Modified: 20121203

— Bind lemma

We'll start with the bind lemma from last time. We'll see how it simplifies these compatibility proofs.

Here's a statement of the bind lemma at the level of the open logical relation (rather than at the level of the model). This matters in the compatibility proofs where we want to reason with open expressions.

Idea: The bind lemmas is a way of showing that the seemingly complicated structure of these compatibility proofs (eg, application) can be abstracted away. We can avoid a lot of reasoning directly in the model.

Lemma (Bind):
If $\Delta; \Gamma \vdash e_1 \approx e_2 : \sigma$
and $\Delta; \Gamma, x{:}\sigma \vdash K_1[x] \approx K_2[x] : \tau$
and $x \notin fv(K_1,K_2)$,
then $\Delta; \Gamma \vdash K_1[e_1] \approx K_2[e_2] : \tau$.

Derek wrote this as an inference rule because it's prettier:
$\Delta; \Gamma \vdash e_1 \approx e_2 : \sigma$
$\Delta; \Gamma, x{:}\sigma \vdash K_1[x] \approx K_2[x] : \tau$
$x \notin fv(K_1,K_2)$
—
$\Delta; \Gamma \vdash K_1[e_1] \approx K_2[e_2] : \tau$

Idea: e_1 and e_2 represent the term in the evaluation position. Bind a variable in the second premise that represents the result of evaluating e_1 and e_2. Enforce the fact that these are in the evaluation position by putting the variable in contexts.

Proof:
Let ··· intros ···.

TS: $(\delta_1\gamma_1(K_1[e_1]),\delta_2\gamma_2(K_2[e_2])) = (\delta_1\gamma_1K_1[\delta_1\gamma_1e_1],\delta_2\gamma_2K_2[\delta_2\gamma_2e_2]) \in E[\tau]\rho$.

Unfolding $E[\tau]\rho$,
let $(K'_1,K'_2) \in K[\tau]\rho$.
TS: $K'_1[\delta_1\gamma_1K_1[e_1])] \downarrow\downarrow K'_2[\delta_2\gamma_2K_2[e_2])]$.

From our first premise,

WK: $(\delta_1\gamma_1 e_1, \delta_2\gamma_2 e_2) \in E[\sigma]\rho$
$\Rightarrow$ $(K'_1[\delta_1\gamma_1 K_1])[\delta_1\gamma_1 e_1] \downarrow\downarrow (K'_2[\delta_2\gamma_2 K_2])[\delta_2\gamma_2 e_2]$.

Question: Does the implicit lemmas about evaluation contexts apply to arbitrary contexts (that capture variables)?
Answer: It's probably fine (if you're careful about hygenic side-conditions).

STS: $(K'_1[\delta_1\gamma_1 K_1], K'_2[\delta_2\gamma_2 K_2]) \in K[\sigma]\rho$.
Let $(v_1, v_2) \in V[\sigma]\rho$.
TS: $(K'_1[\delta_1\gamma_1 K_1])[v_1] \downarrow\downarrow (K'_2[\delta_2\gamma_2 K_2])[v_2]$.

Idea: We're now in good shape to extend our substitutions and apply the second premise.

Set $\gamma'\_i := \gamma\_i, x \mapsto v\_i$.
WK: $(\gamma'_1, \gamma'_2) \in G[\Gamma, x{:}\sigma]\rho$.

By the second premise and the fact $x \notin fv(K_1, K_2)$,
WK: $((\delta_1\gamma_1 K_1)[v_1], (\delta_2\gamma_2 K_2)[v_2]) = (\delta_1\gamma'_1(K_1[x]), \delta_2\gamma'_2(K_2[x])) \in E[\tau]\rho$.

By relatedness of K'_1, K'_2,
WK: $K'_1[\delta_1\gamma_1 K_1[v_1]] \downarrow\downarrow K'_2[\delta_2\gamma_2 K_2[v_2]]$
$\Leftrightarrow$ (Rewriting)
$(K'_1[\delta_1\gamma_1 K_1])[v_1] \downarrow\downarrow (K'_2[\delta_2\gamma_2 K_2])[v_2]$.
Q.E.D.

Having shown this bind lemma, we can work with it rather than fuss with the back-and-forth reasoning between the continuation and term relations when doing these compatibility proofs. (The bind lemma may prove useful in proofs using the model, as well.)

— Applying the bind lemma

Let's reprove compatibility for application.
We'll apply bind with $K\_i := (\bullet\ e'\_i)$.

(Aside from Dave: Derek presented this by building a derivation incrementally. That's likely easier to follow than what I write here.)

Our goal:
$WK_1 :: \Delta; \Gamma \vdash e_1 \approx e_2 : \sigma \to \tau$
$WK_2 :: \Delta; \Gamma \vdash e'_1 \approx e'_2 : \sigma$
—

$\Delta; \Gamma \vdash e_1\ e'_1 \approx e_2\ e'_2 : \tau$

Our derivation requires weakening and a lemma:

$WK_2 :: \Delta; \Gamma \vdash e'_1 \approx e'_2 : \sigma$
A := — (Weakening)
$\Delta; \Gamma,x{:}\sigma \vdash e'_1 \equiv e'_2 : \sigma$

— (Lemma)
B := $\Delta; \Gamma,x{:}\sigma\to\tau,y{:}\sigma \vdash x\ y \approx x\ y : \tau$

A
B
C := —
$\Delta; \Gamma,x{:}\sigma \vdash x\ e'_1 \approx x\ e'_2 : \tau$

$WK_1 :: \Delta; \Gamma \vdash e_1 \approx e_2 : \sigma \to \tau$
D := C
— (Bind)
$\Delta; \Gamma \vdash e_1\ e'_1 \approx e_2\ e'_2 : \tau$

We needed weakening. The proof of weakening immediately reduces to a property of closing substitutions. (Weakening is always a trivial property when working with these “semantic” judgements.)

Lemma:
$\Delta; \Gamma,x{:}\sigma\to\tau,y{:}\sigma \vdash x\ y \approx x\ y : \tau$.
Proof:
This is easy, but not trivial. The Δ and Γ are irrelevant.
Quantify over substitutions for x, y.
Let $\rho \in D[\Delta]$
Let $(f_1,f_2) \in V[\sigma\to\tau]\rho$
Let $(v_1,v_2) \in V[\sigma]\rho$
TS: $(f_1\ v_1,f_2\ v_2) \in E[\tau]\rho$.
This follows by unfolding $V[\sigma\to\tau]\rho$.
Q.E.D.

If you look back at the proofs of the FTLR, there was some redundancy. We repeatedly quantified over arbitrary values, plugged them in, and went forward. In those proofs, the bind lemma might have made the structure clearer.

— Untyped vs Pitts-style typed models

If you add the extra assumptions Pitts employs, you get an extra property: Completeness of the logical relation.

Pitts adds typing assumptions to the model. (We'll define that in a second.) As a result, he gets completness of the logical relation: $\approx = \equiv_{ctx}$.

With our untyped model, we have only soundness: $\approx \subseteq \equiv_{ctx}$.

We'll find examples that are contextually equivalent but that cannot be proven so using our untyped model.

Question: Why work with the unpyted model?
Answer:

> The short answer is that the untyped model works just fine but is somewhat simpler. You don't have all these typing side conditions when you use it. Moreover, the typed model does not scale to reasoning about languages that are not as "well-formed". Example: Relating high- and low-level languages. On the other hand, it doesn't make sense to talk about contexutal equivalence in that setting (as the langages on the left and right differ). In Derek's experience, completeness is somewhat overrated. Many applications don't need it.

Today we'll see the delta between our untyped model and Pitts-style typed models and a use of the typed model to prove the following (not terribly exciting) theorem:

> Theorem (Pitts' 7.7.1(2) "extensionality for values"):
> $\vdash F \equiv F' : \sigma \to \tau$
> $\iff \forall v.\ (\vdash v{:}\sigma \implies \vdash F\ v \equiv F'\ v : \tau)$.

Pitts uses the term "extensionality" rather loosely. He calls this an extensionality theorem. Derek kind of objects to the terminology in general. (It's fine here.) Eg, part 5 of the same theorem is just the reperesentation independence principle we talked about a couple weeks ago. It's not what most people think about when they think about extensionality.

Proof Idea for "extensionality for values":

$\Rightarrow$ Easy using our model.

$\Leftarrow$ We could show F, F' equivalent by first showing that related value arguments go to related results.

Problem: We only have one value, not two related values. It looks like a potentially weaker assumption than what we need in the model.

Problem: We only get contextual equivalence, not logical equivalence from that assumption. If we had completeness, we could make progress using the logical relations.

The proof in Pitts uses something called the closed instantiation of uses theorem (CIU).

Aside: It's probably not so important that Δ is empty in this theorem. Pitts claims we can generalize these theorems to open terms. Derek is pretty sure we could also prove something like:

$\alpha \vdash e_1 \equiv e_2 : \tau$
$\iff \forall \sigma$ closed. $e_1[\sigma/\alpha] \equiv e_2[\sigma/\alpha]$.

(Derek hasn't actually worked it out.)

The point: We'll be able to prove this kind of “extensionality principle”.

How exciting is this? Well, if your goal in life is to prove contextual equivalences, then maybe. But it's not a major loss in Derek's experience.

Semantics folks tend to have a love-hate relationship with contextual equivalence. It's a very clear criterion, easily defined for many languages. Everyone gets it. It makes sense. It's very general. It's transitive: You can transitively compose your contextual equivalence proofs. OTOH, contextual equivalence is very “fiddley”. It's defined in terms of the syntax of the language. Thus there are equivalences that hold for completely obscure reasons. (We'll see a brain teaser later to make this point.)

In some sense, the reason Pitts' model gives you completeness is somewhat technical/artificial. Aside from these extensionality principles, we don't actually get an effective way of proving more

contextual equivalences than we can prove in the untyped model. We don't get proof techniques, we just get theorems.

All that said, let us look at a Pitts-style typed model and this idea of CIU-equivalence.

— Adding typing assumptions to our model

Where might we add these things?

We want to build our LR from syntactically well-typed terms.

Where before we (implicitly) had sets ERel, KRel, VRel satisfying

$\mathcal{E}[\tau]\rho \in \mathrm{ERel}$, $\mathcal{K}[\tau]\rho \in \mathrm{KRel}$, and $\mathcal{V}[\tau]\rho \in \mathrm{VRel}$

where ERel, KRel, and VRel contained relations on closed terms, continuations, and values. Now these will contain relations over *typed* and closed terms, continuations, and values.

To define KRel, we have to spell out typing for continuations,

Judgement:
$\Delta; \Gamma \vdash K \div \sigma \iff \exists \tau.\ \Delta; \Gamma \vdash K : \sigma \rightsquigarrow \tau$.

"K div σ": K expects something of type σ in its hole.

More general judgement, in case we need to compose evaluation contexts:

Judgement: $\Delta; \Gamma \vdash K : \sigma \rightsquigarrow \tau$
(Hole type σ, program type τ.)

—
$\Delta; \Gamma \vdash \bullet : \tau \rightsquigarrow \tau$

$\Delta; \Gamma \vdash K : \sigma \rightsquigarrow (\tau_1 \to \tau_2)$
$\Delta; \Gamma \vdash e : \tau_1$
—
$\Delta; \Gamma \vdash K\ e : \sigma \rightsquigarrow \tau_2$

$\Delta; \Gamma \vdash v : \tau_1 \longrightarrow \tau_2$
$\Delta; \Gamma \vdash K : \sigma \rightsquigarrow \tau_1$
—
$\Delta; \Gamma \vdash v\ K : \sigma \rightsquigarrow \tau_2$

Etcetera

Note how all of these rules treat the hole parametrically. There's always some σ.

Interesting:
If you treat contexts as syntactic objects, then the rules for these K's fall out as a special case of the rules for well-typed contexts C.
These rules are simpler because you don't have to worry about variable hygiene. (Arbitrary contexts are messier than evaluation contexts.)

If you want to do these things properly, you'll have to prove simple syntactic properties.

Example:
If $\Delta; \Gamma \vdash K : \sigma \rightsquigarrow \tau$
and $\Delta; \Gamma \vdash e : \sigma$
then $\Delta; \Gamma \vdash K[e] : \tau$.
Proof: By induction on the first derivation.

Now we can define our $\mathrm{ERel}(\tau_1,\tau_2)$, $\mathrm{KRel}(\tau_1,\tau_2)$, and $\mathrm{VRel}(\tau_1,\tau_2)$.
Aside: If it weren't for ρ, τ_1 and τ_2 would be the same.

CCont = ··· obvious thing analogous to CTyp, etc ···

$\mathrm{KRel}(\tau_1,\tau_2) = \{ R \subseteq \mathrm{CCont} \times \mathrm{CCont} \mid$
$\quad \forall (K_1,K_2) \in R.$
$\quad\quad \vdash K_1 \div \tau_1 \wedge \vdash K_2 \div \tau_2 \}$
$\mathrm{ERel}(\tau_1,\tau_2) = \{ \cdots \mid \cdots \vdash e_1 : \tau_1 \wedge \vdash e_2 : \tau_2 \}$

There are two more (boring) preliminaries to deal with.

Aside: Avoding all of this boring noise is one of the reasons we started with the untyped models. You don't need all of this stuff for our initial application, representation independence.

Before, we had:

(VRel = Cand)

$D[\Delta] = \{\ \rho \in \Delta \longrightarrow \text{VRel}\ \}$

Now, we have:

$D[\Delta] := \{\ \rho \in \Delta \longrightarrow \text{CType} \times \text{CType} \times \text{VRel}\ |$

$\forall \alpha \in \Delta.\ \rho(\alpha) = (\tau_1,\tau_2,R) \Longrightarrow R \in \text{VRel}(\tau_1,\tau_2)\ \}$

Minus the hand-waving:

$\rho \in \Delta \longrightarrow \Sigma\tau_1{:}\text{Ctype}.\ \Sigma\tau_2{:}\text{CType}.\ \text{VRel}(\tau_1,\tau_2).$

Notation:

We write

$\rho_1(\alpha) = \pi_1(\rho(\alpha))$

$\rho_2(\alpha) = \pi_2(\rho(\alpha))$

$\text{hat}\{\rho\}(a) = \pi_3(\rho\alpha).$

We implicitly lift these to arbitrary syntactic classes.

Theorem:

If $\rho \in D[\Delta]$

and $\text{fv}(\tau) \subseteq \text{dom}(\rho)$,

then $V[\tau]\rho \in \text{VRel}(\rho_1\tau,\rho_2\tau)$.

We call this a theorem, but we bake it into the model by adding side conditions. Two examples:

$V[\sigma\longrightarrow\tau]\rho := \{\ (v_1,v_2)\ |$

$\vdash v_1 : \rho_1(\sigma\longrightarrow\tau) \wedge \vdash \rho_2(\sigma\longrightarrow\tau) \wedge \cdots\ \}$

$K[\tau]\rho := \{\ (K_1,K_2)\ |$

$\vdash K_1 \div \rho_1\tau \wedge \vdash K_2 \div \rho_2\tau \wedge \cdots\ \}$

Now let's define $G[\Gamma]\rho$:

$G[\Gamma]\rho := \{\ (\gamma_1,\gamma_2)\ |\ \forall (x{:}\tau)\in\Gamma.\ (\gamma_1 x,\gamma_2 x) \in V[\tau]\rho\ \}$

Trivial fact we'll need later:

If $(\gamma_1,\gamma_2) \in G[\Gamma]\rho$,

then $\vdash \gamma_1 : \rho_1\Gamma$

and $\vdash \gamma_2 : \rho_2\Gamma$.

where $\vdash \gamma : \Gamma :\Longleftrightarrow \forall (x{:}\tau)\in\Gamma.\ \vdash \gamma x : \tau.$

Once slight nice thing: These ρ_1 and ρ_2 will take the place of δ_1 and

δ_2 in our proofs. (Asdie: Now these syntactic types matter a little.)

Now let's state the definition of the open logical relation.

$$\begin{aligned}
&\Delta; \Gamma \vdash e_1 \approx e_2 : \tau :\Longleftrightarrow \\
&\quad \Delta; \Gamma \vdash e_1 : \tau \wedge \\
&\quad \Delta; \Gamma \vdash e_2 : \tau \wedge \\
&\quad \forall \rho \in D[\Delta].\ \forall (\gamma_1,\gamma_2) \in G[\Gamma]\rho. \\
&\qquad (\rho_1\gamma_1 e_1,\ \rho_2\gamma_2 e_2) \in E[\tau]\rho.
\end{aligned}$$

We need a few other small changes to the LR:

$$\begin{aligned}
&V[\forall\alpha.\tau]\rho := \{\ (v_1,v_2) \mid \forall \sigma_1,\sigma_2 \in CTyp.\ \forall R \in VRel(\sigma_1,\sigma_2). \\
&\quad \vdash v_1 : \rho_1(\forall\alpha.\tau) \wedge \\
&\quad \vdash v_2 : \rho_2(\forall\alpha.\tau) \wedge \\
&\quad (v_1\ \sigma_1,\ v_2\ \sigma_2) \in E[\tau]\rho,\alpha\mapsto(\sigma_1,\sigma_2,R)\ \}
\end{aligned}$$

$$\begin{aligned}
&V[\exists\alpha.\tau]\rho := \{\ (\text{pack}[\sigma_1,v_1] \text{ as } \exists\alpha.\tau,\ \text{pack}[\sigma_2,v_2] \text{ as } \exists\alpha.\tau) \mid \\
&\quad \vdash \text{pack}[\sigma_1,v_1] \text{ as } \exists\alpha.\tau : \rho_1(\exists\alpha.\tau) \wedge \\
&\quad \vdash \text{pack}[\sigma_2,v_2] \text{ as } \exists\alpha.\tau : \rho_2(\exists\alpha.\tau) \wedge \\
&\quad \exists R \in VRel(\sigma_1,\sigma_2).\ (v_1,v_2) \in V[\tau]\rho,\alpha\mapsto(\sigma_1,\sigma_2,R)\ \}
\end{aligned}$$

The point: We have to be more careful with the syntactic types.

— CIU equivalence

Compatibility and adequacy give us one direction:
(†) $\Delta; \Gamma \vdash e_1 \approx e_2 : \tau \Longrightarrow \Delta; \Gamma \vdash e_1 \equiv\text{ctx}\ e_2 : \tau.$

The issue: If we want to go $\Longleftarrow$, we need some intermediate steps. Derek has not seen a direct proof of that direction. What Pitts does is use an intermediate equivalence relation, ciu-equivalence. It's like $\equiv$ where (roughly) you restrict to evaluation contexts rather than arbitrary contexts.

We'll denote ciu-equivalence by $\equiv$u. Pitts shows

1. $\Delta; \Gamma \vdash e_1 \equiv\text{ctx}\ e_2 : \tau \Longrightarrow \Delta; \Gamma \vdash e_1 \equiv\text{u}\ e_2 : \tau.$

   Proving this is straightforward since $\equiv$u is roughly a more restricted form of $\equiv$.

2. $\Delta; \Gamma \vdash e_1 \equiv\text{u}\ e_2 : \tau \Longrightarrow \Delta; \Gamma \vdash e_1 \approx e_2 : \tau.$

(1) and (2) together with (†) establishes the equivalence of all three relations.

En passant, Pitts establishes the CIU Theorem: $\equiv_{ctx} = \equiv_u$. Originally this theorem was due to Mason and Talcott in the early '90s. (It predated this work on logical equivalence.) Mason and Talcott gave a proof principle that did not require reasoning about arbitrary contexts. What's nice about ciu-equivalence: It applies in many languages.

The acronym is slightly obscure: CIU = closed instantiations of uses. (You can't make this stuff up.)

As Harper likes to point out, it should be UCI as it's really "uses of closed instantiations". (But they're the same.)

We often use $\equiv_{ciu}$ on closed terms. Then it's just "uses" hence our notation $\equiv_u$. "Uses" means the equivalence is preserved under arbitrary evaluation contexts K (rather than arbitrary syntactic contexts C). You can imagine, intuitively, given how we've set up our model, that biorthogonality somehow bakes in closure under evaluation contexts.

The "closed instantations" part simply means that you quantify over a closing γ up front.

With all these intuitions in place, let's define it.

Definition (CIU-equivalence for closed terms, aka "U-equivalence"):

$\vdash e_1 \equiv_u e_2 : \tau$ if
- $\vdash e_1 : \tau \wedge$
- $\vdash e_2 : \tau \wedge$
- $\forall K.\ (\vdash K \div \tau \Longrightarrow K[e_1] \downarrow\downarrow K[e_2])$.

Note for closed terms it really is similar to $\equiv_c$, but we've replaced $C[-]$ by $K[-]$. For open terms, we close up with δ's and γ's.

Definition (CIU-equivalence, aka "UCI-equivalence"):

$\Delta; \Gamma \vdash e_1 \equiv_u e_2 : \tau$ if
- $\Delta; \Gamma \vdash e_1 : \tau \wedge$
- $\Delta; \Gamma \vdash e_2 : \tau \wedge$
- $\forall \delta, \gamma.$
  - If $\vdash \delta : \Delta \longrightarrow Ctyp$
  - and $\vdash \gamma : \delta\Gamma$,
  - then $\vdash \delta\gamma e_1 \equiv_u \delta\gamma e_2 : \delta\tau$.

In some sense, this cleanly separates the two things that contexts can do to terms:
1. close up over free variables of e_1 and e_2 (covered by δ,γ) and
2. do something with these terms (covered by K).

— Next time

Next time, we'll jump into the proof $\dagger \Longrightarrow 1 \Longrightarrow 2 \Longrightarrow \dagger$.

Incidentally, we may end up covering

> Derek's paper on Plotkin-Abadi logic.
> It gives a much more abstract presentation of a lot of this stuff.
>
> Logical step-indexed logical relations.
> LICS '11. (The journal version.)

— Instructive exercise

After class, Derek answered a question by suggesting the following exercise.

Work out progress and preservation for an abstract machine à la Pitts and Felleisen/Hieb.
If needed, lookup of Felleisen/Hieb's paper (something like syntactic theory of control operators and state).

Define $(K,e) \mapsto (K',e')$.
(We'll need to spell out the full definition of the $\rightsquigarrow$ typing judgement.)

Define $\vdash (K,e)$ ok if $\exists \tau. \vdash e : \tau \wedge \vdash K \div \tau$.

NB: At each reduction, τ might change.

Define $\vdash (K,e)$ terminal if $\vdash e$ is a value $\wedge\ K = \bullet$.

Prove progress:

> If $\vdash (K,e)$ ok,
> then either
> 1. $(K,e) \mapsto (K',e')$ or
> 2. $\vdash (K,e)$ terminal.

Prove preservation:

If $\vdash (K,e)$ ok

and $(K,e) \mapsto (K',e')$,

then $\vdash (K',e')$ ok.