Mehr über Promela

Promela als Spezifikationssprache hat einige Eigenschaften, die gegenüber Programmiersprachen bemerkenswert sind:

- Nebenläufige Prozesse
  Promela erlaubt es, Prozesse mit **proctype** zu definieren. Jeder Prozess besitzt einen eigenen sequentiellen (nichtdeterministischen) Programmfluss, der mit **run** Prozesstypname in Gang gesetzt wird. Prozesse können über globale Variablen (und über Kanäle) kommunizieren.

- Ausführbarkeit von Anweisungen
  Promela–Anweisungen können *ausführbar* oder *blockiert* sein, d. h. die Programmausführung kann an einer Anweisung stoppen und erst nach Eintreten von bestimmten Bedingungen fortgesetzt werden. Boolesche Audrücke (Bedingungen) sind nur dann ausführbar, wenn sie wahr sind und blockieren, wenn sie falsch sind. Beispielsweise lässt sich

  ```
  if
  :: (a<10) -> printf("A␣ist␣kleiner␣als␣zehn.\n");
  fi
  ```

  daher auch kürzer `(a<10) -> printf("A␣ist␣kleiner␣als␣zehn.\n");` schreiben[1]. Tatsächlich ist `->` nur eine andere Notation für `;` — eingesetzt, um bedingte Ausführung zu verdeutlichen.

- Atomare Ausführung von Sequenzen
  Die Ausführung von Folgen von Anweisungen eines Prozesses kann durch die Ausführung von Anweisungen anderer Prozesse unterbrochen werden (*Interleaving*, *Scheduling*). Mit `atomic{`stmt1, stmt2,... `}` kann die atomare Ausführung einer Sequenz (ohne Interleaving) erzwungen werden.

- Zusicherungen
  Mit `assert(`Bedingung`)` kann sichergestellt werden, dass die angegebene Bedingung erfüllt ist. Dies wird z. B. eingesetzt, um in einem eigenen Check-Prozess auf das dauerhaft Einhalten gewünschter Systembedingungen zu prüfen.

2. **Dinierende Philosophen – rechte oder linke Gabel zuerst**
   In der letzten Übung wurde die Aufgabe

   > Bitte spezifizieren Sie das Problem der dinierenden Philosophen in Promela und überprüfen Sie, ob Ihr Modell
   >
   > (a) frei von Deadlocks ist und wenn nicht, lokalisieren Sie bitte den Fehler,
   >
   > (b) sicherstellt, dass niemals zwei Philosophen gleichzeitig die selben Gabeln nehmen.

   gestellt. Hier eine naive Modellierung der Philosophen, bei denen jeder Philosoph erst die linke Gabel nimmt und dann die rechte:

[1] `if` wird nur benötigt, falls es mehrere Alternativen gibt, aus denen (nichtdeterministisch) ausgewählt werden soll.

```
#define NUM_PHIL 4

int fork[ NUM_PHIL ];  // fork[i]=0: Gabel liegt;
                       // fork[i]=n >0: Gabel von n Philosophen genommen

proctype phil(int id) {
  do
  :: printf("Philo␣%d␣is␣thinking\n", id);

     fork[id]==0 -> fork[id]++;  // linke nehmen
     fork[(id+1)%NUM_PHIL]==0 -> fork[(id+1)%NUM_PHIL]++; // rechte nehmen

     printf("Philo␣%d␣is␣eating\n", id);

     fork[id]--;                          // linke ablegen
     fork[(id+1)%NUM_PHIL]--;             // rechte ablegen
  od
}

init {
  int i=0;
  do
  :: i>=NUM_PHIL -> break
  :: else -> run phil (i); i++;
  od
}
```

(a) Simulieren Sie bitte das Modell (`spin philo.pml`). Wie verhält es sich?

(b) Führen Sie bitte eine Überprüfung des Modells durch (`spin -a ...; gcc ...; pan -n`). Was bedeutet der Fehler `invalid end state`? Was ist der Unterschied zum Vorgehen aus 2a? Lokalisieren Sie bitte die Stelle des Problems (`spin -t philo.pml`).

(c) Ändern Sie bitte das Modell so ab, dass es zwei Arten von Philosophen gibt: Solche, die erst die linke Gabel und solche die erst die rechte Gabel nehmen. Sie sollen jeweils abwechselnd sitzen. Wiederholen Sie bitte die Überprüfung des Modells. Was stellen Sie fest?

(d) Formulieren Sie bitte eine Zusicherung, dass jede Gabel von höchsten einem Philosophen genommen wird und nehmen Sie sie bitte in Ihr Modell auf. Erfüllt Ihr Modell diese Zusicherung in allen Zuständen? Wenn nicht, was ist passiert?

(e) Korrigieren Sie bitte ggf. Ihr Modell und wiederholen Sie bitte das Modelchecking.

3. **Dinierende Philosophen — keine Gabel oder alle beide**
Eine andere Strategie für die Philosophen ist das Zurücklegen von Gabeln.

(a) Ändern Sie bitte wiederum das ursprüngliche Modell so ab, dass jeder Philosoph die linke (erste) Gabel wieder zurücklegt, falls es ihm nicht gelingt, die rechte (zweite) Gabel zu greifen. Verifizieren sie wiederum beide Eigenschaften (a) und (b) Ihres Systems.

(b) Wie sieht eine Promela-Spezifikation aus, bei der jeder Philosoph nicht-deterministisch die linke oder rechte Gabel nimmt und danach (mit Zurücklegen) die andere. Stellen Sie wiederum sicher, dass beide Eigenschaften in Ihrem Modell gelten.