MGL7160: Exercice #6 — Solutions

a) 1. Non. La valeur par défaut n'est pas spécifiée. Il faudrait plutôt utiliser la clause suivante:

```
INITIALLY
   elems = create(0)
```

2. Non. La seule opération concernée (avec une transition) est `ajouter`. Pour ce message, aucune vérification n'est faite pour assurer que l'élément ajouté soit bien divisible par 2. Il faudrait modifier `ajouter` comme suit:

```
MESSAGE ajouter( x: nat )
  WHEN (x | 2) & size(domain(elems)) < 100
    TRANSITION
      elems = bind( x, *elems[x]+1, *elems )
  WHEN ~(x | 2)
    REPLY EXCEPTION ajout_element_invalide
  OTHERWISE
    REPLY EXCEPTION maximum_atteint
```

On pourrait aussi ajouter une pré-condition à faire vérifier par le client du module:

```
MESSAGE ajouter( x: nat SUCH THAT 2 | x )
  WHEN size(domain(elems)) < max_elems
    TRANSITION
      elems = bind( x, *elems[x]+1, *elems )
  OTHERWISE
    REPLY EXCEPTION maximum_atteint
```

3. Oui. La (seule) pré-condition `size(domain(elems))` peut toujours être évaluée sans problème, quel que soit le dictionnaire `elems` puisqu'on peut toujours déterminer le domaine d'un dictionnaire et trouver la taille d'un ensemble.

4. Non: il est possible que le quantificateur `MINIMUM` dans le message `obtenir_min` ne soit pas correctement défini si le domaine du dictionnaire est vide. Il faudrait ajouter une pré-condition et une exception:

```
WHEN domain(elems) ~= {}
  REPLY( r: nat )
    WHERE r = MINIMUM( x: nat SUCH THAT x IN elems ::  x )
OTHERWISE
  REPLY EXCEPTION dictionnaire_vide
```

Quant au message `present`, la valeur booléenne associée à `r` peut être produite sans problème puisque `elems[x]` retourne une valeur peu importe `x` (la valeur par défaut est définie) et qu'on peut toujours comparer deux `nat`.

b) 1. On pourrait avoir les deux états suivants, qui sont différents en termes des variables d'état et qui respectent tous deux l'invariant:

```
elems = {[10, 2], [20, 3]; 0}
vs.
elems = {[10, 1], [20, 4]; 0}
```

Or, ces deux états ne peuvent pas être distingués l'un de l'autre à l'aide des messages exportés par le module, et ce peu importe les séries de messages qu'on pourrait envoyer à la machine: les messsages `present` et `obtenir_min` ne permettent pas de distinguer ces deux états. De plus, si on fait appel à `ajouter(x)`, on aura le même comportement: on obtiendra des états équivalents non-distinguables ou une exception sera générée de la même façon dans les deux cas (si `x` est nouveau et `max_items = 2`).

2. Il suffirait d'ajouter le message suivant, qui permet d'observer le nombre entier associé à une clé:

```
MESSAGE nb_occurences( x: nat )
  REPLY( r: nat ) WHERE r = elems[x]
```