What do we do next?

We could work on a few more types. It gets hairy. The encoding of inductive types is not that complicated, but the encoding of the intro and elim forms requires you to define a functorial map over arbitrary types. Interesting, but far afield.

So instead, we'll wrap up this first section of the course. We've been looking at System F's polymorphic types, proving interesting theorems about their inhabitants.

So you can encode a variety of types in System F. What does this give you?

Today, we'll look for useful theorems that might be applicable to doing things with "real" programs. Today, we'll got a little deeper into the study of polymorphic types. Then we'll move on to other uses of relational parametricity.

We'll discuss this idea of free theorems.

• (Wadler, FPCA '89. Theorems for Free!.) (Aside FPCA = Functional Programming and Computer Architecture was a precurser to ICFP.)

Before discussing Wadler's paper, we'll extend the language and model we've defined to include some concrete types (ie, some base types). Derek finds one thing unsatisfying about Wadler's paper: It gives you theorems but doesn't tell you what to do with them.

• (Gill-Launchbury-Peyton Jones, FPCA '93. A short-cut to deforestation.)

This paper validates an optimization. It's actually a very nice paper. They give a proof of one of the free theorems, very much in the style of what we did last time.

— Adding lists to CBV System F

Of course, we can Church-encode lists. We'll see it's actually useful to go between encoded lists and built-in lists.

Syntax:

$\tau ::= \cdots \mid \text{list}(\tau)$

$e, L ::= \cdots \mid \text{nil} \mid \text{cons}(e,e') \mid \text{fold}(e_1,e_2,e_3)$

v ::= nil | cons(v,v')
K ::= ··· cons(K,e) | cons(v,K) | fold(K,e,e') | fold(v,K,e) | fold(v,v',K)

(Aside: You could work with just one form of continuations:
K ::= let x = K in e
and restrict all elimination forms to values; eg, applications have the form v_1 v_2. It's called A-normal form. You can then encode the more general elim forms.)

Fold will be "foldr"; its arguments are what to do with nil, what to do with cons, and the list to fold over.

Statics:

$\Gamma \vdash \text{nil} : \text{list}(\tau)$

$\Gamma \vdash e : \tau$
$\Gamma \vdash L : \text{list}(\tau)$
—
$\Gamma \vdash \text{cons}(e,L) : \text{list}(\tau)$

$\Gamma \vdash L : \text{list}(\tau)$
$\Gamma \vdash n : \sigma$
$\Gamma \vdash c : \tau \to \sigma \to \sigma$
—
$\Gamma \vdash \text{fold}(n,c,L) : \sigma$

Dynamics:

$\text{fold}(v\_n, v\_c, \text{nil}) \mapsto v\_n$

$\text{fold}(v\_n, v\_c, \text{cons}(v,v\_L)) \mapsto v\_c\ v\ (\text{fold}(v\_n,v\_c,v\_L))$

Binary model:

$V[\text{list}(\tau)]\rho = \{ (v_1,v_2) \mid$
$v_1 = v_2 = \text{nil} \vee$
$v_1 = \text{cons}(u_1,L_1) \wedge \text{cons}(u_2,L_2) \wedge (u_1,u_2) \in V[\tau]\rho \wedge (L_1,L_2) \in V[\text{list}(\tau)]\rho \}$

Note the recursive use of $V[\text{list}(\tau)]$ occurs in a strictly postive position. So the recursion is no problem: We're really talking about the least fixed point of all sets satisfying this equation.

Another way is to use the very formal "···" notation:

$$V[\text{list}(\tau)]\rho = \{ ([v_1,\cdots,v\_n], [v'_1,\cdots,v'\_n]) \mid \forall i \in 1..n.\ (v\_i,v'\_i) \in V[\tau]\rho \}$$

It's necessary (but not hard, so omitted) to prove that the fundamental theorem survives this extension.

— Wadler's paper

[Derek projected the paper and talked. I did not attempt to record stuff from Wadler's paper.]

Wadler's theorems are free in the sense that they fall out from FTLR. (As we've seen, they're not exactly free. You have to know what you're doing to prove them.)

Compared to what we've done, Wadler asks what happens when you have polymorphic functions over list types. What properties do you get?

Since you don't know the type of the elements, you can't do anything with them or that depends on what they are.

Rather than try to characterize polymorphic functions intensionally, Wadler did so using commutation properties.

(Aside: Deepak asked if you can state open versions of Wadler's theorems. Derek thinks so.)

The free theorems are nice, but so what?

The crucial bit in the paper is at the end of Section 3.1: "A more convenient version can be derived by specialising to the case where the relation ···". Here, Wadler picks a useful candidate set.

In our notation, he's picking the following interpretation of curly{A}:

$$\text{curly}\{A\} = \{ (v_1,v_2) \mid a\ v_1 \downarrow v_2 \}.$$

So that

$$\text{curly}\{A^*\} = V[\text{list}(\alpha)]\alpha \mapsto \text{curly}\{A\}.$$

If $(xs,xs') \in \text{curly}\{A^*\}$

(meaning a* xs = xs'),
then you'll get the values are component-wise related.

We did a similar thing last time.

Recall our proof of the η property for $\tau_1 \times \tau_2$. When we picked

$S := \{ (v_1,v_2) \mid (v_1[\sigma_2]k_2,v_2]) \in [R] \}$

to instantiate α, it didn't look like we were working with any kind of "map" function. However, we picked a relation $\approx S$ defined by

$v_1 \approx S\ v_2$ iff a $v_1 \approx R\ v_2$
where $a = \lambda x.x[\sigma_1]k_1$.

— Gill's et al paper.

The point of short-cut fusion is to rewrite functions over lists in a "canonical" style (à la a Church encoding for list(τ)) and observe that it's unnecessary to construct the temporary list.

Eg (sum (from a b)) builds a list only to fold over it. We can switch to the Church encoding-inspired implementation from'. The theorem (aka foldr/build rule)

foldr k z (build g) = g k z

states you might as well use the Church-encoded thing directly rather than foldr/build.

(Aside from Deepak: Gill's from' is just a specialization of the fold function to the list (from a b).)

We can prove their theorem (end of Section 3).

NB the theorem can only be stated if we have both a concrete type list(τ) and a Church-encoded type Chlist(τ).

Theorem:
If $\vdash f : Chlist(\tau)$
and $\vdash n : \sigma$
and $\vdash c : \tau \longrightarrow \sigma \longrightarrow \sigma$,
then $fold(n,c,build\ f) \equiv f\ n\ c : \sigma$.
where

build f := f nil cons
cons := λx.λy.Cons(x,y) (a function wrapping the primitive)
Chlist(τ) := ∀α.α ⟶ (τ ⟶ α ⟶ α) ⟶ α (the church encoding)
Chnil := Λα.λn.λc.n
Chcons := λx.λL.Λα.λn.λc.c x L.

Proof:

TS: fold(n,c,f nil cons) ≡ f n c : σ.
By parametricity,
(f,f) ∈ [Chlist(τ)].
Pick R := { (v_1,v_2) | $(\text{fold}(n,c,v_1),v_2) \in [\sigma]$ }
to interpret α.

Aside from Dave: Note that R satisfies
(†) $\forall e_1,e_2.\ (e_1,e_2) \in R \iff (\text{fold}(n,c,e_1),e_2) \in [\sigma]$.
Without observing (†), you'll find this proof quickly becomes mired in explicit reasoning about particular values, expansions, and reductions.

We get
(f[·], f[·]) ∈ [R ⟶ (τ ⟶ R ⟶ R) ⟶ R].
Show
(nil,n) ∈ [R] ∧
(cons,c) ∈ [τ⟶R⟶R]
So,
(f[·]nil cons, f[·]n c) ∈ [R]
⟹ (fold(n,c,f[·]nil cons, f[·]n c) ∈ [σ].

STS: (nil,n) ∈ R ∧ (cons,c) ∈ [τ⟶R⟶R].

— HW: Complete the proof.

— History

There have been a number of follow-up papers related to free theorems and short-cut fusion.

See Patricia Johann's publications. One great thing: She's very pedantic. Two nice examples:
(JFP'98. Short Cut Fusion is Correct.)
(POPL'04. Free theorems in the presences of seq.)