Übungen Logik und formale Methoden – Temporale Logik

[Viele der Übungen sind von Bodo Igler.]

A Aussagen in LTL

1. Aussagen finden

ϕ und ψ seien Formeln der LTL. Geben Sie Formeln für folgende Aussagen an:

(a) „Wenn ϕ das nächste Mal gilt, dann gilt danach nie wieder ψ“

(b) „ϕ gilt genau einmal“

(c) „ϕ gilt mindestens zweimal“

(d) „ψ wird niemals eintreten“

(e) „ϕ wird unendlich oft eintreten“

B Semantik der LTL

2. Semantik von Formeln

Seien ϕ, ψ Formeln der LTL. Beweisen oder widerlegen Sie folgende Aussagen:

(a) Es sei ϕ allgemeingültig. Dann gilt: $\vDash \Box\,\phi$, $\vDash \Diamond\,\phi$ und $\vDash \bigcirc\,\phi$.

(b) $\vDash \Diamond\ \Box\ \phi \wedge \neg\ \Box\ \Diamond\,\phi$

(c) $\vDash (\Box\,\phi \rightarrow \Box\,\psi) \rightarrow \Box\,(\phi \rightarrow \psi)$

(d) $\neg \bigcirc\ \phi \equiv \bigcirc\,\neg\phi$

C Spin

3. Spin-Model-Checker

Machen Sie sich mit dem Spin-Model-Checker (SMC) vertraut:

(a) Installieren Sie SMC. Download SMC: https://github.com/nimble-code/Spin

(b) Führen Sie mit SMC mehrere Simulationsläufe für das Programm in Listing 1 durch.

(c) Provozieren Sie im manuellen Modus des SMC folgende Situation für das Programm in Listing 1: x == 10.

(d) Verifizieren Sie mit Hilfe des SMC für das Programm in Listing 1 folgende Aussagen:

- $\Box\,(x >= 0)$ (Stets gilt: x >= 9.)
- $\Diamond\,(x == 10)$ (Früher oder später gilt ein Mal: x == 10.)
- Deadlock ist möglich.
- $\Box\,\Diamond\,(x == 10)$ (Unendlich oft gilt x == 10.)

Listing 1: PROMELA-Programm

```
byte x=9;

active proctype dec() {
```

```
  do
  ::  x>10 -> x--;
  ::  else -> x++;
  od;
}
active proctype inc() {
  do
  ::  x<10 -> x++;
  ::  else -> x--;
  od;
}
active proctype check() {
  if
  ::  x< 9  -> printf("x<9\n");
  fi;
}
```

4. Lineare temporale Logik

Betrachten Sie den Automat in Abbildung 1. Welche der folgenden Aussagen sind wahr, welche falsch? Begründung!

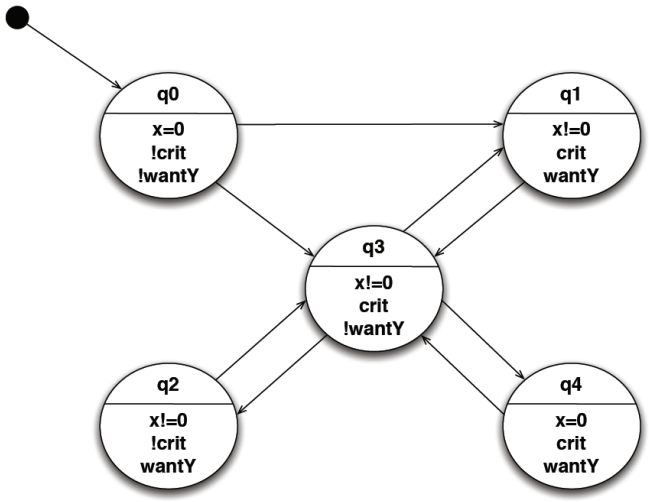

Abbildung 1: Automat

(a) $\Diamond \Box$ wantY

(b) $\Box \Diamond$ wantY

(c) $\bigcirc$!wantY $\Rightarrow \bigcirc\bigcirc$ wantY

(d) $\Box (x = 0)$

(e) (x=0) U $\Box$ (crit $\vee$ wantY)

(f) ($\bigcirc\bigcirc$ crit) U (crit $\vee$ wantY)

(g) $\Box \Diamond$ q2

(h) $\Box \Diamond$ q3

5. Lineare temporale Logik mit Spin

Überprüfen Sie die Aussagen (a), (b), (d), (e), (g) und (h) aus der vorherigen Aufgabe mit Hilfe des SMC. Übersetzen Sie dazu zuerst den Automaten in Abbildung 1 in PROMELA. Als Ausgangspunkt für diese Übersetzung eignet sich Listing 2

Listing 2: PROMELA-Fragment zu Abbildung 1

```
byte x;
book crit, wantY;

active proctype automat() {
q0:
  x=0;
  crit=false;
  wantY=false;
  if
  :: true -> goto q3;
  :: true -> goto q1;
  fi;
q1:
  ...
}
```

6. Guarded Statements

Beantworten ohne Verwendung des SMC: Was gibt das Programm in Listing 3 aus?

Listing 3: PROMELA-Programm

```
byte x=9;

active proctype P() {
  if
  :: x>10 -> printf("x>10\n");
  fi;
  printf("P(): Ende\n");
}
```

7. Interleaving

Ergänzen Sie den Code in Listing 4, so dass Q folgende Eigenschaften erfüllt:

- Q terminiert garantiert, wenn `P` die vorgestellte Form hat.
- Q terminiert möglicherweise nicht, wenn man `atomic` in `P` weglässt.

Listing 4: Prozess P

```
byte x=0;
active proctype P() {
  atomic {
    x=x+1;
    x=x+1;
  }
}
active proctype Q() {
  ...
}
```

8. Allgemeingültige Aussagen der Aussagenlogik

Das Programm in Listing 5 überprüft nach Zufallsprinzip mit Stichproben für p und q die Gültigkeit der De Morganschen Gesetze.

Überprüfen Sie mit Hilfe des SMC, ob die De Morganschen Gesetze für alle möglichen Belegungen von p und q gelten. Passen Sie dazu das Programm aus Listing 5 geeignet an.

Listing 5: De Morgansche Gesetze

```
active proctype de_morgan() {
  bit p,q;
  if
  :: true -> p=0;
  :: true -> p=1;
  fi;
  if
  :: true -> q=0:
  :: true -> q=1;
  fi;
  printf("p=%d, q=%d\n",p,q);
  printf("!(p&&q) == (!p||!q): %d\n",!(p&&q) == (!p||!q));
  printf("!(p||q) == (!p&&!q): %d\n",!(p||q) == (!p&&!q));
}
```

9. Allgemeingültige Aussagen der linearen temporalen Logik

Überprüfen Sie mit Hilfe des SMC die Gültigkeit folgender LTL-Formeln:

(a) Äquivalenzen bei Negation:

- $(\Box p) \leftrightarrow \neg(\Box \neg p)$
- $(\Box p) \leftrightarrow \neg(\Diamond \neg p)$
- $(\Diamond p) \leftrightarrow \neg(\Diamond \neg p)$
- $(\Diamond p) \leftrightarrow \neg(\Box \neg p)$

(b) Äquivalenzen bei der Verknüpfung temporaler Operatoren:

- $(\Box \Box p) \leftrightarrow (\Box p)$
- $(\Diamond \Diamond p) \leftrightarrow (\Diamond p)$
- $(\Box \Diamond p) \leftrightarrow (\Diamond \Box p)$
- $(\Diamond \Box \Diamond p) \leftrightarrow (\Diamond \Box p)$
- $(\Diamond \Box \Diamond p) \leftrightarrow (\Box \Diamond p)$
- $(\Box \Diamond \Box p) \leftrightarrow (\Box \Diamond p)$
- $(\Box \Diamond \Box p) \leftrightarrow (\Diamond \Box p)$

(c) Äquivalenzen bei der Verknüpfung temporaler/nicht-temporaler Operatoren:

- $(\Box p \wedge \Box q) \leftrightarrow \Box (p \wedge q)$
- $(\Box p \vee \Box q) \leftrightarrow \Box (p \vee q)$
- $(\Diamond p \wedge \Diamond q) \leftrightarrow \Diamond (p \wedge q)$
- $(\Diamond p \vee \Diamond q) \leftrightarrow \Diamond (p \vee q)$

(d) Implikationen:

- $(\Box p) \rightarrow p$
- $p \rightarrow (\Box p)$

- $(\Diamond\, p) \rightarrow p$
- $p \rightarrow (\Diamond\, p)$
- $(\Box\, (p \rightarrow q)) \rightarrow ((\Box\, p) \rightarrow (\Box\, q))$

(e) Äquivalenzen mit Until:

- $(\Diamond\, p) \leftrightarrow (trueUp)$
- $(pUq) \leftrightarrow (\neg(\neg qU(\neg p \wedge \neg q)) \wedge (\Diamond\, q))$

10. Nichtdeterminismuns I

Betrachten Sie Listing 6. Das Makro `zufallsZahl(z,min,max)` setzt `z` auf einen zufälligen Wert im Bereich von `min` bis `max`. Dabei gilt folgende Einschränkung:

```
min, max ∈ {0,1,2,3,4,5,6,7,8,9}
```

Änderen Sie `zufallsZahl(z,min,max)` so ab, dass dieses Makro für beliebig ganze Zahlen `min, max` funktioniert. Die neue Version dieses Makros darf nicht mehr als 10 Statements umfassen.

Listing 6: Zufallszahlen generieren

```
inline zufallsZahl(z,min,max) {
  if
  ::  min<=0 && 0<=max -> z=0;
  ::  min<=1 && 1<=max -> z=1;
  ::  min<=2 && 2<=max -> z=2;
  ::  min<=3 && 3<=max -> z=3;
  ::  min<=4 && 4<=max -> z=4;
  ::  min<=5 && 5<=max -> z=5;
  ::  min<=6 && 6<=max -> z=6;
  ::  min<=7 && 7<=max -> z=7;
  ::  min<=8 && 8<=max -> z=8;
  ::  min<=9 && 9<=max -> z=9;
  fi;
}

active proctype f() {
  byte x;
  zufallsZahl(x,0,9);
  printf("Zufallszahl zwischen 0 und 9: %d\n",x);
  zufallsZahl(x,3,5);
  printf("Zufallszahl zwischen 3 und 5: %d\n",x);
}
```

11. Nichtdeterminismus II

Implementieren Sie in PROMELA ein Programm, das in zufälliger Reihenfolge alle Zahlen in einem vorgegebenen Zahlenbereich ausgibt.

12. Nichtdeterminismus & Kontrollstrukturen

Implementieren Sie ausgehend von Listing 7 einen Sortieralgorithmus und überprüfen Sie dessen Korrektheit. Gehen Sie dazu schrittweise vor:

(a) Implementieren Sie den Sortieralgorithmus in der Prozedur `init` an der entsprechenden Stelle.

(b) Implementieren Sie den Algorithmus zur Überprüfung des Sortierergebnis an der entsprechenden Stelle. Es wird überprüft:

- a enthält noch die gleichen Werte wie vorher
- a ist aufsteigend sortiert

(c) Veränderen Sie die Prozedur `init` mit Hilfe des Ergebnisses der Übung „Nichtdeterminismus I“ so, dass die ersten `n` Elemente von `a` mit einem zufälligen Wert von -5 bis $+5$ befüllt werden. `n` soll dabei ein zufälliger Wert von 0 bis `N` sein.

(d) Überprüfeb Sie, ob für alle $n \leq 5 = N$ korrekt sortiert wird.

Listing 7: Sortier-Algorithmus

```
#define N 5
init {
  byte n;
  short a[N];
  byte i;

  n=N;
  // a initialisieren
  a[0]=3;
  a[1]=24;
  a[2]=-2;
  a[3]=7;
  a[4]=-6;

  // a aufsteigend sortieren
  // TODO

  // a ausdrucken
  i=0;
  do
  ::  i<n -> printf("a[%d]: %d\n,i,a[i]); i++;
  ::  else -> break;
  od;

  // a berprfen
  // TODO
}
```

13. Grundlegende Fragestellungen bei Nebenläufigkeiten

Bart und Lisa nehmen meistens den Bus, um zur Schule zu fahren. An manchen Tagen möchte aber auch einer von beiden mit dem Fahrrad zur Schule fahren. Sie haben jedoch nur ein gemeinsames Fahrrad. Wenn Bart und/oder Lisa der Wunsch befällt, mit dem Fahrrad zur Schule zu fahren, dann bleibt dieser Wunsch solange bestehen, bis er erfüllt wurde.

Bezogen auf die Fahrradbenutzung, kann jeder von beiden also drei Zustände einnehmen:

- wunschlos
- möchte Fahrrad benutzen
- benutzt das Fahrrad

Nun zu den Aufgaben:

(a) Stellen Sie die folgenden Aussagen als LTL-Formeln dar:
- Das Fahrrad kann nicht gleichzeitig von Bart und Lisa benutzt werden.
- Das Fahrrad wird immer eine endliche Anzahl von Tagen von einem benutzt.
- Jeder der beiden kann immer mal wieder vom Wunsch befallen werden.
- Die Fahrradbenutzung erfolgt immer abwechselnd.
- Wenn einen der beiden der Wunsch befällt, dann wird dieser irgendwann mal auch erfüllt.

(b) Ersinnen Sie einen Algorithmus, der die Aussagen aus (a) erfüllt (bzw. erfüllen soll).

(c) Versuchen Sie im interaktiven Modus mindestens eine der Aussagen aus (a) zu widerlegen.

(d) Überprüfen Sie nun durch Model Checking die Aussagen aus (a).

(e) Verbessern Sie ggfs. den Algorithmus.

14. Internet-Auktion

Ein Internet-Auktionshaus bietet Auktionen nach folgenden Regeln an: Eine Auktion ist zeitlich begrenzt. Mehrere Bieter können an einer Autkion teilnehmen. Solange die Auktion läuft, kann jeder Bieter das aktuelle Gebot abfragen und ein Gebot abgeben. Die Abgabe einen neuen Gebots muss nicht direkt auf die Abfrage des aktuellen Gebots erfolgen. Es werden nur Gebote akzeptiert, die zum Zeitpunkt des Eingangs des Gebots beim Internet-Auktionshaus um einen Betrag höher sind als das gerade gültige Gebot. Das beim Ende der Auktion gültige Höchstgebot gewinnt.

Es gibt drei Arten von Bietern:

Typ 1:

Hält das aktuelle Gebot im Auge und entscheidet nach dem Zufallsprinzip, ob er ein Gebot abgeben möchte und wie hoch dieses Gebot ausfällt.

Typ 2:

Entscheidet im Vorhinein, wie viel sie zu bieten bereit ist und um welchen Betrag das jeweilis gültige Gebot überboten werden soll. Hält das aktuelle Gebot im Auge und beitet nach der vorher festgelegten Strategie mit.

Typ 3:

Entscheidet im Vorhinein, wie viel sie zu bieten bereit ist. Bietet nur einmal mit und zwar gegen Ende der Auktion mit dem persönlichen Höchstgebot.

Typ 4:

Entscheidet sich zu Beginn der Auktion für einen der Typen 1,2 oder 3.

Analysieren Sie die Auktionsregeln und die Bietertypen mit Hilfe von Spin:

(a) Modellieren Sie die Auktionsregeln und den Bietertyp 1.

(b) Überprüfen Sie mit Hilfe von Spin, ob das Modell für sich genommen brauchbar ist. Beantworten Sie dazu z.B. folgende Frage:

„Kann es zu einem Deadlock kommen?“

Verbesseren Sie das Modell ggfs.

(c) Modellieren Sie nun der Reihe nach die restlichen Bietertypen.

(d) Überprüfen Sie das Modell erneut mit Hilfe von Spin und verbessern Sie es ggfs.

(e) Überprüfen mit Hilfe von Spin, ob es zu unerwünschten Effekten für das Auktionshaus oder einen Bieter kommen kann. Beantworten Sie z.B. folgende Frage:

„Bekommt stets der Bieter den Zuschlag, der bereit was, am meisten zu bieten?“

(f) Veränderen die Regeln im Hinblick auf mindestens einen unerwünschten Effekt, der sich in (e) ergeben hat.

15. Drei gewinnt

Betrachten Sie das Spiel „Drei gewinnt“[1].

(a) Modellieren den Verlauf von Partien des Spiels „Drei gewinnt“ in PROMELA.

(b) Überprüfen Sie mit Hilfe des SMC, ob eine Partie „Drei gewinnt“ mit einem, „unentschieden“ enden:

- muss
- kann

(c) Sowohl für den weißen als auch für den schwarzen Spieler gibt es eine Strategie, mit der er sicherstellen kann, dass er nicht verliert.

- Formulieren Sie für den schwarzen Spieler eine solche Strategie in PROMELA.
- Überprüfen, ob die Strategie die gewünschte Eigenschaft hat.
- Lassen Sie SMC herausfinden, ob es eine solche Strategie auch für den weißen Spieler gibt.

[1] Anderer bekannter Name für das Spiel ist „TicTacToe“.