Examen Programmation I (2018-19)

On insistera sur la correction et la clarté des réponses.

⋆ ⋆ ⋆

On étend le langage Imp du cours avec des allocations mémoire. Par souci de simplicité, les seules structures que l'on pourra allouer seront des couples de deux valeurs.

Commandes

c	::=	$x := e$	affectation
	|	`skip`	ne rien faire
	|	$c_1; c_2$	séquence
	|	`if` e `then` c_1 `else` c_2	conditionnelle
	|	`while` e `do` c	boucle while
	|	`new` x	allocation
	|	$x.\ell := e$	mutation ($\ell \in \{1, 2\}$)
	|	`gc`	récupération mémoire

Expressions

e	::=	x	variables
	|	$\dot{n}$	constante entière ($n \in \mathbb{Z}$)
	|	$e \dot{+} e$	addition
	|	$\dot{-} e$	opposé
	|	$e.\ell$	(sélection, $\ell \in \{1, 2\}$)
	|	`null`	(pointeur nul)

Il y a trois commandes et deux expressions en plus, comparativement à Imp, en bas de chaque colonne.

Nous avons deux sortes de valeurs : les entiers relatifs, et les *adresses*, prises dans un ensemble infini dénombrable $Addr$. On distingue une valeur spéciale $\texttt{NULL} \in Addr$. (Noter la différence de capitalisation avec `null` !)

Les valeurs sont dans la somme disjointe $Val \stackrel{\text{def}}{=} \mathbb{Z} \uplus Addr$. Une *mémoire* est une fonction (partielle) $\mu \colon Addr \to Val \times Val$, de domaine dom μ fini et ne contenant pas `NULL`. On dira que μ est *bien formée* si et seulement si pour tout $a \in \text{dom } \mu$, $\{\mu(a)[1], \mu(a)[2]\} \cap Addr \subseteq \text{dom } \mu \cup \{\texttt{NULL}\}$. Ceci signifie que μ ne contient aucun pointeur vers un objet inexistant.

Un *environnement* est une fonction (totale) $\rho \colon Var \to Val$, où Var est l'ensemble (fini) des variables.

Pour tout couple $p \in Val \times Val$, on note $p[\ell]$ sa ℓème composante, $\ell \in \{1, 2\}$. Une somme $a + b$ ($a, b \in Val$) est *définie* si et seulement a et b sont définies et dans $\mathbb{Z}$. L'expression $-a$ ($a \in Val$) est *définie* si et seulement si a est définie et dans $\mathbb{Z}$. L'expression $\mu(a)$ est *définie* si et seulement si a est définie et dans dom μ. L'expression $p[\ell]$ est *définie* ($p \in Val \times Val$, $\ell \in \{1, 2\}$) si et seulement si p est définie.

La sémantique (dénotationnelle) des expressions est donnée par :

$$
\begin{aligned}
[\![x]\!]\rho\mu &= \rho(x)\\
[\![\dot{n}]\!]\rho\mu &= n\\
[\![e_1 \dot{+} e_2]\!]\rho\mu &= \begin{cases} [\![e_1]\!]\rho\mu + [\![e_2]\!]\rho\mu & \text{si définie}\\ \texttt{Wrong} & \text{sinon}\end{cases}\\
[\![\dot{-} e]\!]\rho\mu &= \begin{cases} -[\![e]\!]\rho\mu & \text{si définie}\\ \texttt{Wrong} & \text{sinon}\end{cases}\\
[\![e.\ell]\!]\rho\mu &= \begin{cases} \mu([\![e]\!]\rho\mu)[\ell] & \text{si définie}\\ \texttt{Wrong} & \text{sinon}\end{cases}\\
[\![\texttt{null}]\!]\rho\mu &= \texttt{NULL}.
\end{aligned}
$$

La sémantique opérationnelle travaille sur des configurations (C, ρ, μ), où C est une liste de commandes—on prend des notations proches de celles du cours. On a, additionnellement, une configuration d'erreur, que l'on notera `WRONG` (à ne pas confondre avec `Wrong`, même si l'intention est la même).

$$
\begin{aligned}
(x := e \cdot C, \rho, \mu) &\to (C, \rho[x \mapsto [\![e]\!]\rho\mu], \mu) && \text{si } [\![e]\!]\rho\mu \neq \texttt{Wrong} && (1)\\
(x := e \cdot C, \rho, \mu) &\to \texttt{WRONG} && \text{si } [\![e]\!]\rho\mu = \texttt{Wrong} && (2)\\
(\texttt{skip} \cdot C, \rho, \mu) &\to (C, \rho, \mu) && && (3)\\
(c_1; c_2 \cdot C, \rho, \mu) &\to (c_1 \cdot c_2 \cdot C, \rho, \mu) && && (4)\\
(\texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2 \cdot C, \rho, \mu) &\to (c_1 \cdot C, \rho, \mu) && \text{si } [\![e]\!]\rho\mu \in \mathbb{Z} \setminus \{0\} && (5)\\
(\texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2 \cdot C, \rho, \mu) &\to (c_2 \cdot C, \rho, \mu) && \text{si } [\![e]\!]\rho\mu = 0 && (6)\\
(\texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2 \cdot C, \rho, \mu) &\to \texttt{WRONG} && \text{si } [\![e]\!]\rho\mu \notin \mathbb{Z} && (7)\\
(\texttt{while } e \texttt{ do } c \cdot C, \rho, \mu) &\to (c \cdot \texttt{while } e \texttt{ do } c \cdot C, \rho, \mu) && \text{si } [\![e]\!]\rho\mu \in \mathbb{Z} \setminus \{0\} && (8)\\
(\texttt{while } e \texttt{ do } c \cdot C, \rho, \mu) &\to (C, \rho, \mu) && \text{si } [\![e]\!]\rho\mu = 0 && (9)\\
(\texttt{while } e \texttt{ do } c \cdot C, \rho, \mu) &\to \texttt{WRONG} && \text{si } [\![e]\!]\rho\mu \notin \mathbb{Z} && (10)
\end{aligned}
$$

$$
\begin{aligned}
(\texttt{new } x \cdot C, \rho, \mu) &\to (C, \rho[x \mapsto a], \mu[a \mapsto (\texttt{NULL}, \texttt{NULL})]) && && (11)\\
&\qquad \text{où } a \in \mathit{Addr} \setminus (\operatorname{dom}\mu \cup \{\texttt{NULL}\}))\\
(x.1 := e \cdot C, \rho, \mu) &\to (C, \rho, \mu[a \mapsto (b, \mu(a)[2]) && && (12)\\
&\qquad \text{si } a \stackrel{\text{def}}{=} \rho(x) \in \operatorname{dom}\mu \text{ et } b \stackrel{\text{def}}{=} [\![e]\!]\rho\mu \neq \texttt{Wrong}\\
(x.1 := e \cdot C, \rho, \mu) &\to \texttt{WRONG} && \text{sinon} && (13)\\
(x.2 := e \cdot C, \rho, \mu) &\to (C, \rho, \mu[a \mapsto (\mu(a)[1], b) && && (14)\\
&\qquad \text{si } a \stackrel{\text{def}}{=} \rho(x) \in \operatorname{dom}\mu \text{ et } b \stackrel{\text{def}}{=} [\![e]\!]\rho\mu \neq \texttt{Wrong}\\
(x.2 := e \cdot C, \rho, \mu) &\to \texttt{WRONG} && \text{sinon} && (15)\\
(\texttt{gc} \cdot C, \rho, \mu) &\to (C, \rho, \mathit{gc}(\rho, \mu)) && && (16)
\end{aligned}
$$

La fonction gc sera définie plus bas. Nous ferons en fait varier la définition de cette fonction, et nous comparerons les sémantiques obtenues en section 1. Nous parlerons ensuite de typage.

1. Cette sémantique opérationnelle est-elle déterministe ? Justifier.

Non : la règle (11) ne l'est pas. Il y une infinité d'adresses a possibles.

1 Récupération de mémoire

Pour toute partie B de $Addr$, pour toute mémoire μ, on définit $\mu[B]$ comme étant l'ensemble des adresses pointées par une adresse de B via μ, autrement dit :

$$\mu[B] \stackrel{\text{def}}{=} \bigcup_{a \in B \cap \text{dom } \mu} \{\mu(a)[1], \mu(a)[2]\} \cap Addr$$

Pour toute partie A de $Addr$, pour toute mémoire μ, on définit l'application $F^\mu_A \colon \mathbb{P}(Addr) \to \mathbb{P}(Addr)$ par :

$$F^\mu_A(B) \stackrel{\text{def}}{=} A \cup \mu[B].$$

2. Pourquoi F^μ_A a-t-elle un plus petit point fixe ?

   F^μ_A est monotone, et $\mathbb{P}(Addr)$ est un treillis complet, et on applique Tarski. On peut aussi observer que F^μ_A est Scott-continue, et utiliser Kleene.

3. On note $R(\mu, A)$ le plus petit point fixe de F^μ_A, et on l'appelle la *partie accessible* de μ à partir de A. Montrer que $R(\mu, A)$ est la plus petite partie B de $Addr$ contenant A et telle que $\mu[B] \subseteq B$.

   Ainsi, pour démontrer une inclusion de la forme $R(\mu, A) \subseteq B$ dans la suite, il suffira de démontrer que B contient A et $\mu[B] \subseteq B$. (Pour citer ce résultat, dites « par (R) ».)

   Par la preuve de Tarski, $R(\mu, A)$ est le plus petit pré-point-fixe de F^μ_A, c'est-à-dire la plus petite partie B telle que $A \cup \mu[B] \subseteq B$: c'est la condition demandée.

4. Montrer que si μ est bien formée, et si $A \subseteq \text{dom } \mu \cup \{\texttt{NULL}\}$, alors $R(\mu, A) \subseteq \text{dom } \mu \cup \{\texttt{NULL}\}$.

   Par (R), ou bien en utilisant directement le fait que $R(\mu, A)$ est un point fixe de F^μ_A, il suffit de montrer que $B \stackrel{def}{=} \text{dom } \mu \cup \{\texttt{NULL}\}$ contient A et que pour tout $a \in B \cap \text{dom } \mu$, $\{\mu(a)[1], \mu(a)[2]\} \cap Addr$ est inclus dans B.

   Que B contienne A est l'une des hypothèses.

   Soit $a \in B \cap \text{dom } \mu$. Comme μ est bien formée, $\{\mu(a)[1], \mu(a)[2]\} \cap Addr$ est inclus dans $\text{dom } \mu \cup \{\texttt{NULL}\} = B$: fini.

On pose $Root(\rho) \stackrel{\text{def}}{=} \{\rho(x) \mid x \in Var\} \cap Addr$. C'est l'ensemble des *racines* de ρ.

On définit deux stratégies de récupération mémoire gc :

— $gc^{\max}$ est la fonction qui à tout environnement ρ et à toute mémoire μ associe la restriction $\mu_{|R(\mu, Root(\rho))}$—elle libère toutes les données inaccessibles ;

— $gc^{\min}_\rho$ est la fonction qui à ρ et μ associe simplement μ—elle ne libère rien du tout.

Notre but va être de montrer que la sémantique de notre langage est la même (dans un sens à préciser), que l'on prenne gc égal à $gc^{\max}$ ou à $gc^{\min}$.

On admettra le lemme :

Lemme 1 *Si $(C, \rho, \mu) \to (C', \rho', \mu')$ par une règle autre que (16) et si μ est bien formée, alors μ' est bien formée. Si de plus $Root(\rho) \subseteq \text{dom } \mu \cup \{\texttt{NULL}\}$ alors $Root(\rho') \subseteq \text{dom } \mu' \cup \{\texttt{NULL}\}$.*

On souhaite montrer le lemme :

Lemme 2 *Si* $(C, \rho, \mu) \to (C', \rho', \mu')$ *par une règle autre que (16), si* μ *est bien formée, et si* $Root(\rho) \subseteq \mathrm{dom}\ \mu \cup \{\texttt{NULL}\}$*, alors* $(C, \rho, gc^{\max}(\rho, \mu)) \to (C', \rho', gc^{\max}(\rho', \mu'))$ *par une règle autre que (16).*

On effectue pour cela une analyse de cas, sur chacune des règles possibles ayant permis l'étape de calcul $(C, \rho, \mu) \to (C', \rho', \mu')$. Les cas des règles (1)–(10) sont évidents, et seront omis.

5. Pour tout environnement ρ, pour toute mémoire bien formée μ telle que $Root(\rho) \subseteq \mathrm{dom}\ \mu \cup \{\texttt{NULL}\}$, pour toute $a \in Addr \setminus (\mathrm{dom}\ \mu \cup \{\texttt{NULL}\})$, si l'on pose $\rho' \stackrel{\text{def}}{=} \rho[x \mapsto a]$ et $\mu' \stackrel{\text{def}}{=} \mu[a \mapsto (\texttt{NULL}, \texttt{NULL})]$, montrer que $R(\mu', Root(\rho')) = R(\mu, Root(\rho)) \cup \{a, \texttt{NULL}\}$. On demande une preuve rigoureuse ; tout argument « intuitif » sera ignoré. On ne peut pas utiliser le lemme 2, non plus, puisque c'est ce que nous cherchons à démontrer.
   Erratum : ceci n'est valide que si $\rho(x)$ n'est, par exemple, pas une adresse. L'inclusion $\subseteq$ est valide, et l'inclusion $\supseteq$ serait valide si $\rho(x)$ était supposée n'être pas une adresse. On supposera cependant la question juste pour la suite.

   ($\subseteq$) Il suffit de montrer que $B \stackrel{def}{=} R(\mu, Root(\rho)) \cup \{a, \texttt{NULL}\}$ *contient* $Root(\rho')$ *et que* $\mu'[B] \subseteq B$*, par (R).*
   B contient $R(\mu, Root(\rho))$ *donc* $Root(\rho)$*, donc* $Root(\rho) \setminus \{\rho(x)\}$*. Il contient aussi a. Or* $Root(\rho') = (Root(\rho) \setminus \{\rho(x)\}) \cup \{a\}$*, donc B contient* $Root(\rho')$*.*
   On montre $\mu'[B] \subseteq B$*. Pour tout* $a' \in B \cap \mathrm{dom}\ \mu'$*, soit* a' *est dans* $R(\mu, Root(\rho))$ *soit* $a' = a$*.*

   — *Dans le premier cas, comme* $\mu[R(\mu, Root(\rho))] \subseteq R(\mu, Root(\rho))$*,* $\{\mu(a')[1], \mu(a')[2]\} \cap Addr \subseteq R(\mu, Root(\rho)) \subseteq B$*. Et* $\mu(a') = \mu'(a')$ *car* $a' \neq a$ *(en effet,* $a \notin \mathrm{dom}\ \mu \cup \{\texttt{NULL}\}$ *par hypothèse, donc* $a \notin R(\mu, Root(\rho))$ *par la question 4, le fait que* μ *est bien formée, et le fait que* $Root(\rho) \subseteq \mathrm{dom}\ \mu \cup \{\texttt{NULL}\}$ *par hypothèse ; or* a' *est dans* $R(\mu, Root(\rho))$*). Donc* $\{\mu'(a')[1], \mu'(a')[2]\} \cap Addr \subseteq B$*.*

   — *Si* $a' = a$*, on a* $\{\mu'(a')[1], \mu'(a')[2]\} \cap Addr = \{\texttt{NULL}\}$*, qui est donc bien inclus dans B.*

   ($\supseteq$) C'était la direction fausse. J'indiquerai où est l'erreur plus bas.
   On doit déjà montrer que $\{a, \texttt{NULL}\} \subseteq R(\mu', Root(\rho'))$*. On a* $a \in Root(\rho') \subseteq R(\mu', Root(\rho'))$*, et* $\texttt{NULL} \in \{\mu'(a)[1], \mu'(a)[2]\} \cap Addr (= \{\texttt{NULL}\}) \subseteq \mu'[R(\mu', Root(\rho'))] \subseteq R(\mu', Root(\rho'))$*.*

   On doit ensuite montrer $R(\mu, Root(\rho)) \subseteq R(\mu', Root(\rho'))$*. Posons* $B \stackrel{def}{=} R(\mu', Root(\rho'))$*. Par (R), il suffit de démontrer que B contient* $Root(\rho)$ *et que* $\mu[B] \subseteq B$*.*

   — *On a* $Root(\rho') = Root(\rho) \cup \{a\}$ *[c'est l'erreur !]* $\subseteq B$*, par définition.*

   — *Pour tout* $a' \in B \cap \mathrm{dom}\ \mu$*, on a aussi* $a' \in B \cap \mathrm{dom}\ \mu'$ *puisque* $\mathrm{dom}\ \mu \subseteq \mathrm{dom}\ \mu'$*. Comme* $\mu'[B] \subseteq B$*,* $\{\mu'(a')[1], \mu'(a')[2]\} \cap Addr$ *est inclus dans B. Or* $a' \neq a$*, parce que* $a' \in \mathrm{dom}\ \mu$ *et* $a \notin \mathrm{dom}\ \mu$*, donc* $\mu'(a') = \mu(a')$*. Il s'ensuit que* $\{\mu(a')[1], \mu(a')[2]\} \cap Addr \subseteq B$*. On a montré que pour tout* $a' \in B \cap \mathrm{dom}\ \mu$*,* $\{\mu(a')[1], \mu(a')[2]\} \cap Addr \subseteq B$*, c'est-à-dire que* $\mu[B] \subseteq B$*.*

6. Démontrer le lemme 2 dans le cas où la règle utilisée est (11).

   Suite à l'erratum de la question 5, il n'était possible de démontrer ceci que si $\rho(x)$ *n'est pas une adresse. Mais nous ferons comme si la question était juste.*

On utilise la même règle, avec la même adresse a. Pour ceci, on doit démontrer $a \in Addr \setminus (\text{dom } gc^{\max}(\rho,\mu) \cup \{\texttt{NULL}\})$. On sait que $a \in Addr \setminus (\text{dom } \mu \cup \{\texttt{NULL}\})$, montrons que a ne peut pas être dans $\text{dom } gc^{\max}(\rho,\mu)$. Il suffit pour cela d'observer que $\text{dom } gc^{\max}(\rho,\mu) \subseteq \text{dom } \mu$, ce qui est évident vu que $gc^{\max}(\rho,\mu)$ s'exprime comme une restriction de μ.

On doit maintenant démontrer que $gc^{\max}(\rho',\mu') = gc^{\max}(\rho,\mu)[a \mapsto (\texttt{NULL},\texttt{NULL})]$, où $\rho' \stackrel{def}{=} \rho[x \mapsto a]$ et $\mu' \stackrel{def}{=} \mu[a \mapsto (\texttt{NULL},\texttt{NULL})]$, autrement dit que $\mu'_{|R(\mu',Root(\rho'))} = \mu_{|R(\mu,Root(\rho))}[a \mapsto (\texttt{NULL},\texttt{NULL})]$.

Pour ceci, on doit calculer :

- *$Root(\rho')$: qui vaut $Root(\rho) \cup \{a\}$;*
- *$R(\mu', Root(\rho'))$: qui vaut $R(\mu, Root(\rho)) \cup \{a, \texttt{NULL}\}$ par la question précédente (qui s'applique vu que μ est bien formée, et considérant les hypothèses d'application de la règle).*

Le domaine de $\mu'_{|R(\mu',Root(\rho'))}$ est donc l'intersection de $\text{dom } \mu' = \text{dom } \mu \cup \{a\}$ avec $R(\mu', Root(\rho')) = R(\mu, Root(\rho)) \cup \{a, \texttt{NULL}\}$. (Erratum :

Par la question 4 (utilisant le fait que μ est bien formée, et le fait que $Root(\rho) \subseteq \text{dom } \mu \cup \{\texttt{NULL}\}$), on a $R(\mu, Root(\rho)) \subseteq \text{dom } \mu$. De plus, a n'est pas dans $\text{dom } \mu \cup \{\texttt{NULL}\}$, il n'est donc pas non plus dans $R(\mu, Root(\rho))$. Il s'ensuit que le domaine de $\mu'_{|R(\mu',Root(\rho'))}$ est l'union disjointe de $R(\mu, Root(\rho))$ et de a. C'est aussi le domaine de $\mu_{|R(\mu,Root(\rho))}[a \mapsto (\texttt{NULL},\texttt{NULL})]$, et les deux fonctions coincident tant sur $R(\mu, Root(\rho))$ que sur a. Elles sont donc égales.

On admettra les autres cas. Nous pouvons désormais admettre le lemme 2.

7. Soit μ une mémoire, e une expression, ρ un environnement. Montrer que si μ' est la restriction de μ à un ensemble A, et si $[\![e]\!]\rho\mu' \neq \texttt{Wrong}$ alors $[\![e]\!]\rho\mu' = [\![e]\!]\rho\mu$.

   Par récurrence sur e.

   C'est évident si $e = x$, si $e = \texttt{null}$, ou si $e = \dot{n}$. Si $e = e_1 \dot{+} e_2$, comme $[\![e]\!]\rho\mu' \neq \texttt{Wrong}$, c'est que la somme $[\![e_1]\!]\rho\mu' + [\![e_2]\!]\rho\mu'$ est définie. En particulier, $[\![e_1]\!]\rho\mu'$ et $[\![e_2]\!]\rho\mu'$ sont différentes de $\texttt{Wrong}$, donc égales respectivement à $[\![e_1]\!]\rho\mu$ et à $[\![e_2]\!]\rho\mu$ par hypothèse de récurrence. On en déduit que $[\![e]\!]\rho\mu' = [\![e_1]\!]\rho\mu + [\![e_2]\!]\rho\mu = [\![e]\!]\rho\mu$.

   Le raisonnement est similaire si e est de la forme $\dot{-}e_1$.

   Si $e = e_1.\ell$ (qui est l'unique cas intéressant), comme $[\![e]\!]\rho\mu' \neq \texttt{Wrong}$, en particulier $[\![e_1]\!]\rho\mu'$ est différent de $\texttt{Wrong}$ donc égal à $[\![e_1]\!]\rho\mu$ par hypothèse de récurrence. Comme μ' est une restriction de μ et $\mu'([\![e_1]\!]\rho\mu)$ est définie, $\mu([\![e_1]\!]\rho\mu)$ l'est aussi (c'est le point important), et donc $\mu([\![e_1]\!]\rho\mu)[\ell]$ aussi ; et ceci est $[\![e]\!]\rho\mu$.

8. Montrer le lemme :

   Lemme 3 *Si $(C,\rho,\mu) \to \texttt{WRONG}$, μ est bien formée, et $Root(\rho) \subseteq \text{dom } \mu \cup \{\texttt{NULL}\}$, alors $(C,\rho,\mu') \to \texttt{WRONG}$, où $\mu' \stackrel{def}{=} gc^{\max}(\rho,\mu)$.*

   On observe d'abord que si $[\![e]\!]\rho\mu = \texttt{Wrong}$ alors $[\![e]\!]\rho\mu' = \texttt{Wrong}$. C'est une conséquence de la question 7 : sinon, $[\![e]\!]\rho\mu' \neq \texttt{Wrong}$ serait égal à $[\![e]\!]\rho\mu$, qui vaut $\texttt{Wrong}$.

   Règle (2). On a $[\![e]\!]\rho\mu = \texttt{Wrong}$, donc $[\![e]\!]\rho\mu' = \texttt{Wrong}$, comme on vient de le voir. Donc la même règle s'applique en remplaçant μ par μ'.

Règle (7). On a $[\![e]\!]\rho\mu \notin \mathbb{Z}$. On en déduit que $[\![e]\!]\rho\mu' \notin \mathbb{Z}$: sinon, $[\![e]\!]\rho\mu'$ serait dans $\mathbb{Z}$, donc différente de `Wrong` *en particulier, et la question 7 impliquerait $[\![e]\!]\rho\mu = [\![e]\!]\rho\mu' \in \mathbb{Z}$. La même règle s'applique donc.*

Règle (10). Même argument.

Règle (13). On a $(x.1 := e \cdot C, \rho, \mu) \to$ `WRONG` *parce que $a \notin \mathrm{dom}\ \mu$ (où $a \stackrel{def}{=} \rho(x)$) ou bien $[\![e]\!]\rho\mu =$* `Wrong`*. Dans le deuxième cas, comme plus haut, on a $[\![e]\!]\rho\mu' =$* `Wrong`*, et on applique la même règle. Dans le premier cas, a n'est pas non plus dans $\mathrm{dom}\ \mu'$ puisque $\mathrm{dom}\ \mu' \subseteq \mathrm{dom}\ \mu$, μ' étant une restriction de μ. On applique alors la même règle.*

Règle (15). Même argument.

9. Montrer que $gc^{\max}(\rho, \_)$ est *idempotente*, au sens où $gc^{\max}(\rho, gc^{\max}(\rho, \mu)) = gc^{\max}(\rho, \mu)$.

   Soit $\mu' = gc^{\max}(\rho, \mu)$, $B = R(\mu, Root(\rho))$, $B' = R(\mu', Root(\rho))$.
   On a $gc^{\max}(\rho, gc^{\max}(\rho, \mu)) = gc^{\max}(\rho, \mu') = \mu'_{|B'} = (\mu_{|B})_{|B'}$, et l'on souhaite montrer que ceci est égal à $\mu' = \mu_{|B}$. Il suffit donc de démontrer que $B \subseteq B'$ (en fait on a $B = B'$).
   On ne peut pas utiliser la question 4, car nous ne faisons pas l'hypothèse que μ est bien formée.
   Par (R), ceci revient à démontrer :
   - *$Root(\rho) \subseteq B'$, ce qui est évident par (R) appliqué à B' ;*
   - *$\mu[B'] \subseteq B'$: μ' est une restriction de μ, donc $\mu[B'] \subseteq \mu'[B'] \subseteq B'$, la dernière inclusion étant par (R) appliqué à B'.*

Appelons *sémantique min* la sémantique opérationnelle déjà donnée dans le cas où $gc = gc^{\min}$, et *sémantique max* celle où $gc = gc^{\max}$.

10. Montrer que si μ est bien formée et $Root(\rho) \subseteq \mathrm{dom}\ \mu \cup \{$`NULL`$\}$, et si $(C, \rho, \mu) \to (C', \rho, \mu')$ (resp., $\to$ `WRONG`) dans la sémantique min, alors $(C, \rho, gc^{\max}(\rho, \mu)) \to (C', \rho, gc^{\max}(\rho, \mu'))$ (resp., $\to$ `WRONG`) dans la sémantique max.

    Errata : les configurations d'arrivée devraient être (C', ρ', μ') et non (C', ρ, μ'), et de même $(C', \rho', gc^{\max}(\rho', \mu'))$ plutôt que $(C', \rho, gc^{\max}(\rho, \mu'))$.

    Si c'est par une règle autre que (16), c'est une conséquence des lemmes 2 et 3. Sinon, on a comme hypothèse que $(\mathtt{gc} \cdot C, \rho, \mu) \to (C, \rho, \mu)$ par (16) en sémantique min, et on souhaite démontrer que $(\mathtt{gc} \cdot C, \rho, gc^{\max}(\rho, \mu)) \to (C, \rho, gc^{\max}(\rho, \mu))$ en sémantique max. Pour ceci, on doit démontrer que $gc^{\max}(\rho, gc^{\max}(\rho, \mu))) = gc^{\max}(\rho, \mu))$, et c'est la question 9.

On peut démontrer (par essentiellement le même argument) que, dans l'autre sens, si $(C, \rho, \mu) \to (C', \rho, \mu')$ (resp., $\to$ `WRONG`) dans la sémantique max, alors $(C, \rho, gc^{\max}(\rho, \mu)) \to (C', \rho, gc^{\max}(\rho, \mu'))$ (resp., $\to$ `WRONG`) dans la sémantique min. On l'admettra.

11. Considérons une trace maximale partant de (C, ρ, μ), et supposons qu'elle soit de la forme $(C, \rho, \mu) \to^* (C', \rho', \mu')$. Pourquoi a-t-on nécessairement $C' = \epsilon$?

    Par inspection, si C' est non vide, donc de la forme $c' \cdot C''$, il existe toujours une règle à appliquer, de $(c' \cdot C'', \rho', \mu')$ vers une autre configuration. Ceci contredirait la maximalité.

On dira que la configuration (C, ρ, μ) *termine* dans l'une quelconque des deux sémantiques si et seulement s'il existe une trace maximale partant de (C, ρ, μ) dans cette sémantique qui soit finie.

12. En déduire que les sémantiques min et max sont *contextuellement équivalentes* : pour toute configuration (C, ρ, μ) où μ est bien formée, $gc^{\max}(\rho, \mu) = \mu$, et $Root(\rho) \subseteq \text{dom}\ \mu \cup \{\texttt{NULL}\}$, (C, ρ, μ) termine en sémantique min si et seulement si (C, ρ, μ) termine en sémantique max.

    Si $(C, \rho, \mu) \to^* (\epsilon, \rho_\infty, \mu_\infty)$ *en sémantique min, par une récurrence sur le nombre d'étapes de réduction (utilisant la question 10 à chaque étape, et le lemme 1 pour s'assurer que ses hypothèses sont vérifiées à chaque étape—autre que (16), mais c'est évident pour cette règle-là), on a* $(C, \rho, gc^{\max}(\rho, \mu)) \to^* (\epsilon, \rho_\infty, gc^{\max}(\rho_\infty, \mu_\infty))$ *en sémantique max. Donc* $(C, \rho, gc^{\max}(\rho, \mu))$ *termine en sémantique max. Par hypothèse,* $gc^{\max}(\rho, \mu) = \mu$*, donc* (C, ρ, μ) *termine en sémantique max.*

    Si $(C, \rho, \mu) \to^* \texttt{Wrong}$ *en sémantique min, de même,* $(C, \rho, \mu) = (C, \rho, \texttt{gc}^{\max}(\rho, \mu)) \to^* \texttt{Wrong}$ *en sémantique max, donc* (C, ρ, μ) *termine aussi en sémantique max.*

    La réciproque suit le même raisonnement, en utilisant la remarque énoncée à la suite de la question 10.

2 Typage

On définit les types suivants :

$\sigma, \tau, \cdots ::= \alpha, \beta, \cdots$	variables de types
$\mid \texttt{int}$	entiers
$\mid \texttt{list}(\tau)$	liste d'éléments de type τ

Un *contexte* Γ est une fonction de Var dans l'ensemble des types. On définit les règles de typage des expressions :

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (Var) \qquad \frac{}{\Gamma \vdash \dot{n} : \texttt{int}}\ (Int)$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \dot{+} e_2 : \texttt{int}}\ (\dot{+}) \qquad \frac{\Gamma \vdash e : \texttt{int}}{\Gamma \vdash \dot{-} e : \texttt{int}}\ (\dot{-})$$

$$\frac{\Gamma \vdash e : \texttt{list}(\tau)}{\Gamma \vdash e.1 : \tau}\ (.1) \qquad \frac{\Gamma \vdash e : \texttt{list}(\tau)}{\Gamma \vdash e.2 : \texttt{list}(\tau)}\ (.2) \qquad \frac{}{\Gamma \vdash \texttt{null} : \texttt{list}(\tau)}\ (\texttt{null})$$

et les règles de typage des commandes, qui dérivent des jugements de la forme $\Gamma \vdash c$ OK (« la commande c est bien typée dans le contexte Γ ») :

$$\frac{\Gamma \vdash x : \tau \quad \Gamma \vdash e : \tau}{\Gamma \vdash x := e \text{ OK}} (:= \text{ OK}) \qquad \frac{}{\Gamma \vdash \texttt{skip OK}} (\texttt{skip OK})$$

$$\frac{\Gamma \vdash c_1 \text{ OK} \quad \Gamma \vdash c_2 \text{ OK}}{\Gamma \vdash c_1; c_2 \text{ OK}} (; \text{ OK}) \qquad \frac{\Gamma \vdash e : \texttt{int} \quad \Gamma \vdash c_1 \text{ OK} \quad \Gamma \vdash c_2 \text{ OK}}{\Gamma \vdash \texttt{if}\, e\, \texttt{then}\, c_1\ \texttt{else}\ c_2 \text{ OK}} (\texttt{if OK})$$

$$\frac{\Gamma \vdash e : \texttt{int} \quad \Gamma \vdash c \text{ OK}}{\Gamma \vdash \texttt{while}\, e\, \texttt{do}\, c \text{ OK}} (\texttt{while OK}) \qquad \frac{\Gamma \vdash x : \texttt{list}(\tau)}{\Gamma \vdash \texttt{new}\, x \text{ OK}} (\texttt{new OK})$$

$$\frac{\Gamma \vdash x : \texttt{list}(\tau) \quad \Gamma \vdash e : \tau}{\Gamma \vdash x.1 := e \text{ OK}} (.1 := \text{ OK}) \qquad \frac{\Gamma \vdash x : \texttt{list}(\tau) \quad \Gamma \vdash e : \texttt{list}(\tau)}{\Gamma \vdash x.2 := e \text{ OK}} (.2 := \text{ OK})$$

$$\frac{}{\Gamma \vdash \texttt{gc OK}} (\texttt{gc OK})$$

13. Montrer que le problème suivant est décidable :
ENTRÉE : une commande c (supposons que les variables qui y apparaissent soient notées $x_1, \ldots, x_n$) ;
QUESTION : existe-t-il un contexte Γ tel que $\Gamma \vdash c$ OK soit dérivable ?
On ne demande pas une preuve complète, mais au moins une description complète de l'algorithme—qui doit être correct, bien entendu.

    Il suffit d'imiter l'algorithme de Hindley. On introduit des variables de type α_e pour toute occurrence de sous-expression e apparaissant dans c, et l'on forme un problème d'unification dont les solutions sont (une fois restreints aux variables de types $\alpha_{x_1}, \ldots, \alpha_{x_n}$) exactement les types $\tau_1, \ldots, \tau_n$ cherchés. J'appellerai sous-expression une occurrence *de sous-expression, pour alléger les notations, mais il faut noter que deux occurrences de la même expression seront associées à deux variables de types distinctes ; c'est important dans le cas de* `null`*, notamment.*

    - *pour chaque sous-expression $\dot{n}$, on ajoute l'équation $\alpha_{\dot{n}} = \texttt{int}$;*
    - *pour chaque sous-expression $e_1 \dot{+} e_2$, on ajoute les équations $\alpha_{e_1} = \texttt{int}$, $\alpha_{e_2} = \texttt{int}$, $\alpha_{e_1 \dot{+} e_2} = \texttt{int}$;*
    - *pour chaque sous-expression $\dot{-}e$, les équations $\alpha_e = \texttt{int}$ et $\alpha_{\dot{-}e} = \texttt{int}$;*
    - *pour chaque sous-expression $e.1$, l'équation $\alpha_e = \texttt{list}(\alpha_{e.1})$;*
    - *pour chaque sous-expression $e.2$, l'équation $\alpha_e = \texttt{list}(\beta)$ et $\alpha_{e.2} = \texttt{list}(\beta)$, où β est une variable de type fraîche ;*
    - *pour chaque occurrence de* `null`*, l'équation $\alpha_{\texttt{null}} = \texttt{list}(\beta)$, où β est une variable de type fraîche ;*
    - *pour chaque commande $x := e$, l'équation $\alpha_x = \alpha_e$;*
    - *pour chaque commande $\texttt{if}\, e\, \texttt{then}\, c_1\ \texttt{else}\ c_2$ ou $\texttt{while}\, e\, \texttt{do}\, c$, l'équation $\alpha_x = \texttt{int}$;*
    - *pour chaque commande $\texttt{new}\, x$, l'équation $\alpha_x = \texttt{list}(\beta)$, où β est une variable de type fraîche ;*

— *pour chaque commande* $x.1 := e$*, l'équation* $\alpha_x = \texttt{list}(\alpha_e)$ *;*

— *pour chaque commande* $x.2 := e$*, les équations* $\alpha_x = \texttt{list}(\beta)$ *et* $\alpha_e = \texttt{list}(\beta)$*, où* β *est une variable de type fraîche.*

On résout ensuite l'ensemble d'équations par l'algorithme d'unification.

Un *typage* d'une mémoire μ bien formée est une application Δ de dom μ vers l'ensemble des types telle que :

— pour toute $a \in \text{dom } \mu$ (= dom Δ), $\Delta(a)$ est un type de la forme $\texttt{list}(\tau)$, où :

— si $a[1] \in \mathit{Addr} \setminus \{\texttt{NULL}\}$ (donc $a[1] \in \text{dom } \mu = \text{dom } \Delta$) alors $\Delta(a[1]) = \tau$;

— si $a[1] \in \mathbb{Z}$ alors $\tau = \texttt{int}$;

— $\Delta(a[2]) = \texttt{list}(\tau)$.

On écrira $\Delta \rtimes \Gamma \models \rho, \mu$ si et seulement si μ est une mémoire bien formée, $\mathit{Root}(\rho) \subseteq \text{dom } \mu \cup \{\texttt{NULL}\}$, Δ est un typage de μ, Γ est un contexte, et pour toute variable x telle que $\rho(x) \in \text{dom } \mu$ (= dom Δ), $\Gamma(x) = \Delta(\rho(x))$.

14. On aimerait montrer que si $\Delta \rtimes \Gamma \models \rho, \mu$ et $\Gamma \vdash (C, \rho, \mu)$ OK alors on n'a pas $(C, \rho, \mu) \to^* \texttt{WRONG}$ (≪ well-typed programs do not go wrong ≫), que ce soit dans la sémantique min ou la sémantique max. Mais ceci est faux : donner un contre-exemple simple, et le justifier.

Pour C*, on peut par exemple prendre* $(y := x.1, \rho, \mu)$*, où* ρ *envoie toute variable vers* NULL*, sauf* y *qui est envoyée vers* 0 *(ou un autre entier), et* μ *est vide. Trivialement,* μ *est bien formée, et* $\mathit{Root}(\rho) = \emptyset$ *est inclus dans* dom $\mu \cup \{\texttt{NULL}\}$*.*

On prend aussi la fonction vide pour Δ*. C'est trivialement un typage de* μ*.*

On définit le contexte Γ *: il envoie chaque variable vers le type* $\texttt{list}(\texttt{int})$*, sauf* y: int*. Comme* Δ *est vide, la condition* $\Gamma(z) = \Delta(\rho(z))$ *pour toute variable* z *telle que* $\rho(z) \in \text{dom } \Delta$ *est trivialement satisfaite.*

Donc $\Delta \rtimes \Gamma \models \rho, \mu$*.*

Par ailleurs, on peut dériver $\Gamma \vdash y := x.1$ OK *:*

$$\dfrac{\dfrac{}{\Gamma \vdash y\colon \texttt{int}}\,(Var) \quad \dfrac{\dfrac{}{\Gamma \vdash x\colon \texttt{list(int)}}\,(Var)}{\Gamma \vdash x.1\colon \texttt{int}}\,(.1)}{\Gamma \vdash y := x.1 \text{ OK}}\,(:=\text{ OK})$$

Mais on a, finalement, le calcul :

$$(y := x.1, \rho, \mu) \to \texttt{WRONG}$$

par (2), puisque $[\![x.1]\!]\rho\mu = \texttt{Wrong}$*, comme* $[\![x]\!]\rho\mu = \texttt{NULL} \notin \text{dom } \mu$*.*