Solution using De Bruijn indices and implicit environments

Arthur Charguéraud

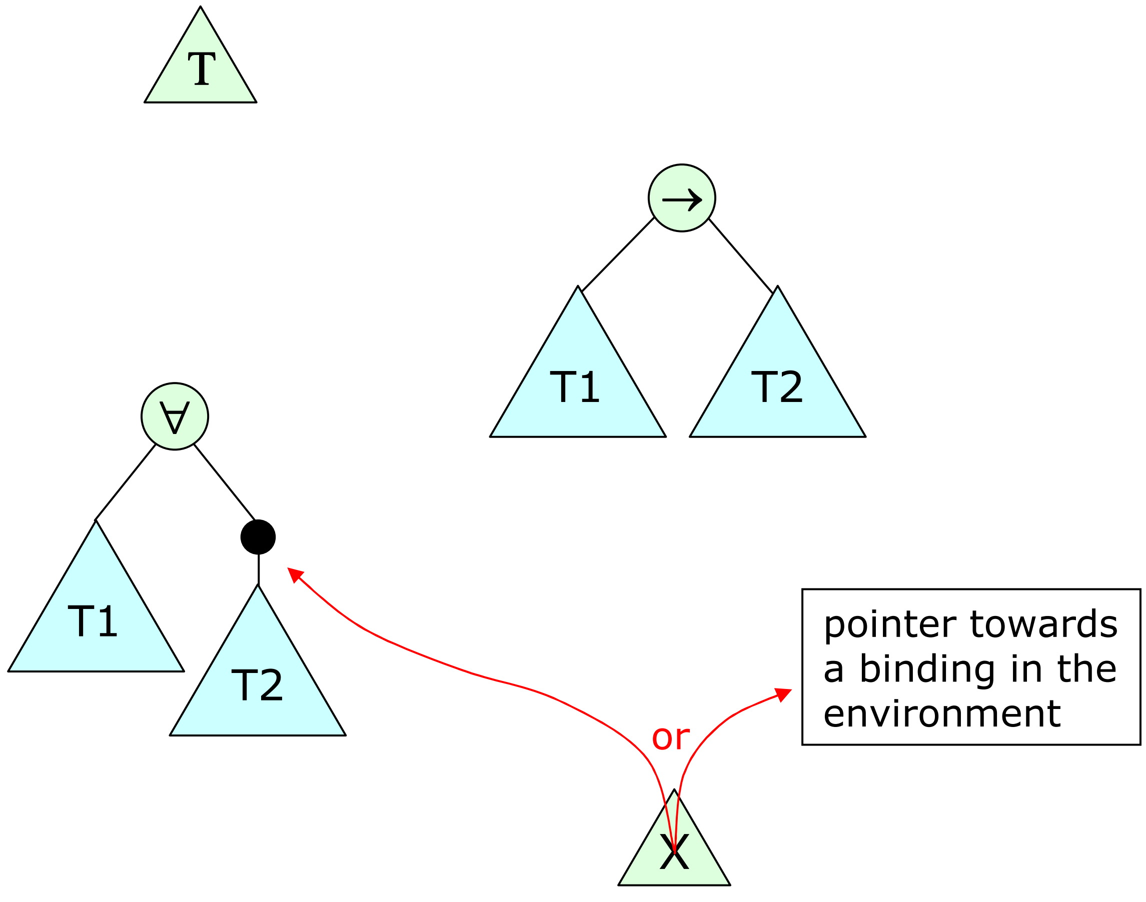
POPLmark Challenge – Part 1A

May 2006

1) Representation of $F_{<:}$

Types in Fsub

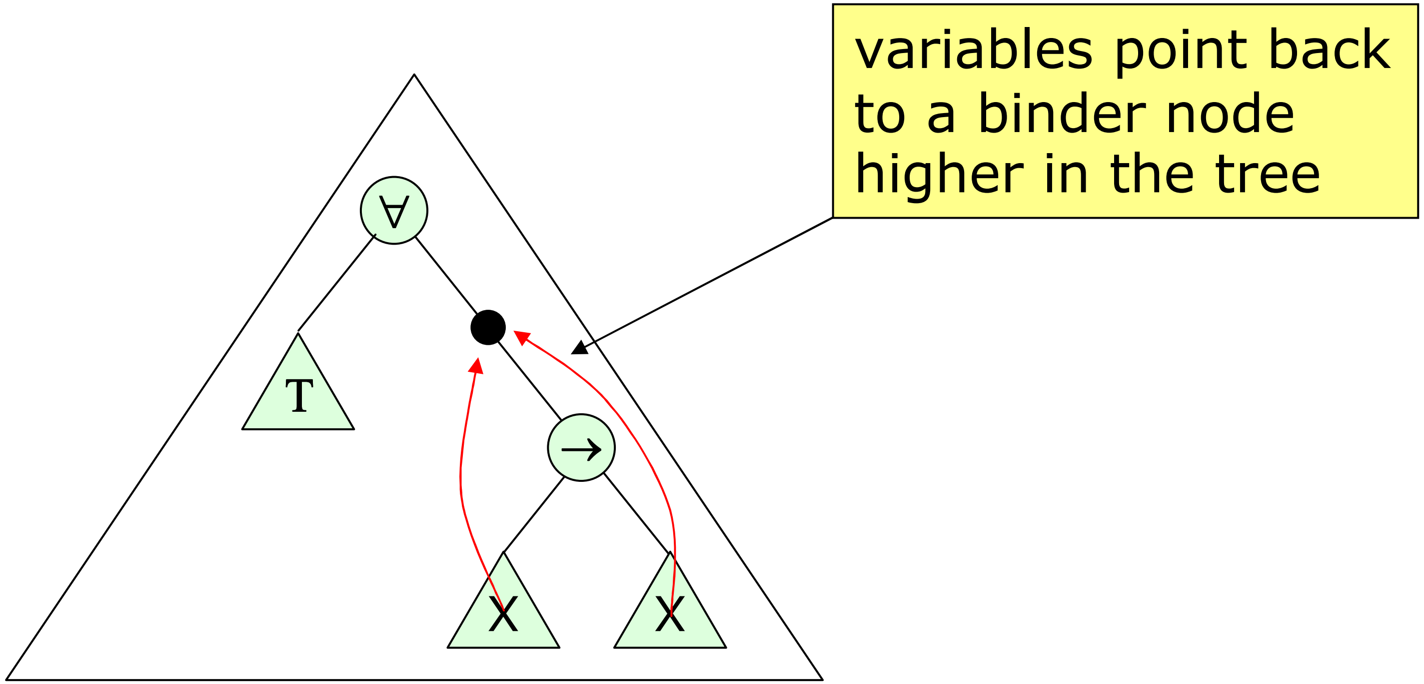
```
Inductive typ :=

  | top   :

  | arrow :

  | all   :

  | var   :
```



Example of a closed type

Polymorphic identity:

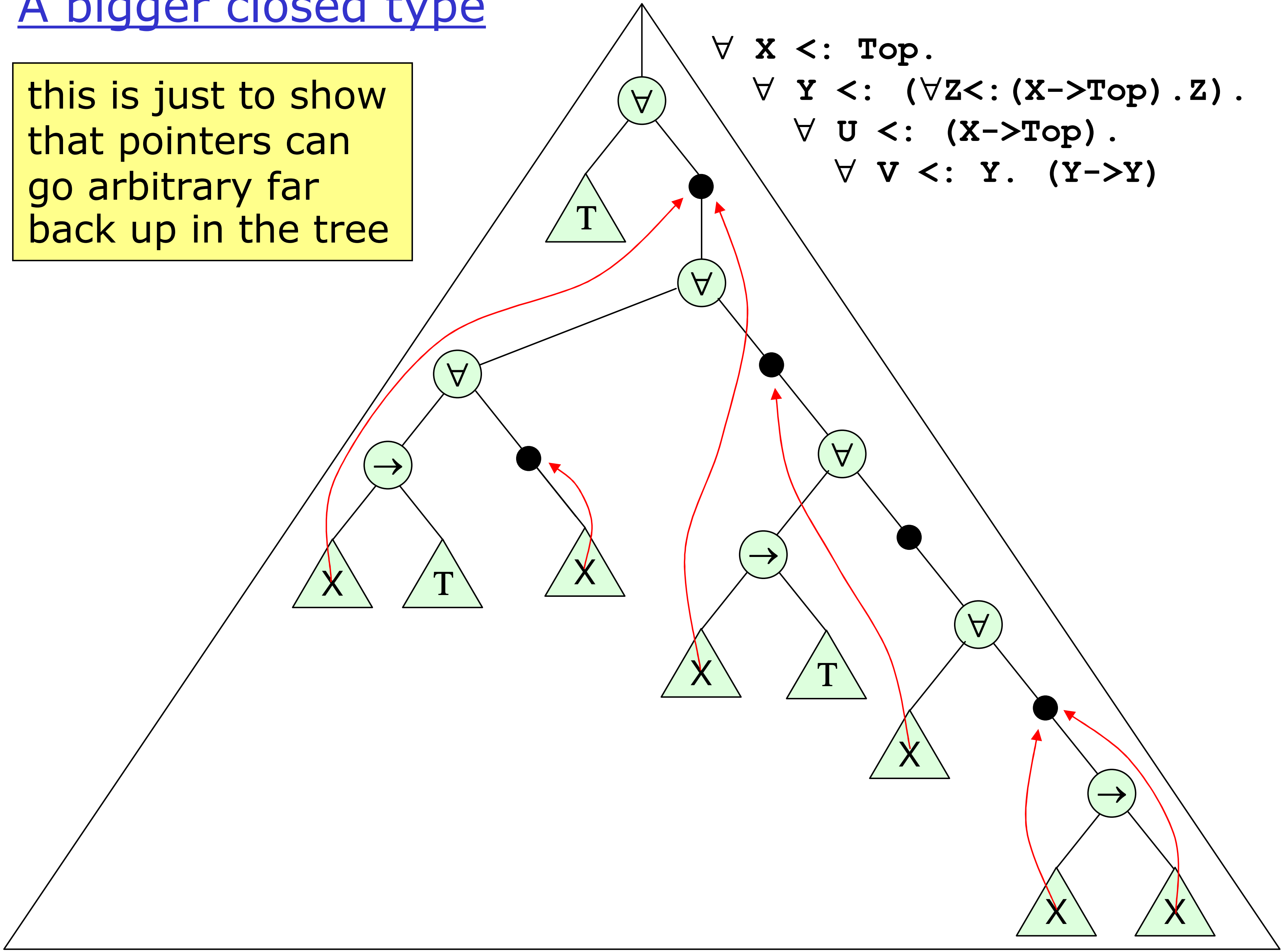
```
∀ X <: Top. (X -> X)
```



A bigger closed type

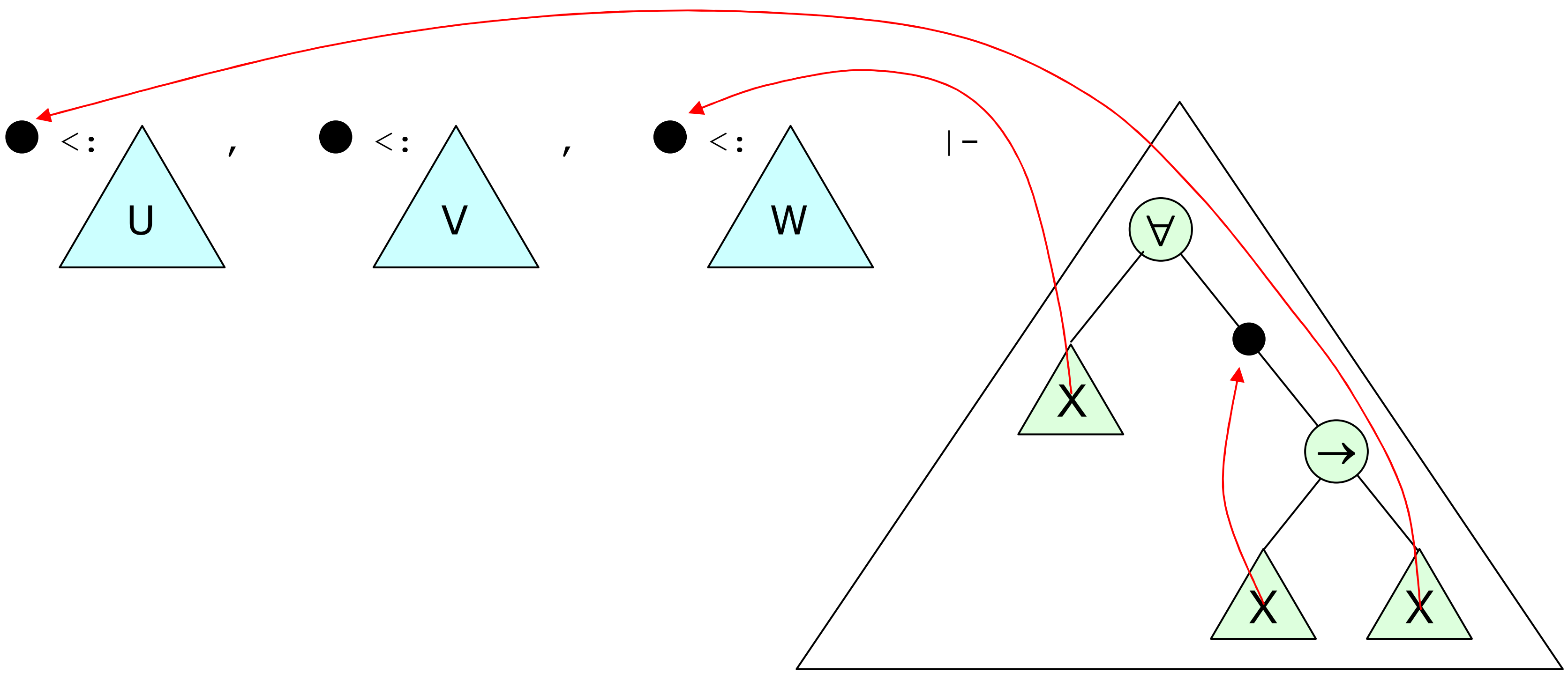
this is just to show that pointers can go arbitrary far back up in the tree

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



Environments and free variables

X <: U, Y <: V, Z <: W |- ∀ P <: Z. (P -> X)

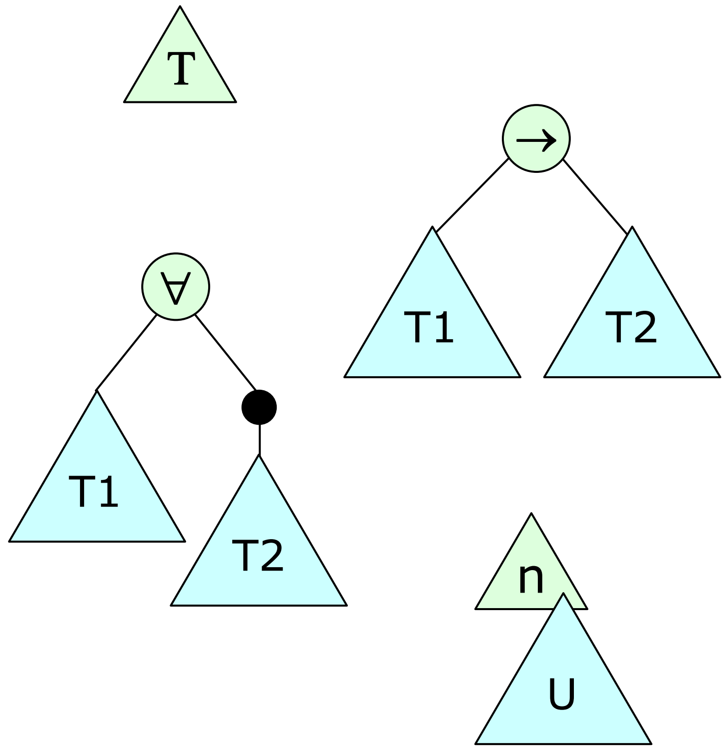


2) Formal definitions

a) Types and environments

Definition of types

```
Inductive typ :=
  | top   : typ
  | arrow : typ -> typ -> typ
  | all   : typ -> typ -> typ
  | var   : nat -> typ -> typ
```



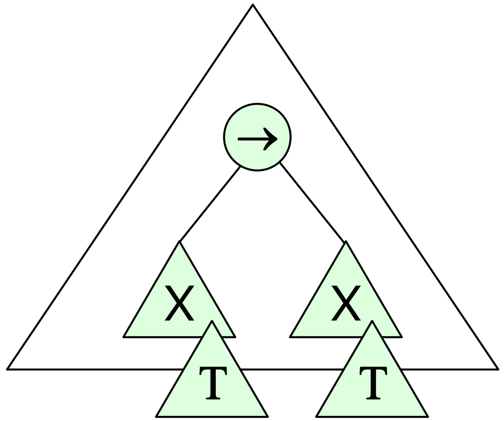
De Bruijn index of the variable

type to which the variable is mapped to, irrelevant if the variable is not free

Example using labels

Polymorphic identity:

```
X <: Top  |-  X -> X
```



to be written with labels as:

```
X <: Top  |-  X^Top -> X^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

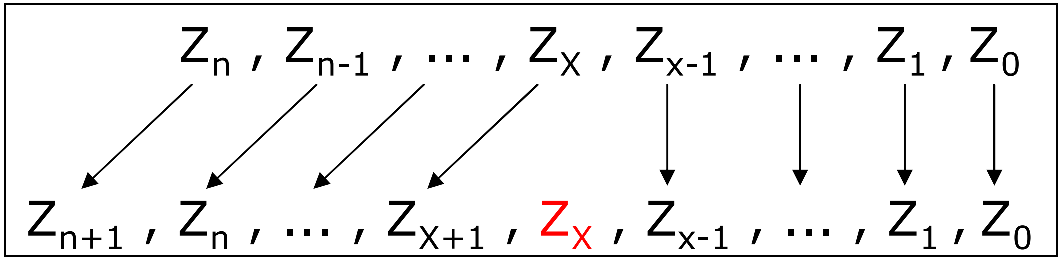
we don't need to give an implementation for type env, since labels on free variables carry all the information that we may need to use

```
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



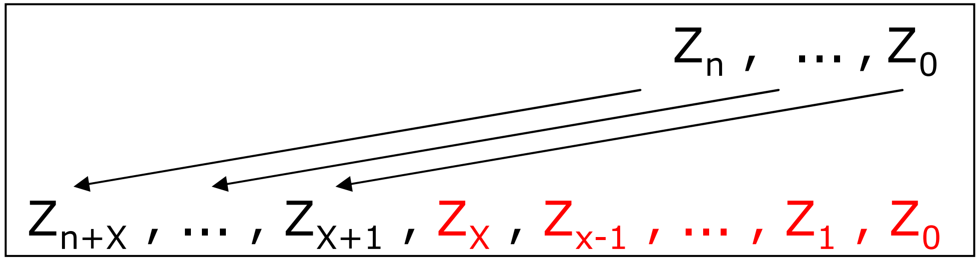
insert a binding at position X in the implicit environment

```
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)
```

cross a binder

shift the index in case X ≤ Y

Definition of weaken



weaken introduces variables at end of the environment

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

note that "weaken X" introduces X+1 variables;
this helps simplify some statements and proofs

Definition of update

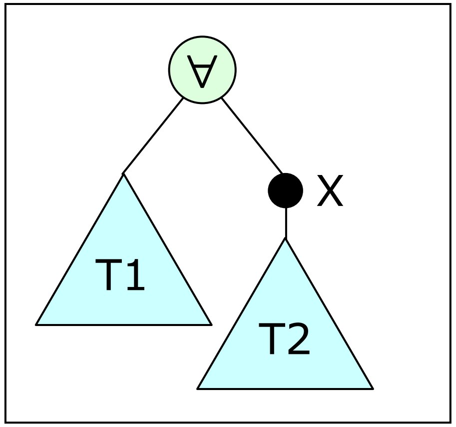
"update X U T" puts a label U on all occurences of X in T

```
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



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

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

```
Definition push := update 0.
```

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"

```
Inductive wf : env -> typ -> Prop :=

| wf E top

| wf E T1 -> wf E T2 -> wf E (arrow T1 T2)

| wf E T1 -> (∀ U : typ, wf (env_push E U) (push U T2))

    -> wf E (all T1 T2)

| env_has E X T1 -> wf E T1 -> 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

```
Inductive wf_env : env -> Prop :=
| wf_env env_empty
| wf_env E -> wf E U -> wf_env (env_push E U)
```

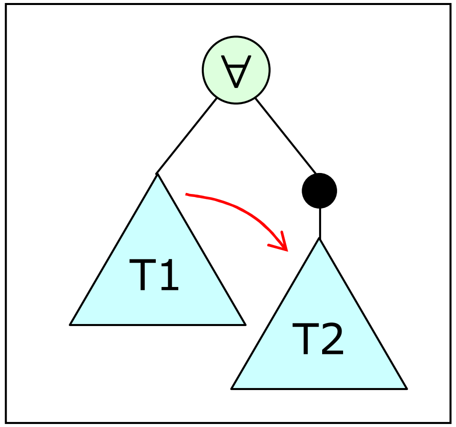
2) Proving results

a) Properties of the operations

Crossing push with insert and update

insert_on_push :

```
  insert (S X) (push T1 T2)
= push (insert X T1) (insert (S X) T2)
```



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

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 :

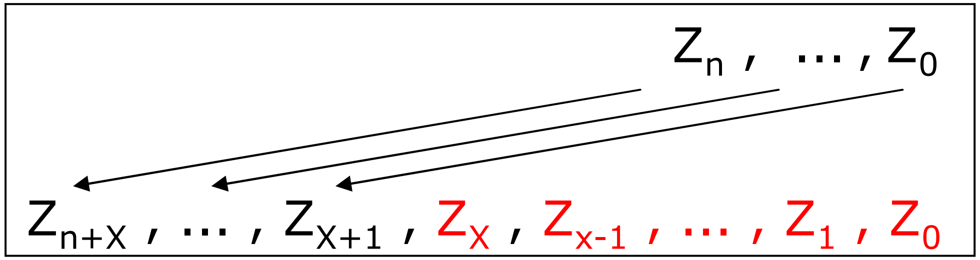
An equivalent result for update

```
  update (S X) P (push T1 T2)
= push (update X P T1) (update (S X) P T2)
```

Crossing update at weaken

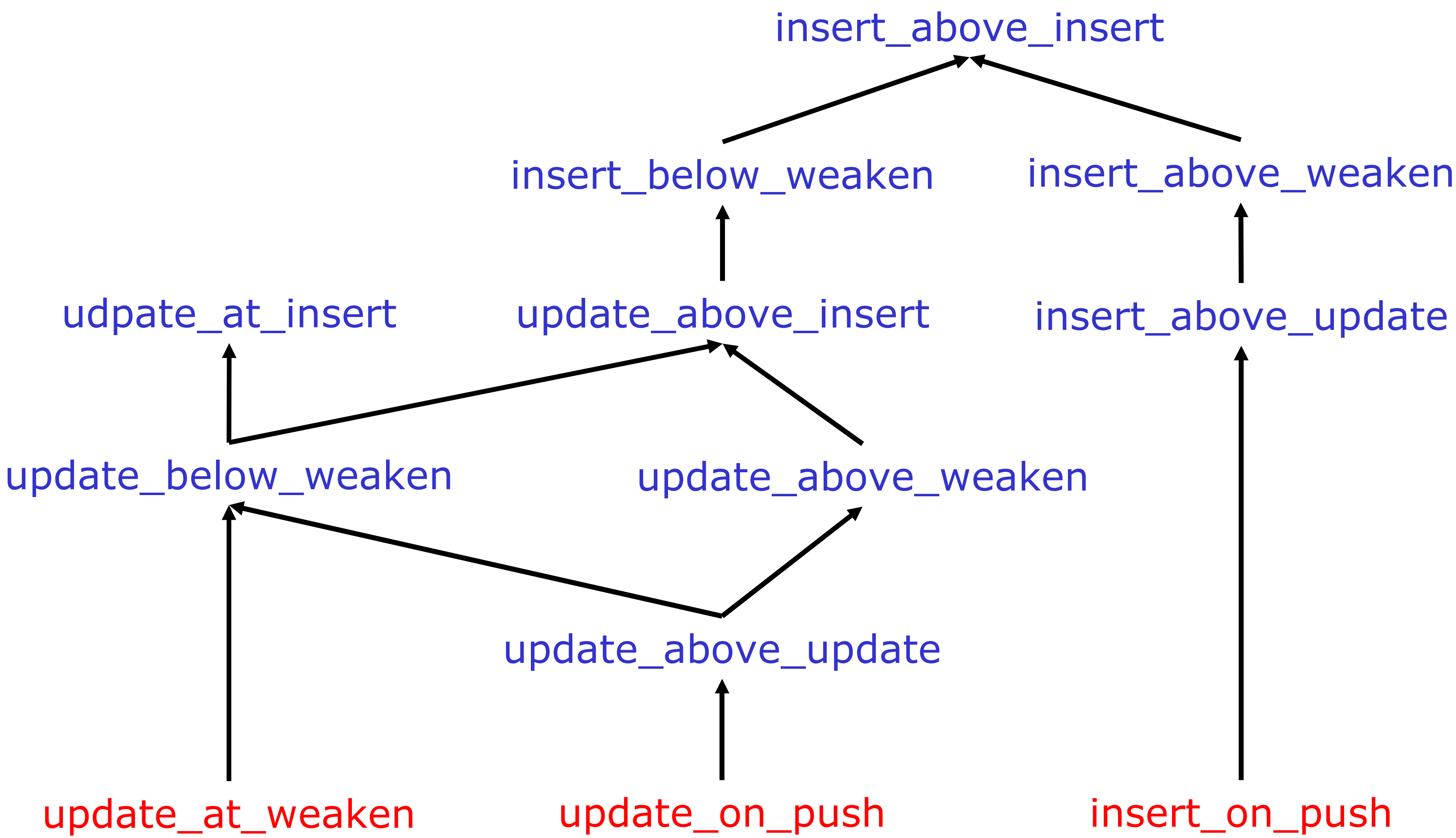
```
update_at_weaken :

    update X U (weaken X T)
  = weaken X T
```

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



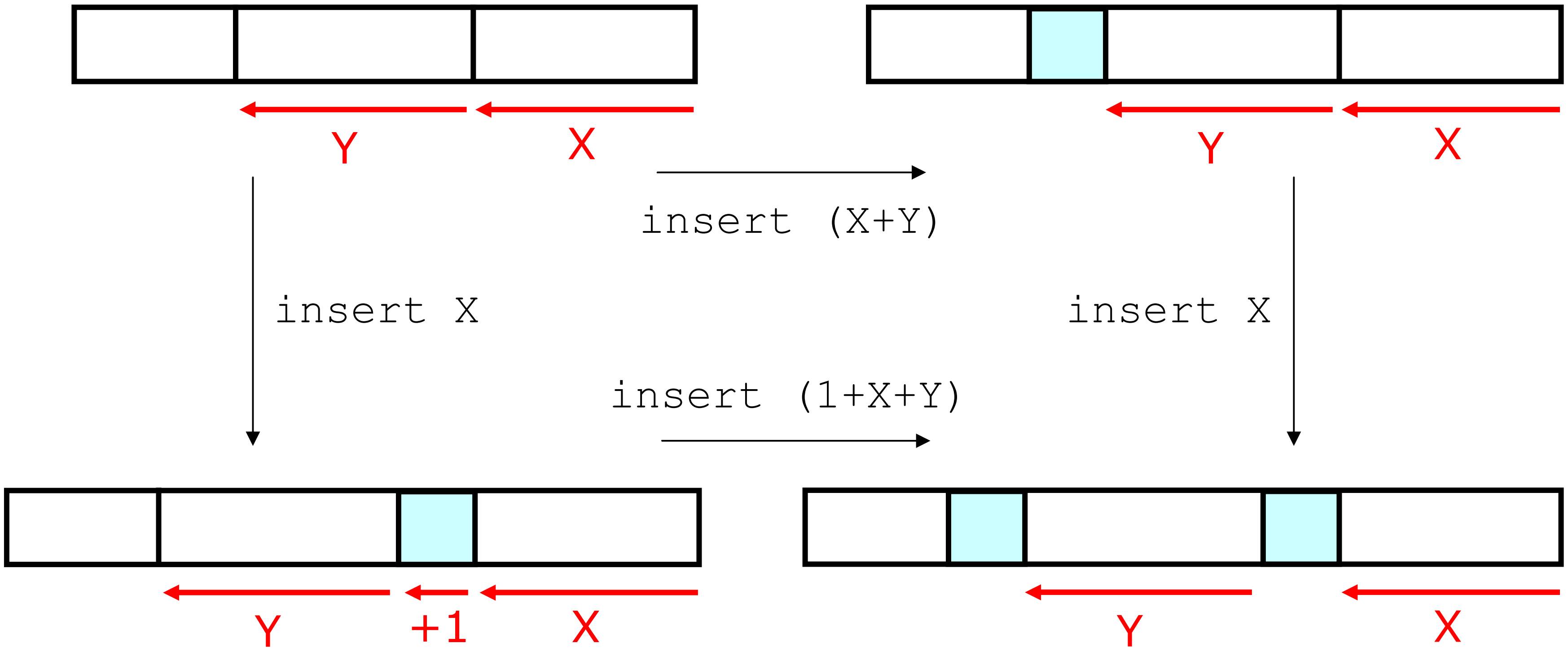
we use this lemma to capture the fact that if we have an environment of the form "Γ1 , X <: T, Γ2" then X has no occurence in T (we need that to prove narrowing)

Proof graph for the crossing lemmas



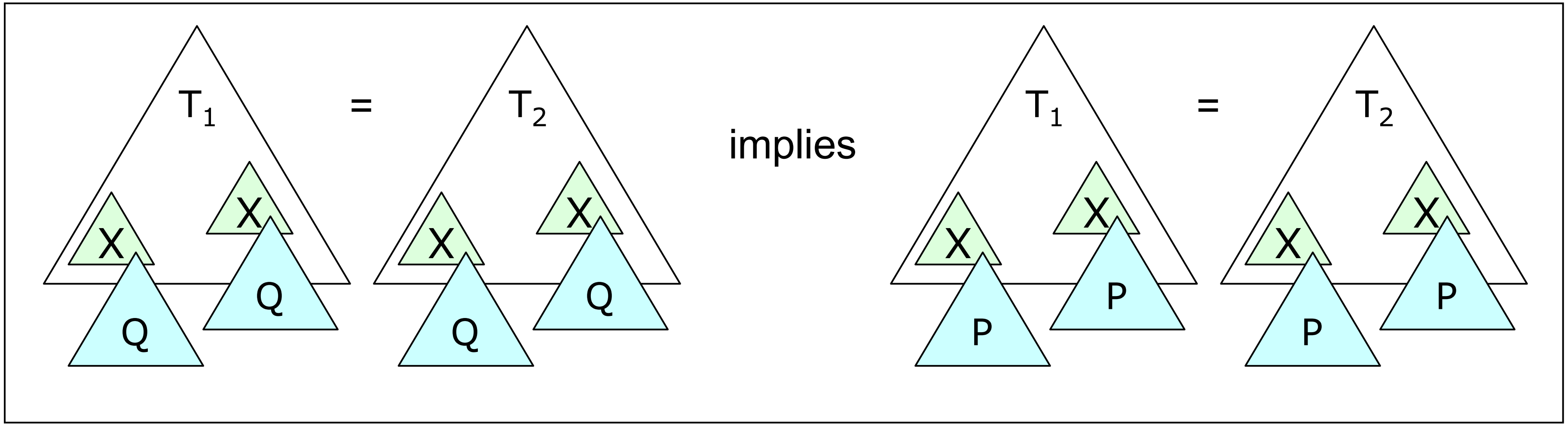
Example of a crossing lemma

```
Lemma insert_above_insert :

    insert (S(X+Y)) (insert X T)
  = insert X (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 udpate the label of each occurence of X in Γ2.

b) Properties of unsafe subtyping

Statements of properties about unsafe subtyping

insert_preserves_sub :

```
T1 <¤ T2  ->  (insert X T1) <¤ (insert X T2)
```

weaken_preserves_sub :

```
T1 <¤ T2  ->  (weaken X T1) <¤ (weaken X T2)
```

sub_reflexivity :

```
T <¤ T
```

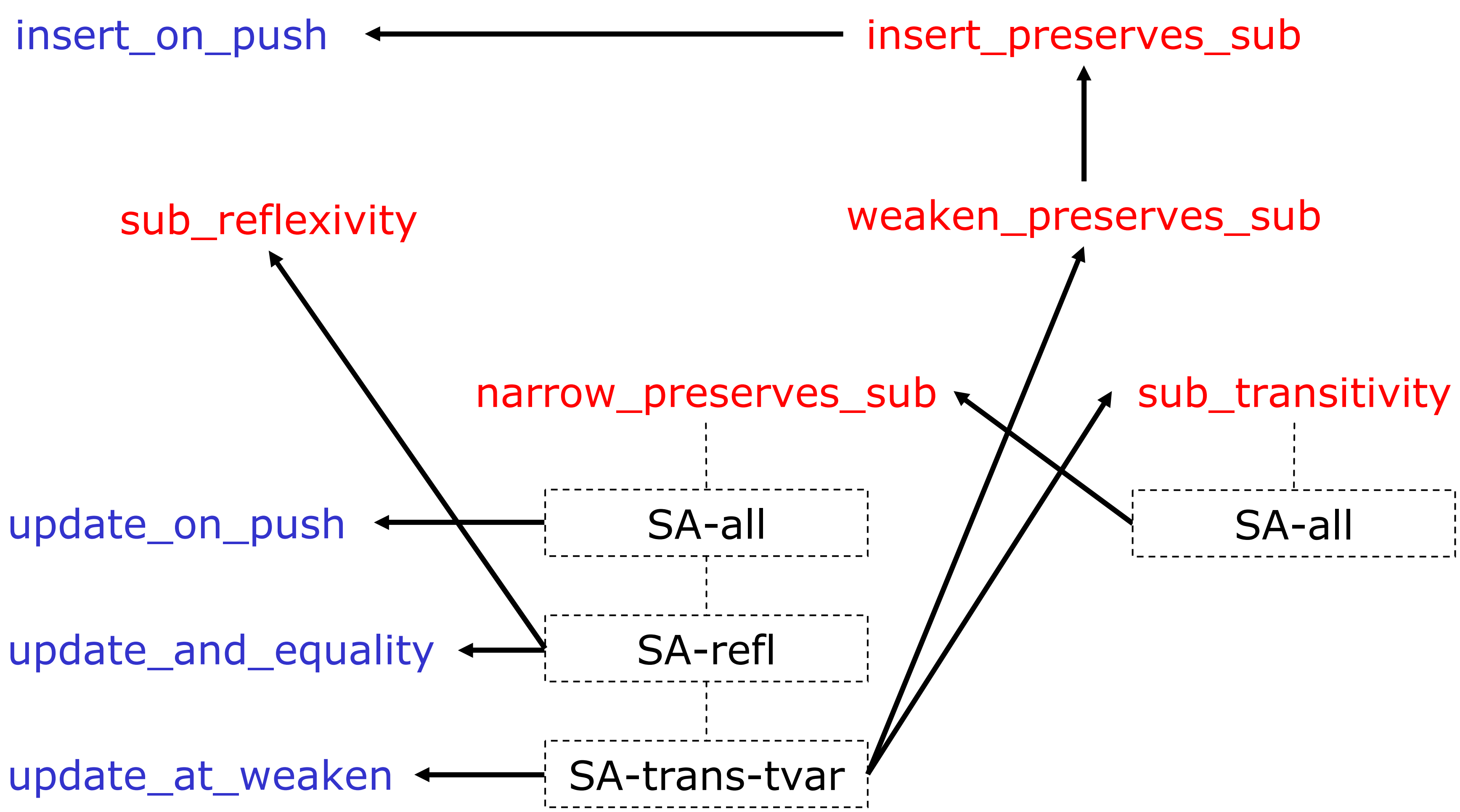
narrowing_preserves_sub :

```
(update X Q S) <¤ (update X Q T)  ->  P <¤ Q  ->
(update X P S) <¤ (update X P T)
```

sub_transitivity :

```
S <¤ Q  ->  Q <¤ T  ->  S <¤ T
```

Proof graph for results about unsafe subtyping



3) Structure of the solution

Structure of the solution (not including tactics)