Solution using De Bruijn indices and implicit environments

Arthur Charguéraud
1) Representation of $F_{\preceq}$
Types in Fsub

Inductive typ :=

| top : T |

| arrow: (T1 → T2) |

| all : (∀ T1. T2) |

| var : X |

pointer towards a binding in the environment
Example of a closed type

Polymorphic identity:

\( \forall X <: \text{Top. } (X \to X) \)
A bigger closed type

∀ X <: Top.
∀ U <: (X->Top).
∀ V <: Y. (Y->Y)

this is just to show that pointers can go arbitrary far back up in the tree
Environments and free variables

\[ X <: U, \ Y <: V, \ Z <: W \quad \vdash \quad \forall \ P <: Z. \ (P \rightarrow X) \]
2) Formal definitions
a) Types and environments
Definition of types

**Inductive typ :=**

| top      : typ     |
| arrow    : typ -> typ -> typ |
| all      : typ -> typ -> typ |
| var      : nat -> typ -> typ |

De Bruijn index of the variable

Type to which the variable is mapped to, irrelevant if the variable is not free
Example using labels

Polymorphic identity:

\[ X <: \text{Top} \mid - X \rightarrow X \]

to be written with labels as:

\[ X <: \text{Top} \mid - X^\text{Top} \rightarrow X^\text{Top} \]

^ is the notation for labels
Definition of environments

Parameter env : Set

Parameter env_empty : env

Parameter env_push : env -> typ -> env

Environment as lists

we don't need to give an implementation for type env, since labels on free variables carry all the information that we may need to use

Parameter env_has : env -> nat -> typ -> Prop

"env_has E X T" is a proposition which says that X is mapped to T in the environment E
b) Operations on types
Definition of insert

Fixpoint insert (X : nat) (T : typ) : typ :=
  match T with
  | top         => top
  | arrow T1 T2 => arrow (insert X T1) (insert X T2)
  | all T1 T2   => all (insert X T1) (insert (S X) T2)
  | var Y T1    => var (if le_gt_dec X Y then S Y else Y)
                        (insert X T1)

insert a binding at position X in the implicit environment

- Cross a binder
- Shift the index in case X ≤ Y
Definition of weaken

Fixpoint weaken (X : nat) (T : typ) : typ :=
match X with
| O => insert 0 T
| S P => insert 0 (weaken P T)
end

Note that "weaken X" introduces X+1 variables; this helps simplify some statements and proofs.
Definition of update

"update X U T" puts a label U on all occurrences of X in T

Fixpoint update (X : nat) (U : typ) (T : typ) : typ :=

match T with
| top         => top
| arrow T1 T2 => arrow (update X U T1) (update X U T2)
| all   T1 T2 => all (update X U T1) (update (S X) U T2)
| var   Y  T1 => var Y (if eq_nat_dec X Y
   then weaken Y U
   else update X U T1)

cross a binder
update the label in case X=Y
Definition of push

Definition push := update 0.

"push" is used to pass a binding when exploring a type

"push T1 T2" labels all occurrences of X in T2 by T1

because X has De Bruijn index 0 relatively to T2
c) Well-formation
Well-formation of types

"wf E T" means "type T is well-formed in environment E"

\[
\text{Inductive } \text{wf} : \text{env} \rightarrow \text{typ} \rightarrow \text{Prop} := \\
\text{wf E top} \\
\text{wf E T1} \rightarrow \text{wf E T2} \rightarrow \text{wf E (arrow T1 T2)} \\
\text{wf E T1} \rightarrow (\forall U : \text{typ}, \text{wf (env_push E U)} \text{ (push U T2)}) \rightarrow \text{wf E (all T1 T2)} \\
\text{env_has E X T1} \rightarrow \text{wf E T1} \rightarrow \text{wf E (var X T1)}
\]

if T1 is the label of the free variable X, then X must be mapped to type T1 in the environment E

we need to be able to map the variable bound in T2 not only to T1 but also to some other types (as needed by the rule SA-All)
Well-formation of environments

"wf_env E" holds if and only if E has been constructed by a succession of push of well-formed types

\[
\text{Inductive } \text{wf\_env} : \text{env} \to \text{Prop} := \\
\quad \text{wf\_env env\_empty} \\
\quad \text{wf\_env E \to wf E U \to wf\_env (env\_push E U)}
\]
2) Proving results
a) Properties of the operations
Crossing push with insert and update

**insert on push**: 

\[
\text{insert} \ (S \ X) \ (\text{push} \ T1 \ T2) = \text{push} \ (\text{insert} \ X \ T1) \ (\text{insert} \ (S \ X) \ T2)
\]

\[\forall T1 \ T2\]

**LHS**: we push T1 into T2 and get a type U, and then we insert at level X above U

\[\forall\]

**RHS**: we insert at level X above T1 and get T1', then insert at level X+1 above T2 and get T2', then we push T1' into T2'.

**update on push**: 

\[
\text{update} \ (S \ X) \ P \ (\text{push} \ T1 \ T2) = \text{push} \ (\text{update} \ X \ P \ T1) \ (\text{update} \ (S \ X) \ P \ T2)
\]

An equivalent result for update
**Crossing update at weaken**

\[
\text{update}_\text{at}_\text{weaken} : \\
\quad \text{update } X U (\text{weaken } X T) \\
\quad = \text{weaken } X T
\]

**Lemma:** This lemma says that after we inserted \( X+1 \) variables at the end of the environment, then the function which will update all occurrences of variable with index \( X \) will change nothing: indeed, this variable does not appear in type \( T \).

We use this lemma to capture the fact that if we have an environment of the form "\( \Gamma_1, X <: T, \Gamma_2 \)" then \( X \) has no occurrence in \( T \) (we need that to prove narrowing).
Proof graph for the crossing lemmas

- insert_above_insert
  - insert_below_weaken
    - update_at_insert
    - update_below_weaken
      - update_at_weaken
      - update_above_insert
      - update_above_weaken
      - update_above_update
  - update_above_insert
  - insert_above_weaken
    - update_on_push
    - insert_on_push
Example of a crossing lemma

Lemma insert_above_insert :

\[
\text{insert } (S(X+Y)) \ (\text{insert } X \ T) \\
= \text{insert } X \ (\text{insert } (X+Y) \ T).
\]
Relation between update and equality

update_and_equality :

update X Q T1 = update X Q T2
-> update X P T1 = update X P T2

the intuition behind this lemma is that in narrowing we change from "Γ1 , X <: Q, Γ2" to "Γ1 , X <: P, Γ2" and so need to update the label of each occurrence of X in Γ2.
b) Properties of unsafe subtyping
Statements of properties about unsafe subtyping

\textbf{insert\_preserves\_sub}:

\[ T_1 \preceq T_2 \rightarrow (\text{insert } X \ T_1) \preceq (\text{insert } X \ T_2) \]

\textbf{weaken\_preserves\_sub}:

\[ T_1 \preceq T_2 \rightarrow (\text{weaken } X \ T_1) \preceq (\text{weaken } X \ T_2) \]

\textbf{sub\_reflexivity}:

\[ T \preceq T \]

\textbf{narrowing\_preserves\_sub}:

\[ (\text{update } X \ Q \ S) \preceq (\text{update } X \ Q \ T) \rightarrow P \preceq Q \rightarrow (\text{update } X \ P \ S) \preceq (\text{update } X \ P \ T) \]

\textbf{sub\_transitivity}:

\[ S \preceq Q \rightarrow Q \preceq T \rightarrow S \preceq T \]
Proof graph for results about unsafe subtyping

- `insert_on_push` -> `insert_preserves_sub`
- `sub_reflexivity`
- `update_on_push` -> `insert_preserves_sub`
- `update_on_push` -> `sub_transitivity`
- `update_and_equality` -> `SA-refl`
- `update_at_weaken` -> `SA-trans-tvar`
- `narrow_preserves_sub` -> `SA-all`
- `weaken_preserves_sub` -> `SA-all`
3) Structure of the solution
Structure of the solution (not including tactics)

- Definition of types and the 4 operations
- 9 lemmas about crossing operations
- 4 lemmas describing properties operations
- Definition of size and operations preserve size
- Definition of unsafe subtyping
- Definition of environments and well-formation
- Properties of unsafe subtyping
- Definition of subtyping
- Equivalence of subtyping and unsafe subtyping
- Properties of subtyping