— Syntactic minimal invariants

The example I'll give next time is the so-called syntactic minimal invariants theorem. The theorem (presented in the LSLR paper to be discussed next time) reads:

If $\tau = \mu\alpha.1+(\alpha\longrightarrow\alpha)$,
then
$\lambda x{:}\tau.x \equiv$
fix $f(x{:}\tau)$.
case (unfold x) of
$\text{inj}_1\ () \Longrightarrow \text{fold}(\text{inj}_1\ ())$
| $\text{inj}_2\ g \Longrightarrow$ (* $g : \tau\longrightarrow\tau$ *)
(* basically, we do an η-expand of $\tau\longrightarrow\tau$ *)
$\text{fold}(\text{inj}_2(\lambda y{:}\tau.f(g(f\ y))))$.

It's kind of like an extensionality (or η-expansion) property. You need fixed points to even state it. Note τ is kind of like the type representing domains for the untyped λ-calculus. In the inj_2 case, we basically want to apply g. It gets somewhat annoying to prove the theorem in the model directly.

This theorem is related to a proof method Derek hasn't taught us. See Pitts' "Relational properties of domain" 1996. The approach boils down to trying to deal with both positive and negative occurrences of type variables in types using more advanced (categorical) techniques. Later, Birkedal and Harper (1999) presented a syntactic version of Pitts' "minimal invariants" construction. Later, Crary and Harper (2007) extended the B&H approach to general recursive types and polymorphism; that is, to the same exact setting we're considering with the step-indexed model. They can prove the same kind of examples we can prove with Amal's method. In Derek's opinion, Amal's method is easier to understand. (This minimal invariants property underlies all this work.)

Aside: Pitts has recently argued that step-indexing combined with biorthogonality is the way to go. (There may well be examples that can be proven with "minimal invariants" but not with step-indexing, but Derek can't say off the top of his head.)

— Encoding fixed points

Recall, it wasn't easy to prove (directly via admissibility) that succ ones ≡ twos. We'll give a third proof that relies on the basic coninductive reasoning we get from the step-indexed model.

Amal's model doesn't assume the language has fixed points. Given recursive types, we can encode fixed points. In case you haven't seen that encoding:

fix f(x).e := λy.(unfold v) v y
v := fold(λz.(λf.λx.e) (λy.(unfold z) z y))
where y, z ∉ fv(e).

Basically, you write down the CBV version of the Y combinator, inserting folds and unfolds in the right places. If you ignore the outermost η-expansion λy.— y, we have basically

λz.(λf.λx.e)(z z) =: F

and

Y F = (λz.F (z z))(λz.F (z z)).

We can prove the property we want fix to have:

(fix f(x).e) v ↦+ e[fix f(x).e/f,v/x].

The important thing is this takes a positive number of steps to unfold.

— Example: ones and twos

Recall:

τ = μα.1⟶(int×α)
ones = fold(fix f().<1,fold f>
twos = fold(fix f().<2,fold f>
succ = fix f(s:τ):τ.
let c = (unfold s)() in
fold λ().<1+π_1c,f(π_2 c)>

We want to prove

twos ≈ succ ones : τ

⇐

twos ≾ succ ones : τ ∧
succ ones ≾ twos : τ.

Consider the first conjunct. We'll sketch half of the proof to show how such things go.

TS: twos ≾ succ ones : τ.

Let n ∈ ℕ.
TS: (n,twos,succ ones) ∈ E[τ].

Set twos' := fold ($\lambda()$.<1+π_1<1,ones>,succ(π_2<1,ones>)>).
By closure under expansion,
STS: (n,twos,twos') $\in V[\tau]$.
$\Leftarrow$ (n-1,
fix f().<2,fold f>,
$\lambda()$.<1+π_1<1,ones>,succ(π_2<1,ones>)>) $\in V[1 \to (\text{int} \times \tau)]$.

Let $j \leq n-1$.
By closure under expansion (and downward closure),
STS: (j,
<2,fold f>,
<1+π_1<1,ones>,succ(π_2<1,ones>)>) $\in E[\text{int} \times \tau]$.
$\Leftarrow$
(j,
<2,twos>,
<2,succ ones>) $\in E[\text{int} \times \tau]$
$\Leftarrow$
(j,
<2,twos>,
<2,twos'>) $\in E[\text{int} \times \tau]$
$\Leftarrow$ (j,twos,twos') $\in V[\tau]$.

On the left-hand side we took a number of reduction steps in unrolling the fix. (Extra steps on the right don't cause complications.) Why is this still valid reasoning? By downward closure.

At this point in the proof, it looks like we're trying to prove twos$\approx$twos' assuming twos$\approx$twos'. But j<n. So when we introduce n, we need to insert an induction. When we reach such a loop in a step-indexed proof, it's critical that we can prove the base case. We then reason with the induction principle

$P(0) \wedge \forall j<n.P(j) \Longrightarrow \forall n.P(n)$.

For our proof, the base case is trivial.

Note that while we're doing induction, the reasoning is coinductive. We made some progress (were productive), then relied on induction.

— Completeness and Transitivity

Completeness of the LR: How does that story play out for

step-indexing? It's (mildly) interesting and can be discussed without a lot of technical detail.

Combining step-indexing with biorthogonality is interesting and very useful if you work with a language that needs $\top\top$-closure.

There's also the point of transitivity of the LR. Appel and McAllister's original model was conjectured it was a PER-model. Amal observed the model fails to be transitive. She showed one way of regaining transitivity for the language with just recursive types and not with polymorphic types.

Why care about transitivity?

It depends on your application. If you only care about contextual equivalences, you don't need a transitive proof method. You can use the transitivity of contextual equivalence/approximation to connect your subproofs.

If you cannot rely on contextual equivalence, then you want the LR to be transitive; eg, if you're working with different langauges (no single notion of context) or you're using the LR to give the model of some equational theory.

Amal used a trick to show transitivity for her LR for recursive types. Basically, bake syntactic typing conditions into the model. (Ironically, the original motivation for the step-indexed model was to avoid the syntactic typing judgement in foundational PCC.) Transitivity is generally interesting. Amal's specific technique less so.

So we'll talk about completeness, achieved by combining the model with $\top\top$-closure.

— Completness

Let's briefly discuss completeness.

Basically, we could use the device of CIU-approximation we've seen before.

Recall $\equiv_{ctx} \Longrightarrow \equiv_{ciu}$ was easy. Recall $\approx \Longrightarrow \equiv_{ctx}$. What were the key things we used in proving $\equiv_{ciu} \Longrightarrow \approx$?

Intuitively, we proved

$e_1 \equiv_{ciu} e_2 : \tau$

$e_1 \approx e_1 : \tau$

—

$e_1 \approx e_2 : \tau$

using $\top\top$-closure. In words, we used $(e_1,e_1) \in E[\tau]$ together with

$e_1 \equiv_{u} e_2 : \tau$

to prove

$e_1 \approx e_2 : \tau$.

Intuitively, we now need

$e_1 \precsim_{ciu} e_2 : \tau$

$e_1 \precsim_{log} e_1 : \tau$

—

$e_1 \precsim_{log} e_2 : \tau$.

Without $\top\top$-closure, existential types cause problems.

In Amal's TR, she writes out the flawed proof and points out the precise problem. Interestingly, all the other cases go through. Roughly, this suggests step-indexing "gets at" some of the same things that $\top\top$-closure gets at. Step-indexing doesn't quite give you admissibility—like $\top\top$-closure does—it just takes "reasoning at the limit" off the table. Somehow, to a large extent, step-indexing gives you the equivalence-preserving property. For purposes of time, we'll not work through the proofs Amal gives (for all but existential types) of

If $(n,e_1,e_2) \in E[\tau]\rho$

and $e_2 \precsim_{ciu} e_3 : \rho_2\tau$,

then $(n,e_1,e_3) \in E[\tau]\rho$.

If you look at the Crary-Harper paper (same language, basically), they just leave out existentials and say "ah, you can Church-encode them". That works fine and the corresponding LR is complete. The model of the Church encoding for existentials does not say there exists blah; it just says for any client, the client cannot tell the difference blah. In that respect, $\top\top$-closure corresponds to the Church-encoding. Both use a form of double-negation.

— Combining step-indexing and $\top\top$-closure

We'll stick with an untyped model (but the pros and cons are subtle).

Recall we defined

$e_1 \downarrow\downarrow e_2$ as $e_1\downarrow \iff e_2\downarrow$.

That works fine in a typed language. In an untyped language, it works or not based on how you define $e_1\downarrow$. In the untyped model, terms can get stuck. You have to decide if getting stuck is part of "terminates". We'd like to say "you must eliminate the possibility that terms are stuck".

Without going through the exercise of working out the model with stuck terms to see what, if anything, goes wrong, it's hard to say crisply what goes wrong.

In the untyped version of the Pitts-style model, we'll avoid stuck states:

$e_1 \downarrow\downarrow e_2 := (e_1\downarrow \wedge e_2\downarrow) \vee (e_1\uparrow \wedge e_2\uparrow)$

Where $e\uparrow$ (diverges) means infinte reduction.

Derek's gut feeling: We can finesse the superficial complication of this verison (i.e., we must perform case analyses).

We'll fudge. We'll present a typed model without types and without the case distinctions in $\downarrow\downarrow$.

What should change from the model we have now to add $\top\top$-closure? The E relation.

We want:

$E[\tau]\rho := \{ (n,e_1,e_2) \mid \forall K_1,K_2.\ (n,K_1,K_2) \in K[\tau]\rho \Longrightarrow K_1[e_1] \downarrow\downarrow n\ K_2[e_2] \}$

But how do we define $\downarrow\downarrow n$?

One nice thing given us by $\top\top$-closure: We can define things symmetrically again. Where

$e_1 \downarrow\downarrow n\ e_2 := (e_1\downarrow k<n \Longrightarrow e_2\downarrow) \wedge (e_2\downarrow k<n \Longrightarrow e_1\downarrow)$

That's fine in the typed model. We write $e_1\downarrow k<n$ to mean e_1 terminates to a value in $k<n$ steps. (We're ruling out stuck states.)

For the untyped model, we'd want

$e_1 \downarrow\downarrow n\ e_2 := (e_1\downarrow \wedge e_2\downarrow) \vee (e_1 \mapsto n\ e'_1 \wedge e_2 \mapsto n\ e'_2)$.

This seems awkward to work with because you'd have to do case analysis throughout your proofs. In fact you can avoid that with some lemmas. (In the typed setting, these are equivalent.)

$K[\tau]\rho := \{ (n,K_1,K_2) \mid \forall v_1,v_2.\forall j\leq n.$

$(j,v_1,v_2) \in V[\tau]\rho \Longrightarrow K_1[v_1] \downarrow\downarrow j\ K_2[v_2]\ \}$

You can use an attempt to prove downard closure to decide where to put the quantifications over j. But there's also an intuition.

Recall from last time that we don't care if E is downward closed but we do care that K is. We don't generally expect terms to be "portable". The only thing we are ever "given" as hypotheses are values (in CBV). To show terms are related at n steps, we only care about running them right now. Continuations are like functional values. They have to be applicable later. (Especially since we don't know how long it will take for the argument passed to the continuation to evaluate.)

That's it. With general Kripke structures, we'd quantify over future worlds in $K[\tau]\rho$.

— Metatheory

Given the time we have remaining, lets just prove some compatibility lemmas. The structure of the proofs will be different now that we have $\top\top$-closure.

We'll start with the bind lemma.

Lemma (Bind for closed terms):
If $(n,f_1,f_2) \in V[\sigma\to\tau]\rho$
and $(n,v_1,v_2) \in V[\sigma]\rho$,
then $(n,f_1\ v_1,f_2\ v_2) \in E[\tau]\rho$.

Pf:
By our first assumption,
$f_1 = \lambda x.e_1 \wedge$
$f_2 = \lambda x.e_2 \wedge$
$(n,e_1[v_1/x],e_2[v_2/x]) \in E[\tau]\rho$.
WK: $f_1\ v_1 = (\lambda x.e_1)\ v_1 \mapsto e_1[v_1/x] \wedge$
$f_2\ v_2 = (\lambda x.e_2)\ v_2 \mapsto e_2[v_2/x]$.
We want a closure under expansion property so that we can conclude
$(n,f_1\ v_1,f_2\ v_2) \in E[\tau]\rho$.

Let $(n,K_1,K_2) \in K[\tau]\rho$.
TS: $K_1[f_1\ v_1] \downarrow\downarrow(n+1)\ K_2[f_2\ v_2]$
$\Leftarrow$ $K_1[e_1[v_1/x]] \downarrow\downarrow n\ K_2[e_2[v_2/x]]$.

(Such closure under expansion properties work out with either variant of $\downarrow\downarrow n$.)

Q.E.D.

The POPL'11 paper includes a general version of this property that was very useful. It basically said that if two programs coterminate at n steps and they both expand for k steps, then they coterminate for n+k steps.