|  |
| --- |
| Aufgabenblatt 4 |
|  |
| MEETING = (hello -> conserve -> goodbye -> STOP).  // \* hello -> converse -> goodbye |
| JOB = (arrive -> work -> leave -> JOB).  // \* arrive -> work -> leave -> arrive -> ... |
| GAME = (one -> WIN  |two -> WIN  |three -> LOSE),  WIN = (win -> GAME),  LOSE = (lose -> GAME).  // \* one -> win -> one  // \* one -> win -> two  // \* one -> win -> three -> lose -> one  // \* two -> win -> three -> lose -> two  // \* three -> lose -> one -> win -> three  // \* one -> win -> one -> win -> one  // \* three -> lose -> three -> lose -> three |
| DOUBLE = ( in[1] -> out[2] -> DOUBLE  | in[2] -> out[4] -> DOUBLE  | in[3] -> out[6] -> DOUBLE). |
| MOVE = (ahead -> AHEAD),  AHEAD = (left -> STOP | right -> MOVE).  // \* ahead -> left  // \* ahead -> right -> ahead -> left  // \* ahead -> right -> ahaed -> right -> ahead -> left  // \* ahead -> right -> ahead -> right -> |
| PERSON = WEEK,  WEEK = (weekday -> WEEKDAY|weekend -> WEEKEND),  WEEKDAY = (sleep -> work -> PERSON),  WEEKEND = (sleep -> (play -> PERSON|shop -> PERSON)).  // \* weekday -> sleep -> work -> weekday -> sleep -> work  // \* weekday -> sleep -> work -> weekend -> sleep -> play  // \* weekend -> sleep -> shop -> weekday -> sleep -> work  // \* weekend -> sleep -> play -> weekend -> sleep -> shop  // \* weekday -> sleep -> work -> weekend -> sleep -> play -> weekday -> sleep -> work |
| FOURTICK(N=4) = FOURTICK[0],  FOURTICK[i:0..N] = (when (i<N) tick -> FOURTICK[i+1]  |when (i==N) STOP).  // \* tick -> tick -> tick -> tick  // \* tick -> tick -> tick  // \* tick -> tick  // \* tick |
|  |
| range P = 0..9  SENSOR = SENSOR[5],  SENSOR[i:P] = (  when (i >= 0 && i <= 2) niedrig -> SENSOR[i]  |when (i >= 8 && i <= 9) hoch -> SENSOR[i]  |when (i >= 3 && i <= 7) normal -> SENSOR[i]  |pegel[j:P] -> SENSOR[j]  ). |
|  |
|  |
| const PRICE = 60  const MAX\_SUM = PRICE + 50  range N = 0..MAX\_SUM  GETRAENKEAUTOMAT = AUTOMAT[0],  AUTOMAT[s:N] = (  when (s < PRICE && s + 10 <= MAX\_SUM) einwurf10 -> AUTOMAT[s + 10]  |when (s < PRICE && s + 20 <= MAX\_SUM) einwurf20 -> AUTOMAT[s + 20]  |when (s < PRICE && s + 50 <= MAX\_SUM) einwurf50 -> AUTOMAT[s + 50]  |when (s >= PRICE) ausgabe -> WECHSELGELD[s]  ),  WECHSELGELD[s:N] = (  when (s > PRICE) wechselgeld[s - PRICE] -> AUTOMAT[0]  |when (s == PRICE) keinWechselgeld -> AUTOMAT[0]  ). |
|  |
|  |
| Bei S1 haben P und Q beide den Prozess b, und laut FSP müssen beide Prozesse b synchronisieren.  Mögliche Aktionsfolgen:  P führt a aus, Q führt c aus, dann synchronisieren sie b.   * a -> c -> b   Q führt c aus, P führt a aus, dann synchronisieren sie b   * c -> a -> b   Danach kehren beide zum Anfangszustand zurück. Somit sind mögliche Traces von S1:   * a -> c -> b -> … * c -> a -> b -> …   Bei S2 gibt es zwei Alternativen  Alternativ 1:   * a -> c -> b -> …   Alternativ 2:   * c -> a -> b -> …   Mögliche Traces von S2:   * a -> c -> b -> … * c -> a -> b -> ...   Das nochmal in Vergleich mit möglichen Traces von S1:   * a -> c -> b -> … * c -> a -> b -> …   Beide Prozesse haben identische Traces, somit zeigen beide dasselbe Verhalten. |
|  |
| range Produktion = 0..1  range Montage = 0..1  Fertigung = Fertigung[0][0][0],  Fertigung[prod1:Produktion][prod2:Produktion][montageStatus:Montage] = (  when(prod1 == 0) produziere1 -> Fertigung[1][prod2][montageStatus]  |when(prod2 == 0) produziere2 -> Fertigung[prod1][1][montageStatus]  |when(montageStatus == 0 && prod1 == 1 && prod2 == 1) fertig -> montage -> Fertigung[prod1][prod2][1]  |when(montageStatus == 1 && prod1 == 1 && prod2 == 1) montiert -> Fertigung[0][0][0]  ). |
|  |
|  |
| const N = 4 // Anzahl der internen Speicherzellen  range ID = 0..N-1 // ID  Speicher = (rein -> raus -> Speicher). // Prozess Speicher  ||SCHIEBER = forall [i:ID] a[i]:Speicher. // Parallele Komposition der Speicherzellen. Es werden vom Prozess Speicher vier Instanzen erzeugt, jeweils mit einem Label-Präfix a[i] versetzt.  SCHIEBEREGISTER = SCHIEBER  /{ rein/a[0].rein, // Äußere Eingabe mit der ersten Zelle verbinden  raus/a[N-1].raus, // Äußere Ausgabe mit der letzten Zelle verbinden  a[0].raus/a[1].rein, // Verbindung von Zelle 0 zu Zelle 1  a[1].raus/a[2].rein, // Verbindung von Zelle 1 zu Zelle 2  a[2].raus/a[3].rein // Verbindung von Zelle 2 zu Zelle 3  }.    // Jede Speicherzelle im Schieberegister ist eine Instanz des gegebenenen Prozesses Speicher, der das Verhalten einer einzelnen Speicherzelle modelliert.  // Durch die parallele Komposition von vier Speicherzellen haben wir ein Schieberegister mit vier internen Speicherzellen erstellt.  // Das Schieberegister besitzt nach außen nur die Aktionen rein als Eingabe und raus als Ausgabe wie in der Aufgabenstellung gefordert  // Die Daten werden intern von einer Zelle zur nächsten weiterverschoben, indem die raus-Aktion einer Zelle mit der rein-Aktion der nächsten Zelle verbunden wird.  // Das Modell kann durch ein Strukturdiagramm dargestellt werden, in dem die vier Speicherzellen in Reihe geschaltet sind  // und die Daten von der ersten zur letzten Zelle wandern.  Ich habe bei dieser Aufgabenstellung das Problem, dass ich keinen Prozess Schieber bekomme, sondern mehrere Prozesse Speicher. Ich habe vieles versucht, aber ich bekomme einfach nicht weiter. Gibt es hierzu eventuell eine Musterlösung? Können Sie vielleicht meinen Fehler erkennen? |
|  |
| Bei dieser Aufgabenstellung habe ich ein ähnliches Problem. Ich habe soweit ich weiß, zusammenhängenden Aktionen synchronisiert, aber einen Prozess ClientServer bekomme ich nicht, sondern mehrere Einzelprozesse |
|  |
| Hier habe ich dasselbe Problem |
|  |
| In geschweiften Klammern steht: {entsperrenAusgang, entsperrenEingang, sperrenAusgang, sperrenEingang}  Ich habe bei dieser Aufgabenstellung auch das Problem, dass ich keinen Prozess MUSEUM bekomme, sondern nur die Einzelprozesse. Ich habe versucht durch Relabelling dieselben Aktionen zu synchronisieren. Hier nochmal der Code per Text: const N = 5  range B = 0..N  EINGANG = (  entsperrenEingang -> eintritt -> sperrenEingang -> EINGANG  ).  AUSGANG = (  entsperrenAusgang -> austritt -> sperrenAusgang -> AUSGANG  ).  STEUERUNG = STEUERUNG[0],  STEUERUNG[b:B] = (  when(b < N) eintritt -> STEUERUNG[b + 1]  |when(b > 0) austritt -> STEUERUNG[b - 1]  |oeffnenSteuerung -> STEUERUNG[b]  |schliessenSteuerung -> STEUERUNG[b]  ).  WAERTER = (  oeffneEingang -> oeffnenSteuerung -> warteBesuchszeit -> schliesseEingang -> schliessenSteuerung -> WAERTER  ).  ||MUSEUM = (EINGANG || AUSGANG || STEUERUNG || WAERTER)  /{  entsperrenEingang/oeffnenSteuerung,  sperrenEingang/schliessenSteuerung,  entsperrenAusgang/oeffnenSteuerung,  sperrenAusgang/schliessenSteuerung,  eintritt/eintritt,  austritt/austritt  }. |