

### (Besprechung 22.5)

Nutzen Sie die zur Verfügung gestellte SEU. Starten Sie StartSpin.bat. Beim ersten Speichern legen Sie das dann aktuell genutzte Arbeitsverzeichnis fest.

Hinweis: Fast alle Lösungen stehen auf der Web-Seite zum Buch, aber dies sind nur Beispiele, keine perfekten Muster. Entwickeln Sie Ihren eigenen Ansatz und arbeiten Sie Vor- und Nachteile Ihrer Lösungen gegenüber den Beispiellösungen heraus.

### Aufgabe 9

- a) Geben Sie folgende Spezifikation in ISPIN ein.

```
byte x = 0;

active proctype P1(){
    x = x + 1;
    x = x + 1
}

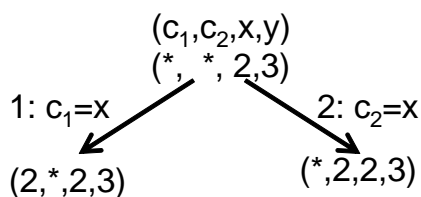
active proctype P2(){
    x = 2
}
```

Überprüfen Sie die Syntax, stellen Sie als Simulationsart „Interactive“ ein und starten Sie die Simulation. Versuchen Sie dann durch verschiedene von Hand gesteuerte Simulationen unterschiedliche Ergebnisse für x zu erreichen. Welche Ergebnisse sind möglich? (Um eine Simulation zu beenden, drücken Sie „Cancel“ im Fenster „Simulation Output“.)

- b) Schreiben Sie eine PROMELA-Spezifikation, mit der Sie feststellen können, wie sich der Datentyp byte bei einem Überlauf und bei einem Unterschreiten des Wertebereichs verhält.
- c) Nutzen Sie den Nichtdeterminismus, um in einer globalen Variable x eine zufällige ganze Zahl zwischen 0 und 100 zu generieren; dabei soll die Zahl nicht durch 10 teilbar sein. (Modulorechnung  $7 \% 3$  ist 1, mit 1 als Rest bei der Division, funktioniert auch in PROMELA.) Sie sollten jede erlaubte Zahl durch eine interaktive Simulation erreichen können.

### Aufgabe 10

Gegeben sei die Spezifikation auf der rechten Seite. Überlegen Sie sich mit einer Skizze, welche Werte x und y am Ende haben können. Der Anfang der Skizze könnte wie folgt aussehen; bedenken Sie, dass c jeweils eine lokale Variable ist.



Überprüfen Sie die möglichen Endergebnisse, indem Sie diese durch Simulation mit ISPIN erzeugen.

```
byte x;
byte y;

proctype P(){
    byte c = x;
    x = y;
    y = c
}

init{
    x = 2;
    y = 3;
    run P();
    run P()
}
```

### Aufgabe 11

Zu entwickeln ist eine Spezifikation mit N (PROZESSE) Prozessen, wobei jeder Prozess eine Boolesche Variable drucken enthält, diese sind in einem Array `bool drucken[N]`; global definiert. Jeder der Prozesse erhält zum Start eine eindeutige Identifikationsnummer, weiterhin sind alle Werte des Arrays `drucken` per Default auf `false` gesetzt. Jeder Prozess  $P_i$  möchte jetzt immer wieder drucken. Wenn er druckt, setzt er seine Variable `drucken[i]` auf `true`, nach dem Drucken auf `false`. Überlegen Sie sich ein Ihnen z. B. aus den verteilten Systemen bekanntes Verfahren ohne die Nutzung von `atomic` und `d_step`, so dass sichergestellt ist, dass immer nur ein Prozess druckt, es also maximal ein  $i$  gibt mit `drucken[i]==true`. Ein von Ihnen zu erweiternder Ausschnitt der Spezifikation sieht damit wie folgt aus:

```
#define PROZESSE 5
bool drucken[PROZESSE];

proctype P(byte id){
    /** was tolles **/
        drucken[id]=true;
        drucken[id]=false;
    /** was tolles **/
    od;
}

init{
    atomic{
        byte i=0;
        do
            :: i==PROZESSE-> break;
            :: i<PROZESSE ->
                run P(i);
                i=i+1;
        od;
    }
}
```

Nutzen Sie zur Verifikation folgenden Prozess, den Sie in Ihrer Spezifikation ergänzen.

```
active proctype Pruefung(){
    atomic{
        byte aktiv = 0;
        byte i = 0;
        do
            :: i < PROZESSE ->
                aktiv = aktiv + drucken[i];
                i = i + 1;
            :: else -> break;
        od;
        assert(aktiv <= 1);
    }
}
```

Da wir bis jetzt nur simulieren können, prüfen Sie nach der Einführung des Model Checkings, ob Ihre Spezifikation korrekt ist.