Implémentation de réseaux de Kahn

Christophe Cordero, Li-yao Xia

26 mai 2013

1 Composition du projet

Nous présentons maintenant les implémentations dans l'ordre où elles ont été écrites, qui est aussi, selon nous, dans l'ordre de technicité croissante.

2 Implémentation séquentielle

On représente un processus par le type suivant :

```
type 'a process =
  | Proc of unit -> 'a process
  | Res of 'a
```

Le constructeur `Proc` contient une fonction qui effectue un pas *atomique* de calcul et :

1. Si le calcul n'est pas terminé, il renvoie la suite du calcul sous forme d'un autre processus `Proc`,
2. Sinon, il renvoie le résultat `Res`.

L'exécution `run` d'un tel processus est simple : on fait avancer le calcul jusqu'à obtenir un résultat.

Cette représentation permet alors de simuler un processus `p` obtenu avec la fonction `doco`, qui représente léxecution parallèle d'une liste de processus.

On définit un pas de calcul atomique pour `p` comme l'éxecution d'un pas de calcul pour chacun des processus contenus dans la liste en argument de `doco`.

Les canaux communication sont des files.

```
type 'a in_port = 'a Queue.t
type 'a out_port = 'a Queue.t
```

On remarquera qu'on représente le blocage d'un processus `get qi` en retournant le même processus quand la file `qi` est vide.

Cette implémentation ne pose pas de problème majeur et produit un code très court.

3 En utilisant des processus et des tubes

On représente un processus par le type le plus simple :

```
type 'a process = unit -> 'a
```

Le parallélisme est implémenté avec l'utilisation de `Unix.fork`.

Les canaux de communication sont construits à l'aide des fonctions `Unix.pipe`, `Unix.in_channel_of_descr`, `Unix.out_channel_of_descr`.

```
type 'a in_port = in_channel
type 'a out_port = out_channel
```

Les types `in_channel` et `out_channel` permettent d'utiliser le module `Marshal` directement.