# Un interpréteur abstrait pour la règle des signes (1er essai)

```
type signe = POS | NEG | TOP;;
type signenv = var -> signe;;
let rec expr_sign rho e = match e with
      V x -> rho x
    | C c -> (match c with
               Z n -> if n>=0 then POS else NEG (*1*)
             | _ -> TOP)                        (*2*)
    | ...
    | DIV (e1, e2) ->
      (match (expr_sign rho e1, expr_sign rho e2) with
         (POS, POS) -> POS                      (*3*)
       | ...)
    | EGAL (e1, e2) -> TOP
    | NON e -> TOP;;
```

```
let rec com_sign rho c = match c with
      SET (x, e) ->
        (let v = expr_sign rho e
         in fun y -> if y=x then v else rho y)
    | INPUT x -> (fun y -> if y=x then TOP else rho y)
    | IF (e, c1, c2) ->
      signenv_sup (com_sign rho c1) (com_sign rho c2) (*4*)
    | WHILE (e, c) -> signenv_sup rho
               (com_sign rho (SEQ (c, WHILE (e, c)))) (*5*)
    | SEQ (c1, c2) -> com_sign (com_sign rho c1) c2
    | SKIP -> rho;;
```

# Observations

1. `(*1*)` : 0 est $+$ (`POS`), pas $-$ (`NEG`) : c'est un choix arbitraire, qui nous interdira de conclure que $0 + (-1) \leq 0$ par exemple (perte d'information absurde).

2. `(*2*)` : prendre le signe d'une chaîne ou d'un booléen n'a aucun sens; retourner `TOP` (="on ne sait pas") est-il correct ?

3. `(*3*)` : 1 / $0 \hat{=}$ Err est abstrait en $+/+ \hat{=} +$; est-ce correct ?

4. `(*4*)` : on ne peut pas savoir quelle branche du test va être executée : on est obligé de calculer les signes sur les deux branches, et de combiner les résultats sur les deux branches (fonction `signenv_sup`, à définir).

Un point un peu plus subtil. **Ex:** `if x=0 then skip else x:=3` ne retourne pas que `x` est $+$, pourquoi ?

5. `(*5*)` : dans la boucle while, où est passé le test d'arrêt ? En fait, tel qu'il est, l'analyseur boucle dès qu'il tombe sur une boucle while...

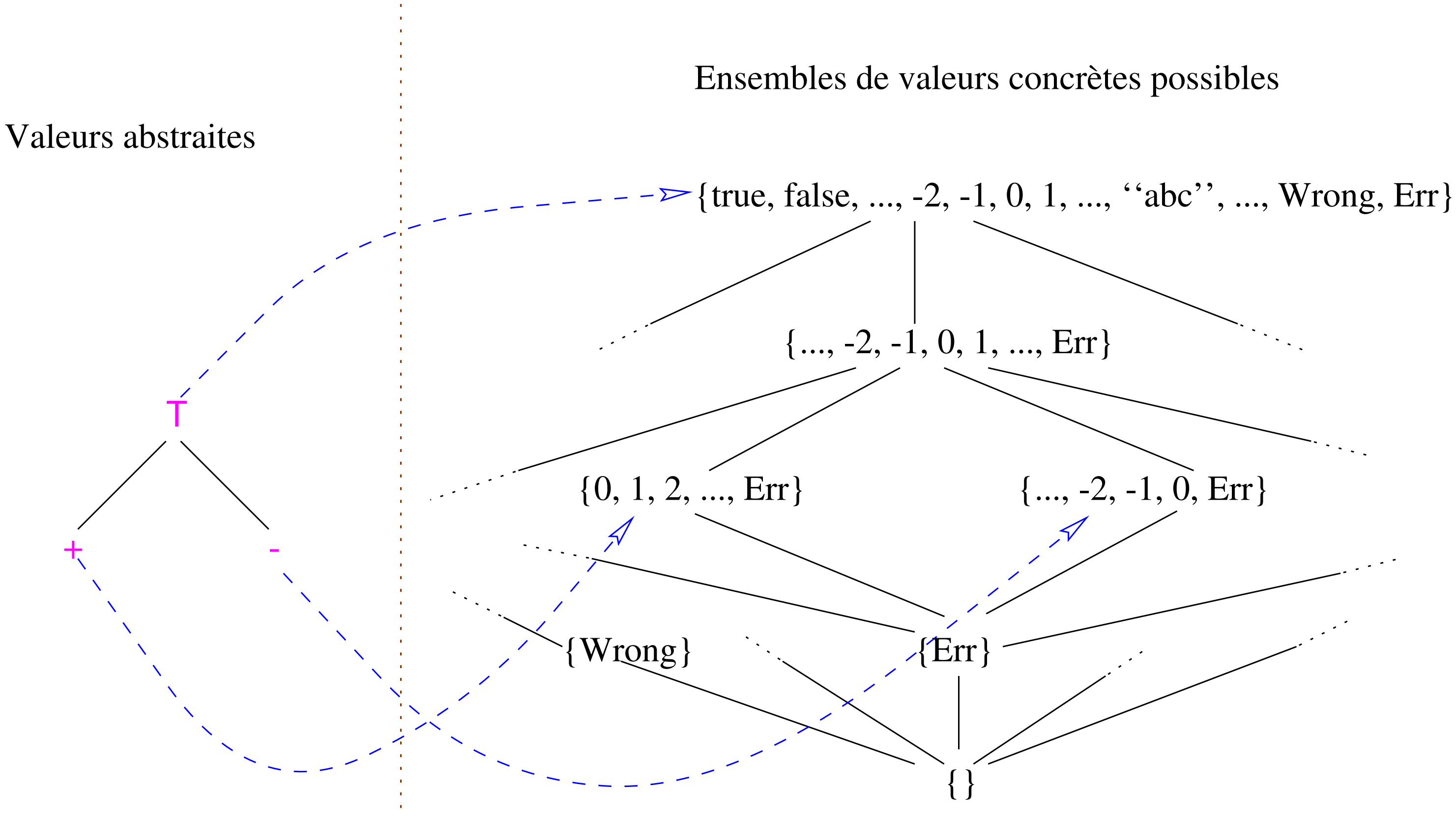
6. Finalement, l'analyseur ne donne des informations que sur les environnements à la fin du programme (s'il termine), pas sur les points de programmes intermédiaires.

Pour les points 2 et 3 notamment, on aura besoin de sémantiques. Pour le point 6, on aura besoin d'une sémantique concrète opérationnelle à petit pas.

# Approximations et information

- Ordres d’information;
- Concrétisation, monotonie;
- Treillis;
- Correspondances de Galois.

# Ordres d'information



# Fonction de concrétisation

$$
\begin{array}{rcll}
L^\flat & \hat{=} & \mathbb{P}(\mathcal{V}al \cup \{\mathsf{Err}, \mathsf{Wrong}\}) & \text{ensembles de valeurs concrètes} \\
& & & = \text{propriétés portant sur des valeurs concrètes} \\
L^\sharp & \hat{=} & \{+, -, \top\} & \text{valeurs abstraites} \\
& & & \text{(représentent donc certaines propriétés choisies)} \\
\gamma & : & L^\sharp \to L^\flat & \text{concrétisation} \\
& & & \text{(“sémantique des valeurs abstraites”)} \\
\gamma(\top) & \hat{=} & \mathcal{V}al \cup \{\mathsf{Err}, \mathsf{Wrong}\} &
\end{array}
$$

$$
\gamma(+) \hat{=} \{0, 1, 2, \ldots, \mathsf{Err}\} \qquad \gamma(-) \hat{=} \{\ldots, -2, -1, 0, \mathsf{Err}\}
$$

Correction : On veut que, par exemple, si l'interprète abstrait retourne $\mathtt{x} \mapsto +$, alors dans toute exécution concrète, $\rho(\mathtt{x}) \in \gamma(+)$.

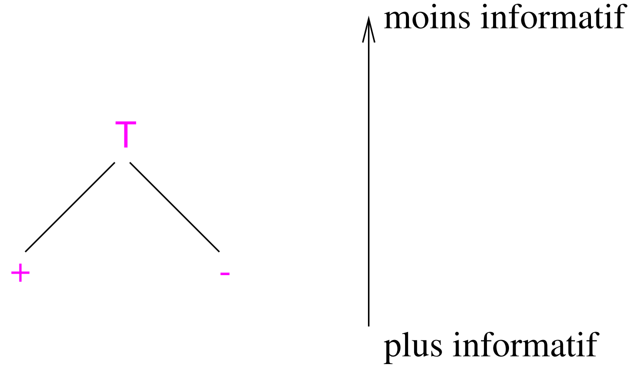
# Monotonie

L'ordre $\subseteq$ sur $L^\flat$ (= l'implication $\Rightarrow$ sur les propriétés de valeurs) est "est plus informatif que" : si $A \subseteq B$, alors $v \in A$ est plus informatif que $v \in B$.

L'ordre $\subseteq$ induit un ordre $\sqsubseteq^\sharp$ sur $L^\sharp$ :

$$+ \sqsubseteq^\sharp \top \qquad - \sqsubseteq^\sharp \top \qquad + \text{ et } - \text{ incomparables}$$

$\ldots$ maximal tel que $\gamma$ soit monotone : $v_1 \sqsubseteq^\sharp v_2 \Rightarrow \gamma(v_1) \subseteq \gamma(v_2)$.



## Problème `(*4*)` et sup-demi-treillis complets

Slogan : on veut pouvoir calculer la valeur abstraite la plus informative possible (dans l'ensemble $L^\sharp$ fixé).

Cas d'un `if` : si $x \mapsto v_1$ sur la première branche, et $x \mapsto v_2$ sur la seconde, le résultat est $x \mapsto v_1 \sqcup v_2$. On demande donc :

$$\text{Pour tous } v_1, v_2 \in L^\sharp, \text{ il existe } v_1 \sqcup v_2 \hat{=} \min_{\sqsubseteq^\sharp}\{v \mid v_1 \sqsubseteq^\sharp v \wedge v_2 \sqsubseteq^\sharp v\}$$

(le sup binaire).

Généralisation au `while` : si $x \mapsto v_i$ lorsque la boucle termine après $i$ tours, le résultat est $x \mapsto \bigsqcup_{i \in I} v_i$. On demande :

Pour toute famille $(v_i)_{i \in I}$, il existe

$$\bigsqcup_{i \in I} v_i \hat{=} \min^\sharp_{\sqsubseteq}\{v \in L^\sharp \mid \forall i \in I \cdot v_i \sqsubseteq^\sharp v\} \text{ (le sup).}$$

Quelques rappels :

Un *ordre* $\sqsubseteq$ est une relation binaire *réflexive* ($x \sqsubseteq x$), *antisymétrique* (si $x \sqsubseteq y$ et $y \sqsubseteq x$, alors $x = y$) et *transitive* (si $x \sqsubseteq y$ et $y \sqsubseteq z$ alors $x \sqsubseteq z$).

Un *majorant* $v$ de $F \subseteq L$ est un élément de $L$ tel que $x \sqsubseteq v$ pour tout $x \in F$.
Le *supremum* $\bigsqcup F$ de $F$ est le plus petit de tous les majorants de $F$, $\min_{\sqsubseteq}\{v \mid v \text{ majore } F\}$ (s'il existe) :

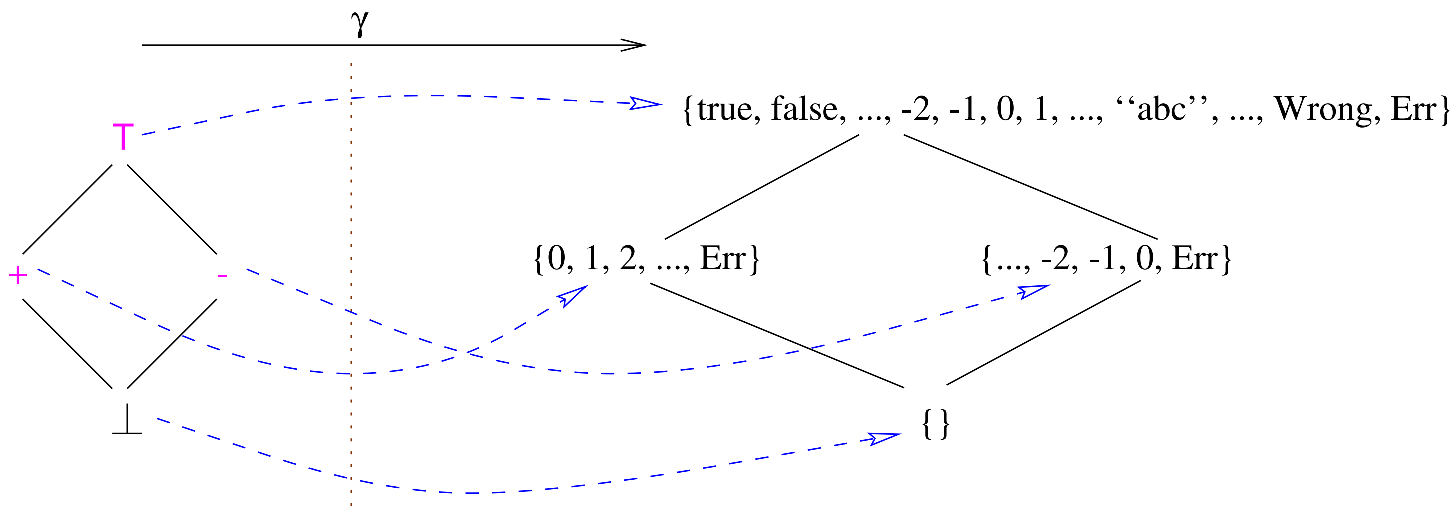
$$\begin{array}{ll} \forall x \in F \cdot x \sqsubseteq \bigsqcup F & \text{(majorant)} \\ \forall y \cdot (\forall x \in F \cdot x \sqsubseteq y) \Rightarrow \bigsqcup F \sqsubseteq y & \text{(min)} \end{array}$$

# L'élément $\perp$

**Corollaire:** il existe $\perp \hat{=} \bigsqcup_{i\in\emptyset} v_i$, l'élément minimal de $L^\sharp$.

On obtient par exemple $x \mapsto \perp$ à la sortie d'une boucle `while` qui ne termine pas.

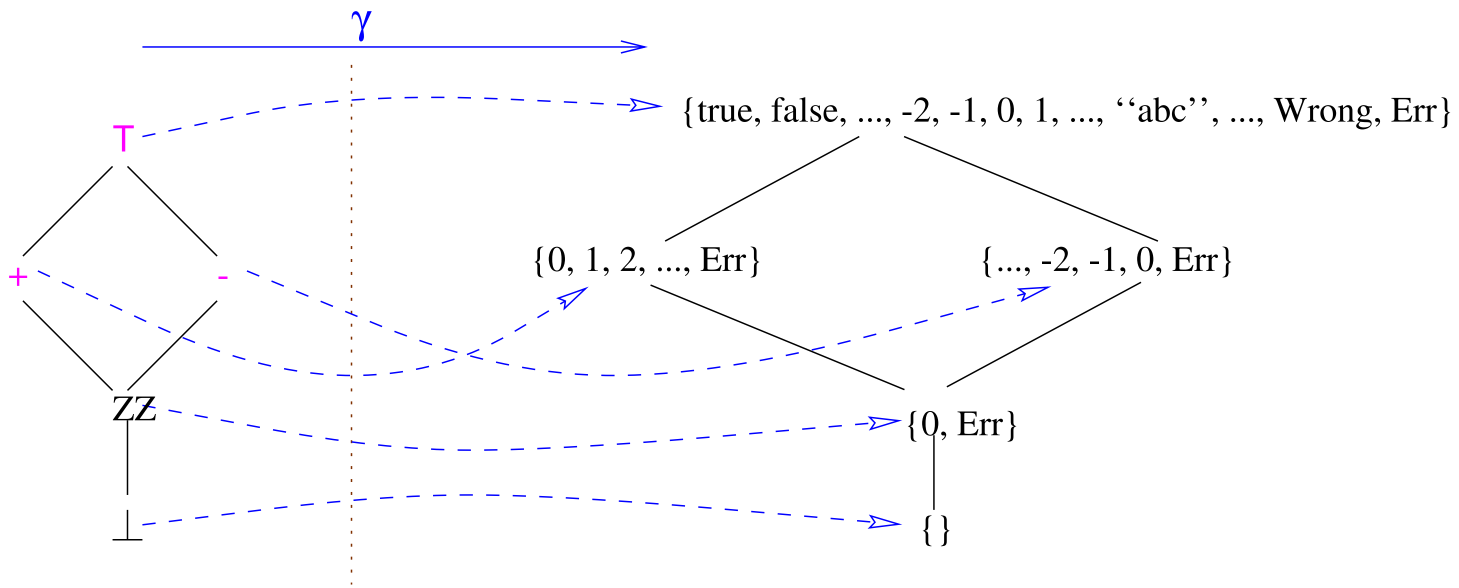
On a $\gamma(\perp) \hat{=} \emptyset$ : si $x \mapsto \perp$ à un point de programme donné, c'est qu'on ne peut pas atteindre ce point de programme ! (code mort)



## Problème `(*1*)`, signe de $0$

On voudrait attribuer à 0 une information ZZ qui soit $\sqsubseteq^\sharp +$ et $\sqsubseteq^\sharp -$ mais toujours correcte : $0, \mathsf{Err} \in \gamma(\mathrm{ZZ})$ (“le calcul retourne 0, ou une erreur”).

Appliquons le slogan : $\gamma(\mathrm{ZZ}) = \min_\subseteq \{A \in \mathbb{P}(\mathcal{V}al \cup \{\mathsf{Err}, \mathsf{Wrong}\}) \mid 0 \in A, \mathsf{Err} \in A, A \subseteq \gamma(+), A \subseteq \gamma(-)\}$.

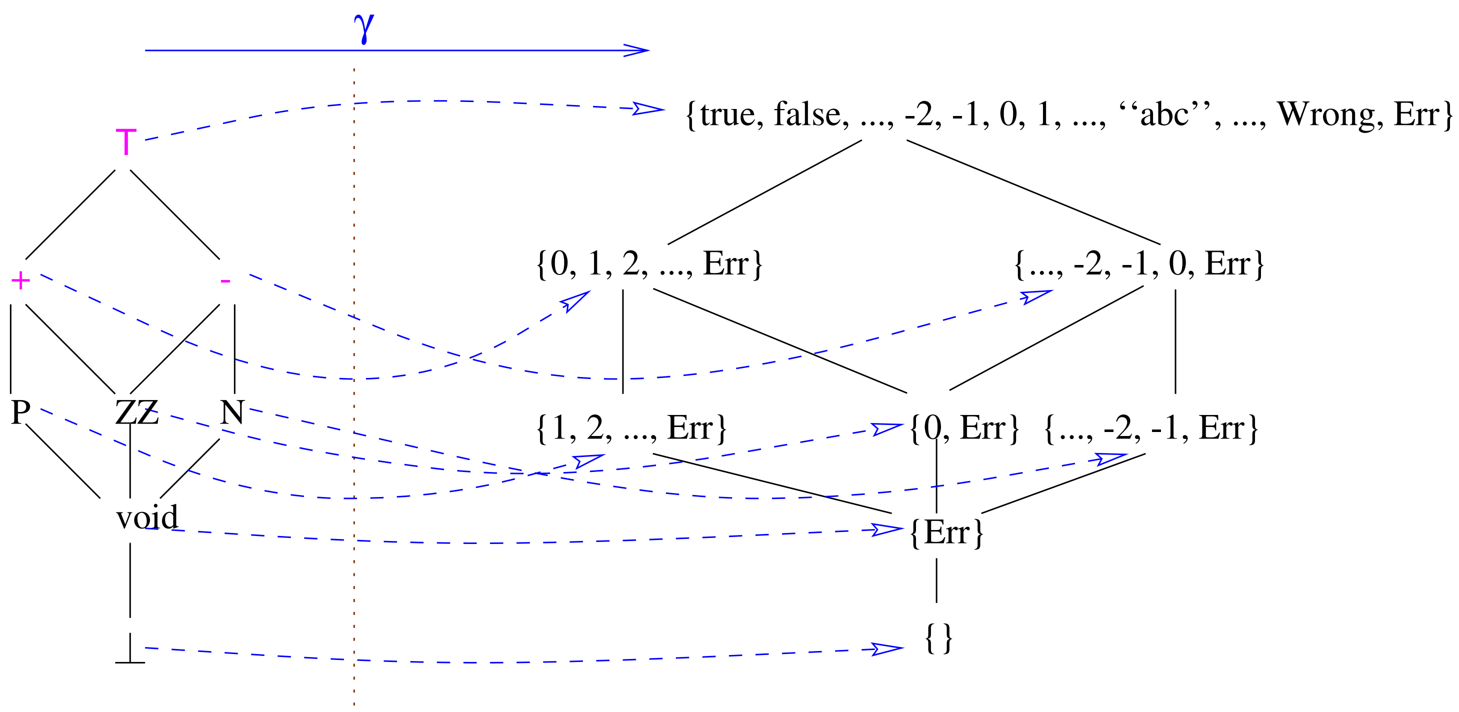


# On peut faire mieux . . .

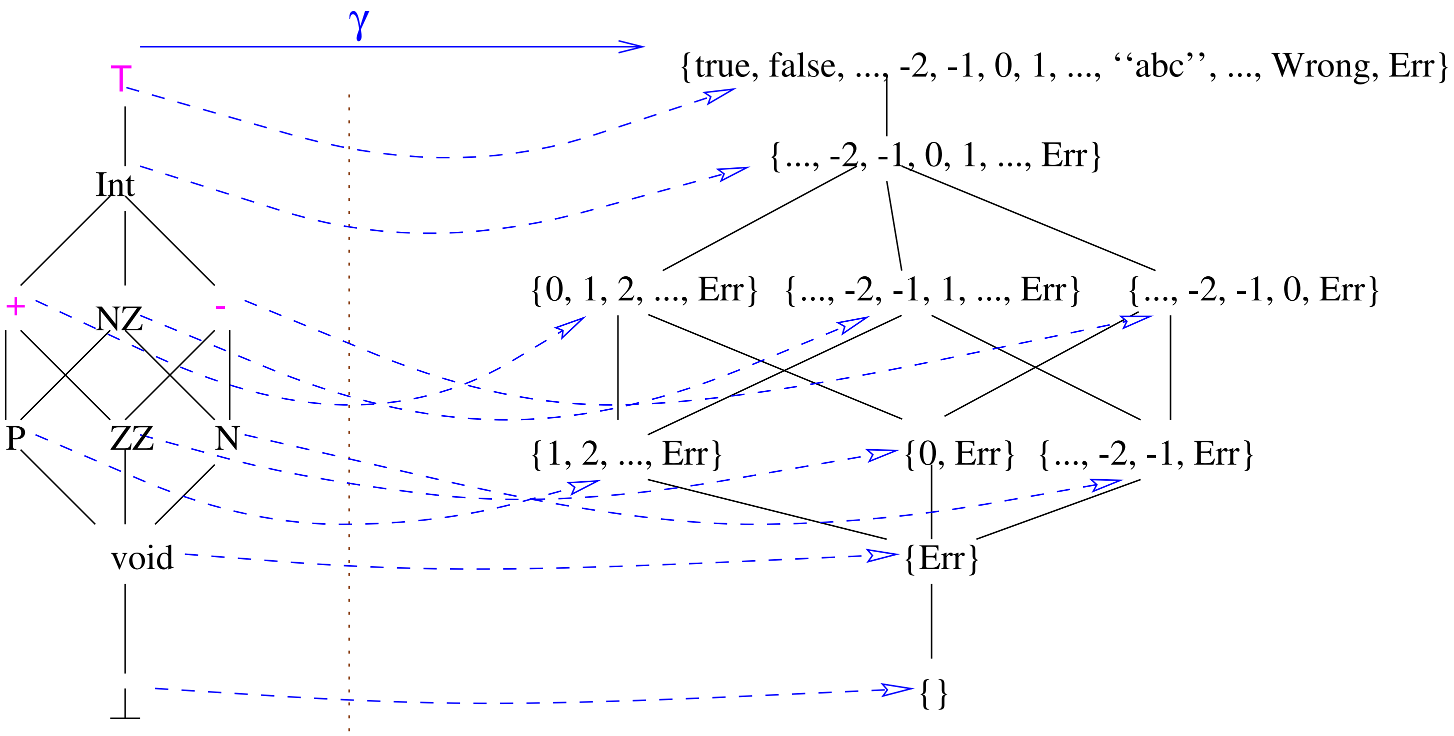
P ("$> 0$") : $\gamma(\mathrm{P}) \hat{=} \{1, 2, \ldots, \mathsf{Err}\}$;

N ("$< 0$") : $\gamma(\mathrm{N}) \hat{=} \{\ldots, -2, -1, \mathsf{Err}\}$;

void ("rien") : $\gamma(\mathrm{void}) \hat{=} \{\mathsf{Err}\}$.



# …et encore mieux …



## …et toujours mieux

On peut toujours améliorer la précision de l'analyse
… jusqu'à obtenir $L^\sharp$ isomorphe à $L^\flat = \mathbb{P}(\mathcal{V}al \cup \{\mathsf{Err}, \mathsf{Wrong}\})$.

Mais les éléments de $L^\flat$ ne sont pas représentables sur ordinateur, et les opérations ne sont pas calculables.

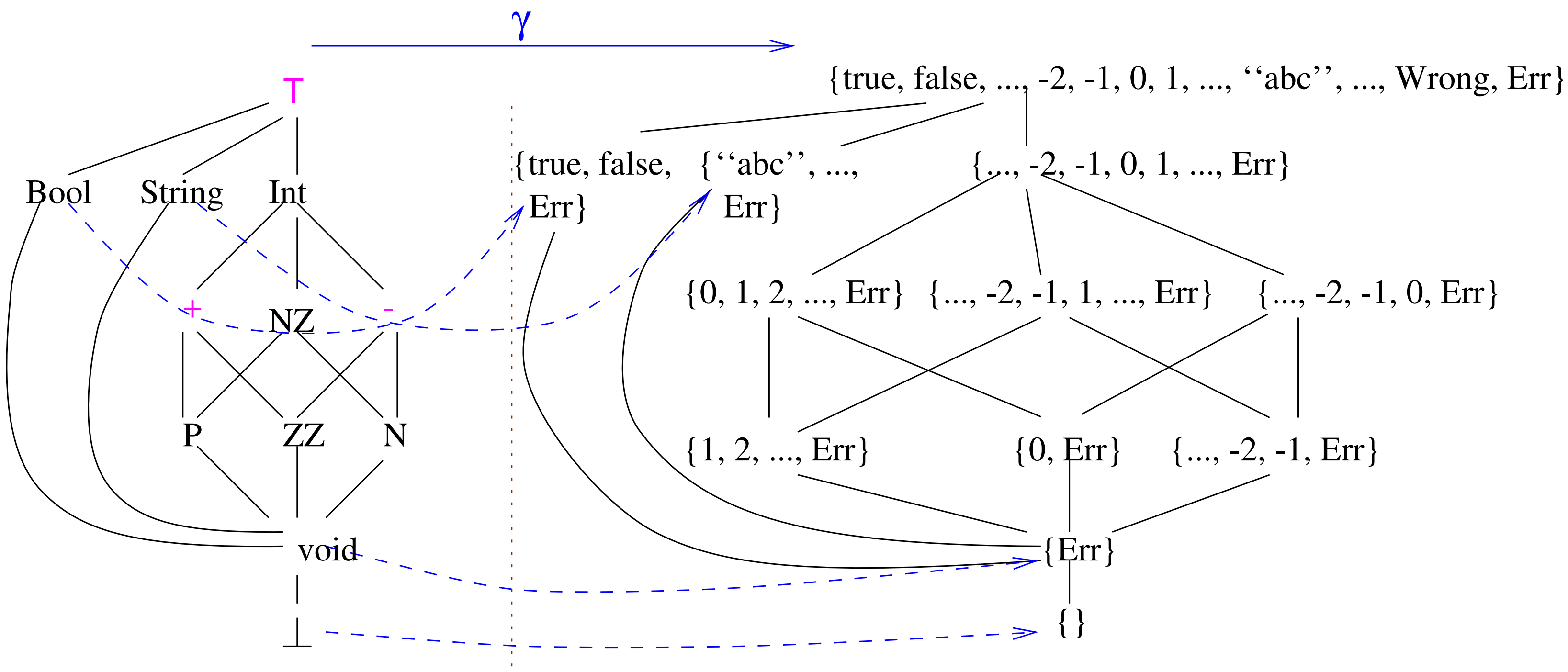
Il faut donc s'arrêter quelque part !

## Problème `(*2*)` : le signe des chaînes

On applique le slogan : le signe abstrait de “abc” est $\min_{\sqsubseteq^\sharp}\{v \in L^\sharp \mid \text{“abc”} \in \gamma(v)\}$ (si ce min existe).

Dans les demi-treillis considérés jusqu’ici, c’est $\top$.

**Ex:** dans le demi-treillis suivant, que serait le signe abstrait de “abc” ?




(Il s’agit en fait de typage.)