# Nebenläufigkeit

Wir haben die Nebenläufigkeit immer nach demselben Prinzip implementiert:

Für jede Funktionalität, die nebenläufig realisiert werden soll, gibt es eine eigene Klasse A. In dieser Klasse ist steht dann eine Unterklasse B, die das Interface Runnable implementiert. Zunächst muss ein Objekt der Klasse A erzeugt werden, auf welche dann die calculate-Methode aus A ausgeführt. In jener Methode – beziehungsweise in von ihr aufgerufenen privaten Methoden – werden dann die Instanzen von B erstellt und an den ExecuterService übergeben. Dann wird gewartet bis jene Runnables vollständig ausgeführt und der ExecuterService terminiert. So wird sichergestellt, dass der nächste Berechnungsschritt im Hauptprogramm erst ausgeführt wird, wenn der vorherige fertig ist. Die Kommunikation der Agenten läuft dann über von Java vorimplementierte Monitor-Datentypen wie ConcurrentHashMap oder es wird explizit mit ReentrantLock gelockt.

# Berechnung der schwachen Transitionen

Die Berechnung der schwachen Transitionen wird in der Klasse WeakLtsCalculator nebenläufig realisiert. Zunächst wird eine Instanz dieser Klasse erzeugt, worauf dann calculate() ausgeführt wird. Dort werden echt nacheinander die unsichtbaren (calculateInvisible) und sichtbaren (calculateVisible) Transitionen berechnet, wobei beide Schritte jeweils nebenläufig implementiert sind.

**InternalReachabilityChecker**

Für die Implementierung

# Partionierung in schwach bisimilare Äquivalenzklassen

# Entfernung redundanter Transitionen

# Rückgabe