Sémantique et applications à la vérification
Examen (durée : 2h) — 27 mai 2015

27 mai 2015

Nous allons étudier des programmes manipulant un tableau de structures, dont les champs ont des types variés (caractères, scalaires entiers et flottants), et vérifier la correction des accès en lecture ou écriture dans ce tableau. *Dans ce sujet, nous ne nous intéressons qu'à la correction des accès, et nous ignorons complètement les valeurs contenues dans le tableau.*

Les accès au tableau se feront toujours en utilisant un *offset entier* à partir du début du tableau, qui décrit une position par une référence à un nombre d'octets à partir du début de la zone mémoire qui le contient.

Dans ce sujet, nous considérons trois types de base :

type	taille, en octets	désignation
char	1	caractères (type ASCII)
int	4	entier "32-bits"
double	8	flottant double précision

Une structure est décrite par une liste de champs qui ont chacun un type et un offset entier. Les champs étant collés les uns aux autres, l'offset du n-ième correspond à la somme des tailles des $n-1$ premiers champs.

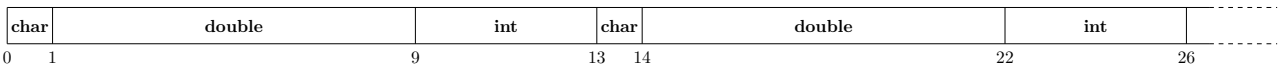
À titre d'exemple, nous considérons la description de structure ci-dessous :

char;	premier champ
double;	second champ
int;	troisième champ

Alors, la taille de la structure est de 13 octets, et les offsets des champs sont respectivement 0, 1 et 9. Par conséquent, lorsque l'on considère un tel tableau de structures,

- les offsets de la forme $13 \cdot k$ correspondent à un champ de type **char**;
- les offsets de la forme $13 \cdot k + 1$ correspondent à un champ de type **double**;
- les offsets de la forme $13 \cdot k + 9$ correspondent à un champ de type **int**.

Cela correspond à la structure ci-dessous :



Ce type d'accès n'est pas possible dans un langage de programmation de haut niveau comme OCaml ou Java (tous les accès à des champs de structures se font en utilisant le nom des champs), mais il est possible si l'on considère un langage tel que C (qui permet de faire un calcul en arithmétique de pointeurs et de calculer un offset numérique pour accéder à un tableau) et utilisé systématiquement si l'on considère des programmes en assembleur. Dans la suite nous allons considérer un langage impératif simple, mais qui ne permet que l'arithmétique de pointeurs (les champs de structures n'ont pas de noms, et les accès à un champ d'une structure stockée dans le tableau se font en donnant l'offset entier en octets).

τ	::=	**char** | **int** | **float**	type de base
v	$\in$	$\mathbb{V}$	valeur de type **char**, **int**, ou **float**
		`t`	(unique) variable décrivant le tableau
`x`	::=	`c`	variable of type **char** $(\mathtt{c} \in \mathbb{X}_{\mathtt{c}})$
	|	$\mathtt{i}_k$	variable of type **int** $(k \in \{0, \dots, n\})$
	|	`d`	variable of type **double** $(\mathtt{c} \in \mathbb{X}_{\mathtt{d}})$
S	::=	τ	structure réduite à un champ
	|	$\tau; S$	structure ayant au moins deux champs
C	::=	$\mathtt{x} = v;$	écriture d'une constante dans une variable
	|	$\mathtt{x} = \mathtt{x} \odot \mathtt{x};$	application d'un opérateur binaire $\odot \in \{+, -, \times\}$
	|	$\mathtt{x} = \mathtt{t}[\mathtt{i}_k];$	lecture d'un champ de structure, à l'offset défini par $\mathtt{i}_k$
	|	$\mathtt{t}[\mathtt{i}_k] = \mathtt{x};$	écriture d'un champ de structure, à l'offset défini par $\mathtt{i}_k$
	|	$C; C$	séquence
	|	**if**$(\star)\{C\}$**else**$\{C\}$	condition (avec condition aléatoire)
	|	**while**$(\star)\{C\}$	boucle (avec nombre d'itérations aléatoire)
P	::=	S, C	programme

Figure 1: Définition du langage étudié

Pour rendre l'énoncé plus compact, nous allons parfois faire des hypothèses simplificatrices restreignant artificiellement la définition du langage.

En particulier, nous faisons l'hypothèse que les programmes considérés manipulent un tableau unique `t`. Nous supposons également que les structures contenues dans les cellules successives d'un tableau sont collées les unes aux autres (pour ceux qui connaissent les notions d'alignement et de "padding" : nous n'insérons aucun bit de padding dans la représentation du tableau de structures, et nous supposons qu'il n'y a aucune contrainte d'alignement).

De plus, dans les parties **??** à **??**, nous ignorons la taille du tableau, considérant effectivement qu'il est infini. Cette hypothèse n'est bien sûr par réaliste, mais elle est cohérente avec le fait que, dans ce sujet, nous nous intéressons tout d'abord au bon alignement des accès aux cellules du tableau.

1 Sémantique des accès

La définition du langage que nous allons étudier est présentée dans la Figure **??**. On suppose deux ensembles $\mathbb{X}_{\mathtt{c}}$ et $\mathbb{X}_{\mathtt{d}}$ décrivent les variables de types **char** et **double**. Les variables de type **int** sont appelées $\mathtt{i}_0, \dots, \mathtt{i}_n$. Les affectations sont de quatre sortes : les initialisations de variables à l'aide d'une constante (qu'on supposera toujours du type de la variable initialisée), les opérations arithmétiques appliquées à une paire de variables (les opérations incluent l'addition, la soustraction et la multiplication —dans les trois cas, sur des variables de même type, et vers une variable du même type), les lectures dans le tableau, et les écritures dans le tableau. Lors d'une lecture ou écriture dans le tableau, l'indice est nécessairement une variable entière. Par ailleurs, les autres commandes incluent les classiques séquences, tests et boucles (lorsqu'une branche d'un test est vide, nous écrirons aussi par exemple, pour simplifier **if**$(\star)\{C\}$). Toutefois, et afin de rendre le langage plus simple, nous supposerons les tests aléatoires, que ce soit dans les structures conditionnelles ou dans les boucles (cela revient à considérer le nombre d'itérations des boucles aléatoires). Une structure S est définie par une liste de types, qui décrivent ses champs. Un programme P est défini par une liste de déclarations de variables et par une commande C.

Nous supposerons que toutes les affectations de la forme $\mathtt{x} = v;$ ou $\mathtt{x}_0 = \mathtt{x}_1 \odot \mathtt{x}_2$ sont bien typées.

Par conséquent, **l'objet de l'analyse statique que nous allons concevoir est de s'assurer que les accès au tableaux respectent les alignements et les types, autrement dit, on va essayer de vérifier les types et les alignements pour les instructions de la forme** $\mathtt{x} = \mathtt{t}[\mathtt{i}_k]$ **et** $\mathtt{t}[\mathtt{i}_k] = \mathtt{x}$.

Un *état mémoire* (ou *état*) est une fonction s, qui associe à toute variable et à tout offset valide du tableau (c'est-à-dire correspondant à un champ) une valeur de son type (cette fonction est donc totale sur les variables, mais partielle sur les offsets). Nous notons $\mathbb{M}$ l'ensemble de ces états mémoires. De plus, nous notons Ω un état d'erreur, et $\mathbb{M}_\Omega = \mathbb{M} \uplus \{\Omega\}$.

Dans cette partie, nous allons formaliser la sémantique concrête de chaque construction de ce langage. On note que la sémantique d'une commande est paramétrée par la structure employée, car les accès au tableau dépendent de celle-ci. Nous choisissons dans la suite de décrire la sémantique d'une commande à l'aide d'une fonction sur ensembles d'états :

$$[\![C]\!]_S : \mathcal{P}(\mathbb{M}) \longrightarrow \mathcal{P}(\mathbb{M}_\Omega)$$

À titre d'exemple, nous donnons la sémantique de quelques unes des commandes :

$$\begin{array}{rcll}
[\![\mathtt{x} = v;]\!]_S(\mathcal{E}) & = & \{m[\mathtt{x} \mapsto v] \mid m \in \mathcal{E}\} & \\
[\![\mathtt{x}_0 = \mathtt{x}_1 \odot \mathtt{x}_1]\!]_S(\mathcal{E}) & = & \{m[\mathtt{x}_0 \mapsto f_\odot(m(\mathtt{x}_1), m(\mathtt{x}_2))] \mid m \in \mathcal{E}\} & \text{où } f_\odot \text{ décrit l'opération } \odot \\
[\![C_0; C_1]\!]_S(\mathcal{E}) & = & [\![C_1]\!]_S([\![C_0]\!]_S(\mathcal{E})) & \\
[\![\textbf{if}(\star)\{C\}\textbf{else}\{C\}]\!]_S(\mathcal{E}) & = & [\![C_0]\!]_S(\mathcal{E}) \cup [\![C_1]\!]_S(\mathcal{E}) &
\end{array}$$

Par ailleurs, on supposera que les variables et champs du tableau peuvent initialement contenir n'importe quelle valeur de leur type.

Question 1 Représentation d'une structure.

Dans la suite, nous allons devoir raisonner sur la taille d'une structure, et sur sa représentation. Définir $\textbf{taille}(S)$ *qui calcule la taille en octets d'une structure, ainsi que la fonction partielle* $\textbf{repr}(S)$ *qui associe à chaque offset dans* $[0, \textbf{taille}(S)[$ *une valeur du type de ce champ (on écrira des fonctions récursives). Décrire ces fonctions pour la structure présentée Figure* **??**.

Corrigé 1 *Définition de* **taille** *:*

$$\begin{array}{rcl}
\textbf{taille}(\textbf{\textit{char}}) & = & 1 \\
\textbf{taille}(\textbf{\textit{int}}) & = & 4 \\
\textbf{taille}(\textbf{\textit{float}}) & = & 8 \\
\textbf{taille}(\tau; S) & = & \textbf{taille}(\tau) + \textbf{taille}(S)
\end{array}$$

Pour **repr**, *nous utilisons une fonction récursive auxilliaire* f *:*

$$\begin{array}{rcl}
f(k, \tau) & = & \{k \mapsto \tau\} \\
f(k, \tau; S) & = & \{k \mapsto \tau\} \uplus f(k + \textbf{taille}(\tau), S)
\end{array}$$

et nous pouvons alors définir $\textbf{repr}(S) = f(0, S)$.

On obtient $\textbf{taille}(S) = 21$ *et :*

$$\begin{array}{rcll}
\textbf{repr}(S) : & 0 & \longmapsto & \textbf{\textit{int}} \\
& 4 & \longmapsto & \textbf{\textit{int}} \\
& 12 & \longmapsto & \textbf{\textit{double}} \\
& 16 & \longmapsto & \textbf{\textit{int}} \\
& 20 & \longmapsto & \textbf{\textit{char}}
\end{array}$$

Question 2 Sémantique des opérations de lecture et d'écriture.

Nous considérons l'instruction $\mathtt{x} = \mathtt{t}[\mathtt{i}_k]$. *Celle-ci plante si la valeur de* $\mathtt{i}_k$ *ne correspond pas à un offset d'un champ dans le tableau de structures ou si le champ correspondant n'est pas du type de* x *; sinon, elle stocke le contenu de* $\mathtt{t}[\mathtt{i}_k]$ *dans la variable* x. *Formaliser la sémantique de cette instruction (on supposera que* x *a le type* $\tau \in \{\textbf{\textit{char}}, \textbf{\textit{int}}, \textbf{\textit{float}}\}$*).*
Faire de même avec l'instruction $\mathtt{t}[\mathtt{i}_k] = \mathtt{x}$, *qui plante si* $\mathtt{i}_k$ *ne correspond pas à l'offset d'un champ de type correspondant à celui de* x *et qui recopie le contenu de* x *dans la cellule correspondante du tableau sinon.*

Corrigé 2 *On note* $n \bmod p$ *le reste dans la division euclidienne de* n *par* p.

$$\begin{array}{rcl} [\![\mathtt{x} = \mathtt{t}[\mathtt{i}_k]]\!]_S(\{m\}) & = & \begin{cases} m[\mathtt{x} \mapsto m(m(\mathtt{i}_k))] & \textit{si } \mathbf{repr}(m(\mathtt{i}_k) \bmod \mathbf{taille}(S)) = \tau \\ \Omega & \textit{sinon} \end{cases} \\ [\![\mathtt{x} = \mathtt{t}[\mathtt{i}_k]]\!]_S(\mathcal{E}) & = & \bigcup \{[\![\mathtt{x} = \mathtt{t}[\mathtt{i}_k]]\!]_S(\{m\}) \mid m \in \mathcal{E}\} \end{array}$$

De même :

$$\begin{array}{rcl} [\![\mathtt{t}[\mathtt{i}_k] = \mathtt{x}]\!]_S(\{m\}) & = & \begin{cases} m[m(\mathtt{i}_k) \mapsto m(\mathtt{x})] & \textit{si } \mathbf{repr}(m(\mathtt{i}_k) \bmod \mathbf{taille}(S)) = \tau \\ \Omega & \textit{sinon} \end{cases} \\ [\![\mathtt{t}[\mathtt{i}_k] = \mathtt{x}]\!]_S(\mathcal{E}) & = & \bigcup \{[\![\mathtt{t}[\mathtt{i}_k] = \mathtt{x}]\!]_S(\{m\}) \mid m \in \mathcal{E}\} \end{array}$$

Question 3 Sémantique d'une boucle.

Donner la sémantique d'une boucle, à l'aide d'un point fixe, dont on justifiera l'existence.

Corrigé 3 *Pour définir la sémantique d'une boucle, on souhaite poser :*

$$[\![\textbf{\textit{while}}(\star)\{C\}]\!]_S(\mathcal{E}_0) = \textbf{\textit{lfp}}_{\mathcal{E}_0}(\lambda\mathcal{E} \cdot \mathcal{E} \cup [\![C]\!]_S(\mathcal{E}))$$

Toutefois, pour cela, on doit prouver que $[\![C]\!]_S$ *est croissante (si l'on souhaite appliquer le théorême de Tarski) ou continue (si l'on souhaite appliquer le théorême de Kleene). Cette preuve se fait par induction sur la syntaxe des commandes, en incluant le cas des boucles. On peut donc commencer par vérifier la croissance de* $[\![C]\!]_S$ *pour toute commande* C *ne contenant pas de boucles, puis procéder par induction sur le nombre de boucles imbriquées.*

2 Une abstraction à base de congruences

Pour vérifier la correction d'un accès (lecture ou écriture) au tableau, nous allons devoir raisonner en termes de congruences. Afin de rendre l'analyse automatique, nous nous proposons de concevoir un *domaine abstrait de congruences*.

Le domaine concret décrit les ensembles d'entiers, ordonnés par l'inclusion ($(\mathcal{P}(\mathbb{Z}), \subseteq)$). Les valeurs abstraites sont :

- $\bot_c$ qui décrit l'ensemble vide ;
- les paires $(n, p) \in \mathbb{Z} \times \mathbb{N}$ telles que, soit $p = 0$, soit $p \neq 0$ et $0 \leq n < p$; une telle paire (n, p) décrit tout ensemble inclus dans $\{n + kp \mid k \in \mathbb{Z}\}$.

Par conséquent, la fonction de concrétisation est définie comme suit :

$$\begin{array}{rcl} \gamma_c(\bot_c) & = & \emptyset \\ \gamma_c(n, p) & = & \{n + kp \mid k \in \mathbb{Z}\} \end{array}$$

Nous noterons $D_c^\sharp$ l'ensemble de ces valeurs abstraites.

Question 4 Ordre abstrait.

Définir la relation d'ordre sur les éléments abstraits $\sqsubseteq_c$, qui décrit exactement l'inclusion des concrétisations, c'est-à-dire, tel que $v_0^\sharp \sqsubseteq_c v_1^\sharp \iff \gamma_c(v_0^\sharp) \subseteq \gamma_c(v_1^\sharp)$

Corrigé 4 *Soient $(n,p),(n',p') \in D_c^\sharp$. Alors :*

$$\begin{array}{rcl} \gamma(n,p) \subseteq \gamma(n',p') & \iff & \{n+kp \mid k \in \mathbb{Z}\} \subseteq \{n'+kp' \mid k \in \mathbb{Z}\} \\ & \implies & \left\{ \begin{array}{l} \exists k_0 \in \mathbb{Z},\ n = n' + k_0 p' \\ \exists k_1 \in \mathbb{Z},\ n+p = n' + k_1 p' \end{array} \right. \\ & \implies & (p = p' \wedge n = n') \bigvee \left\{ \begin{array}{l} p' | (n-n') \\ p' | p \end{array} \right. \end{array}$$

Réciproquement, si $p'|(n-n')$ et $p'|p$, alors il existe a,b tels que $n = n' + ap'$ et $p = bp'$, donc $n + kp = n' + ap' + kbp' = n' + (a+kb)p'$.

Par conséquent, l'ordre abstrait est défini par :

$$\begin{array}{c} \bot_c \sqsubseteq_c \bot_c \\ \forall (n,p) \in D_c^\sharp,\ \bot_c \sqsubseteq_c (n,p) \\ \forall (n,p),(n',p') \in D_c^\sharp,\ (n,p) \sqsubseteq_c (n',p') \iff p'|p \wedge p'|(n-n') \end{array}$$

Question 5 Abstraction et correspondance de Galois.

Définir la fonction d'abstraction α_c correspondante. Montrer que $\alpha_c \circ \gamma_c$ est l'identité.

Corrigé 5 *On commence par éliminer les cas les plus simples :*

- *la meilleure abstraction de l'ensemble vide est $\bot_c$;*
- *la meilleure abstraction d'un singleton $\{n\}$ est la paire $(n,0)$.*

Soit un ensemble d'entiers V qui contient au moins deux éléments. On définit $V_\delta = \{v - v' \mid v, v' \in V\}$. Cet ensemble contient au moins une valeur non nulle.

Supposons $V \subseteq \gamma_c(n,p)$ et caractérisons (n,p). On observe que toute valeur de V_δ est nécessairement divisible par p. Donc p divise le PGCD *des éléments de V_δ.*

Réciproquement, si on pose $p = \mathrm{PGCD}(V_\delta)$, et prenons pour n le reste dans la division euclidienne d'un élément quelconque de V par p.

On peut alors montrer par analyse de cas que $V \subseteq \gamma_c(v^\sharp) \iff \alpha_c(V) \sqsubseteq_c v^\sharp$.

De même, la dernière propriété se montre aussi par analyse de cas.

Question 6 Top.

À quoi correspond l'élément $\top_c$, tel que $\gamma(\top_c) = \mathbb{Z}$?

Corrigé 6 *Le plus grand élément de $D_c^\sharp$ est $(0,1)$ et on a bien $\gamma_c(0,1) = \mathbb{Z}$.*

Question 7 Retour sur les éléments abstraits.

Nous nous sommes limités aux paires (n,p) telles que, lorsque $p \neq 0$, on a également $0 \leq n < p$. Que se passerait-il si nous n'avions pas imposé cette limitation ?

Corrigé 7 *Les éléments abstraits $(n,p),(n+p,p),(n-p,p),(n+2p,p),(n-2p,p),\ldots$ représenteraient exactement les mêmes ensembles de valeurs concrêtes. L'ordre abstrait que nous avons défini à la question* **??** *ne serait plus une relation d'ordre sur $D_c^\sharp$, mais seulement un pré-ordre (nous n'aurions plus la propriété d'antisymétrie).*

Dans la suite, et pour rendre certaines définitions plus concises, on pourra utiliser la fonction Φ définie par $\Phi(n,p) = (\text{reste}(n,p), p)$

Question 8 Bornes supérieures et inférieures.

Définir les bornes inférieures et supérieures pour toute paire d'éléments du treillis abstrait (on les notera $v_0^\sharp \sqcap_c v_1^\sharp$ et $v_0^\sharp \sqcup_c v_1^\sharp$).

Corrigé 8 *Pour tout $v^\sharp \in D_c^\sharp$, on a bien évidemment $\bot_c \sqcap_c v^\sharp = v^\sharp \sqcap_c \bot_c = \bot_c$ et $\bot_c \sqcup_c v^\sharp = v^\sharp \sqcup_c \bot_c = v^\sharp$.*

Considérons $(n,p), (n',p') \in D_c^\sharp$ et commençons par la borne supérieure :

$$\begin{array}{rcl}(n,p) \sqsubseteq_c (n'',p'') \wedge (n',p') \sqsubseteq_c (n'',p'') & \iff & p''|p \wedge p''|p' \wedge p''|n-n'' \wedge p''|n'-n'' \\ & \implies & p''|p \wedge p''|p' \wedge p''|n-n'\end{array}$$

Cela suggère donc de prendre $p'' = \text{PGCD}(p,p',n-n')$. On peut alors définir n'' comme le reste dans la division euclidienne de n. On obtient donc :

$$(n,p) \sqcup_c (n',p') = \Phi(n, \text{PGCD}(p,p',n-n'))$$

Le cas de la borne inférieure est plus complexe.

$$\begin{array}{rcl}(n'',p'') \sqsubseteq_c (n,p) \wedge (n'',p'') \sqsubseteq_c (n',p') & \iff & p|p'' \wedge p'|p'' \wedge p|n''-n \wedge p'|n''-n' \\ & \implies & \text{PPCM}(p,p')|p'' \wedge \text{PGCD}(p,p')|n-n'\end{array}$$

Cela implique que si $n-n'$ n'est pas divisible par $\text{PGCD}(p,p')$, alors, la borne inférieure est $\bot_c$. Sinon, n'' peut être défini comme le plus petit élément de $\{m \in \mathbb{Z} \mid m \geq 0 \wedge \exists k, m = n+kp \wedge \exists k', m = n+kp'\}$. Cela donne la définition de la borne inférieure.

Question 9 Arithmétique : addition, soustraction et multiplication.

Définir des fonctions (que l'on rendra aussi précises que possibles) $f_+^\sharp : D_c^\sharp \times D_c^\sharp \to D_c^\sharp$ $f_-^\sharp : D_c^\sharp \times D_c^\sharp \to D_c^\sharp$ et $f_\times^\sharp : D_c^\sharp \times D_c^\sharp \to D_c^\sharp$ telles que :

$$\begin{array}{ll}\forall v_0^\sharp, v_1^\sharp \subseteq D_c^\sharp, & \{v_0+v_1 \mid \forall i,\ v_i \in \gamma_c(v_i^\sharp)\} \subseteq \gamma_c(f_+^\sharp(v_0^\sharp, v_1^\sharp)) \\ \forall v_0^\sharp, v_1^\sharp \subseteq D_c^\sharp, & \{v_0-v_1 \mid \forall i,\ v_i \in \gamma_c(v_i^\sharp)\} \subseteq \gamma_c(f_-^\sharp(v_0^\sharp, v_1^\sharp)) \\ \forall v_0^\sharp, v_1^\sharp \subseteq D_c^\sharp, & \{v_0\times v_1 \mid \forall i,\ v_i \in \gamma_c(v_i^\sharp)\} \subseteq \gamma_c(f_\times^\sharp(v_0^\sharp, v_1^\sharp))\end{array}$$

En déduire que pour chacune de ces fonctions abstraites, on a aussi :

$$\forall V_0, V_1 \subseteq \mathbb{Z},\ \alpha_c(\{v_0 \odot v_1 \mid v_i \in V_i\}) \sqsubseteq_c f_\odot^\sharp(\alpha_c(V_0), \alpha_c(V_1))$$

Corrigé 9 *On constate que dans tous les cas, on peut définir les cas où apparaît $\bot_c$ en considérant que celui-ci est absorbant :*

$$\forall v^\sharp \in D_c^\sharp,\ f_+^\sharp(\bot_c, v^\sharp) = f_+^\sharp(v^\sharp, \bot_c) = f_-^\sharp(\bot_c, v^\sharp) = f_+^\sharp(v^\sharp, \bot_c) = f_\times^\sharp(\bot_c, v^\sharp) = f_+^\sharp(v^\sharp, \bot_c) = \bot_c$$

Considérons donc des éléments différents de $\bot_c$. Alors :

$$\begin{array}{rcl}\{v_0+v_1 \mid v_i \in \gamma_c(n_i,p_i)\} & = & \{(n_0+k_0p_0)+(n_1+k_1p_1) \mid k_0,k_1 \in \mathbb{Z}\} \\ & = & \{(n_0+n_1)+(k_0p_0+k_1p_1) \mid k_0,k_1 \in \mathbb{Z}\} \\ & = & \{(n_0+n_1)+k\text{PGCD}(p_0,p_1) \mid k \in \mathbb{Z}\} \\ & = & \gamma_c(\Phi(n_0+n_1, \text{PGCD}(p_0,p_1)))\end{array}$$

Ce qui nous permet de définir (le cas de la soustraction étant clairement similaire) :

$$\begin{aligned} f^\sharp_+((n_0,p_0),(n_1,p_1)) &= \Phi(n_0+n_1, \mathrm{PGCD}(p_0,p_1)) \\ f^\sharp_-((n_0,p_0),(n_1,p_1)) &= \Phi(n_0-n_1, \mathrm{PGCD}(p_0,p_1)) \end{aligned}$$

Considérons à présent le cas de la multiplication :

$$\begin{aligned} \{v_0 v_1 \mid v_i \in \gamma_c(n_i,p_i)\} &= \{(n_0+k_0p_0)(n_1+k_1p_1) \mid k_0,k_1 \in \mathbb{Z}\} \\ &= \{n_0n_1 + k_0n_1p_0 + k_1n_0p_1 + k_0k_1p_0p_1 \mid k_0,k_1 \in \mathbb{Z}\} \\ &\subseteq \{(n_0n_1) + k\mathrm{PGCD}(p_0p_1, n_0p_1, n_1p_0) \mid k \in \mathbb{Z}\} \\ &= \gamma_c(\Phi(n_0n_1, \mathrm{PGCD}(p_0p_1, n_0p_1, n_1p_0))) \end{aligned}$$

Cela donne :

$$f^\sharp_\times((n_0,p_0),(n_1,p_1)) = \Phi(n_0n_1, \mathrm{PGCD}(p_0p_1, n_0p_1, n_1p_0))$$

Finalement, pour obtenir la dernière égalité, on peut composer les égalités obtenues ci-dessus avec α_c, et on obtient alors :

$$\alpha_c(\{v_0 \odot v_1 \mid v_i \in \gamma_c(\alpha_c(V_i))\}) \sqsubseteq_c \alpha_c \circ \gamma_c(f^\sharp_\odot(\alpha_c(V_0), \alpha_c(V_1)))$$

Pour conclure, il suffit alors d'utiliser :

- *le fait que $\alpha_c \circ \gamma_c$ est l'identité ;*
- *le fait que $\gamma_c \circ \alpha_c$ est extensive, et que α_c est croissante*

3 Abstraction pour ensembles d'états

Nous considérons à présent l'abstraction d'ensembles d'états. Les éléments du domaine abstrait $D^\sharp$ sont définis par :

- $\bot$, qui décrit l'ensemble vide d'états ;
- les éléments de la forme $(v_0^\sharp, \ldots, v_n^\sharp) \in (D_c^\sharp)^n$ et où $v_i^\sharp \neq \bot_c$ pour tout i ; l'élément abstrait $(v_0^\sharp, \ldots, v_n^\sharp)$ décrit l'ensemble des états mémoires m, tels que $\forall k \in \{0,\ldots,n\},\ m(\mathtt{i}_k) \in \gamma_c(v_k^\sharp)$.

L'ordre abstrait est défini par $\bot \sqsubseteq (v_0^\sharp, \ldots, v_n^\sharp)$, et $(v_0^\sharp, \ldots, v_n^\sharp) \sqsubseteq (w_0^\sharp, \ldots, w_n^\sharp) \iff (\forall i,\ v_i^\sharp \sqsubseteq_c w_i^\sharp)$, pour tous $(v_0^\sharp, \ldots, v_n^\sharp), (w_0^\sharp, \ldots, w_n^\sharp) \in (D_c^\sharp)^n$.

On peut noter que cette abstraction ignore complètement les valeurs stockées dans le tableau.

Question 10 Relation d'abstraction.

Définir les fonctions d'abstraction et de concrétisation correspondantes à cette description informelle et prouver qu'elles établissent une correspondance de Galois.

Corrigé 10 *L'abstraction ignore les valeurs des variables non entières. On obtient ainsi :*

$$\begin{aligned} \alpha(\emptyset) &= \bot \\ \alpha(E^\sharp) &= (\alpha_c(\{m(\mathtt{i}_0) \mid m \in E^\sharp\}), \ldots, \alpha_c(\{m(\mathtt{i}_n) \mid m \in E^\sharp\})) \text{ si } E^\sharp \neq \emptyset \end{aligned}$$

La concrétisation s'exprime par :

$$\begin{aligned} \gamma(\bot) &= \emptyset \\ \gamma(v_0^\sharp, \ldots, v_n^\sharp) &= \{m \in \mathbb{M} \mid \forall k,\ \mathtt{i}_k \in \gamma_c(v_k^\sharp)\} \end{aligned}$$

La propriété des correspondances de Galois se vérifie par analyse de cas.

4 Analyse statique des opérations simples

Nous allons maintenant concevoir une sémantique abstraite $[\![C]\!]^\sharp_S : D^\sharp \longrightarrow D^\sharp$ qui fournisse une sur-approximation sûre de la sémantique $[\![C]\!]_S$, aussi précise que possible. Dans cette partie, il sera toujours possible de définir une sémantique abstraite vérifiant la propriété de *correction* $\alpha \circ [\![C]\!]_S(\mathcal{E}) \subseteq [\![C]\!]^\sharp_S \circ \alpha(\mathcal{E})$.

Question 11 Initialisation de variables.

Définir la sémantique abstraite de l'instruction d'initialisation d'une variable $\mathtt{x} = v$. *On supposera que le type* τ *de* v *correspond bien à celui de* $\mathtt{x}$, *et on considérera tous les cas possibles pour ce type. On montrera aussi la correction.*

Corrigé 11 *Par cas :*

- *si* $\tau =$ ***char*** *ou* $\tau =$ ***double***, *les valeurs des variables entières ne changent pas, donc*

$$[\![\mathtt{x} = v]\!]^\sharp(E^\sharp) = E^\sharp$$

- *si* $\tau =$ ***int*** *:*

$$\begin{array}{rcl} [\![\mathtt{i}_k = v]\!]^\sharp(\bot) & = & \bot \\ [\![\mathtt{i}_k = v]\!]^\sharp(v_0^\sharp, \ldots, v_n^\sharp) & = & (v_0^\sharp, \ldots, v_{k-1}^\sharp, \alpha_c(\{v\}), v_{k+1}^\sharp, \ldots, v_n^\sharp) \end{array}$$

La correction découle de la définition de α_c.

Question 12 Opérations binaires.

Faire de même pour l'instruction $\mathtt{x}_0 = \mathtt{x}_1 \odot \mathtt{x}_2$.

Corrigé 12 *Par cas :*

- *si* $\tau =$ ***char*** *ou* $\tau =$ ***double***, *les valeurs des variables entières ne changent pas, donc*

$$[\![\mathtt{x}_0 = \mathtt{x}_1 \odot \mathtt{x}_2]\!]^\sharp(E^\sharp) = E^\sharp$$

- *si* $\tau =$ ***int*** *:*

$$\begin{array}{rcl} [\![\mathtt{i}_k = \mathtt{x}_l \odot \mathtt{x}_{l'}]\!]^\sharp(\bot) & = & \bot \\ [\![\mathtt{i}_k = \mathtt{x}_l \odot \mathtt{x}_{l'}]\!]^\sharp(v_0^\sharp, \ldots, v_n^\sharp) & = & (v_0^\sharp, \ldots, v_{k-1}^\sharp, f_\odot^\sharp(v_l^\sharp, v_{l'}^\sharp), v_{k+1}^\sharp, \ldots, v_n^\sharp) \end{array}$$

La correction découle de la définition de $f_\odot^\sharp$ *(question* **??***).*

Question 13 Accès au tableau : analyse et vérification.

Définir la sémantique abstraite des deux instructions d'accès au tableau (en lecture et en écriture) et justifier leur correction. (on rappelle que notre abstraction ignore le contenu du tableau, et on ne s'intéresse ici qu'aux bornes).
De plus, on souhaite savoir si une pré-condition abstraite $m^\sharp$ *guarantit la correction d'un accès au tableau. Pour cela, concevoir une fonction* **verif** *qui renvoie un booléen, égal à* ***true*** *si l'analyse prouve l'accès correct, et à* ***false*** *sinon (préciser les arguments que doit prendre cette fonction).*

Corrigé 13 *Si on considère un accès* $\mathtt{t}[\mathtt{i}_k]$ *pour une lecture ou écriture d'une valeur de type* τ, *celui-ci peut s'exécuter sans planter si et seulement si* $\mathtt{i}_k$ *contient la valeur d'un offset de type* τ. *Par conséquent,* **verif** *prend en argument la structure* S, *le type* τ, *l'indice* k *de la variable entière qui sert d'indice et la pré-condition abstraite* $E^\sharp$. *De plus :*

structure S :	program P_1, de corps C_1 :	program P_2 de corps C_2 (partie **??**) :
int;	$\mathbb{X}_{\mathtt{c}} = \{\mathtt{x}\}; \mathbb{X}_{\mathtt{d}} = \emptyset$	$\mathbb{X}_{\mathtt{c}} = \{\mathtt{x}\}; \mathbb{X}_{\mathtt{d}} = \emptyset; T = 10$
double;		
int;	$\mathtt{i}_0 = 42;$	$\mathtt{i}_0 = 42;\ \mathtt{i}_1 = 20;$
int;	$\mathtt{i}_1 = 20;$	$\mathtt{i}_0 = \mathtt{i}_0 \times \mathtt{i}_2;$
char;	**if**$(\star)\{\mathtt{i}_1 = \mathtt{i}_0 + \mathtt{i}_1;\}$	$\mathtt{i}_1 = \mathtt{i}_0 + \mathtt{i}_1;$
	$\mathtt{x} = \mathtt{t}[\mathtt{i}_1];$	**if**$(0 \leq \mathtt{i}_1 \mathtt{\&\&} \mathtt{i}_1 < 210)\{\mathtt{x} = \mathtt{t}[\mathtt{i}_1];\}$

Figure 2: Exemples

- *si* $E^\sharp = \bot$, *alors* $\mathbf{verif}(S, \tau, k, E^\sharp) = \boldsymbol{true}$
- *si* $E^\sharp = (v_0^\sharp, \ldots, v_n^\sharp)$ *où* $v_k^\sharp = (n_k, p_k)$,

$$\mathbf{verif}(S, \tau, k, E^\sharp) = \begin{cases} \boldsymbol{true} & si\ \mathbf{taille}(S) | p_k\ et\ \mathbf{repr}(S)(\mathrm{reste}(n_k, \mathbf{taille}(S))) = \tau \\ \boldsymbol{true} & si\ p_k | \mathbf{taille}(S)\ et\ \mathbf{repr}(S)(n_k) = \mathbf{repr}(S)(n_k + p_k) = \ldots = \tau \\ \boldsymbol{false} & sinon \end{cases}$$

Le premier cas renvoyant ***true*** *correspond à un accès pouvant correspondre à au plus un champ par structure tandis que le second décrit les cas où une structure contient plusieurs champs de même type, qui peuvent être accédés indifféremment (mais avec la contrainte que l'on retrouve ce champ à tous les offsets modulo* p_k*).*

On considère maintenant la lecture $\mathtt{x} = \mathtt{t}[\mathtt{i}_k]$. *La vérification d'absence de plantage se fait à l'aide de* $\mathbf{verif}(S, \tau, k, E^\sharp)$ *et :*

- *si* $\tau =$ ***char*** *ou* $\tau =$ ***double***, *les valeurs des variables entières ne changent pas, donc*

$$[\![\mathtt{x} = \mathtt{t}[\mathtt{i}_k]]\!]^\sharp(E^\sharp) = E^\sharp$$

- *si* $\tau =$ ***int*** *:*

$$\begin{array}{rcl} [\![\mathtt{i}_j = \mathtt{t}[\mathtt{i}_k]]\!]^\sharp(\bot) & = & \bot \\ [\![\mathtt{i}_j = \mathtt{t}[\mathtt{i}_k]]\!]^\sharp(v_0^\sharp, \ldots, v_n^\sharp) & = & (v_0^\sharp, \ldots, v_{k-1}^\sharp, \top_c, v_{k+1}^\sharp, \ldots, v_n^\sharp) \end{array}$$

Le cas d'une écriture est plus simple car l'abstraction ne tient pas compte des valeurs stockées dans le tableau, donc, dans tous les cas, et après avoir effectué la vérification d'absence de plantage $\mathbf{verif}(S, \tau, k, E^\sharp)$,

$$[\![\mathtt{t}[\mathtt{i}_k] = \mathtt{x}]\!]^\sharp(E^\sharp) = E^\sharp$$

Question 14 Analyse statique des instructions composées.

Définir la sémantique abstraite des séquences, des tests, et des boucles, et justifier leur correction, ainsi que la terminaison de l'analyse. Décrire l'analyse du programme $S; C_1$ *de la Figure* **??**, *en partant de l'état abstrait* $(\top_c, \top_c)$ *(on suppose qu'on part d'un état indéfini). Donner et commenter un exemple d'analyse de boucle.*

Corrigé 14 *On obtient les définitions classiques :*

$$\begin{array}{rcl} [\![C_0; C_1]\!]^\sharp(E^\sharp) & = & [\![C_1]\!]^\sharp \circ [\![C_0]\!]^\sharp(E^\sharp) \\ [\![\boldsymbol{if}(\star)\{C\}\boldsymbol{else}\{C\}]\!]^\sharp(E^\sharp) & = & [\![C_0]\!]^\sharp(E^\sharp) \sqcup [\![C_0]\!]^\sharp(E^\sharp) \\ [\![\boldsymbol{while}(\star)\{C\}]\!]^\sharp(E^\sharp) & = & \boldsymbol{lfp}_{E^\sharp}(\lambda E^\sharp \cdot E^\sharp \sqcup [\![C]\!]^\sharp(E^\sharp)) \end{array}$$

(la définition du point fixe abstrait découle du fait que la sémantique abstraite que nous avons définie est croissante)

L'analyse termine car le treillis des congruences vérifie la condition de chaîne.
On obtient l'analyse suivante :

$$
\begin{array}{ll}
 & (\top_c, \top_c) \\
\mathtt{i}_0 = 42; & \\
 & ((42,0), \top_c) \\
\mathtt{i}_1 = 20; & \\
 & ((42,0),(20,0)) \\
\textbf{\textit{if}}(\star)\{ & \quad ((42,0),(20,0)) \\
\quad \mathtt{i}_1 = \mathtt{i}_0 + \mathtt{i}_1; & \quad ((42,0),(62,0)) \\
\}\textbf{\textit{else}}\{ & \\
 & ((42,0),(20,0)) \\
\} & \\
 & ((42,0),(20,42)) \\
\mathtt{x} = \mathtt{t}[\mathtt{i}_1]; & \\
 & ((42,0),(20,42))
\end{array}
$$

L'accès au tableau est correct car la taille de la structure est 21 (qui divise 42) et que 20 est bien l'offset d'un champ de type ***char****.*

5 Ajout de contraintes d'invervalles

Jusqu'à présent, nous avons complètement ignoré le fait que, dans tout langage réaliste, les tableaux ont une longueur finie, et que tout accès en dehors des bornes de tableaux déclenche une erreur ou un comportement non défini. Nous nous proposons maintenant de traiter à la fois des dépassements de tableaux et les problèmes d'alignements considérés jusqu'ici.

Question 15 Extension du langage et de la sémantique concrête.

On suppose dans la suite que le tableau contient T éléments correspondant à des instances de la structure S. Redéfinir la sémantique concrête des accès au tableau, en tenant compte du fait que tout accès à un offset négatif ou supérieur à la taille du tableau en octets plante.
Par ailleurs, on souhaite à présente tenir compte des tests (dans les instructions conditionnelles et les boucles). On prendra en compte les tests de la forme $\mathtt{i}_k < v$. Modifier la syntaxe du langage, ainsi que la sémantique concrête.

Corrigé 15 *Le cas d'une écriture étant similaire, on ne traite que celui d'une lecture :*

$$
\begin{array}{rcl}
[\![\mathtt{x} = \mathtt{t}[\mathtt{i}_k]]\!]_S(\{m\}) & = & \left\{ \begin{array}{ll} m[\mathtt{x} \mapsto m(m(\mathtt{i}_k))] & \textit{si } \mathbf{repr}(m(\mathtt{i}_k) \bmod \mathbf{taille}(S)) = \tau \textit{ et } 0 \leq m(\mathtt{i}_k) < T\mathbf{taille}(S) \\ \Omega & \textit{sinon} \end{array} \right. \\
[\![\mathtt{x} = \mathtt{t}[\mathtt{i}_k]]\!]_S(\mathcal{E}) & = & \bigcup \{ [\![\mathtt{x} = \mathtt{t}[\mathtt{i}_k]]\!]_S(\{m\}) \mid m \in \mathcal{E} \}
\end{array}
$$

L'extension des tests est classique (voir cours).

Dans la suite, nous remplaçons $D_c^\sharp$ par un nouveau domaine abstrait $D_{ic}^\sharp$ qui décrit un ensemble d'entiers à l'aide de la conjonction d'une contrainte d'intervalle et d'une contrainte de congruence.

Question 16 Abstraction des ensembles de valeurs entières.

Définir les valeurs abstraites, l'ordre abstrait, ainsi que les fonctions d'abstraction et de concrétisation de manière à exprimer ces propriétés abstraites contenant à la fois une contrainte d'intervalle et une contrainte de congruence.

Corrigé 16 *On note $D_i^\sharp$ le domaine des intervalles (et de même pour ses fonctions d'abstraction, concrétisation, etc). Alors :*

- $D_{ic}^\sharp = D_c^\sharp \times D_i^\sharp$
- *l'ordre abstrait est l'ordre produit*
- $\gamma_{ic}(v^\sharp, I^\sharp) = \gamma_c(v^\sharp) \cap \gamma_i(I^\sharp)$
- $\alpha(\mathcal{E}) = (\alpha_c(\mathcal{E}), \gamma_c(\mathcal{E}))$

Question 17 Analyse statique des tests.

On considère la fonction concrête $f : \mathcal{E} \mapsto \{m \in \mathcal{E} \mid m(\mathtt{i}_k) < v\}$. Définir une fonction abstraite (que l'on essaiera de rendre aussi précise que possible) qui sur-approxime cette fonction.
Montrer l'effet de l'analyse du test dans l'exemple du programme $S; C_2$ de la Figure **??**.

Corrigé 17 *L'analyse raffine les bornes de l'intervalle, mais les contraintes de congruences peuvent alors raffiner encore plus les bornes de l'intervalle. On appelle cela une* réduction.

On obtient :

$$
\begin{array}{l}
\qquad ((\top_c, \top_i), (\top_c, \top_i)) \\
\mathtt{i}_0 = 42; \\
\qquad (((42, 0), [42, 42]), (\top_c, \top_i)) \\
\mathtt{i}_1 = 20; \\
\qquad (((42, 0), [42, 42]), ((20, 0), [20, 20])) \\
\mathtt{i}_0 = \mathtt{i}_0 \times \mathtt{i}_2; \\
\qquad (((0, 42), \top_i), ((20, 0), [20, 20])) \\
\mathtt{i}_1 = \mathtt{i}_0 + \mathtt{i}_1; \\
\qquad (((0, 42), \top_i), ((20, 42), \top_i)) \\
\textbf{\textit{if}}(0 \leq \mathtt{i}_1 \&\& \mathtt{i}_1 < 210)\{ \\
\qquad (((0, 42), \top_i), ((20, 42), [20, 188])) \\
\quad \mathtt{x} = \mathtt{t}[\mathtt{i}_1]; \\
\}
\end{array}
$$

L'effet du test est le suivant :

- *les bornes de $\mathtt{i}_1$ sont d'abord réduites à $[0, 210[$;*
- *ensuite, la contrainte de congruence sur la valeur de cette variable permet d'exclure les valeurs comprises dans $[0, 19]$ et $[189, 210[$, et on obtient donc $[20, 188]$*

Cela permet ensuite de vérifier à la fois la contrainte de congruence et la contrainte d'intervalle, permettant de valider l'accès au tableau.

Question 18 Analyse statique des accès au tableau.

Re-définir la fonction **verif** *et la sémantique abstraite pour les opérations d'accès au tableau.*

Corrigé 18 *La principale différence est la nouvelle fonction* **verif**, *qui doit aussi s'assurer que l'intervalle de valeurs possibles est bien inclus dans l'intervalle $[0, T\textbf{taille}(S)[$. Par ailleurs, dans le cas où l'analyse découvre qu'un accès est* à coup sûr *en dehors des bornes du tableau, celle-ci renverra $\bot$.*