JML - TD 4

Maximilien Faure - Alexandre Sahuc

1er novembre 2015

I Préconditions et invariants

I.I Invariants

Invariant 1

```
(0 <= nb_inc && nb_inc < 50);
```

Le nombre de règles d'incompatibilité doit être compris entre 0 et 49 pour ne pas dépasser du tableau.

Invariant 2

```
(0 <= nb_assign && nb_assign < 30);
```

Le nombre d'assignement doit être compris entre 0 et 29 pour ne pas dépasser du tableau.

Invariant 3

```
(\forall int i; 0 <= i && i < nb_inc;
   incomp[i][0].startsWith("Prod") && incomp[i][1].startsWith("Prod"));
```

Une incompatibilité associe deux produits.

Invariant 4

```
(\forall int i; 0 <= i && i < nb_assign;
   assign[i][0].startsWith("Bat") && assign[i][1].startsWith("Prod"));
```

Chaque assignement doit être entre un bâtiment et un produit (d'abord le bâtiment puis le produit).

Invariant 5

```
(\forall int i; 0 <= i && i < nb_inc; !(incomp[i][0]).equals(incomp[i][1]));
```

Un produit ne peut pas être incompatible avec lui-même.

Invariant 6

```
(\forall int i; 0 <= i && i < nb_inc;
   (\exists int j; 0 <= j && j < nb_inc;
      (incomp[i][0]).equals(incomp[j][1])
        && (incomp[j][0]).equals(incomp[i][1])));
```

La relation d'incompatibilité est réflexive.

Invariant 7

```
(\forall int i; 0 <= i && i < nb_assign;
   (\forall int j; 0 <= j && j < nb_assign;
      (i != j && (assign[i][0]).equals(assign [j][0])) ==>
      (\forall int k; 0 <= k && k < nb_inc;
        (!(assign[i][1]).equals(incomp[k][0]))
          || (!(assign[j][1]).equals(incomp[k][1])))));
```

Deux produits incompatibles ne peuvent être placés dans un même bâtiment.

I.II Préconditions

add_incomp

```
requires nb_inc >= 0;
requires nb_inc < 49;
\end
Le nombre d'incompatibilits  dclares  avant l'appel  la fonction doit tre compris entre 0
    et 48 (respect de l'invariant 1)
\begin{lstlisting}
requires !(prod1.equals(prod2));
```

Les deux produits déclarés comme incompatibles doivent être différents (par respect de l'invariant 5).

```
requires prod1.startsWith("Prod") && prod2.startsWith("Prod")
```

Les deux arguments passées en paramètre doivent représenter deux produits (respect de l'invariant 3).

```
requires
   (\forall int i; 0 <= i && i < nb_assign;
      (\forall int j; 0 <= j && j < nb_assign;
           (i != j && (assign[i][0]).equals(assign[j][0])) &&
               (!(assign[i][1].equals(assign[j][1]))) ==>
             (!((assign[i][1].equals(prod1) || assign[i][1].equals(prod2)) &&
                (assign[j][1].equals(prod1) || assign[j][1].equals(prod2))))));
```

Deux produits déclarés incompatibles ne peuvent pas avoir été placés dans un même bâtiment auparavant (invariant 7).

add_assign

```
requires nb_assign < 29;
requires nb_assign >= 0;
```

Le nombre d'assignments déclarés avant l'appel à la fonction doit être compris entre 0 et 28 (respect de l'invariant 2).

```
requires bat.startsWith("Bat");
requires prod.startsWith("Prod");
```

Les paramètres doivent être un bâtiment et un produit (respect de l'invariant 4).

```
requires
    (\forall int i; 0 <= i && i < nb_assign;
        (bat.equals(assign [i][0])) ==>
         (\forall int k; 0 <= k && k < nb_inc;
              (!prod.equals(incomp[k][0])) || !(assign[i][1]).equals(incomp[k][1])));
```

Le produit `prod` placé dans le bâtiment `bat` ne peut pas être incompatible avec d'autres produits

de ce bâtiment (respect de l'invariant 7).

compatible

```
requires prod1.stratsWith("Prod");
requires prod2.startsWith("Prod");
```

Les deux arguments doivents être des produits.

```
ensures \result == true ==>
   (\forall int i; 0 <= i && i < nb_inc;
      ((!incomp[i][0].equals(prod1)) || (!incomp[i][1].equals(prod2))));
```

La fonction renvoie `true` si les deux produits ne sont pas incompatibles.

findBat Pour la fonction `findBat`, nous avons décidé de renvoyer le bâtiment où est stocké le produit si ce dernier a déjà été stocké. Sinon, nous renvoyons le premier bâtiment rencontré où le produit peut être stocké. Si, dans tous les bâtiments où un produit à déjà été assigné, il n'y a aucun bâtiment qui peut contenir le produit passé en paramètre, la fonction renvoie `null`.

```
ensures \result != null <==>
        (\forall int i; 0 <= i && i < nb_assign;
           (!assign[i][0].equals(\result)) || compatible(assign[i][1], prod));
```

Si le résultat est différent de null (donc un bâtiement a été trouvé), le produit n'est pas incompatible avec les autres produits de ce bâtiment.