Today Derek will sketch his proof for the unwinding theorem.

Some time soon, he will try to show how we use the LR to reason about equivalences in the presence of recursion. (Pitts doesn't much /use/ the LR in his chapter.)

Soon, Derek'll introduce types over which you'd write recursive programs. We'll talk about inductive and coinductive types. Derek doesn't know of a particularly clear presentation of this. On Tuesday, he'll extend the langauge and model with primitive inductive and coinductive types.

Today, we'll take a detour. We'll extend the language and model with continuations (in particular, call/cc).

— Long proofs and the Unwinding Theorem

Nobody completed the homework.

One of the things Derek is trying to teach is how to “separate the wheat from the chaff”. In doing a proof, try to identify what are the interesting bits and what's the boring stuff (that should be skipped).

Derek believes the basic argument for the Unwinding Theorem is straightfoward.

The phrases “observe that φ” and “note that φ” in published proofs usually mean someone's intuition says φ is true, but they've not proven it as it's too boring. (In making this observation, Derek is just being honest.)

Theorem (Unwinding): Let F = fix f(x).hat(e).
1. If e[F/f]↓, then ∃n. e[F_n/f]↓.
2. If e[F_n/f]↓, then e[F/f]↓.

Proof:

Intuition (1): Suppose e[F/f] $\mapsto^*$ v in n steps, then F was unrolled at most n times, so e[F_n/f] ought to terminate.

Intuition (2): The termination behavior of F_n is worse than that of F.

Derek's idea for proving both involves essentially the same argument.

This becomes tricky as you do it operationally. You'd like to reason as if you had a simulation relation between F and F_n. But F_{n+1} unrolls to F_n.

We'll set up a simulation relation. A kind of congruence permitting the two programs to differ in a limited way.

There are two directions of approximation.

For one, define the relation

$e_1 \sqsupseteq n\ e_2 :\Longleftrightarrow$
The congruence closure of
$F \sqsupseteq n\ F\_m$ if $m \geq n$.
$F\_m \sqsubseteq n\ F$ if $m \geq n$.

Congruence closure: The smallest relation closed under the congruence rules.

Pf of (1). Suppose $e[F/f] \downarrow m$ ($\Longleftrightarrow$ takes exactly m steps to terminate).

We'll generalize to the following.
(†) $\forall k.\forall n.\ n>k \wedge e_1 \sqsupseteq n\ e_2 \wedge e_1 \downarrow k \Longrightarrow e_2\downarrow$.

If we can prove this, then instantiating it with k=m, n=m+1, $e_1 = e[F/f]$, $e_2 = e[F\_n/f]$ reduces our proof burden to $e_1 \sqsupseteq n\ e_2$.

Observe that $e[F/f] \sqsupseteq n\ e[F\_n/f]$ since $\sqsupseteq n$ is a congruence (and is substitutive, etcetera).

Proof of (†).
There's a boring part and an interesting part. (We'll have an "observe" in our proof.)

Proof by induction on k.

Case k = 0:
e_1 a value. since $e_1 \sqsupseteq n\ e_2$, observe that e_2 value.

Case k = k'+1.

Then $e_1 \mapsto e'_1 \downarrow k'$.

Case (the interesting case):

$e_1 = K_1[F\ v_1] \wedge$

$e_2 = K_2[F\_m\ v_2]$

where

$K_1 \sqsupseteq n\ K_2 \wedge$

$v_1 \sqsupseteq n\ v_2 \wedge$

$m \geq n$.

Then $e'_1 = K_1[\text{hat}\{e\}[F/f][v_1/x]]$.

$e_2 \mapsto e'_2 = K_2[\text{hat}\{e\}[F\_\{m\text{-}1\}/f][v_2/x]]$.

Aside: We've reduced k to k'. We've reduced m to m-1. We've got to reduce n.

To instantiate the IH, pick n' := n-1
and k' := k-1.
To apply the IH,
TS: $e'_1 \sqsupseteq n'\ e'_2$.
WK: $K_1 \sqsupseteq n\ K_2 \Longrightarrow K_1 \sqsupseteq n'\ K_2$.
WK: $v_1 \sqsupseteq n\ v_2 \Longrightarrow v_1 \sqsupseteq n'\ v_2$.
WK: $F \sqsupseteq \{n\text{-}1\}\ F\_\{m\text{-}1\} \Longleftarrow m\text{-}1 \geq n\text{-}1$.

By induction, $e'_2\downarrow \Longrightarrow e_2\downarrow$.

Case (some other β reduction):

$e_1 = K_1[v_1\ v'_2]$

$e_2 = K_2[v_2\ v'_2]$

Where all these things are congruences:

$K_1 \sqsupseteq n\ K_2$

$v_1 \sqsupseteq n\ v_2$

$v'_1 \sqsupseteq n\ v'_2$

WK: $v_1 = \text{fix}\ f(x).\text{hat}\{e_1\}$

$v_2 = \text{fix}\ f(x).\text{hat}\{e_2\}$

where $\text{hat}\{e_1\} \sqsupseteq n\ \text{hat}\{e_2\}$

By assumption

$e_1 \mapsto K_1[\text{hat}\{e_1\}[v_1/f][v'_1/x]] \downarrow\{k\text{-}1\}$

Observe

$e_2 \mapsto K_2[\text{hat}\{e_2\}[v_2/f][v'_2/x]]$.

TS: $K_2[\text{hat}\{e_2\}[v_2/f][v'_2/x]] \downarrow$.

By properties of CC,
$\text{hat}\{e_2\}[v_2/f][v'_2/x] \sqsupseteq n \text{ hat}\{e_2\}[v_2/f][v'_2/x]$.
TS: $k-1 < k < n$.

Pick $k' = k-1$, $n' = n$. (Deepak says $n' = n-1$.)
By induction: $K_2[\text{hat}\{e_2\}[v_2/f][v'_2/x]] \downarrow$.

Case (other cases): Similar to the previous case.
Q.E.D.

— Homework for next Tuesday: Prove (2).

— Contextual equivalence at existential types

Recall the brain teaser:

$\tau = \exists\alpha.(\alpha\to\alpha)\to\text{bool}$
$v_1 = \text{pack } [\text{unit}, \lambda f.\ f() = ()] \text{ as } \tau$
$v_2 = \text{pack } [\text{bool}, \lambda f.\ f\ \text{true} = \text{true} \wedge f\ \text{false} = \text{false}] \text{ as } \tau$

The false assumption: The context has no access to α.

C = unpack • as [α,x] in (x : (α → α) →bool)
x (λy:α. (we now have a y:α and can fool the thing!)
if x (λz.y) then y else ⊥)

The point: Contextual (in)equivalence for arbitrary values of existential type can be nontrivial.

Here's a step toward a new brain teaser:

$\tau = \exists\alpha.(\alpha\to\alpha)\to\text{bool}$
$v_1 = \text{pack } [\text{int}, \lambda f.\ f(0) = 0 \wedge f(1) = 1] \text{ as } \tau$
$v_2 = \text{pack } [\text{bool}, \lambda f.\ f\ \text{true} = \text{true} \wedge f\ \text{false} = \text{false}] \text{ as } \tau$

This can be proven using representation independence.
Pick R = { (0,true), (1,false) }.
Then $f \in [R\to R]$.

Now consider

$\tau = \exists\alpha.(\alpha\to\alpha)\to\text{bool}$

$v_1 = \text{pack } [\text{int}, \lambda f.\ f(0) = 0 \wedge f(1) = 1 \wedge f(2) = 2] \text{ as } \tau$
$v_2 = \text{pack } [\text{bool}, \lambda f.\ f\ \text{true} = \text{true} \wedge f\ \text{false} = \text{false}] \text{ as } \tau$

Do we have $v_1 \equiv v_2 : \tau$ or not?

Pitts' chapter includes one or two versions of our original example.

The verison in System F with fix:

$\tau = \exists\alpha.\ (\alpha \to \text{bool}) \to \text{bool}.$
$v_1 = \text{pack } [\text{void}, \lambda f.\bot] \text{ as } \tau$
$v_2 = \text{pack } [\text{bool}, \lambda f.(f\ (\text{true}) \neq f\ (\text{false})) \text{ orelse } \bot] \text{ as } \tau$

Idea: The only thing the context can supply is the divergent function or a constant function. In either case v_2's function always diverges.

We'll see $v_1 \equiv v_2 : \tau$. (‡)

[Aside for Neel: A version in System F:
$\tau = \exists\alpha.\ (\alpha \to \text{bool}) \to \text{bool}.$
$v_1 = \text{pack } [\text{void}, \lambda f.\text{false}] \text{ as } \tau$
$v_2 = \text{pack } [\text{bool}, \lambda f.f\ (\text{true}) \neq f\ (\text{false})] \text{ as } \tau$
We could show $v_1 \equiv v_2 : \tau$.]

The key to proving ‡ simply is to use ≡'s transitivity.

Our untyped ≈ fails to be transitive.
But we know: $e_1 \approx e_2 \wedge e_2 \approx e_3 \Longrightarrow e_1 \equiv_{ctx} e_3$.

So can we come up with intermediate expressions helping us morph v_1 to v_2?

A good first step: Change void to bool.

$\tau = \exists\alpha.\ (\alpha \to \text{bool}) \to \text{bool}.$
$v'_1 = \text{pack } [\text{bool}, \lambda f.\bot]$
$v_1 = \text{pack } [\text{void}, \lambda f.\bot]$

$v'_2 = \text{pack } [\text{bool}, \lambda f.(f\ (\text{true}) \neq f\ (\text{true})) \text{ orelse } \bot]$
$v_2 = \text{pack } [\text{bool}, \lambda f.(f\ (\text{true}) \neq f\ (\text{false})) \text{ orelse } \bot]$

To show $v'_1 \approx v_1$, pick $R = \emptyset$ (the only choice).

To show $v'_2 \approx v_2$, pick R = {(true,true), (false,true)}.

f_1 true $\downarrow$ v and f_2 true $\downarrow$ v
f_1 false $\downarrow$ v' and f_2 true $\downarrow$ v'
by determinism of evaluation, v = v'.
so f_1 true = f_1 false. so the v'_2 always diverges.

This is a much shorter proof than the one Pitts offers.

This example also serves as a counterexample to the transitivity of our untyped logical relation.

It's also a counterexample to the transitivity of the typed /value/ logical relation. The typed term relation is transitive. The typed value relation is not transitive at existential types.

(Aside: The $\top\top$-closed term relation coincides with contextual equivalence. But you do representation independence proofs in the value relation.)

— Coincidence for existentials

Let's try to do the proof of the coincidence property for existentials, to see where things break down.

Idea: Clients of an abstract datatype may be unable to differentiate two different implementations for “very bizzare reasons”.

Idea: Knowing two existentials are contextually equivalent doesn't tell you very much. It could be just a syntactic fact that happens to be true because the type system fails to make some distinction. (Semantically, you'd like to say two things are contextually equivalent if there's a representation independence proof connecting them.)

We'll start by stating this in a way that can't be right.

We want to show “something like”:
If $(v_1,v_2) \in E[\exists\alpha.\tau]\rho$,
then $(v_1,v_2) \in V[\exists\alpha.\tau]\rho$.

We really want to put some condition on τ. We want its representatives to be $\top\top$-closed. We might state this (fuzzily):
$\forall\rho.\ \forall v_1,v_2.\ (v_1,v_2) \in E[\tau]\rho \Longrightarrow (v_1,v_2) \in V[\tau]\rho'$

To make progress, suppose we're doing this for the typed relation. We

know $v_i = \text{pack}\ [\sigma_i, w_i]\ \text{as}\ \cdots$ for $i \in \{1,2\}$.

Inuitively: There's nothing in the structure of these packs that tells us what relation to use.

You'll find in the homework a similar problem with sum types (but it's solvable). In the case for sums, you can say $\exists i \in \{1,2\}$ such that you can use inj_i in both. The choice of 1, 2 is evident in the structure of the term. Put another way, the logical relation is “proof relevant” whereas with existentials it's “proof irrelevant”.

Aside: There's recent work on “proof relevant logical relations” trying to address this issue with existentials (warning: categories):

> Nick Benton, Martin Hoffmann, and Vivek Nigam.
> Abstract Effects and Proof-Relevant Logical Relations.
> Draft, 2012.
> http://research.microsoft.com/en-us/um/people/nick/setoids.pdf