There is a lot more we could talk about wrt free theorems and parametricity in the setting of System F and results of the kind we worked on.

We'll move on.

Later, we might have students prepare talks on specific advanced topics. One that would be very interesting is the work by Andrew Kennedy (POPL'97) on unary parametricity for units of measure. There's a paper coming up in POPL'13 that extends Kennedy's work.

We'll start today with representation independence. The idea is to reason about program correctness in a modular way, using parametricity. This will be a recurring theme throughout the course.

Today, we'll clarify these ideas with an example and then extend the langauge with recursion.

Much of the material from this and the next several lectures is covered by Pitts' chapter in Pierce's ATTPL (2005). We'll do things differently, but Pitts is a good source.

What are the standard rules for existential types? (Aside: Such types are encodable in System F.)

What reasoning principle for representation indepdendence do we want?

— Adding existentials to the langauge

For at least the next several lectures, we'll work with System F extended with $\exists\alpha.\tau$. (Church encodings cease being faithful once we switch to richer languages, so we'll have to start adding primitive types eventually.)

Syntax:

$\tau ::= \cdots \mid \exists\alpha.\tau$
e ::= ··· | pack [σ,e] as $\exists\alpha.\tau$ | unpack_σ e as [α,x] in e'
v ::= ··· | pack [σ,v] as $\exists\alpha.\tau$
K ::= ··· | pack [σ,K] as $\exists\alpha.\tau$ | unpack_σ K as [α,x] in e'

> Some people write unpack e as [α,x] in (e':σ). In examples, we'll often omit types.

Statics:

$$\frac{ftv(\sigma) \subseteq \Delta \quad \Delta; \Gamma \vdash e : \tau[\sigma/\alpha]}{\Delta; \Gamma \vdash \text{pack } [\sigma,e] \text{ as } \exists\alpha.\tau : \exists\alpha.\tau}$$

$$\frac{ftv(\sigma) \subseteq \Delta \quad \Delta; \Gamma \vdash e : \exists\alpha.\tau \quad \Delta, \alpha; \Gamma, x{:}\tau \vdash e' : \sigma}{\Delta; \Gamma \vdash \text{unpack}\_\sigma\ e \text{ as } [\alpha,x] \text{ in } e' : \sigma}$$

Dynamics:

$$\text{unpack}\_\sigma' \text{ (pack } [\sigma,v] \text{ as } \exists\alpha.\tau) \text{ as } [\alpha,x] \text{ in } e' \mapsto e'[\sigma/\alpha][v/x]$$

— Church encoding for existentials

Recall:

$$\exists\alpha.\tau := \forall\beta.(\forall\alpha.\tau \to \beta) \to \beta$$
$$\text{pack } [\sigma,e] \text{ as } \exists\alpha.\tau := \Lambda\beta.\lambda k.k[\sigma]e$$
$$\text{unpack}\_\sigma' \text{ e as } [\alpha,x] \text{ in } e' := e\ \sigma'\ (\Lambda\alpha.\lambda x.e').$$

(Think of $\forall\alpha.\tau \to \beta$ as the type of the pack constructor. That's one way to think of all these Church encodings.)

We could prove the relevant β and η properties.

— Representation independence example

We'll motivate representation independence using our encoding of existentials.

Idea: If we have two modules that both have the same interface (eg, two existential packages with the same existential type), then we want to show them contextually equivalent so long as we can change one to the other in a “coherent way”.

Concrete example (Pitts, Ex 7.3.5):

> Suppose we've encoded integers, products, booleans, and some operations.

Sem := $\exists\alpha.\alpha \times (\alpha \to \alpha) \times (\alpha \to \text{bool})$

(Idea: α represents a bit. The first operation flips the bit.
The second operation tells you if the bit is set.)

sem_1 := pack [bool, <true, λx.not x, λx.x>] as Sem
sem_2 := pack [int, <1, λx.0 - 2*x, λx.x$\geq$0>] as Sem

Idea: $\text{sem}_1 \equiv \text{sem}_2$: Sem.

That is, no well-typed context expecting a Sem can observe a difference between sem_1 and sem_2.

We want "the implementation of the operations are logically related at the type of the operations".

Idea (representation independence principle):
Suppose R $\in$ Cand s.t.
$(v_1,v_2) \in V[\tau]\alpha \to R$
then $(\text{pack}[\sigma_1,v_1] \text{ as } \exists\alpha.\tau, \text{pack}[\sigma_2,v_2] \text{ as } \exists\alpha.\tau) \in V[\exists\alpha.\tau]$
then, by soundness of the LR, we'd have
$\text{pack}[\sigma_1,v_1] \text{ as } \exists\alpha.\tau \equiv \text{pack}[\sigma_1,v_2] \text{ as } \exists\alpha.\tau : \exists\alpha.\tau$

Here's a loose R we might try:

$R := \{ (\text{true},n) \mid n > 0 \} \cup \{ (\text{false},n) \mid n < 0 \}$

Pitts offers the tighter

$R' := \{ (\text{true},m) \mid m = (-2)^n \text{ for some even } n \geq 0 \} \cup$
$\{ (\text{false},m) \mid m = (-2)^n \text{ for some odd } n \geq 0 \}.$

TS: $(e_1,e_2) \in E[\tau]\alpha\mapsto R$
where
e_1 = <true, λx.not x, λx.x>
e_2 = <1, λx.0 - 2*x, λx.x$\geq$0>
$\tau = \alpha \times (\alpha \to \alpha) \times (\alpha \to \text{bool})$.

While we haven't covered proof principles for showing things in the LR at product types, once we do, we'll know
STS: $(\text{true},1) \in V[\alpha]\alpha\mapsto R \wedge$
$(\lambda x.\text{not } x, \lambda x.0\text{-}2^*x) \in V[\alpha\to\alpha]\alpha\mapsto R \wedge$
$(\lambda x.x, \lambda x.x\geq 0) \in V[\alpha\to\text{bool}]\alpha\mapsto R.$

(Completing such a proof is now routine. Interestingly, our interpretation for α expects booleans on the left and integers on the right.)

— Reasoning principles for products and existentials

Let's validate the reasoning princples for pair and existential types.

Lemma (Pairs):
If $(v'_1,v'_2) \in V[\tau']\rho$
and $(v''_1,v''_2) \in V[\tau'']\rho$,
then $(\langle v'_1,v''_1\rangle,\langle v'_2,v''_2\rangle) \in V[\tau'\times\tau'']\rho$.

Recall:
$\sigma\times\tau := \forall\alpha.(\sigma\to\tau\to\alpha)\to\alpha$
$\langle v_1,v_2\rangle := \Lambda\alpha.\lambda k.k\ v_1\ v_2$.

Proof:
TS: $(\lambda\alpha.\lambda k.k\ v'_1\ v''_2, \lambda\alpha.\lambda k.k\ v''_1\ v''_2) \in V[\forall\alpha.(\tau'\to\tau''\to\alpha)\to\alpha]\rho$

Let $R \in Cand$, $(\sigma_1,\sigma_2) \in CTyp$, $(k_1,k_2) \in V[\tau'\to\tau''\to\alpha]\rho,\alpha\mapsto R$.
TS: $(k_1\ v'_1\ v''_2, k_2\ v'_2\ v''_2) \in E[\alpha]\rho,\alpha\mapsto R$
$\Leftarrow$ $(v'_1,v'_2) \in V[\tau']\rho \wedge (v''_1,v''_2) \in V[\tau'']\rho$.
Q.E.D.

This principle, along with the obvious encoding of triples in terms of pairs, justifies one step in our example.

Lemma (Existentials):
If $R \in Cand$
and $(v_1,v_2) \in V[\tau]\rho,\alpha \to R$,
then $(\text{pack}[\sigma_1,v_1]\text{ as }\exists\alpha.\tau, \text{pack}[\sigma_2,v_2]\text{ as }\exists\alpha.\tau) \in V[\exists\alpha.\tau]\rho$.

Proof:
TS: $(\Lambda\beta.\lambda k.k\ \sigma_1\ v_1, \Lambda\beta.\lambda k.k\ \sigma_2\ v_2) \in V[\forall\beta.(\forall\alpha.\tau\to\beta)\to\beta]\rho$.

Let $S \in Cand$, $(k_1,k_2) \in V[\forall\alpha.\tau\to\beta]\rho,\beta\mapsto S$.
TS: $(k\ \sigma_1\ v_1, k\ \sigma_2\ v_2) \in E[\beta]\rho,\beta\mapsto S$.

Unfolding $V[\forall\alpha.\tau\to\beta]\rho,\beta\mapsto S$, picking R as the representation of α, and using irrelevance,
STS: $(v_1,v_2) \in V[\tau]\rho,\alpha\mapsto R$.
Q.E.D.

— What these principles don't give us

What we've proven for each type is, in some sense, a one-way principle. We can show two values are related by the existential type or the pair type. Our principles don't tell us what happens if we're given values related at those types.

One issue: When you're working with Church encodings, you cannot inspect the structure of values.

In other words, we'd like to be able to show
If $(v_1,v_2) \in V[\tau' \times \tau'']\rho$,
then $v_1 = \langle v'_1, v''_1 \rangle$
and $v_2 = \langle v'_2, v''_2 \rangle$
and $(v'_1,v'_2) \in V[\tau']\rho$
and $(v''_1,v''_2) \in V[\tau'']\rho$.

This is NOT TRUE. Just because things are logically related, we don't know they necessarily have predictable syntactic forms.

In reality (ie, for the purpose of verifcation), we don't really care that these values are syntactically of this pair form. All we can do is use them; that is, project their components.

We CAN show:
If $(v_1,v_2) \in V[\tau' \times \tau'']\rho$,
then $(\pi_1\ v_1, \pi_1\ v_2) \in E[\tau']\rho$
and $(\pi_2\ v_1, \pi_2\ v_2) \in E[\tau'']\rho$.

A similar story holds for existentials. (It's a little more annoying to state.)

Once we have (read: need) a language with pairs, sums, existentials, etc as primitive types, we can just define the LR to do exactly what we want.

— Adding pairs, sums, and existentials

From now on, make pairs, sums, and existentials primitive.

$$V[\tau' \times \tau'']\rho := \{ ((v'_1,v''_1), (v'_2,v''_2)) \mid (v'_1,v'_2) \in V[\tau']\rho \wedge (v''_1,v''_2) \in V[\tau'']\rho \}$$

$V[\tau'+\tau'']\rho := \{ (\text{inl } v_1, \text{inl } v_2) \mid (v_1,v_2) \in V[\tau']\rho \}$
$\quad \cup \{ (\text{inr } v_1, \text{inr } v_2) \mid (v_1,v_2) \in V[\tau'']\rho \}$

$V[\exists\alpha.\tau]\rho := \{ (\text{pack}[\sigma_1,v_1] \text{ as } \exists\alpha.\tau_1, \text{pack}[\sigma_2,v_2] \text{ as } \exists\alpha.\tau_2) \mid$
$\quad \exists R \in \text{Cand}. (v_1,v_2) \in V[\tau]\rho,\alpha\mapsto R \}$

Note we've baked in the principles we've just proven. That doesn't mean we've wasted effort. When we extend the metatheory for the logical relation to these new cases, we'll use the same reasoning.

Aside: We'll leave off extending the metatheory for now. Next time, we'll have to change the term relation to deal with recursion. We'll deal with these base types and with recursion in one batch of homework.

— Adding recurison

(Foreshadowing our next lecture.)

Syntax:

$e ::= \cdots \mid \text{fix } f(x).e$
$v ::= \cdots \mid \text{fix } f(x).e$

We'll drop $\lambda x.e$ in favor of the encoding
$\lambda x.e := \text{fix } f(x).e$ where $f \notin fv(e)$.

Statics:

$$\frac{\Delta; \Gamma, f:\sigma\to\tau, x:\sigma \vdash e : \tau}{\Delta; \Gamma \vdash \text{fix } f(x).e : \sigma \to \tau}$$

Dynamics:

$$(\text{fix } f(x).e)\ v \mapsto e[\text{fix } f(x).e/f][v/x]$$

Alternative approach:

We might treat
fix x.e
as a (recursive) expression satisfying
fix x.e $\mapsto$ e[fix x.e/x].

We can perfectly well define substitution e[term/var] instead

of e[val/var].

Once we add μα.τ, we'll be able to encode fix using μ and plain old λ.