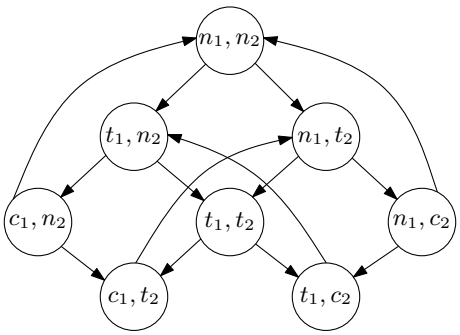
Prof. Martin Hofmann
Dr. Ulrich Schöpp

Ludwig-Maximilians-Universität München
Institut für Informatik
SS 2014

# Übungen zur Vorlesung
# **Formale Spezifikation und Verifikation**

Blatt 5

**Aufgabe 5-1** Betrachten Sie folgendes Transitionssystem:




Führen Sie den Labelling-Algorithmus aus der Vorlesung für folgende Formeln aus:

a) $\mathrm{AG}\,(t_1 \Rightarrow \mathrm{AF}\,c_1)$ (benutzen Sie hierbei sowohl den einfachen, als auch den effizienteren Algorithmus)

b) $\mathrm{EF}\,[c_1 \wedge \mathrm{E}[c_1\,\mathrm{U}\,(\neg c_1 \wedge E[\neg c_2\,\mathrm{U}\,c_1])]]$.

Geben Sie jeweils das mit Formeln beschriftete Transitionssystem an.

**Aufgabe 5-2** In dieser Aufgabe sollen zwei Beweise, die in der Vorlesung weggelassen wurden, nachgeholt werden.

a) Begründen Sie folgende Aussage: Ist $s_1 \to s_2 \to s_3 \to \ldots$ unendlicher Pfad in einem endlichen Graphen $G$, so existiert eine echte SCC in $G$, welche von $s_1$ aus erreichbar ist.

b) Sei $\mathcal{I}$ Interpretation mit Transitionssystem $(S, \to)$, $\phi, \psi$ Formeln und $s_1 \to s_2 \to s_3 \ldots$ unendlicher Pfad in $S$. Begründen Sie, dass die folgenden Aussagen äquivalent sind:

   - Es gibt keine Zahl $n \geq 1$, sodass $s_n \models_{\mathcal{I}} \psi$ und $s_i \models_{\mathcal{I}} \phi$ für alle $i < n$
   - Für alle $n \geq 1$ gilt $s_n \models_{\mathcal{I}} \neg\psi$, **oder** es gibt $m \geq 1$, sodass $s_m \models_{\mathcal{I}} \neg\phi \wedge \neg\psi$ und für alle $j < m$ gilt $s_j \models_{\mathcal{I}} \neg\psi$.

   Bitte $\phi$ "phi" und $\psi$ "psi" nicht verwechseln!

**Aufgabe 5-3** Für das Problem, zwei parallele Prozesse so zu synchronisieren, dass sie nicht beide gleichzeitig in einem kritischen Bereich sein können, gibt es viele Lösungen. Gegeben seien drei Lösungsvorschläge, deren Eigenschaften wir untersuchen wollen.

*Lösungsvorschlag 1.* Im ersten Vorschlag sollen die beiden parallel laufenden Prozesse durch eine Boolesche Variable `flag` synchronisiert werden, auf welche beide Prozesse zugreifen können. Die Idee ist, dass ein Prozess das Flag setzt bevor er den kritischen Bereich betritt und dass er den kritischen Bereich nur dann betritt, wenn das Flag nicht schon gesetzt.

Prozess $p_0$:

```
0: if flag then goto 0; else goto 1;
1: flag := true; goto 2;
2: /* kritischer Bereich */; goto 3
3: flag := false; goto 4;
4: if done[0] then goto 5; else goto 0;
5: goto 5; // fertig
```

Prozess $p_1$:

```
0: if flag then goto 0; else goto 1;
1: flag := true; goto 2;
2: /* kritischer Bereich */; goto 3
3: flag := false; goto 4;
4: if done[1] then goto 5; else goto 0;
5: goto 5; // fertig
```

Hierbei sind `done[0]` und `done[1]` zwei Boolesche Variablen, die angeben, ob das jeweilige Programm den kritischen Bereich noch einmal betreten will. Ihr Wert kann sich zum Beispiel im kritischen Bereich ändern. Wie er sich ändert, ist aber nicht bekannt.

Die Variablen `flag`, `done[0]` und `done[1]` sollen anfangs den Wert `false` haben und die beiden Prozesse $p_0$ und $p_1$ beginnen beide mit der Zeile 0.

*Lösungsvorschlag 2.* Im einem zweiten Ansatz sollen die beiden Programme durch eine Variable `turn` synchronisiert werden. Diese kann die Werte 0 und 1 annehmen und gibt an, welcher Prozess als nächstes den kritischen Bereich betreten darf.

Prozess $p_0$:

```
0: if turn=1 then goto 0; else goto 1;
1: /* kritischer Bereich */; goto 2
2: turn := 1-turn; goto 3;
3: if done[0] then goto 4; else goto 0;
4: goto 4; // fertig
```

Prozess $p_1$:

```
0: if turn=0 then goto 0; else goto 1;
1: /* kritischer Bereich */; goto 2
2: turn := 1-turn; goto 3;
3: if done[1] then goto 4; else goto 0;
4: goto 4; // fertig
```

Die Variable `turn` hat anfangs den Wert 0 und `done[0]` und `done[1]` haben anfangs den Wert `false`. Beide Prozesse $p_0$ und $p_1$ beginnen wieder in Zeile 0.

*Lösungsvorschlag 3.* Im letzten Vorschlag werden sowohl Flags als auch eine Variable `turn` benutzt.

Prozess $p_0$:

```
0: flag[0] := true; goto 1;
1: if flag[1] then goto 2; else goto 6;
2: if turn = 0 then goto 1; else goto 3;
3: flag[0] := false; goto 4;
4: if turn = 0 then goto 5; else goto 4;
5: flag[0] := true; goto 2;
6: /* kritischer Bereich */; goto 7
7: turn := 1; goto 8;
8: flag[0] := false; goto 9;
9: if done[0] then goto 10; else goto 0;
10: goto 10; // fertig
```

Prozess $p_1$:

```
0: flag[1] := true; goto 1;
1: if flag[0] then goto 2; else goto 6;
2: if turn = 1 then goto 1; else goto 3;
3: flag[1] := false; goto 4;
4: if turn = 1 then goto 5; else goto 4;
5: flag[1] := true; goto 2;
6: /* kritischer Bereich */; goto 7
7: turn := 0; goto 8;
8: flag[1] := false; goto 9;
9: if done[1] then goto 10; else goto 0;
10: goto 10; // fertig
```

Mit der `flag`-Variable zeigen die Prozesse an, dass sie den kritischen Bereich betreten wollen. Die Variable `turn` gibt an, welcher Prozess dabei Priorität hat. Möchten beide Prozesse den kritischen Bereich betreten, so zieht der Prozess, der gerade nicht die Priorität hat, seine Absicht zurück und wartet so lange, bis er Priorität hat.

Die Variable `turn` hat anfangs wieder den Wert 0, die Variablen `flag[0]`, `flag[1]`, `done[0]` und `done[1]` haben anfangs alle den Wert `false`. Beide Prozesse $p_0$ und $p_1$ beginnen mit der Zeile 0.

Untersuchen Sie mit NuSMV, welche der folgenden drei Eigenschaften die jeweiligen Lösungen haben:

a) Safety: Es kann nicht passieren, dass zwei Prozesse gleichzeitig im kritischen Bereich sind.

b) Liveness: Wenn ein Prozess die Zeile 0 erreicht, dann wird er auch auf jeden Fall irgendwann den kritischen Bereich betreten dürfen.

c) Es ist immer möglich, dass beide Prozesse ihre Berechnung beenden, d.h. dass sie beide ihre letzte Zeile erreichen können.

Bei allen Lösungsvorschlägen können sie annehmen, dass jeder Prozess den kritischen Bereich gleich nach dem Betreten wieder verlässt. Erreicht zum Beispiel $p_0$ im ersten Lösungsvorschlag die Zeile 2, so geht er im nächsten Schritt direkt zu Zeile 3. Weiterhin soll das Scheduling der Prozesse fair sein, entsprechend der Annahme "`JUSTICE running`" im Peterson-Beispiel aus der Vorlesung.

Gesucht sind NuSMV-Dateien, in denen die obigen drei Eigenschaften der drei Lösungsvorschläge überprüft werden. Für Eigenschaften, die nicht gelten, soll außerdem noch informell erklärt werden, warum sie nicht gelten.