1 Saut de pointeur

Tout d'abord, regardons le pointeur next. Nous savons qu'au départ $next_i(0) = i + 1$ et à chaque pas $next_i(n+1) = next_{next_i}(n)$. On prouvons par récurence que $next_i$ à la n itération est égale à $next_i(n) = 2^n + i$.

Admettons que pour n ce soit vrai. $next_i(n+1) = next_{next_i}(n) = next_{2^n+i}(n) = 2^n + 2^n + i = 2^{n+1} + i$.

Maintenant dérécursivons la valeur de chaque somme partielle de la Pram. Nous savons qu'au départ v_i a une valeur. Nous voulons montrer qu'après $n = log_2(i)$ étape $v_i(n+1) = \sum_{j=i-2^n+1}^{i} v_j$ avec $v_i(0) = v_i$. A l'étape 0 l'invariant est vérifié.

Nous savons,d'après la première partie et l'algorithme qu'à l'étape $n+1$ c'est la PRAM $i - 2^n$ qui va ajouter sa valeur : $v_{next_i}(n+1) = v_{next_i}(n) + v_i(n) \equiv v_{2^n+i}(n+1) = v_i(n) + v_{2^n+i}(n)$. d'où : $v_i(n+1) = v_i(n) + v_{i-2^n}(n) = \sum_{j=i-2^{n-1}+1}^{i} v_j + \sum_{j=2^{n-1}+i-2^n+1}^{i-2^n} v_j = \sum_{j=i-2^n+1}^{i} v_j$

En prenant $n = log_2(i)$ et $i = p$ on obtient : $\sum_{j=1}^{p} v_j$.

2 Hyman

```
int turn=0;
boolean flag[]={false,false};

void Pmutex (int me){
    int you=1-me;

1.        flag[me] = true ;
2.        while (turn != me){
3.             while (flag[you])
4.                Thread.yield();
5.             turn = me;
          }
6.
}

void Vmutex(int me){
7.        flag[me] = false ;
}
```

2.1 Preuve de non deadlock

Il n'y a qu'une variable partagé turn. le principe est le même on introduit $after_0$ et $after_1$ vrai si le controle est après l'instruction $turn = 0$ respectivement $turn = 1$ Pour avoir un deadlock, il faut que les deux processus tourne infiniment dans une des boucles. Les deux processus ne peuvent pas entrer en même temps dans la ligne 3 ni 5 car cela donne : $flag1 \wedge turn0 \wedge flag0 \wedge turn1$ et respectivement : $!flag1 \wedge flag0 \wedge flag1 \wedge !floag0$.

La seule solution pour le faire rentrer dans la première boucle est de changer le $turn$ avec la ligne 5 le processus 0. Mais une fois le turn changé, on a : $flag0 \wedge turn0$. Si on continue on en sort. De l'autre coté $P0$ pour le rechanger doit passer par $flag0 \wedge !flag1$. Il n'est pas posslible d'avoir la variable $flag[1]$ à $true$ et $false$ à la fois.

2.2 Contre Exemple sécurité

```
void P0 {
1.        flag[0] = true ;
{flag0}
2.        while (turn == 1){
{turn1 & flag0}
3.              while (flag[1])
{flag1 & flag0}
4.                  Thread.yield();
{!flag1 & flag0}
5.              turn = 0;
{turn0 & flag0}
          }
}

void Vmutex(int me){
6.        flag[0] = false ;
}
```

```
void P1{
1.        flag[1] = true ;
{flag1}
2.        while (turn == 0){
{flag1 & turn0}
3.              while (flag[0])
{flag0 & flag1}
4.                  Thread.yield();
{flag1 & !flag0}
5.              turn = 1;
{flag1 & turn1}

          }
}
void Vmutex(int me){
6.        flag[1] = false ;
}
```

```
P1   |    P0
     |
 1   |              flag[1]==true
 2   |
 3   |
     |    1         flag[0]==true
     |    2
     |    6
 5   |              turn==1
 6   |              Les deux Processus sont dans la subsection
    critique.
     |
```