TP 4

Le TP est à remettre sur Moodle avant 22:00, le dimanche le 12 novembre.
Le TP se divise en trois parties: la première est qui vise à mieux comprendre l'évaluation par environnement et le méta-évaluateur applicatif ; la seconde vise à améliorer votre maîtrise de l'appariement de forme ; et la troisième porte sur la création de nouvelle formes syntaxiques.

Partie 1 (9 points)

1. 3 méta-évaluateur applicatif.rkt, le gabarit de ce numéro contient tout le code de «3 méta-évaluateur applicatif.rkt» vu en classe. Vous devez le modifier pour implémenter letrec. Voici une description du comportement du letrec
   (letrec ([id val-expr] ...) body ...+)
   a. un nouvel environnement est attaché à l'environnement courant ;
   b. tous les id sont ajoutés à ce nouvel environnement et leurs valeurs est undefined ;
   c. dans l'ordre, chaque val-expr est évalué dans ce nouvel environnement et immédiatement lié à id correspondant ;
   d. la séquence body...+ est évalué et la dernière évaluation est retournée.

   Le fichier trace que votre implémentation crée doit être identique (ou presque) pour le programme «progr-letrec.txt» au fichier «trace progr-letrec.txt» et pour le programme «progr-letrec.1.txt» au fichier «trace progr-letrec.1.txt» .

   Remarque 1 : Vous implémentation doit se faire uniquement en modifiant le fichier «3 méta-évaluateur applicatif.rkt». Il est cependant permis d'ajuster les identificateurs path-progr et path-trace en fonction de votre installation.

   Remarque 2 : Pour ce numéro vous ne remettez que le gabarit que vous aurez modifier pour intégrer votre implémentation du letrec.

   Voici cependant les liens vers les trois autres fichiers :
   1 initialisation.rkt

   2 structures de données - constructeur accesseur mutateur.rkt

   4 REPL.rkt

Partie 2 (24 points)
(ce numéro sera donné demain)

Partie 3 (15 points)

Vous devez coder les formes syntaxiques my-apply, one-before et ~~. Ces trois formes syntaxiques doivent être coder dans les gabarits correspondants : my-apply.rkt, one-before.rkt, quasi-quote.rkt.

2. my-apply est une forme syntaxique qui simule le apply de Racket (http://docs.racket-lang.org/reference/procedures.html) pour les cas où le dernier argument est de la forme '(...)

   Exemple
   (my-apply + 1 2 '(3 4 5)) ⇒ 15

(my-apply * '(3 4 5)) ⇒ 60
(my-apply + 1 (+ 1 5) '(4 5)) ⇒ 16
Mais vous n'avez pas à traiter des situations similaires aux suivantes :
(my-apply + 1 2 3 (map sqr '(3 4)))
(my-apply * 1 (+ 1 5) (list 4 5))

3. one-before est une forme syntaxique qui ressemble à begin sauf qu'elle retourne l'avant-dernière expression. Si la séquence d'expression est vide, one-before retourne le message d'erreur suivant "one-before vide". Si la séquence d'expression ne contient qu'un élément alors cet élément est premièrement affiché et ensuite retourné.

   Exemple :

   ```
   (one-before
     (displayln "ligne 1")
     (displayln "la prochaine ligne \"(sqrt 12345)\" est l'expression à retourner")
     (sqrt 12345)
     (displayln "la dernière ligne"))
   ⇒
   ligne 1
   la prochaine ligne "(sqrt 12345)" est l'expression à retourner
   la dernière ligne
   111.1080555135405
   (one-before) Þ "one-before vide"
   '(one-before
     ((λ () (displayln "Cette ligne doit être affichée qu'une fois.")
           (sqrt 12345))))
   ⇒
   ((λ () (displayln Cette ligne doit être affichée qu'une fois.) (sqrt 12345)))
   Cette ligne doit être affichée qu'une fois.
   111.1080555135405
   (one-before (begin (+ 1 2 3 4) (* 1 2 3 4)))
   ⇒
   (begin (+ 1 2 3 4) (* 1 2 3 4))
   24
   ```

4. ~~ est une forme syntaxique identique (ou presque) quasiquote (https://docs.racket-lang.org/reference/quasiquote.html?q=quasiquote) qui permet facilite la création de liste à l'aide de deux littéraux ~ et ~@.
   - ~~ seul a le même comportement que quote (').
     Par exemple : (~~ a b (+ 1 2)) ⇒ '(a b (+ 1 2))
   - Si un élément de la liste est précédé du littéral ~ alors il est évalué avant d'être mis dans la liste.
     Par exemple :
     (define b 2)
     (define c 3)

(~~ ~ b ~ c (a b c)) ⇒ '(2 3 (a b c))
(~~ x b ~ b y ~ (+ b 5)) ⇒ '(x b 2 y 7)

- Si un élément de la liste est précédé du littéral ~@ alors il est évalué et le résultat doit être une liste dont les éléments sont insérés dans la liste qui sera retournée par ~~.
  Par exemple
  (define ls (list '+ '* '&))
  (~~ ~@ ls) ⇒ '(+ * &)
  '(~~ 3 ~@ ls ~@ (range 5) 12) ⇒ '(3 + * & 0 1 2 3 4 12)