Agrégation externe de mathématiques, session 2017
Épreuve de modélisation, option informatique

Résumé : Le coût *en temps* de l'évaluation d'une expression arithmétique est fonction linéaire de sa taille. Le coût *en espace* dépend de la structure de l'arbre sous-jacent. En mode interprété, l'évaluation se fait au fil de la lecture, mais en mode compilé on peut optimiser l'allocation d'espace. Le texte analyse et compare les coûts dans ces deux situations

Mots clefs : expression arithmétique, évaluation, complexité en moyenne

---

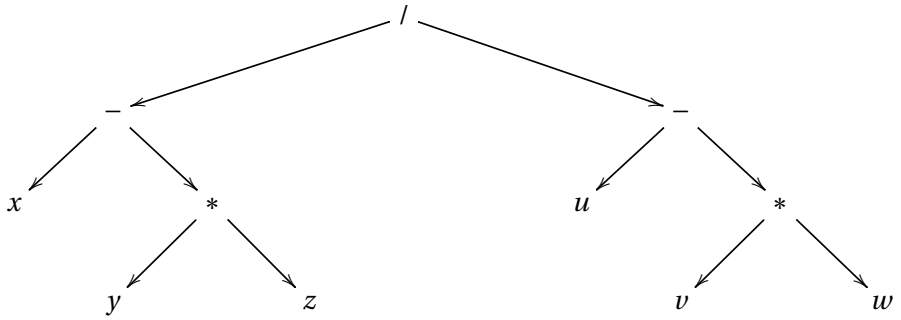
► *Il est rappelé que le jury n'exige pas une compréhension exhaustive du texte. Vous êtes laissé(e) libre d'organiser votre discussion comme vous l'entendez. Des suggestions de développement, largement indépendantes les unes des autres, vous sont proposées en fin de texte. Vous n'êtes pas tenu(e) de les suivre. Il vous est conseillé de mettre en lumière vos connaissances à partir du fil conducteur constitué par le texte. Le jury appréciera que la discussion soit accompagnée d'exemples traités sur ordinateur.*

1. Introduction

Commençons par décrire un modèle des objets manipulés. On se donne un ensemble fini X de lettres désignant des variables ou des constantes explicites (mais nous dirons "variables" dans tous les cas), et un ensemble Ω de symboles d'opérations binaires. Une *expression arithmétique* est définie inductivement : toute lettre de X est une expression arithmétique ; si e_1 et e_2 sont des expressions arithmétiques et si $\omega \in \Omega$ est un symbole d'opération binaire, alors $\omega(e_1, e_2)$ est une expression arithmétique. On définit ainsi l'*algèbre des termes* $\mathcal{T}_{\Omega,X}$. L'emploi, dans ce modèle, d'une notation fonctionnelle (opérateurs manipulés comme des symboles de fonctions, opérandes manipulés comme des arguments) ne préjuge pas de la notation utilisée en pratique. La notation traditionnelle met les opérateurs en position infixe, mais elle est ambiguë : $1-1-1$ admet les deux interprétations $1-(1-1)$ et $(1-1)-1$. On pallie cette ambiguïté par un parenthésage systématique. La notation préfixe et la notation postfixe (ou polonaise) sont non ambiguës sans parenthésage et nécessitent donc moins de symboles. (C'est pourquoi, sur certaines calculettes dont l'écran d'affichage est petit, on utilise la notation postfixe).

Exemples. **1)** Prenons $X = \{u, v, w, x, y, z\}$ et $\Omega = \{\texttt{Add}, \texttt{Sub}, \texttt{Mul}, \texttt{Div}\}$. Nous noterons **Exp** l'ensemble $\mathcal{T}_{\Omega,X}$ ainsi obtenu. Le terme $\texttt{Div}(\texttt{Sub}(x, \texttt{Mul}(y, z)), \texttt{Sub}(u, \texttt{Mul}(v, w)))$ est traditionnellement noté $\dfrac{x-yz}{u-vw}$ (mathématiques) ou bien `(x - y*z)/(u - v*w)` (informatique) ou

encore `x y z * - u v w * - /` (notation postfixe), ou en représentation arborescente :



2) Prenons $X = \{F\}$ et $\Omega = \{N\}$. Alors $\mathscr{T}_{\Omega,X}$ s'identifie à l'ensemble **Arb** des arbres binaires. Le terme $N(N(F,N(F,F)),N(F,N(F,F)))$ désigne l'arbre (non étiqueté) sous-jacent à l'expression ci-dessus.

Pour définir l'*évaluation* d'une expression arithmétique, il faut se donner au départ :

(1) Un *domaine de valeurs* $\mathscr{D}$, par exemple, $\mathscr{D} = \mathbf{R} \cup \{\perp\}$ (le symbole $\perp$ est là pour prendre en compte les valeurs indéfinies, comme $0/0$).

(2) Une application d'*assignation* $x \mapsto \overline{x}$ de X dans $\mathscr{D}$. Par exemple, on assigne à chaque variable x sa valeur $\overline{x}$ dans un contexte et un environnement donnés.

(3) Une *interprétation* de chaque $\omega \in \Omega$ par une application $\overline{\omega} : \mathscr{D} \times \mathscr{D} \to \mathscr{D}$. Par exemple, `Add` est interprété comme l'application $(v_1, v_2) \mapsto v_1 + v_2$, etc.

On définit alors inductivement l'évaluation ev comme une application de $\mathscr{T}_{\Omega,X}$ dans $\mathscr{D}$:

$$\forall x \in X\ ,\ \mathrm{ev}(x) = \overline{x},\ \text{et}\ \forall \omega \in \Omega\ ,\ \forall e_1, e_2 \in \mathscr{T}_{\Omega,X}\ ,\ \mathrm{ev}\big(\omega(e_1,e_2)\big) = \overline{\omega}\big(\mathrm{ev}(e_1),\mathrm{ev}(e_2)\big).$$

Ce schéma est d'ailleurs très général et permet aussi bien des "interprétations abstraites" :

— En prenant $\mathscr{D} = \mathbf{N}$, $\overline{x} = 0$ pour tout x, et $\overline{\omega}(v_1, v_2) = 1 + v_1 + v_2$, on trouve pour $\mathrm{ev}(e)$ le nombre total de symboles d'opérateurs présents dans e. On l'appelle *taille de e* et on le note $|e|$. (Il est bien connu que le nombre de symboles de variables est alors égal à $|e| + 1$.)

— En prenant $\mathscr{D} = \mathbf{Arb}$, $\overline{x} = F$ pour tout x, et $\overline{\omega}(v_1, v_2) = N(v_1, v_2)$, on trouve pour $\mathrm{ev}(e)$ l'arbre binaire sous-jacent au terme e.

Nous nous intéressons ici à l'évaluation numérique des expressions arithmétiques : c'est donc l'application ev : **Exp** $\to \mathbf{R} \cup \{\perp\}$ définie plus haut. Le coût en temps de l'évaluation d'une expression $e \in$ **Exp** est, pour toute méthode "raisonnable", une fonction linéaire de la taille $|e|$ de e, et nous ne nous en occuperons pas. Il n'en est pas de même du coût en espace. La définition récursive donnée plus haut sous-entend la présence d'une pile, donc d'une zone mémoire allouée dynamiquement. Cette mémoire peut être considérée comme "chère" (ressource rare) dans certains cas : en mode interprété sur une petite calculette, dont la pile est de hauteur limitée (c'est un exemple du problème général du "calcul embarqué") ; ou encore, en mode compilé, dans la zone des registres d'un ordinateur. Cette zone dispose d'un bus d'accès direct et rapide à l'UAL (unité arithmétique et logique), ce qui n'est pas le cas de l'ensemble de la mémoire.

() Option informatique

Nous prouverons que, dans le cas d'une évaluation en mode interprété (donc, sans optimisation de l'usage de la mémoire), le coût moyen est *au moins* en $\Theta(\sqrt{n})$; alors qu'en mode compilé, on peut se ramener à un coût *au pire* en $O(\log n)$.

2. Interprétation d'une expression en écriture postfixe

2.1. *Algorithme d'évaluation par pile*

L'évaluation à partir de la représentation postfixe est réalisable au fil de la lecture. On se donne une pile pour ranger des éléments de $\mathscr{D}$. À la fin de l'exécution, le contenu antérieur de la pile a été préservé et le résultat figure en sommet de pile.

```
tant que possible
   (lire un symbole;
    si c'est un symbole de variable x alors empiler la valeur de x;
    si c'est un symbole d'operateur op alors
       (depiler v2; depiler v1; empiler op(v1,v2)));;
```

Théorème 1. *L'algorithme ci-dessus réalise correctement l'évaluation d'une expression écrite en notation postfixe (et laisse la pile en l'état) si, et seulement si, la chaine lue est bien formée (i.e., c'est la représentation postfixe d'une expression). La hauteur maximum occupée pendant l'exécution de l'algorithme peut être calculée inductivement comme suit :*

$$\forall x \in X\,,\ h(x) = 1,\ \textit{et}\ \forall \omega \in \Omega\,,\ \forall e_1, e_2 \in \mathscr{T}_{\Omega,X}\,,\ h\big(\omega(e_1,e_2)\big) = \max\big(h(e_1), 1 + h(e_2)\big).$$

Indication de démonstration. La preuve se fait par induction structurelle sur l'expression (et non par récurrence sur la longueur de la chaine lue). L'hypothèse d'induction prend nécessairement en compte le cas d'une pile qui n'est pas vide au départ.

Exemple. Dans le cas de `x y z * - u v w * - /`, voici les états successifs de la pile :

$$\begin{bmatrix} \\ x \end{bmatrix}, \begin{bmatrix} \\ y \\ x \end{bmatrix}, \begin{bmatrix} z \\ y \\ x \end{bmatrix}, \begin{bmatrix} \\ y*z \\ x \end{bmatrix}, \begin{bmatrix} \\ r_0 \end{bmatrix}, \begin{bmatrix} \\ u \\ r_0 \end{bmatrix}, \begin{bmatrix} v \\ u \\ r_0 \end{bmatrix}, \begin{bmatrix} w \\ v \\ u \\ r_0 \end{bmatrix}, \begin{bmatrix} v*w \\ u \\ r_0 \end{bmatrix}, \begin{bmatrix} \\ u-v*w \\ r_0 \end{bmatrix}, \begin{bmatrix} \\ r \end{bmatrix},$$

où on a posé $r_0 := x - y*z$, $r := (x - y*z)/(u - v*w)$.

Remarque 1. *Cet algorithme reste valable quelle que soit l'interprétation donnée à l'évaluation. Par exemple, si l'on gère une pile d'arbres, il permet de reconstituer l'expression à partir de sa représentation postfixe et fournit donc une preuve de la non-ambiguïté de cette représentation.*

On peut remarquer que la suite des hauteurs de la pile est liée à la structure de la chaine de symboles lue, et en déduire un critère combinatoire pour reconnaitre les chaines bien formées. À une chaine $c = [c_1, \ldots, c_n]$ de symboles de $X \cup \Omega$, on associe la suite $(h_1, \ldots, h_n) \in \mathbf{Z}^n$ ainsi définie : $h_k = \sum_{i=1}^{k} \big(1 - \alpha(c_i)\big)$, où l'on a noté $\alpha(c)$ l'arité du symbole c : donc 0 pour

les constantes, 2 pour les opérateurs binaires ... Les h_i sont donc les hauteurs successives atteintes par la pile pendant l'exécution de l'algorithme d'évaluation. Dans notre exemple, on trouve : $1,2,3,2,1,2,3,4,3,2,1$.

Corollaire 1. *La chaine c est bien formée si, et seulement si,* $h_1,\ldots,h_{n-1}>0$ *et* $h_n=1$.

2.2. *Coût en espace de l'évaluation*

Puisque l'utilisation du fond de la pile pour rendre le résultat est incontournable, nous considèrons comme *coût en espace* la hauteur maximum atteinte par la pile, diminuée de 1. Le coût minimum 0 correspond à l'évaluation d'une expression réduite à une variable de X. Dans notre exemple, la hauteur maximum atteinte est 4 et le coût est 3.

Corollaire 2. *Soit t l'arbre binaire sous-jacent à une expression e. Le coût en espace de l'évaluation de e ne dépend que de t. C'est le paramètre* $\chi(t)$ *défini inductivement par les règles suivantes :* $\chi(F)=0$ *et* $\chi\big(N(t_1,t_2)\big)=\max\big(\chi(t_1),1+\chi(t_2)\big)$.

Notre but est maintenant l'analyse du coût moyen d'évaluation d'une expression de taille n. On le définit ainsi : notons $\mathbf{Exp}_n$ l'ensemble des expressions de taille n et, temporairement, $\kappa(e)$ le coût en espace de l'évaluation de e. Le coût moyen recherché est :

$$\overline{\kappa}_n=\frac{1}{\operatorname{Card}\,\mathbf{Exp}_n}\sum_{e\in\mathbf{Exp}_n}\kappa(e).$$

La taille de l'expression e est égale à la taille $|t|$ (nombre de noeuds N) de l'arbre binaire sous-jacent t. En oubliant les étiquettes de l'arbre d'évaluation, on obtient

Proposition 1. *On a* $\overline{\kappa}_n=\overline{\chi_n}$, *défini par :* $\overline{\chi_n}=\dfrac{1}{c_n}\displaystyle\sum_{t\in\mathbf{Arb}_n}\chi(t)$, *où* $c_n=\dfrac{1}{n+1}\binom{2n}{n}$.

L'analyse complète de $\overline{\chi_n}$ est difficile, et nous admettrons qu'il existe une constante $K>0$ telle que $\overline{\chi_n}\sim K\sqrt{n}$ lorsque $n\to+\infty$.

3. Compilation et allocation de registres

3.1. *La méthode d'Ershov*

On suppose donnés un tableau de registres `R[0]`,`R[1]`,... et un langage de programmation de ces registres. Les instructions élémentaires sont de la forme `R[i] := x` (affectation) et `R[i] := R[j] op R[k]` pour chacune des opérations binaires `op` possibles. Les indices sont des entiers naturels quelconques.

Le programme d'évaluation de l'expression `(x - y*z)/(u - v*w)` prendrait alors la forme suivante :

```
00: % Allocation gauche-droite          | 06: R[1] := u ;
01: R[0] := x ;                         | 07: R[2] := v ;
02: R[1] := y ;                         | 08: R[3] := w ;
```

```
03: R[2] := z ;                          | 09: R[2] := R[2] * R[3] ;
04: R[1] := R[1] * R[2] ;                | 10: R[1] := R[1] - R[2] ;
05: R[0] := R[0] - R[1] ;                | 11: R[0] := R[0] / R[1] ;
```

À la fin de l'exécution, le registre `R[0]` contient le résultat de l'évaluation. La preuve est en tout point similaire à celle du théorème 1. Il est de même évident que les coûts en temps et en espace sont les mêmes qu'en mode interprété : l'indice maximum de registre utilisé est égal à $\chi(t)$, où t est l'arbre binaire sous-jacent à l'expresion concernée. Cette méthode d'allocation de registres porte le nom de *méthode d'allocation gauche-droite*.

Cependant, la compilation permet d'améliorer l'allocation de registres. On peut mettre à profit les déséquilibres dans l'arbre t pour évaluer d'abord les expressions les plus profondes. Modifions ainsi la deuxième partie du code ci-dessus en évaluant `v*w` avant de charger `u` :

```
00: % Allocation d'Ershov                | 06: R[1] := v ;
01: R[0] := x ;                          | 07: R[2] := w ;
02: R[1] := y ;                          | 08: R[1] := R[1] * R[2] ;
03: R[2] := z ;                          | 09: R[2] := u ;
04: R[1] := R[1] * R[2] ;                | 10: R[1] := R[2] - R[1] ;
05: R[0] := R[0] - R[1] ;                | 11: R[0] := R[0] / R[1] ;
```

L'indice maximum est maintenant 2 au lieu de 3. La méthode d'allocation qui met à profit les déséquilibres est appelée *méthode d'Ershov*. Nous considérerons comme coût en espace de cette méthode l'indice maximum de registre utilisé.

3.2. *Le paramètre ρ*

Lemme 1. *Soit t l'arbre binaire sous-jacent à une expression e. Le coût en espace de l'évaluation de e par la méthode d'Ershov ne dépend que de t. C'est le paramètre $\rho(t)$ défini inductivement par les règles suivantes : $\rho(F) = 0$ et $\rho\big(N(t_1,t_2)\big) = \begin{cases} 1+\rho(t_1) \text{ si } \rho(t_1)=\rho(t_2) \\ \max\big(\rho(t_1),\rho(t_2)\big) \text{ si } \rho(t_1)\neq\rho(t_2) \end{cases}$.*

L'analyse en moyenne de ρ est difficile, mais on peut facilement le majorer.

Théorème 2. *Pour tout arbre binaire t, on a : $\rho(t) \leq \log_2(|t|+1)$.*

Démonstration. Posons provisoirement $\rho'(t) = 2^{\rho(t)}$. Alors $\rho'(F) = 1$ et on a la formule inductive : $\rho'\big(N(t_1,t_2)\big) = \begin{cases} 2\rho'(t_1) \text{ si } \rho'(t_1)=\rho'(t_2) \\ \max\big(\rho'(t_1),\rho'(t_2)\big) \text{ si } \rho'(t_1)\neq\rho'(t_2) \end{cases}$.

Dans tous les cas, $\rho'\big(N(t_1,t_2)\big) \leq \rho'(t_1)+\rho'(t_2)$. On en déduit par induction structurelle que $\rho'(t)$ est majoré par le nombre de feuilles de t, c'est-à-dire $|t|+1$.

En fait, on peut déterminer les cas meilleur et pire de la méthode d'Ershov. Le cas le pire est celui où, à taille donnée, l'arbre est aussi équilibré que possible : les feuilles se trouvent sur

au plus deux niveaux. Le cas le meilleur est celui d'un arbre filiforme. On voit même, en déplaçant un noeud extrême à la fois, que toutes les valeurs intermédiaires entre le minimum $\rho_{n,min}$ et le maximum $\rho_{n,max}$ peuvent être prises.

3.3. *Conclusion*

Le coût le pire $\rho_{n,max}$ par la méthode d'Ershov est, lorsque $n \to +\infty$, négligeable devant le coût moyen $\overline{\chi_n}$ par la méthode gauche-droite. Cependant, la mise en oeuvre de la méthode d'Ershov est elle-même coûteuse et ne se justifie que lors d'une compilation, où l'on s'attend à ce que l'expression soit souvent évaluée.

Exercice de programmation :

► *Il vous est demandé de rédiger un programme conforme aux spécifications ci-dessous dans l'un des langages C, Caml ou Java à votre choix. Ce programme devra être accompagné d'un exemple d'exécution permettant d'en vérifier le bon fonctionnement. La clarté et la concision du programme seront des éléments importants d'appréciation pour le jury.*

Écrire une fonction qui reçoit en argument une expression algébrique donnée sous forme arborescente et *décore* cette expression en calculant pour chaque nœud interne quelle est la valeur du paramètre ρ et quelle branche doit être évaluée en premier selon l'algorithme d'Ershov.

Suggestions pour le développement

► *Soulignons qu'il s'agit d'un menu à la carte et que vous pouvez choisir d'étudier certains points, pas tous, pas nécessairement dans l'ordre, et de façon plus ou moins fouillée. Vous pouvez aussi vous poser d'autres questions que celles indiquées plus bas. Il est très vivement souhaité que vos investigations comportent une partie traitée sur ordinateur et, si possible, des représentations graphiques de vos résultats.*

— Préciser les cas d'exception de l'algorithme d'évaluation par pile (page 3) et leurs significations. Justifier le calcul de la hauteur maximum (théorème 1).
— Indiquer quelle modification de l'algorithme d'évaluation par pile permet de traiter des opérateurs d'arités autres que 2.
— Expliquer quelle modification de l'algorithme permet de reconstituer l'expression sous forme arborescente à partir de son écriture postfixe (remarque 1 page 3).
— Démontrer le corollaire 2 page 4 et calculer $\chi(t)$ pour tous les arbres binaires de taille ≤ 3. Décrire les arbres de taille donnée qui réalisent le minimum et le maximum de χ.
— Formuler précisément l'algorithme d'allocation d'Ershov et prouver le lemme 1 page 5. Estimer le coût de la mise en oeuvre de l'algorithme et en discuter l'interêt.
— Prouver les assertions qui suivent le théorème 2 page 5, calculer les valeurs extrêmes $\rho_{n,min}$ et $\rho_{n,max}$ et donner des exemples d'expressions qui réalisent les cas examinés.