Sémantique et applications à la vérification
Devoir Maison : Analyse de Teintes

Mai 2020

Dans ce problème, nous allons étudier une version simplifiée d'analyse de teinte, qui permet de détecter les éléments d'un programme susceptibles d'influencer chacune des valeurs qu'il produit.

Les différentes parties du problème correspondent à des questions de difficulté croissante, qu'il est conseillé de traiter dans l'ordre.

On attachera une grande importance à la clarté des réponses. Il est possible de rendre le devoir soit sous la forme d'un `.pdf` produit avec `LaTeX` ou du scan de copies rédigées sur papier (dans ce second cas, il est préférable si possible d'assembler l'ensemble des pages dans l'ordre et au sein d'un seul fichier, ou bien à défaut, dans une archive `.tgz` en prenant soin de nommer les fichiers suivant les numéros de page).

Expressions arithmétiques simples

Dans un premier temps, nous allons considérer un langage d'expressions très simple que nous allons réutiliser plus tard. Nous supposons fixés un ensemble fini de variables $\mathbb{X}$ et un ensemble de valeurs numériques (par exemple $\mathbb{Z}$). On suppose que $\mathbb{V}$ désigne cet ensemble de valeurs auquel on a ajouté la constante $\bullet$ décrivant une valeur non définie. Une mémoire est une fonction des variables vers les valeurs, que l'on note $\mathbb{M}$. Si $m \in \mathbb{M}$, alors $m(x) = \bullet$ signifie que m n'est pas définie pour x. On a donc $\mathbb{M} = \mathbb{X} \to \mathbb{V}$. Si $X \subseteq \mathbb{X}$, on note $\mathbb{M}_X$ l'ensemble des mémoires définies exactement sur X (c'est à dire telles que $m(x) = \bullet \iff x \notin X$). Étant donnée une mémoire m, on note $|m|$ l'ensemble des variables pour lesquelles elle est définie. On observe donc que, pour toute mémoire m, $m \in \mathbb{M}_{|m|}$.

Les expressions sont définies par la grammaire suivante :

e	$::=$	v	constante $v \in \mathbb{V}$
	$\mid$	x	variable $x \in \mathbb{X}$
	$\mid$	$e_0 \oplus e_1$	opération binaire $\oplus \in \{+, -, *\}$ appliquée à deux expressions e_0, e_1

L'évaluation d'une expression dans une mémoire donnée se fait de manière récursive. Par exemple, si $m(x) = 8, m(y) = 7, m(z) = 6$ alors $2 * (x + y)$ s'évalue en 30. On note que cette expression n'utilise que les variables $\{x, y\}$ et que z ne joue aucun rôle. Changer la valeur de cette dernière variable ne changerait pas le résultat.

Question 1 —Variables présentes dans une expression.

On note $\mathcal{V}(e)$ l'ensemble des variables qui apparaissent dans l'expression e. Définir formellement $\mathcal{V}$.

Question 2 —Sémantique.

On note $[\![e]\!]_{\mathtt{e}}$ la sémantique d'une expression e. Il s'agit d'une fonction $[\![e]\!]_{\mathtt{e}} : \mathbb{M} \longrightarrow \mathbb{V}$.
Formaliser la définition de cette sémantique. Quand cette fonction renvoie-t'elle une valeur bien définie (c'est à dire différente de $\bullet$) ?

Soient deux mémoires $m_0, m_1 \in \mathbb{M}$ et $X \subseteq \mathbb{X}$ un ensemble de variables. On dit que m_0, m_1 sont *équivalentes* sur X et on note $m_0 \equiv_X m_1$ si et seulement si :

$$X \subseteq |m_0| \wedge X \subseteq |m_1| \wedge \forall x \in X,\ m_0(x) = m_1(x)$$

Question 3 —Indépendance.

Montrer que si $m_0 \equiv_X m_1$ et $\mathcal{V}(e) \subseteq X$, alors $[\![e]\!]_{\mathtt{e}}(m_0) = [\![e]\!]_{\mathtt{e}}(m_1)$.

Commandes

Nous allons maintenant construire un langage un peu plus complet, mais qui reste très simple pour faciliter la définition d'une analyse.

Un programme est défini par un ensemble de variables d'entrées et une commande. Une commande consiste soit en une expression simple, soit en une définition de variable, soit en une condition. Ces constructions sont décrites par la grammaire ci-dessous :

p	::=	$\mathbf{input}(\{x_0, \ldots, x_n\});\ c$	programme
c	::=	e	expression simple
	|	$\mathbf{var}\ x = e_0\ \mathbf{in}\ c_1$	(re)-définition d'une variable
	|	$\mathbf{if}\ x \geq 0\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1$	condition

L'éxécution d'une commande ou d'un programme retourne la valeur de la dernière expression évaluée. Par ailleurs, on considère que la construction $\mathbf{var}\ x = e_0\ \mathbf{in}\ c_1$ a le comportement suivant : si la variable x n'est pas définie dans la mémoire courante, alors elle est ajoutée (avec la valeur correspondante au résultat de l'évaluation du membre droit) ; sinon, alors la valeur de la variable x est mise à jour (c'est la raison pour laquelle on nomme ici ces constructions *commandes* et non *expressions* : la valeur d'une variable peut évoluer au cours de l'éxécution). Enfin, un programme $\mathbf{input}(\{x_0, \ldots, x_n\});\ c$ s'éxécute par convention à partir d'une mémoire définissant exactement $\{x_0, \ldots, x_n\}$ dont les valeurs initiales sont des paramètres du programme. Si une valeur non définie est lue à n'importe quel moment de l'éxécution alors, le programme retourne $\bullet$

Question 4 —Comportement des (re)-définitions de variables.

Comparer le comportement de $\mathbf{var}\ x = e_0\ \mathbf{in}\ c_1$ a celui d'un `let x = e0 in e1` *en OCaml.*

Question 5 —Sémantique.

Formaliser la sémantiques des commandes et des programmes, qui prendront la forme de deux fonctions :

$$[\![c]\!]_{\mathtt{c}} : \mathbb{M} \to \mathbb{V} \qquad [\![\mathbf{input}(X);\ c]\!]_{\mathtt{p}} : \mathbb{M}_X \to \mathbb{V}$$

On pourra donner un exemple de programme pour lequel on obtient $\bullet$.

Question 6 —Analyse de bonne définition.

En s'inspirant de ce qui a été vu dans la première partie, proposer une analyse statique qui garantisse qu'un programme s'évalue sans renvoyer • pour toute mémoire définie exactement sur les variables apparaissant dans l'ensemble argument de **input**$(\ldots)$*). Cette analyse ne devra pas évaluer les expressions. Définir et prouver l'analyse. Est-elle exacte ou bien induit-elle une approximation ?*

Sémantique non standard embarquant un calcul de teintes

Nous allons progressivement nous rapprocher d'une analyse statique qui permet de déterminer quelles sont les entrées d'un programme qui ont une influence sur son résultat au sens de la dernière question de la première partie. Pour cela nous proposons de définir une sémantique non-standard, dans laquelle chaque valeur est marquée par l'ensemble des variables prises en argument par le programme, et qui ont participé directement ou indirectement (par exemple dans les conditions ou bien dans dans d'autres calculs intermédiaires) au calcul de cette valeur. On note (v, X) la valeur v *annotée* par l'ensemble X (où v est un scalaire). On considère également • comme valeur annotée. On note $\overline{\mathbb{V}} = \mathbb{V} \times \mathcal{P}(\mathbb{X}) \uplus \{\bullet\}$ l'ensemble des *valeurs annotées* et $\overline{v}$ une valeur annotée. De même, $\overline{\mathbb{M}} = \mathbb{X} \to \overline{\mathbb{V}}$ désigne l'ensemble des *mémoires annotées* et on note $\overline{m}$ une telle mémoire. On utilise aussi les notations $\overline{\mathbb{M}}_X$ et $|\overline{m}|$ avec le même sens que précédemment.

Avant de formaliser cette sémantique, considérons le programme ci-dessous et une mémoire $\overline{m}$ qui associe respectivement à x_0, x_1 et x_2 les valeurs annotées $(0, \{x_0\}), (2, \{x_1\})$ et $(8, \{x_2\})$ (la valeur de chaque variable est annotée par un singleton ne contenant que la variable elle-même) :

input$(\{x_0, x_1, x_2\})$;
var $x_1 = x_0 + 1$ **in**
if $x_1 \geq 0$ **then**
    var $x_2 = 2 * x_2 + 2$ **in**
    var $x_3 = 7$ **in**
    $x_2 - x_3$
else
    2

Alors, ce programme produit la valeur $(11, \{x_0, x_2\})$; la dernière expression $x_2 - x_3$ s'évalue dans la mémoire annotée $\overline{m}'$ telle que :

$$\overline{m}'(x_0) = (0, \{x_0\}) \qquad \overline{m}'(x_1) = (1, \{x_0\}) \qquad \overline{m}'(x_2) = (18, \{x_0, x_2\}) \qquad \overline{m}'(x_3) = (7, \{x_0\})$$

Cet exemple illustre les principaux éléments de la sémantique non-standard que nous allons utiliser :

- la valeur initiale de x_1 n'est pas utilisée, et la valeur de x_1 est redéfinie à partir de celle de x_0 ;
- la condition sur x_1 est déterminée par la valeur initiale de x_0 et influence donc les valeurs de tous les éléments définis dans le corps de la condition et donc aussi la valeur finale.

Voyons un autre exemple :

input$(\{x_0\})$;
if $x_0 \geq 0$ **then**
    5
else
    2

Alors, l'éxécution partant de la mémoire qui associe à x_0 la valeur annotée $(1, \{x_0\})$ produit la valeur annotée $(5, \{x_0\})$. L'annotation contenant x_0 reflète le fait que, même si la valeur retournée est la constante 5, elle ne l'est qu'après avoir évalué une condition qui dépend de la valeur initiale de x_0.

L'intuition derrière les annotations correspond d'une notion de *teinte* colorée. En effet, on peut considérer que chaque variable initiale du programme comporte une teinte qui lui est propre. Ensuite, cette teinte marque toutes les valeurs obtenues en combinant la valeur de cette variable avec une autre valeur, ou bien selon un test lui même marqué par cette teinte. Intuitivement, chaque calcul produit un résultat teinté par les variables dont il dépend, ce que nous allons formaliser par la suite.

Question 7 —Définition de la sémantique non standard.

Définir formellement la sémantique non standard à base de teintes qui produit les comportements décrits ci-dessus (on conseille de vérifier cela soigneusement sur chaque exemple). Plus précisément :

- *la sémantique d'une expression e est une fonction* $\overline{[\![e]\!]}_{\mathtt{e}} : \overline{\mathbb{M}} \to \overline{\mathbb{V}}$ *;*
- *la sémantique d'une commande c est une fonction* $\overline{[\![c]\!]}_{\mathtt{c}} : \mathcal{P}(\mathbb{X}) \times \overline{\mathbb{M}} \to \overline{\mathbb{V}}$ *qui prend en arugment un ensemble de variables décrivant les teintes de la branche courante en plus d'une mémoire annotée ;*
- *la sémantique d'un programme p est une fonction* $\overline{[\![p]\!]}_{\mathtt{p}} : \mathbb{M} \to \overline{\mathbb{V}}$ *et reflète le fait que l'on part toujours avec une mémoire où on annote chaque variable par un singleton ne comprenant qu'elle même (l'argument de cette dernière sémantique est donc bien* $\mathbb{M}$ *et non* $\overline{\mathbb{M}}$*).*

Question 8 —Lien avec la sémantique standard.

Démontrer que les valeurs produites par les deux sémantiques sont fortement liées. Par exemple, $[\![p]\!]_{\mathtt{p}}(m) = v$ *si et seulement si il existe un ensemble de variables* X *tel que* $\overline{[\![p]\!]}_{\mathtt{p}}(m) = (v, X)$ *; de même* $[\![p]\!]_{\mathtt{p}}$ *renvoie* $\bullet$ *si et seulement si il en va de même pour* $\overline{[\![p]\!]}_{\mathtt{p}}$*.*

Soient deux mémoires annotées $\overline{m}_0, \overline{m}_1$ et un ensemble de variables X. On dit que $\overline{m}_1$ est similaire à $\overline{m}_0$ vis à vis de l'ensemble de teintes X et on note $\overline{m}_0 \bowtie_X \overline{m}_1$ si et seulement si elles ont même domaine (i.e., $|\overline{m}_0| = |\overline{m}_1|$) et :

$$\forall x \in \mathbb{X},\ \forall (v_0, X_0), (v_1, X_1) \in \overline{\mathbb{V}},\ \overline{m}_0(x) = (v_0, X_0) \wedge \overline{m}_1(x) = (v_1, X_1) \Longrightarrow \left\{ \begin{array}{ll} & (X_0 = X_1) \\ \wedge & (X_0 \subseteq X \Longrightarrow v_0 = v_1) \end{array} \right.$$

Intuitivement, cela signifie que $\overline{m}_0$ et $\overline{m}_1$ définissent les mêmes teintes et produisent la même valeur pour toute variable dont les teintes sont incluses dans X (autrement dit, tout ce qui sera calculé en utilisant seulement les teintes X donnera le même résultat, que ce soit avec $\overline{m}_0$ ou avec $\overline{m}_1$).

Question 9 —Propriétés de $\bowtie_X$.

Montrer que $\bowtie_X$ *est une relation d'équivalence. Montrer que, pour tous* $X, X' \subseteq \mathbb{X}$ *et toutes mémoires annotées* $\overline{m}_0, \overline{m}_1$*, on a :*

$$\overline{m}_0 \bowtie_X \overline{m}_1 \wedge X' \subseteq X \Longrightarrow \overline{m}_0 \bowtie_{X'} \overline{m}_1$$

Nous allons montrer dans les questions suivantes que la sémantique non-standard à base de teintes permet de prouver l'absence de flot d'information. Attention : si une preuve ne marche pas, il est possible que votre sémantique non-standard soit incorrecte.

Question 10 —Teintes et expressions.

Soient une expression e, deux mémoires $\overline{m}_0, \overline{m}_1$. On suppose que $\overline{[\![e]\!]}_{\mathsf{e}}(\overline{m}_0)$ s'évalue en $(v, X) \in \overline{\mathbb{V}}$ et que $\overline{m}_0 \bowtie_X \overline{m}_1$.
Montrer que $\overline{[\![e]\!]}_{\mathsf{e}}(\overline{m}_1) = (v, X) = \overline{[\![e]\!]}_{\mathsf{e}}(\overline{m}_0)$.

Question 11 —Teintes et commandes (contrôle).

Soient une commande c, une mémoire $\overline{m}$ et $X_c \subseteq \mathbb{X}$ un ensemble de variables décrivant les teintes liées au contrôle, au début de l'éxécution de c. On suppose que $\overline{[\![c]\!]}_{\mathsf{c}}(X_c, \overline{m}) \neq \bullet$ et s'évalue en $(v, X) \in \overline{\mathbb{V}}$. Montrer que $X_c \subseteq X$.

Question 12 —Teintes et commandes (résultat final).

Soient une commande c, deux mémoires $\overline{m}_0, \overline{m}_1$ et un ensemble de variables X_c. On suppose que $\overline{[\![c]\!]}_{\mathsf{c}}(X_c, \overline{m}_0) \neq \bullet$ et s'évalue en $(v, X) \in \overline{\mathbb{V}}$ et que $\overline{m}_0 \bowtie_X \overline{m}_1$.
Montrer que $\overline{[\![c]\!]}_{\mathsf{c}}(\overline{m}_1) = (v, X) = \overline{[\![c]\!]}_{\mathsf{c}}(\overline{m}_0)$.

Question 13 —Teintes et programmes.

En s'inspirant des questions précédentes, énoncer et prouver une propriété pour un programme complet $\mathbf{input}(X)$; *c.*

Analyse de teintes

Dans la partie précédente, nous avons mis au point une sémantique non-standard qui propage des teintes tout en calculant le résultat du programme. Une analyse statique a en général pour but d'inférer une propriété sémantique *sans évaluer* le programme. Nous allons donc mettre au point une analyse statique, appelée *analyse de teintes*.

On définit les domaines ci-dessous :

$$\begin{array}{rcll} \overline{\mathbb{V}}^\sharp & = & \mathcal{P}(\mathbb{X}) & \text{abstraction des valeurs annotées} \\ \overline{\mathbb{M}}^\sharp & = & \mathbb{X} \to \overline{\mathbb{V}}^\sharp & \text{abstraction des mémoires annotées} \end{array}$$

Comme plus haut, on note $\overline{\mathbb{M}}^\sharp_X$ le sous-ensemble $X \to \overline{\mathbb{V}}^\sharp$ de $\overline{\mathbb{M}}^\sharp$.

On fixe les relations d'abstractions $\vdash_{\mathbb{V}} \subseteq \mathcal{P}(\overline{\mathbb{V}}) \times \overline{\mathbb{V}}^\sharp$ et $\vdash_{\mathbb{M}} \subseteq \mathcal{P}(\overline{\mathbb{M}}) \times \overline{\mathbb{M}}^\sharp$:

$$\begin{array}{rcl} \forall \overline{V} \subseteq \overline{\mathbb{V}},\ \forall \overline{v}^\sharp \in \overline{\mathbb{V}}^\sharp, \quad \overline{V} \vdash_{\mathbb{V}} \overline{v}^\sharp & \iff & \forall (v, X) \in \overline{V},\ X \subseteq \overline{v}^\sharp \\ \forall \overline{M} \subseteq \overline{\mathbb{M}},\ \forall \overline{m}^\sharp \in \overline{\mathbb{M}}^\sharp, \quad \overline{M} \vdash_{\mathbb{M}} \overline{m}^\sharp & \iff & \forall x \in \mathbb{X},\ \{\overline{m}(x) \mid \overline{m} \in \overline{M}\} \vdash_{\mathbb{V}} \overline{m}^\sharp(x) \end{array}$$

Intuitivement, $\vdash_{\mathbb{V}}$ abstrait les valeurs annotées en ne retenant que les teintes, tandis que $\vdash_{\mathbb{M}}$ accomplit la même chose au niveau des mémoires annotées.

Question 14 —Abstraction à base de teintes.

Montrer que chacune des relations $\vdash_{\mathbb{V}}$ et $\vdash_{\mathbb{M}}$ permet de définir une correspondance de Galois. On précisera les relations d'ordre qui doivent être utilisées. Expliquer comment abstraction traite $\bullet$.

Question 15 —Analyse de teintes et correction.

Sur la base de ce qui précède, proposer des sémantiques abstraites $[\![e]\!]^\sharp_{\mathtt{e}}$, $[\![c]\!]^\sharp_{\mathtt{c}}$ *et* $[\![p]\!]^\sharp_{\mathtt{p}}$ *avec les types ci-dessous :*

$$\begin{array}{rrcl} [\![e]\!]^\sharp_{\mathtt{e}} : & \overline{\mathbb{M}}^\sharp & \to & \overline{\mathbb{V}}^\sharp \\ [\![c]\!]^\sharp_{\mathtt{c}} : & \overline{\mathbb{M}}^\sharp \times \mathcal{P}(\mathbb{X}) & \to & \overline{\mathbb{V}}^\sharp \\ [\![p]\!]^\sharp_{\mathtt{p}} & \in \overline{\mathbb{V}}^\sharp & & \end{array}$$

*On montrera aussi les relations de correction satisfaites par chacune de ces sémantiques. (*Hint *: on pourra lier les sémantiques abstraites aux sémantiques concrêtes "relevées" aux ensembles, c'est-à-dire opérant sur des ensembles de mémoires et produisant des ensembles de valeurs).*
Interpréter $[\![p]\!]^\sharp_{\mathtt{p}}$ *au vu de la question 13. (Facultatif : on pourra faire le lien avec la notion de non-interférence vue en cours)*
Donner le résultat de l'analyse sur les exemples donnés dans la section précédente.

Ajout de fonctions

Jusqu'à présent, les programmes considérés sont relativement simples puisqu'ils ne contiennent aucun principe itératif. Nous allons donc étendre la syntaxe des programmes en ajoutant une fonction qui pourra être récursive (rajouter un ensemble de fonctions récursives mutuelles ne serait pas plus difficile, mais alourdirait les notations). La nouvelle syntaxe est la suivante :

e	$::=$	v	constante $c \in \mathbb{V}$
	$\mid$	x	variable $x \in \mathbb{X}$
	$\mid$	$e_0 \oplus e_1$	opération binaire $\oplus \in \{+, -, *, \ldots\}$
c	$::=$	e	expression simple
	$\mid$	$\mathbf{var}\ x = e_0\ \mathbf{in}\ c_1$	(re)-définition d'une variable
	$\mid$	$\mathbf{var}\ x = f(e_0, \ldots, e_k)\ \mathbf{in}\ c_1$	(re)-définition d'une variable
	$\mid$	$\mathbf{if}\ x \geq 0\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1$	condition
f	$::=$	$\mathbf{fun}\, f(x_0, \ldots, x_k)\{c_f\}$	
p	$::=$	$f; \mathbf{input}(\{x_0, \ldots, x_n\});\ c$	programme

On adopte la convention que seules les variables arguments de la fonction peuvent apparaître dans son corps ($x_0, \ldots, x_k$ avec les notations ci-dessus).

Question 16 —Extension naïve des sémantiques.

Procéder à l'extension de la première sémantique que nous avons vue (la sémantique standard), en rajoutant simplement une règle pour l'appel de fonction, qui procède par "inlining". Expliquer pourquoi cette solution ne va pas déboucher sur une analyse satisfaisante ?

Pour résoudre ce problème, nous allons donner une sémantique à la définition de la fonction f à l'aide d'un point-fixe.

Question 17 —Sémantique de la définition de fonction.

Supposons tout d'abord que la sémantique de f est connue et décrite par une fonction $\Phi : \overline{\mathbb{M}}_{\{x_0,\ldots,x_k\}} \to \overline{\mathbb{V}}$. Exprimer sous cette hypothèse une nouvelle sémantique non-standard des commandes, en incluant le cas d'un appel de fonction. Cette sémantique prendra la forme d'une fonction $\overline{[\![c]\!]}_{\mathtt{c},\Phi} : \mathcal{P}(\mathbb{X}) \times \overline{\mathbb{M}} \to \overline{\mathbb{V}}$. Prendre soin à bien évaluer la propagation des teintes.
En déduite une définition par point fixe du corps de la fonction.

Question 18 —Analyse de teintes.

Déduire de la question précédente une analyse statique pour un programme comportant une fonction.