3. Übungsblatt zur Vorlesung “Formale Methoden im Software Entwurf”

TECHNISCHE UNIVERSITÄT DARMSTADT

Modellierung nebenläufiger Systeme (Shared Memory)
Diskussion der Aufgaben und Lösungen in den Tutorien: Kalenderwoche 46

Aufgabe 1 Nebenläufigkeit: Shared Variables

Schreiben Sie ein PROMELA Modell mit einem Prozess `p()` der eine globale Variable `b` erhöht, wie in Listing 1 gezeigt:

Listing 1: File: exercises/SharedVariables.pml

```
byte b = 0;

proctype p() {
  b++
}
```

Starten Sie zwei Instanzen des Prozesses `p`. Warten Sie bis beide terminiert sind (verwenden Sie dafür den in der Vorlesung vorgestellten „Join“-Trick) und überprüfen Sie, ob die Variable `b` dann den Wert 2 hat.

Ändern Sie die Implementierung von `p` wie folgt ab:

```
proctype p() {
  byte tmp = b;
  tmp++;
  b = tmp
}
```

Gilt die Zusicherung in diesem Fall? Falls nicht, lässt sich dies mit einem atomaren Block lösen?

Aufgabe 2 Büro Drucker

In einem Büro teilen sich zwei Rechner einen einzigen Drucker. Modellieren Sie diese Situation mit PROMELA und stellen Sie sicher, dass niemals beide Rechner gleichzeitig drucken. Verwenden Sie dazu eine globale Variable sowie atomare „Test & Set“Operationen. Verifizieren Sie, dass Ihre Lösung korrekt ist mit Hilfe von Spin.

Aufgabe 3 Mutual Exclusion

Es gibt eine Vielzahl von Mutual Exclusion Protokollen, die sicher stellen sollen, dass eine geteilte Ressource nicht gleichzeitig von mehreren Prozessen belegt werden kann.

Ein Nachteil, des in der Vorlesung vorgestellten Algorithmus ist, dass einem Prozess u.U. niemals ermöglicht wird, die Ressource zu nutzen (den kritischen Abschnitt zu betreten), sondern er immer von dem anderen Prozess überholt wird.

In dieser Aufgabe sehen wir uns zunächst zwei einfache Algorithmen an, die am Ende zum Algorithmus von Peterson [1] zusammengeführt werden. Letzterer Algorithmus ist aufgrund seiner Einfachheit recht bekannt.

Aufgabe 3.1 Einfache Algorithmen

Die PROMELA Modelle in Listing 2 und Listing 3 zeigen zwei einfache Mutual Exclusion Protokolle. Das erste Protokoll kommt sogar ohne ein Exit-Protokol aus, d.h. nach dem Verlassen des kritischen Abschnittes finden keine weiteren Aktionen, wie z.B. das Rücksetzen globaler Variablen, statt.

Listing 2: File: exercises/MutualExclusion1

```
byte turn = 0;

active proctype p() {
    turn = 1;
    turn == 2;
    // enter critical section
    // leave critical section
}

active proctype q() {
    turn = 2;
    turn == 1;
    // enter critical section
    // leave critical section
}
```

Listing 3: File: exercises/MutualExclusion2

```
bool q1 = false;
bool q2 = false;

active proctype p() {
  q1 = true;
  q2 == false;
  // enter critical section
  // leave critical section
  q1 = false;
}

active proctype q() {
  q2 = true;
  q1 == false;
  // enter critical section
  // leave critical section
  q2 = false;
}
```

Untersuchen Sie die Korrektheit der beiden Protokolle entlang der folgenden Teilaufgaben:

a) Sichern beide Protokolle zu, dass zwei Prozesse nicht gleichzeitig den kritischen Abschnitt betreten können? Falls nein, geben Sie bitte einen Gegenbeispiel/Fehlertrace[1] an (von Hand nicht von Spin erzeugt).

b) Welches Problem haben beide Protokolle (unabhängig davon, ob Sie Mutual Exclusion zusichern oder nicht)? Geben Sie jeweils entsprechende Fehlertraces an (von Hand und ohne Unterstützung durch Spin).

c) Ändert sich das Problem aus b), falls das Modell in Listing 2 so geändert wird, dass

- einer von beiden Prozessen
- beide Prozesse

wiederholt (unbegrenzt) den kritischen Abschnitt betreten wollen und dazu das Protokoll durchlaufen? Wie sieht es diesbzgl. mit Protokoll aus Listing 3 aus?

[1] Interleaving, das einen die Mutual Exclusion Eigenschaft verletzenden Simulationslauf beschreibt

d) Überprüfen/Beweisen Sie nun ihre vorherigen Antworten mit Hilfe von Spin. Falls notwendig führen Sie geeignete Ghost Variablen wie in der Vorlesung besprochen ein.

Aufgabe 3.2 Peterson's Algorithmus

Peterson's Algorithmus [1] ist eine Kombination beider Algorithmen aus Teilaufgabe Aufgabe 3.1. Ein PROMELA Modell des Algorithmus finden Sie in Listing 4.

Listing 4: File: exercises/Petersen2.pml

```
bool pIsWaiting = false;
bool qIsWaiting = false;
byte turn = 0;

active proctype p() {
 do ::
    atomic {
      pIsWaiting = true;
      turn = 2; // allow other process to proceed first
    }
    !qIsWaiting || turn == 1;
    // enter critical section
    // leave critical section
    pIsWaiting = false
 od
}

active proctype q() {
 do ::
    atomic {
      qIsWaiting = true;
      turn = 1; // allow other process to proceed first
    }
    !pIsWaiting || turn == 2;
    // enter critical section
    // leave critical section
    qIsWaiting = false
 od
}
```

a) Wie lange muss ein Prozess höchstens Warten bis er den kritischen Abschnitt betreten darf?

b) Verifizieren Sie mit Hilfe von Spin, ob der Algorithmus wie angegeben Mutual Exclusion zusichert.

c) Ist der **atomic** Block notwendig? Falls nein, ist die Reihenfolge der Zuweisungen im **atomic** Block von Bedeutung? Beantworten Sie die Frage zunächst ohne Toolunterstützung und geben Sie, falls passend, einen Trace für ein Interleaving an, welches zu einer Verletzung der Mutual Exclusion Eigenschaft oder zu einem anderen Problem führt. Überprüfen Sie dann ihre Antwort mit Hilfe von Spin.

Literatur

[1] Gary L. Peterson. Myths About the Mutual Exclusion Problem. *Inf. Process. Lett.*, 12(3):115–116, 1981.