M1 — Option P(F)A — 2 mini-projets

Avril 2015

Les deux mini-projets qui vous sont proposés pour la fin de l'option P(F)A et qui serviront à vous mettre des notes à caractère « pratique » consistent en des implantations — en Scheme et en Ruby — de deux simulations d'un distributeur de boissons chaudes. Nous donnons d'abord les consignes générales, puis fournissons des directions pour chaque réalisation.

1 Distributeur de boissons chaudes

Vous avez à réaliser la simulation d'un distributeur de boissons chaudes proposant du café noir, du café au lait, du chocolat et du thé. Ce distributeur accepte des pièces d'un euro, de 10, 20 et 50 cents, le prix uniforme de chaque boisson étant de 50 cents [1]. À la mise en service, le distributeur possède de quoi servir 30 gobelets de chaque boisson, et son monnayeur comprend 10 pièces de 10 cents, 10 pièces de 20 cents et 10 pièces de 50 cents. Nous ne nous préoccuperons pas du réapprovisionnement du distributeur, que ce soit en boissons ou en pièces de monnaie.

Le distributeur doit réagir à la sollicitation d'un utilisateur, qui précise le type désiré pour la boisson et, pour chaque type de pièces, le nombre introduit, les différents types de pièces étant spécifiés par montant décroissant. Par exemple, l'ordre caractérisé par les informations suivantes :

```
black-coffee   0   0   1   3
```

dit qu'un utilisateur désire du café noir et a présenté zéro pièce d'un euro, zéro pièce de 50 cents, une pièce de 20 cents et 3 pièces de 10 cents.

Le distributeur doit vérifier que la somme introduite n'est pas inférieure au prix d'une boisson et qu'il peut rendre la monnaie [2], puis s'assurer que la boisson est disponible [3]. Si tout se passe bien, la boisson est servie, le prix de cette boisson empoché, et la monnaie rendue à l'utilisateur. Sinon, la monnaie est intégralement rendue à l'utilisateur.

Dans le cas où la boisson demandée par un utilisateur n'est pas disponible — et donc que sa monnaie lui a été rendue —, l'utilisateur peut indiquer un nouveau choix comme suit :

```
white-coffee     as-before
```

qui signifie que l'utilisateur a inséré le même jeu de pièces que pour l'ordre précédent. Cette fonctionnalité — l'usage du mot-clé `as-before` — n'est disponible que si la transaction précédente a été invalidée pour cause de boisson manquante.

1. Nous sommes moins cher que le distributeur du bâtiment de Métrologie. Ce n'est pas beau, la concurrence ?
2. . . . ce qui est bien évidemment trivial si l'utilisateur a présenté la somme exacte.
3. Si l'utilisateur a indiqué une boisson qui n'existe pas, tout se passe comme si cette boisson n'était pas disponible.

2 Réalisation en Scheme

Le distributeur sera représenté par une fonction à deux arguments dans votre réalisation en Scheme. Le premier de ces deux arguments est un symbole pour la boisson choisie, le second est soit une liste linéaire de quatre entiers naturels figurant un paiement, soit le symbole `as-before`, comme nous l'avons expliqué au § 1. Le résultat est soit la valeur `#f` — ce qui signifie qu'il est impossible de rendre le service et que la monnaie a été intégralement restituée à l'utilisateur —, soit une liste linéaire de quatre entiers naturels représentant le rendu de monnaie [4] — auquel cas le service a été rendu. Cette fonction modélisant le distributeur utilise un environnement en clôture lexicale, mis à jour au gré des appels successifs de cette fonction.

Style de programmation

Vous devez réaliser ce projet dans un style fonctionnel. Les effets de bord — utilisation de formes telles que `set!`, `set-car!`, `set-cdr!` — ne seront permis que sur les définitions locales. Ce qui signifie — si l'on considère l'exemple suivant :

```
(define x
  (let ((y ...))
    ...
    (set! y ...)
    ...))
```

— que l'effet de bord sur `y` (variable locale à la définition de `x`) est permis, mais *pas* l'effet de bord « `(set! x ...)` », qui touche une variable accessible par l'utilisateur.

3 Réalisation en Ruby

Dans cette seconde version en Ruby, vous réaliserez le distributeur sous forme d'un *thread*. La commande d'une boisson s'effectuera par une fonction séparée, qui « réveille » le *thread*, celui-ci « s'endormant » à nouveau après avoir traité l'ordre. Ne pas oublier qu'un *thread* de Ruby peut communiquer avec son environnement par le biais de variables, aussi bien en lecture qu'en écriture. Nous emploierons les deux symboles suivants pour le passage d'une commande au distributeur et la récupération de sa réponse :

```
:order      :answer
```

Les conventions sont :

— pour la variable accessible par le symbole `:order`, c'est un tableau de 2 ou 5 éléments : le premier élément est le nom de la boisson — donné par une chaîne de caractères ou un symbole [5], à votre libre choix — ; les éléments suivants sont soit l'information `as-before` [6], soit quatre entiers naturels correspondant aux nombres de pièces introduites, donnés par montant décroissant ;

4. Suivant la même convention que pour le montant introduit par l'utilisateur, c'est-à-dire que les nombres de pièces sont donnés par montant décroissant.

5. Dans ce second cas — l'utilisation d'éléments de classe `Symbol` —, ne pas perdre de vue que le caractère « - » ne peut pas être utilisé dans un symbole, procéder comme dans des langages de programmation comme C ou en Java, c'est-à-dire, substituer ce caractère par le caractère « _ » de soulignement.

6. Là aussi, vous pouvez la donner sous la forme d'une chaîne de caractères ou d'un symbole (mais voir la note 5 précédente).

— pour la variable accessible par le symbole `:answer`, c'est soit la valeur `nil` — ce qui signifie qu'il est impossible de rendre le service et que la monnaie a été intégralement restituée à l'utilisateur —, soit un tableau de quatre entiers naturels représentant le rendu de monnaie [7].

4 Modalités

Ce projet est à rendre au plus tard le **mardi 19 mai 2015, à 8 H 00**, sous forme de fichiers sources Scheme et Ruby à m'envoyer par *mail* (`jmhuffle@femto-st.fr`). Vous pouvez bien sûr y joindre quelques jeux d'essais. Les projets pourront être réalisés par groupes de 2 ou 3 étudiants. Il va de soi que sans être « sacqués », les groupes de 3 seront jugés plus sévèrement que les groupes de 2.

Des questions ?

Pourquoi pas ? `jmhuffle@femto-st.fr` est à l'écoute... Mais il n'y sera pas répondu personnellement. Toutes les questions et réponses seront répertoriées ci-après si besoin est. Aussi, n'oubliez pas de consulter régulièrement d'éventuelles nouvelles versions de ce fichier à l'adresse *Web* suivante :

`http://lifc.univ-fcomte.fr/home/~jmhufflen/PFA/2015/projects.pdf`

(Pas de questions pour l'instant.)

HAVE FUN!

7. Par montant décroissant, bien entendu.