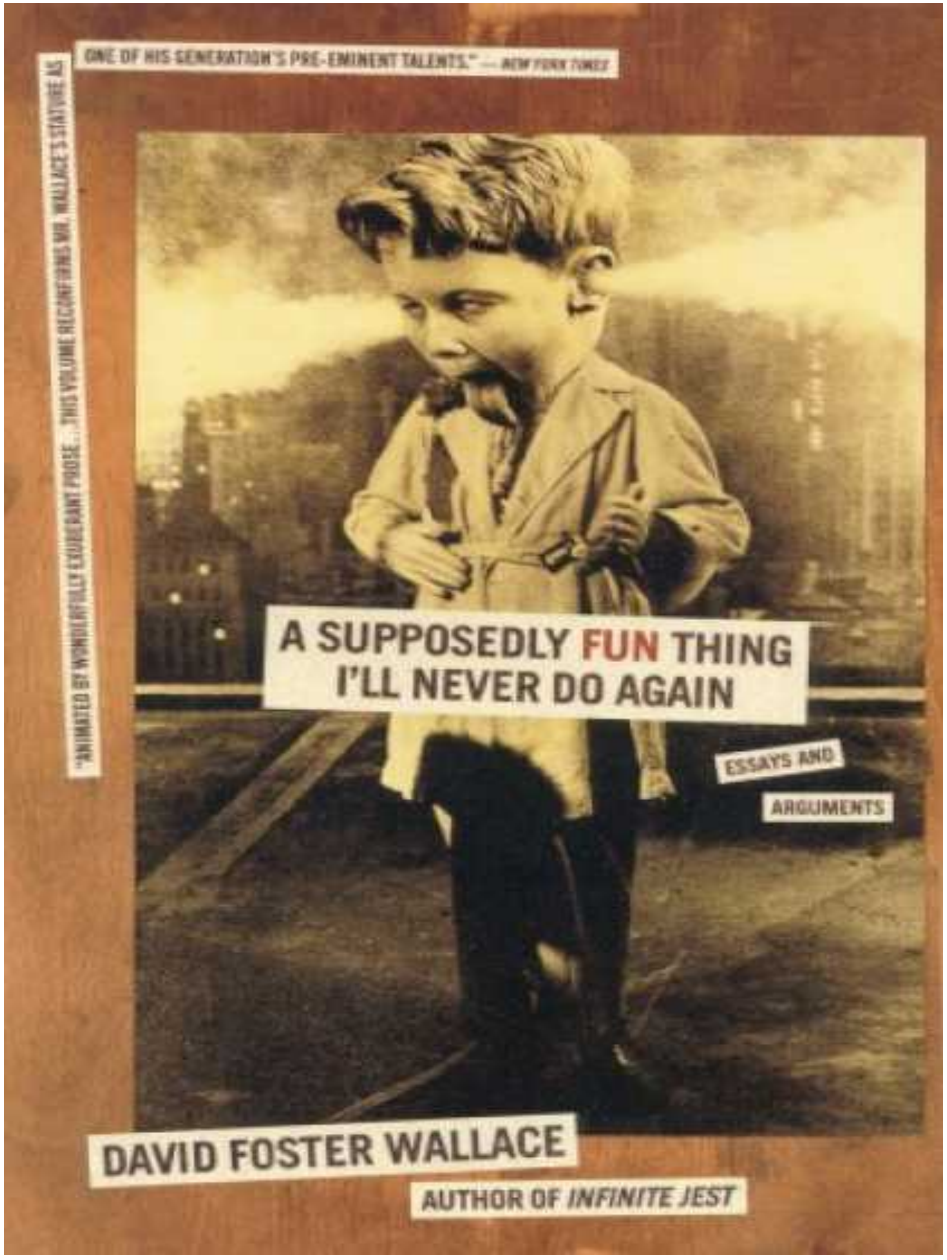
A Supposedly Fun Thing I May Have to Do Again
A HOAS encoding of Howe's method

Alberto Momigliano
DI, Milano

July 9, 2012

DFW


ONE OF HIS GENERATION'S PRE-EMINENT TALENTS." — NEW YORK TIMES
"ANIMATED BY WONDERFULLY EXUBERANT PROSE . . . THIS VOLUME RECONFIRMS MR. WALLACE'S STATURE AS
A SUPPOSEDLY FUN THING I'LL NEVER DO AGAIN
ESSAYS AND ARGUMENTS
DAVID FOSTER WALLACE
AUTHOR OF INFINITE JEST

Intro

- Howe's method is a remarkably flexible *syntactic* technique to show the congruence properties of functional programs equalities in the presence of higher-order functions.
- I'm not interested in showing pieces of code equivalent (let alone with a proof checker), but in the method itself, as it's a hard benchmark for HOAS systems.
- As of today, only Abella can do it, although the way we have to go about is harder than it should and looks too much like nominal logic – and this gotta be bad, right? – meaning it's mostly done in the reasoning logic.
- Using Abella with predicate quantification (thanks Kaustuv) makes life better.
- With HH^{ω} could be even better. Or a bit of Tac, if we're stuck in the reasoning logic.

Background

- Program equivalence as contextual equivalence is intuitive, but hard to reason about.
- Applicative (bi)similarity: m and n are bisimilar if whenever m evaluates to a value, so does n, and all the "subprograms" of the resulting values also bisimilar, and vice versa.
- To break the circularity we take the greatest fixed point induced by that.

Background

- Program equivalence as contextual equivalence is intuitive, but hard to reason about.
- Applicative (bi)similarity: m and n are bisimilar if whenever m evaluates to a value, so does n, and all the "subprograms" of the resulting values also bisimilar, and vice versa.
- To break the circularity we take the greatest fixed point induced by that.

Definition (Applicative simulation for the λ-calc with lazy lists)

- $m \preccurlyeq_{\sigma \supset \sigma'} n$ iff whenever $m \Downarrow \lambda x.\, p$ for any p, there exists a q such that $n \Downarrow \lambda y.\, q$ and for every $r{:}\sigma$, $p[r/x] \preccurlyeq_{\sigma'} q[r/y]$;
- $m \preccurlyeq_{[\sigma]} n$ iff $m \Downarrow \mathit{nil}$ entails $n \Downarrow \mathit{nil}$ or, if $m \Downarrow (x :: xs)$ for all x, xs there are y, ys such that $n \Downarrow (y :: ys)$ for which $x \preccurlyeq_{\sigma} y$ and $xs \preccurlyeq_{[\sigma]} ys$.

Background II

- A (typed) pre-congruence is a transitive relation $\mathcal{R}_\sigma$ that is *compatible*, that is it respects the way λ-terms are constructed (we drop lists for brevity):

  (C1) $\Gamma, x{:}\sigma \vdash x \ \mathcal{R}_\sigma \ x$;

  (C2) $\Gamma, x{:}\sigma \vdash m \ \mathcal{R}_{\sigma'} \ n$ entails $\Gamma \vdash \lambda x.\ m \ \mathcal{R}_{\sigma \supset \sigma'} \ \lambda x.\ n$;

  (C3) $\Gamma \vdash m_1 \ \mathcal{R}_{\sigma \supset \sigma'} \ n_1$ and $\Gamma \vdash m_2 \ \mathcal{R}_\sigma \ n_2$ entails $\Gamma \vdash (m_1 \ m_2) \ \mathcal{R}_) \ (n_1 \ n_2 \sigma'$;

- C1–C3 entail reflexivity and weakening.

Background II

- A (typed) pre-congruence is a transitive relation $\mathcal{R}_\sigma$ that is *compatible*, that is it respects the way λ-terms are constructed (we drop lists for brevity):

  (C1) $\Gamma, x{:}\sigma \vdash x\ \mathcal{R}_\sigma\ x$;
  (C2) $\Gamma, x{:}\sigma \vdash m\ \mathcal{R}_{\sigma'}\ n$ entails $\Gamma \vdash \lambda x.\ m\ \mathcal{R}_{\sigma\supset\sigma'}\ \lambda x.\ n$;
  (C3) $\Gamma \vdash m_1\ \mathcal{R}_{\sigma\supset\sigma'}\ n_1$ and $\Gamma \vdash m_2\ \mathcal{R}_\sigma\ n_2$ entails $\Gamma \vdash (m_1\ m_2)\ \mathcal{R}_)\ (n_1\ n_2\sigma'$;

- C1–C3 entail reflexivity and weakening.
- A relation is *substitutive* (Sub) if $\Gamma, y{:}\sigma \vdash m\ \mathcal{R}_{\sigma'}\ m'$ and $\Gamma \vdash n\ \mathcal{R}_\sigma\ n'$ entails $\Gamma \vdash m[n/y]\ \mathcal{R}_{\sigma'}\ m'[n'/y]$.
- If $\mathcal{R}_\sigma$ is substitutive *and* reflexive, then it is also *closed under substitution* (Cus): $\Gamma, y{:}\sigma \vdash m\ \mathcal{R}_{\sigma'}\ m'$ and $\cdot \vdash n : \sigma$ entails $\Gamma \vdash m[n/y]\ \mathcal{R}_{\sigma'}\ m'[n/y]$.

What do we want?

- To show that similarity is a pre-congruence, but
- We have defined it for programs only, while a pre-congruence concerns open terms. *Extend* similarity to *open terms* via instantiation. For $\Gamma = x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$,

$$\Gamma \vdash m \preccurlyeq^{\circ}_{\sigma} m' \text{ iff for all i and closed } p_i{:}\sigma_i, m[p_i/x_i] \preccurlyeq_{\sigma} m'[p_i/x_i]$$

What do we want?

- To show that similarity is a pre-congruence, but
- We have defined it for programs only, while a pre-congruence concerns open terms. *Extend* similarity to *open terms* via instantiation. For $\Gamma = x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$,

$$\Gamma \vdash m \preccurlyeq^{\circ}_{\sigma} m' \text{ iff for all i and closed } p_i{:}\sigma_i, m[p_i/x_i] \preccurlyeq_{\sigma} m'[p_i/x_i]$$

- Since similarity is a pre-order, so is open similarity
- By construction (Cus) holds, but to show C3 (cong for app) you need (Sub) and this is not obvious.
- Howe's method: introduce a *candidate* relation $\Gamma \vdash m \preccurlyeq^{\mathcal{H}}_{\sigma} m'$, which contains (open) similarity and is almost immediately a pre-congruence;
- Then show that it does coincide with (open) similarity.

The candidate relation

$$\frac{\Gamma \vdash \langle\rangle \preccurlyeq^{\circ}_{\top} n}{\Gamma \vdash \langle\rangle \preccurlyeq^{\mathcal{H}}_{\top} n}\ ep \qquad \frac{\Gamma, x{:}\sigma \vdash x \preccurlyeq^{\circ}_{\sigma} n}{\Gamma, x{:}\sigma \vdash x \preccurlyeq^{\mathcal{H}}_{\sigma} n}\ var$$

The candidate relation

$$\frac{\Gamma \vdash \langle\rangle \preccurlyeq^{\circ}_{\top} n}{\Gamma \vdash \langle\rangle \preccurlyeq^{\mathcal{H}}_{\top} n}\ ep \qquad \frac{\Gamma, x{:}\sigma \vdash x \preccurlyeq^{\circ}_{\sigma} n}{\Gamma, x{:}\sigma \vdash x \preccurlyeq^{\mathcal{H}}_{\sigma} n}\ var$$

$$\frac{\Gamma, x{:}\sigma \vdash m \preccurlyeq^{\mathcal{H}}_{\sigma'} m' \qquad \Gamma \vdash \lambda x.\ m' \preccurlyeq^{\circ}_{\sigma\supset\sigma'} n}{\Gamma \vdash \lambda x.\ m \preccurlyeq^{\mathcal{H}}_{\sigma\supset\sigma'} n}\ fun$$

The candidate relation

$$\frac{\Gamma \vdash \langle\rangle \preccurlyeq^{\circ}_{\top} n}{\Gamma \vdash \langle\rangle \preccurlyeq^{\mathcal{H}}_{\top} n}\ ep \qquad \frac{\Gamma, x{:}\sigma \vdash x \preccurlyeq^{\circ}_{\sigma} n}{\Gamma, x{:}\sigma \vdash x \preccurlyeq^{\mathcal{H}}_{\sigma} n}\ var$$

$$\frac{\Gamma, x{:}\sigma \vdash m \preccurlyeq^{\mathcal{H}}_{\sigma'} m' \qquad \Gamma \vdash \lambda x.\ m' \preccurlyeq^{\circ}_{\sigma \supset \sigma'} n}{\Gamma \vdash \lambda x.\ m \preccurlyeq^{\mathcal{H}}_{\sigma \supset \sigma'} n}\ fun$$

$$\frac{\Gamma \vdash m_1 \preccurlyeq^{\mathcal{H}}_{\sigma \supset \sigma'} m_1' \quad \Gamma \vdash m_2 \preccurlyeq^{\mathcal{H}}_{\sigma} m_2' \qquad \Gamma \vdash m_1'\ m_2' \preccurlyeq^{\circ}_{\sigma'} n}{\Gamma \vdash m_1\ m_2 \preccurlyeq^{\mathcal{H}}_{\sigma'} n}\ app$$

Proof outline

1. $(\asymp^{\mathcal{H}} \circ \asymp^{\circ}) \subset \asymp^{\mathcal{H}}$ By case analysis using transitivity of open similarity.

Proof outline

1. $(\precsim^{\mathcal{H}} \circ \precsim^{\circ}) \subset \precsim^{\mathcal{H}}$ By case analysis using transitivity of open similarity.
2. Howe is reflexive. Induction on typing, using reflexivity of open similarity.

Proof outline

1. $(\asymp^{\mathcal{H}} \circ \asymp^{\circ}) \subset \asymp^{\mathcal{H}}$ By case analysis using transitivity of open similarity.
2. Howe is reflexive. Induction on typing, using reflexivity of open similarity.
3. $\asymp^{\circ} \subset \asymp^{\mathcal{H}}$; from (1) and (2).

Proof outline

1. $(\asymp^{\mathcal{H}} \circ \asymp^{\circ}) \subset \asymp^{\mathcal{H}}$ By case analysis using transitivity of open similarity.
2. Howe is reflexive. Induction on typing, using reflexivity of open similarity.
3. $\asymp^{\circ} \subset \asymp^{\mathcal{H}}$; from (1) and (2).
4. $\asymp^{\mathcal{H}}$ is substitutive. By induction on the first premise.

Proof outline

1. $(\precsim^{\mathcal{H}} \circ \precsim^{\circ}) \subset \precsim^{\mathcal{H}}$ By case analysis using transitivity of open similarity.
2. Howe is reflexive. Induction on typing, using reflexivity of open similarity.
3. $\precsim^{\circ} \subset \precsim^{\mathcal{H}}$; from (1) and (2).
4. $\precsim^{\mathcal{H}}$ is substitutive. By induction on the first premise.
5. The Howe relation "mimics" the simulation conditions: If $\lambda x.\ m \precsim^{\mathcal{H}}_{\sigma \cup \sigma'} n$, then $n \Downarrow \lambda x.\ m'$ and for every $q{:}\sigma$ we have $m[q/x] \precsim^{\mathcal{H}}_{\sigma'} m'[q/x]$.

Proof outline

1. $(\precsim^{\mathcal{H}} \circ \precsim^{\circ}) \subset \precsim^{\mathcal{H}}$ By case analysis using transitivity of open similarity.
2. Howe is reflexive. Induction on typing, using reflexivity of open similarity.
3. $\precsim^{\circ} \subset \precsim^{\mathcal{H}}$; from (1) and (2).
4. $\precsim^{\mathcal{H}}$ is substitutive. By induction on the first premise.
5. The Howe relation "mimics" the simulation conditions: If $\lambda x.\ m \precsim^{\mathcal{H}}_{\sigma \cup \sigma'} n$, then $n \Downarrow \lambda x.\ m'$ and for every $q{:}\sigma$ we have $m[q/x] \precsim^{\mathcal{H}}_{\sigma'} m'[q/x]$.
6. ("downward closure") If $p \precsim^{\mathcal{H}}_{\sigma} q$ and $p \Downarrow v$, then $v \precsim^{\mathcal{H}}_{\sigma} q$. Induction on evaluation, and inversion on Howe and simulation, with an additional case analysis on v.

Proof outline

1. $(\precsim^{\mathcal{H}} \circ \precsim^{\circ}) \subset \precsim^{\mathcal{H}}$ By case analysis using transitivity of open similarity.
2. Howe is reflexive. Induction on typing, using reflexivity of open similarity.
3. $\precsim^{\circ} \subset \precsim^{\mathcal{H}}$; from (1) and (2).
4. $\precsim^{\mathcal{H}}$ is substitutive. By induction on the first premise.
5. The Howe relation "mimics" the simulation conditions: If $\lambda x.\ m \precsim^{\mathcal{H}}_{\sigma \cup \sigma'} n$, then $n \Downarrow \lambda x.\ m'$ and for every $q{:}\sigma$ we have $m[q/x] \precsim^{\mathcal{H}}_{\sigma'} m'[q/x]$.
6. ("downward closure") If $p \precsim^{\mathcal{H}}_{\sigma} q$ and $p \Downarrow v$, then $v \precsim^{\mathcal{H}}_{\sigma} q$. Induction on evaluation, and inversion on Howe and simulation, with an additional case analysis on v.
7. $p \precsim^{\mathcal{H}}_{\sigma} q \Longrightarrow p \precsim_{\sigma} q$. By coinduction, using the obvious invariant, point (5) and (6).

Proof outline

1. $(\precsim^{\mathcal{H}} \circ \precsim^{\circ}) \subset \precsim^{\mathcal{H}}$ By case analysis using transitivity of open similarity.
2. Howe is reflexive. Induction on typing, using reflexivity of open similarity.
3. $\precsim^{\circ} \subset \precsim^{\mathcal{H}}$; from (1) and (2).
4. $\precsim^{\mathcal{H}}$ is substitutive. By induction on the first premise.
5. The Howe relation "mimics" the simulation conditions: If $\lambda x.\ m \precsim^{\mathcal{H}}_{\sigma \cup \sigma'} n$, then $n \Downarrow \lambda x.\ m'$ and for every $q{:}\sigma$ we have $m[q/x] \precsim^{\mathcal{H}}_{\sigma'} m'[q/x]$.
6. ("downward closure") If $p \precsim^{\mathcal{H}}_{\sigma} q$ and $p \Downarrow v$, then $v \precsim^{\mathcal{H}}_{\sigma} q$. Induction on evaluation, and inversion on Howe and simulation, with an additional case analysis on v.
7. $p \precsim^{\mathcal{H}}_{\sigma} q \Longrightarrow p \precsim_{\sigma} q$. By coinduction, using the obvious invariant, point (5) and (6).

Theorem

$\Gamma \vdash p \precsim^{\mathcal{H}}_{\sigma} q$ *iff* $\Gamma \vdash p \precsim^{\circ}_{\sigma} q$

Encoding

- We know how to do evaluation/typing, etc. Simulation is slightly more interesting:

```
CoDefine sim : tm -> tm -> ty -> prop by
 sim M1 M2 (arr S S') :=
     {of M1 (arr S S')} /\ {of M2 (arr S S')} /\
     (forall M, {eval M1 (abs M)} ->
        exists M' , {eval M2 (abs M')} /\
          (forall x, {of x S} -> sim (M x) (M' x) S')); ...
```

- How do we encode Howe?

$$\frac{\Gamma, x{:}\sigma \vdash m \preccurlyeq^{\mathcal{H}}_{\sigma'} m' \qquad \Gamma \vdash \lambda x.\ m' \preccurlyeq^{\circ}_{\sigma \supset \sigma'} n}{\Gamma \vdash \lambda x.\ m \preccurlyeq^{\mathcal{H}}_{\sigma \supset \sigma'} n}\ \mathit{fun}$$

Encoding

- We know how to do evaluation/typing, etc. Simulation is slightly more interesting:

```
CoDefine sim : tm -> tm -> ty -> prop by
 sim M1 M2 (arr S S') :=
     {of M1 (arr S S')} /\ {of M2 (arr S S')} /\
     (forall M, {eval M1 (abs M)} ->
        exists M' , {eval M2 (abs M')} /\
          (forall x, {of x S} -> sim (M x) (M' x) S')); ...
```

- How do we encode Howe?

$$\frac{\Gamma, x{:}\sigma \vdash m \preccurlyeq^{\mathcal{H}}_{\sigma'} m' \qquad \Gamma \vdash \lambda x.\ m' \preccurlyeq^{\circ}_{\sigma\supset\sigma'} n}{\Gamma \vdash \lambda x.\ m \preccurlyeq^{\mathcal{H}}_{\sigma\supset\sigma'} n}\ \mathit{fun}$$

- At the SL?

```
howe (abs M) N (arr S S') :-
  (pi x\ (pi Q\ howe x Q S :- sim x Q S) =>
      sigma M'\ howe (M x) (M' x) S', sim (abs M') N (arr S S')).
```

Encoding Howe

- Well, no. This is third order and with bad nesting of quantifiers, but more importantly, `sim` is coinductive and what calls a ML relation, must stay in ML (kinda like Vegas)
- OK, so Howe at the ML, but how? On programs only?

```
Define howe : tm -> tm -> tp -> prop by
howe (abs M) N (arr S S') :=
  (forall x\ (forall Q\ sim x Q S -> howe x Q S) ->
      exists M'\ howe (M x) (M' x) S' /\ sim (abs M') N (arr S S'));...
```

- Forget about it. It's not stratified (for lists), and we do not get very far induction-wise.

Encoding Howe

- Well, no. This is third order and with bad nesting of quantifiers, but more importantly, `sim` is coinductive and what calls a ML relation, must stay in ML (kinda like Vegas)
- OK, so Howe at the ML, but how? On programs only?

```
Define howe : tm -> tm -> tp -> prop by
howe (abs M) N (arr S S') :=
  (forall x\ (forall Q\ sim x Q S -> howe x Q S) ->
      exists M'\ howe (M x) (M' x) S' /\ sim (abs M') N (arr S S'));...
```

- Forget about it. It's not stratified (for lists), and we do not get very far induction-wise.
- Bite the bullet and use explicit (ML) contexts:
  `Define howe : olist -> tm -> tm -> ty -> prop.`
  But then, I have to extend similarity to open terms as well.

Encoding: open simulation and Howe

```
Define howe : olist -> tm -> tm -> ty -> prop by
  howe L ep N top              := osim L ep N top;
  howe L (app F A) N S'        := exists F' A' S,
     howe L F F' (arr S S') /\ howe L A A' S /\ osim L (app F' A') N S';
  howe L (abs M) N (arr S S'):= exists M',nabla x,
     howe (of x S :: L) (M x) (M' x) S' /\ osim L (abs M') N (arr S S');
  howe L X M S                 := name X /\ member (of X S) L /\ osim L X M S.

Define osim : olist -> tm -> tm -> ty -> prop by
           osim nil M1 M2 S := {of M1 S} /\ {of M2 S} /\ sim M1 M2 S;
  nabla x, osim (of x S :: L) (M1 x) (M2 x) S':=
   {L |- of (abs M1) (arr S S')} /\ {L |- of (abs M2) (arr S S')} /\
       forall N, {of N S} -> osim L (M1 N) (M2 N) S'.
```

- How nominal is that??!!
- Note analogy with *arbitrary cascading substitutions*.

Main development

```
Theorem howe_of: ctx L -> howe L M1 M2 S -> {L |- of M1 S} /\ {L |- of
M2 S}.
Theorem howe_weaken: howe L1 M N S -> (forall X, member X L1 -> member X L2) ->
Theorem howe_trans: howe L P Q S -> osims L Q R S -> howe L P R S.
Theorem howe_refl: ctx L -> {L |- of M S} -> howe L M M S.
Theorem osim_howe: ctx L -> {L |- of E S} -> osims L E F S -> howe L E F S.
```

- So far so good. The substitution lemma (4) brings in instead an Abella-specific difficulty. It requires exchange, so it's not a structural induction. So we induct on the size of the term with a hole.

```
Theorem howe_subst: nabla x, mtm (A1 x) -> howe (of x S :: L) (A1 x) (A2 x) S'
                    {of B2 S} ->  howe L B1 B2 S -> howe L (A1 B1) (A2 B2) S'.
```

Main development

- Now it's easy sailing (well the next proof is hard)

```
Theorem down_closed: {eval P V} -> howe nil P Q S -> howe nil V Q S.
Theorem howe_sim: howe nil M N S -> sim M N S.
Theorem howe_osim: ctx L -> howe L M N S -> osim L M N S.
```

- It was simple to show that `howe` is compatible. Now apply the latter result to show that so is `(o)sim`

Lessons learned

- what did we (re)discover about the paper-and-pencil proof by Andy Pitts?
  - it heavily relies on the implicit machinery of typed relations
  - some minor, minor stuff:
    - `howe_trans` is not proved by induction but simply by inversion
    - some typos in the def of the Howe relation
- What about Abella?
  - Abella is a beautifully designed, robust, easy to use, yet very expressive system which provides features that no other HOAS system can at the moment match.
  - This is not to say that all is peachy.

SL vs. ML contexts

- SL contexts are your friends, but ML level ones are your foes.
- No support for ML-level contexts in ML means that ironically we, the HOAS people, are back to worry about weakening, exchange, strengthening (well, we have to worry about that in the SL as well, but were we to extend *subordination*...).
- Bad interaction with Abella's structural (co)induction restriction.
- *Lists* are a natural first choice but overly concrete. At the very least, I don't want to worry about exchange. So let's do bags, as we agreed.
- An aside: we may want to think how context *predicates* relate to each other, i.e. the issue of *context subsumption*.

Structural (co)induction

- How can you *not* love Abella's annotation-based approach to (co)induction? Especially if you have worked with explicit heights before (I did).
- However, we are not allowed to appeal to a lemma, even one that does not increase the height of the proof, viz. exchange, weakening, before applying the induction hypothesis.
- Need to find something else to induct on:
  - Sometimes not too bad (see `howe_subst`)
  - Sometimes (if you need complete induction) explicit heights will pollute the whole development
- Can we relax the annotation-based system to tolerate the application of non-height increasing lemmata? Maybe with a `trustme` annotation to begin with?

Weakness of the type system

- Maybe not a fair criticism, but is is painful to maintain type invariants such as

$$m_1 \preccurlyeq_\sigma \ m_2 \Longrightarrow m_i : \sigma$$

, which now need to be explicitly asserted (e.g. the def of simulation), while in *Twelf*...

- and simulating induction on *individuals* relationally is OK theoretically, but it has unpleasant backward and forward effects:
- For example case analysis on types needed for reflexivity of similarity is done via a `is_ty` judgments and this percolates back to the very definition of the static semantics of the OL language:

```
of (abs M) (arr S U) :- is_ty S, pi x (of x S => of (M x) U).
```

- it creates several (easy, but boring) proof obligations (around a dozen time in the proof of the main properties of the Howe).

Brittleness of proof scripts, a.k.a. the spaghetti script problem

- Not just an Abella's problem (see the Isabelle vs. Isar controversy), but more compelling here due to the lack of automation.
- Adding/removing a hypothesis in a `Theorem` breaks the whole thing – you `apply` to the wrong/non-existent hypotheses:
  - backchain/apply with unknown help
  - Kaustuv's user naming should also help, but I haven't tried as not in the forallp branch.
- Slippery slope towards a full tactical language.
- Still I could use some simple things such as preferring/postponing goals, maybe defining "macros" for patterns of commands...
- Not to mention something like Tac's `prove`.

$\forall p$Abella

- Howe's account is generic: construct $\mathcal{R}^H$ from a putative equality relation $\mathcal{R}$:

$$\frac{\Gamma, x{:}\sigma \vdash m\ \mathcal{R}^{\mathcal{H}}_{\sigma'}\ m' \qquad \Gamma \vdash \lambda x.\ m'\ \mathcal{R}_{\sigma\supset\sigma'}\ n}{\Gamma \vdash \lambda x.\ m\ \mathcal{R}^{\mathcal{H}}_{\sigma\supset\sigma'}\ n}\ \mathit{fun}$$

- Lemma `osim_howe` an instance of the Lemma for which if $\mathcal{R}$ is a preorder, then $\mathcal{R} \subset \mathcal{R}^{\mathcal{H}}$
- HOA would lead to HOAS-style relational reasoning-

Speculations: Reflection

- Are we really stuck with the "one way only" SL–ML relationships?
- Maybe we should really try to be able to encode `howe` at the SL level

Further speculations

- Using delay/force: we could work entirely in the SL. On the other hand, working with *thunks* is never problem-free.
- Using an infinitary SL (see David's cyclic proof, Carsten's infinitary sequent calculus).

Conclusions

- Howe's method is a good source of inspirations for HOAS systems