

Abstractions de sémantiques

TAS : Typage et analyse statique
M2, Master STL INSTA, Sorbonne Université

Antoine Miné

Année 2021–2022

Cours 9
4 mars 2022

Plan du cours

But :

construire une analyse statique par abstraction de la sémantique concrète

- notion **intuitive** d'abstraction
les signes
- **formalisation** de la notion d'abstraction
correspondance de Galois, abstraction **sûre**, abstraction **optimale**
quelques preuves * *
- analyse **non-relationnelle**
dérivation systématique depuis une **abstraction de valeurs**
- exemple 1 : le domaine des **constantes**
- exemple 2 : le domaine des **intervalles**
gestion précise des tests * *
- conseils d'implantation pour le projet :
analyse non-relationnelle en OCaml
analyse des constantes à compléter
analyse des intervalles à réaliser

Interprétation abstraite

- **Cadre unifié pour les sémantiques** :
sémantiques définies comme des **points-fixes**
sémantiques définies par induction sur la syntaxe (interprétation)
- **Comparer le pouvoir d'expression des sémantiques**
via des fonctions d'abstraction et de concrétisation
déterminer ce qui peut et ne peut pas être prouvé par une sémantique
- **Dériver systématiquement des analyses statiques**
par abstraction d'une sémantique concrète
- Deux aspects à la notion d'abstraction :
 - approximation
 $\{0, 80\}$ n'est pas exprimable dans les intervalles, il est approximé par l'intervalle $[0, 80]$
 - représentation
 $[0, 80]$ n'est pas représenté par un ensemble $\{0, 1, \dots, 80\}$, mais par la paire $(0, 80)$
- Assurer la **sûreté** (*soundness*)
toute propriété prouvée dans l'abstrait est vraie dans le concret
incomplétude : on ne peut pas tout prouver dans l'abstrait
- Assurer la **calculabilité effective**
domaines finis (signes) ou accélération de point-fixes (cours 10)

Rappel : sémantique concrète collectrice (1/2)

$E[expr] : \mathcal{E} \rightarrow \mathcal{P}(\mathbb{Z})$ état mémoire \rightarrow valeurs *possibles* de l'expression *expr*

$$E[V] \rho \stackrel{\text{def}}{=} \{\rho(V)\}$$

$$E[c] \rho \stackrel{\text{def}}{=} \{c\}$$

$$E[\text{rand}(a, b)] \rho \stackrel{\text{def}}{=} \{x \in \mathbb{Z} \mid a \leq x \leq b\}$$

$$E[e_1 + e_2] \rho \stackrel{\text{def}}{=} \{v_1 + v_2 \mid v_1 \in E[e_1] \rho, v_2 \in E[e_2] \rho\}$$

...

$C[cond] : \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$ états mémoire qui *peuvent* passer le test *cond*

$$C[\text{true}] R \stackrel{\text{def}}{=} R$$

$$C[\text{false}] R \stackrel{\text{def}}{=} \emptyset$$

$$C[c_1 \wedge c_2] R \stackrel{\text{def}}{=} C[c_1] R \cap C[c_2] R$$

$$C[c_1 \vee c_2] R \stackrel{\text{def}}{=} C[c_1] R \cup C[c_2] R$$

$$C[e_1 = e_2] R \stackrel{\text{def}}{=} \{\rho \in R \mid \exists v_1 \in E[e_1] \rho, v_2 \in E[e_2] \rho : v_1 = v_2\}$$

$$C[e_1 < e_2] R \stackrel{\text{def}}{=} \{\rho \in R \mid \exists v_1 \in E[e_1] \rho, v_2 \in E[e_2] \rho : v_1 < v_2\}$$

...

Environnements : $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \rightarrow \mathbb{Z}$

états mémoire, associant à chaque variable dans \mathbb{V} une valeur dans \mathbb{Z}

Rappel : sémantique concrète collectrice (2/2)

$S\llbracket \text{stat} \rrbracket : \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$

$S\llbracket \text{skip} \rrbracket R$	$\stackrel{\text{def}}{=}$	R
$S\llbracket \text{halt} \rrbracket R$	$\stackrel{\text{def}}{=}$	\emptyset
$S\llbracket X \leftarrow e \rrbracket R$	$\stackrel{\text{def}}{=}$	$\{ \rho[X \mapsto v] \mid \rho \in R, v \in E\llbracket e \rrbracket \rho \}$
$S\llbracket s_1; s_2 \rrbracket R$	$\stackrel{\text{def}}{=}$	$S\llbracket s_2 \rrbracket (S\llbracket s_1 \rrbracket R)$
$S\llbracket \text{if } c \text{ then } s_1 \text{ else } s_2 \rrbracket R$	$\stackrel{\text{def}}{=}$	$S\llbracket s_1 \rrbracket (C\llbracket c \rrbracket R) \cup S\llbracket s_2 \rrbracket (C\llbracket \neg c \rrbracket R)$
$S\llbracket \text{assert } c \rrbracket R$	$\stackrel{\text{def}}{=}$	$C\llbracket c \rrbracket R$
$S\llbracket \text{while } c \text{ do } s \rrbracket R$	$\stackrel{\text{def}}{=}$	$C\llbracket \neg c \rrbracket (\text{Ifp } \lambda X. R \cup S\llbracket s \rrbracket (C\llbracket c \rrbracket X))$

états mémoire en **entrée** de *stat* \rightarrow états mémoire **possibles en sortie** de *stat*

Limite à l'automatisation

La sémantique concrète n'est pas calculable car :

- 1 les éléments du domaine concret $\mathcal{P}(\mathcal{E})$
ne sont **pas tous représentables** en mémoire ;
- 2 le point fixe $\text{lfp } \lambda X. R \cup S \llbracket s \rrbracket (C \llbracket c \rrbracket X)$
peut faire intervenir un nombre **infini** d'itérations
théorème de Kleene

L'interprétation abstraite fournit une solution à ces deux problèmes :

- 1 remplacer $\mathcal{P}(\mathcal{E})$ par un **domaine abstrait**
dont les éléments sont représentables en mémoire
 \implies **ce cours**
- 2 calculer les itérations de point fixe avec **accélération de convergence**
 \implies **prochain cours**

Analyse abstraite, analyse approchée

Plutôt que de raisonner sur le comportement réel des programmes, nous raisonnons à un niveau d'**abstraction** plus simple.

Exemple : l'abstraction des **signes**

- **oublier** la valeur exacte des variables
et ne se souvenir que de leur signe : ≥ 0 , ≤ 0 , 0 , \top (aucune information)
 $\implies 2$ devient ≥ 0
- remplacer un calcul sur les valeurs par un calcul sur les signes ;
au lieu de $2 + 2 = 4$, nous écrivons $(\geq 0) + (\geq 0) = (\geq 0)$.
 \implies plus efficace !
- **perte de précision**
dans les **concret** (valeurs) : $2 - 1 = 1$, qui est positif
après **abstraction** (signes) : $(\geq 0) - (\geq 0) = \top$ (pas d'information)
- si l'analyseur indique $X = (\geq 0)$ en fin d'analyse,
nous savons que X sera **positif** (sans connaître précisément ses valeurs).

Ensembles et propriétés

Nous raisonnons sur des **propriétés**, i.e., des **ensembles** de comportements.

Exemple : abstraction d'une valeur entière par un signe

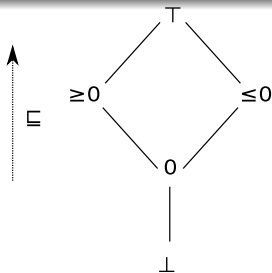
- domaine concret : $\mathcal{D} \stackrel{\text{def}}{=} \mathcal{P}(\mathbb{Z})$
- domaine abstrait : $\mathcal{D}^\# \stackrel{\text{def}}{=} \{0, \geq 0, \leq 0, \top, \perp\}$
 - ≥ 0 signifie « positif » $\simeq \mathbb{N}$
 - 0 signifie « nul » $\simeq \{0\}$
 - \top signifie « pas d'information » $\simeq \mathbb{Z}$
 - \perp signifie « impossible » $\simeq \emptyset$

La valeur $x \in \mathbb{Z}$ a la propriété Y si $x \in Y$.

L'ensemble $X \in \mathcal{P}(\mathbb{Z})$ a la propriété Y si $X \subseteq Y$.

\implies une valeur, un ensemble peut satisfaire **plusieurs propriétés** de $\mathcal{D}^\#$

Ordre d'information, meilleur abstraction



Intuitivement, les propriétés peuvent être ordonnées par un **ordre d'information** \sqsubseteq :
 $\perp \sqsubseteq 0 \sqsubseteq (\geq 0) \sqsubseteq \top$.

C'est un **ordre partiel** : ≤ 0 et ≥ 0 ne sont pas comparables.

Pour les signes, nous avons même une structure de **treillis complet**.

L'ordre est **compatible** avec l'interprétation ensembliste,
en effet : $\perp \sqsubseteq 0 \sqsubseteq (\geq 0) \sqsubseteq \top$ implique $\emptyset \subseteq \{0\} \subseteq \mathbb{N} \subseteq \mathbb{Z}$.

Tout ensemble d'entiers a une **meilleur** représentation sous forme
d'information de signe (i.e., plus petite pour \sqsubseteq)

0 est une meilleur représentation que ≥ 0 ou \top pour $\{0\}$

elle donne plus d'information, elle représente un ensemble plus petit d'entiers

Opérateurs abstraits

Exemple : la règle des signes

\times	≥ 0	≤ 0	0	\top	\perp
≥ 0	≥ 0	≤ 0	0	\top	\perp
≤ 0	≤ 0	≥ 0	0	\top	\perp
0	0	0	0	0	\perp
\top	\top	\top	0	\top	\perp
\perp	\perp	\perp	\perp	\perp	\perp

$+$	≥ 0	≤ 0	0	\top	\perp
≥ 0	≥ 0	\top	≥ 0	\top	\perp
≤ 0	\top	≤ 0	≤ 0	\top	\perp
0	≥ 0	≤ 0	0	\top	\perp
\top	\top	\top	\top	\top	\perp
\perp	\perp	\perp	\perp	\perp	\perp

Principe :

Remplacer un calcul d'opérateur \diamond sur \mathbb{Z} par un calcul abstrait $\diamond^\#$ sur $\mathcal{D}^\#$.

Raisonnement dans les signes $\mathcal{D}^\#$ compatible avec celui dans $\mathcal{P}(\mathbb{Z})$:

- si $a, b \in \mathbb{Z}$ ont la propriété $a^\#, b^\# \in \mathcal{D}^\#$
alors $a \diamond b$ a bien la propriété $a^\# \diamond^\# b^\#$
 \implies sûreté
- parfois, $\diamond^\#$ donne la plus forte propriété dans $\mathcal{D}^\#$
 \implies optimalité

La sûreté sera toujours garantie ;
l'optimalité sera garantie dans certains cas seulement.

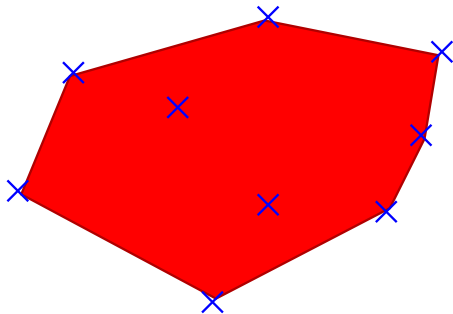
Exemples de domaines abstraits numériques



domaine concret \mathcal{D} :

$\{(0, 3), (6, 0), (12, 7), \dots\}$

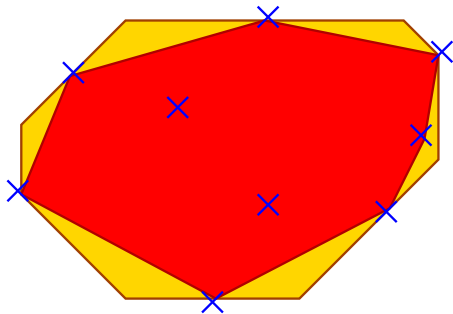
Exemples de domaines abstraits numériques



domaine concret \mathcal{D} : $\{(0, 3), (6, 0), (12, 7), \dots\}$

abstraction des polyèdres $\mathcal{D}_p^\#$: $6X + 11Y \geq 33 \wedge \dots$

Exemples de domaines abstraits numériques

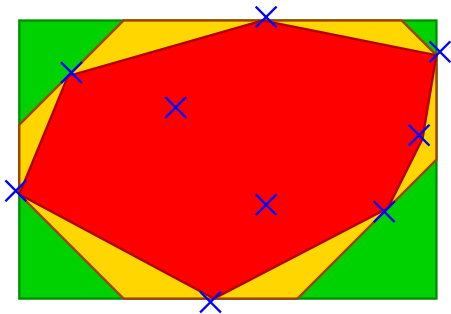


domaine concret \mathcal{D} : $\{(0, 3), (6, 0), (12, 7), \dots\}$

abstraction des polyèdres $\mathcal{D}_p^\#$: $6X + 11Y \geq 33 \wedge \dots$

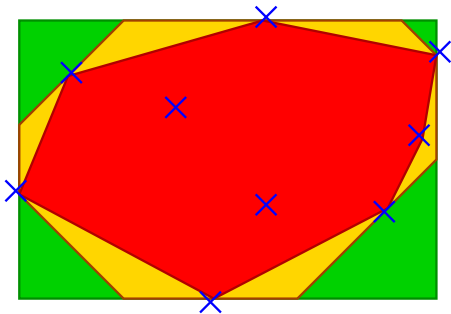
abstraction des octogones $\mathcal{D}_o^\#$: $X + Y \geq 3 \wedge Y \geq 0 \wedge \dots$

Exemples de domaines abstraits numériques



domaine concret \mathcal{D} : $\{(0, 3), (6, 0), (12, 7), \dots\}$
abstraction des polyèdres $\mathcal{D}_p^\#$: $6X + 11Y \geq 33 \wedge \dots$
abstraction des octogones $\mathcal{D}_o^\#$: $X + Y \geq 3 \wedge Y \geq 0 \wedge \dots$
abstraction des intervalles $\mathcal{D}_i^\#$: $X \in [0, 12] \wedge Y \in [0, 8]$

Exemples de domaines abstraits numériques



domaine concret \mathcal{D} : $\{(0, 3), (6, 0), (12, 7), \dots\}$

abstraction des polyèdres $\mathcal{D}_p^\#$: $6X + 11Y \geq 33 \wedge \dots$

abstraction des octogones $\mathcal{D}_o^\#$: $X + Y \geq 3 \wedge Y \geq 0 \wedge \dots$

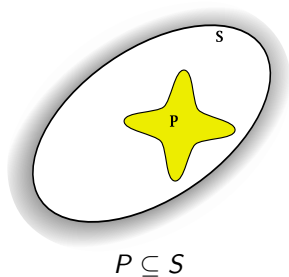
abstraction des intervalles $\mathcal{D}_i^\#$: $X \in [0, 12] \wedge Y \in [0, 8]$

non calculable
coût exponentiel
coût cubique
coût linéaire

L'abstraction correspond à une **sur-approximation**.

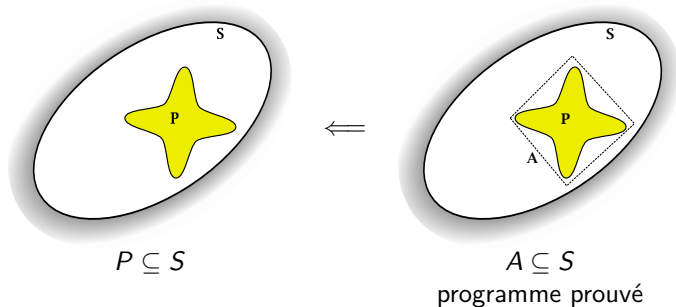
Compromis entre coût et expressivité / précision.

Sûreté et fausses alarmes



But : prouver qu'un programme P satisfait sa spécification S

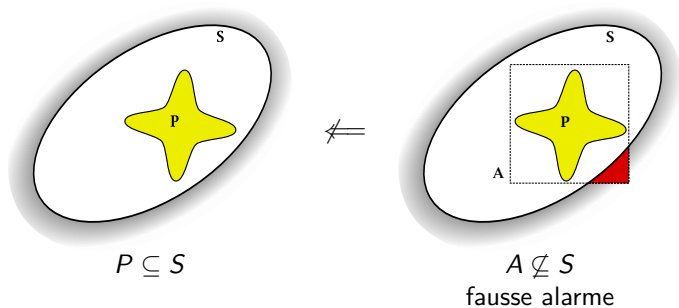
Sûreté et fausses alarmes



But : prouver qu'un programme P satisfait sa spécification S

Une abstraction polyédrique A peut prouver la correction.

Sûreté et fausses alarmes



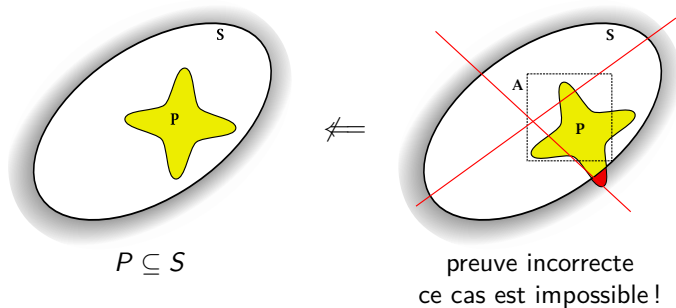
But : prouver qu'un programme P satisfait sa spécification S

Une abstraction polyédrique A peut prouver la correction.

Une abstraction d'intervalle ne peut pas prouver la correction

\Rightarrow fausse alarme.

Sûreté et fausses alarmes



But : prouver qu'un programme P satisfait sa spécification S

Une abstraction polyédrique A peut prouver la correction.

Une abstraction d'intervalle ne peut pas prouver la correction

\Rightarrow **fausse alarme.**

L'analyse est **toujours sûre** \Rightarrow jamais de faux négatif!

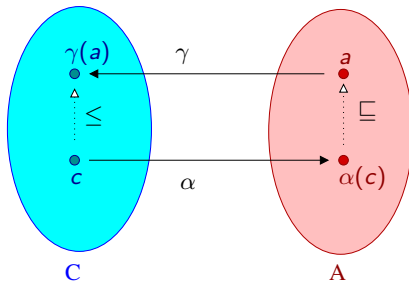
Formalisation de la correspondance concret/abstrait

Correspondance de Galois

Étant donnés deux posets (C, \leq) et (A, \sqsubseteq) ,
la paire $(\alpha : C \rightarrow A, \gamma : A \rightarrow C)$ est une **correspondance de Galois** si :

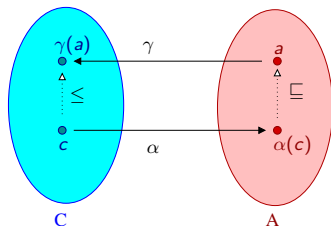
$$\forall a \in A, c \in C, \alpha(c) \sqsubseteq a \iff c \leq \gamma(a)$$

ce que nous notons : $(C, \leq) \xleftrightarrow[\alpha]{\gamma} (A, \sqsubseteq)$.



- α est l'**adjoint supérieur**, ou **abstraction** ; A est le domaine abstrait ;
- γ est l'**adjoint inférieur**, ou **concrétisation** ; C est le domaine concret.

Exemple : correspondance de Galois pour les signes



Les signes représentent des ensembles d'entiers :

- $C \stackrel{\text{def}}{=} \mathcal{P}(\mathbb{Z})$;
- $A \stackrel{\text{def}}{=} \{\perp, 0, \geq 0, \leq 0, \top\}$;
- l'ordre concret \leq est \sqsubseteq .

$$\begin{aligned}
 \gamma(\perp) &\stackrel{\text{def}}{=} \emptyset \\
 \gamma(0) &\stackrel{\text{def}}{=} \{0\} \\
 \gamma(\geq 0) &\stackrel{\text{def}}{=} \mathbb{N} \\
 \gamma(\leq 0) &\stackrel{\text{def}}{=} -\mathbb{N} \\
 \gamma(\top) &\stackrel{\text{def}}{=} \mathbb{Z}
 \end{aligned}$$

$$\alpha(S) \stackrel{\text{def}}{=} \begin{cases} \perp & \text{si } S = \emptyset \\ 0 & \text{si } S = \{0\} \\ \geq 0 & \text{sinon, si } \forall s \in S, s \geq 0 \\ \leq 0 & \text{sinon, si } \forall s \in S, s \leq 0 \\ \top & \text{sinon} \end{cases}$$

À vérifier : $\alpha(c) \sqsubseteq a \iff c \subseteq \gamma(a) \dots$

Caractérisation alternative

Une paire $(\alpha : C \rightarrow A, \gamma : A \rightarrow C)$ est une **correspondance de Galois** si et seulement si elle satisfait :

- 1 γ est **croissante**
 $\forall a, a', a \sqsubseteq a' \implies \gamma(a) \leq \gamma(a')$
- 2 α est **croissante**
 $\forall c, c', c \leq c' \implies \alpha(c) \sqsubseteq \alpha(c')$
- 3 $\gamma \circ \alpha$ est **extensive**
 $\forall c, c \leq \gamma(\alpha(c))$
- 4 $\alpha \circ \gamma$ est **réductrice**
 $\forall a, \alpha(\gamma(a)) \sqsubseteq a$

En particulier :

- les ordres abstraits et concrets sont compatibles (croissance)
- passer par l'abstrait est une sur-approximation vis-à-vis du concret
 ($\gamma \circ \alpha$ est extensive)

Quelques preuves sur les correspondances de Galois $\star\star$

Si $\forall a, c, \alpha(c) \sqsubseteq a \iff c \leq \gamma(a)$, alors :

1 $\gamma \circ \alpha$ est extensive : $\forall c, c \leq \gamma(\alpha(c))$

$\star\star$ preuve : $\alpha(c) \sqsubseteq \alpha(c) \implies c \leq \gamma(\alpha(c))$

2 $\alpha \circ \gamma$ est réductrice : $\forall a, \alpha(\gamma(a)) \sqsubseteq a$

3 α est croissante

$\star\star$ preuve : $c \leq c' \implies c \leq \gamma(\alpha(c')) \implies \alpha(c) \sqsubseteq \alpha(c')$

4 γ est croissante

5 $\gamma \circ \alpha \circ \gamma = \gamma$

$\star\star$ preuve :

$\alpha(\gamma(a)) \sqsubseteq \alpha(\gamma(a)) \implies \gamma(a) \leq \gamma(\alpha(\gamma(a)))$ et
 $a \sqsupseteq \alpha(\gamma(a)) \implies \gamma(a) \geq \gamma(\alpha(\gamma(a)))$

6 $\alpha \circ \gamma \circ \alpha = \alpha$

7 $\alpha \circ \gamma$ est idempotente : $\alpha \circ \gamma \circ \alpha \circ \gamma = \alpha \circ \gamma$

8 $\gamma \circ \alpha$ est idempotente : $\gamma \circ \alpha \circ \gamma \circ \alpha = \gamma \circ \alpha$

Meilleur abstraction, unicité des adjoints

Si $(C, \leq) \xleftrightarrow[\alpha]{\gamma} (A, \sqsubseteq)$

alors chaque adjoint peut être **défini de manière unique** en fonction de l'autre :

① $\alpha(c) = \sqcap \{ a \mid c \leq \gamma(a) \}$

② $\gamma(a) = \sqcup \{ c \mid \alpha(c) \sqsubseteq a \}$

Conséquence importante

$\alpha(c) = \sqcap \{ a \mid c \leq \gamma(a) \}$ signifie :

$\alpha(c)$ est la **meilleur abstraction** de c dans A

i.e., la sur-approximation la plus précise

* * Preuve : de 1

$\forall a, c \leq \gamma(a) \implies \alpha(c) \sqsubseteq a$, donc $\alpha(c)$ est un minorant de $\{ a \mid c \leq \gamma(a) \}$.

Supposons que a' est un autre minorant.

Alors, $\forall a, c \leq \gamma(a) \implies a' \sqsubseteq a$.

Par correspondance de Galois, il vient $\forall a, \alpha(c) \sqsubseteq a \implies a' \sqsubseteq a$.

Ceci implique : $a' \sqsubseteq \alpha(c)$.

Donc, le plus grand minorant de $\{ a \mid c \leq \gamma(a) \}$ existe, et vaut $\alpha(c)$.

La preuve de 2 est similaire.

Propriétés additionnelles des correspondances de Galois

Si $(\alpha : C \rightarrow A, \gamma : A \rightarrow C)$ est une correspondance de Galois, alors :

- 1 $\forall X \subseteq A$, si $\sqcap X$ existe, alors $\gamma(\sqcap X) = \wedge \{ \gamma(x) \mid x \in X \}$.
- 2 $\forall X \subseteq C$, si $\sqcup X$ existe, alors $\alpha(\sqcup X) = \sqcup \{ \alpha(x) \mid x \in X \}$

Conséquence importante

Le domaine abstrait est clos par conjonction

la conjonction de deux propriétés exprimables exactement dans l'abstrait est également exprimable exactement dans l'abstrait

* * Preuve : de 2

Par définition des lubs, $\forall x \in X, x \leq \sqcup X$.

Par croissance, $\forall x \in X, \alpha(x) \sqsubseteq \alpha(\sqcup X)$.

Or, $\alpha(\sqcup X)$ est un majorant de $\{ \alpha(x) \mid x \in X \}$.

Supposons que y soit un autre majorant de $\{ \alpha(x) \mid x \in X \}$.

Alors, $\forall x \in X, \alpha(x) \sqsubseteq y$.

Par correspondance de Galois, $\forall x \in X, x \leq \gamma(y)$.

Par définition des lubs, $\sqcup X \leq \gamma(y)$.

Par correspondance de Galois, $\alpha(\sqcup X) \sqsubseteq y$.

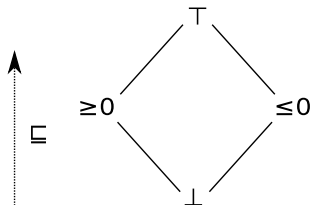
Donc $\{ \alpha(x) \mid x \in X \}$ a un lub, égal à $\alpha(\sqcup X)$.

La preuve de 1 est similaire.

Optimalité et clôture par conjonction de propriétés

Nous avons : $\gamma(a \sqcap a') = \gamma(a) \wedge \gamma(a')$

Contre-exemple : un domaine des signes imparfait



$$C \stackrel{\text{def}}{=} \mathcal{P}(\mathbb{Z})$$

dans C la conjonction de propriétés
est l'intersection d'ensembles d'entiers

$$A \stackrel{\text{def}}{=} \{\perp, \leq 0, \geq 0, \top\}$$

$$\gamma(\leq 0) \cap \gamma(\geq 0) = \{0\} \notin \gamma(A)$$

pas de meilleur abstraction pour $\{0\}$

\implies pas de correspondance de Galois

Corrections possibles :

- compléter A par \cap : $A \stackrel{\text{def}}{=} \{\perp, 0, \leq 0, \geq 0, \top\}$;
- vider A , en enlevant des éléments : $A \stackrel{\text{def}}{=} \{\perp, \geq 0, \top\}$;
- modifier des éléments : $A \stackrel{\text{def}}{=} \{\perp, < 0, \geq 0, \top\}$.

Opérations concrètes sur les ensembles entiers

Rappel : le monde concret est $C \stackrel{\text{def}}{=} \mathcal{P}(\mathbb{Z})$ (ensembles d'entiers)

But : définir les opérations sémantiques élémentaires sur C

Briques de base pour la définition de la sémantique concrète $E[\]$, $C[\]$, $S[\]$.

- opérations arithmétiques :

$+$, $-$, \times , $/$, étendues aux ensembles $(\mathcal{P}(\mathbb{Z}))^n \rightarrow \mathcal{P}(\mathbb{Z})$

$$-X \stackrel{\text{def}}{=} \{ -x \mid x \in X \}$$

$$X + Y \stackrel{\text{def}}{=} \{ x + y \mid x \in X, y \in Y \}$$

$$X - Y \stackrel{\text{def}}{=} \{ x - y \mid x \in X, y \in Y \}$$

$$X \times Y \stackrel{\text{def}}{=} \{ x \times y \mid x \in X, y \in Y \}$$

$$X / Y \stackrel{\text{def}}{=} \{ x / y \mid x \in X, y \in Y, y \neq 0 \}$$

- opérations ensemblistes : \cup , \cap

- relation d'ordre : \subseteq

- filtres : \leq , \geq , ... présentés plus tard

Abstraction d'opérateurs : abstraction sûre

Étant donnée une correspondance de Galois $(C, \leq) \xrightleftharpoons[\alpha]{\gamma} (A, \sqsubseteq)$

et un opérateur concret $F : C \rightarrow C$

comment **modéliser** F dans l'abstrait ?

c'est à dire, comme une fonction dans $A \rightarrow A$

Sûreté :

$a \in A$ est une **abstraction sûre** de $c \in C$ si $c \leq \gamma(a)$

ou, de manière équivalente : $\alpha(c) \sqsubseteq a$.

Donc, $F^\# : A \rightarrow A$ est une **abstraction sûre** de $F : C \rightarrow C$

si $\forall a: F(\gamma(a)) \leq \gamma(F^\#(a))$

ou, de manière équivalente : $\alpha(F(\gamma(a))) \sqsubseteq F^\#(a)$.

Nous le notons en raccourci : $F \circ \gamma \leq \gamma \circ F^\#$

un pas dans l'abstrait sur-approxime un pas dans le concret

Se généralise aux opérateurs n -aires : $F(\gamma(a_1), \dots, \gamma(a_n)) \leq \gamma(F^\#(a_1, \dots, a_n))$

Note : pour définir la sûreté, il nous suffit de la concrétisation γ

l'existence d'une abstraction α n'est pas nécessaire : pas de correspondance de Galois !

Abstraction d'opérateurs : abstraction optimale

Optimalité :

F^\sharp définie par $F^\sharp \stackrel{\text{def}}{=} \alpha \circ F \circ \gamma$ est **optimale**.

En effet, F^\sharp est sûre $\iff (\alpha \circ F \circ \gamma)(a) \sqsubseteq F^\sharp(a)$
donc $\alpha \circ F \circ \gamma$ est l'abstraction sûre de F la plus précise !

Conséquences :

- la sémantique abstraite F^\sharp peut être **dérivée systématiquement** de la sémantique concrète F ,
étant donnée une correspondance de Galois (α, γ) ;
- mais $\alpha \circ F \circ \gamma$ n'est qu'une définition mathématique,
encore faut-il trouver un **algorithme effectif** pour l'implanter...

Exemple : domaine des signes $\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\perp, 0, \leq 0, \geq 0, \top\}$

- dans le concret $X \overline{\times} Y \stackrel{\text{def}}{=} \{a \times b \mid a \in X, b \in Y\}$
- $\times^\sharp \stackrel{\text{def}}{=} \alpha(\gamma(X^\sharp) \overline{\times} \gamma(Y^\sharp))$ redonne la règle des signes
 $(\geq 0) \times^\sharp (\geq 0) = (\geq 0)$, $(\geq 0) \times^\sharp (\leq 0) = (\leq 0)$, $(\geq 0) \times^\sharp 0 = 0$, ...
- $X^\sharp /^\sharp Y^\sharp = \top$ est sûr, mais n'est pas optimal !

Abstraction d'opérateurs : abstraction exacte

Exactitude :

$F^\#$ est une abstraction exacte de F si $F \circ \gamma = \gamma \circ F^\#$
 \implies aucune perte à effectuer l'opération dans l'abstrait

$F^\#$ exacte implique $F^\#$ sûre et optimale
 mais les fonctions optimales ne sont pas toujours exactes !
 \implies l'abstraction α génère une perte d'information.

Exemples : domaine des signes alternatif $\mathcal{D}^\# \stackrel{\text{def}}{=} \{\perp, 0, < 0, > 0, \top\}$

- la règle des signes pour $\times^\#$ reste sûre, optimale, et exacte ;
- l'union abstraite $X^\# \cup^\# Y^\# \stackrel{\text{def}}{=} \alpha(\gamma(X^\#) \cup \gamma(Y^\#))$ est optimale
 mais pas exacte : $(> 0) \cup^\# (< 0) = \top$
 or $\gamma(> 0) \cup \gamma(< 0) = \mathbb{Z} \setminus \{0\} \neq \mathbb{Z}$!

Dans la plus part des domaines, $\cup^\#$ n'est pas exact. . .

Quizz : comment définir $\cap^\#$ optimale ? est-elle exacte ?

Abstraction d'opérateurs : composition

- la composition d'abstractions sûres est une abstraction sûre :

si F est croissante

et F^\sharp, G^\sharp abstraient F et G de manière sûre

alors $F^\sharp \circ G^\sharp$ est une abstraction sûre de $F \circ G$

Preuve : $\forall a, (F \circ G \circ \gamma)(a) \leq (F \circ \gamma \circ G^\sharp)(a) \leq (\gamma \circ F^\sharp \circ G^\sharp)(a)$

- la composition d'abstractions exactes est une abstraction exacte :

si $F \circ \gamma = \gamma \circ F^\sharp$ et $G \circ \gamma = \gamma \circ G^\sharp$,

alors $(F \circ G) \circ \gamma = \gamma \circ (F^\sharp \circ G^\sharp)$.

Principe

- réduire la sémantique concrète du langage à une composition d'un petit nombre d'opérations élémentaires
- abstraire chaque opération élémentaire
- composer les opérations abstraites pour obtenir une analyse

Abstraction d'opérateurs : composition (suite)

- la composition d'abstractions optimales est **sûre**
mais **pas forcément optimale**

$(\alpha \circ F \circ \gamma) \circ (\alpha \circ G \circ \gamma)$ n'est pas $\alpha \circ (F \circ G) \circ \gamma$
(le $\gamma \circ \alpha$ peut générer une perte de précision)

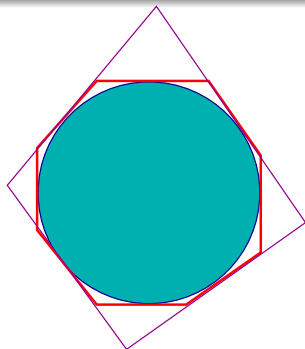
Exemple :

- dans $\mathcal{P}(\mathbb{Z})$,
prenons $F(X) \stackrel{\text{def}}{=} \{x + 1 \mid x \in X\}$ et $G(X) \stackrel{\text{def}}{=} \{x - 1 \mid x \in X\}$
on a donc $(G \circ F)(\{0\}) = \{0\}$;
- dans les signes :
 $F^\sharp(0) = (\alpha \circ F \circ \gamma)(0) = (\geq 0)$ et $G^\sharp(\geq 0) = (\alpha \circ G \circ \gamma)(\geq 0) = \top$
donc $(G^\sharp \circ F^\sharp)(0) = \top$;
- pourtant $(\alpha \circ (G \circ F) \circ \gamma)(0) = \alpha(\{0\}) = 0$!

Conclusion

La granularité des opérations élémentaire compte.
Une décomposition trop fine cause une perte de précision !

Absence de correspondance de Galois



Exemple : le domaine des polyèdres.

Il n'existe pas de meilleure sur-approximation d'un disque par un polygone.

nous pouvons toujours raffiner le polygone en ajoutant des arêtes

L'emploi d'un opérateur optimal $\alpha \circ F \circ \gamma$ n'est pas toujours possible :

- certains domaines abstraits n'ont **pas de correspondance de Galois** ;
- $\alpha \circ F \circ \gamma$ peut être **difficile** ou **coûteux** à implanter ;

Sans α , la notion de **sûreté**, $F \circ \gamma \leq \gamma \circ F^\#$ reste toujours utilisable !

\implies nous nous contentons alors d'**abstractions sûres**, non optimales.

En pratique, l'analyse statique par interprétation abstraite avec seulement γ est fréquente !

Analyse non-relationnelle

Abstraction des environnements

- nous avons vu des abstractions de $\mathcal{P}(\mathbb{Z})$;
- mais notre sémantique concrète $S[\![stat]\!]$ manipule des **ensembles d'environnements** $\mathcal{P}(\mathcal{E}) = \mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$.

Principe de l'analyse non-relationnelle :

- associer une valeur abstraite à chaque variable ;
(exemple : X est positif et Y est négatif)
- $\mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$ est abstrait par $\mathcal{E}^\# \stackrel{\text{def}}{=} \mathbb{V} \rightarrow \mathcal{D}^\#$
où $\mathcal{D}^\#$ est une abstraction arbitraire de $\mathcal{P}(\mathbb{Z})$
(exemple : $\mathcal{D}^\#$ est le domaine des signes)

Les opérations sur $\mathcal{E}^\#$ seront **systématiquement dérivées** des opérations sur $\mathcal{D}^\#$.

Abstraction cartésienne

Tout domaine non-relationnel **oublie les relations** entre variables.

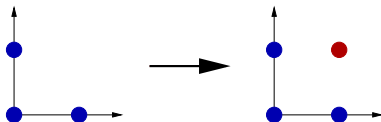
Le domaine non-relationnel le plus précis serait $\mathcal{E}^\sharp \stackrel{\text{def}}{=} \mathbb{V} \rightarrow \mathcal{P}(\mathbb{Z})$:

- abstraction variable par variable
- mais pas (encore) d'abstraction sur l'ensemble des valeurs possibles

alors : $\mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z}) \xleftrightarrow[\alpha]{\gamma} \mathbb{V} \rightarrow \mathcal{P}(\mathbb{Z})$

- $\alpha(R) \stackrel{\text{def}}{=} \lambda V \in \mathbb{V}. \{ \rho(V) \mid \rho \in R \}$
- $\gamma(R^\sharp) \stackrel{\text{def}}{=} \{ \rho \mid \forall V \in \mathbb{V} : \rho(V) \in R^\sharp(V) \}$
- $(\gamma \circ \alpha)(R) = \{ \rho \mid \forall V \in \mathbb{V} : \exists \rho_V \in R : \rho(V) = \rho_V(V) \} \supseteq R$
 \implies forme de nouveaux environnements en mélangeant des environnements de R

Exemple : $(\gamma \circ \alpha)(\{(X, Y) \mid X \in \{0, 2\}, Y \in \{0, 2\}, X + Y \leq 2\}) = \{0, 2\} \times \{0, 2\}$.



Signature de l'abstraction des valeurs



Abstraction de valeurs $\mathcal{D}^\#$:

$\mathcal{D}^\#$	ensemble de valeurs abstraites représentables en mémoire
$\gamma : \mathcal{D}^\# \rightarrow \mathcal{P}(\mathbb{Z})$	concretisation
$\alpha : \mathcal{P}(\mathbb{Z}) \rightarrow \mathcal{D}^\#$	abstraction (optionnelle)
\sqsubseteq	ordre partiel compatible avec γ (γ croissant)
\perp, \top	représentation de \emptyset et \mathbb{Z}
$+\#, -\#, \times\#, /\#$	abstractions sûres de $\overline{+}, \overline{-}, \overline{\times}, \overline{/}$
$c\#, [a, b]^\#$	abstractions sûres de $\{c\}, [a, b]$
$\cup\#, \cap\#$	abstractions sûres de \cup et \cap

Dérivation systématique de $\mathcal{D}^\#$

$\mathcal{E}^\#$ est défini par :

- $\mathcal{E}^\# \stackrel{\text{def}}{=} \mathbb{V} \rightarrow \mathcal{D}^\#$
- ordre **point à point** : $X_1^\# \dot{\subseteq} X_2^\# \iff \forall V \in \mathbb{V} : X_1^\#(V) \subseteq X_2^\#(V)$
- union : $X_1^\# \dot{\cup}^\# X_2^\# \stackrel{\text{def}}{=} \lambda V. X_1^\#(V) \cup^\# X_2^\#(V)$
- intersection : $X_1^\# \dot{\cap}^\# X_2^\# \stackrel{\text{def}}{=} \lambda V. X_1^\#(V) \cap^\# X_2^\#(V)$
- $\dot{\perp}(V) \stackrel{\text{def}}{=} \lambda V. \perp$
- $\dot{\top}(V) \stackrel{\text{def}}{=} \lambda V. \top$

Note : la structure de $\mathcal{D}^\#$ se retrouve sur $\mathcal{E}^\#$

- $(\mathcal{E}^\#, \dot{\subseteq})$ est un ordre partiel
- si $\mathcal{D}^\#$ est un treillis (complet), alors $\mathcal{E}^\#$ est aussi un treillis complet
- on peut définir une correspondance de Galois sur $\mathcal{E}^\#$:
 - $(\mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z}), \subseteq) \xleftrightarrow[\alpha]{\hat{\gamma}} (\mathbb{V} \rightarrow \mathcal{D}^\#, \dot{\subseteq})$
 - $\dot{\alpha}(E) \stackrel{\text{def}}{=} \lambda V. \alpha(\{\rho(V) \mid \rho \in R\})$
 - $\dot{\gamma}(X^\#) \stackrel{\text{def}}{=} \{\rho \mid \forall V \in \mathbb{V} : \rho(V) \in \gamma(X^\#(V))\}$

Notation : un point ' sert à distinguer les opérations sur $\mathcal{D}^\#$ de celles sur $\mathbb{V} \rightarrow \mathcal{D}^\#$.

Sémantique abstraite des expressions

$E^\# \llbracket \text{expr} \rrbracket : \mathcal{E}^\# \rightarrow \mathcal{D}^\#$ est une abstraction de $E \llbracket \text{expr} \rrbracket : \mathcal{E} \rightarrow \mathcal{P}(\mathbb{Z})$

$E^\# \llbracket V \rrbracket X^\#$	$\stackrel{\text{def}}{=} X^\#(V)$
$E^\# \llbracket c \rrbracket X^\#$	$\stackrel{\text{def}}{=} c^\#$
$E^\# \llbracket \text{rand}(a, b) \rrbracket X^\#$	$\stackrel{\text{def}}{=} [a, b]^\#$
$E^\# \llbracket -e \rrbracket X^\#$	$\stackrel{\text{def}}{=} -^\# E^\# \llbracket e \rrbracket X^\#$
$E^\# \llbracket e_1 + e_2 \rrbracket X^\#$	$\stackrel{\text{def}}{=} E^\# \llbracket e_1 \rrbracket X^\# +^\# E^\# \llbracket e_2 \rrbracket X^\#$
$E^\# \llbracket e_1 - e_2 \rrbracket X^\#$	$\stackrel{\text{def}}{=} E^\# \llbracket e_1 \rrbracket X^\# -^\# E^\# \llbracket e_2 \rrbracket X^\#$
$E^\# \llbracket e_1 \times e_2 \rrbracket X^\#$	$\stackrel{\text{def}}{=} E^\# \llbracket e_1 \rrbracket X^\# \times^\# E^\# \llbracket e_2 \rrbracket X^\#$
$E^\# \llbracket e_1 / e_2 \rrbracket X^\#$	$\stackrel{\text{def}}{=} E^\# \llbracket e_1 \rrbracket X^\# /^\# E^\# \llbracket e_2 \rrbracket X^\#$

Définition par induction sur la syntaxe en suivant le modèle de $E \llbracket \text{expr} \rrbracket$ mais dans l'abstrait !

Sûreté

$\bigcup_{\rho \in \gamma(X^\#)} E \llbracket e \rrbracket \rho \subseteq \gamma(E^\# \llbracket e \rrbracket X^\#)$
 par composition de la sûreté de $+^\#, -^\#, \text{etc.}$

Sémantique abstraite (partielle)

$S^\# \llbracket \text{stat} \rrbracket : \mathcal{E}^\# \rightarrow \mathcal{E}^\#$ version abstraite de $S \llbracket \text{stat} \rrbracket : \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$

- $S \llbracket \text{skip} \rrbracket R \stackrel{\text{def}}{=} R$
 $S^\# \llbracket \text{skip} \rrbracket X^\# \stackrel{\text{def}}{=} X^\#$ (identité)
- $S \llbracket \text{halt} \rrbracket R \stackrel{\text{def}}{=} \emptyset$
 $S^\# \llbracket \text{halt} \rrbracket X^\# \stackrel{\text{def}}{=} \perp$ (arrêt)
- $S \llbracket s_1; s_2 \rrbracket R \stackrel{\text{def}}{=} S \llbracket s_2 \rrbracket (S \llbracket s_1 \rrbracket R)$
 $S^\# \llbracket s_1; s_2 \rrbracket X^\# \stackrel{\text{def}}{=} S^\# \llbracket s_2 \rrbracket (S^\# \llbracket s_1 \rrbracket X^\#)$ (composition)
- $S \llbracket V \leftarrow e \rrbracket R \stackrel{\text{def}}{=} \{ \rho[V \mapsto v] \mid \rho \in R, v \in E \llbracket e \rrbracket \rho \}$
 $S^\# \llbracket V \leftarrow e \rrbracket X^\# \stackrel{\text{def}}{=} \begin{cases} X^\# [V \mapsto E^\# \llbracket e \rrbracket X^\#] & \text{si } E^\# \llbracket e \rrbracket X^\# \neq \perp \\ \perp & \text{si } E^\# \llbracket e \rrbracket X^\# = \perp \end{cases}$

Réduction : si $E^\# \llbracket e \rrbracket X^\# = \perp$, alors $S \llbracket V \leftarrow e \rrbracket$ retourne \emptyset
 \implies nous renvoyons donc \perp , qui est la représentation la plus précise de \emptyset pour \sqsubseteq

Sémantique abstraite (partielle) : tests

$C^\# \llbracket \text{cond} \rrbracket : \mathcal{E}^\# \rightarrow \mathcal{E}^\#$ version abstraite de $C \llbracket \text{cond} \rrbracket : \mathcal{P}(\mathcal{E}) \rightarrow \mathcal{P}(\mathcal{E})$

Cas de base et par induction :

- $C^\# \llbracket \text{true} \rrbracket X^\# \stackrel{\text{def}}{=} \top$
- $C^\# \llbracket \text{false} \rrbracket X^\# \stackrel{\text{def}}{=} \perp$
- $C^\# \llbracket c_1 \vee c_2 \rrbracket X^\# \stackrel{\text{def}}{=} (C^\# \llbracket c_1 \rrbracket X^\#) \dot{\cup}^\# (C^\# \llbracket c_2 \rrbracket X^\#)$
- $C^\# \llbracket c_1 \wedge c_2 \rrbracket X^\# \stackrel{\text{def}}{=} (C^\# \llbracket c_1 \rrbracket X^\#) \dot{\cap}^\# (C^\# \llbracket c_2 \rrbracket X^\#)$

La comparaison d'expressions $C^\# \llbracket e_1 \bowtie e_2 \rrbracket$, $\bowtie \in \{=, \neq, <, >, \leq, \geq\}$ sera présentée plus tard sur des exemples...

Sûreté

$$C \llbracket c \rrbracket \gamma(X^\#) \subseteq \gamma(C^\# \llbracket c \rrbracket X^\#)$$

Sémantique abstraite (partielle)

$$\bullet \ S \llbracket \text{assert } c \rrbracket R \stackrel{\text{def}}{=} C \llbracket c \rrbracket R$$

$$S^\# \llbracket \text{assert } c \rrbracket X^\# \stackrel{\text{def}}{=} C^\# \llbracket c \rrbracket X^\#$$

$$\bullet \ S \llbracket \text{if } c \text{ then } s_1 \text{ else } s_2 \rrbracket R \stackrel{\text{def}}{=} S \llbracket s_1 \rrbracket (C \llbracket c \rrbracket R) \cup S \llbracket s_2 \rrbracket (C \llbracket \neg c \rrbracket R)$$

$$S^\# \llbracket \text{if } c \text{ then } s_1 \text{ else } s_2 \rrbracket X^\# \stackrel{\text{def}}{=} S^\# \llbracket s_1 \rrbracket (C^\# \llbracket c \rrbracket X^\#) \dot{\cup} S^\# \llbracket s_2 \rrbracket (C^\# \llbracket \neg c \rrbracket X^\#)$$

$$\bullet \ S \llbracket \text{while } c \text{ do } s \rrbracket R \stackrel{\text{def}}{=} C \llbracket \neg c \rrbracket (\text{lfp } \lambda X. R \cup S \llbracket s \rrbracket (C \llbracket c \rrbracket X))$$

pour l'instant, nous pouvons écrire naïvement

$$S^\# \llbracket \text{while } c \text{ do } s \rrbracket X^\# \stackrel{\text{def}}{=} C^\# \llbracket \neg c \rrbracket (\text{lfp } \lambda Y^\#. X^\# \cup S^\# \llbracket s \rrbracket (C^\# \llbracket c \rrbracket Y^\#))$$

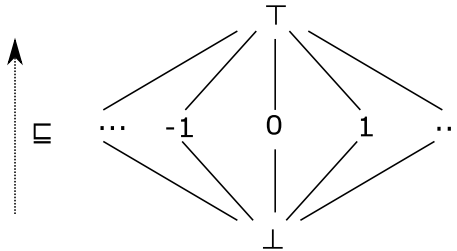
le prochain cours expliquera le traitement des boucles en détail. . .

Sûreté

$$S \llbracket c \rrbracket \gamma(X^\#) \subseteq \gamma(S^\# \llbracket c \rrbracket X^\#)$$

Le domaine des constantes

Treillis des constantes



Propriétés abstraites

Valeurs possibles d'une variable à un point de programme donné :

- $c \in \mathbb{Z}$: constante numérique (une seule valeur possible) ;
- \top : non constante (plusieurs valeurs possibles) ;
- \perp : code non accessible (aucune valeur possible) ;

Treillis complet, infini en largeur, mais « plat ».

Opérations sur les constantes

Correspondance de Galois :

$$\begin{aligned} \gamma(\perp) &\stackrel{\text{def}}{=} \emptyset \\ \gamma(c) &\stackrel{\text{def}}{=} \{c\} \\ \gamma(\top) &\stackrel{\text{def}}{=} \mathbb{Z} \end{aligned} \qquad \alpha(S) \stackrel{\text{def}}{=} \begin{cases} \perp & \text{si } S = \emptyset \\ c & \text{si } S = \{c\} \\ \top & \text{sinon} \end{cases}$$

Opérateurs optimaux dérivés :

- $\cup^\#$ et $\cap^\#$ sont \sqcup et \sqcap pour l'ordre partiel ;
- $c^\# \stackrel{\text{def}}{=} c$;
- $[a, b]^\# \stackrel{\text{def}}{=} a$ si $a = b$, \top sinon ;
- $X^\# +^\# Y^\# \stackrel{\text{def}}{=} \begin{cases} \perp & \text{si } X^\# \text{ ou } Y^\# = \perp \\ \top & \text{sinon si } X^\# \text{ ou } Y^\# = \top \\ X^\# + Y^\# & \text{sinon} \end{cases}$
- $X^\# \times^\# Y^\# \stackrel{\text{def}}{=} \begin{cases} \perp & \text{si } X^\# \text{ ou } Y^\# = \perp \\ 0 & \text{sinon si } X^\# \text{ ou } Y^\# = 0 \\ \top & \text{sinon si } X^\# \text{ ou } Y^\# = \top \\ X^\# \times Y^\# & \text{sinon} \end{cases}$
- ...

Opérations sur les constantes (suite)

Exemples de test :

- $$C^\# \llbracket X - c = 0 \rrbracket R^\# \stackrel{\text{def}}{=} \begin{cases} \perp & \text{si } R^\#(X) \notin \{c, \top\} \\ R^\#[X \mapsto c] & \text{sinon} \end{cases}$$
- $$C^\# \llbracket X - Y - c = 0 \rrbracket R^\# \stackrel{\text{def}}{=} \left(\begin{cases} C^\# \llbracket X - (R^\#(Y) + c) = 0 \rrbracket R^\# & \text{si } R^\#(Y) \notin \{\perp, \top\} \\ R^\# & \text{sinon} \end{cases} \right) \dot{\cap}^\# \left(\begin{cases} C^\# \llbracket Y - (R^\#(X) - c) = 0 \rrbracket R^\# & \text{si } R^\#(X) \notin \{\perp, \top\} \\ R^\# & \text{sinon} \end{cases} \right)$$

une valeur constante détermine l'autre valeur constante

- Note :

$C^\# \llbracket c \rrbracket R^\# \stackrel{\text{def}}{=} R^\#$ est toujours un choix possible, sûr mais peu précis.

Exemple d'analyse

Exemple :

```

X ← 0; Y ← 10;
while X < 100 do
  Y ← Y - 3;
  X ← X + Y; •
  Y ← Y + 3
done

```

Nous supposons ici que le **while** est calculé avec des **itérations de Kleene**, comme dans la sémantique concrète.

(c.f. cours suivant pour l'analyse détaillée des boucles)

L'analyse dans le domaine des constante trouve à • : $\begin{cases} X = \textcolor{red}{T} \\ Y = \textcolor{blue}{7} \end{cases}$

Note :

l'analyse découvre des constantes **qui n'apparaissent pas syntaxiquement** dans le programme ; nous avons une **analyse sémantique**.

Le domaine des intervalles

Les intervalles entiers

Idée :

abstraire les comportements du programme
par une borne supérieure et une borne inférieure pour chaque variable.

$$\mathcal{D}^\# \stackrel{\text{def}}{=} \{ [a, b] \mid a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{+\infty\}, a \leq b \} \cup \{\perp\}$$

- Les valeurs de borne $-\infty$, $+\infty$ sont nécessaires ;
elles permettent de représenter des ensembles non-bornés ;

$[-\infty, +\infty] = \top$ représente \mathbb{Z} ,

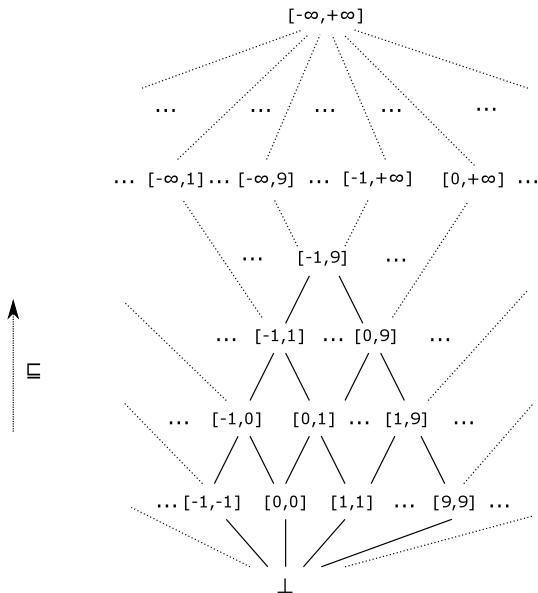
$[0, +\infty]$ représente \mathbb{N} , etc.

\implies tout ensemble d'entiers a une sur-approximation dans $\mathcal{D}^\#$

au pire, cette sur-approximation est $\top = [-\infty, +\infty]$

- \perp est l'unique représentant de \emptyset
dans $[a, b]$, nous imposons $a \leq b$; les intervalles non- \perp ne sont jamais vides

Le treillis des intervalles



Structure algébrique

Ordre partiel : \sqsubseteq

- $\forall I \in \mathcal{D}^\# : \perp \sqsubseteq I$
- $[a, b] \sqsubseteq [c, d] \iff (a \geq c) \wedge (b \leq d)$

où \leq est étendu naturellement à $\mathbb{Z} \cup \{-\infty, +\infty\}$ par : $\forall c \in \mathbb{Z} : -\infty < c < +\infty$

Treillis : \sqcup, \sqcap

- plus petit majorant \sqcup pour \sqsubseteq
 - $\forall I \in \mathcal{D}^\# : \perp \sqcup I = I \sqcup \perp = I$
 - $[a, b] \sqcup [c, d] = [\min(a, c), \max(b, d)]$
- plus grand minorant \sqcap :
 - $\forall I : \perp \sqcap I = I \sqcap \perp = \perp$
 - $[a, b] \sqcap [c, d] = \begin{cases} [\max(a, c), \min(b, d)] & \text{si } \max(a, c) \leq \min(b, d) \\ \perp & \text{si } \max(a, c) > \min(b, d) \end{cases}$

Structure algébrique (suite)

Notes :

- le treillis est **complet** ;

$\forall I \subseteq \mathcal{D}^\# : \sqcup I$ et $\sqcap I$ existent

$$\sqcup \{ [a_j, b_j] \mid j \in J \} = [\min_{j \in J} a_j, \max_{j \in J} b_j]$$

$$\sqcap \{ [a_j, b_j] \mid j \in J \} = [\max_{j \in J} a_j, \min_{j \in J} b_j] \text{ si } \max \leq \min, \text{ ou } \perp \text{ sinon}$$

- $\cap^\# = \sqcap$

l'ensemble des intervalles est **clos par \cap** ;

$$[a, b] \cap [c, d] = [a, b] \sqcap [c, d]$$

- $\cup^\# = \sqcup$


l'ensemble des intervalles n'est **pas clos par \cup** .

$$[0, 0] \cup [2, 2] = \{0, 2\}, \text{ qui n'est pas un intervalle ;}$$

$$[0, 0] \sqcup [2, 2] = [0, 2]$$

\Rightarrow perte de précision potentielle dans l'analyse !

Correspondance de Galois pour les intervalles

Correspondance de Galois : $(\mathcal{P}(\mathbb{Z}), \subseteq) \xrightleftharpoons[\alpha]{\gamma} (\mathcal{D}^\#, \sqsubseteq)$ 

- $\begin{cases} \gamma(\perp) \stackrel{\text{def}}{=} \emptyset \\ \gamma([a, b]) \stackrel{\text{def}}{=} \{x \in \mathbb{Z} \mid a \leq x \leq b\} \end{cases}$
- $\alpha(X) \stackrel{\text{def}}{=} \begin{cases} \perp & \text{si } X = \emptyset \\ [\min X, \max X] & \text{si } X \neq \emptyset \end{cases}$

**Preuve : on a bien $\alpha(X) \sqsubseteq I \iff X \subseteq \gamma(I)$, car

$$\begin{aligned}
 & \alpha(X) \sqsubseteq (a, b) \\
 \iff & \min X \geq a \wedge \max X \leq b && (\text{def. } \alpha, \sqsubseteq) \\
 \iff & \forall x \in X: a \leq x \leq b && (\text{def. min, max}) \\
 \iff & \forall x \in X: x \in \{y \mid a \leq y \leq b\} \\
 \iff & \forall x \in X: x \in \gamma([a, b]) && (\text{def. } \gamma) \\
 \iff & X \subseteq \gamma([a, b]) && (\text{prop. } \subseteq)
 \end{aligned}$$

Arithmétique d'intervalles : addition, soustraction

- $-^\# [a, b] = [-b, -a]$
- $[a, b] +^\# [c, d] = [a + c, b + d]$
- $[a, b] -^\# [c, d] = [a - d, b - c]$
- $\forall I \in \mathcal{D}^\# : -^\# \perp = \perp +^\# I = I +^\# \perp = \dots = \perp$
les opérateurs sont stricts : ils retournent \perp si un argument est \perp

où $+$ et $-$ sont étendus aux bornes $+\infty$ et $-\infty$ par :

$\forall x \in \mathbb{Z} : (+\infty) + x = +\infty, (-\infty) + x = -\infty, -(+\infty) = (-\infty), \dots$

**Preuve : optimalité de $+^\#$

$$\begin{aligned}
 & \alpha(\gamma([a, b]) \overline{+} \gamma([c, d])) \\
 &= \alpha(\{x \mid a \leq x \leq b\} \overline{+} \{y \mid c \leq y \leq d\}) \\
 &= \alpha(\{x + y \mid a \leq x \leq b \wedge c \leq y \leq d\}) \\
 &= [\min \{x + y \mid a \leq x \leq b \wedge c \leq y \leq d\}, \max \{x + y \mid a \leq x \leq b \wedge c \leq y \leq d\}] \\
 &= [a + c, b + d] \\
 &= [a, b] +^\# [c, d]
 \end{aligned}$$

Quizz : $+^\#$, $-^\#$ sont-ils exacts ?

Arithmétique d'intervalles : multiplication

- $$[a, b] \times^\# [c, d] = [\min(a \times c, a \times d, b \times c, b \times d), \max(a \times c, a \times d, b \times c, b \times d)]$$

où \times est étendu aux bornes $+\infty$ et $-\infty$ par la règle des signes :

$$c \times (+\infty) = (+\infty) \text{ si } c > 0, (-\infty) \text{ si } c < 0$$

$$c \times (-\infty) = (-\infty) \text{ si } c > 0, (+\infty) \text{ si } c < 0$$

et aussi la règle non-standard : $0 \times (+\infty) = 0 \times (-\infty) = 0$

e.g., $[0, +\infty] \times^\# [0, 0] = [0, 0]$, grâce à $+\infty \times 0 = 0 \times 0 = 0$

Quizz : $\times^\#$ est-il exact ? optimal ?

Arithmétique d'intervalles : division

- $/^\#$ **par cas** :

$$([a, b] /^\# ([c, d] \cap^\# [1, +\infty))) \cup^\# ([a, b] /^\# ([c, d] \cap^\# [-\infty, -1]))$$

où

$$[a, b] /^\# [c, d] = \begin{cases} [\min(a/c, a/d), \max(b/c, b/d)] & \text{si } 1 \leq c \\ [\min(b/c, b/d), \max(a/c, a/d)] & \text{si } d \leq -1 \end{cases}$$

et $/$ est étendu aux bornes $+\infty$ et $-\infty$ par la **règle des signes** :

$$c/(+\infty) = c/(-\infty) = 0, \text{ en particulier } (+\infty)/(+\infty) = 0$$

$$(+\infty)/c = (+\infty) \text{ si } c > 0, (-\infty) \text{ si } c < 0$$

$$(-\infty)/c = (-\infty) \text{ si } c > 0, (+\infty) \text{ si } c < 0$$

Exemples :

$$[-5, 5] /^\# [0, 0] = \perp$$

$$[5, 10] /^\# [-1, 1] = ([5, 10] /^\# [1, 1]) \cup^\# ([5, 10] /^\# [-1, -1]) = [5, 10] \cup^\# [-10, -5] = [-10, 10]$$

Test dans les intervalles : cas particuliers simples

Tests simples : comparaison entre variables et constantes

on note ici : $X^\#(V) = [a, b]$ et $X^\#(W) = [c, d]$

$$\bullet C^\# \llbracket V \leq x \rrbracket X^\# \stackrel{\text{def}}{=} \begin{cases} X^\# [V \mapsto [a, \min(b, x)]] & \text{si } a \leq v \\ \perp & \text{si } a > v \end{cases}$$

$$\bullet C^\# \llbracket V \leq W \rrbracket X^\# \stackrel{\text{def}}{=} \begin{cases} X^\# \left[\begin{array}{l} V \mapsto [a, \min(b, d)], \\ W \mapsto [\max(a, c), d] \end{array} \right] & \text{si } a \leq d \\ \perp & \text{si } a > d \end{cases}$$

la borne supérieure de W raffine la borne supérieure de V
la borne inférieure de V raffine la borne inférieure de W

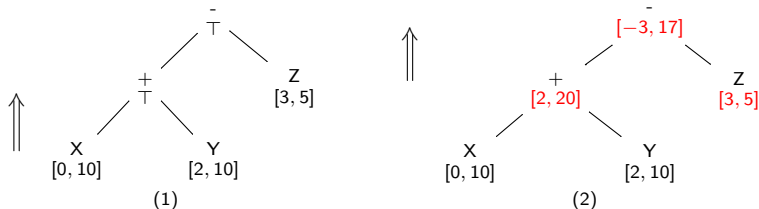


Test dans les intervalles : cas complexe (étape 1) [★]_{★★}

Exemple : $C^\sharp \llbracket X + Y - Z \leq 0 \rrbracket X^\sharp$

où $X^\sharp = \{ X \mapsto [0, 10], Y \mapsto [2, 10], Z \mapsto [3, 5] \}$

Première étape : **annoter** l'arbre d'expression avec des intervalles



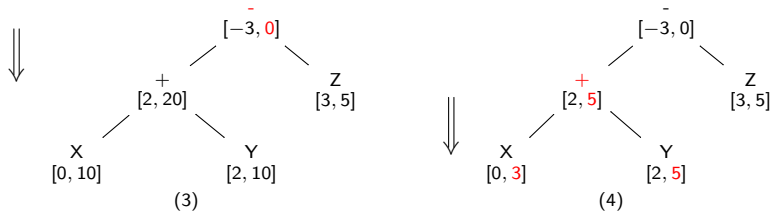
Évaluation *bottom-up* similaire à l'affectation d'intervalles,
 en utilisant les opérateurs abstraits $+^\sharp$, $-^\sharp$, etc.
 mais on se souvient en plus de tous les résultats intermédiaires.

Test dans les intervalles : cas complexe (étape 2) [★]★★

Exemple : $C^\sharp \llbracket X + Y - Z \leq 0 \rrbracket X^\sharp$

avec $X^\sharp = \{ X \mapsto [0, 10], Y \mapsto [2, 10], Z \mapsto [3, 5] \}$

Deuxième étape : **raffinement** de l'expression *top-down*.



- raffine l'intervalle à la **racine**, sachant que le résultat est négatif ;
- **propage** les intervalles raffinés **vers le bas**, vers les feuilles ;
- les intervalles aux **feuilles variables** fournissent des nouvelles informations, à stocker dans l'environnement abstrait :
 $\{ X \mapsto [0, 3], Y \mapsto [2, 5], Z \mapsto [3, 5] \}$

Opérateurs arithmétiques et de comparaison en arrière [★]_{★★}

Les opérateurs en arrière qui **raffinent de manière sûre** leurs arguments.

- Opérateur de comparaison, appliqué à la racine de l'expression :

$$\leq^{\#} 0^{\#}(X^{\#}) \stackrel{\text{def}}{=} X^{\#} \cap^{\#} [-\infty, 0]^{\#}$$

- Opérateurs arithmétiques en arrière, appliqués aux nœuds :

$$\leq^{\#}(X^{\#}, R^{\#}) \stackrel{\text{def}}{=} X^{\#} \cap^{\#} (-^{\#} R^{\#})$$

$$\text{car } R = -X \implies X = -R$$

$$\neq^{\#}(X^{\#}, Y^{\#}, R^{\#}) \stackrel{\text{def}}{=} (X^{\#} \cap^{\#} (R^{\#} -^{\#} Y^{\#}), Y^{\#} \cap^{\#} (R^{\#} -^{\#} X^{\#}))$$

$$\text{car } R = X + Y \implies (X = R - Y) \wedge (Y = R - X)$$

$$\leq^{\#}(X^{\#}, Y^{\#}, R^{\#}) \stackrel{\text{def}}{=} (X^{\#} \cap^{\#} (R^{\#} +^{\#} Y^{\#}), Y^{\#} \cap^{\#} (X^{\#} -^{\#} R^{\#}))$$

$$\leq^{\#}(X^{\#}, Y^{\#}, R^{\#}) \stackrel{\text{def}}{=} (X^{\#} \cap^{\#} (R^{\#} /^{\#} Y^{\#}), Y^{\#} \cap^{\#} (R^{\#} /^{\#} X^{\#}))$$

$$\leq^{\#}(X^{\#}, Y^{\#}, R^{\#}) \stackrel{\text{def}}{=} (X^{\#} \cap^{\#} (S^{\#} \times^{\#} Y^{\#}), Y^{\#} \cap^{\#} ((X^{\#} /^{\#} S^{\#}) \cup^{\#} [0, 0]^{\#}))$$

où $S^{\#} = R^{\#} +^{\#} [-1, 1]^{\#}$, car $/$ tronque le résultat !

Étant donnés des arguments $X^{\#}$, $Y^{\#}$ originaux, et un résultat $R^{\#}$ raffiné,

$\leq^{\#}$ retourne $X^{\#}$ et $Y^{\#}$ qui raffinent $X^{\#}$ et $Y^{\#}$,

tout en couvrant le résultat raffiné $R^{\#}$: $R^{\#} \subseteq X^{\#} \leq^{\#} Y^{\#}$ (sûreté).

Implantation de la sémantique abstraite en OCaml

Rappel : itérateur générique et domaine concret

L'**itérateur** est générique

et paramétré par un **domaine d'interprétation** :

- foncteur `Interprete` dans `interpreter/interpreter.ml`
- paramétré par un domaine obéissant à la signature `DOMAIN` dans `domains/domain.ml`.
abstraction de $\mathcal{P}(\mathbb{V} \rightarrow \mathbb{Z})$

Dans le cours précédent, nous avons vu l'interprète concret :

- module `Concrete` dans `domains/concrete_domain.ml` qui implante `DOMAIN`.

Rappel : schéma de fonctionnement de l'interprète concret

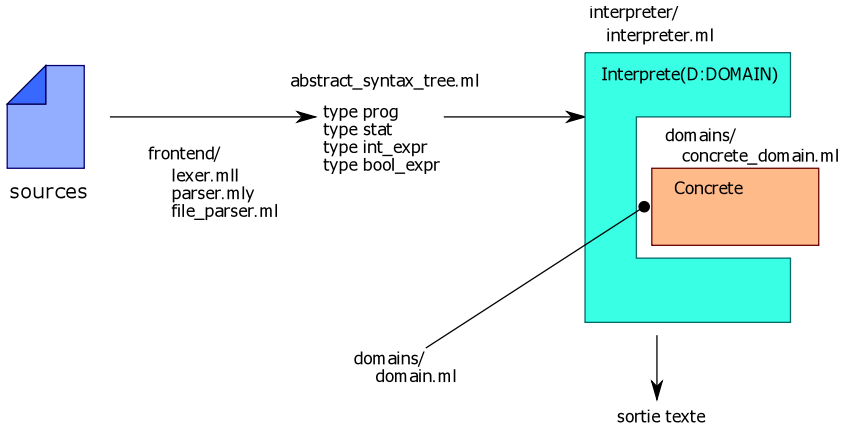
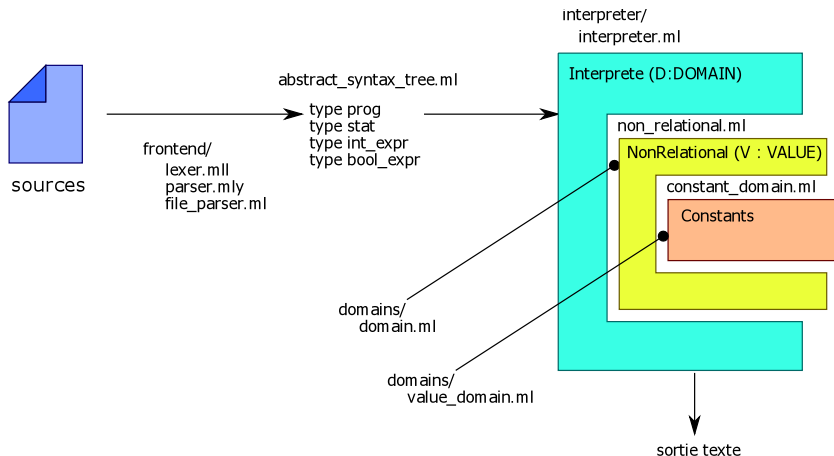


Schéma de fonctionnement de l'interprète non-relationnel



Analyse non-relationnelle

- pas de changement à l'itérateur pour l'instant...
- ajout d'une signature de **domaines pour les valeurs**, fournie
 - **VALUE_DOMAIN** dans `domains/value_domain.ml` ;
 - modélise des ensembles d'entiers $\mathcal{P}(\mathbb{Z})$.
- exemple de domaines de valeurs :
 - **Constants** dans `domains/constant_domain.ml`
domaine des **constantes** (implantation partielle fournie, à compléter)
 - domaine des **intervalles** : non fourni, à faire intégralement !
- un **foncteur non-relationnel générique**, fourni :
 - **NonRelational** dans `domains/non_relational` ;
 - prend un domaine de valeurs **VALUE** en argument ;
 - fournit un domaine d'environnements **DOMAIN** avec la méthode vue en cours.

L'ajout d'une analyse non-relationnelle nécessite uniquement l'ajout d'un nouveau domaine de valeurs, de signature **VALUE_DOMAIN** :

⇒ plus simple que d'ajouter un nouveau module de signature **DOMAIN** !

Domaine de valeurs : signature

domains/value_domain.ml

```

module type VALUE_DOMAIN = sig
  type t                                (*  $\mathcal{P}(\mathbb{Z})$  *)

  val top: t                            (*  $\top$  *)
  val bottom: t                         (*  $\perp$  *)
  val const: Z.t -> t                  (*  $\{c\}$  *)
  val rand: Z.t -> Z.t -> t           (*  $[a, b]$  *)

  val join: t -> t -> t                (*  $\cup$  *)
  val meet: t -> t -> t                (*  $\cap$  *)
  val subset: t -> t -> bool           (*  $\subseteq$  *)
  val is_bottom: t -> bool             (*  $= \emptyset ?$  *)

  val print: Format.formatter -> t -> unit

  val unary: t -> int_unary_op -> t    (*  $-$  *)
  val binary: t -> t -> int_binary_op -> t    (*  $+, -, \times, /$  *)
  val compare: t -> t -> compare_op -> (t * t)    (*  $=, \neq, <, >, \leq, \geq$  *)

  (* pour les tests complexes, optionnel *)
  val bwd_unary: t -> int_unary_op -> t -> t
  val bwd_binary: t -> t -> int_binary_op -> t -> (t * t)

  (* pour les boucles, voir le prochain cours *)
  val widen: t -> t -> t
end

```

Domaine des constantes : représentation et ordre

domains/constant_domain.ml

```

module Constants = struct
  type t = Cst of Z.t (* c *)
          | BOT        (*  $\perp$  *)
          | TOP        (*  $\top$  *)

  let top      = TOP
  let bottom   = BOT
  let const c  = Cst c
  let rand x y = if x=y then Cst x else if x<y then TOP else BOT

  let subset a b = match a,b with
  | BOT,_ | _,TOP -> true
  | Cst x, Cst y -> x=y
  | _ -> false

  let join a b = match a,b with
  | BOT,x | x,BOT -> x
  | Cst x, Cst y when x=y -> a
  | _ -> TOP

  let meet a b = match a,b with
  | TOP,x | x,TOP -> x
  | Cst x, Cst y when x=y -> a
  | _ -> BOT

  ...
end: VALUE_DOMAIN

```

Domaine des constantes : opérateurs

domains/constant_domain.ml

```

(* applique f à une constante,  $\perp \rightarrow \perp$ ,  $\top \rightarrow \top$  *)
let lift1 f x = match x with
  | BOT -> BOT
  | TOP -> TOP
  | Cst a -> Cst (f a)

(* idem pour une fonction binaire*)
let lift2 f x y = ...

(* ces opérations sont-elles optimales ? *)
let neg = lift1 Z.neg
let add = lift2 Z.add
let sub = lift2 Z.sub
let mul = lift2 Z.mul
let div a b = if b = Cst Z.zero then BOT else lift2 Z.div a b

(* à faire *)
let eq a b = a,b
...

(* dispatch *)
let unary x op = match op with ...
let binary x y op = match op with ...
let compare x y op = match op with ...

(* tests complexes, à lire chez soi *)
let bwd_unary = ...

```

Domaine non-relationnel générique : représentation

domains/non_relational.ml

```

module NonRelational(V:VALUE_DOMAIN) =
struct

  module VarMap = Mapext.Make(String)
  type env = V.t VarMap.t  (*  $\mathbb{V} \rightarrow \mathcal{D}_{\#}$  *)

  type t = Val of env | BOT (*  $\mathcal{E}_{\#} \stackrel{\text{def}}{=} (\mathbb{V} \rightarrow \mathcal{D}_{\#}) \cup \{\perp\}$  *)

  (* arbre d'expression annoté *)
  type atree =
    | A_unary   of int_unary_op * atree * V.t
    | A_binary  of int_binary_op * atree * V.t * atree * V.t
    | A_var     of var * V.t
    | A_cst     of V.t

  (* évaluation d'expression : arbre annoté et valeur abstraite *)
  let rec eval (m:env) (e:int_expr) : atree * V.t = match e with ...

  ...

end: DOMAIN

```

Domaine non-relationnel générique : environnement

domains/non_relational.ml

```
let init ()    = Val VarMap.empty

let bottom () = BOT

let add_var a var = match a with
| BOT -> BOT
| Val m -> Val (VarMap.add var (V.const Z.zero) m)

let del_var a var = match a with
| BOT -> BOT
| Val m -> Val (VarMap.remove var m)

(* assignment *)
let assign a var e = match a with
| BOT -> BOT
| Val m ->
    let _,v = eval m e in
    if V.is_bottom v then BOT
    else Val (VarMap.add var v m)
```

Domaine non-relationnel générique : ordre

domains/non_relational.ml

```

let subset a b = match a,b with
| BOT,_ -> true
| _,BOT -> false
| Val m, Val n -> VarMap.for_all2z (fun _ x y -> V.subset x y) m n

let join a b = match a,b with
| BOT,x | x,BOT -> x
| Val m, Val n -> Val (VarMap.map2z (fun _ x y -> V.join x y) m n)

exception Empty

let meet a b = match a,b with
| BOT,x | x,BOT -> x
| Val m, Val n ->
    try Val
      (VarMap.map2z
        (fun _ x y ->
          let r = V.meet x y in
          if V.is_bottom r then raise Empty;
          r
        ) m n)
    with Empty -> BOT

```


Domaine des intervalles : manipulation des bornes

Implanter d'abord l'arithmétique dans $\mathbb{Z} \cup \{\pm\infty\}$

qui sert pour manipuler les bornes d'intervalles.

domains/interval_domain.ml

```
(*  $\mathbb{Z} \cup \{\pm\infty\}$  *)
type bound =
  | Int of Z.t    (*  $\mathbb{Z}$  *)
  | PINF          (*  $+\infty$  *)
  | MINF          (*  $-\infty$  *)

(*  $-a$  *)
let bound_neg (a:bound) : bound = match a with
  | MINF -> PINF | PINF -> MINF | Int i -> Int (Z.neg i)

(*  $a + b$  *)
let bound_add (a:bound) (b:bound) : bound = match a,b with
  | MINF,PINF | PINF,MINF -> invalid_arg "bound_add"  (*  $(+\infty) + (-\infty)$  *)
  | MINF,_ | _,MINF -> MINF
  | PINF,_ | _,PINF -> PINF
  | Int i, Int j -> Int (Z.add i j)

(* compare a et b, retourne -1, 0 ou 1 *)
let bound_cmp (a:bound) (b:bound) : int = match a,b with
  | MINF,MINF | PINF,PINF -> 0
  | MINF,_ | _,PINF -> -1
  | PINF,_ | _,MINF -> 1
  | Int i, Int j -> Z.compare i j
...
```

Domaine des intervalles : intervalles

Puis implanter la signature `VALUE_DOMAIN`

en s'inspirant de `constant_domain.ml`.

domains/interval_domain.ml

```
(* { [a, b] | a ≤ b } ∪ {⊥} *)
type t = Itv of bound * bound | BOT

(* extension de f par f(⊥) = ⊥ *)
let lift1 f x = match x with
| Itv (a,b) -> f a b
| BOT -> BOT

(* idem pour f(⊥, y) = f(x, ⊥) = ⊥ *)
let lift2 f x y = match x,y with ...

(* -x dans les intervalles *)
let neg (x:t) : t =
  lift1 (fun a b -> Itv (bound_neg b, bound_neg a)) x

(* x ⊆ y dans les intervalles *)
let subset (x:t) (y:t) : bool = match x,y with
| BOT,_ -> true
| _,BOT -> false
| Itv (a,b), Itv (c,d) -> bound_cmp a c >= 0 && bound_cmp b d <= 0

...
```