演習課題 3

提出日： 2022 年 7 月 28 日
所属： 知能システム科学コース
名前： 織田 康太郎

1. トランザクション T、S、U に対して、以下のスケジュールで実行されたとする。これらのスケジュールが競合直列化可能か否か判定せよ。

(1) $R_T(x)R_S(x)R_U(x)R_T(y)R_S(y)W_U(x)W_T(x)W_S(z)$

(2) $R_T(x)R_S(y)R_U(z)R_T(y)R_S(x)R_U(y)W_T(z)W_S(z)W_U(z)$

(3) $R_T(x)R_S(x)R_U(x)R_T(y)R_S(z)W_T(y)R_U(z)W_S(z)W_T(z)W_U(x)$

以下に示す db3.py を用いてスケジュールが競合直列化可能であるかの判定を行った。

また、競合直列化可能か正しく判定できているかを確認するために、オリジナルのスケジュールを作成して、それについても判定を行った。

オリジナルのスケジュール

$R_S(x)R_S(y)R_T(x)R_T(z)W_S(y)R_U(y)R_U(z)W_U(y)W_T(x)W_T(z)$

ソースコード 1: db3.py

```python
import itertools

# DB に対する操作のこと、 Rt(x) だと
# R: 操作の内容（ R/W のいずれか）
# t: どのトランザクションに属する操作か
# x: 操作の対象となるデータ
# position は元の課題に記載されているスケジュールでの位置
class Sousa:
    position = 0

    def __init__(self, sousa, transaction, data):
        self.sousa = sousa
        self.transaction = transaction
        self.data = data
        self.position = Sousa.position
        Sousa.position += 1

# 描画準備
def print_sousa(sousa):
    return sousa.sousa.upper() + sousa.transaction + '(' + sousa.data + ')'

# 描画
def print_schedule(schedule):
    N = len(schedule)
    for i in range(N):
        print(print_sousa(schedule[i]), end=' ')
```

```
    print()

# 競合しない操作のペアのリストを作成
def check_conflict(schedule):
    N = len(schedule)
    a, b = 0, 1
    ans = []
    for i in range(N*(N-1)//2):
        if not check(schedule[a], schedule[b]):
            ans.append([schedule[a], schedule[b]])
        b += 1
        if b == N:
            a += 1
            b = a + 1
    return ans

# 競合の確認
# #04.pdf の 28, 29 ページを元にして競合の確認を行う。
def check(a, b):
    if a.sousa == 'r' and b.sousa == 'r':
        return False
    if a.data != b.data:
        if a.sousa != b.sousa and a.transaction == b.transaction:
            return True
        return False
    return True

# 目的となる直列スケジュールの作成
# ここで直列スケジュールはトランザクションごとに操作をまとめたものをつなげたものとする。
# そのため、今回のようにトランザクションが
# 3つある場合、直列化可能スケジュールは 3! 個つまり6個考えられる。
def make_goal(schedule):
    transaction_T = []
    transaction_S = []
    transaction_U = []
    for i in schedule:
        if i.transaction == 't':
            transaction_T.append(i)
        elif i.transaction == 's':
            transaction_S.append(i)
        else:
            transaction_U.append(i)
    transactions = [transaction_S, transaction_T, transaction_U]
    goals = []
    for i in list(itertools.permutations(transactions)):
        serial_schedule = []
        for j in i:
            serial_schedule += j
        goals.append(serial_schedule)
    return goals
```

```
77
78 # スケジュールを構成するデータベースに対する操作を並べ替えて、
79 # 新たなスケジュールを構築する。
80 # 並べ替えには、リストの全ての並びを生成する itertools.permutations を使用する。
81 # そのとき、操作の順番が入れ替わっているものについては、競合を確認する。
82 # 全ての操作について競合しないスケジュールを返す。
83 def search_schedule(schedule, nconflicts):
84     schedules = list(itertools.permutations(schedule))
85     enable_schedules = []
86     for s in schedules:
87         for i in range(1, len(s)):
88             for j in range(i):
89                 if s[j].position > s[i].position:
90                     if [s[i], s[j]] not in nconflicts:
91                         break
92             else:
93                 continue
94             break
95         else:
96             enable_schedules.append(s)
97     return enable_schedules
98
99 # 競合直列化可能であるかを確認する。
100 # まず、競合の起きない操作のペアのリストを check_conflict を用いて作成する。
101 # 次に、 search_schedule を用いて、元のスケジュールと競合の起きないスケジュールのリストを作成する。
102 # その後、 make_goal で元のスケジュールから直列スケジュールを作成する。
103 # 最後に、直列スケジュールのいずれかが元のスケジュールと競合の起きないスケジュールのリストに
104 # 含まれるかを確認して、含まれる場合、「競合直列化可能」と表示する。
105 # また。含まれない場合、「競合直列化不可能」と表示する。
106 def solve(schedule):
107     print(' 元のスケジュール：', end='')
108     print_schedule(schedule)
109     nconflicts = check_conflict(schedule)
110     es = search_schedule(schedule, nconflicts)
111     gs = make_goal(schedule)
112     hantei = False
113     for g in gs:
114         print(' goal: ', end='')
115         print_schedule(g)
116         if tuple(g) in es:
117             print(' 可能')
118             hantei = True
119         else:
120             print(' 不可能')
121
122     print(' 結果：', end='')
123     if hantei:
124         print(' 競合直列化可能')
125     else:
126         print(' 競合直列化不可能')
```

```
    print()


S1 = [ # (1) のスケジュール
        Sousa("r", "t", "x"),
        Sousa("r", "s", "x"),
        Sousa("r", "u", "x"),
        Sousa("r", "t", "y"),
        Sousa("r", "s", "y"),
        Sousa("w", "u", "x"),
        Sousa("w", "t", "x"),
        Sousa("w", "s", "z")
    ]

S2 = [ # (2) のスケジュール
        Sousa("r", "t", "x"),
        Sousa("r", "s", "y"),
        Sousa("r", "u", "z"),
        Sousa("r", "t", "y"),
        Sousa("r", "s", "x"),
        Sousa("r", "u", "y"),
        Sousa("w", "t", "z"),
        Sousa("w", "s", "z"),
        Sousa("w", "u", "z")
    ]

S3 = [ # (3) のスケジュール
        Sousa("r", "t", "x"),
        Sousa("r", "s", "x"),
        Sousa("r", "u", "x"),
        Sousa("r", "t", "y"),
        Sousa("r", "s", "z"),
        Sousa("w", "t", "y"),
        Sousa("r", "u", "z"),
        Sousa("w", "s", "z"),
        Sousa("w", "t", "z"),
        Sousa("w", "u", "x")
    ]

S4 = [ # オリジナルのスケジュール
        Sousa('r', 's', 'x'),
        Sousa('r', 's', 'y'),
        Sousa('r', 't', 'x'),
        Sousa('r', 't', 'z'),
        Sousa('w', 's', 'y'),
        Sousa('r', 'u', 'y'),
        Sousa('r', 'u', 'z'),
        Sousa('w', 'u', 'y'),
        Sousa('w', 't', 'x'),
        Sousa('w', 't', 'z')
```

```
    ]

print(' (1) のスケジュール')
solve(S1)

print(' (2) のスケジュール')
solve(S2)

print(' (3) のスケジュール')
solve(S3)

print(' オリジナルのスケジュール')
solve(S4)
```

(2) Result

```
(1) のスケジュール
元のスケジュール：Rt(x) Rs(x) Ru(x) Rt(y) Rs(y) Wu(x) Wt(x) Ws(z)
goal: Rs(x) Rs(y) Ws(z) Rt(x) Rt(y) Wt(x) Ru(x) Wu(x)
不可能
goal: Rs(x) Rs(y) Ws(z) Ru(x) Wu(x) Rt(x) Rt(y) Wt(x)
不可能
goal: Rt(x) Rt(y) Wt(x) Rs(x) Rs(y) Ws(z) Ru(x) Wu(x)
不可能
goal: Rt(x) Rt(y) Wt(x) Ru(x) Wu(x) Rs(x) Rs(y) Ws(z)
不可能
goal: Ru(x) Wu(x) Rs(x) Rs(y) Ws(z) Rt(x) Rt(y) Wt(x)
不可能
goal: Ru(x) Wu(x) Rt(x) Rt(y) Wt(x) Rs(x) Rs(y) Ws(z)
不可能
結果： 競合直列化不可能

(2) のスケジュール
元のスケジュール：Rt(x) Rs(y) Ru(z) Rt(y) Rs(x) Ru(y) Wt(z) Ws(z) Wu(z)
goal: Rs(y) Rs(x) Ws(z) Rt(x) Rt(y) Wt(z) Ru(z) Ru(y) Wu(z)
不可能
goal: Rs(y) Rs(x) Ws(z) Ru(z) Ru(y) Wu(z) Rt(x) Rt(y) Wt(z)
不可能
goal: Rt(x) Rt(y) Wt(z) Rs(y) Rs(x) Ws(z) Ru(z) Ru(y) Wu(z)
不可能
goal: Rt(x) Rt(y) Wt(z) Ru(z) Ru(y) Wu(z) Rs(y) Rs(x) Ws(z)
不可能
goal: Ru(z) Ru(y) Wu(z) Rs(y) Rs(x) Ws(z) Rt(x) Rt(y) Wt(z)
不可能
goal: Ru(z) Ru(y) Wu(z) Rt(x) Rt(y) Wt(z) Rs(y) Rs(x) Ws(z)
不可能
結果： 競合直列化不可能

(3) のスケジュール
元のスケジュール：Rt(x) Rs(x) Ru(x) Rt(y) Rs(z) Wt(y) Ru(z) Ws(z) Wt(z) Wu(x)
```