MGL7160: Exercice #5

27 février 2001

Dans les machines modernes, le processeur est toujours plus rapide que la mémoire. Le temps requis pour accéder à la mémoire correspond à plusieurs cycles de processeur. Pour réduire le temps d'accès, on utilise des caches, plus rapides mais plus petites que la mémoire. Une cache ne contient qu'une copie *partielle* de la mémoire: lorsque la cache est pleine, il faut alors éjecter de l'information déjà présente.

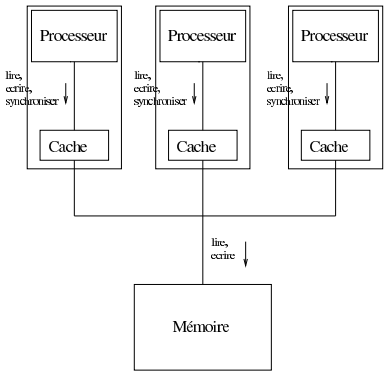

Figure 1: Flux des messages pour une machine multi-processeurs à mémoire partagée

Vous devez développer un type mutable modélisant le comportement d'une cache. La figure 1 illustre la structure générale d'une multi-processeurs à mémoire partagée, où chaque processeur possède sa propre cache et où les processeurs accèdent à la même mémoire. Les identificateurs indiqués près des flèches sur cette figure dénotent les noms des messages échangés entre les divers objets, messages que nous allons décrire plus en détail dans les paragraphes qui suivent.

Pour votre spécification, vous *devez* utiliser les modules suivants, disponibles dans la bibliothèque des types `Spec`:[1]

```
DEFINITION types_pour_cache
  CONCEPT adresse: type
  CONCEPT valeur: type
END

TYPE memoire
  INHERIT mutable{memoire}
  IMPORT adresse valeur FROM types_pour_cache

  MESSAGE lire  ( m: memoire, a: adresse )  REPLY( v: valeur )
  MESSAGE ecrire( m: memoire, a: adresse, v: valeur )
END
```

Vous devez spécifier un type mutable `cache` exportant les messages suivants:

- `creer( taille_cache: nat, m: memoire SUCH THAT taille_cache > 0 )`: Cette opération crée une nouvelle `cache` pouvant contenir de l'information pour au plus `taille_cache` adresses. Lorsqu'on crée la `cache`, on doit spécifier la mémoire (argument `m`) à laquelle cette cache est associée. Ceci est nécessaire parce que les opérations qui suivent vont devoir, dans certains cas, accéder à cette mémoire pour obtenir ou mettre à jour l'information de la (à l'aide des opérations `lire` ou `ecrire` exportées par le module `memoire`).
- `lire( c: cache, a: adresse )`: Cette opération retourne la `valeur` associée à l'adresse `a`. Si une copie de l'information pour cette adresse est déjà présente dans la cache, alors aucun accès à la mémoire n'est requis. Sinon, un appel à l'opération `lire` à la mémoire associée à la cache doit être effectué, créant ainsi une copie de l'information dans la cache. Dans le cas où la cache est déjà pleine, il faut alors éjecter une des adresses déjà présente — voir plus bas le comportement requis lorsque l'adresse éjectée correspond à une adresse dont le contenu avait été modifié.

[1] `http://www.info.uqam.ca/~inf3140/bibliotheque-spec.cgi`.

- `ecrire( c: cache, a: adresse, v: valeur )`: Cette opération vise à écrire la valeur `v` dans l'adresse `a`. Si cette adresse est déjà présente dans la cache, alors aucun accès à la mémoire n'est requis. Notons que l'approche d'écriture utilisée ici est dite "*write-back*": une valeur écrite dans la cache n'est *pas* recopiée dans la mémoire immédiatement; l'écriture se fait uniquement lorsque l'adresse doit être éjectée de la cache, ou lorsque l'opération `synchroniser` est appelée. Notons que cela implique qu'il est nécessaire de savoir, pour chaque adresse conservée dans la cache, si cette adresse contient une valeur possiblement différente de celle contenue dans la mémoire — on parle alors d'une adresse qui est *sale* (*dirty*).

  Notons aussi que si l'adresse n'était pas déjà présente dans la cache, il faut alors l'ajouter. Comme dans le cas d'une lecture, si la cache est pleine, il faut alors éjecter l'une des adresses déjà présentes. Tant dans le cas d'une lecture que d'une écriture, il est préférable, lorsque c'est possible, d'éjecter une adresse qui n'est pas sale, puisque cela n'entraîne aucun accès (`ecrire`) à la mémoire; lorsqu'on éjecte une adresse qui est sale, alors on doit transférer la valeur contenue dans la cache vers la mémoire (c'est-à-dire, mettre à jour la copie de la mémoire à l'aide du message `ecrire`).

- `synchroniser( c: cache )`: Ce message permet au processeur de nettoyer la cache. Plus précisément, l'effet de ce message est de transférer vers la mémoire une copie des valeurs associées à chacune des adresses sales. Après l'opération, les contenus de la cache et de la mémoire sont donc équivalents, synchronisés l'un avec l'autre.

Vous trouverez plus bas un exemple d'utilisation d'une cache. Quelques explications concernant cet exemple:

- La colonne de gauche indique des messages envoyés à une cache `c`. Celle du centre indique, de façon informelle, le contenu de `c` (qui contient au plus 2 adresses: `c = creer(2, m)`). Celle de droite indique les messages envoyés à la mémoire `m` par la cache, pour lire ou écrire, en fonction du message reçu par la cache et de l'état de la cache.
- Initialement, pour simplifier la présentation, on suppose que la mémoire `m` contient, à une adresse `a`, la valeur `a` (la même valeur que l'adresse elle-même).
- Le symbole "`=>`" indique la valeur retournée au processeur par un message `lire` envoyé à la cache.
- Dans la colonne centrale (contenu de la cache), "`a -> v`" indique que la valeur `v` est associée à l'adresse `a`. Le symbole "`-S>`" indique qu'il s'agit d'une adresse sale, donc qui devra être écrite en mémoire le cas échéant.

```
Operations sur c          Contenu de la cache c      Operations sur m
----------------          ---------------------      ----------------
c = creer( 2, m )
lire(c, 10) => 10         10 -> 10                   lire(m, 10)
lire(c, 20) => 20         10 -> 10, 20 -> 20         lire(m, 20)
ecrire(c, 20, 22)         10 -> 10, 20 -S> 22
lire(c, 20) => 22         10 -> 10, 20 -S> 22
lire(c, 30) => 30         30 -> 30, 20 -S> 22        lire(m, 30)
ecrire(c, 30, 33)         30 -S> 33, 20 -S> 22
lire(c, 40) => 40         40 -> 40,  20 -S> 22       ecrire(m, 30, 33); lire(m, 40)
synchroniser(c)           20 -> 22, 40 -> 40         ecrire(m, 20, 22)
```

Remarque et indice:

- Pour effectuer une opération `lire` sur la mémoire, il n'est pas nécessaire, *contrairement* à `ecrire` qui a un effet de bord, d'utiliser un `SEND`. Un appel implicite de message suffit. Par ex., pour lire l'adresse `a` de la mémoire `m`:

  ```
  CHOOSE( val_mem: valeur SUCH THAT val_mem = lire(m, a) )
  ```

- Je vous suggère de définir et d'utiliser le concept suivant (à vous de définir les concepts auxiliaires) pour simplifier le choix de l'adresse devant être éjectée de la cache lorsque celle-ci est pleine:

  ```
  CONCEPT ejectable( c: cache, a_out: adresse SUCH THAT cache_plein(c) )
          VALUE( b: boolean )
    WHERE
      b <=> existe_propre(c) => dans_cache(c, a_out) &  propre(c, a_out)
            &
           ~existe_propre(c) => dans_cache(c, a_out) & ~propre(c, a_out)
  ```