レポート

哲学者の食事問題

XX 大学　X 年
12345678　k5-mot

1　元のコード

問題に沿って、モデルを記述する。コード 1 内の philosopher プロセスは、哲学者の行動を表している。

コード 1　元のコード (dining_philosophers_v1.pml)

```
/* The Dining Philosophers Problem - version 1.0 */
#define N 5 /* Number of philosophers */
mtype = { UNLOCKED, LOCKED }; /* State of mutex */
mtype fork[N] = UNLOCKED; /* Mutex of forks */

/* Philosopher - Lazy person */
proctype philosopher(int id) {
    int right = id;
    int left = (id + 1) % N;
again:
    atomic { fork[right] == UNLOCKED -> fork[right] = LOCKED; };
    atomic { fork[left] == UNLOCKED -> fork[left] = LOCKED; };
    skip; /* EATING */
    atomic { fork[right] = UNLOCKED; };
    atomic { fork[left] = UNLOCKED; };
    goto again;
}

init {
    int cnt;
    for (cnt : 0 .. (N - 1)) { run philosopher(cnt); }
}
```

2　デッドロックの回避

コード 1 では、全ての哲学者が右手にフォークを持っていた場合、全ての哲学者が左側にフォークが置かれるのを待つ状態になり、これ以上プロセスが進まないデッドロックに陥る。

このデッドロックを回避するために、哲学者は、

1. 右側のフォークを取る
2. 左側のフォークが既に取られていた場合、右手のフォークを置く

という操作を行う (コード 2)。

コード 2　デッドロックを回避するコード (dining_philosophers_v2.pml)

```
/* The Dining Philosophers Problem - version 2.0 */
#define N 5 /* Number of philosophers */
mtype = { UNLOCKED, LOCKED }; /* State of mutex */
mtype fork[N] = UNLOCKED; /* Mutex of forks */

/* Philosopher - Lazy person */
proctype philosopher(int id) {
    int right = id;
    int left = (id + 1) % N;
again:
    atomic { fork[right] == UNLOCKED -> fork[right] = LOCKED; };
    atomic {
        if
        ::(fork[left] == UNLOCKED) -> fork[left] = LOCKED;
        ::else -> fork[right] = UNLOCKED; goto again;
        fi;
    };
progress:
    skip; /* EATING */
    atomic { fork[right] = UNLOCKED; };
    atomic { fork[left] = UNLOCKED; };
    goto again;
}

init {
    int cnt;
    for (cnt : 0 .. (N - 1)) { run philosopher(cnt); }
}
```

3　進行性の成立

コード 2 では、ある哲学者が連続して右手にフォークを持ち続けた場合、その右の哲学者は左手にフォークを持つことができないため、進行性が成立しない。

進行性を成立させるために、哲学者の食事の管理を行う管理者を用意した。管理者は哲学者の食事要請をキューに格納し、順番に一人ずつ哲学者に食事をさせる。

管理者と哲学者は、

1. 哲学者が管理者に食事要請メッセージを送る
2. 管理者は食事要請メッセージをキューに入れる
3. 管理者はキュー先頭の食事要請メッセージに対して、食事開始メッセージを送る
4. 哲学者は食事開始メッセージを受ける
5. 哲学者は右・左のフォークを持つ
6. 哲学者は食事する
7. 哲学者は右・左のフォークを置く
8. 哲学者は管理者に食事終了メッセージを送る
9. 管理者は食事終了メッセージを受ける
10. 3. へ戻り、キューの食事要請を処理する

の流れでやりとりを行う (コード 3)。コード 3 の manager プロセスは、管理者の行動を表している。

コード 3　進行性が成立するコード (dining_philosophers_v3.pml)

```
1  /* The Dining Philosophers Problem - version 3.0 */
2  #define N 5 /* Number of philosophers */
3  mtype = { UNLOCKED, LOCKED }; /* State of mutex */
4  mtype fork[N] = UNLOCKED; /* Mutex of forks */
5  chan request = [N] of { int }; /* Meal request queue */
6  chan ready[N] = [0] of { int }; /* Ready for meal */
7  chan done[N] = [0] of { int }; /* Done for meal */
8
9  /* Manager - Strict person */
10 proctype manager() {
11     int p0, p1;
12 accept:
13     request?p0;
14     ready[p0]!p0;
15     done[p0]?p1;
16     goto accept;
17 }
18
19 /* Philosopher - Lazy person */
20 proctype philosopher(int i) {
21     int ack;
22     int right = i;
23     int left = (i + 1) % N;
```

```
24 again:
25     request!i; // Sends a request to coordinator.
26     ready[i]?ack; // Wait until a request is granted.
27     atomic { fork[right] == UNLOCKED -> fork[right] = LOCKED; };
28     atomic { fork[left] == UNLOCKED -> fork[left] = LOCKED; };
29 progress:
30     skip; /* EATING */
31     atomic { fork[right] = UNLOCKED; };
32     atomic { fork[left] = UNLOCKED; };
33     done[i]!ack;
34     goto again;
35 }
36
37 init {
38     int cnt;
39     run manager();
40     for (cnt : 0 .. (N - 1)) { run philosopher(cnt); }
41 }
```