RIPOLL Guillaume

Algorithmes du traitement des signaux absents :

1 Traitement explicite :

Dans le cas du traitement du langage Signal vers du Java, où on imagine un thread qui calcule une ou plusieurs assignations Signal :

Cela revient à définir l'absence comme une valeur booléenne :

```
public boolean present;
```

De plus, il y a un booléen pour savoir si le signal est calculé isComputed :

```
public boolean isComputed;
```

Ainsi, une entrée E non présente est définie avec `E.present = false;`

Lorsqu'une variable est calculée, on sait si les variables qu'elle utilise sont définies :

- soit à la compilation dans le cas où la seconde variable est une entrée ou dans un cas de séquentialité
- soit grâce à isComputed dans le cas de dépendance avec un autre thread :

```
while(variable_attendue.isNotComputed())
    variable_attendue.wait();
```

De plus, une variable définie par une autre variable non présente n'est pas présente, et sa valeur n'est donc pas définie. Si elle est utilisée par d'autres threads (on le sait à la compilation), isComputed passe à true et un notifyAll() est généré.

De la sorte, une variable de sortie peut être calculée comme non présente. Une fois le cycle finis, toute variable utilisée dans d'autres threads que celui qui la calcule voit isComputed passer à false. S'il y a des variables de délais à t-1, elles conservent la présence de leur variable à t0.

2 Traitement par détection de blocage :

Cette fois, on ne cherche pas à être capable de dire si une variable n'est pas présente :

1- On suppose que le scheduler des threads est capable de détecter un deadlock, c'est-à-dire un cas où chaque thread est en train d'attendre.

2- On suppose aussi qu'un thread ne calcule que des variables dépendantes entre elles :
Si

a -> x -> y
a -> b -> y

Alors x et y peuvent être dans le même thread,
Mais x et b ne peuvent pas être dans le même thread.
Cette 2ème hypothèse nous garantit que si un thread est bloqué pendant le calcul d'une variable, les calculs suivant n'auraient pas pu être faits.

3- Il y a aussi besoin d'un booléen pour savoir si le signal est calculé isComputed :

```
public boolean isComputed;
```

En effet, pour un cas comme :

a -> x -> y
a -> b -> y

où le thread T0 calcule x et y, T0 a besoin de savoir quand b est calculé.

Ainsi, comme un thread qui a finit un cycle attend les autres threads pour continuer, soit on arrive sur un deadlock, soit toutes les variables sont présentes. La détection du deadlock est alors chargée de terminer le cycle de chaque thread, ce qui en démarre un nouveau.

L'usage d'une variable à t-1 serait calculée par un thread dédié, de sorte à bloquer s'il n'a pas de valeur à donner. Pour détecter le calcul ou non de la variable (le fait de devoir ou non bloquer), il faudrait utiliser isComputed().