Mechanizing type environments in weak HOAS

metatheory

e.g

a.k.a

weak
`Coq`

typing environment *i.e*

[1] *type*
language <:

first <:

i.e

lists
deep

<:

second
shallow
<: *adequate* *i.e* *w.r.t*

Theory of Contexts
(ToC)

<:

[1] From the point of view of the typical issues to be faced when mechanizing a formal system.

deep vs.

shallow

concise/elegant/usable/etc.

i.e how close

how far *w.r.t*

α

Synopsis. $<:$

Coq

Coq

2. A paradigmatic case study: System F $<:$

i.e $<:$

on paper $<:$

$<:$ *well-scoping*

on purpose

$<:$

shallow

i.e α

In fact, the first task of the Challenge addresses the relationship between the two subtyping versions, as it consists to prove that the transitivity property (3) is a derivable property within the algorithmic system (the same is required for reflexivity (2), a goal which is less problematic, though).

The proof of the transitivity is challenging essentially in two respects: it has to be proved together with the narrowing property, and the whole proof requires a mutual and nested induction proof argument[2].

Proposition 1 (Transitivity). Let $\Gamma \in Env$, and $S, Q, T, X, M, N, P \in Type$:
1) If $\Gamma \vdash S <: Q$ and $\Gamma \vdash Q <: T$, then $\Gamma \vdash S <: T$.
2) If $\Gamma, X <: Q, \Delta \vdash M <: N$ and $\Gamma \vdash P <: Q$, then $\Gamma, X <: P, \Delta \vdash M <: N$.

Proof. 1), 2) are proved together, by induction on the structure of the type Q.

1) The proof for transitivity proceeds by an inner induction on the structure of the derivation $\Gamma \vdash S <: Q$, with a case analysis on the final rule of such a derivation and on that of the second hypothesis $\Gamma \vdash Q <: T$. We illustrate the crucial case when both the derivations end with an application of the (All) rule (where it holds $S \equiv \forall X <: S_1.S_2$, $Q \equiv \forall X <: Q_1.Q_2$, $T \equiv \forall X <: T_1.T_2$):

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash Q_1 <: S_1} \quad \dfrac{\vdots}{\Gamma, X <: Q_1 \vdash S_2 <: Q_2}}{\Gamma \vdash \forall X <: S_1.S_2 <: \forall X <: Q_1.Q_2}\ (\text{All}) \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash T_1 <: Q_1} \quad \dfrac{\vdots}{\Gamma, X <: T_1 \vdash Q_2 <: T_2}}{\Gamma \vdash \forall X <: Q_1.Q_2 <: \forall X <: T_1.T_2}\ (\text{All})$$

To conclude $\Gamma \vdash \forall X <: S_1.S_2 <: \forall X <: T_1.T_2$ via the (All) rule, two premises are needed: first, $\Gamma \vdash T_1 <: S_1$ may be derived by induction hypothesis from the third and the first subderivations; however, the induction hypothesis cannot be applied to the second and fourth ones (to deduce $\Gamma, X <: T_1 \vdash S_2 <: T_2$), because their environments are different. Hence, the narrowing property, i.e., the outer induction hypothesis (being Q_1 structurally smaller than Q) has to be exploited, to derive $\Gamma, X <: T_1 \vdash S_2 <: Q_2$ from the second and the third subderivations. To construct the required derivation $\Gamma, X <: T_1 \vdash S_2 <: T_2$ from this last hypothesis and the fourth subderivation, it is necessary to apply again the outer induction hypothesis (i.e., the transitivity itself, with Q_2 structurally smaller than Q).

2) Similarly, the proof for narrowing proceeds by an inner induction on the structure of the derivation $\Gamma, X <: Q, \Delta \vdash M <: N$, again with a case analysis on the final rule applied. The treatment of this "twin" property is even subtler when the last rule applied is (Trans), and M is exactly X:

$$\dfrac{\dfrac{\vdots}{\Gamma, X <: Q, \Delta \vdash Q <: N}}{\Gamma, X <: Q, \Delta \vdash M \equiv X <: N}\ (\text{Trans})$$

Now, $\Gamma, X <: P, \Delta \vdash Q <: N$ is derived by induction hypothesis, and $\Gamma, X <: P, \Delta \vdash P <: Q$ via a straightforward weakening property. This time, the outer induction

[2] The proof of transitivity is reported in [10, 22], albeit not in a fully detailed fashion.

with the same Q

$\Gamma, X<:P, \Delta \vdash P <: N$

$\Gamma, X<:P, \Delta \vdash X <: N \quad \square$

2.2. System $F_{<:}$ in sequent-style

$<:$

scoping

types
subtyping

Coq

type environment
$\langle X, T \rangle$ *formally*
closure
T w.r.t Γ *closed*
Type Env *T*
well-formedness *ok Env*
$\langle X, T \rangle$ Γ *X*
w.r.t Γ *T* *T* *w.r.t*
fv T
T
$2, \not\in$

Definition 1 . *For* $\Gamma = \langle X_1, T_1 \rangle, \ldots, \langle X_n, T_n \rangle$ *Env,* *T* 2 *Type, the* *of* Γ *and the predicates* , *are defined as follows:*

dom $\triangleq \{X_1, \ldots, X_n\}$ *closed T,* $\triangleq \forall Y.\ Y \in fv\ T \Rightarrow \exists\ U.\ \langle Y, U \rangle \in \Gamma$

$$\frac{}{ok\ \cdot} \quad ok\ \cdot \qquad \frac{ok\ \Gamma \quad X \notin dom\ \Gamma \quad closed\ T,}{ok\ \Gamma, \langle X, T \rangle} \quad ok\ pair$$

$X \not\in fv\ T$

ok pair

$\Gamma \vdash S <: T \Delta$ *sub* Γ, S, T

sub *sub* *Env Type Type*

Definition 2 . *Assume* $\Gamma \in Env, S, S_1, S_2, T, T_1, T_2, U \in Type$. *Then,*

the predicate is defined by induction, as follows:

$$\frac{ok \quad closed\ S,}{sub\ \ ,S,Top}\ top \qquad \frac{ok \quad \mathsf{h}X,U\mathsf{i}\ 2\ \square}{sub\ \ ,X,X}\ var$$

$$\frac{\mathsf{h}X,U\mathsf{i}\ 2\ \square \quad sub\ \ ,U,T}{sub\ \ ,X,T}\ trs \qquad \frac{sub\ \ ,T_1,S_1 \quad sub\ \ ,S_2,T_2}{sub\ \ ,S_1!\ \ S_2,T_1!\ \ T_2}\ arr$$

$$\frac{sub\ \ ,T_1,S_1 \quad for\ all\ X,\ ok\ \ ,\mathsf{h}X,T_1\mathsf{i}\ \ )\ \ sub\ \ \square,\mathsf{h}X,T_1\mathsf{i}\ \ ,S_2,T_2}{sub\ \ ,\mathsf{8}X{<}S_1.S_2,\mathsf{8}X{<}T_1.T_2}\ all$$

sequent-style
i.e *derivation asser-*
tions *premises* *conclusions* $S{<}T$

same
well-formedness *compatibility*
formally

Lemma 1 . *Let be* $\square 2 Env$, *and* $S,T 2 Type$:
1) $sub\ \ ,S,T$) $ok\ \ ;$
2) $sub\ \ ,S,T$) *closed* $S,\ \ \wedge$ *closed* $T,\ \ .$

Proof.
$sub\ \ ,S,T$ □

Theorem 1 . *Let be* $\square 2 Env$, *and* $S,T 2 Type$:
$sub\ \ ,S,T$ *if and only if* $\square \ `\ S < T.$

Proof. □

w.r.t
l *perm*

Lemma 2 . *Let be* $\square,\ \ 2 Env$, *and* $X,P,Q,S,T 2 Type$:
1) Well-formedness: $ok\ \ ,\mathsf{h}X,Q\mathsf{i},\ \ \wedge\ sub\ \ ,P,Q$) $ok\ \ ,\mathsf{h}X,P\mathsf{i},\ \ ;$
2) Permutation: $sub\ \ ,S,T\ \wedge\ ok\ perm\ \ )\ sub\ perm\ \ ,S,T\ ;$
3) Weakening: $sub\ \ ,S,T\ \wedge\ ok\ \ ,\ \ )\ sub\ \ \square\ \ ,S,T\ .$

Proof.
$sub\ \ ,S,T$
$sub\ \ ,S,T$ □

Proposition 2 . *Let be* , 2 *Env, S, Q, T, X, M, N, P* 2 *Type:*

Reflexivity: ok ^ *closed S,*) *sub* , *S, S* .
Transitivity: sub , *S, Q* ^ *sub* , *Q, T*) *sub* , *S, T* .
Narrowing: sub , hX, Qi , , *M, N* ^ *sub* , *P, Q*) *sub* , hX, Pi , , *M, N* .

Proof. (Reflexivity) *S* *(Transitivity and Narrowing)* *Q*
w.r.t *Q*
(Transitivity) *Q* *Top* *Q* *Y*
sub , *S, Y* *Q* *U* ! *V*
sub , *S, U* ! *V* *i.e*
U *V*
Q *Q* 8*Y* < *U.V* *sub* , *S,* 8*Y* < *U.V*

U *V* *U* *V*
Q
(Narrowing)
sub , hX, Qi , , *M, N* *trs*
Q *Top*
i.e *Q*
trs *Q* *Top*
□

2.3. System $F_{<:}$ *in natural deduction-style*

<:
natural deduction

conclusions A
premises and
‘ A ‘ A
root A *leaves*

‘ *S* T *sub*$_N$ *S, T*
sub$_N$ *sub*$_N$ *Type Type*
associations
bookkeeping *book* *Var Type*
Var

scoping *closed*$_N$ *Type*
T

$$closed_N\ T \triangleq \forall Y.\ Y \in fv\ T \Rightarrow \exists\ U.\ book\ Y, U$$

<:

De nition 3 . *Assume* $S, S_1, S_2, T, T_1, T_2, X, U 2 Type$. *In the* all_N *rule below, let* $fresh\ X$ *stand for the two conditions* $X 2 / fv\ S_1$ [$fv\ T_1$ [$fv\ S_2$ nf$X$g [$fv\ T_2$ nf$X$g *and there does not exist any* $V 2 Type$ *different from* T_1 *such that* $book\ X, V$. *Then, the predicate* sub_N *is defined by induction:*

$$\frac{closed_N\ S}{sub_N\ S, Top}\ top_N \qquad \frac{book\ X, U \quad sub_N\ U, T}{sub_N\ X, T}\ trs_N$$

$$\frac{book\ X, U}{sub_N\ X, X}\ var_N \qquad \frac{sub_N\ T_1, S_1 \quad sub_N\ S_2, T_2}{sub_N\ S_1 !\ S_2, T_1 !\ T_2}\ arr_N$$

$$\frac{\begin{array}{c} fresh\ X\ \hat{}\ book\ X, T_1 \\ \vdots \\ sub_N\ T_1, S_1 \qquad sub_N\ S_2, T_2 \end{array}}{sub_N\ 8X{<}S_1.S_2, 8X{<}T_1.T_2}\ all_N$$

all_N *conditional*

discharged i.e
local
$sub_N\ S, T$
S T *book*

Lemma 3 . *Let be* $S, T 2 Type$: $sub_N\ S, T$) $closed_N\ S\ \hat{}\ closed_N\ T$

Proof. $sub_N\ S, T$ □

Coq

w.r.t

3. A rst encoding with explicit type environments

Coq <:
sequent-style

3.1. Higher-order representation of syntax and binders

α
weak HOAS

Var *non-inductive*
Coq
<: *variables*
X Y X Y Var

inductive Tp $<:$ *types*[3]

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

$<:$ 8 fa
higher-order Var
Tp
Coq X
8$X<S.T$ S S
T[X] T
X X
8$X<S.T$ (fa S (fun X:Var => T[X]))
X fun
α
Coq

3.2. Type environments as lists of pairs

type environments
lists
Var Tp

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

i.e

i.e

w.r.t
isin notin

```
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).
Inductive notin (X:Var): Tp -> Prop := notin_top: notin X top
| notin_var: forall Y:Var, ~X=Y -> notin X Y
| notin_arr: forall S T:Tp, notin X S -> notin X T -> notin X (arr S T)
| notin_fa : forall S:Tp, forall U:Var->Tp, notin X S ->
    (forall Y:Var, ~X=Y -> notin X (U Y)) -> notin X (fa S U).
```

[3]Notice that var is declared as a coercion operator, which avoids to type explicitly the constructor, where a variable should stand for a term of type Tp.

(isin X T) X T

$X \in fv\ T$ (notin X T)

$X \notin fv\ T$

Tp top isin

notin_ho notin

i.e

```
Definition notin_ho:= fun X: Var => fun S: Var->Tp =>
  forall Y: Var, ~X=Y -> (notin X (S Y)).
```

$X \notin dom$ Gfresh $\langle X, T \rangle \in$ isinG

closed T, Gclosed *w.r.t*

```
Inductive Gfresh (X:Var): envTp -> Prop := GfVoid: Gfresh X nil
| GfGrow: forall G:envTp, forall Y:Var, forall T:Tp,
      Gfresh X G -> ~X=Y -> Gfresh X (cons (Y,T) G).
Inductive isinG (X:Var) (T:Tp): envTp -> Prop :=
  checkG: forall G:envTp, forall y:Var, forall U:Tp,
      (X=Y /\ T=U) \/ isinG X T G -> isinG X T (cons (Y,U) G).
Definition Gclosed (T:Tp) (G:envTp): Prop :=
  forall X:Var, (isin X T) -> exists U:Tp, isinG X U G.


Inductive okEnv: envTp -> Prop := okVoid: okEnv nil
| okGrow: forall G:envTp, forall x:Var, forall T:Tp, okEnv G ->
      Gfresh X G -> Gclosed T G -> okEnv (cons (X,T) G).
```

3.3. Encoding the subtyping system

sub

subG_fa

```
Inductive subGTp: envTp -> Tp -> Tp -> Prop :=
  subG_top: forall G:envTp, forall S:Tp,
          okEnv G -> Gclosed S G -> subGTp G S top
| subG_var: forall G:envTp, forall X:Var, forall U:Tp,
          okEnv G -> isinG X U G -> subGTp G X X
| subG_trs: forall G:envTp, forall X:Var, forall U T:Tp,
          isinG X U G -> subGTp G U T -> subGTp G X T
| subG_arr: forall G:envTp, forall S1 S2 T1 T2:Tp,
          subGTp G T1 S1 -> subGTp G S2 T2 ->
          subGTp G (arr S1 S2) (arr T1 T2)
| subG_fa: forall G:envTp, forall S1 T1:Tp,
          forall S2 T2:Var->Tp, subGTp G T1 S1 ->
          (forall X:Var, okEnv (cons (X,T1) G) ->
              subGTp (cons (X,T1) G) (S2 X) (T2 X)) ->
          subGTp G (fa S1 S2) (fa T1 T2).
```

subG_fa *all*

[4] X

X T1

G X *w.r.t*

G X T1 *X* T_1

$\forall X <: T_1. T_2$ (fa T1 T2)

X S2

T2 S2 T2 X

4. The Theory of Contexts

$<:$ $\forall$ Coq

e.g

Var *i.e*

exotic terms[5]

Theory of Contexts

variable *context*[6]

e.g Coq

M *context* *i.e*

holes *e.g* *M* ! ! *Top*

M *X*

M X *X*! *X* ! *Top* *i.e*

X *M*

Var->T Var

T

M X (M X)

e.g *N* $\forall$ *Y* <: *T*. ! ! ! *Y* *N X Z*

$\forall Y <: T.$ *X*! *Z* ! *X*! *Y*

N

Var->Var->T *N X Z* (N X Z)

---

[4] The freshness of X is granted by the scoping rules of Coq. This means that if, by chance, S2 or T2 should contain an occurrence of a free variable with the same name, the locally bound X would be automatically renamed by the system, in order to avoid captures. However, such notion of freshness, being delegated to the metalanguage, is not available at the object level. If we need it during our proof development activity, we must explicitly add some premises using the notin/notin_ho predicates (see, *e.g.*, the shallow encoding of the subtyping relation in Section 8.2 and, in particular, the encoding of the universal binder rule).

[5] Exotic terms are legal terms derivable in the LF at hand, which do not correspond to any entity of the object language. Hence, they hinder the adequacy of the encoding [3].

[6] Contexts are "terms with holes", where the holes can be filled in by variables.

Decidability of equality over variables. $X \quad Y$
$X \quad Y \quad X \mathbin{6} Y$
Coq

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

LEM *Law of Excluded Middle*[7]

Freshness/Unsaturation. M X
i.e
Tp

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

```
Axiom unsat': forall T U:Tp, exists X:Var,
      notin X T /\ notin X U /\ envBook X U.
```

envbook

Extensionality.
$M\ X \quad N\ X \quad X \not\in fv\ M \quad [\ fv\ N \quad M \quad N$

```
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.
```

```
Axiom envTp_ext: forall (x : Var) (G G' : Var -> envTp),
      notin_envTp_ho x G -> notin_envTp_ho x G' ->
      G x = G' x -> G = G'.
```

β-expansion.
[8] $M \quad X \quad N$
$N\ X \quad M \quad X \not\in fv\ N \quad \beta$
i.e
i.e

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

[7] This is the minimal classical flavor that we require to reason about (free) occurrences of variables. Such an assumption is very close to the common practice, "on paper", with nominal systems like, *e.g.*, process algebras or typing systems.

[8] In presence of binders, such a property is not derivable.

9

fresh-renaming

β-expansion *extensionality*

structural induction over contexts
monotonicity

isin

```
Lemma isin_mono: forall T:Var->Tp, forall X Y:Var, X<>Y ->
         isin X (T Y) -> (forall Z: Var, X<>Z -> isin X (T Z)).
```

T:Var->Tp Coq
i.e (T Y)
T (T
Y) Y T

```
Lemma pre_isin_mono: forall n:nat, forall U:Tp,
      lntp U n -> forall V:Var, forall T:Var->Tp,
      notin_ho V T -> U=(T V) ->
      forall X Y:Var, X<>Y -> isin X (T Y) ->
      forall Z:Var, X<>Z -> isin X (T Z).
```

lntp Tp

```
Inductive lntp: Tp -> nat -> Prop :=
| lntp_top: lntp top 1
| lntp_var: forall X:Var, lntp X 1
| lntp_arr: forall T T':Tp, forall n1 n2:nat,
  lntp T n1 -> lntp T' n2 -> lntp (arr T T') (S (plus n1 n2))
| lntp_fa : forall T:Tp, forall U:Var->Tp, forall n1 n2:nat,
  lntp T n1 -> (forall X:Var, lntp (U X) n2) ->
  lntp (fa T U) (S (plus n1 n2)).
```

lntp
Coq Tp

fresh renamings
(lntp U n) U n
Tp *complete*
n pre_isin_mono
U (lntp U n) *β*
U':Var->Tp U=(U' V)

[9]Their consistency has been proved in [16], starting from an idea of M. Hofmann [24].

V U’
T=U’ U’ λ

(lntp U 1)
U=(T
V) U=top U’=(fun X:Tp => top)
(T V)=top=(U’ V) (T V)=((fun X:Tp =>
top) V)
T=(fun X:Tp => top) *i.e*
T

5. A formal development of the metatheory of System F $<:$

Coq
i.e

5.1. Basic properties

```
Lemma Gclosed_lemma: forall G:envTp, forall S T:Tp,
                     subGTp G S T -> Gclosed S G /\ Gclosed T G.
```

(subGTp G S T)
G
S T G
(subGTp G S T)
unsatG *w.r.t* G

```
Lemma unsatG: forall G:envTp, exists X:Var, Gfresh X G.
```

unsatG
unsat G
(X1,T1) (Xn,Tn) G
(arr X1 (arr ... (arr Xn top) ...))
unsat
G

```
Lemma domGtoT_notin: forall G:envTp, forall X:Var,
                     notin X (domGtoT G) -> Gfresh X G.
```

domGtoT G

```
Fixpoint domGtoT (G:envTp):= match G with
         | nil => top | (X,T)::G' => (arr X (domGtoT G')) end.
```

domGtoT_notin G
LEM_Var

5.2. Reflexivity, transitivity and narrowing

reflexivity
w.r.t

```
Lemma reflexivity: forall T:Tp, forall G:envTp,
                  okEnv G -> Gclosed T G -> subGTp G T T.
```

T
LEM_Var
isin

Q

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

transitivity unsat
fa
(subGTp G S Q)

induction
narrowing
(subGTp G' M N) G' Coq (app D (cons (X,Q) G)) *append* app
subG_var subG_trs LEM_Var
w.r.t

Coq *lists*
Gfresh isinG Gclosed okEnv
Coq
permutations

$8A, B$ *Prop.* $A \wedge A) B) A \wedge B$ A B

subG_trs
i.e *starting* Q

Q=top

6. Records in System F $_{<:}$

record types i.e Coq

6.1. Adding records on paper

$Type$ S, T $\mathsf{f}\, l_i \;\; T_i \mathsf{g}^{i \in 1..n}$ l_i $n 2 \mathbb{N}$

$$\frac{\mathsf{f}\, l_i \mathsf{g}^{i \in 1..n} \;\; \mathsf{f} \;\; k_j \mathsf{g}^{j \in 1..m} \qquad k_j \;\; l_i \qquad \square \; \text{`} \;\; S_j < T_i}{\square \; \text{`} \; \mathsf{f} \;\; k_j \;\; S_j \mathsf{g}^{j \in 1..m} < \mathsf{f}\, l_i \;\; T_i \mathsf{g}^{i \in 1..n}} \; \square\square\square\square\square$$

$<:$

labels *wt*
$Type$ *sub*

De nition 4 . *For* Z hX1, T1i,..., hXn, Tni2 *Env*, T2 *Type*, *the predicate* *is defined by induction, as follows:*

$$\frac{}{wt\ Top}\ wt\ top \qquad \frac{wt\ S \quad wt\ T}{wt\ S!\ T}\ wt\ arr \qquad \frac{wt\ S \quad wt\ T}{wt\ 8X < S.T}\ wt\ all$$

$$\frac{}{wt\ X}\ wt\ var \qquad \frac{distinct \mathsf{f}\, l_1, ..., l_n \mathsf{g} \quad for\ all\ i 2\ ..n,\ wt\ T_i}{wt\ \mathsf{f}\, l_i \;\; T_i \mathsf{g}^{i \in 1..n}}\ wt\ rcd$$

De nition 5 . *Assume* 2 *Env*, *I* ..*n*, *J* ..*m*, $T_j, S_i 2 Type$ $8j 2 J, i 2 I$. *The predicate* *of Definition 2 is augmented with:*

$$\frac{ok \qquad closed\ S, \qquad wt\ S \quad \mathsf{f}\, l_i \mathsf{g}^{i \in I}\ \mathsf{f} \;\; k_j \mathsf{g}^{j \in J} \quad k_j \;\; l_i \;\;) \;\; sub \;\; , S_j, T_i}{sub \;\; , S\, \mathsf{f} \;\; k_j \;\; S_j \mathsf{g}^{j \in J}, T\, \mathsf{f} \;\; l_i \;\; T_i \mathsf{g}^{i \in I}}\ rcd$$

formal
w.r.t
with

closed *wt*
S *T*

Lemma 4 . *Let be* 2 *Env*, $S\ \mathsf{f} \;\; k_j \; S_j \mathsf{g}^{j \in J}$, $T\ \mathsf{f} \;\; l_i \; T_i \mathsf{g}^{i \in I} 2 Type$:
1) $wt\ S \;\; \wedge \mathsf{f}\, l_i \mathsf{g}^{i \in I}\ \mathsf{f} \;\; k_j \mathsf{g}^{j \in J} \;\;) \;\; wt\ T$;
2) $\mathsf{f}\, l_i \mathsf{g}^{i \in I}\ \mathsf{f} \;\; k_j \mathsf{g}^{j \in J} \wedge \;\; k_j \;\; l_i) \;\; closed\ T_i, \quad) \;\; closed\ T, \quad .$

Proof. T □

w.r.t

Proposition 3 . *Let be* □, 2 *Env*, S, Q, T, X, M, N, P2 *Type:*

Reflexivity: ok ^ *closed* S, ^ *wt* S) *sub* , S, S .
Transitivity: sub , S, Q ^ *sub* , Q, T) *sub* , S, T .
Narrowing: sub □ hX, Qi , , M, N ^ *sub* , P, Q) *sub* □ hX, Pi , , M, N .

Proof. (Reflexivity) S

(Transitivity and Narrowing) Q Q f l_i T_ig$^{i \in I}$ *sub* , S, Q

sub □ hX, Qi , , M, N □

6.2. Encoding records and subtyping

`Tp` `Coq`

```
Definition Lab := nat.
Inductive Tp: Set := ... | rcd: list (Lab * Tp) -> Tp.
```

lists

<: *i.e* `Tp`

`Tp` *ignored*[10] `Coq`

`Tp`

mutual

`Coq` `Tp`

`Tp_rcd` `Tp`

sub

[10] These types, occurring recursively in a list, are named *nested types* in the literature.

```
Inductive subGTp: envTp -> Tp -> Tp -> Prop := ...
| subG_rcd: forall G: envTp, forall P Q: list (Lab*Tp),
            okEnv G -> Gclosed (rcd P) G ->
            NoDup (proj_lab P) -> incl (proj_lab Q) (proj_lab P) ->
            (forall p q:Lab*Tp, In p P /\ In q Q /\ (fst p = fst q) ->
                                subGTp G (snd p) (snd q)) ->
            subGTp G (rcd P) (rcd Q).
```

subG_rcd

NoDup

incl *set*

In

fst snd

proj_lab

6.3. Proving reflexivity, transitivity and narrowing

Coq

reflexivity

i.e

à la

Tp_rec Coq

Tp

```
Hypothesis Rcd_case: forall P: Tp->Prop, forall L: list (Lab*Tp),
                     Rcd2Tp Lab Tp P L -> P (rcd L).
```

P (rcd L) P

L Rcd2Tp

L P

Tp

Tp_rec Tp_rec_ext

Rcd_case

narrowing

transitivity Q

(rcd L)

L

Q

Tp_rec_ext Tp_rec

7. Practical remarks

Coq

Coq <:

isin notin	

Table 1: Coq scripts statistics.

<: Coq

isin notin

<: *i.e* Coq

8. A second encoding with implicit type environments

Coq

8.1. The bookkeeping technique in Coq

book

Coq

```
Parameter envBook: Var -> Tp -> Prop.
```

Var envBook *open i.e non-inductive*

$X_1 <: T_1, \ldots, X_n <: T_n$ Xi:Var Ti:Tp

```
Parameter d1:envBook X1 T1. ... Parameter dn:envBook Xn Tn.
```

i.e *closed*

```
Definition closed (T:Tp): Prop := forall X:Var,
          isin X T -> exists U:Tp, envBook X U.
```

8.2. A shallow encoding of the subtyping system

sub_N

subGTp

```
Inductive subTp: Tp -> Tp -> Prop :=
| sub_top: forall S:Tp, closed S -> subTp S top
| sub_var: forall X:Var, forall U:Tp, envBook X U -> subTp X X
| sub_trs: forall X:Var, forall U T:Tp,
          envBook X U -> subTp U T -> subTp X T
| sub_arr: forall S1 S2 T1 T2:Tp, subTp T1 S1 -> subTp S2 T2 ->
          subTp (arr S1 S2) (arr T1 T2)
| sub_fa : forall S1 T1:Tp, forall S2 T2:Var->Tp,
          forall L:list(Var), subTp T1 S1 ->
          (forall X:Var, (notin X S1) -> (notin X T1) ->
                (notin_list X L) ->
                (notin_ho X S2) -> (notin_ho X T2) ->
                (envBook X T1 -> subTp (S2 X) (T2 X))
          ) -> subTp (fa S1 S2) (fa T1 T2).
```

sub_fa all_N X

T1

envBook S1

X *fresh w.r.t*

X

some L L

notin_list

X

T $\forall X < S.T$

S_2 $,T_2$

notin_ho

S2 T2 X

9. Internal adequacy

consistency

(envBook X T)

e.g envBook

subGTp subTp

e.g

envBook

sub_fa

X notin notin_ho notin_list[11]

e.g h$X <$ Ti2 I
d (envBook X T)

Coq
e.g

envBook
excessive degree of freedom

discipline i.e

Coq *i.e* *internal*

```
Lemma exp2imp: forall G:envTp, forall S T:Tp,
  subGTp G S T -> (book2Prop (envTp2envBook G)) -> subTp S T.
Lemma imp2exp: forall S T:Tp, subTp S T -> forall G:envTp, okEnv G ->
  (forall X:Var,forall U:Tp,envBook X U <-> isinG X U G) -> subGTp G S T.
```

exp2imp
G (subGTp G S T)
envBook envTp2envBook: envTp -> list
Prop (envBook
X T)
(subTp S T)
imp2exp (subGTp G S T)
(subTp S T) envBook
G *i.e* (okEnv G)
envBook

```
forall X:Var, forall U:Tp, envBook X U <-> isinG X U G
```

[11] As already noticed in Section 8, the list L may take into account any variable in the current typing environment, not only those directly involved in the judgments at hand.

(envBook X U) *if and only if* X U G

9.1. Completeness

exp2imp (subGTp G S T) ⊢ $S < T$

```
forall G S T, subGTp G S T -> okEnv G
```

Gclosed *w.r.t* closed

```
forall S G, Gclosed S G ->
(book2Prop (envTp2envBook G)) -> closed S
```

⟨X, U⟩ ∈ G

X U

```
forall G X U, isinG X U G ->
(book2Prop (envTp2envBook G)) -> (envBook X U)
```

9.2. Soundness

Coq imp2exp (subTp S T) envBook (subGTp G S T) G envBook

fresh variable-renamings *complete* Coq induction

```
Lemma Gfresh_rw: forall G:envTp, forall G':Var->envTp,
  forall Z:Var, notin_envTp_ho Z G' -> G=(G' Z) ->
  forall X:Var, X<>Z -> Gfresh X G ->
  forall Y:Var, X<>Y -> Gfresh X (G' Y).
```

Gfresh_rw $Z \notin Z$ Z $' \notin X \neq Z$ $X \notin dom$ Y $X \neq Y$ $X \notin dom$ $' Y$ *i.e* X *dom*

```
Lemma isinG_rw: forall G:envTp, forall U:Tp, forall G':Var->envTp,
  forall X:Var, notin_envTp_ho X G' -> G=(G' X) -> isinG X U G ->
  forall Y:Var,
    X<>Y -> notin_envTp_ho Y G' -> exists U':Tp, isinG Y U' (G' Y).
```

isinG_rw

$X<U \ 2 \ \square$ U $X2fv \ '$ $Z\square X$ $Y \ X6Y$ $Y2fv \ '$ U' $Y<U' \ 2 \ \square' Y$

```
Lemma Gclosed_rw: forall T:Tp, forall T':Var->Tp,
  forall G:envTp, forall G':Var->envTp,
  forall X:Var, notin_ho X T' -> T=(T' X) ->
    notin_envTp_ho X G' -> G=(G' X) -> Gclosed T G ->
  forall Y: Var, X<>Y -> notin_ho Y T' -> notin_envTp_ho Y G' ->
    Gclosed (T' Y) (G' Y).
```

closed

$closed \ T, \quad X2fv \ ' \quad [\ fv \ T'$ $Z\square X \quad T \quad T' \ X$ $closed \ T' \ Y$,
$\square' Y$ Y $X6Y$ $Y2fv \ ' \quad [\ fv \ T'$

```
Lemma okEnv_rw: forall G:envTp, forall G':Var->envTp,
  forall X:Var, notin_envTp_ho X G' -> G=(G' X) ->
  forall Y:Var, X<>Y -> notin_envTp_ho Y G' ->
  okEnv G -> okEnv (G' Y).
```

okEnv_rw $\square$

$\square$ $X2fv \ '$ $Z\square' X$
$\square' Y$ Y $X6Y$ $Y2fv \ '$

```
Lemma subGTp_rw: forall S T:Var->Tp, forall X:Var,
  forall U:Tp, forall G:envTp, notin_ho X S -> notin_ho X T ->
  subGTp ((X, U) :: G) (S X) (T X) ->
  forall Y:Var,
    X<>Y -> notin Y U -> notin_ho Y S -> notin_ho Y T -> Gfresh Y G ->
    subGTp ((Y, U) :: G) (S Y) (T Y).
```

subGTp_rw

$, X<U \ ` \ S \ X < T \ X$
$X2fv \ S \quad [\ fv \ T$ $, Y<U \ ` \ S \ Y < T \ Y$
$Y \quad X6Y \quad Y2fv \ S \quad [\ fv \ T$

G Gfresh_rw okEnv_rw
Tp isinG_rw Gclosed_rw
subGTp subGTp_rw *i.e*

lntp
Coq

9.3. Deep vs. shallow

i.e Coq

Coq

imp2exp

Coq

```
forall X:Var, forall U:Tp, envBook X U <-> isinG X U G
```

Coq

protocol

imp2exp (okEnv G)

G

G

G *i.e* (okEnv G) Coq

$$\dfrac{\dfrac{\dfrac{book(Z,Y)}{closed(Z)}\,(def.)}{sub(Z,Top)}\,(top) \qquad \dfrac{book(X,Z)\,(1) \qquad \dfrac{\dfrac{book(Z,Y)}{sub(Z,Y)}\,(trs) \qquad \dfrac{book(Y,Top)}{sub(Y,Y)}\,(var)}{sub(Z,Y)}\,(trs)}{sub(X,Y)}}{sub(\forall X{<}Top.X,\ \forall X{<}Z.Y)}\,(all)$$

[12]

all

```
Lemma sampleShallow: forall Y Z:Var,
      envBook Y top -> envBook Z Y ->
      subTp (fa top (fun X:Var => X)) (fa Z (fun X:Var => Y)).
```

```
Lemma envWF: forall Y Z:Var, Y <> Z -> okEnv ((Z,(var Y))::(Y,top)::nil).
```

informally

tout court

[12] As usual, local hypotheses are indexed with the rules they are discharged by.

```
Lemma sampleDeep: forall Y Z:Var, Y <> Z ->
      subGTp ((Z,(var Y))::(Y,top)::nil)
             (fa top (fun X:Var => X)) (fa Z (fun X:Var => Y)).
```

one

e.g

$X < Q$

$X < P$ P 6 Q

(envBook X Q) (envBook X P)

envBook

subTp

i.e

10. Conclusion

10.1. Related work

α

freshness

Coq *traditional*

Var *i.e*

exotic terms

i.e

validity judgment *i.e*

reify

i.e

equivariance[13]

`Coq`

Parametric HOAS (PHOAS)

<:

`PTp` `8V` `. tp V`

`tp V` `Tp`
w.r.t `Var` `V`

The POPLmark Challenge.

`Isabelle/HOL`
`Coq` *pure* *locally nameless*

`Coq`
`Matitа` *named variables*
`Coq`

[13]More precisely, equivariance is a property of sentences of the form $\forall\vec{x}.\phi(\vec{x})$, *i.e.* $\forall\pi,\vec{x}.(\phi(\vec{x}) \Leftrightarrow \phi(\pi\cdot\vec{x}))$, where π is a permutation action.

Coq *nested abstract syntax*

full HOAS Abella

<: 8 all

```
ty type.  top ty.  arrow ty -> ty -> ty.  all ty -> (ty->ty) -> ty.
```

ty

bound:ty->ty->o

ctx:olist->prop

<: wfty:ty->prop

Twelf <:

```
tp: type.  ...  forall: tp -> (tp->tp) -> tp.
```

assm:tp->tp->type

Abella var:tp->type

Abella Twelf

ty tp

<:

Var

```
Parameter Var: Set. Inductive Tp: Set := ... | fa: Tp -> (Var->Tp) -> Tp.
```

α

Coq

Tp Abella Twelf

types i.e

Tp

Isabelle/HOL

Nominal (Logic)

atom decl *tyvrs*
nominal datatype *ty*
j *Tvar tyvrs* j *Top* j *Arrow ty ty* ! ,
j *Forall tyvrs ty ty*

<:

8 ...

α

finite support *freshness*
`isin`
`notin`

Ind
`Coq`

Ind

10.2. Future work

`Coq`
ς
w.r.t
internally `Coq`
w.r.t *i.e*

e.g

e.g *Abella*

References

LNCS

LNCS

π

LNAI

LNCS