Applications, produit réduit

- Lecture de variables non initialisées;
- Comment détecter à la fois les variables non initialisées et les types;
- Produit réduit;

Variables non initialisées

On souhaite détecter les lectures de variables non initialisées.

```
int f (int i)
{
  int j, k, m;

  k = 2*i+j+1;
  m = k+3;
  j = k+m;
  return j;
}
```

Avec `gcc -c -O -Wall` :

```
a.c: In function `f':
a.c:4: warning: `j' might be used uninitialized
       in this function
```

Sémantique concrète modifiée

Idée : on ajoute une valeur d'erreur Err_{init}; les variables initiales des variables sont toutes Err_{init}; lire une variable de valeur Err_{init} provoque une erreur.

Comme au transparent 21 :

$$
\begin{array}{rcll}
v, \ldots \in \mathcal{V}al & \hat{=} & \mathbb{B} \mid \mathbb{Z} \mid \mathcal{S}tring & \text{valeurs} \\
\rho, \ldots \in \mathcal{E}nv & \hat{=} & \mathcal{V}ar \to (\mathcal{V}al \cup \{\mathsf{Err}_{init}\}) & \text{environnements}
\end{array}
$$

$$
\begin{array}{rcll}
\mathcal{E}[\![\_]\!]\_ & : & \mathcal{E}xpr \times \mathcal{E}nv \to (\mathcal{V}al \cup \{\mathsf{Err}, \mathsf{Wrong}, \mathsf{Err}_{init}\}) & \\
\mathcal{E}[\![x]\!]\rho & \hat{=} & \rho(x) & \text{variables} \\
\mathcal{E}[\![a]\!]\rho & \hat{=} & a & \text{constantes} \\
 & & \ldots &
\end{array}
$$

Tables de sémantique concrète :

<table>
<tr><th>+</th><th>Err_{init}</th><th>Err</th><th>Wrong</th><th>autres valeurs</th></tr>
<tr><td>Err_{init}</td><td>Err_{init}</td><td>Err</td><td>Wrong</td><td>Err_{init}</td></tr>
<tr><td>Err</td><td>Err_{init}</td><td>Err</td><td>Wrong</td><td>Err</td></tr>
<tr><td>Wrong</td><td>Err_{init}</td><td>Err</td><td>Wrong</td><td>Wrong</td></tr>
<tr><td>autres valeurs</td><td>Err_{init}</td><td colspan="3">cf. transparent 51</td></tr>
</table>

–	Err_{init}	Err	Wrong	autres
	Err_{init}	Err	Wrong	cf. transparent 51

etc., de sorte que $\mathcal{E}[\![e]\!]\rho = \mathsf{Err}_{init}$ si et seulement si l'évaluation de e provoque la lecture d'une variable non initialisée.

Treillis abstrait

$$(L^\sharp_{init}, \sqsubseteq^\sharp_{init}) \hat{=} \begin{array}{c}\overline{\mathsf{init}}\\ |\\ \mathsf{init}\end{array}$$

$$\begin{array}{rcl}\gamma_{init}(\overline{\mathsf{init}}) & \hat{=} & \mathcal{V}al \cup \{\mathsf{Err}, \mathsf{Wrong}, \mathsf{Err}_{init}\}\\ \gamma_{init}(\mathsf{init}) & \hat{=} & \mathcal{V}al \cup \{\mathsf{Err}, \mathsf{Wrong}\}\end{array}$$

$\overline{\mathsf{init}}$ = "possible erreur d'initialisation";

init = "pas d'erreur d'initialisation".

$\alpha_{init}(A) \hat{=} \overline{\mathsf{init}}$ si $\mathsf{Err}_{init} \in A$, init sinon.

Sémantique abstraite

$$\begin{array}{rcl}
\mathcal{E}[\![x]\!]^{\sharp}_{init}\rho^{\sharp} & \hat{=} & \rho^{\sharp}(x) \\
\mathcal{E}[\![a]\!]^{\sharp}_{init}\rho^{\sharp} & \hat{=} & \mathsf{init} \\
\mathcal{E}[\![e_1 \,\mathbf{+}\, e_2]\!]^{\sharp}_{init}\rho^{\sharp} & \hat{=} & \mathcal{E}[\![e_1]\!]^{\sharp}_{init}\rho^{\sharp} \sqcup^{\sharp} \mathcal{E}[\![e_2]\!]^{\sharp}_{init}\rho^{\sharp}
\end{array}$$

(idem pour toutes les opérations binaires...)

$$\mathcal{E}[\![\mathbf{-}\, e]\!]^{\sharp}_{init}\rho^{\sharp} \quad \hat{=} \quad \mathcal{E}[\![e]\!]^{\sharp}_{init}\rho^{\sharp}$$

(idem pour toutes les opérations unaires...)

Équations sémantiques abstraites

On cherche
$cond^\sharp_{init}[\![instr]\!]\rho^\sharp_1 \sqsupseteq^*_{init} \alpha\{\rho \mid \exists\rho_1 \in \gamma_{init,NR}(\rho^\sharp_1) \cdot cond[\![instr]\!](\rho_1, \rho)\}$
(correction).

$$\begin{array}{rcl}
cond^\sharp_{init}[\![x := e]\!]\rho^\sharp_1 & \hat{=} & \rho^\sharp_1[x \leftarrow \mathcal{E}[\![e]\!]^\sharp_{init}\rho^\sharp_1] \\
cond^\sharp_{init}[\![\texttt{input}\, x]\!]\rho^\sharp_1 & \hat{=} & \rho^\sharp_1[x \leftarrow \mathsf{init}] \quad \text{(et non } x \leftarrow \top_{init} \hat{=} \overline{\mathsf{init}}\text{)} \\
cond^\sharp_{init}[\![e \text{ vrai?}]\!]^\sharp_{init}\rho^\sharp_1 & \hat{=} & \rho^\sharp_1 \\
cond^\sharp_{init}[\![e \text{ faux?}]\!]^\sharp_{init}\rho^\sharp_1 & \hat{=} & \rho^\sharp_1 \\
cond^\sharp_{init}[\![\texttt{skip}]\!]^\sharp_{init}\rho^\sharp_1 & \hat{=} & \rho^\sharp_1
\end{array}$$

Note : on commence alors l'analyse à $E^\sharp_0 \hat{=} \lambda p \cdot \begin{cases} \lambda x \cdot \overline{\mathsf{init}} & \text{si } p = I \\ \lambda x \cdot \mathsf{init} & \text{sinon} \end{cases}$

Quelques remarques

- $\bot_{init} \hat{=} \mathsf{init}$ ne dénote pas du code mort comme avant ($\gamma_{init}(\bot_{init}) \neq \emptyset$);
- on pourrait remplacer l'erreur Err_{init} ("une variable utilisée est non initialisée") par $\mathsf{Err}_{init}(x)$ ("la variable "x" est non initialisée), ce qui donnerait des messages d'erreur plus précis; vérifier que le treillis abstrait serait alors isomorphe à $\mathbb{P}(\mathcal{V}ar)$.
- on pourrait éviter les erreurs en cascade : dans `y:=x; z:=y`, `y` et `z` sont trouvées non initialisées. Pour ceci, changer la sémantique concrète en (par ex.) :

$$cond[\![x := e]\!](\rho_1, \rho_2) \quad \hat{=} \quad \begin{cases} \rho_2 = \rho_1[x \leftarrow \mathcal{E}[\![e]\!]\rho_1] & \text{si } \mathcal{E}[\![e]\!]\rho_1 \neq \mathsf{Err}_{init} \\ \rho_2 = \rho_1[x \leftarrow \mathsf{Err}] & \text{sinon} \end{cases}$$

Comment combiner non-initialisations et types ?

- Faire deux analyseurs, un pour les non-initialisations, un pour les types : pourquoi pas, mais on perd de l'information, en général.

  ```
  b:=false;
  if b
    then x:=y /*!*/
  else skip
  ```

  $\Rightarrow$ l'analyseur se plaindra que y est possiblement non initialisée dans `x:=y`, alors que ce code est mort.

- ou calculer dans le treillis produit.

Treillis produit

Étant donnés deux treillis complets $(L_1, \sqsubseteq_1)$ et $(L_2, \sqsubseteq_2)$, leur produit est $(L_1 \times L_2, \sqsubseteq)$, où :

- $L_1 \times L_2 \hat{=} \{(v_1, v_2) \mid v_1 \in L_1, v_2 \in L_2\}$ (produit cartésien);
- $(v_1, v_2) \sqsubseteq (v_1', v_2')$ si et seulement si $v_1 \sqsubseteq_1 v_1'$ et $v_2 \sqsubseteq_2 v_2'$.

Théorème : Si $L_1^\sharp, \sqsubseteq_1^\sharp \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \mathbb{P}(V), \subseteq$ et $L_2^\sharp, \sqsubseteq_2^\sharp \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \mathbb{P}(V), \subseteq$

sont des correspondances de Galois, alors

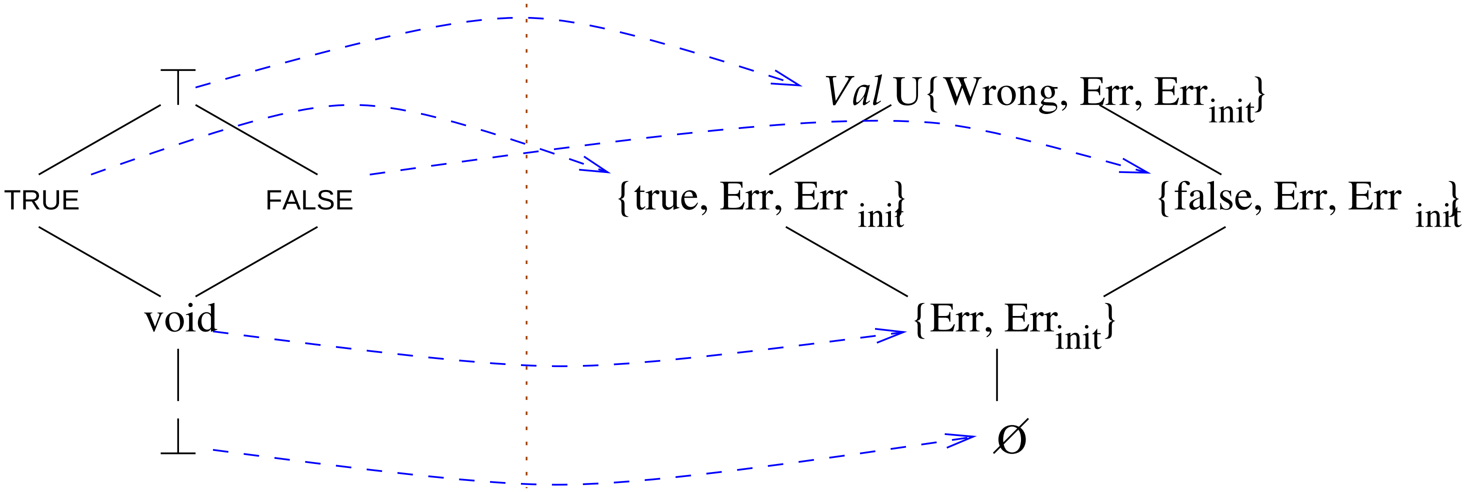
$$L_1^\sharp \times L_2^\sharp, \sqsubseteq^\sharp \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathbb{P}(V), \subseteq$$

aussi, où $\gamma(v_1, v_2) \hat{=} \gamma_1(v_1) \cap \gamma_2(v_2)$, et $\alpha(A) \hat{=} (\alpha_1(A), \alpha_2(A))$.

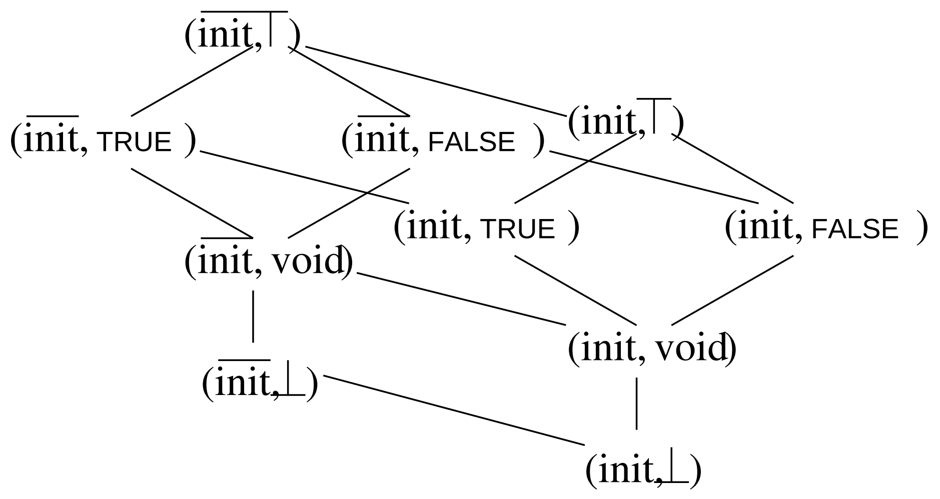
Preuve : $\alpha(A) \sqsubseteq (v_1, v_2) \Leftrightarrow (\alpha_1(A), \alpha_2(A)) \sqsubseteq (v_1, v_2)$
$\Leftrightarrow \alpha_1(A) \sqsubseteq_1 v_1 \wedge \alpha_2(A) \sqsubseteq_2 v_2 \Leftrightarrow A \subseteq \gamma_1(v_1) \wedge A \subseteq \gamma_2(v_2)$
$\Leftrightarrow A \subseteq \gamma(v_1, v_2)$.

Exemple : non-initialisations et constantes

On prend le treillis des constantes booléennes :

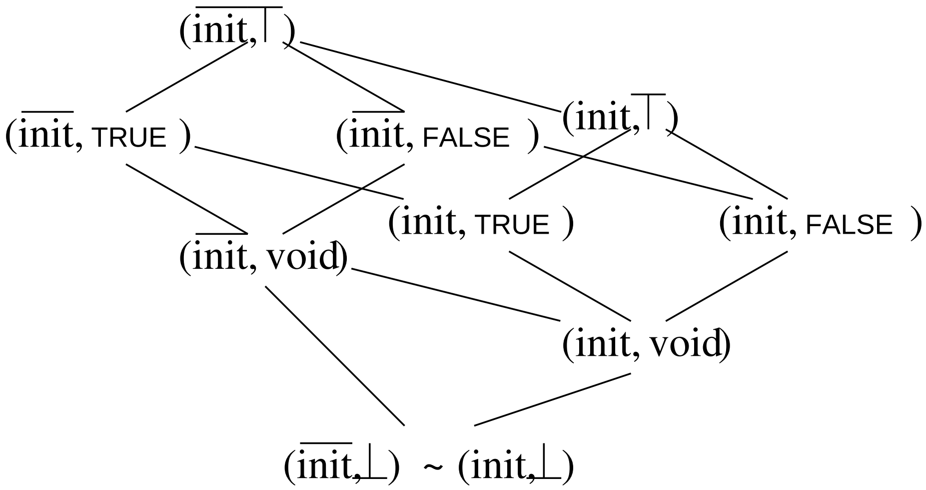


Le produit initialisations $\times$ constantes boolénnes



Réduction

Mais $\gamma(\overline{\mathsf{init}}, \bot) = \gamma(\mathsf{init}, \bot) = \emptyset$, on peut donc identifier les éléments abstraits du bas :



Le produit réduit

Exemple : la valeur abstraite $(\overline{\mathsf{init}}, \bot)$ est telle que
$\gamma(\overline{\mathsf{init}}, \bot) = \gamma_{init}(\overline{\mathsf{init}}) \cap \gamma_{type}(\bot) = (\mathcal{V}al \cup \{\mathsf{Err}, \mathsf{Wrong}, \mathsf{Err}_{init}\}) \cap \emptyset = \emptyset$.
Mais $\gamma(\mathsf{init}, \bot) = \emptyset$ aussi.

En somme, $(\overline{\mathsf{init}}, \bot)$ et $(\mathsf{init}, \bot)$ ont la même sémantique, et on peut les considérer comme équivalentes. ("Peu importe que le code lise une variable non initialisée ou non, de toute façon il est mort.")

On définit une relation d'équivalence $\sim$ par : $(v_1, v_2) \sim (v_1', v_2')$ si et seulement si $\gamma(v_1, v_2) = \gamma(v_1', v_2')$, et on va normaliser tout (v_1, v_2) à un élément canonique $\sim$-équivalent : c'est la réduction.

La réduction, formellement

L'élément canonique associé à $(v_1, v_2) \in L_1^\sharp \times L_2^\sharp$ est $\alpha(\gamma(v_1, v_2))$.

(Par exemple, $\alpha(\gamma(\overline{\mathsf{init}}, \bot)) = (\mathsf{init}, \bot)$.)

Lemme : $\alpha(\gamma(v_1, v_2)) \sim (v_1, v_2)$
Preuve : car $\gamma \circ \alpha \circ \gamma = \gamma$.

Lemme : $\alpha(\gamma(v_1, v_2)) = \min_{\sqsubseteq}\{(v_1', v_2') \mid (v_1', v_2') \sim (v_1, v_2)\}$.
Preuve : si $(v_1', v_2') \sim (v_1, v_2)$, alors $\gamma(v_1, v_2) = \gamma(v_1', v_2')$, donc $\gamma(v_1, v_2) \subseteq \gamma(v_1', v_2')$, donc $\alpha(\gamma(v_1, v_2)) \sqsubseteq (v_1', v_2')$. Donc $\alpha(\gamma(v_1, v_2)) = \bigsqcap\{(v_1', v_2') \mid (v_1', v_2') \sim (v_1, v_2)\}$. Par le lemme précédent, c'est un min.

Corollaire : $(v_1, v_2) \sim (v_1', v_2')$ si et seulement si les formes normales $\alpha(\gamma(v_1, v_2))$ et $\alpha(\gamma(v_1', v_2'))$ coincident.

Produit réduit

Le treillis produit réduit $L_1^\sharp \times_{\alpha,\gamma} L_2^\sharp$ de $L_1^\sharp$ avec $L_2^\sharp$ est l'image $\alpha(\gamma(L_1^\sharp \times L_2^\sharp))$:

- $(v_1, v_2) \sqsubseteq_{\alpha,\gamma} (v_1', v_2')$ si et seulement si $(v_1, v_2) \sqsubseteq (v_1', v_2')$;
- $\bigsqcup_{\alpha,\gamma} F \hat{=} \alpha(\gamma(\bigsqcup F))$.

Lemme : $\bigsqcup_{\alpha,\gamma}$ est bien un sup dans le produit réduit.

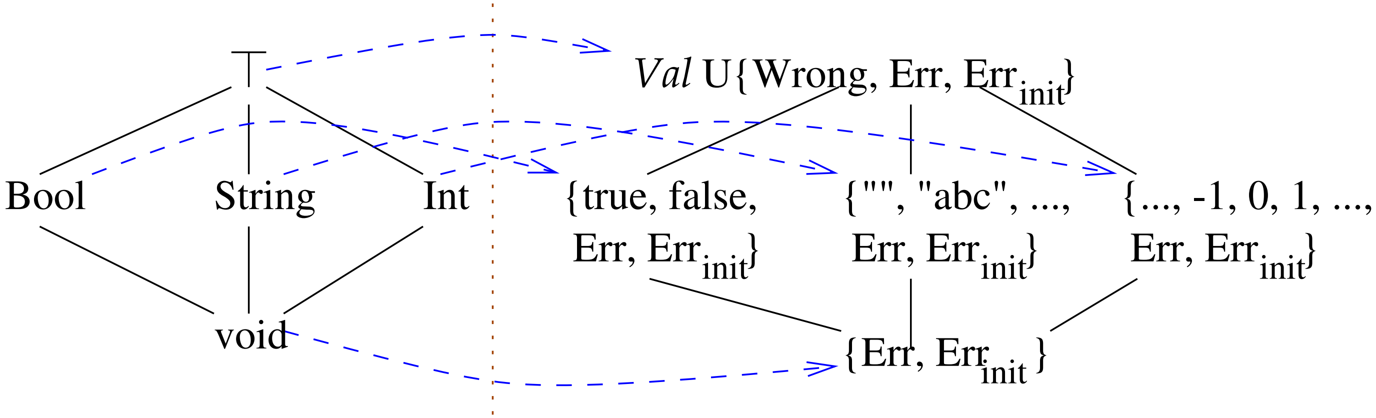
Preuve : (majorant) Soit $x \in F \subseteq L_1^\sharp \times_{\alpha,\gamma} L_2^\sharp$: $\bigsqcup F \sqsupseteq x$, donc $\alpha(\gamma(\bigsqcup F)) \sqsupseteq \alpha(\gamma(x))$ par monotonie. Donc $\bigsqcup_{\alpha,\gamma} F = \alpha(\gamma(\bigsqcup F)) \sqsupseteq \alpha(\gamma(x))$.
Or $x \in L_1^\sharp \times_{\alpha,\gamma} L_2^\sharp$ est de la forme $\alpha(\gamma(y))$ pour un certain y. Donc $\alpha(\gamma(x)) = \alpha(\gamma(\alpha(\gamma(y)))) = \alpha(\gamma(y))$ (car $\alpha \circ \gamma \circ \alpha = \alpha$) $= x$, donc $\bigsqcup_{\alpha,\gamma} F \sqsupseteq x$.
(majorant min) Si $y \in L_1^\sharp \times_{\alpha,\gamma} L_2^\sharp$ majore F, alors $y \sqsupseteq \bigsqcup F \sqsupseteq \alpha(\gamma(\bigsqcup F)) = \bigsqcup_{\alpha,\gamma} F$.

La morale

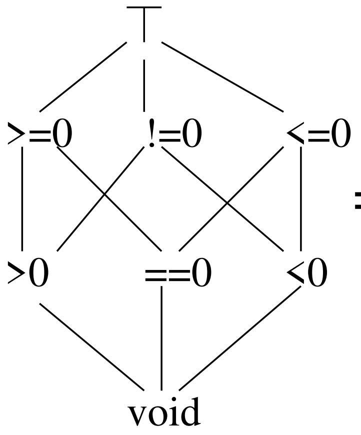
On peut représenter des treillis complexes comme des produits réduits.

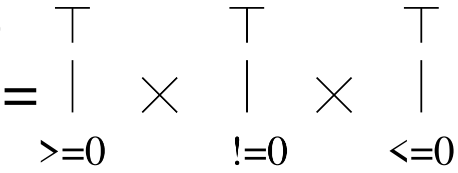
Le treillis des types et des signes est par exemple un produit réduit d'un treillis de types avec un treillis de signes (voir la suite).

Treillis des types



Treillis des signes

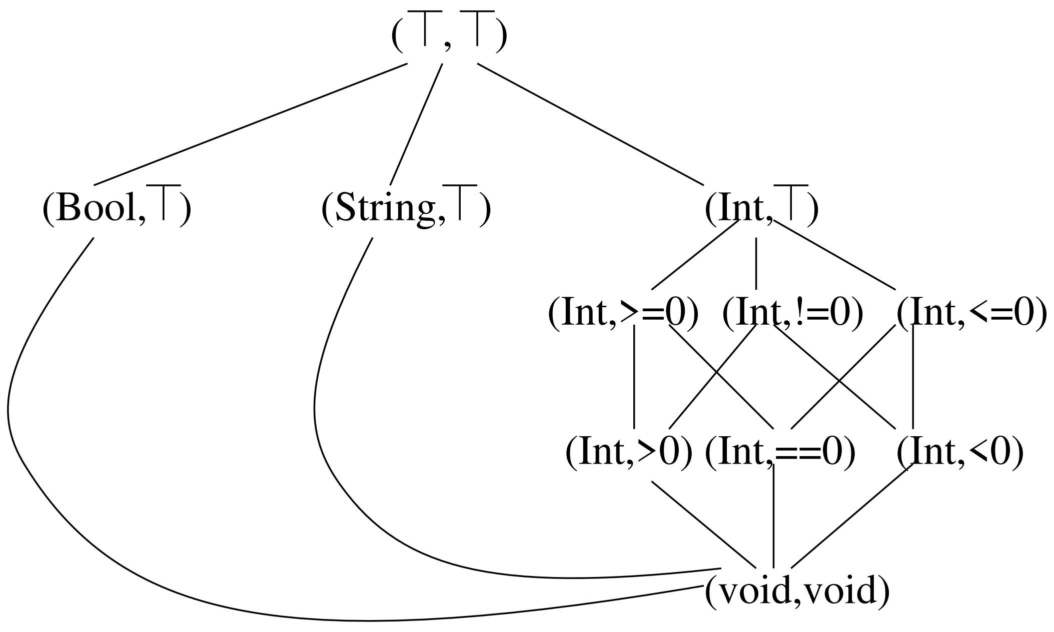




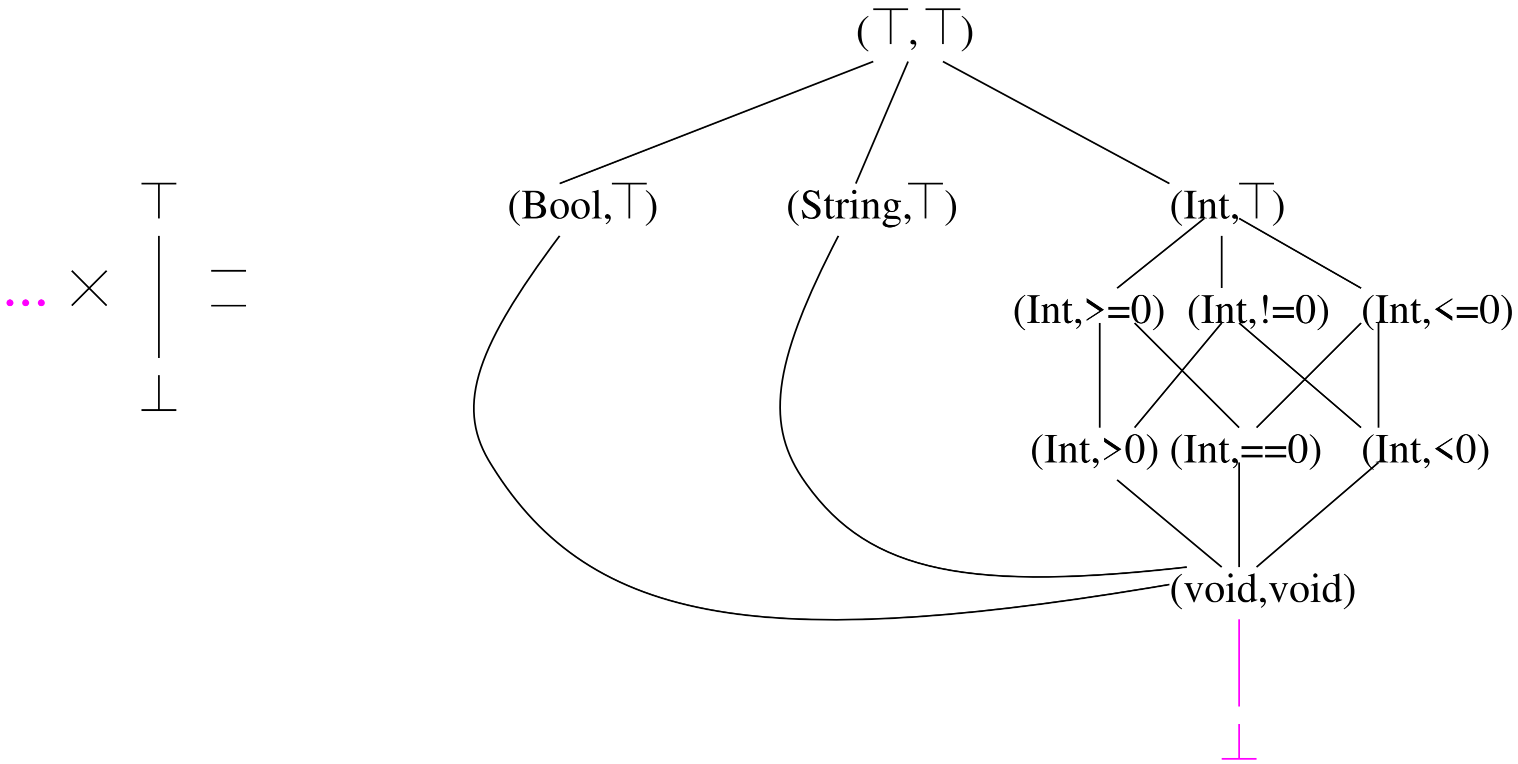
(produit simple,
pas de réduction)

Produit réduit types $\times$ signes

$$
\begin{array}{rcll}
(type, \mathrm{void}) & \leadsto & (\mathrm{void}, \mathrm{void}) & \\
(\mathrm{void}, signe) & \leadsto & (\mathrm{void}, \mathrm{void}) & \\
(type, signe) & \leadsto & (\mathrm{void}, \mathrm{void}) & (type = \mathrm{Bool}, \mathrm{String}; signe \neq \top, \mathrm{void}) \\
(\top, signe) & \leadsto & (\mathrm{Int}, signe) & (signe \neq \top, \mathrm{void})
\end{array}
$$


(⊤, ⊤)
(Bool,⊤)
(String,⊤)
(Int,⊤)
(Int,>=0)
(Int,!=0)
(Int,<=0)
(Int,>0)
(Int,==0)
(Int,<0)
(void,void)

Produit réduit avec un treillis de code mort



… c'est le treillis du transparent 41 ($\gamma(\perp)\hat{=}\emptyset$).

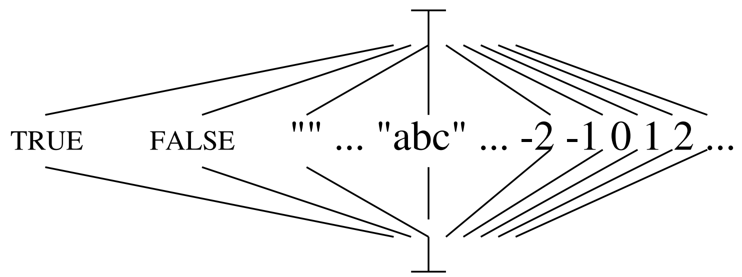
Produit réduit en Caml

```
type types = TTOP | TBOOL | TSTR | TINT | TVOID;;
let types_inf x y = match x with
    TTOP -> y  | TVOID -> TVOID
  | _ -> match y with
               TTOP -> x  | TVOID -> TVOID
             | _ -> if x=y then x else TVOID;;
type signes = bool * bool * bool;; (* false='>=0'/'!=0'/'<=0' *)
let signes_inf (x,y,z) (x',y',z') = (x&&x', y&&y', z&&z');;
let ts_reduit ts = match ts with
    (_, (false, false, false))
  | (TVOID, _) -> (TVOID, (false, false, false))
  | (_, (true, true, true)) -> ts
  | (TTOP, s) -> (TINT, s)
  | (TINT, s) -> (TINT, s)
  | _ -> (TVOID, (false, false, false));;
let ts_inf (t1,s1) (t2,s2) =
    ts_reduit (types_inf t1 t2, signes_inf s1 s2);;
```

Exercices

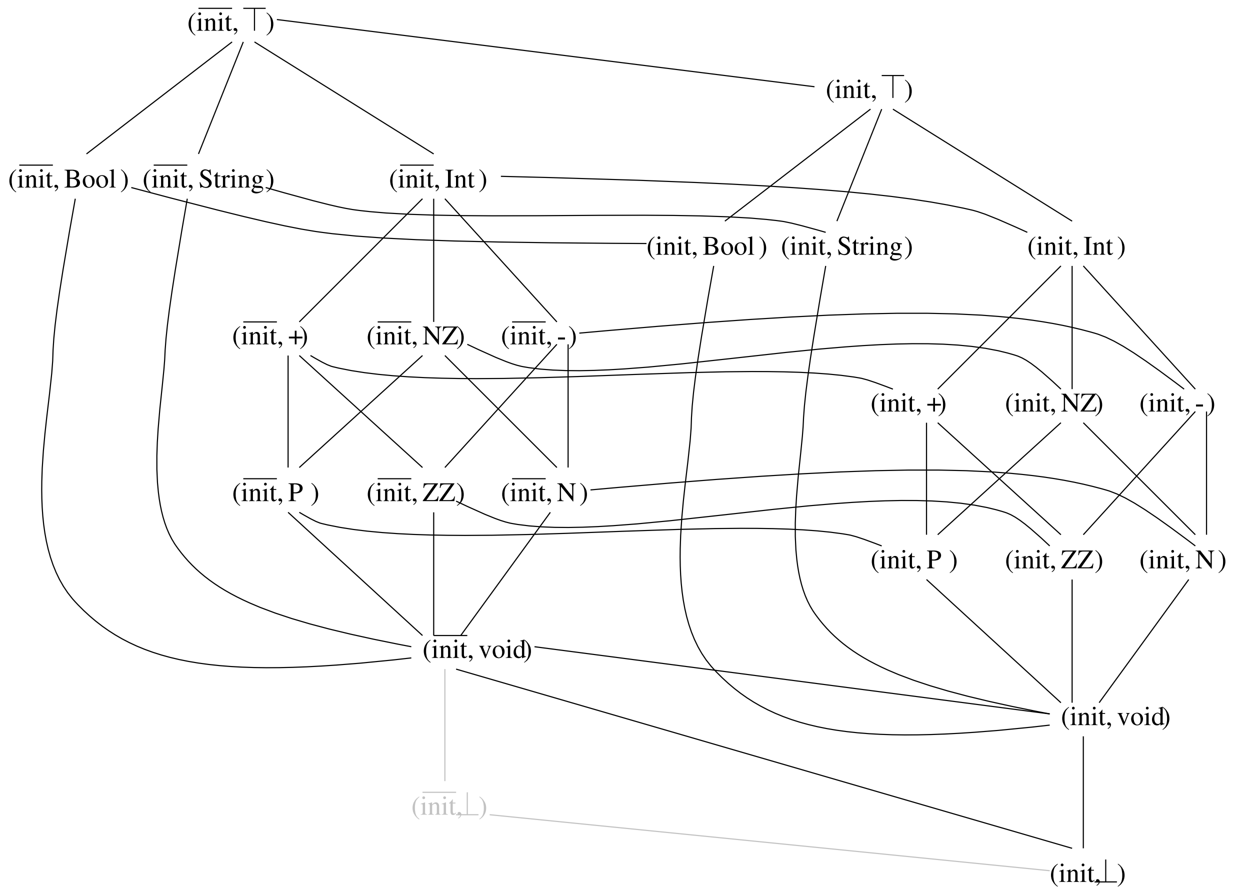
Montrer que le treillis des constantes booléennes du transparent 110 est un treillis produit réduit d'un treillis $(\top, \text{TRUE}, \text{FALSE}, \text{void})$ avec le treillis $(\top, \bot)$ du transparent précédent.

Coder le produit réduit des types et des signes avec le treillis des constantes :



Dans un langage avec pointeurs, imaginer une analyse permettant de détecter les pointeurs déréférencés qui sont possiblement `NULL`.

Le produit réduit des initialisations et des types



… mais c'est plus simple à coder que ça n'en a l'air !

Calcul dans le produit réduit

Rappel : à chaque opération concrète $f : L^\flat \to L^\flat$ correspond $f^\sharp : L^\sharp \to L^\sharp \hat{=} \lambda v \cdot \alpha\{f(x) \mid x \in \gamma(v)\}$.

$$\text{Si } L^\sharp = L_1^\sharp \times L_2^\sharp, \sqsubseteq^\sharp \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} \mathbb{P}(V), \subseteq \text{ alors } L_1^\sharp \times_{\alpha,\gamma} L_2^\sharp = \alpha(\gamma(L^\sharp))$$

Lemme : $f^\sharp : \alpha(\gamma(L^\sharp)) \to \alpha(\gamma(L^\sharp))$, donc $f^\sharp$ est définie et correcte sur le produit réduit.

Preuve :

$f^\sharp(v) = \alpha\{f(x) \mid x \in \gamma(v)\} = \alpha(\gamma(\alpha\{f(x) \mid x \in \gamma(v)\})) \in \alpha(\gamma(L^\sharp))$.