Conception de langage
Examen final – Janv. 2009

Un lambda terme

Soit le lambda-terme $\lambda x.\lambda y.(y\ ((x\ x)\ y))$, appelons le p. Soit l'application $(p\ p)$, appelons la Π. Soit t un lambda-terme quelconque.

1. donnez quelques étapes de réduction de $(\Pi\ t)$.
2. que pouvez vous dire de Π ?

Un programme simple

Soit le petit programme suivant

```
Var x = 3; Var y = 4; x := x+y; y := x-y; x := x-y
```

1. on se place dans la sémantique simple avec
   ```
   P   :  Prog -> Mem
   D   :  Dec -> (Env * Mem) -> (Env * Mem)
   S   :  Stat -> (Env * Mem) -> Mem
   E   :  Expr -> (Env * Mem) -> Val
   ```
   Pour simplifier, on pourra noter la mémoire ou l'environnement sous forme de listes d'associations en oubliant les indications `inData` et `inAdr`. Par convention, les adresses seront notées `a1`, `a2`, etc. Exemples : `[]`, `[x,0]`, `[x,0 ; z,a1]`, etc. pour les environnement et `[]`, `[a1,12 ; a2,22]`, etc. pour les mémoires.
   (a) en utilisant les équations sémantiques, donnez l'expression fonctionnelle correspondant au petit programme proposé.
   (b) en réduisant cette expression, donnez la valeur (ie. la mémoire) résultant de l'exécution de ce programme.
2. on se place maintenant dans la sémantique à continuation avec
   ```
   P   :  Prog -> Mem
   D   :  Dec -> (Env * Mem) -> (Env * Mem)
   S   :  Stat -> (Cont * Env * Mem) -> Mem
   E   :  Expr -> (Env * Mem) -> Val
   ```
   On pourra également simplifier comme ci-dessus les notations pour mémoire et environnement.
   (a) en utilisant les équations sémantiques avec continuation, donnez l'expression fonctionnelle correspondant au petit programme proposé.
   (b) en réduisant cette expression, donnez la valeur (ie. la mémoire) résultant de l'exécution de ce programme.

.../...

Une boucle

On introduit dans le langage le boucle

'do' s 'until' e

où s est le corps de la boucle et e l'expression booléenne de contrôle. Intuitivement,

l'instruction 'do' s 'until' e est équivalente à la séquence : s ';' 'while' e 'do' s.

On se place dans la sémantique simple (*cf* question précédente). On donne l'équation sémantique simplifiée du 'while' suivante :

```
S[['while' e 'do' s]](r,m') =
 (!w.\m'.
    if (E[[e]](r,m))
    then (w (S[[s]](r,m')))
    else m'
  m)
```

On rappelle que `!x.t` se réduit en `t[!x.t/x]`.

1. en vous basant sur l'équivalence signalée ci-dessus et en utilisant l'équation sémantique (simplifiée) du 'while' également ci-dessus rappelée, donnez l'équation sémantique, non récursive, de la boucle 'do' .. 'until'.
2. utilisez vos équations pour «exécuter» le programme

```
Var x = 2; do x := x-1 until (x=0)
```

Une autre boucle

On se donne ici une boucle sans clause de garde, de la forme

'loop' s

ainsi qu'une instruction atomique

'exit'.

Intuitivement, 'loop' boucle sans fin et 'exit' fait sortir du corps de la boucle.

On se place dans la sémantique à continuation (*cf* deuxième question). Pour donner la sémantique de nos nouvelles instructions, on s'inspirera de celle des exceptions. Mais ce sera ici plus simple : on utilisera le mot clef 'exit' comme un identificateur ; en entrée d'une boucle 'loop' on mémorise dans l'environnement l'association entre l'identificateur 'exit' et la continuation courante ; l'instruction 'exit' est traitée comme une variable, elle donnera accès à la continuation qui lui est associée dans l'environnement.

1. donnez les équations sémantiques des instructions 'loop' et 'exit'.
2. quelle solution préconiseriez vous pour traiter les 'exit' apparaissant en dehors du corps d'une boucle ?