Solution proposée pour l'examen Septembre 2022 de LINFO1104

Solutions par @norahrmoire et ré-écris en Latex par @tfloow sur discord. Si vous trouvez une faute merci de le signaler ici.

1. Translate the following program into kernel language. Be careful to do a complete translation that uses only kernel language instructions ! :

```
local F C in
  C={NewCell 1}
  fun {F A}
    @A
  end
  C:={F C}+1
end
```

2. Basic concepts.
   (a) Define "scope" in a programming language. Give an example.

   Solution: La portée d'un identificateur est la partie du programme où l'occurence se réfère à la même déclaration de variable :

```
local X in X=42
  {Browse X} % Affiche 42
  local X in X=11
    {Browse X} % Affiche 11
  end
  {Browse X} % Affiche 42
end
```

   (b) Define "contextual environment" of a procedure. Give an example.

   Solution: L'environnement contextuel d'une fonction (ou procédure) contient tous les identificateurs qui sont utilisés dans la fonction mais déclarés en dehors.

```
declare
A=1
proc {Inc X Y} Y=X+A end
```

3. Lambda calcul
4. Give a code example Ofa functional Object Calling a functional objectreturns another objectwith a new value inside. For example, CI=NewCounter creates the counter Object CI with initial count O. Then returns a new counter Object C2 with count 1, and returns a new counter Object C3 with count 2. Hints : A functional objectis purely declarative ; cells are not allowed. You need to define an auxiliary function CounterObject I that returns a counter Object with count I. A counter Object is a record with an 'inc' field that contains an increment function.

   Solution:

```
local
  fun {CounterObject I}
    fun {Inc} {CounterObject I+1} end
    fun {Get} I end
  in
    count(inc:Inc get:Get)
  end
in
  fun {NewCounter} {CounterObject 0} end
end
```

5. Nondeterminism
   (a) Define the concept of nondeterminism in a programming language. Be careful to give a precise definition ! It is a bit subtle. Give an example of a program that shows nondeterminism.

Solution: C'est la capacité d'un système à faire des décisions indépendamment du développeur.

```
declare X
thread X=1 end
thread X=2 end
{Browse X}    % Le resultat est parfois 1 ou 2
```

   (b) Deterministic dataflow has the strong property that it has no observable nondeterminism. For this question, first give a specification of a client/server application, second use this specification to explain Why the client/server cannot be v ;ritten as a deterministic dataflow program. Third, briefly define the concept of a port and fourth show in a few lines of code how to write a client/server with a port.

Solution: Sans non-déterminisme observable, le résultat est toujours le même. Or, dans une application Client Serveur, le résultat dépend de quand on envoit une requête. On reçoit une requête, on compile et on répond au client. Il ne peut donc **pas** y avoir de non-déterminisme. Le résultat est choisi par le développeur et non pas le scheduler donc problématique.

Un port est un *stream* tel que :

```
P = {NewPort S} % Cree un port avec un stream S
{Send P N} % Ajoute N au stream de notre port
```

Client serveur avec un port :

```
proc {Server S}
  case S
  of H|T then {Handle H} {Server T} end % Handle etant une procedure realisant
    quelque chose
end
```

6. Erlang
   (a) An Erlang process is similar to a port Object. An important primitive operation to support fault tolerance in Erlang is linking, where processes are linked together. For this question, define whathappens when processes are linked. First, explain what happens when a linked process terminates normally. Second, explain what happens when a linked process terminates with a run-time error. Third, what is the difference in behavior between a worker process and a supervisor process when either is linked ? Fourth, what happens when more than processes are linked together ?

Solution:

1. un processus termine par un motif de sortie (*exit* reason) qui est envoyé comme un signal à tous les processus liés. Quand il se termine normalement, il renvoie l'atome "*normal*"
2. Quand il y a un *run-time error*, le motif de sortie est *{Reason, Stack}*. C'est envoyé à tous les autres processus reliés, ils crashent aussi.
3. Un superviseur est un processus dont le seul but est de démarrer, surveiller et redémarrer les *workers*.
4. Comme un processus renvoie son motif de sortie à tous les processus auxquels il est lié, si le message est autre que "*normal*", les processus liés crashent aussi.

   (b) In Erlang it is possible to update the code of a running system without stopping. Erlang has basic mechanisms to support this. For this question, explain the two mechanisms and give small code fragments in Erlang to illustrate them (as we did in the course). Hint : one mechanism is supported by the implementation, whereas the other mechanism is just a consequence of higher-

order programming.

Solution: Erlang permet d'avoir 2 versions du code qui tournent en même temps. (permet de modifier le code sans l'arrêter).
Quand un code est changé, les processus peuvent choisir de continuer avec l'ancien code ou d'utiliser le nouveau. Les nouveaux processus sont d'office liés à la version la plus récente.

```
1 -module(m).
2
3 loop(Data, F) ->
4   receive
5   (From,Q} ->
6     {Reply,Data1}=F(Q,Data),
7     m:loop(Data1, F)
8 end.
```

Listing 1 – use new version

```
1 -module(m).
2
3 loop(Data, F) ->
4   receive
5   {From,Q} ->
6     {Reply,Data1}=F(Q,Data),
7     loop(Data1, F)
8 end.
```

Listing 2 – use old version