Récursivité et Interprétation des langages

Université Montpellier-II - UFR - ULIN201 - Langages Applicatifs -
Chapitre 10 :
C.Dony

17 Récursivité et Interprétation des langages

Réalisons un langage de programmation défini par sa fonction d'interprétation de ses expressions. Dotons le minimalement pour débuter.

– Des types de base et leurs fonctions primitives
  - nombres
  - booléens
  Des identificateurs
  Une conditionnelle, nommée "si", ayant la même sémantique que la traditionnelle conditionnelle.
– Une syntaxe. Pour simplifier, prenons celle de Scheme. Nous disposons d'un analyseur syntaxique tout prêt, implanté par la fonction "read".
– Une sémantique des construction. Prenons celles de Scheme et mettons la en oeuvre via une fonction d'interprétation : `eval302`. Rendons la utilisable via une boucle "toplevel" implantée par la fonction `scheme302`.

```
(define (scheme302)

  ;; seul type de base : int, boolean
  ;; sans fonctions utilisateur
  ;; sans gestion de la pile
  ;; avec identificateurs définis par let
  ;; avec gestion des erreurs

  (let ((**global-env (list (list '+ +) (list '- -) (list '* *) (list '/ /) (list '= =)))
        (**primitives '(+ - * / =))
        (**returnToToplevel #f)
        (**fin? #f))

    (define (primitive? f) (memq f **primitives))

    (define (eval302 e env)
      (cond ((or (number? e) (boolean? e) (string? e) (procedure? e)) e)
            ((symbol? e) (env-value e env))
            ((list? e)
             (cond ((eq? (car e) 'fin) (set! **fin? #t) 'Au-revoir)
                   ((eq? (car e) 'si) (eval-fi e env))
                   ((eq? (car e) 'let) (eval-let e env))
                   ((primitive? (car e))
                    ;; c'est une fonction primitive de notre interprète, on repasse la main
                    ;; à notre système hôte pour appliquer la fonction, en prenant soin néanmoins
                    ;; d'évaluer préalablement les arguments
                    (apply (eval302 (car e) env) (evlis (cdr e) env)))
                   (#t (error302 "fonction inconnue" (car e)))))
            (#t (error302 "construction_inconnue" e))))

    (define (eval-cite e env) (cadr e))

    (define (eval-fi e env)
      (if (eval302 (cadr e) env)
          (eval302 (caddr e) env)
          (eval302 (cadddr e) env)))
```

```
    (define (eval-let e env)
      (let ((newenv (append (parallel-eval-bindings (cadr e) env) env)))
        (eval-sequence (cddr e) newenv)))

    (define (parallel-eval-bindings bindings env)
      ;; rend un environnement augmenté des nouvelles liaisons des variables au
      ;; résultat de l'évaluation des expressions correspondantes, dans l'environnement env
      (if (null? bindings)
          ()
          (append (list (list (caar bindings) (eval302 (cadar bindings) env)))
                  (parallel-eval-bindings (cdr bindings) env))))

    (define (env-value symbol env)
      ;; rend la valeur d'une variable si elle se trouve dans l'environnement courant env
      (let ((liaison (assq symbol env)))
        (if liaison
            (cadr liaison)
            (error302 "variable_indéfinie" symbol))))

    (define (eval-sequence lexp env)
      ;; evalue les expressions en séquence et rend
      ;; la valeur de la dernière
      (letrec ((boucle (lambda (lexp value)
                         (if (null? lexp)
                             value
                             (boucle (cdr lexp) (eval302 (car lexp) env))))))
              (boucle lexp ())))

    (define (evlis l env)
      ;; reçoit une liste d'expressions et rend la liste de leurs valeurs
      (map (lambda (e) (eval302 e env)) l))

    (define (error302 s l)
      ; gestion primitive des exceptions
      ; display + retour au toplevel
      (display "Erreur: ") (display s) (display ", ") (display l) (display ".\n")
      (**returnToToplevel **returnToToplevel))

    (define (print302 e)
      (display "= ") (display e) (display "\n"))

    (do ()
      (**fin? 'A_bientôt) ;; pour quitter la boucle, écrire "(fin)"
      (display **returnToToplevel)
      (or **returnToToplevel
          (set! **returnToToplevel
                (call-with-current-continuation (lambda (returnHere) returnHere))))

      (print302 (eval302 (read) **global-env)))))

(scheme302)
```