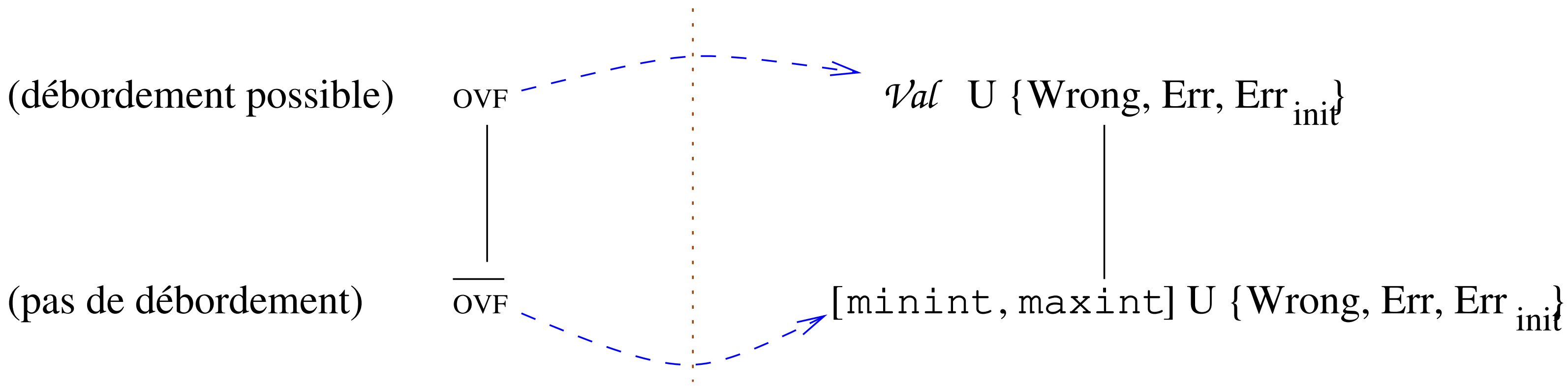
Intervalles, élargissements

- Détection de débordements arithmétiques;
- Treillis des intervalles;
- Élargissements;
- Autres applications.

Débordements arithmétiques

Débordement arithmétique = nombre $\notin [\mathtt{minint}, \mathtt{maxint}]$ (en 32 bits signé, $\mathtt{minint} = -2^{31}$, $\mathtt{maxint} = 2^{31} - 1$).

On peut essayer de les détecter par le treillis :



…mais ça ne fonctionne pas, car ce treillis n'est pas assez précis.

$+^\sharp$	ovf	$\overline{\mathsf{ovf}}$
ovf	ovf	ovf
$\overline{\mathsf{ovf}}$	ovf	ovf

$-^\sharp$	
ovf	ovf
$\overline{\mathsf{ovf}}$	ovf

(Prendre $\texttt{minint} = -2^{31}$, $\texttt{maxint} = 2^{31} - 1$.)

Le treillis des intervalles

Un intervalle non vide est un couple $(\![a, b]\!)$, où $a \in \mathbb{Z} \cup \{-\infty\}$, $b \in \mathbb{Z} \cup \{+\infty\}$ et $a < b$.

$$\gamma((\![a, b]\!)) \hat{=} \{n \in \mathbb{Z} \mid a \leq n \wedge n \leq b\} \cup \{\mathsf{Wrong}, \mathsf{Err}, \mathsf{Err}_{init}\}$$

Un intervalle est un intervalle non vide ou bien le symbole spécial $\boxslash$.

$$\gamma(\boxslash) \hat{=} \{\mathsf{Wrong}, \mathsf{Err}, \mathsf{Err}_{init}\}$$

$L^\sharp_{itv} \hat{=}$ ensemble des intervalles + $\top$.

Ordre sur les intervalles

$$
\begin{array}{rcll}
\boxslash & \sqsubseteq_{itv} & I & \text{pour tout intervalle } I \\
(\![a, b]\!) & \sqsubseteq_{itv} & (\![a', b']\!) & \text{ssi } a' \leq a \text{ et } b \leq b' \\
(\![a, b]\!) & \not\sqsubseteq_{itv} & \boxslash & \\
I & \sqsubseteq_{itv} & \top &
\end{array}
$$

$$
\begin{array}{rcll}
\alpha_{itv}(A) & \hat{=} & \top & (A \cap (\mathrm{Bool} \cup \mathrm{String}) \neq \emptyset) \\
\alpha_{itv}(A) & \hat{=} & \boxslash & (A \subseteq \{\mathsf{Err}, \mathsf{Err}_{init}, \mathsf{Wrong}\}) \\
\alpha_{itv}(A) & \hat{=} & \multicolumn{2}{l}{(\![\inf(A \cap \mathbb{Z}), \sup(A \cap \mathbb{Z})]\!)} \\
 & & \multicolumn{2}{r}{(A \subseteq \mathbb{Z} \cup \{\mathsf{Err}, \mathsf{Err}_{init}, \mathsf{Wrong}\}, A \cap \mathbb{Z} \neq \emptyset)}
\end{array}
$$

Note : $L^\sharp_{itv}$ est un produit réduit

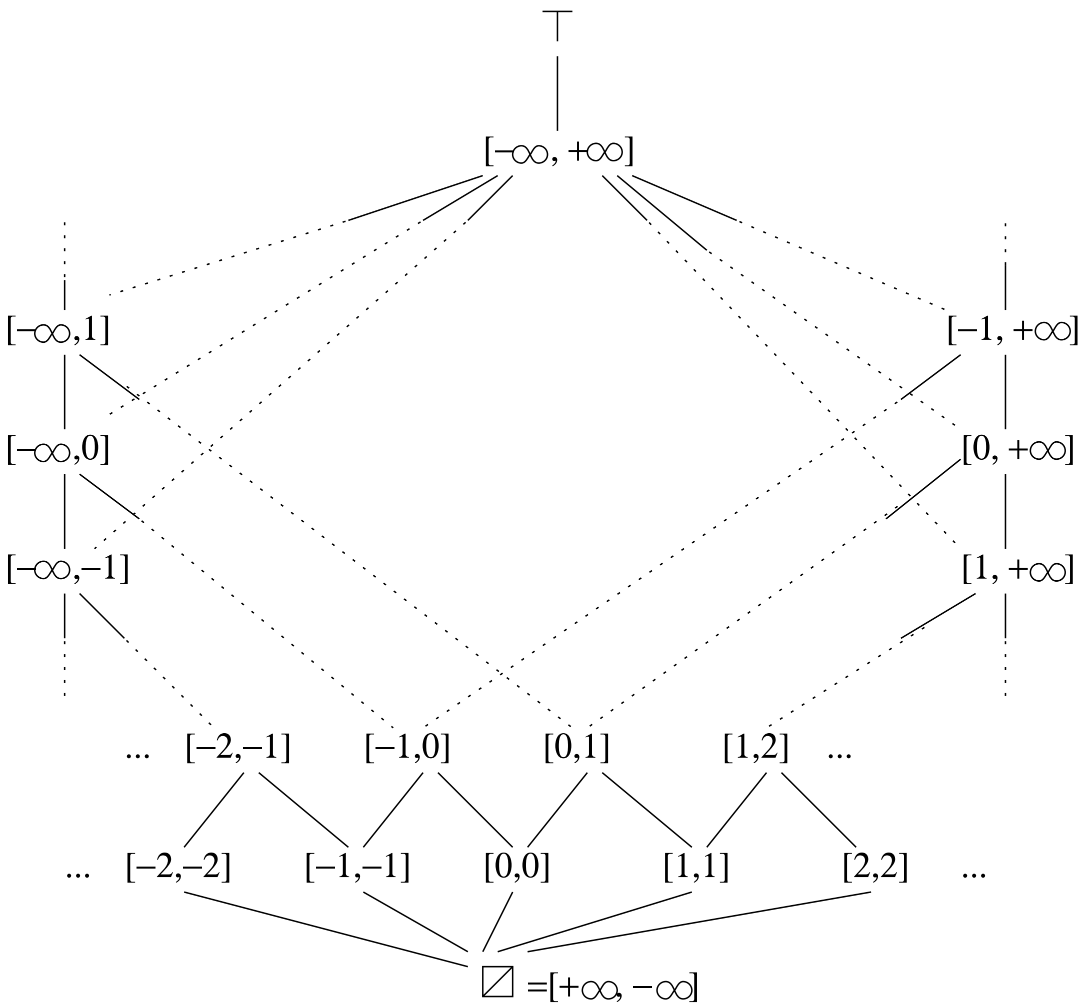
$L^\sharp_{itv,1} \hat{=} \mathbb{Z} \cup \{-\infty, +\infty\}$	$L^\sharp_{itv,2} \hat{=} \mathbb{Z} \cup \{-\infty, +\infty\}$
$a \sqsubseteq^\sharp_{itv,1} a'$ ssi $a \geq a'$ $\gamma_1(a) \hat{=} [a, +\infty[$ $\cup \{\mathsf{Wrong}, \mathsf{Err}, \mathsf{Err}_{init}\}$ $\alpha_1(A) \hat{=} \inf(A \cap \mathbb{Z})$	$b \sqsubseteq^\sharp_{itv,1} b'$ ssi $b \leq b'$ $\gamma_2(a) \hat{=}]-\infty, b]$ $\cup \{\mathsf{Wrong}, \mathsf{Err}, \mathsf{Err}_{init}\}$ $\alpha_2(A) \hat{=} \sup(A \cap \mathbb{Z})$

Ex : vérifier que le produit réduit de $L^\sharp_{itv,1}$ et $L^\sharp_{itv,2}$, plus $\top$, est bien $L^\sharp_{itv}$. Vérifier que $\boxslash$ y est représenté sous la forme du couple $([\![+\infty, -\infty]\!])$.

Réalisation en Caml

```
(ocamlmktop -custom -o ocamlnum nums.cma -cclib -lnums;
 ocamlnum)
open Big_int;;
type z = ZVAL of big_int | PINF | MINF;;
let (<==) x y = match x with
    MINF -> true   | PINF -> y=PINF
  | ZVAL ix -> match y with
               PINF -> true
             | ZVAL iy -> compare_big_int ix iy<=0
             | MINF -> false;;
type itv = ITVTOP | ITV of z * z;;
let itv_reduit (x, y) = if x<==y then (x, y) else (PINF, MINF);;
let itv_plus_precis v1 v2 = match v2 with ITVTOP -> true
  | ITV (x2, y2) -> match v1 with ITVTOP -> false
  | ITV (x1, y1) -> x2<==x1 && y1<==y2;;
```

Le treillis des intervalles en images



Problème de terminaison

Ce treillis n'a pas la condition de chaîne croissante : la terminaison de l'interprétation abstraite n'est pas assurée.

En fait, elle ne termine pas :

y $\in$ [1,1]

```
y:=1;
while (x!=0) {
  y:=y+1;
  x:=x-1;
}
```

1er tour: y $\in$ [1,1]
2eme tour: y $\in$ $[1,1] \sqcup^\# [2,2] = [1,2]$
3eme tour: y $\in$ $[1,2] \sqcup^\# [2,3] = [1,3]$
...

Élargissements

Idée : aux entrées de `while`, remplacer le calcul du $oldv \sqcup newv$ par un $oldv \nabla newv$, tel que :

$$oldv \sqcup newv \sqsubseteq^\sharp oldv \nabla newv \qquad \text{(correction)}$$

$$\text{toute chaîne } v_{n+1} \hat{=} v_n \nabla v'_n$$

$$\text{est stationnaire : } \exists n \cdot v_n = v_{n+1} = v_{n+2} = \ldots \qquad \text{(terminaison)}$$

Dans les intervalles, on peut poser :

$$\begin{array}{rcl} ([a,b]) \nabla ([a',b']) & \hat{=} & ([\mathtt{if}\ a' < a\ \mathtt{then}\ -\infty\ \mathtt{else}\ a, \\ & & \mathtt{if}\ b' > b\ \mathtt{then}\ +\infty\ \mathtt{else}\ b]) \\ ([a,b]) \nabla \top \hat{=} \top \nabla ([a,b]) & \hat{=} & \top \end{array}$$

La terminaison retrouvée

$y \in [1,1]$

```
y:=1;
while (x!=0) {
  y:=y+1;
  x:=x-1;
}
```

1er tour: $y \in [1,1]$

2eme tour: $y \in [1,1] \nabla [2,2] = [1,+\infty]$

3eme tour: $y \in [1,+\infty] \nabla [2,+\infty] = [1,+\infty]$

(En fait, ici, on a même trouvé l'information exacte $y \in [1, +\infty]$.)

L'analyseur, avec élargissements (cf. transparent 97)

$$
\begin{array}{l}
\texttt{let rec cfg\_type } E^\sharp\ \rho^\sharp\ \mathtt{graphe}\ \mathtt{sommet} = \\
\quad \texttt{if } E^\sharp(\mathtt{sommet}) = \rho^\sharp \\
\quad\quad \texttt{then return;} \\
\quad \texttt{else } E^\sharp(\mathtt{sommet}) := \begin{cases} E^\sharp(\mathtt{sommet}) \nabla^* \rho^\sharp & \text{si } \mathtt{sommet} \text{ est un point d'élargissement} \\ E^\sharp(\mathtt{sommet}) \sqcup^{\sharp *} \rho^\sharp & \text{sinon} \end{cases} ; \\
\quad\quad \texttt{let } \mathtt{arcs\_sortants} = \{p \mid \mathtt{sommet} \xrightarrow{instr} p \text{ arc de } \mathtt{graphe}\} \\
\quad\quad \texttt{in } \text{pour tout } (\mathtt{sommet} \xrightarrow{instr} p) \in \mathtt{arcs\_sortants} : \\
\quad\quad\quad \texttt{cfg\_type } E^\sharp\ (cond^\sharp [\![instr]\!] \rho^\sharp)\ \mathtt{graphe}\ p;;
\end{array}
$$

... termine si toute boucle dans le graphe de contrôle passe par un point d'élargissement.

Comment calculer les points d'élargissement

Dans notre langage, on pourrait le faire à la construction du graphe de contrôle, en décidant de mettre un point d'élargissement à l'entrée de chaque boucle `while`.

En général (dans un langage avec des `goto`), il est plus difficile d'identifier les boucles.

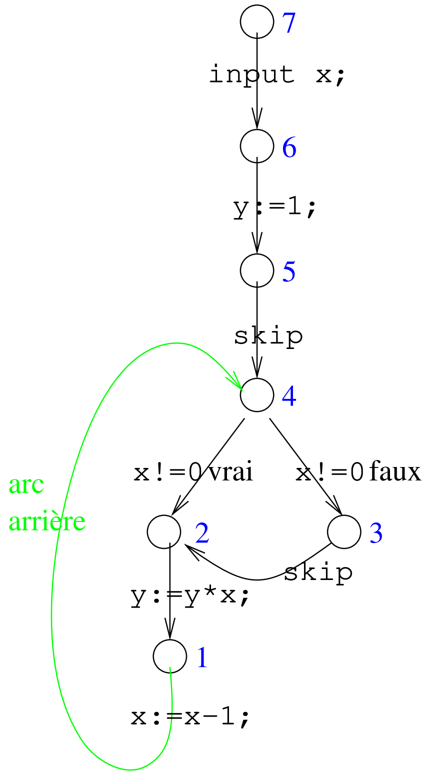
On trouve les points d'élargissement par tri topologique inverse.

Tri topologique inverse

Principe : numéroter les nœuds du graphe de contrôle de sorte que tout arc $m \stackrel{instr}{\longrightarrow} n$ soit tel que $m > n$, sauf les arcs arrière pour lesquels $m \leq n$; et si on élimine les arcs arrière, on élimine toutes les boucles.

```
let revtopsort graphe sommet =
    if sommet ∈ dejavus
       then return;
    else dejavus := dejavus ∪ {sommet};
         let arcs_sortants = {p | sommet –instr→ p arc de graphe}
         in pour tout (sommet –instr→ p) ∈ arcs_sortants
                     revtopsort graphe p;
            n:=n + 1;
            donner à sommet le numéro n;
dejavus := ∅; n := 0; revtopsort graphe I;;
```

Exemple de tri topologique inverse



Il suffit de prendre comme points d'élargissements les cibles d'arcs arrières (ici, le point 4) :

$$
\begin{array}{l}
\texttt{let rec cfg\_type } E^\sharp\ \rho^\sharp\ \mathtt{graphe}\ \mathtt{sommet}\ \mathtt{elargir} = \\
\quad \texttt{if } E^\sharp(\mathtt{sommet}) = \rho^\sharp \\
\qquad \texttt{then return;} \\
\quad \texttt{else } E^\sharp(\mathtt{sommet}) := \left\{ \begin{array}{ll} E^\sharp(\mathtt{sommet}) \nabla^* \rho^\sharp & \text{si } \mathtt{elargir} = \mathtt{true} \\ E^\sharp(\mathtt{sommet}) \sqcup^{\sharp *} \rho^\sharp & \text{sinon} \end{array} \right. ; \\
\qquad \texttt{let } \mathtt{arcs\_sortants} = \{p \mid \mathtt{sommet} \xrightarrow{instr} p \text{ arc de } \mathtt{graphe}\} \\
\qquad \texttt{in } \text{pour tout } (\mathtt{sommet} \xrightarrow{instr} p) \in \mathtt{arcs\_sortants} : \\
\qquad\quad \texttt{cfg\_type } E^\sharp\ (cond^\sharp [\![instr]\!] \rho^\sharp)\ \mathtt{graphe}\ p\ (p \geq \mathtt{sommet});; \\
\texttt{cfg\_type } (\lambda p, x \cdot \bot)\ (\lambda x \cdot \top)\ \mathtt{graphe}\ I\ \mathtt{false};;
\end{array}
$$

Remarques

Pour détecter les débordements arithmétiques, on peut utiliser un treillis qui a la condition de chaîne croissante : celui des $(\![a, b]\!)$, avec $a, b \in \{-\infty\} \cup [\mathtt{minint}, \mathtt{maxint}] \cup \{+\infty\}$.

On n'a alors pas besoin de `big_int`.
On n'a pas non plus (en théorie) besoin d'élargissement ...
sauf que les chaînes croissantes peuvent contenir jusqu'à $\mathtt{maxint} - \mathtt{minint} + 3 \approx 2^{32}$ éléments, ce qui fait que la convergence de l'analyse peut être très lente !
On utilise alors un élargissement pour accélérer la convergence.

Indices de tableaux

C'est une autre application du treillis des intervalles. Dans :

```
int a[100];
p:  ... a[i] ...
```

si $\rho^\sharp = Env^\sharp(\texttt{p})$ et $\rho^\sharp(\texttt{i}) \sqsubseteq^\sharp ([0, 99])$, alors `a[i]` est un accès correct au tableau.

Sinon, on peut :

- Afficher un warning "débordement de tableau possible" (mais `gcc` ne le fait pas, par exemple);
- Compiler une vérification de bornes de tableau à l'exécution (comme en Java, Ada, Pascal); on peut alors supprimer cette vérification si l'analyseur trouvait $\rho^\sharp(\texttt{i}) \sqsubseteq^\sharp ([0, 99])$.