LINFO1104 : TP4 Langage Noyau et Machine Abstraite

Certains exercices mentionnent le concept de *langage noyau*. Un langage noyau est un langage simple et transparent qui facilite le raisonemment sur l'exactitude d'un programme. Tout langage dit pratique, c'est à dire muni d'une riche syntaxe et d'outils pour aider le programmeur peut être traduit en langage noyau. Un langage transparent signifie que rien n'est caché tout devient visible, par exemple le code suivant :

```
fun {Fact N}
  if N==0 then 1
  else N*{Fact N-1} end
end
```

Devient en langage noyau:

```
declare Fact in
proc {Fact N ?R}
  local B Zero in
    Zero = 0
    B = (N == Zero)
    if B then
      R = 1
    else
      local R1 N1 One in
        One = 1
        N1 = N - One
        {Fact N1 R1}
        R = N * R1
      end
    end
  end
end
```

Le programme devient plus long, mais tout est visible.

Remarquez que:

- Fact devient une procédure avec un argument qui contiendra la valeur de retour.
- Les variables intémédiaires, comme B, deviennent visibles.
- Le "`?`" avant l'argument "`R`" de la procédure `Fact` est seulement ajouté à titre indicatif. Il averti le programmeur que cet argument est la "valeur de retour" de la procédure. Le mettre avant le nom de l'argument ou non ***ne change en rien*** l'interprétation du programme par le compilateur. Il l'ignore tout simplement.
- Pour respecter totalement la syntaxe du langage noyau, on aurait dû déclarer **un seul** identificateur par instruction `local`. C'est-à-dire que la ligne :

  ```
  local R1 N1 One in
      % instructions
  end
  ```

  Deviendrait :

  ```
  local R1 in
    local N1 in
      local One in
        % instructions
      end
    end
  end
  ```

  Cependant, le nombre de lignes du programme augmente et la lisibilité du code est plus compliquée. Nous vous autorisons alors de grouper la déclaration des identificateurs sur une seule ligne. C'est la seule régression par rapport au langage noyau que nous acceptons ;-) !

1. **Traduction en langage noyau** Soit les deux programmes ci-dessous, traduisez-les en langage noyau.

```
declare
fun {Sum N}
  if N == 1 then 1
  else N*N + {Sum N-1} end
end
```

```
declare
fun {SumAux N Acc}
  if N == 1 then Acc + 1
  else {SumAux N-1 N*N+Acc} end
end

fun {Sum N}
  {SumAux N 0}
end
```

   Quelle différence remarquez-vous au niveau de l'appel récursif?

2. **Seulement des enregistrements** Dans le langage noyau, il n'y a que des enregistrements. Un tuple est un enregistrement et les listes sont des tuples. Représentz les structures suivantes comme des enregistrements.

   - `[1]`
   - `[1 2 3]`
   - `nil`
   - `state(4 f 3)`

3. **Environnement Contextuel** Pour les instructions suivantes, donnez l'environnement contextuel de chaque procédure ou fonction.

```
proc {Q A} {P A+1} end
```

```
proc {P} {Browse A} end
```

```
local P Q in
  proc {P A R} R=A+2 end
  local P R in
    fun {Q A}
      {P A R}
      R
    end
    proc {P A R} R=A-2 end
  end
  %% Qu'affiche {Browse {Q 4}} ?
end
```

4. **Peloton d'exécution.** En utilisant la sémantique formelle du langage, exécutez les deux programmes suivants. Faites attention à l'ordre des instructions dans chacun des cas!

```
%% Programme 1
local Res in
   local Arg1 Arg2 in
      Arg1=7          % 1
      Arg2=6          % 2
      Res=Arg1*Arg2   % 3
   end
   {Browse Res}
end
```

```
%% Programme 2
local Res in
   local Arg1 Arg2 in
      Arg1=7          % 1
      Res=Arg1*Arg2   % 3
      Arg2=6          % 2
   end
   {Browse Res}
end
```

L'environnement de départ est {`Browse` → *browse*}. Il mentionne l'identificateur `Browse`, qui est lié à une variable

browse, que nous supposons définie par le système. Par souci de lisibilité, nous écrivons les variables en italique. La pile sémantique initiale pour le programme 1 est donc

$$\left[\left(\begin{array}{l}\texttt{local Res in}\\ \quad\texttt{local Arg1 Arg2 in}\\ \quad\quad\texttt{Arg1=7}\\ \quad\quad\texttt{Arg2=6}\\ \quad\quad\texttt{Res=Arg1*Arg2}\\ \quad\texttt{end}\\ \quad\texttt{\{Browse Res\}}\\ \texttt{end}\end{array}\quad,\{\texttt{Browse} \to \mathit{browse}\}\right)\right],(\mathit{browse} = (\texttt{proc \{\$ X\} ... end}, ...))$$

L'effet de l'instruction `local` est de créer une variable en mémoire, disons *res*, et d'ajouter l'association `Res` $\to$ *res* à l'environnement. La pile sémantique devient

$$\left[\left(\begin{array}{l}\texttt{local Arg1 Arg2 in}\\ \quad\texttt{Arg1=7}\\ \quad\texttt{Arg2=6}\\ \quad\texttt{Res=Arg1*Arg2}\\ \texttt{end}\\ \texttt{\{Browse Res\}}\end{array}\quad,\left\{\begin{array}{r}\texttt{Browse} \to \mathit{browse}\\ \texttt{Res} \to \mathit{res}\end{array}\right\}\right)\right],\left(\begin{array}{l}\mathit{browse} = (\texttt{proc \{\$ X\} ... end}, ...)\\ \quad\mathit{res}\end{array}\right)$$

Ensuite, il faut séparer la séquence d'instructions puisqu'on ne peut en exécuter qu'une seule à la fois. La pile sémantique devient

$$\left[\begin{array}{l}\left(\begin{array}{l}\texttt{local Arg1 Arg2 in}\\ \quad\texttt{Arg1=7}\\ \quad\texttt{Arg2=6}\\ \quad\texttt{Res=Arg1*Arg2}\\ \texttt{end}\end{array}\quad,\left\{\begin{array}{r}\texttt{Browse} \to \mathit{browse}\\ \texttt{Res} \to \mathit{res}\end{array}\right\}\right)\\ \left(\texttt{\{Browse Res\}}\quad,\left\{\begin{array}{r}\texttt{Browse} \to \mathit{browse}\\ \texttt{Res} \to \mathit{res}\end{array}\right\}\right)\end{array}\right],\left(\begin{array}{l}\mathit{browse} = (\texttt{proc \{\$ X\} ... end}, ...)\\ \quad\mathit{res}\end{array}\right)$$

Continuez l'exécution du programme, en notant bien ce qui se trouve en mémoire. Ensuite, faites de même pour le programme 2. Quel est précisément l'état final de chaque programme? Qu'affichent-ils chacun? Quelles sont les variables créées par chacun? Que deviennent ces variables lorsque les programmes se terminent?

5. **Sémantique et procédures.** Utilisez la sémantique du langage pour détermniner les résultats affichés par le programme ci-dessous. Les fonctions `Add1` et `Add2` sont définies par le même code. Pourquoi renvoient-elles des résultats différents? Expliquez sur base de la sémantique.

```
local MakeAdd Add1 Add2 in
  proc {MakeAdd X Add}
    proc {Add Y Z}
      Z=X+Y
    end
  end
  {MakeAdd 1 Add1}
  {MakeAdd 2 Add2}

  local V in
    {Add1 42 V} {Browse V}
  end

  local V in
    {Add2 42 V} {Browse V}
  end
end
```