Programmation IV	Série 13	8 juin 2005

Exercice 1

Introduction

Vous trouverez sur la page Web du cours une version de l'interpréteur PROLOG vu au cours, et que nous allons utiliser pour cette série. Nous nous contenterons ici de l'utiliser, sans le modifier d'aucune manière.

Notez encore que l'interpréteur PROLOG vous permet de charger le contenu d'un fichier au moyen de la commande `:l` semblable à celle de `scalaint`. Pour quitter, utilisez `:q`.

Exercice

Le but de cet exercice est de réaliser un programme PROLOG permettant de jouer avec les « cadavres exquis ». Les cadavres exquis sont des phrases obtenues en combinant des mots aléatoires en s'assurant que la phrase produite soit grammaticalement correcte.[1] En fait, ce programme produira *toutes* les phrases possibles à partir d'un vocabulaire donné, mais cela n'est pas important.

La réalisation d'un tel programme est fort simple : il suffit de décrire la structure grammaticale d'une phrase, et de donner des règles pour en produire les différentes composantes.

Pour commencer avec un cas très simple, admettons qu'une phrase est composée d'un simple sujet suivi d'un verbe intransitif. Le programme PROLOG suivant permet de produire toutes les phrases de cette forme avec les sujets *Jean* et *Marie*, et les verbes *mange* et *dort*.

```
sujet(jean).
sujet(marie).
verbe(mange).
verbe(dort).
phrase([S,V]) :- sujet(S), verbe(V).
```

Notez que ce programme permet aussi bien de produire toutes les phrases de cette forme, avec le vocabulaire donné, que de vérifier si un sujet et un verbe donnés forment bien une phrase.

[1] Le nom *cadavre exquis* vient semble-t-il du premier poème ainsi réalisé : « Le cadavre exquis boira du vin nouveau »

Partie 1 Sur la base de cette idée, commencez par définir une structure plus complète pour les phrases. On peut par exemple imaginer qu'une phrase est composée d'un sujet, d'un verbe transitif et d'un complément. Le complément est à son tour composé d'un article, d'un adjectif et d'un nom. N'oubliez pas de définir un certain nombre d'articles, d'adjectifs et de noms.

Partie 2 Un problème que vous constaterez immédiatement est celui du genre : si *le* et *la* sont des articles, *belle* et *grand* des adjectifs et *table* et *vélo* des noms, le programme accepte *la belle vélo* comme complément, ce qui est grammaticalement faux. Modifiez donc votre programme afin qu'il n'admette comme compléments que ceux composés d'un article, d'un adjectif et d'un nom de même genre.

Partie 3 Finalement, pour corser encore un peu le tout, faites en sorte que votre complément puisse comporter zéro, un ou deux adjectifs avant le nom. Bien entendu, tous ces adjectifs devront avoir le même genre que le nom qu'ils qualifient. Pour ce faire, utilisez les listes vues au cours, définies au moyen des constructeurs `nil` et `cons`.

Exercice 2 (optionnel)

On désire modifier l'interpréteur LISP vu la semaine dernière pour avoir un environnement global qui puisse être étendu par des définitions de l'utilisateur.

Avec la version actuelle de l'interpréteur, l'environnement global ne contient que les définitions primitives du langage LISP, comme les fonctions `cons`, `car`, etc. Il n'est pas possible d'ajouter des définitions à cet environnement.

Pour remédier à ce problème, on décide d'introduire une nouvelle forme de définition (`def`) qui modifie l'environnement global. Cette nouvelle forme se distingue de la précédente par l'absence de la dernière expression : la nouvelle forme n'est composé que d'un nom et d'une valeur.

Cette nouvelle forme permet par exemple de définir la fonction factorielle au moyen d'une commande, puis de l'utiliser plus tard, comme l'exemple suivant l'illustre :

```
lisp> (def fact (lambda (x) (if (= x 0) 1 (* x (fact (- x 1))))))
()
lisp> (fact 5)
120
```

À noter que cette nouvelle forme de définition n'est autorisée qu'au niveau le plus externe, c'est-à-dire qu'elle ne peut être intégrée à une expression plus complexe.

Pour procéder à cet ajout, nous vous recommandons d'effectuer les phases ci-dessous l'une après l'autre, dans l'ordre donné.

Phase 1 Modifiez la définition de la valeur `globalEnv` qui contient l'environnement global pour en faire une variable. Cela permet au nouveau type de définitions de modifier l'environnement global pour l'étendre.

Phase 2 Modifiez ensuite la fonction d'évaluation `eval1` afin qu'elle reconnaisse et traite correctement la nouvelle forme de définition. Encore une fois, ces nouvelles formes de définitions ne sont composées que d'un nom et d'une valeur, et ont pour effet de lier la valeur au nom dans l'environnement global.

Pour vous simplifier la vie, vous pouvez admettre que ce nouveau type de définition retourne toujours la liste vide.

Et encore... Pour parachever l'œuvre, modifiez maintenant la fonction `normalize` pour que soient acceptées les définitions du type :

```
lisp> (def (fact x) (if (= x 0) 1 (* x (fact (- x 1)))))
```