Option PF2, L2-Info&Math, Automne 2016

TP 11

« Programmation Scheme Avancée »

µScheme v1

Rappel : il s'agit de développer pas à pas l'évaluateur de Peter Norvig, dont on pourra lire la page Web APRES avoir rédigé soi-même la solution, si l'on veut que la chose présente de l'intérêt...

Exercice 11.1 Vérifiez le bon fonctionnement du *parser*, la fonction `read_from`, dans l'analyse de la chaîne `'(+ 1 2)'`.

Exercice 11.2 Il y a cependant un petit bug qui traîne dans la fonction `read_from`, personne n'est parfait. Vérifiez sur l'exemple de l'analyse de la chaîne `'(+ 1 2'` qui devrait lever l'exception `SyntaxError` avec le message d'erreur *Unexpected EOF while reading* et qui au lieu de cela affiche un autre message d'erreur moins agréable sur un débordement d'indice !

Exercice 11.3 Complétez le texte de la méthode `find` dans la classe `Env`. Elle doit retourner le premier dictionnaire de l'environnement `self` (une suite chaînée de dictionnaires) contenant la variable `var`.

Exercice 11.4 **a)** Rajoutez une nouvelle primitive à µScheme dans l'environnement global, la fonction `(sqr x)`.
b) Et si l'on voulait rajouter toutes les primitives Python du module `math` ? On pourrait le faire à la main par exemple en rajoutant par exemple dans le dictionnaire le couple `'sqrt':math.sqrt` mais ce serait trop long. Heureusement Python possède une intéressante primitive `vars`. Contemplez le résultat de `vars(math)` et avec `update` rajoutez le contenu de `math` à µScheme !

Exercice 11.5 Complétez le cas 1 de l'évaluation d'un symbole. Il s'agit de retourner la valeur du symbole dans l'environnement.

Exercice 11.6 Complétez le cas 2 lorsque l'expression est une constante (ce n'est pas une liste et ce n'est pas non plus un symbole). Il s'agit de retourner la valeur du symbole dans l'environnement. *Notez au passage que les booléens ont une représentation externe `True` ou `False` mais ne sont pas des objets de 1ère classe. On ne peut pas écrire `(define x True)`.*

Exercice 11.7 Complétez le cas 3 lorsque l'expression est de la forme `(if p q r)`. Vous êtes censé connaître le fonctionnement du `if`.

Exercice 11.8 a) Complétez le cas 4 lorsque l'expression est une affectation, de la forme `(set! var e)`. Vous êtes censé connaître le fonctionnement d'une affectation (voir le cours).
b) Complétez le cas 5 lorsque l'expression est une définition, de la forme `(define` e_1 e_2`)`.

Exercice 11.9 Complétez le cas 6 lorsque l'expression est une séquence, de la forme `(begin` e_1 e_2 `...)`. On évalue dans l'ordre les expressions e_1, e_2,... et l'on retourne le résultat de la dernière expression e_n calculée.

• On vous a fait cadeau du code de l'évaluation d'une lambda-abstraction (c'est une terminologie issue du λ-calcul). Ce n'est pas une raison pour l'admettre sans bien le comprendre ! Idem pour l'application.

Exercice 11.10 Ajoutez à l'environnement global les fonctions de base sur les listes. Uniquement sur les vraies listes, comme en Python : on n'acceptera pas `(cons 1 2)` comme en full Scheme par exemple. Le second argument de `cons` sera une liste !

```
(cons x L), (car L), (cdr L), (null? L), (equal? L1 L2),
(list? x), (length L), (symbol? x), (append L1 L2)
```

• Testez bien entendu à fond les fonctions sur les listes. Les résultats seront affichés à la Python, ce n'est pas grave car :

Exercice 11.11 **a)** Complétez la définition de la fonction `to_string` dans le scribe de manière à afficher les listes à la Scheme.
b) Ajoutez la *commande* `.quit` au toplevel. Le symbole `.quit` n'est pas évalué, il déclenche immédiatement une réaction du toplevel, qui décide de quitter la boucle REP.

• Si vous avez réussi à faire tout ça en 1h30, je vous tire mon chapeau !