotational models.

(Aside from Neel: The operational semantics was behind the scenes in the 20 years of research on denotational approaches.)

(Aside from Dave: I prefer stating these definability of types results using terms rather than values. I leave things as Derek proved them.)

Example: System F has no closed values of type $\forall\alpha.\alpha$.

> (Aside: Rather than clutter our proof, we simply drop empty ρ's and we let the reader infer domains of quantification from the model; for example, writing $\forall\sigma$ rather than $\forall\sigma \in \mathrm{CTyp}$.)
>
> $\vdash v : \forall\alpha.\alpha \Longrightarrow v \in V[\forall\alpha.\alpha]$
> $\Longleftrightarrow \forall\sigma,S.\ v\ \sigma \in E[\alpha]\alpha \mapsto S.$
>
> Pick any σ and $S := \emptyset$. There exists v' satisfying
> $\quad v\ \sigma \in E[\alpha]\alpha \mapsto \emptyset$
> $\quad \Longrightarrow v\ \sigma \mapsto_* v' \in V[\alpha]\alpha \mapsto \emptyset = \emptyset,$
> a contradiction.

We want to prove that if $\vdash f : \forall\alpha.\alpha \to \alpha$, then f "behaves like the identity". We won't define "behavior equivalence" until we get to the relational model.

Example:
> If $f \in \mathrm{CVal}$
> and $\vdash f : \forall\alpha.\alpha \to \alpha$,
> then $\forall\sigma \in \mathrm{CTyp}, v \in \mathrm{CVal}.\ f\ \sigma\ v \downarrow v$.
>
> (Incidentally, this is the kind of theorem you don't see in the work with denotational models. They're concerned with a coarser level of equality. Here, v on the right is syntactically identical to v on the left.)
>
> $\vdash f : \forall\alpha.\alpha \to \alpha \Longrightarrow f \in V[\forall\alpha.\alpha \to \alpha]$

$\iff \forall\sigma', S.\ f\ \sigma' \in E[\alpha \to \alpha]\alpha \mapsto S.$

Take $\sigma' := \sigma$ and $S := \{v\}$. There exist v_f and v' satisfying
$f\ \sigma \in E[\alpha \to \alpha]\alpha \mapsto \{v\}$
$\implies f\ \sigma \mapsto_* v\_f \in V[\alpha \to \alpha]\alpha \mapsto \{v\}$
$\implies v\_f\ v \in E[\alpha]\alpha \mapsto \{v\}$
$\implies v\_f\ v \mapsto_* v' \in V[\alpha]\alpha \mapsto \{v\} = \{v\}.$

Example:
Set bool := $\forall\alpha.\alpha\to\alpha\to\alpha$
and true := $\Lambda\alpha.\lambda x.\lambda y.x$
and false := $\Lambda\alpha.\lambda x.\lambda y.y$.

If $f \in$ CVal,
and $\vdash f$: bool,
then ?.

With this model, we won't be able to show f "behaves like" either true or false.

We would like to show:
$(\forall\sigma, v_1, v_2.\ f\ \sigma\ v_1\ v_2 \downarrow v_1) \vee$
$(\forall\sigma, v_1, v_2.\ f\ \sigma\ v_1\ v_2 \downarrow v_2).$

We can only show: $\forall\sigma, v_1, v_2.\ f\ \sigma\ v_1\ v_2 \downarrow v \in \{v_1, v_2\}$.

This is weaker: With a particular (v,v') f might return v; with some other (w,w'), f might return w'.

So let's prove what we can with this model. We'll be quick about these free theorems.

By FTLR. $f \in V[\forall\alpha.\alpha \to \alpha \to \alpha]$.
Pick $S := \{v_1, v_2\}$.
We'll get our (weaker) goal.

— Theorems provable in the unary vs binary models

Let's state the boolean example in a slightly different way. (This will generalize nicely to the other kinds of types that are definable in System F.)

Theorem (a free theorem):
(a) $\vdash f$: bool $\implies f$ bool true false $\downarrow v \in$ {true,false}.

Understand “f bool true false” as just an η-expansion of f *at the type bool*. Informally, the η-expansion of a value v:bool is “if v then true else false”; that is, v bool true false.

Idea: The unary model shows that if you η-expand, you literally get one of the canonical forms. The free theorem is just an instance of the more general thing we've shown (the fundamental theorem).

Structuring this in terms of η-expansion allows us to see clearly what part we can prove with unary parametericity vs binary parametericity. What's missing is

(*) $f \approx f\ \text{bool true false}$

for some notion of equivalence ≈. We need relational parametricity for this notion of equivalence and this part of the proof.

Once we know (*), we can compose it with (a) to get a proof that $v \approx \text{true}$ or $v \approx \text{false}$.

Put another way, we're decomposing our eventual proof into two parts: Everything we can show in unary LR and the bit we need binary LR to show.

Think of the η-expansion as doing a fold over the type and producing canonical values.

— Product types (some proofs in the unary model)

In System F, we can encode products, sums, existentials, etc.

Example: Products.

$$\tau_1 \times \tau_2 := \forall\alpha.(\tau_1 \longrightarrow \tau_2 \longrightarrow \alpha) \longrightarrow \alpha.$$

(Aside: You can always think of these Church encodings as CPS.)

The constructor value “pair”; the intro form <–,–>; and the projections π_1 and π_2 are defined as follows.

$$\text{pair} := \lambda x_1.\lambda x_2.\langle x_1, x_2\rangle$$

$$\langle v_1, v_2\rangle := \Lambda\alpha.\lambda k.k\ v_1\ v_2.$$

$\pi_1\ e := e\ \tau_1\ (\lambda x.\lambda y.x)$
$\pi_2\ e := e\ \tau_2\ (\lambda x.\lambda y.y)$

To show these do what we expect, write out the β-reduction property:

TS: $\pi_1(<v_1,v_2>) \longmapsto_* v_1$ (and similarly for π_2).

We have:
$\pi_1\ <v_1,v_2> = (\Lambda\alpha.\lambda k.k\ v_1\ v_2)\ \tau_1\ (\lambda x.\lambda y.x)$
$\longmapsto_* (\lambda x.\lambda y.x)\ v_1\ v_2$
$\longmapsto_* v_1.$

What could we hope to show using unary vs binary parametricity?

TS (binary):
If $\vdash f : \tau_1 \times \tau_2$
then $f \approx <v_1,v_2>$ for some v_1, v_2.

What we can show is a property of the η-expansion of f and evaluation:

TS (unary):
If $\vdash f : \tau_1 \times \tau_2$
then $f\ (\tau_1\times\tau_2)\ (pair) \downarrow <v_1,v_2>$ for some v_1,v_2.
Pf:
By FTLR, $f \in V[\tau_1\times\tau_2]$.
Pick $S := \{ <v_1,v_2> \mid v_1, v_2 \in CVal \}$.
(We probably don't need $v_1 \in V[\tau_1]$, $v_2 \in V[\tau_2]$. There can't be junk in S by other arguments; eg, type preservation up-front.)
TS: $f\ (\tau_1\times\tau_2)\ (pair) \in E[\alpha]\alpha \mapsto S$.
(Which means that $f\ (\tau_1\times\tau_2)\ (pair) \longmapsto^* v \in S$. One step remains: Prove that pair is in the logical relation.)

TS: $pair \in V[\tau_1\longrightarrow\tau_2\longrightarrow\alpha]\alpha \mapsto S$.
Let $v_1 \in V[\tau_1]$ and $v_2 \in V[\tau_2]$ be given.
TS: $pair\ v_1\ v_2 \in E[\alpha]\alpha \mapsto S$.
$pair\ v_1\ v_2 \longmapsto_* <v_1,v_2> \in S$. (By construction of S.)

— Next time and HW

We'll discuss relational parametricity next Tuesday.

There is homework due in two weeks. The first part follows. The second

part will come next Tuesday. (It will be to generalize the proof of fundamental property to the relational model, in the style with v's rather than λ's and Λ's in the definition of V[τ]ρ.)

HW:

Define:

$\tau_1 + \tau_2 := \forall\alpha.(\tau_1 \to \alpha) \to (\tau_2 \to \alpha) \to \alpha$.
inj_i := λx.bar{inj_i} x(i ∈ {1,2})
bar{inj_i} v := $\Lambda\alpha.\lambda k_1.\lambda k_2.$k_i v (i ∈ {1,2})
case_τ e of $inj_1\ x_1 \Rightarrow e_1 \mid inj_2\ x_2 \Rightarrow e_2 := e\ \tau\ (\lambda x_1.e_1)\ (\lambda x_2.e_2)$.

1. State and prove the β-reduction properties.

2. Use unary parametricity to show

If $\vdash f : \tau_1+\tau_2$,
then f $(\tau_1+\tau_2)$ inj_1 $inj_2 \downarrow$ inj_i(v) for some i, v.

— Asides from Dave:

• All positive (co)inductive types?

Derek mentioned that all products, sums, and positive inductive types are definable in System F. Define μ and ν constructors (when the bound type variables occur positively). State and prove the free theorems.

If necessary, find background info in Chapter 3 of Nax Mendler's thesis (Mendler, 1988); Chapter 11 of Proofs and Types; and Derek.

• Surjective pairs?

Definition (Categorical products):

$p_1 : A \leftarrow A\times B \to B : p_2$ is a product of A and B if
for any Z and $f : A \leftarrow Z \to B : g$,
there exists a unique $\langle f,g\rangle : Z \to A\times B$ satisfying
$p_1 \circ \langle f,g\rangle = f$ and $p_2 \circ \langle f,g\rangle = g$.

We have the right β-reductions

$\pi_1\langle v_1,v_2\rangle \downarrow v_1$
$\pi_2\langle v_1,v_2\rangle \downarrow v_2$

because (we chose the right definition for $\tau_1\times\tau_2$ and) our dynamic semantics includes β-reductions. We do not have the η-expansion

$v \downarrow \langle\pi_1 v,\pi_2 v\rangle$

because our semantics excludes η-expansions.

Surjective pairing *means* $\langle \pi_1 v, \pi_2 v \rangle = v$; that is, categorical products.

From a quick scan, the following paper seems to connect System F (not CBV System F), categorical products, and βη-equality.

> C. Barry Jay and Neil Ghani (1995). The virtues of etaexpansion. Journal of Functional Programming, 5, pp 135–154 doi:10.1017/S0956796800001301