UPMC

Master informatique 2 – STL
NI503 – CONCEPTION DE LANGAGES
Notes III

Basile STARYNKEVITCH, http://starynkevitch.net/Basile/ *
travaillant au CEA, LIST sur **gcc-melt.org**
reprenant le cours de
Pascal MANOURY, http://www.pps.univ-paris-diderot.fr/~eleph/

2013-2014

courriel : **basile@starynkevitch.net** *et* **basile.starynkevitch@cea.fr**
forum : conception-langage-upmc2013@googlegroups.com

Langages à connaître.

Il faut avoir vu les langages suivants [1] :

— **Ocaml**, qu'on peut même essayer dans son navigateur ; il faut être capable par exemple de coder (sans s'appuyer sur des fonctions usuelles du module `List`) la fonction `applimpair` à deux arguments :et une fonction f et une liste $l = [x_0; x_1; \ldots; x_n]$ et qui renvoie la liste des applications de f sur les éléments de rang impair, donct $[f(x_1); f(x_3); \ldots]$ en s'appuyant sur une fonction récursive terminale. Ainsi `applimpair (fun x -> x*3) [3;5;6;10;-1]` renvoie `[15;30]`. Voir aussi Haskell à essayer dans votre navigateur

— Lua 5.2 (ou mieux). Comprenez bien le mécanisme des méta-tables et méta-méthodes. Regardez aussi l'implémentation.

— Prolog, par exemple par son implémentation Eclipse-CLP [2] ; voir également Icon

— regardez aussi pour l'amusement Brainfuck et d'autres langages de programmation ésotériques

— Neko est utile à connaître, en particulier sa machine virtuelle NekoVM.

— La machine virtuelle Parrot est intéressante, notamment pour les languages dynamiquement typés.

— Le langage à pile PostScript est compris par beaucoup d'imprimantes. Il a influencé le format PDF (et autrefois le système de fenêtrage NeWS). Voir aussi Forth

*Toutes les notes de cours sont disponible sur mon site web.

1. En plus des autres langages traités dans ce cours !
2. Malgré la similitude du nom, ça n'a rien à voir avec l'éditeur Eclipse souvent utilisé en Java.

— Tout le monde connaît Java. Mais regarder aussi la JVM (la machine virtuelle Java), son chargeur de classe (cf la classe `java.class.ClassLoader` etc...). Le module d'enseignement Aladyn (par J.MALENFANT) (et ses cours) décrit aussi les aspects réflexifs de Java. ...). Voir aussi l'introspection de la pile d'appel, et le passionnant livre *Artificial Beings (the Conscience of a Conscious Machine)* de J.PITRAT qui explique pourquoi c'est si important. En *C*, on a des bibliothèques comme `libbacktrace`
— Les langages Python et Ruby (à essayer) sont des langages de script intéressants.

1 Scheme

Scheme (comme Common Lisp) est homoïconique et dynamiquement typé. Les AST du langage sont des listes : **(** *operateur opérande ...* **)** . En Scheme comme en Ocaml, tout est expression (mais certaines ont un effet de bord). Lire le SICP et TSPL3

En TP utilisez **Racket** qui est un dialecte moderne de Scheme, avec l'interface graphique `drracket`

Quelques exemples :

— l'expression `(+ 3 (* 4 5))` $\rightarrow$[3] 23
— les identificateurs sont dit *symboles* et peuvent avoir des caractères non alphabetiques `a-b` est un seul symbole, comme `even?` ; la casse est non-significative ; les commentaires commencent par le point-virgule **;** jusqu'au bout de la ligne
— la quotation `(quote x)` abrégé `'x` $\rightarrow$ le *symbole* `x`
— les litéraux `42` -entier-, `#t` (vrai) `#f` (faux) -booléens- `"abc"` -chaîne- et peut-être `()` (nil) ; attention aux booléens (seul `#f` est faux en Scheme ; tandis qu'en Lisp ou Melt `()` est faux).
— la construction d'une liste `(list '+ 2 3)` $\rightarrow$ `(+ 2 3)` (construction d'une expression à l'exécution) et les opérateurs `cons`, `car`, `cdr` (nom de registres `CAR` et `CDR` sur l'IBM 704)
— les tests (pour typage) : `list?`, `cons?`, `number?`, `even?`, `string?`, `symbol?`
— définition de constante `(define deux 2)` puis `deux` $\rightarrow 2$
— definition de factorielle
`(define (fact n) (if (<= n 0) 1 (* n (fact (- n 1)))))`
— l'évaluateur `(eval '(+ 2 3))` $\rightarrow 5$
— les fonctions sont des valeurs par exemple + $\rightarrow$ `<procedure:+>`
— l'affectation `set!` ainsi `(let ( (x 1) ) (set! x 0) x)` $\rightarrow 0$
— les fonctions anonymes : `(map (lambda (x) (+ 1 x)) (list 1 2))` $\rightarrow$ la liste `(2 3)`
— la définition locale `(let ( (x 1) (y 2) ) (+ x y))` $\rightarrow 3$
— le `begin` (ou `progn` en Lisp) pour évaluer plusieurs expressions pour leur effet de bord, et `(begin (display 1) (newline) 2)` $\rightarrow 2$ avec affichage préalable d'un 1 ou simplement en Racket `(begin (format "~a\n" 1) 2)`
— les structures : `(struct point (x y))` définit `point`, `point?`, `point-x`, `point-y` `set-point-x` etc...
— le **`call/cc`** ("call with current continuation") capture la continuation courante[4] ; ainsi `(call/cc expr)` (où `expr` "est une fonction")

1. évalue `expr` en une fonction ϕ
2. capture la continuation courante
3. construit une fermeture κ qui réifie cette continuation

3. Lire "s'évalue en"
4. Voir Cours de K.Lee

4. applique ϕ à cette continuation réifiée κ
5. renvoie le résultat de $\phi(\kappa)$ sauf si l'évaluation dans `expr` de ϕ appelle κ sur une valeur v qui est alors renvoyée

Lire attentivement une sémantique formelle de Scheme Operational Semantics for R^5RS Scheme (J.MATTHEWS, R.B.FINDLER). ou R5RS formal (JAFFER)

La wikipage sur transformation CPS est aussi à lire

2 Melt

Un langage lispien domaine spécifique pour étendre le compilateur GCC : voir **`gcc-melt.org`**
Les données : valeurs et trucs de Melt.
Un petit rappel sur les ramasse-miettes, notamment Algorithme de Cheney ; les `Gimple` et `Tree` de GCC.

3 en TP

Jouer avec `call/cc`
— un programme (sans récursion ni tération, avec `call/cc` uniquement, qui compte $0, 1, 2, 3...$ indéfiniment
— un programme qui ajoute deux arbres (représentés comme des listes) et meure tout de suite dès qu'un sous-arbre a une "forme" différente à gauche et à droite
— si vous avez le temps et d'autres idées sur `call/cc`

Projet 2 (suite)

Selon les usages du libre, testable en ligne de commande sous Linux Debian, étendre l'interprète de votre choix (voir note de cours précédente). Si possible mettre le nom du binôme dans le dépôt `git`. Commencer cette semaine ! envoyer sur **`conception-langage-upmc2013@googlegroups.com`**