IFT2015 automne 2013 — Devoir 2

Miklós Csűrös

17 septembre 2013 — remise le 30 septembre à 20 :15

2.0 Le langage PostScript (5 points)

Un fichier PostScript (extension `.ps` ou `.eps`) est un fichier régulier de texte qui contient le code source pour un programme interprété (par l'imprimante ou par un émulateur). Le langage PostScript utilise la notation *postfixe* : les arguments précèdent les opérateurs. On calcule 20+15, par exemple, en écrivant `20 15 add`. La notation postfixe permet l'exécution par pile : le code contient des constantes (comme 20 et 15) qui sont empilés, et des opérateurs qui sont exécutés immédiatement. L'exécution d'un opérateur change le contenu de la pile ou implique des effets à bord (comme l'allocation d'objet ou le dessin). Ainsi, le fragment précédent correspond à la séquence d'opérations $\mathsf{push}(20)$; $\mathsf{push}(15)$; $\mathsf{push}\big(\mathsf{pop}() + \mathsf{pop}()\big)$ sur la pile. On définit une variable en spécifiant son nom, suivi par sa valeur, et l'opérateur `def`. (exemple : `/mavar 20 def` — la barre oblique '/' introduit le nom). On peut aussi définir de nouveaux opérateurs (ou redéfinir presque tous les mots-clés). On empile son nom, la procédure (séquence d'instructions en un bloc embracé par accolades), et exécute l'opérateur `def` (exemple : `/monop { 20 15 add } def`). On crée un tableau par ses éléments entre crochets (exemple : `[20 15]` — le crochet fermant ']' est en vérité un opérateur qui alloue la mémoire pour le tableau, y copie les éléments de la pile, et empile une référence au tableau créé). Figure 1 montre quelques aspects fondamentaux du langage par l'exemple d'un fichier complet.

Quelques opérateurs

Le tableau 1 montre les opérateurs basiques nécessaires dans ce devoir. ▶ Consultez la spécification [1] pour la description détaillée des opérateurs avec des exemples de code.

Édition et débogage

Pour éditer du PostScript, on utilise n'importe quel éditeur de texte. Bien qu'on puisse vérifier l'exécution par impression, il est plus pratique de se servir de l'émulateur `ghostscript`. L'exécutable s'appelle `gs` sur Linux. En lançant `gs` sans argument, vous entrez dans l'intérpréteur et vous pouvez taper votre code pour tester. On peut lancer `gs` aussi avec un fichier comme argument, et produire la graphique soit sur l'écran soit dans un autre format de fichier. (Souvent, les convertisseurs `ps2pdf`, `epstopdf`, etc. sont simplement des scripts appellant `gs`.) Ghostscript n'aide pas trop le débogage : les messages d'erreur sont plutôt cryptiques, mais les deux premières lignes montrent le nom d'erreur ainsi que le contenu de la pile. Afin de suivre l'exécution de votre code ou examiner l'effet des instructions dans `gs`, il est utile de placer la commande `=` (précédée par **dup** ou **copy** pour garder la pile originale) et **stack** dans le code pendant son développement (mais on les enlève de la version finale destinée à l'imprimante). Les opérateurs `=` et **stack** affichent au flux standard (visible en Ghostscript) et non pas dans l'espace graphique. Une bonne pratique est de souvent annoter les changements de la pile dans le fichier. ▶ Consultez [2] pour des explications plus détaillées du

TAB. 1 – Quelques opérateurs de base

pile avant	opérateur	pile après	description
Arithmétique			
$x_1\, x_2$	**add**	x	addition ($x = x_1 + x_2$)
$x_1\, x_2$	**sub**	x	soustraction ($x = x_1 - x_2$)
$x_1\, x_2$	**mul**	x	multiplication ($x = x_1 \cdot x_2$)
$x_1\, x_2$	**div**	x	division ($x = x_1/x_2$)
$n\, m$	**mod**	r	reste ($r = n \bmod m$)
n	**neg**	$-n$	négative
Manipulation de la pile			
x	**pop**	—	dépile le haut
$x_1\, x_2$	**exch**	$x_2\, x_1$	échange les deux éléments en haut
x	**dup**	$x\, x$	duplique l'élément en haut
$x_1\, x_2\, \ldots\, x_n\, n$	**copy**	$x_1\, x_2\, \ldots\, x_n\, x_1\, x_2\, \ldots\, x_n$	duplique n éléments en haut
$x_n\, x_{n-1}\, \ldots x_0\, n$	**index**	$x_n\, x_{n-1}\, \ldots x_0 x_n$	copiage d'un élément arbitraire
$x_{n-1}\, x_{n-2}\, \ldots\, x_0\, n\, j$	**roll**	$x_{(j-1) \bmod n}\, \ldots x_0, x_{n-1}\, \ldots x_{j \bmod n}$	décalage circulaire
Opérateurs de contrôle			
$b\, p$	**if**	—	exécute p si b est vraie
$b\, p_1\, p_2$	**ifelse**	—	exécute p_1 si b est vraie ; p_2 sinon
$n_0\, d\, n_{\max}\, p$	**for**	—	boucle exécutée avec $n_0, n_0 + d, \ldots n_{\max}$
$n\, p$	**repeat**	—	boucle exécutée n fois (sans variables d'itération)
Tableaux			
—	**[**	*marque*	début de construction de tableau
marque $x_0\, \ldots x_{n-1}$	**]**	*tableau*	finit la construction du tableau
tableau i	**get**	x_i	accès à élément avec indice i (=0 pour le premier)
tableau	**length**	n	longueur du tableau
tableau p	**forall**	—	boucle exécutée pour chaque élément du tableau
Affichage sur sortie standard			
x	**=**	—	affiche le haut de la pile sur le flux standard
—	**stack**	—	affiche le contenu de la pile sans la détruire

graphisme et de modification de code PostScript. (Vous avez 5 points automatiquement si vous lisez les références et étudiez les exemples de code.)

Références

[1] Adobe Systems Inc. *PostScript Language Reference*. Third edition. 1999. `http://www.adobe.com/products/postscript/pdfs/PLRM.pdf` *(spécification du langage ; chapitre 8 décrit tous les opérateurs en détail)*

[2] J.-M. Friedt. Introduction au langage PostScript, 2012. `http://jmfriedt.free.fr/lm_ps.pdf` *(survol de PostScript en français ; chapitres 2–3 discutent le travail avec Ghostscript et les bases du graphisme ; chapitre 5 contient 2 exemples bien annotés)*

[3] B. Casselman. *Mathematical Illustrations : A Manual for Geometry and PostScript*. Cambridge University Press, 2004. `http://www.math.ubc.ca/~cass/graphics/manual/` *(si vous voulez savoir davantage sur des astuces de produire des dessins)*

[4] G. C. Reid. *Thinking in PostScript*. Addison-Wesley, 1990. *(très bonne discussion de la pratique de programmation en PostScript)*

2.1 Qu'est-ce qu'il fait ? (15 points)

a. Manipulation de la pile. ► Quel est le contenu de la pile après l'exécution du code entre a. et b. sur Figure 1 ?

b. Opérateur mystique. ► Quel est le contenu de la pile après l'exécution de $a\ b$ **mystique** de Figure 1 ? (a et b sont deux nombres.)

c. Parcours d'un tableau. ► Que fait l'opérateur **tbl. ?** sur Figure 1 ?

d. Opérateur énigmatique. ► Que fait l'opérateur **enigme** défini ci-dessous ? (Quelle est la valeur v quand on exécute n **enigme** avec un nombre entier $n > 1$?)

```
/enigme % n enigme v
{
    0 1 3 -1 roll 1 sub {exch 1 index add} repeat exch pop
} def
```

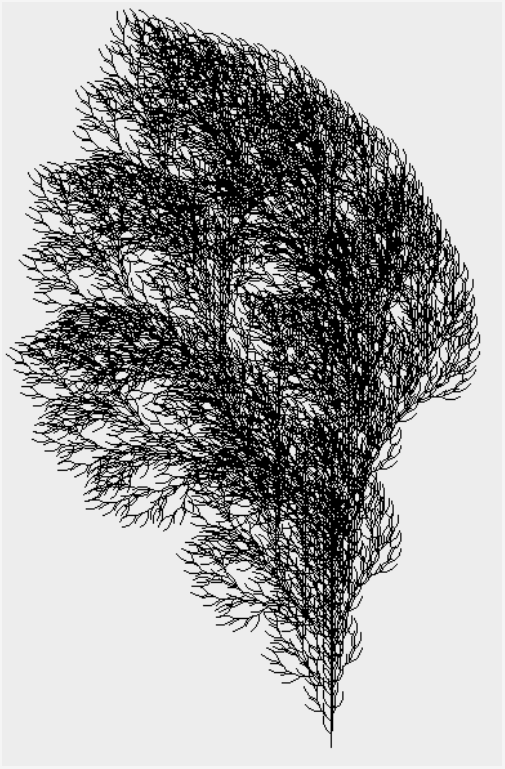
e. Maximum dans un tableau. ► Écrire un opérateur qui calcule le maximum dans un tableau de nombres non-négatifs.

```
/tbl.max % [x0 x1 ...] tbl.max m
{ ...} def
```

```
%!PS-Adobe-3.0 EPSF-3.0          `%%' dénote un commentaire spécial. Ici on montre le préambule minimal
%%BoundingBox: 0 0 612 792       selon la spécification EPSF (encapsulated PostScript) qui inclut l'indication de la
%%EndComments                    boîte englobant tout le dessin (BoundingBox) en points (=1/72 pouce)

% a. manipulation de la pile  ←`%' dénote un commentaire jusqu'à la fin de la ligne (comme // en Java).
1 2 3.25 4e10 -5 6 7 8 % valeurs empilees ←on écrit les nombres comme en Java
4 1 roll      ←rotation circulaire de quatre éléments vers le bas de la pile
pop       ← dépile l'élément en haut
exch      ← échange les deux éléments en haut de la pile
7 -4 roll     ← rotation circulaire de sept éléments vers le haut de la pile

% b. nouvel operateur          on définit un nouvel opérateur op par /op { … } def
/mystique   ← `/' sert comme caractère d'échappement; ceci évite l'évaluation immédiate
{             ← accolades enferment un bloc d'instructions; ceci aussi évite l'évaluation immédiate
    2 copy  ← duplique les deux éléments en haut (avant: x y, après: x y x y)
    gt        ← test logique de «plus grand que»; remplace les deux éléments en haut par true ou false
    {
        exch
    } if    ← énoncé conditionnel; s'écrit par { … } if. Cela consume la valeur booléenne en haut de la pile,
    pop     et exécute le code bloc si elle est vraie. Pour avoir une branche «sinon», utiliser {code pour si}
} def       {code pour sinon} ifelse

10 25 add /dfrnc exch def    ← on définit une variable var par /var valeur def. Souvent, on combine
                             avec exch quand la valeur est le résultat d'un calcul (pour lisibilité).
/sigle (IFT2015\tA13:) def   ← une chaîne de caractères s'écrit entre parenthèses (et non pas guillemets);
                             '\t' est le caractère de tabulation, échappée comme en Java.
/montableau [1 2 3 -4] def   ← on définit un tableau par crochets
   [0 2 1000 {} for] ← définition d'un tableau contenant 0,2,4,6,…,1000; for définit une boucle (init,
/autretbl exch def   incrément, valeur finale). (Logique de PostSript: il exécute le bloc {…} après avoir placé la
                     valeur de la variable d'itération sur la pile à chaque itération. Comme on ne dépile pas,
                     toutes les valeurs restent sur la pile et ']' les enlève jusqu'au crochet ouvert. )
% c. parcours d'un tableau
/tbl.? % [x0 x1 ...] tbl.? x
{                                  Par convention, on décrit l'usage d'un opérateur en spécifiant les
    0 exch                         arguments (de gauche à droit, dans l'ordre d'empiler), et le résultat
    {                              de l'opération remplaçant les arguments. Il n'y a pas de restriction
        0 lt {1 add } if  ← lt est le test logique de «plus petit que»   sur le nom d'opérateurs — on peut utiliser tout caractère sauf %.
    } forall ← boucle sur les éléments du tableau: {…} forall. En chaque itération,
} def        le prochain élément du tableau est automatiquement empilé et le code bloc
             est exécuté; le tableau même est dépilé avant la première itération.

montableau tbl.?   ← exécuter tbl.? avec montableau
20 string cvs   ← conversion en string: le syntaxe est x s cvs s' , où x est la valeur à convertir, s est un string
                de longueur suffisante, allouée par 20 string. Le résultat s' remplace le contenu de s
/Times-Roman 12 selectfont ← sélection de police de 12 pt. Times est une des polices prédéfinies.
10 20 moveto ← déplacement aux coordonnées (10, 20). Axes X et Y vont de gauche à droit et de bas vers haut
sigle show show    ← dessin de 2 strings à partir de la position courante (10,20).
5 3.5 rmoveto 612 792 lineto stroke ← déplacement+dessin de ligne; stroke raie le chemin construit
%%EOF  ←commentaire spécial exigé par la spécification EPSF à la fin du fichier.
```

FIG. 1 – Fichier `.eps` annoté illustrant quelques aspects fondamentaux du langage PostScript.

2.2 Comment dessiner un arbre (30+5 points)



Système de Lindenmayer

Vous avez à implanter un programme qui dessine des graphiques aléatoires à l'aide d'un *système de Lindenmayer* ou système L. En particulier, on veut produire des dessins qui ressemblent à des plantes. Le système L était inventé pour ce but : il permet de modéliser le développement de structures végétales.

Un système L est une grammaire formelle qui définit la génération de chaînes de caractères sur un alphabet. Dans ce travail, on utilise l'alphabet `Ff+-[]X`. Le système est spécifié par la chaîne de départ ω et un ensemble de règles de réécriture dans la forme «caractère → chaîne». Exemple :

$$\omega = \texttt{F}$$
$$\texttt{F} \to \texttt{FF-F}$$

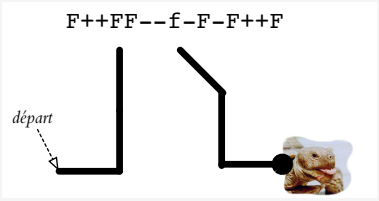
Dans un tel système, on génère des chaînes $S_0, S_1, S_2, \ldots$ en appliquant les règles de remplacement à tous les caractères de S_i en parallèle pour arriver à S_{i+1}. On commence par $S_0 = \omega$: $\texttt{F} \Rightarrow \texttt{FF-F} \Rightarrow \texttt{FF-FFF-F-FF-F} \Rightarrow \cdots$ S'il existe plusieurs règles avec le même côté gauche, on choisit une des règles applicables au hasard (avec probabilité uniforme).

$$\omega = \texttt{F}$$
$$\texttt{F} \to \texttt{F-F}$$
$$\texttt{F} \to \texttt{+F}$$

$$\texttt{F} \underbrace{\Rightarrow}_{\text{avec proba } 1/2} \texttt{F-F} \underbrace{\Rightarrow}_{\text{avec proba } 1/4} \texttt{+F-F-F} \Rightarrow \cdots$$

Graphisme tortue

On interprète la chaîne finale obtenue par l'application des règles comme des instructions de dessin pour une tortue graphique.

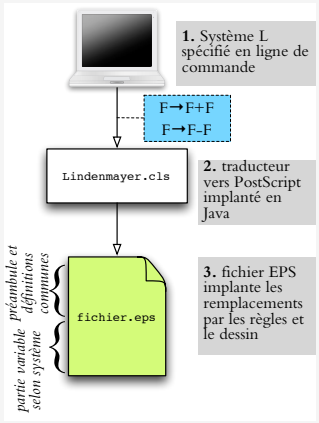


La tortue possède un crayon et peut bouger en avant (par une distance fixe D) en traçant une ligne (symbole `F`) ou non (symbole `f`). La tortue peut aussi tourner (par une angle fixe δ) dans le sens d'aiguille (symbole `-`) ou contre (symbole `+`). L'état de la tortue comprend sa position (x, y) ainsi que l'angle θ de son nez par rapport à la ligne horizontale. Un dessin est spécifié par une chaîne de caractères du système L, en interprétant les caractères un-à-un (l'exemple à la gauche utilise $\delta = 45^\circ$ pour tourner)

`F`	Avance la tortue par D, en dessinant une ligne entre la position de départ et celle de l'arrivée. L'état de la tortue change de (x, y, θ) à $(x + D\cos\theta, y + D\sin\theta, \theta)$ où D est un paramètre global du dessin spécifiant l'échelle.
`f`	Avance la tortue par D, mais sans dessin. L'état de la tortue change de (x, y, θ) à $(x + D\cos\theta, y + D\sin\theta, \theta)$.
`+`	Tourne la tête de la tortue. L'état de la tortue change de (x, y, θ) à $(x, y, \theta + \delta)$ où δ est un paramètre global du dessin.
`-`	Tourne la tête de la tortue. L'état de la tortue change de (x, y, θ) à $(x, y, \theta - \delta)$ où δ est un paramètre global du dessin.
`[`	Empile l'état courant de la tortue sur la pile d'états sauvegardés. L'état de la tortue ne change pas.
`]`	Dépile l'état de la tortue et fait l'affectation de l'état courant. L'état de la tortue donc change de (x, y, θ) à (x', y', θ') où (x', y', θ') est l'état le plus récemment sauvegardé par `[`.
`X`	Aucun effet sur la tortue, ignoré dans le dessin.

D'où vient la connexion entre les grammaires et les plantes ? Le principe est que les règles décrivent le développement de la plante : la règle `F` → `F[+F]F`, par exemple, capture la poussée avec une branche à côté. Des alphabets et de règles plus sophistiqués peuvent modéliser la formation d'organes différentes [P. Prusinkiewicz et A. Lindenmayer. *The Algorithmic Beauty of Plants*, Springer-Verlag, 2004].

Implantation



Vous devez écrire un programme Java (le «traducteur») qui prend un système L, et produit un fichier eps (Encapsulated PostScript) qui implante le système. En particulier, le fichier eps est un programme qui exécute et dessine le résultat d'expansion de règles du système. Un exemple (`lindenmayer.eps`) vous est fourni sur le site du cours. Entre autres, l'exemple donne le code entier du graphisme tortue et du choix de règles au hasard. Le trauducteur peut simplement copier la première partie de l'exemple (jusqu'à la ligne avec `BEGIN`), et écrire la définition des opérateurs qui implantent les remplacements par récurrence. La sortie finit par la ligne `%%EOF`.

Logique du code généré. Le résultat PostScript exploite la récurrence, en traduisant les règles différentes en opérateurs qui prennent un seul argument. Ce seul argument spécifie la profondeur d'expansion : si on arrive à 0, on exécute l'opération appropriée avec la tortue, sinon on soustrait 1 de la profondeur et applique une règle. Le code suivant esquisse cette idée pour la règle `F` → `F-F`.

```
/F % iter F -
{
    dup 0 eq % si iter==0
    { % cas de base...
     10 T:draw % dessin avec la tortue
    }{ % cas recursif: expansion de regle
        1 sub % decrementer le compteur d'expansion
        dup F % dupliquer le compteur pour le deuxieme appel F
        -22.5 T:turn % virage de la tortue
        F
    } ifelse
} def
```

Spécification Le traducteur est lancé comme

`java Lindenmayer` ⟨arguments optionnels⟩ n ω **règle**$_1$ **règle**$_2$ ...

Exemple :

```
% java -cp build/classes Lindenmayer -D 2 -x 250 -y 0 -delta 22.5 -a 90 6 F \
   'F:F[+F]F[-F]F' 'F:F[+F]FF' 'F:F[-F]F' > lindenmayer.eps
```

Arguments obligatoires.

n entier non-négatif, c'est le nombre d'itérations dans la dérivation ou le maximum de la profondeur de récurrences

ω chaîne de départ dans le système

règle$_i$ est une chaîne sans espace où le côté gauche et droit sont séparés par '`:`' (dénotant donc le symbole $\rightarrow$)

Arguments optionnels.

`-D double` spécifie l'échelle du dessin (=2 par défaut)
`-delta double` spécifie l'angle d'unité pour la tournée de la tortue en degrés (=22.5 par défaut)
`-x double` coordonnée X de la position initiale de la tortue (=250 par défaut)
`-y double` coordonnée Y de la position initiale de la tortue (=0 par défaut)
`-a double` angle initial de la tortue en degrés (=90 par défaut)

Exemples. Quelques exemples inspirants :

Nom	Paramètres	ω	Règles
Flocon	$n = 4, \delta = 90^\circ$	`-F`	`F` $\rightarrow$ `F+F-F-F+F`
Ilots	$n = 2, \delta = 90^\circ$	`F+F+F+F`	`F` $\rightarrow$ `F+f-FF+F+FF+Ff+FF-f+FF-F-FF-Ff-FFF`
			`f` $\rightarrow$ `ffffff`
Plante	$n = 5, \delta = 25.7^\circ$	`F`	`F` $\rightarrow$ `F[+F]F[-F]F`
Buisson	$n = 5, \delta = 22.5^\circ$	`F`	`F` $\rightarrow$ `FF-[-F+F+F]+[+F-F-F]`
Plante	$n = 5, \delta = 22.5^\circ$	`F`	`F` $\rightarrow$ `F[+F]F[-F]F`
			`F` $\rightarrow$ `F[+F]F`
			`F` $\rightarrow$ `F[-F]F`

Vous pouvez avoir jusqu'à 5 points de boni pour un système L original et créatif. (Exemple : étudiez les embranchements sur la photo d'un arbre.)

Remise

Avant 20 :15 le 30 septembre, soumettez votre travail en un email à `csuros@iro.umontreal...` :

- ⋆ réponses aux questions de 2.1 avec justifications dans le texte du courriel
- ⋆ code Java traducteur en pièce joint
- ⋆ exemple d'application du traducteur : arguments dans le texte du courriel, EPS résultant en pièce joint