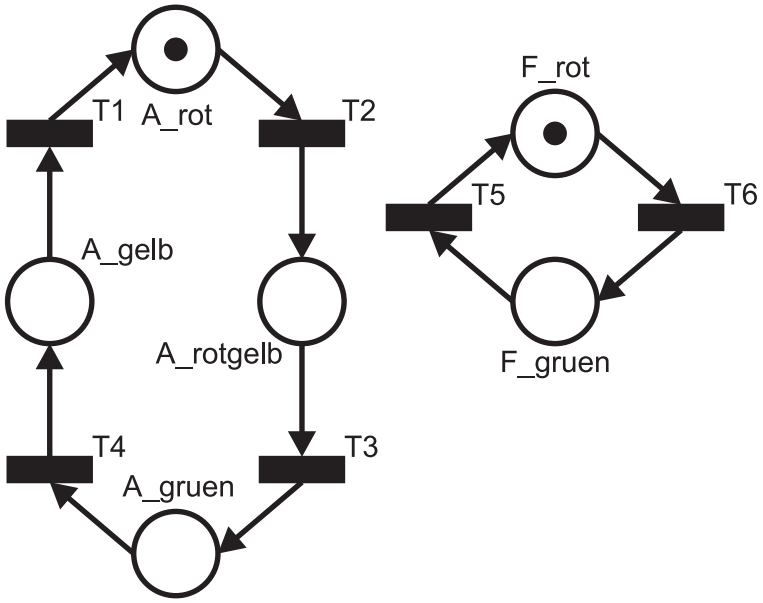
PFP-Team
pfp@i2.cs.fau.de

Übungsblatt 3

Abgabe bis 09.05.2018, 18:00 Uhr

Aufgabe 3.1: Erreichbarkeitsgraph

Gegeben sei folgendes paralleles Petri-Netz für eine einfache Fußgängerampelschaltung:



a) Zeichnen Sie den vollständigen Erreichbarkeitsgraphen für das Petri-Netz.

b) Markieren Sie alle Belegungen, die zu einer ungültigen Ampelschaltung führen.

c) Wann heißt ein Petri-Netz lebendig? Ist das Beispiel-Netz lebendig?

d) Wann heißt ein Petri-Netz beschränkt? Ist das Beispiel-Netz beschränkt?

e) Überlegen Sie sich 3 lebendige, 3 nicht lebendige, 3 beschränkte und 3 nicht beschränkte Netze, die jeweils aus mindestens 3 Stellen bestehen.

Aufgabe 3.2: Java-Synchronisation

a) An welchen Stellen im Code kann das Schlüsselwort `synchronized` verwendet werden?

b) Wie nennt sich die Synchronisations-Art, die damit betrieben wird?

c) Beschreiben Sie in eigenen Worten, wie eine Synchronisation mittels `synchronized` wirkt.

Aufgabe 3.3: Schreibtischlauf

Welche möglichen Ausgaben haben folgende parallele Programme:

a) ohne `synchronized`:

```
public class NoSync {

    static int a = 0;

    public static void main(String[] args) {
        System.out.println("Start.");
        for (int i = 0; i < 5; i++) {
            new Thread() {
                @Override
                public void run() {
                    System.out.println("This␣is␣Thread␣"
                            + a++);
                }
            }.start();
        }
        System.out.println("End.");
    }
}
```

b) mit `synchronized`:

```
public class ClassSync {

    static int a = 0;
    private static synchronized int getNextNumber() {
        return a++;
    }

    public static void main(String[] args) {
        System.out.println("Start.");
        for (int i = 0; i < 5; i++) {
            new Thread() {
                @Override
                public void run() {
                    System.out.println("This␣is␣Thread"
                            + getNextNumber());
                }
            }.start();
        }
        System.out.println("End.");
    }
}
```

Bonusaufgabe 3.4: Funktionszeichner **4 Punkte**

Laden Sie sich das Archiv `FuncPainter.zip` von der Übungshomepage herunter. Vervollständigen Sie in der Klasse `FuncPainterImpl` die Methoden des `FuncPainter`-Interfaces. Alle Methoden erhalten als Parameter zuerst eine Oberfläche (`Screen`), anschließend eine Funktion (`Function`) und als letzten Parameter die Anzahl an `Threads`. Mit diesen Parametern soll jeweils die übergebene Funktion parallel ausgewertet und das Ergebnis grafisch dargestellt werden. Die Breite und die Höhe der Oberfläche lässt sich mit den Methoden `getWidth()` bzw. `getHeight()` ermitteln. Geben Sie die Datei `FuncPainterImpl.java` im EST ab.

a) In der `randomPaint`-Methode soll jeder erzeugte Thread zufällig Punkte auf der Oberfläche auswählen, die Funktion an dieser Stelle auswerten und der Oberfläche den erhaltenen Wert für diesen Punkt übergeben (siehe `setValue`-Hilfsmethode in `FuncPainterImpl`). Dies soll aber nur geschehen, wenn für den Punkt noch kein Wert gesetzt wurde. Wenn auf der Oberfläche genügend Punkte bearbeitet wurden (`finished() == true`), sollen sich die Threads beenden. Achten Sie darauf, dass die Punktauswahl der Threads wirklich zufällig ist und dass jeder Thread prinzipiell jeden Punkt der Oberfläche beschreiben könnte. Benutzen Sie keine Mechanismen zur Synchronisation.

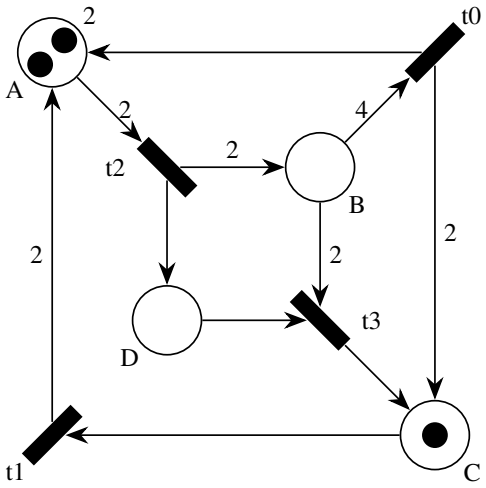
b) In der `synchronizedPaint`-Methode sollen die Threads wie in der `randomPaint`-Methode zufällig Punkte auswählen, die Funktion auswerten und das Ergebnis der Oberfläche übergeben. Der Unterschied besteht darin, dass bei den Threads in der `synchronizedPaint`-Variante der kritische Abschnitt synchronisiert werden soll, so dass bei jeder Ausführung alle Felder gefärbt werden.

c) In der `syncFreePaint`-Methode soll auf eine Synchronisation verzichtet werden. Trotzdem sollen bei jeder Ausführung alle Felder gefärbt sein, sobald `finished() == true` gilt. Es ist nicht mehr notwendig, dass die Threads die Punkte zufällig auswählen. Achten Sie auf eine möglichst gute Arbeitsaufteilung, eine parallele Ausführung und darauf, dass jeder Punkt nur einmal gefärbt wird (wenn Sie dies bei jeder Ausführung garantieren können, ist ein zusätzlicher Aufruf von `finished()` nicht mehr notwendig).

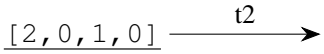
Hinweis: Achten Sie darauf, dass `FuncPainterImpl` einen parameterlosen Konstruktor besitzt.

Bonusaufgabe 3.5: Erreichbarkeitsgraph **2 Punkte**

Gegeben sei folgendes Petri-Netz:



Erstellen Sie den Erreichbarkeitsgraphen zu obigem Petri-Netz. Notieren Sie die Tokenzahlen der jeweiligen Markierungen im Schema `[A, B, C, D]`. Vermerken Sie an jeder Kante, welche Transition (t0...t3) gefeuert hat. Geben Sie Ihre Lösung in der Datei **pfp_abgabe_3.pdf** im EST ab.



6 Punkte