Schwache Transitionen erstellen:  
  
a)  
Um alle schwachen Transitionen zu erhalten, iteriere ich über alle Zustände in meinem LTS, für jeden Zustand wird dann eine Funktion generateWeakTransitionForState aufgerufen, diese bekommt als Argumente den Startzustand, den aktuellen Zustand, einen Boolean Wert, der sich merkt ob schon eine starke Transition genutzt wurde, die genutzte starke Transition und die bereits besuchten Zustände.   
  
**public** **void** **generateWeakTransitionForState**(**State** start, **State** current, **Boolean** UsedStrong, **Action** StrongAction, **HashSet**<State> alreadyVisited)   
  
Der Aufruf erfolgt in unserer LTS-Klasse mit   
**this**.generateWeakTransitionForState(i,i, **false**, **null**, alreadyVisited);  
wobei i ein Status ist.  
  
Innerhalb der Funktion wird zunächst der aktuelle Zustand zur Menge der bereits besuchten Zustände hinzugefügt. Für diesen Zustand wird dann wieder über alle ausgehenden starken Transitionen iteriert. Hier muss zuerst getestet werden ob der Zielzustand noch nicht besucht wurde. Falls die Aktion der Transition intern ist, wird eine neue schwache Transition vom Anfangszustand zum Ziel der Transition erzeugt, falls bereits eine starke Aktion auf dem Weg genutzt wurde erhält die schwache Transition die genutzte starke Aktion, ansonsten eine interne Aktion.   
Falls die Aktion stark ist muss getestet werden ob bereits eine starke Aktion auf dem bisher zurückgelegten Weg genutzt wurde, nur wenn noch keine starke Aktion genutzt wurde, wird dann eine neue schwache Transition vom Anfangszustand aus mit der neuen, starken Aktion erstellt. Die jeweils erstelle schwache Transition wird in die Menge der schwachen Transitionen hinzugefügt. Zum Schluss wird dann die Funktion auf die Folgezustände wieder aufgerufen, der Startzustand ändert sich jedoch nicht. Die genutzen starken Aktionen werden entsprechend angepasst.  
  
b)  
Monitor und Nebenläufige Berechnung Wie wollen das nebenläufig zu bearbeitende Problem wie folgt lösen. Wir erstellen uns hierzu einen Monitor der mit impliziten Locks in Java arbeitet. Im Konstruktor des Monitors übergeben wir die schwache Transitionsrelation, die wir wie zuvor beschrieben berechnet wurde. Außerdem übergeben wir die Menge von Zuständen die partitioniert werden sollen. Diese Menge wird dann der Menge von Blöcken zugefügt, die die Partitionierung represäntiert. Außer diesen beiden Menge verwaltete der Monitor zusätzlich noch zwei Listen. Die erste Liste besteht aus Paaren von Blöcken die noch untersucht werden müssen. Die zweite Liste besteht aus Paaren von Blöcken die gerade untersucht werden. Versucht ein neuer Thread mit der Arbeit loszulegen und eine mögliche Blockzerlegung zu berechnen, so untersucht er beide Listen und nimmt aus der ersten Liste ein Paar, von dem keines der Elemente bereits in einem anderen Paar untersucht wird und in der Zweiten Liste steckt. Dies ermöglicht, dass beim Berechnen neuer Paare(hier muss der Thread das Lock auf die TODO-Liste halten) keine "falschen" Merges entstehen. Und nur eindeutige Quellen für die verschiedenen Blöcke vorhanden sind. Am Ende der Berechnung erstellt der Thread dann die neuen Blöcke und berechnet die Liste der zu bearbeitenden Paare neu. Dies geschieht als Monitormethode mit impliziten Locks(Kritischer Abschnitt,gesichert), dass kein neuer Thread gerade auf vielleicht irrelevante Paare zugreift und anfängt diese zu bearbeiten. Die Threads arbeiten so nebenläufig und Arbeiten die Liste der zu bearbeitenden Paare ab. Sind beide Listen leer, so sind alle Blöcke abgearbeitet und es gibt keine noch nicht untersuchten Blockpaare, Die Partitionierung hat somit den gröbsten Zustand erreicht und die Arbeit ist getan. Die Threads können beendet werden und das Erstellen des minimierten LTS kann begonnen werden.   
  
c) Um das beobachtunskongruente LTS zu dem ursprünglich gegebenen LTS zu erhalten wird zuerst für jeden Block der Paritionierung ein neuer Zustand erstellt. Danach wird aus jedem Block exemplarisch ein Zustand herausgenommen und es werden alle von diesem Zustand ausgehenden schwachen Transitionen untersucht. Für jede Transition die eine Kommunikationsaktion enthält wird in unserem neuen LTS eine Transition in den Block erstellt, die den Zielzustand enthält. Bei internen Aktionen muss diese Aktion nur dann erstellt werden, wenn das Ziel in einem anderen Block liegt, beziehungsweise wenn der Block den Startzustand enthält (wegen Beobachtungskongruenz).