Revisiting the bookkeeping technique in HOAS-based encodings

Alberto Ciaffaglione Ivan Scagnetto

Università di Udine, Italia
Dipartimento di Matematica e Informatica
{alberto.ciaffaglione,ivan.scagnetto}@uniud.it

TYPES 2013 - 19th Conference "Types for Proofs and Programs"
Toulouse, France - April 22–26, 2013

Outline

Computer Aided Formal Reasoning and Type Theory

- Formal proofs about programming language metatheory and semantics are long and tedious:
  - their complexity is essentially due to the management of the details;
  - small mistakes or missed subtle cases cause to invalidate large amounts of work;
  - this effect worsens as languages scale.
- In particular, two recurring issues arise in type theory based LFs:
  - representing languages with binders without resorting to "cumbersome" encodings,
  - formally developing the metatheory of the encoded languages, in a "natural" way (*e.g.* close to the informal practice with pencil and paper).

The encoding issues in a LF

- Variables (α-conversion, capture-avoiding substitution)
  - traditional solutions (*e.g.* de Bruijn indices, first-order variables)
  - Higher-Order Abstract Syntax (HOAS) encapsulates the complexity, thus providing an high level of abstraction: representation by metavariables (functional constructors; functional application)
- Incompatibility between HOAS and inductive types
  - no "full" HOAS: $(T \to T) \to T$ violates the positivity constraint
  - lack of higher-order recursion and induction principles
  - no inductive representation: $(Var \to T) \to T$ generates parasite terms
  - difficulty to reason about concepts delegated to the metalanguage
- New logics (*e.g.* Nominal Logic, $FO\lambda^{\Delta\nabla}$)
- A more conservative approach
  - weak HOAS
  - the Theory of Contexts

The object language: System $F_{<:}$

- Why System $F_{<:}$?
  - Its syntax is rather simple (featuring a small number of constructors).
  - Nevertheless, it rises many common issues both in the encoding process and in the metatheory development (*e.g.*, variable binding, complex induction).
  - It is well known to proof assistant practitioners, since it was chosen as a test-bed for the POPLMark Challenge.
- We focus on the pure type language and on part 1a of the POPLMark Challenge.
- The work is carried out in the Coq proof assistant.

The (pure) type language

- Syntax of types:

$$
\begin{array}{llll}
\mathit{Type}: & S, T & ::= & X \qquad \text{type variable} \\
 & & & \mathit{Top} \qquad \text{maximal type} \\
 & & & S \to T \qquad \text{function type} \\
 & & & \forall X <: S.\, T \qquad \text{universal type}
\end{array}
$$

- Syntax of type environments:

$$
\begin{array}{llll}
\mathit{Env}: & \Gamma, \Gamma' & ::= & \emptyset \qquad \text{empty type environment} \\
 & & & \Gamma',\ X <: T \qquad \text{type variable binding} \\
 & & & \text{(with scoping discipline)}
\end{array}
$$

Algorithmic subtyping (for well-scoped types)

- Subtyping:

$$\frac{}{\Gamma \vdash S <: Top}\ (Top) \qquad \frac{}{\Gamma \vdash X <: X}\ (Refl)$$

$$\frac{X<:U \in \Gamma \quad \Gamma \vdash U <: T}{\Gamma \vdash X <: T}\ (Trans)$$

$$\frac{\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2}{\Gamma \vdash S_1 \to S_2 <: T_1 \to T_2}\ (Arr)$$

$$\frac{\Gamma \vdash T_1 <: S_1 \quad \Gamma, X<:T_1 \vdash S_2 <: T_2}{\Gamma \vdash \forall X<:S_1.S_2 <: \forall X<:T_1.T_2}\ (All)$$

- **Proposition 1 (Transitivity and Narrowing)**

  $\Gamma \vdash S <: Q \wedge \Gamma \vdash Q <: T \Rightarrow \Gamma \vdash S <: T$

  $\Gamma, X<:Q, \Delta \vdash M <: N \wedge \Gamma \vdash P <: Q \Rightarrow \Gamma, X<:P, \Delta \vdash M <: N$

Encoding: types and type environments (deep encoding)

- Variables as metavariables of a parametric, non-inductive type:

```
Parameter Var: Set.
```

- Types as terms of an inductive type:

```
Inductive Tp: Set := top: Tp | var: Var -> Tp
                   |  arr: Tp -> Tp -> Tp
                   |  fa : Tp -> (Var -> Tp) -> Tp.
Coercion var: Var >-> Tp.
```

  Example: $\forall X <: Top.X$ is encoded by `(fa top (fun X:Var => X))`

- Type environments as lists of pairs (deep encoding)

```
Definition envTp: Set := (list (Var * Tp)).
```

Encoding: subtyping

- The "(non) occurrence" concept (`isin` stands for $X \in fv(T)$):

```
Inductive isin (X:Var): Tp -> Prop :=
  isin_var: isin X X
| isin_arr: forall S T:Tp,
  isin X S \/ isin X T -> isin X (arr S T)
| isin_fa : forall S:Tp, forall U:Var->Tp,
  isin X S \/ (forall Y:Var, ~X=Y -> isin X (U Y)) ->
  isin X (fa S U).
```

- The auxiliary judgments: $X \notin dom(\Gamma)$ (`Gfresh`), $\langle X, T \rangle \in \Gamma$ (`isinG`), $closed(T, \Gamma)$ (`Gclosed`), $ok(\Gamma)$ (`okEnv`)
- Subtyping (`subTp`):

```
Inductive subTp: envTp -> Tp -> Tp -> Prop := ...
| sub_fa: forall G:envTp, forall S1 T1:Tp, forall S2 T2:Var->Tp,
  subTp G T1 S1 ->
  (forall X:Var, okEnv (cons (X,T1) G) ->
                 subTp (cons (X,T1) G) (S2 X) (T2 X)) ->
  subTp G (fa S1 S2) (fa T1 T2).
```

Formal development of the POPLmark Challenge

Main properties (*i.e.* part 1a of the POPLmark Challenge):

- ```
  Lemma reflexivity: forall T:Tp, forall G:envTp,
    okEnv G -> Gclosed T G -> subTp G T T.
  ```
- ```
  Theorem trans_narrow: forall Q:Tp,
    (forall S:Tp, forall G:envTp,
      (subTp G S Q) -> forall T:Tp, (subTp G Q T) -> (subTp G S T))
    /\
    (forall G':envTp, forall M N:Tp, (subTp G' M N) ->
      forall D G:envTp, forall X:Var, forall P:Tp,
      G'=(app D (cons (X,Q) G)) -> subTp G P Q ->
      subTp (app D (cons (X,P) G)) M N).
  ```

The Theory of Contexts

1. Decidability of equality over variables For any variables x and y, it is always possible to decide whether $x{=}y$ or $x{\neq}y$:

   ```
   Axiom LEM_Var: forall X Y:Var, X=Y \/ ~X=Y.
   ```

2. Freshness/Unsaturation For any term M, there exists a variable x which does not occur free in it:

   ```
   Axiom unsat: forall T:Tp, exists X:Var, notin X T.
   ```

3. Extensionality Two contexts are equal if they are equal on a fresh variable; *i.e.*, if $M(x){=}N(x)$ and $x{\notin}M(\cdot), N(\cdot)$, then $M(\cdot){=}N(\cdot)$:

   ```
   Axiom tp_ext: forall X:Var, forall S T:Var->Tp,
         (notin_ho X S) -> (notin_ho X T) -> (S X)=(T X) -> S=T.
   ```

4. β-expansion It is always possible to split a term into a context applied to a variable; *i.e.*, given a term M and a variable x, there exists a context $N(\cdot)$ such that $N(x){=}M$ and $x{\notin}N(\cdot)$:

   ```
   Axiom tp_exp: forall S:Tp, forall X:Var, exists S': Var->Tp,
         (notin_ho X S') /\ S=(S' X).
   ```

The Theory of Contexts at work

$\Rightarrow$ Reasoning by structural induction over contexts

Example: Monotonicity of "occurrence" $\in$ (`isin`):

$$x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

We recover the capability of "mimicking" the application of a higher-order induction principle by means of:

- a predicate $\mathit{measure}(T(z))$, which counts the number n of constructors occurring in $T(z)$ (where $z \notin T(\cdot)$)
- complete induction over the natural number n
- β-expansion, extensionality

The Theory of Contexts at work

$\Rightarrow$ Reasoning by structural induction over contexts

Example: Monotonicity of "occurrence" $\in$ (`isin`):

$$x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

We recover the capability of "mimicking" the application of a higher-order induction principle by means of:

- a predicate $\mathit{measure}(T(z))$, which counts the number n of constructors occurring in $T(z)$ (where $z \notin T(\cdot)$)
- complete induction over the natural number n
- β-expansion, extensionality

Lemma (preliminary):

$$z \notin T(\cdot) \wedge \mathit{measure}(T(z)) = n \wedge x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

The Theory of Contexts at work

$\Rightarrow$ Reasoning by structural induction over contexts

Example: Monotonicity of "occurrence" $\in$ (`isin`):

$$x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

We recover the capability of "mimicking" the application of a higher-order induction principle by means of:

- a predicate $measure(T(z))$, which counts the number n of constructors occurring in $T(z)$ (where $z \notin T(\cdot)$)
- complete induction over the natural number n
- β-expansion, extensionality

Lemma (preliminary):

$$z \notin T(\cdot) \wedge measure(T(z)) = n \wedge x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

Proof (complete induction on n, inversion of $measure(T(z)) = n$):

The Theory of Contexts at work

$\Rightarrow$ Reasoning by structural induction over contexts

Example: Monotonicity of "occurrence" $\in$ (`isin`):

$$x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

We recover the capability of "mimicking" the application of a higher-order induction principle by means of:

- a predicate $\mathit{measure}(T(z))$, which counts the number n of constructors occurring in $T(z)$ (where $z \notin T(\cdot)$)
- complete induction over the natural number n
- β-expansion, extensionality

Lemma (preliminary):

$$z \notin T(\cdot) \wedge \mathit{measure}(T(z)) = n \wedge x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

Proof (complete induction on n, inversion of $\mathit{measure}(T(z)) = n$):

1. β-expansion: $\exists T'(\cdot).\ T'(z) = T(z) \wedge z \notin T'(\cdot)$

The Theory of Contexts at work

$\Rightarrow$ Reasoning by structural induction over contexts

Example: Monotonicity of "occurrence" $\in$ (`isin`):

$$x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

We recover the capability of "mimicking" the application of a higher-order induction principle by means of:

- a predicate $\mathit{measure}(T(z))$, which counts the number n of constructors occurring in $T(z)$ (where $z \notin T(\cdot)$)
- complete induction over the natural number n
- β-expansion, extensionality

Lemma (preliminary):

$$z \notin T(\cdot) \wedge \mathit{measure}(T(z)) = n \wedge x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

Proof (complete induction on n, inversion of $\mathit{measure}(T(z)) = n$):

1. β-expansion: $\exists T'(\cdot).\ T'(z) = T(z) \wedge z \notin T'(\cdot)$
2. extensionality: $T'(\cdot) = T(\cdot)$

The Theory of Contexts at work

$\Rightarrow$ Reasoning by structural induction over contexts

Example: Monotonicity of "occurrence" $\in$ (`isin`):

$$x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

We recover the capability of "mimicking" the application of a higher-order induction principle by means of:

- a predicate $\mathit{measure}(T(z))$, which counts the number n of constructors occurring in $T(z)$ (where $z \notin T(\cdot)$)
- complete induction over the natural number n
- β-expansion, extensionality

Lemma (preliminary):

$$z \notin T(\cdot) \wedge \mathit{measure}(T(z)) = n \wedge x \in T(y) \wedge x \neq y \Rightarrow x \in T(\cdot)$$

Proof (complete induction on n, inversion of $\mathit{measure}(T(z)) = n$):

1. β-expansion: $\exists T'(\cdot).\ T'(z) = T(z) \wedge z \notin T'(\cdot)$
2. extensionality: $T'(\cdot) = T(\cdot)$

We can "lift" structural information about $T(\cdot)$ to the level of terms.

The Theory of Contexts at work: an example

The Theory of Contexts at work: an example

Let us consider the case, where $measure(T(z)) = n0 = S(n1 + n2)$.

The Theory of Contexts at work: an example

Let us consider the case, where $measure(T(z)) = n0 = S(n1 + n2)$.

- ...then, inverting such hypothesis, we get one subcase where $T(z) = (arr\ s0\ t0)$.

The Theory of Contexts at work: an example

Let us consider the case, where $measure(T(z)) = n0 = S(n1 + n2)$.

- ...then, inverting such hypothesis, we get one subcase where $T(z) = (arr\ s0\ t0)$.
- ...then, we apply β-expansion to $s0$, $t0$ and z, yielding two contexts $s0'(\cdot)$ and $t0'(\cdot)$ such that $s0'(z) \equiv s0$ and $t0'(z) \equiv t0$.

The Theory of Contexts at work: an example

Let us consider the case, where $measure(T(z)) = n0 = S(n1 + n2)$.

- ...then, inverting such hypothesis, we get one subcase where $T(z) = (arr\ s0\ t0)$.
- ...then, we apply β-expansion to $s0$, $t0$ and z, yielding two contexts $s0'(\cdot)$ and $t0'(\cdot)$ such that $s0'(z) \equiv s0$ and $t0'(z) \equiv t0$.
- ...in particular, we have $T(z) = (arr\ s0\ t0) = (arr\ s0'(z)\ t0'(z))$, whence we can infer $T(z) = (\lambda x : Tp.(arr\ s0'(x)\ t0'(x)))(z)$.

The Theory of Contexts at work: an example

Let us consider the case, where $measure(T(z)) = n0 = S(n1 + n2)$.

- ...then, inverting such hypothesis, we get one subcase where $T(z) = (arr\ s0\ t0)$.
- ...then, we apply β-expansion to $s0$, $t0$ and z, yielding two contexts $s0'(\cdot)$ and $t0'(\cdot)$ such that $s0'(z) \equiv s0$ and $t0'(z) \equiv t0$.
- ...in particular, we have $T(z) = (arr\ s0\ t0) = (arr\ s0'(z)\ t0'(z))$, whence we can infer $T(z) = (\lambda x : Tp.(arr\ s0'(x)\ t0'(x)))(z)$.
- Finally, by extensionality we infer $T(\cdot) = \lambda x : Tp.(arr\ s0'(x)\ t0'(x))$.

The Theory of Contexts at work: an example

Let us consider the case, where $measure(T(z)) = n0 = S(n1 + n2)$.

- ...then, inverting such hypothesis, we get one subcase where $T(z) = (arr\ s0\ t0)$.
- ...then, we apply β-expansion to $s0$, $t0$ and z, yielding two contexts $s0'(\cdot)$ and $t0'(\cdot)$ such that $s0'(z) \equiv s0$ and $t0'(z) \equiv t0$.
- ...in particular, we have $T(z) = (arr\ s0\ t0) = (arr\ s0'(z)\ t0'(z))$, whence we can infer $T(z) = (\lambda x : Tp.(arr\ s0'(x)\ t0'(x)))(z)$.
- Finally, by extensionality we infer $T(\cdot) = \lambda x : Tp.(arr\ s0'(x)\ t0'(x))$.

$\Rightarrow$ We have lifted structural information from first-order term to its higher-order counterpart.

The Theory of Contexts at work: an example

Let us consider the case, where $measure(T(z)) = n0 = S(n1 + n2)$.

- ...then, inverting such hypothesis, we get one subcase where $T(z) = (arr\ s0\ t0)$.
- ...then, we apply β-expansion to $s0$, $t0$ and z, yielding two contexts $s0'(\cdot)$ and $t0'(\cdot)$ such that $s0'(z) \equiv s0$ and $t0'(z) \equiv t0$.
- ...in particular, we have $T(z) = (arr\ s0\ t0) = (arr\ s0'(z)\ t0'(z))$, whence we can infer $T(z) = (\lambda x : Tp.(arr\ s0'(x)\ t0'(x)))(z)$.
- Finally, by extensionality we infer $T(\cdot) = \lambda x : Tp.(arr\ s0'(x)\ t0'(x))$.

$\Rightarrow$ We have lifted structural information from first-order term to its higher-order counterpart.
$\Rightarrow$ Now, we can use the `rewrite` tactic to take advantage of the inferred structural information about $T(\cdot)$.

ToC and Higher-Order Induction

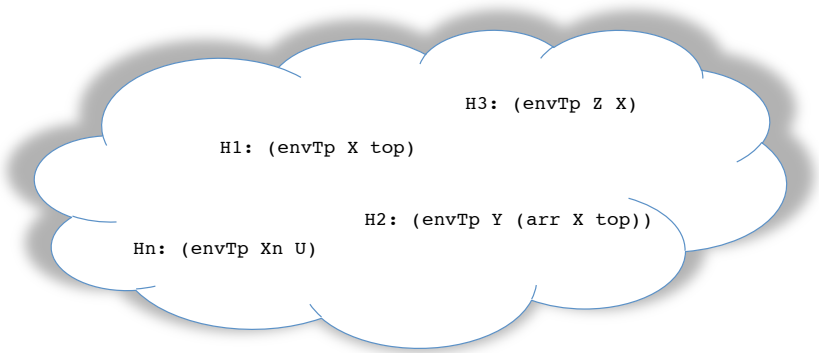
- Using the *measure technique*, it is possible to formally derive in Coq the following higher-order induction principle on terms of type Tp:

```
Lemma HO_TP_IND: forall P: (Var -> Tp) -> Prop,
(* top subcase *) (P (fun X:Var => top)) ->
(* var subcase 1 *) forall X:Var, (P (fun Y:Var => X)) ->
(* var subcase 2 *) (P var) ->
(* arr subcase *) (forall S T: Var -> Tp, (P S) -> (P T) ->
                   (P (fun X:Var => (arr (S X) (T X))))) ->
(* fa subcase *) (forall S:Var->Tp, forall T: Var -> Var -> Tp,
                  (P S) ->
                  (forall Y:Var, (P (fun X:Var=>(T X Y)))) ->
                  (P (fun X:Var => (fa (S X) (T X))))) ->
forall S:Var -> Tp, (P S).
```

Bookkeeping typing assumptions

- The *deep* representation of type environments forces the user to prove a large number of auxiliary lemmas about their management:
  - in our development 12.7 KB are devoted to this task (the proofs of Reflexivity, Transitivity and Narrowing are 16 KB long).
- Idea: type environments should be no more explicitly embedded into subtyping judgments as lists of pairs, but they should be rendered as global assumptions:
  - we follow the bookkeeping technique, using an open judgment à la LF in order to "record" typing assumptions about variables:

    ```
    Parameter envTp: Var -> Tp -> Prop.
    ```
  - Thus, typing assumptions are no longer arguments of the `subTp` predicate, but they are "globally" available:



Issues

- Being `envTp` an open predicate, we must enforce by some `Axioms` the usual good formation conditions about typing contexts.
- Representing informal statements like, *e.g.*, the narrowing property is no more straightforward:

$$\Gamma, X<:Q, \Delta \vdash M<:N \wedge \Gamma \vdash P<:Q \Rightarrow \Gamma, X<:P, \Delta \vdash M<:N.$$

  - the variable typings $X<:Q$ and $X<:P$ belong to two distinct environments, which are involved in distinct derivations;
  - a naive formalization would introduce two assumptions of type `(envTp X Q)` and `(envTp X P)`, in the same *global* environment, yielding a non well-formed environment.

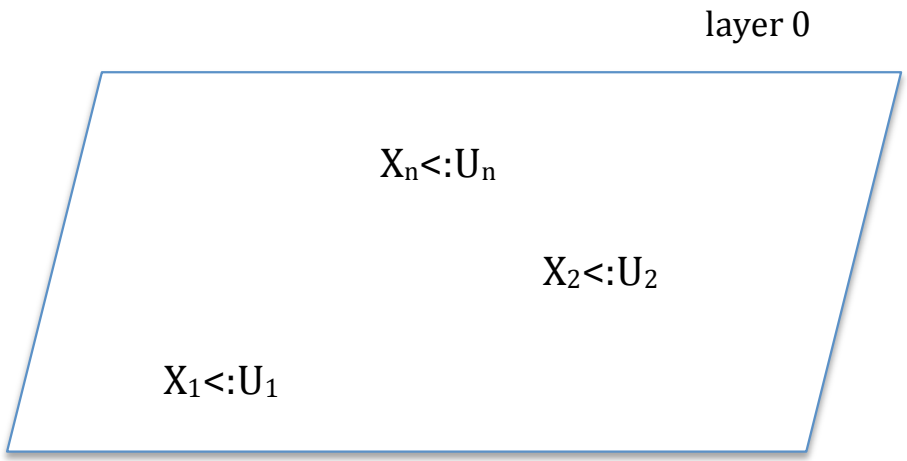
Solution 1: non-clashing renamings

To avoid the previous inconsistencies, we use distinct (non-clashing) variables:

```
Theorem trans_narrow: forall Q:Tp, ...
  /\
  forall X:Var, forall M N: Var->Tp, forall P:Tp,
  (notin_ho X M) -> (notin_ho X N) ->
  (envTp X Q) -> (subTp (M X) (N X)) ->
  (subTp P Q) -> forall Y:Var, ~X=Y ->
  (notin_ho Y M) -> (notin_ho Y N) ->
  (envTp Y P) -> (subTp (M Y) (N Y)).
```

This solution heavily relies upon the fact that all the involved predicates are *equivariant*.

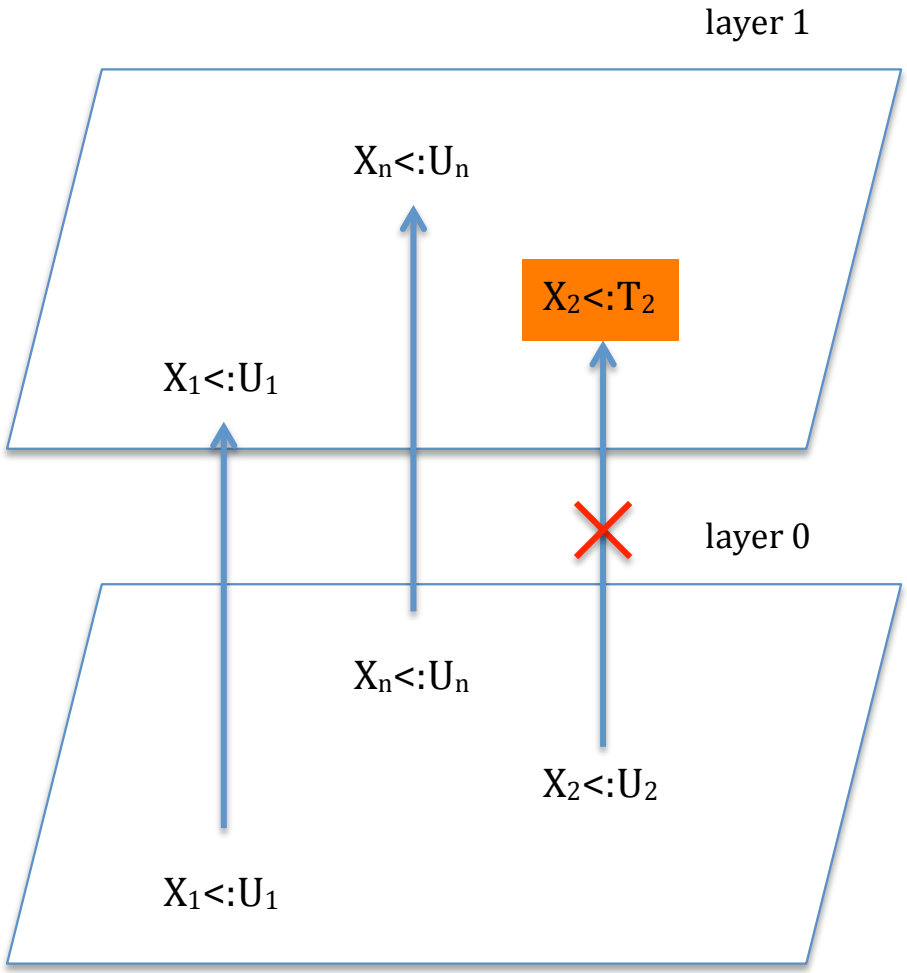
Solution 2: layering typing environments



Γ

➔ $\Gamma_0 \vdash S <:_0 T$

Solution 2: layering typing environments



➔ $\Gamma_1 \vdash S' <:_1 T'$

➔ $\Gamma_0 \vdash S <:_0 T$

Γ

Layering typing environments

- Our bookkeeping predicate becomes the following one:

  ```
  Parameter envTp: nat -> list(Var) -> Var -> Tp -> Prop.
  ```

  where:

  - the argument of type `nat` represents the *layer*, we are considering;
  - the argument of type `list(Var)` keeps track of the variables whose type has been redefined w.r.t. previous layers;
  - the remaining arguments represent the variable and its type.

- So doing, we are able to let the "untouched" declarations come up from the lower layers:

  ```
  forall n:nat, forall L L':list(Var), forall X:Var, forall U:Tp,
  envTp n L' X U -> (notinList X L) -> envTp (n+1) L X U.
  ```

Main proof - revisited

```
Theorem trans_narrow:  forall Q:Tp, ...
  /\
  forall n:nat, forall L L': list(Var),
  forall X:Var, forall M N P:Tp,
  (envTp n L' X Q) -> (notinList X L) -> (subTp n M N) ->
  (subTp (n+1) P Q) ->
  (envTp (n+1) (cons X L) X P) -> (subTp (n+1) M N).
```

Conclusions

- The technique of non-clashing renamings ultimately fails during the transitivity/narrowing proof because:
  - a renamed variable can occur into the type of another variable...
  - ...and substituting such occurrence with a fresh variable can lead to inconsistencies (all `envTp` assumptions are globally available):
    - let us assume `H1:(envTp X U)` and `H2:(envTp Y X)`,
    - if we need to narrow `X`, we can introduce a fresh `Z` such that `H3:(envTp Z U')` (where `(subTp U' U)`),
    - at this point we would also like to have `H4:(envTp Y Z)`, but this would lead to an inconsistent typing context.
- The layering technique seems to work well (the formal development is still in progress), but it complicates a bit the bookkeeping predicate and the related machinery.
  - However, the necessary overhead is reasonable and less cumbersome than that of the deep embedding approach.