Modified: 20121203

We'll start with the proof of the triangle theorem ($\equiv$u = $\approx$ = $\equiv$ctx).

We'll contrast our setup with Pitts'.

We'll return to some questions raised a few lectures ago about admissibility.

We'll maybe consider some examples involving existential types.

— Completeness of the logical relation

Recall: $\vdash \gamma : \Gamma :\Longleftrightarrow \forall(x{:}\tau)\in\Gamma.\ \vdash \gamma x : \tau$.
Notation: $\vdash \delta : \Delta :\Longleftrightarrow \delta \in \Delta \longrightarrow$ CTyp.

Recall our definitions:

$e_1 \equiv$u $e_2 : \tau$ if $\forall K \div \tau.\ K[e_1] \downarrow\downarrow K[e_2]$
$\wedge \vdash e_1 : \tau \wedge \vdash e_2 : \tau$.

$\Delta; \Gamma \vdash e_1 \equiv$u $e_2 : \tau$ if
$\forall \delta : \Delta, \gamma : \delta\Gamma.\ \vdash \delta\gamma e_1 \equiv$u $\delta\gamma e_2 : \delta\tau \wedge$
$\Delta; \Gamma \vdash e_1 : \tau \wedge \Delta; \Gamma \vdash e_2 : \tau$.

Theorem:
$\Delta; \Gamma \vdash e_1 \equiv$ctx $e_2 : \tau \Longrightarrow$
$\Delta; \Gamma \vdash e_1 \equiv$u $e_2 : \tau$.

Proof:
(For Pitts, who bakes substitutivity into the definition of congruence, this is trivial. We didn't.)

Let $\delta{:}\Delta$ and $\gamma : \delta\Gamma$ and $K \div \delta\tau$.
TS: $K[\delta\gamma e_1] \downarrow\downarrow K[\delta\gamma e_2]$.

We want to reduce or expand $\delta\gamma$e_i so we end up with something of the form K'[e_i] on either side.
(Reductions and expansions work for proving cotermination.)

Suppose $\Delta = \alpha_1,\cdots,$α_n and $\Gamma = x_1{:}\tau_1, \cdots,$ x_m:τ_m.
$K[\delta\gamma e_1] \mathrel{{}^*\!\!\leftarrow} K[(\Lambda\alpha_1. \cdots. \Lambda$α_n.$\lambda x_1. \cdots. \lambda$x_m. $e_1)[\delta\alpha_1]\cdots[\delta$α_n]$(\gamma x_1)\cdots(\gamma$x_m)]
$K[\delta\gamma e_2] \mathrel{{}^*\!\!\leftarrow} K[(\Lambda\alpha_1. \cdots. \Lambda$α_n.$\lambda x_1. \cdots. \lambda$x_m. $e_2)[\delta\alpha_1]\cdots[\delta$α_n]$(\gamma x_1)\cdots(\gamma$x_m)].

TS: $K[\delta\gamma e_1] \downarrow\downarrow K[\delta\gamma e_2]$

STS: $K[(\Lambda\alpha_1. \cdots. \Lambda\alpha\_n.\lambda x_1. \cdots. \lambda x\_m.\ e_1)[\delta\alpha_1]\cdots[\delta\alpha\_n](\gamma x_1)\cdots(\gamma x\_m)] \downarrow\downarrow$
$K[(\Lambda\alpha_1. \cdots. \Lambda\alpha\_n.\lambda x_1. \cdots. \lambda x\_m.\ e_2)[\delta\alpha_1]\cdots[\delta\alpha\_n](\gamma x_1)\cdots(\gamma x\_m)]$.

This follows from ≡ctx: Applications of the compatibility lemmas.

(These ⋯'s can be replaced by an inductive definition.)
Q.E.D.

Theorem:
$\Delta; \Gamma \vdash e_1 \equiv u\ e_2 : \tau \Longrightarrow \Delta; \Gamma \vdash e_1 \approx e_2 : \tau$.
Proof:
By FTLR, $\Delta; \Gamma \vdash e_1 \approx e_1 : \tau$.
By CIU transitivity, $\Delta; \Gamma \vdash e_1 \approx e_2 : \tau$.
Q.E.D.

Theorem (CIU-equivalence-respecting property aka "CIU-transitivity"):
If $\Delta; \Gamma \vdash e_1 \approx e_2 : \tau$
and $\Delta; \Gamma \vdash e_2 \equiv u\ e_3 : \tau$
then $\Delta; \Gamma \vdash e_1 \approx e_3 : \tau$.
Proof:
Let $\rho \in D[\Delta]$ and $(\gamma_1,\gamma_2) \in G[\Gamma]\rho$.
TS: $(\rho_1\gamma_1 e_1, \rho_2\gamma_2 e_3) \in E[\tau]\rho$.
By first assumption,
WK: $(\rho_1\gamma_1 e_1,\rho_2\gamma_2 e_2) \in E[\tau]\rho$.
Instantiating with ρ_2 and γ_2 and the second assumption,
WK: $\rho_2\gamma_2 e_2 \equiv u\ \rho_2\gamma_2 e_3 : \rho_2\tau$.
By CIU-transitivity for closed terms,
we're done.
Q.E.D.

Theorem (CIU-transitivity for closed terms):
If $(e_1,e_2) \in E[\tau]\rho$
and $\vdash e_2 \equiv u\ e_3 : \rho_2\tau$
then $(e_1,e_3) \in E[\tau]\rho$.

Note that we need the syntactic typing assumptions in ρ to even state this theorem.

Proof:
Let $(K_1,K_2) \in K[\tau]\rho$.
TS: $K_1[e_1] \downarrow\downarrow K_2[e_3]$.

By first assumption,
WK: $K_1[e_1] \downarrow\downarrow K_2[e_2]$.

By second assumption,
WK: $K_2[e_2] \downarrow\downarrow K_2[e_3]$
$\Leftarrow$ $K_2 \div \rho_2 \tau$.
But that's a side condition baked into ρ.

By transitivity of $\downarrow\downarrow$,
we're done.
Q.E.D.

Aside: The proofs we just gave assume the language has type and term abstractions. We didn't even look at the defintion of $V[-]\rho$. Most languages have such abstractions. The moment you do ⊤⊤-closure, you get completeness for free. (But completeness doesn't help. Somehow it doesn't make it easier to prove anything.)

— Other uses of CIU-equivalence

Aside: With five minutes thought, it does not seem to be the case that a direct proof $\equiv_{ctx} \Longrightarrow \approx$ is impossible. We may not need $\equiv_u$ for completeness of the LR. Derek seems to recall there is some reason to go with $\equiv_u$; perhaps related to a more complicated model.

$\equiv_u$ serves other purposes: The CIU-theorem: $\equiv_u \subseteq \equiv_{ctx}$.

From the CIU-theorem we get simple syntactic reduction/expansion properties for free. (Honsell-Mason-Smith-Talcott, 1995).

Sketchy Corollary of the CIU-theorem (See Pitts, 7.5.8)[Conversions]:
- $\Delta; \Gamma \vdash (\text{fix } f(x).e)v \equiv_{ctx} e[\text{fix } f(x).e/f][v/x] : \tau$

- let $x_1 = e_1$ in (let $x_2 = e_2$ in e) $\equiv_{ctx}$
  let x_2 = (let $x_1 = e_1$ in e_2) in e : τ
  $\Leftarrow$ $x_1 \notin fv(e)$.

— Pitts' extensionality theorem

Now we're ready to prove some theorems that seem to need the typed model.

Let's prove (part of) Pitts' extensionality theorem. Derek thinks its sort of misnamed. Several properties are called extensionality.

Theorem (Pitts, 7.7.1)[Extensionality for values]:

Part 2:

(Suppose $f, f' \in$ CVal.)

$\vdash f \equiv f' : \sigma \longrightarrow \tau$ iff $\forall v{:}\sigma.\ \vdash f\ v \equiv f'\ v : \tau$.

Proof:

($\Rightarrow$)

Observe, by equivalence of $\equiv$ctx and $\approx$,

$\vdash f \equiv f' : \sigma \longrightarrow \tau$

iff

$\forall v \equiv v' : \sigma.\ f\ v \equiv f'\ v' : \tau$ (Intermediate characterization.)

$\Rightarrow$ $\forall v{:}\sigma.\ \vdash f\ v \equiv f'\ v : \tau$.

($\Leftarrow$)

Suppose $v \equiv v' : \sigma$ and $\forall v''{:}\sigma.\ f\ v'' \equiv f'\ v'' : \tau$

TS: $\vdash f\ v \equiv f'\ v : \tau$.

Instantiating,

WK: $\vdash f\ v \equiv f'\ v : \tau$.

$\Rightarrow$ ($\equiv$ a congruence)

$\vdash f\ v \equiv f'\ v' : \tau$.

Q.E.D.

— Comparing our approach to Pitts' approach

How does Pitts do things?

He does not split between a value relation and a term relation. Everything is one big relation. So here's a little guide to reading Pitts.

When he wants to talk about the values that come from his LR, he restricts to values.

Roughly, for example

(†) $E[\sigma \longrightarrow \tau]\rho := E[\sigma]\rho \longrightarrow E[\tau]\rho$

where $\longrightarrow$ is the relational action of the arrow type constructor. Pitts proves several theorems (that are at first glance hard to parse) that boil down to "his approach and our approach coincide".

He shows (Lemma 7.6.13) roughly:

(7.20) $E[\sigma]\rho \longrightarrow E[\tau]\rho = V[\sigma]\rho \longrightarrow E[\tau]\rho$

What Pitts writes, literally:

$\text{fun}((r_1)^{vst}, (r_2)^{st}) = \text{fun}(r_1, (r_2)^{st})$.

Pitts' more abstract approach makes it easier to prove certain closure properties. Our approach seems somehow more direct.

This kind of theorem is necessary for Pitts because he defines the LR like †, but he wants to use it like (7.20).

Note the E-only relations make a lot of sense when you begin with a call-by-name setting. You certainly want function arguments to be terms, not values. (To be fair, that's where Pitts started.)

He shows roughly:
(7.19) $(v_1,v_2) \in V[\sigma\longrightarrow\tau]\rho \Longleftrightarrow (v_1,v_2) \in E[\sigma\longrightarrow\tau]\rho$

What Pitts writes:
$\text{fun}(r_1, (r_2)^{st}) \Longleftrightarrow \text{fun}(r_1, (r_2)^{st})^{stv}$

This came up when we proved Admissibility. We used ⊤⊤-closure to prove admissibility. The question was "does admissibility hold for E-related recursive functions in the value relation"? The answer is no.

At the end of the day, you can prove a similar thing (⊤⊤-closure adds nothing) at every type except type variables.

We could change the model so that only ⊤⊤-closed relations are candidates, fixing type variables.

However, we have a fundamental problem with existentials. If you know two existentials are contextually equivalent, you cannot conclude much. (They may be equivalent for some "really disturbing reasons".)

The following result holds (probably) for both the untyped and typed models. We'll drop the typing side conditions.

Lemma (Half of coincidence at function type):
$(v_1,v_2) \in E[\sigma\longrightarrow\tau]\rho \Rightarrow (v_1,v_2) \in V[\sigma\longrightarrow\tau]\rho$.
Proof:
Suppose $(v_1,v_2) \in E[\sigma\longrightarrow\tau]\rho$.
TS: $(v_1,v_2) \in V[\sigma\longrightarrow\tau]\rho$.

Let $(v'_1,v'_2) \in V[\sigma]\rho$.
TS: $(v_1\, v'_1, v_2\, v'_2) \in E[\tau]\rho$.

Let $(K_1,K_2) \in K[\tau]\rho$.
TS: $K_1[v_1\ v'_1] \downarrow\downarrow K_2[v_2\ v'_2]$.

Set K'_i := $K_1[\bullet\ v'\_i]$.
STS: $(K'_1,K'_2) \in K[\sigma\to\tau]\rho$.

Let $(v''_1,v''_2) \in V[\sigma\to\tau]\rho$.
TS: $K'_1[v''_1] \downarrow\downarrow K'_2[v''_2]$
$\Leftrightarrow$ $K_1[v''_1\ v'_1] \downarrow\downarrow K_2[v''_2\ v'_2]$.

WK: $(v''_1,v''_2) \in V[\sigma\to\tau]\rho\ \wedge$
$(v'_1,v'_2) \in V[\sigma]\rho$
$\Rightarrow$ $(v''_1\ v'_1,\ v''_2\ v'_2) \in E[\tau]\rho$.

WK: $(K_1,K_2) \in K[\tau]\rho \wedge (v''_1\ v'_1,\ v''_2\ v'_2) \in E[\tau]\rho$
$\Rightarrow$ We're done.
Q.E.D.

— HW: Due next Tuesday.

Prove the coincidence lemmas for product and sum types. (They aren't completely trivial and you get some practice with $\top\top$-closure proofs.)

Suppose
i. $\forall v_1,v_2.\ (v_1,v_2) \in E[\sigma]\rho \Rightarrow (v_1,v_2) \in V[\sigma]\rho$
ii. $\forall v_1,v_2.\ (v_1,v_2) \in E[\tau]\rho \Rightarrow (v_1,v_2) \in V[\tau]\rho$

Prove:
1. $\forall v_1,v_2.\ (v_1,v_2) \in E[\sigma\times\tau]\rho \Rightarrow (v_1,v_2) \in V[\sigma\times\tau]\rho$
2. $\forall v_1,v_2.\ (v_1,v_2) \in E[\sigma+\tau]\rho \Rightarrow (v_1,v_2) \in V[\sigma+\tau]\rho$

Recall:
$V[\sigma\times\tau]\rho := \{\ ((v_1,v'_1),\ (v_2,v'_2))\ |\ (v_1,v_2) \in V[\sigma]\rho \wedge (v'_1,v'_2) \in V[\tau]\rho\ \}$
$V[\sigma+\tau]\rho := \{\ (\mathrm{inj}_1\ v_1,\ \mathrm{inj}_1\ v_2)\ |\ (v_1,v_2) \in V[\sigma]\rho\ \}$
$\quad \cup \{\ (\mathrm{inj}_2\ v_1,\ \mathrm{inj}_2\ v_2)\ |\ (v_1,v_2) \in V[\tau]\rho\ \}$

— Counterexamples to coincidence at exisential types.

It's nice to have counterexamples.

Pitts' language includes a real void type. Our language has a value $\Lambda\alpha.\bot : \forall\alpha.\alpha$ (because our intro form is $\Lambda\alpha.e$). For reasons that are unexplained (except for the following example going through), Pitts applies the ML value restriction. His intro form is $\Lambda\alpha.v$ rather than

$\Lambda\alpha.e$.

We could get the effect of a value restriction by adding a void type satisfying

$$V[\mathrm{void}]\rho := \emptyset.$$

Pitts' argument for why this is a counterexample probably depends a bit on his use of a typed model. We will probably be able to work around that.

The counterexample: We want two packaged values that are contextually equivalent. So in the (typed) term relation, they're logically related. But there will be no relation between their internal data representations such that the operations preserve it. In other words, the representation independence property will break down. In other words, they won't be in the value relation.

```
τ := ∃α.(α⟶bool)⟶bool
v₁ := pack [void,λf.⊥] as τ
v₂ := pack [bool,λf.
        if (f true) then
            if (f false) then ⊥ else true
        else ⊥] as τ
```

Speaking ML:

```
v₂ := pack [bool,λf.
        ((f true) = true andalso
        (f false) = false) orelse ⊥] as τ.
```

Intuition: The context cannot pass in the identity function. The guy on the right diverges unless it receives the identity function. So the two are contextually equivalent.

But v_1 and v_2 cannot be in the value relation. We'd have to have a relation between types void and bool. The only candidate relation (since void is empty) is the empty relation. If we plug in the empty relation, then we have to show that the functions

† $(\lambda f.\bot,$
$\lambda f.\mathrm{if}\ (f\ \mathrm{true})\ \mathrm{then}\ (\mathrm{if}\ (f\ \mathrm{false})\ \mathrm{then}\ \bot\ \mathrm{else\ true})\ \mathrm{else}\ \bot) \in$
$V[(\alpha\rightarrow\mathrm{bool})\rightarrow\mathrm{bool}]\alpha\mapsto\emptyset$.

We'll suppose † to arrive at a contradiction:

$$(\lambda x.\bot,\ \lambda x.x) \in V[\alpha\rightarrow\mathrm{bool}]\alpha\mapsto\emptyset.$$

Vacuously, any two functions of the right type are related at $V[\alpha \rightarrow \text{bool}]\alpha \mapsto \emptyset$. But applying the functions in †, you get (diverge, true) so the terms † are not related.

There's another counterexample (a real brain teaser) that doesn't depend on the void type. Consider:

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

The context has no constructors for type α. It can only produce the identity and divergent functions as arguments f. In either one, we call f. If it's divergent, they both diverge. If it's the identity, they both return true.

Yet, amazingly,
$\neg(v_1 \equiv v_2 : \tau)$. (Attribution: Summi)

Challenge: Find a distinguishing context.

The point of these example is to show the subtlety of contextual equivalence. This is the first example Derek came up with (based on the original example in Pitts 2005).

The point: There are only four possible interpretations of α:
$(\emptyset, \{ ((),\text{true}) \}, \{ ((),\text{false}) \}, \{ ((),\text{true}), ((),\text{false}) \} \}$
and none of them work.

Aside: Pitts' example is a pretty clear proof of why transitivity does not hold for the untyped logical relation. We can prove that v_1 and v_2 are contextually related by transitively chaining together some logical relations proofs (that's a useful way to prove contextual equivalence).