

## Devoir d'OCaml À rendre pour le 7 décembre 2011

La série des jeux de simulation Anno propose au joueur de mettre en place des ensembles de producteurs et de consommateurs de ressources, afin de faire vivre et se développer une colonie à l'époque de la Renaissance. Pour se faire une petite idée de la complexité de ce programme, la liste des ressources possibles et des personnages les consommant est accessible à la page suivante :

[http://anno1404.wikia.com/wiki/Production\\_and\\_consumption\\_rates](http://anno1404.wikia.com/wiki/Production_and_consumption_rates)

*Exemple* : Une usine de pêche produit sans rien consommer 1 tonne de poisson toutes les 30 secondes. Un paysan consomme à lui tout seul 10 kg de poisson et 4.4 kg de cidre par minute (les personnages de ce jeu sont particulièrement bien portants).

Supposons vouloir mettre en place un simulateur simplifié pour ce type de jeu, sous la forme d'un système multi-agent interagissant à travers ces ressources. Pour représenter ces agents, on propose d'utiliser des threads. En OCaml, la bibliothèque Lwt (<http://ocsigen.org/lwt/manual>) offre la possibilité de programmer relativement facilement un ensemble de threads, en fournissant des primitives pour la création, l'exécution parallèle, et même l'interaction de ces threads entre eux.

*Exemple* : le code suivant crée deux threads en parallèle, qui affichent "Heads" et "Tails" respectivement après 1 et 2 secondes :

```
open Lwt
let out_s = Lwt_io.write Lwt_io.stdout;;

let () =
  (out_s "Started\n") >>
    (((Lwt_unix.sleep 1.) >> (out_s "Heads\n")) <&>
     ((Lwt_unix.sleep 2.) >> (out_s "Tails\n"))) >>
    (out_s "Finished\n");;
```

Noter l'utilisation des opérateurs `>>` et `<&>` pour combiner les threads.

L'idée de ce devoir consiste alors à représenter chaque usine par un agent qui produit des ressources à intervalles réguliers, et chaque groupe de personnages par un agent qui consomme lesdites ressources, là aussi à intervalles réguliers. Toutes ces opérations sont faites en parallèle, et l'un des objectifs de ce devoir est de détecter les moments où les ressources viennent à manquer. Au final, il devient alors possible, pour une population donnée, de construire la liste de producteurs nécessaires à la survie de cette population.

*Questions :*

1. Écrire un ensemble de types inductifs permettant de représenter les valeurs manipulées par le simulateur.
2. Les producteurs et consommateurs doivent pouvoir communiquer entre eux pour se transmettre les ressources. Quel type de la bibliothèque Lwt répond le mieux à cette problématique de communication ?
3. Créer un entrepôt global dans lequel toutes les ressources disponibles seront placées.
4. Créer un ensemble de threads producteurs, transformateurs, et consommateurs, qui interagissent entre eux sans se bloquer les uns les autres, de manière à pouvoir détecter les ressources indisponibles.
5. Améliorer le système en rajoutant un superviseur capable de rajouter de nouveaux threads lorsque des ressources sont trop souvent indisponibles.
6. Quelles sont les techniques de programmation fonctionnelle encouragées à travers l'utilisation la bibliothèque Lwt ?

*Remarques :*

- Il est recommandé de faire attention à ne faire *aucun* appel système potentiellement bloquant qui n'appartienne déjà à la bibliothèque Lwt, sous peine d'obtenir des threads sans aucun parallélisme. Cela concerne en particulier les entrées/sorties (`Printf.printf`) mais aussi les fonctions système (`Unix.sleep`).
- Il n'est pas nécessaire d'utiliser l'extension de syntaxe propre à Lwt pour réaliser ce devoir, même si cela n'est pas non plus interdit.
- Il est recommandé de fixer une unité globale de ressource (par exemple, la tonne), et une unité globale de temps (par exemple, la minute), et de considérer que *tous* les agents ne consomment ou ne produisent que des nombres *entiers* d'unités lors d'une action.
- L'exactitude numérique des taux de production et de consommation n'est pas une priorité du devoir, même si elle sera appréciée. Il n'est pas demandé de mettre en place l'ensemble des lignes de production possibles, mais au moins de permettre de simuler les deux types de personnes les moins évoluées (*peasant* et *citizen*).
- La documentation de la bibliothèque Lwt (<http://ocsigen.org/lwt/api>), ainsi que les exemples disponibles (<http://ocsigen.org/darcsweb/?r=lwt;a=tree;f=/examples>) constituent une aide précieuse pour comprendre le fonctionnement de l'ensemble.

**Modalités de rendu :** Ce devoir est à rendre pour le 7 décembre 2011. Il doit être rendu par mail à l'adresse `renault@labri.fr`, avec au début de l'en-tête l'identificateur [Devoir TAP] et le nom des personnes ayant réalisé le devoir.

Ce mail est censé contenir un fichier `.ml` contenant les réponses aux questions ci-dessus séparées par des commentaires identifiant les numéros des questions et enfin éventuellement toute remarque supposée utile. Il est possible de faire ce devoir par groupes de *2 personnes au plus*.

Dans le cadre de ce devoir, l'utilisation du style impératif (par l'utilisation de structures de données ou de constructions impératives) doit se limiter *au strict nécessaire*. Il est recommandé de se servir des fonctions de la bibliothèque standard dans la mesure où elles allègent l'écriture des nouvelles fonctions. Il n'est pas nécessaire de se soucier outre mesure de la complexité des fonctions écrites. Par contre, la façon dont le code est présenté (indentation, commentaires. . .), la qualité des commentaires et l'exhaustivité des tests sont des éléments jouant un rôle prépondérant dans l'évaluation du devoir.

## Annexe

### 1 Exemple de Makefile

```
OCAML_PACK = unix,lwt,lwt.unix,lwt.syntax

all :
    @ocamlfind ocamlc -syntax camlp4o \
    -package $(OCAML_PACK) devoir.ml \
    -linkpkg -o devoir

clean :
    rm -f devoir a.out
```

## Références

[Wad92] Philip Wadler. Comprehending monads. In *Mathematical Structures in Computer Science*, pages 61–78, 1992. An online version of this article is available at the following address <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.5381>.