Today we'll talk about recursive types. Before we do general recursive types (probably next time), we'll start with strictly positive recursive types. The reason: That's a fairly straightforward extension of the model we have so far and will let us work out an interesting example. We'll also extend the model with continuations (relatively simple) and work on an example involving both extensions.

— Recursive Types (Strictly Positive)

Syntax:

Types	$\tau ::= \cdots \mid \mu\alpha.\tau$
Terms	$e ::= \cdots \mid \text{fold } e \mid \text{unfold } e$
Values	$v ::= \cdots \mid \text{fold } v$
Eval.Ctxs	$K ::= \cdots \mid \text{fold } K \mid \text{unfold } K$

Syntactic restriction: α may occur only strictly positively in $\mu\alpha.\tau$; that is, it only appears to the right of arrows.

Examples:

$\text{Nat} := \mu\alpha.1+\alpha$

$\text{Stream}(\tau) := \mu\alpha.\tau\times(1 \to \alpha)$

There's another encoding of streams, often seen:

$\text{Stream}'(\tau) := \mu\alpha.\tau x\alpha.$

But we're in a CBV language. Stream'(τ) has no inhabitants. (This'll become clear when we extend the model and even when we write the typing rules.)

Statics:

$\Delta; \Gamma \vdash e : \tau[\mu\alpha.\tau/\alpha]$

—

$\Delta; \Gamma \vdash \text{fold } e : \mu\alpha.\tau$

$\Delta; \Gamma \vdash e : \mu\alpha.\tau$

—

$\Delta; \Gamma \vdash \text{unfold } e : \tau[\mu\alpha.\tau/\alpha]$

Dynamics:

$K[\text{unfold (fold } v)] \mapsto_r K[v]$

How might we extend the model?

Intuitively, we'd like to write

$V[\mu\alpha.\tau]\rho = \{ (\text{fold } v_1,\text{fold } v_2) \mid (v_1,v_2) \in V[\tau[\mu\alpha.\tau/\alpha]]\rho \}$ (I wish)

This is not a valid definition: The type on the right has not gotten smaller and we're defining our LR by induction on τ.

A similar problem came up when we first tried to define $V[\forall\alpha.\tau]\rho$.

We'll do a similar trick in this setting:

$$V[\mu\alpha.\tau]\rho := \text{fp } \lambda R.\{ (\text{fold } v_1,\text{fold } v_2) \mid (v_1,v_2) \in V[\tau]\rho,\alpha\mapsto R \}.$$

where fp represents some unspecified fixed point operator—either greatest or least—(for reasons that will become clear (or unclear) in a moment). The function in this defintion has type VRel ⟶ VRel. In general, we can take a fixed point if the function is monotone. (Thanks to the Tarski fixed point theorem.)

Recall that F monotone means "the more things that are in R, the more are in f(R)". For us, $X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$.

By the Tarski fixed-point theorem, we'll be able to conclude the equation we want:

† $V[\mu\alpha.\tau]\rho = \{ (\text{fold } v_1,\text{fold } v_2) \mid (v_1,v_2) \in V[\tau]\rho,\alpha\mapsto V[\mu\alpha.\tau]\rho \}$

For this to work, we need our F : VRel ⟶ VRel monotone. (We might be able to relax from "strictly positive" to "positive"; eg, permitting types like $(\text{int} \to \alpha) \to \alpha$.)

Aside: With equi-recursive types, things are not so easy. You have a non-trivial syntactic type equality $\mu\alpha.\tau \equiv \tau[\mu\alpha.\tau/\alpha]$. (For all of our languages so far, we have no conversions at the type level. Equi- as opposed to our iso-recursive types don't work that way.) To do this properly, you have to define type equality co-inductively. In the setting of this simple language, equi-recursive types are not a big deal. Type-checking is decidable and admits a simple algorithm.

So, how do we show this desired monotonicity property?

Question: What's fp?
Answer:

We're punting on fp (= either least or greatest fixed point) because Derek cannot tell if it matters. Derek and Neel conjecture that in our language, lfp and gfp coincide.

Conjecture: The least and greatest fixed points coincide.

Evidence: Regardless of which one you choose, just based on † and the Scott Induction Principle for fixed points, you can do "coinductive reasoning" about types like streams. You don't have to rely on this being the greatest fixed point to do such reasoning. With least fixed points, you can rely on the implicitly coinductive Scott Induction Principle.

There's more online. Neel happens to have blogged about this a few weeks ago:
http://semantic-domain.blogspot.de/2012/11/polymorphism-and-limit-colimit.html

For specificity, we'll use the following definition:

$V[\mu\alpha.\tau]\rho := \text{gfp}\ \lambda R.\{ (\text{fold}\ v_1, \text{fold}\ v_2) \mid (v_1,v_2) \in V[\tau]\rho,\alpha\mapsto R \}$.

Derek is unsure how this encoding of recursive types relates to the Church encodings of inductive and coinductive types in (CBV) System F.

Aside: Without recursion in our term language, adding recursive types our way makes little sense. With the Church-encodings, the iteration is built into the encoding.

— Montonicity

Theorem (Monotonicity of the LR):
If α positive in τ, then
i. $V[\tau]\rho$, $E[\tau]\rho$, $K[\tau]\rho$ well-defined
ii. $X \subseteq Y \Rightarrow V[\tau]\rho,\alpha\mapsto X \subseteq V[\tau]\rho,\alpha\mapsto Y$
iii. $X \subseteq Y \Rightarrow K[\tau]\rho,\alpha\mapsto Y \subseteq K[\tau]\rho,\alpha\mapsto X$.
iv. $X \subseteq Y \Rightarrow E[\tau]\rho,\alpha\mapsto X \subseteq E[\tau]\rho,\alpha\mapsto Y$

Aside from Dave:
In class, these were numbered (0,i,iii,ii).

While taking my notes, I added (v) $X \subseteq Y \Rightarrow \text{fp}(F(X)) \subseteq \text{fp}(F(Y))$ where $F : \text{VRel} \to \text{VRel} := \lambda R.\{ (\text{fold}\ v_1, \text{fold}\ v_2) \mid (v_1,v_2) \in V[\tau]\rho,\alpha\mapsto R \}$.
Derek proved the inclusion he needs in the $\mu\beta.\sigma$ case.

Proof:

By lexiographic induction, first on τ and then on the order $[i < ii < iii < iv]$.

Boring case $\tau = \tau_1 \times \tau_2$:

Let $(<v_1,v_2>,<v'_1,v'_2>) \in V[\tau_1 \times \tau_2]\rho,\alpha \mapsto X$.

WK: $(v_1,v'_1) \in V[\tau_1]\rho,\alpha \mapsto X$

$(v_2,v'_2) \in V[\tau_2]\rho,\alpha \mapsto X$

By IH, change X to Y

$(<v_1,v'_1>,<v_2,v'_2>) \in V[\tau_1 \times \tau_2]\rho,\alpha \mapsto Y$.

Interesting Case $\tau = \tau_1 \longrightarrow \tau_2$:

Let $(v_1,v_2) \in V[\tau_1 \longrightarrow \tau_2]\rho,\alpha \mapsto X$

Let $(v'_1,v'_2) \in V[\tau_1]\rho,\alpha \mapsto Y$

TS: $(v_1\ v'_1, v_2\ v'_2) \in E[\tau_2]\rho,\alpha \mapsto Y$.

Since α does not appear in τ,

WK: $(v'_1,v'_2) \in V[\tau_1]\rho,\alpha \mapsto Y$

$\Longrightarrow$ (Irrelevance)

$(v'_1,v'_2) \in V[\tau_1]\rho,\alpha \mapsto X$.

By assumption

$(v_1\ v'_1, v_2\ v'_2) \in E[\tau_2]\rho,\alpha \mapsto X$.

By IH, we're done.

(Failed) Interesting Case $\tau = \mu\beta.\sigma$:

Let $(\text{fold } v_1, \text{fold } v_2) \in V[\mu\beta.\sigma]\rho,\alpha \mapsto X$.

So $(v_1,v_2) \in V[\sigma]\rho,\alpha \mapsto X,\beta \mapsto (V[\mu\beta.\sigma]\rho,\alpha \mapsto X)$

$= \quad V[\sigma]\rho,\beta \mapsto (V[\mu\beta.\sigma]\rho,\alpha \mapsto X),\alpha \mapsto X$.

By IH,

$= \quad V[\sigma]\rho,\beta \mapsto (V[\mu\beta.\sigma]\rho,\alpha \mapsto X),\alpha \mapsto Y$.

⋯ oops. There's still an X ⋯

Interesting Case $\tau = \mu\beta.\sigma$:

Let $(\text{fold } v_1, \text{fold } v_2) \in V[\mu\beta.\sigma]\rho,\alpha \mapsto X$

$= \quad \text{gfp}(\lambda R. \{(\text{fold } w_1, \text{fold } w_2) \mid (w_1,w_2) \in V[\sigma]\rho,\alpha \mapsto X,\beta \mapsto R\})$.

STS:

$\quad \text{gfp}(\lambda R. \{(\text{fold } w_1, \text{fold } w_2) \mid (w_1,w_2) \in V[\sigma]\rho,\alpha \mapsto X,\beta \mapsto R\})$.

$\subseteq \quad \text{gfp}(\lambda R. \{(\text{fold } w_1, \text{fold } w_2) \mid (w_1,w_2) \in V[\sigma]\rho,\alpha \mapsto Y,\beta \mapsto R\})$.

This follows from the IH; the claim

If $F \le G$ (ie, $\forall X.\ F(X) \subseteq G(X)$),

then $\text{gfp } F \subseteq \text{gfp } G$;

and basic properties of lattices.

There are more boring cases; they should go through.

Argument for K, part (iii):

    Aside: In class, Derek dealt with things out of order (E[] before K[]).

    Let $(v_1,v_2) \in V[\tau]\rho,\alpha\mapsto X$
    WK: $(K_1,K_2) \in K[\tau]\rho,\alpha\mapsto Y$
    TS: $K_1[v_1]\downarrow\downarrow K_2[v_2]$
    $\Leftarrow$ (By IH part (i))
        $(v_1,v_2) \in V[\tau]\rho,\alpha\mapsto Y$.

Argument for E, part (iv):

    Let $(K_1,K_2) \in K[\tau]\rho,\alpha\mapsto Y$
    WK: $(e_1,e_2) \in E[\tau]\rho,\alpha\mapsto X$
    TS: $K_1[e_1]\downarrow\downarrow K_2[e_2]$
    $\Leftarrow$ (By IH part (iii))
        $(K_1,K_2)\in K[\tau]\rho,\alpha\mapsto X$.

Q.E.D.

— Example: Stream types

Greatest fixed points give you a coinduction principle.

The idea behind Tarski's fixed point theorm:

    TS: $x \in \text{gfp } F$
    STS: $x \in A \subseteq F(A)$.

(Called a post-fixed point of F. Sometimes written "A is F-consistent".)

In other words, if you assume the things in A are related, then in fact they behave in a related way. You get to make that assumption (the thing you're proving) in a coinductive way. Tarski's theorem says the gfp is the union of all such A's. (Kleene's starts with the full set and takes the intersection at the limit. It's only for continuous functions.)

Define:

    $\tau := \mu\alpha.1\longrightarrow(\text{int} \times \alpha)$
    ones : τ := fold (fix f().<1,fold f>)
    twos : τ := fold (fix f().<2,fold f>)

```
succ : τ ⟶ τ := fix f(s).
        let c = unfold (s()) in
        fold (λ().<1 + π₁c, f(π₂c)>)
```

Prop:

$\quad$ succ ones ≡ twos : τ.

Proof:

First, we evaluate the left hand side to a value so we can work in the value relation.

$$\begin{array}{ll} & \text{succ ones} \\ \longmapsto^* & \text{let } c = \langle 1,\text{ones}\rangle \text{ in fold } (\lambda().\langle 1 + \pi_1 c, f(\pi_2 c)\rangle \\ \longmapsto^* & \text{fold } (\lambda().\langle 1+\pi_1\langle 1,\text{ones}\rangle, \text{succ}(\pi_2\langle 1,\text{ones}\rangle)) \\ =: & \text{twos}'. \end{array}$$

STS: $(\text{twos}',\text{twos}) \in V[\tau] = V[\mu\alpha.1\longrightarrow(\text{int} \times \alpha)]$.

(Aside: We could have reasoned with ≡ up front to deal with these β reductions. In the middle of a proof, you can't generally do that. However with models closed under ciu-equivalence (such as the typed model) we can often use ciu-equivalence to β-reduce open terms mid-proof.)

We'll use Tarski's fixed point theorem. Even if we had used the least fixed point, we could still do the proof with our unrolling property †.

So we'll pick $x := (\text{twos}',\text{twos})$,
pick $A := \{ (\text{twos}',\text{twos}') \}$,
$F := \lambda R. \{(\text{fold } v_1,\text{fold } v_2) \mid (v_1,v_2) \in V[\sigma]\alpha\mapsto R \}$.
where $\sigma := 1\longrightarrow(\text{int} \times \alpha)$
STS: $A \subseteq F(A)$
TS: $(\lambda().\langle 1+\pi_1\langle 1,\text{ones}\rangle, \text{succ}(\pi_2\langle 1,\text{ones}\rangle)$,
$\quad$ fix $f().\langle 2,\text{fold } f\rangle) \in V[\sigma]\alpha\mapsto A$.
⇐ (β reducing to values)
$\quad (\langle 2,\text{twos}'\rangle,\langle 2,\text{twos}\rangle) \in V[\text{int}\times\alpha]\alpha\mapsto A$
⇐ $\quad (\text{twos}',\text{twos}) \in V[\alpha]\alpha\mapsto A = A$.

Q.E.D.

Alternative Non-proof (Unfolding τ and applying the Scott Induction Principle):

Define some notation

$\quad$ twos =: fold F

twos' =: fold F'

STS: (F',F) ∈ V[1⟶(int × τ)].

How does Scott Induction work?

To show two functions are related, it suffices to show all their finite approximations are related.

STS: ∀n. (F'_n,F_n) ∈ V[1⟶(int×τ)].

By induction on n.

Case 0: Divergent function on both sides. Trivial.

Case:

Assume ∀k<n. (F'_k,F_k) ∈ V[1⟶(int×τ)].
TS: (F'_n,F_n) ∈ V[1⟶(int×τ)].

Applying these to related unit values, we get bodies that must be shown related:
STS: (<2,twos'>, <2,fold F_{n-1}>) ∈ E[int×τ].

Since twos' is not recursive, all its approximiations except F'_0 are identical. In particular, twos' = fold F'_{n-1}.

··· Damn: We can't do the case for n=1 as there's no way to lift F'_0 to F'_1. ···

End-Non-Proof

Derek thinks we can fix the preceding non-proof. He sees two potential fixes.

The more involved way:

Prove a one-sided version of Scott Induction allowing us to keep F' fixed and only consider approximations F_i of F.

Nevermind: This won't work for n=1 (thanks Beta).

The other approach: HW for Tuesday.

You might need to transitively compose some proofs.

The point is not this example. It's to show that we don't rely on "gfp".

Idea: We have

twos = fold F

twos' = fold F'

Define a twos'' between the two, prove directly that (twos', twos'') are related and use scott induction to prove that (twos'', F) are related.

— Aside: Stuck states

This came up after class.

Our notion of cotermination

$$e{\downarrow\downarrow}e' \text{ iff } (e{\downarrow} \Leftrightarrow e{\downarrow})$$

permits stuck states. They cause problems in the untyped model. For example, the proof of

$$\begin{array}{ll} & (\forall v_1,v_2.\ (v_1,v_2) \in E[\tau_1]\rho \Rightarrow (v_1,v_2) \in V[\tau_1]\rho) \\ \wedge & (\forall v_1,v_2.\ (v_1,v_2) \in E[\tau_2]\rho \Rightarrow (v_1,v_2) \in V[\tau_2]\rho) \\ \Rightarrow & (\forall v_1,v_2.\ (v_1,v_2) \in E[\tau_1{\times}\tau_2]\rho \Rightarrow (v_1,v_2) \in V[\tau_1{\times}\tau_2]\rho) \end{array}$$

does not go through.

Fix: Define cotermination to rule out stuck states. One alternative:

$$e{\downarrow\downarrow}e' :\Leftrightarrow (e{\downarrow} \wedge e'{\downarrow}) \vee (e{\uparrow} \wedge e'{\uparrow})$$

where $e{\uparrow}$ (read “e diverges”) means e reduces indefinitely.