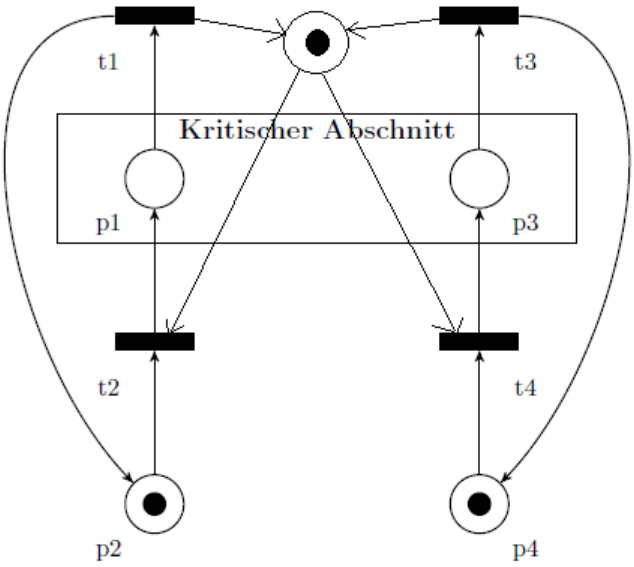
PFP SS13 Lösung

Alle Lösungen ohne Gewähr, Have Fun beim Lernen!

Aufgabe 1)

a) richtig
b) richtig
c) falsch → nur bis zu einer bestimmten Maxiamlanzahl an Threads
d) falsch
e) falsch → datenparallel, nicht task-parallel
f) 42s
g) Keine Ahnung, kein Klausurstoff mehr
h) falsch

Aufgabe 2)
a)



b)

S =

A	0
B	1
C	1
D	0
E	0
F	0

M =

	t1	t2	t3	t4	t5	t6
A	-1	-1	0	0	1	0
B	1	0	0	-1	0	0
C	0	0	1	0	-1	0
D	1	0	-1	0	0	0
E	0	1	0	0	1	-1
F	0	0	0	1	-1	0

c)

Das Petrinetz ist nicht lebendig (alle Marken können über t6 verschwinden).

d) t4 → t5 → t6 → t1 → t4

Aufgabe3)

a)

```
1       for (int i = 0; i < threads.length; i++) {
1       BlockingQueue<Integer> inQueue = queues[i];
        BlockingQueue<Integer> outQueue = queues[i+1];
        threads[i] = new StageThread(i, inQueue, outQueue, lastThread);
```

b)

```
2       Integer value = inQueue.take();
3       outQueue.add(runStage(value));
4       if (this.lastThread == true) {
                return;
        }
        out.Queue.add(value);
        return;
```

Aufgabe4)

a) SimpleLock ist nicht thread-sicher, weil z.B. in Zeile 11 und 12 eine Prüfe-Handle-Wettlaufsituation auftritt.

b) Wennn in der transfer-Methode f und t den selben Wert haben, zeigen from und to auf das Selbe Objekt. Wird dann auf beide Objekte lock() aufgerufen, kommt es zu einer IllegalMonitorStateException weil owner == self.

c) Es kann zu einem Deadlock kommen, soabld ein Thread von A nach B und der andere von B nach A buchen möchte. Jeder Thread wartet darauf, dass der jeweils andere Thread fertig wird und das Lock freigibt, was aber nie passieren wird (vgl Philosophenproblem).

d) Die Situation kann auftreten, wenn, nachdem total abgerufen wurde, withdraw() aufgerufen wird. Eine Lösung wäre beispielsweise, das lock über die for-Schleife (oder gleich die ganze Methode) auszudehnen. Alternativ: syncronized()

Aufgabe5)

a)

```
def construct: (Int => Boolean) => ((Int=>Int), (Int=>Int)) => Int => Int = {
                i               =>      (e,             o) =>   n =>
                if (i(n) == true)
                        e(n)
                else
                        o(n)
}
```

b)

```
def f: Int => Stream[Int] = {
```

```
        n => n #:: f(c(n))
}

c)
def oddCount: (Int, Int) => Int = {
                (n, k) => f(n).take(k).toList.foldLeft(0)(
                        (a, b) => if (isEven(b) == false) a+1
                                else a)
}
```

Aufgabe6)

```
a)
def getFreeColor: List[Color] => Color = {
        a => (for (i ← List.range(0, a.length) if (!a.contains(i)) yield i).head
}


b)
def colorGraph: List[Node] => List[Coloring] => List[Coloring] = {
                ns              => cs           =>
                if (ns == Nil) cs
                else
                nextFreeColor(cs) (ns.head)::colorGraph(ns.tail)(cs)
}
```