Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Gidon Ernst

Erste Probeklausur
Formale Spezifikation und Verifikation

München, 20. Februar 2024

Diese Klausur soll entwertet werden: □

Hinweise:

- Notieren Sie Ihre Lösung jeweils unter der Aufgabe auf dem **jeweiligen Aufgabenblatt**.
  Nutzen Sie gegebenenfalls auch die **Rückseite**.
- Geben Sie auch alle **unbearbeiteten Aufgabenblätter** ab.
- Verwenden Sie jedes Blatt inklusive Rückseite nur für **die jeweilige Aufgabe**.

Zusatzblätter:

- Ein **Zusatzblatt** finden Sie am Ende der Klausur, weitere Blätter erhalten Sie auf Nachfrage.
- Schreiben Sie auf jedes Zusatzblatt **Ihren Namen**, **Ihre Matrikelnummer** und **die Nummer der Aufgabe**.
- Die Verwendung eigener Blätter ist verboten.
- Verwenden Sie jedes Zusatzblatt nur für **eine Aufgabe**.

Sonstiges:

- Schalten Sie Ihr Mobiltelefon aus! Ausnahme: Nutzung der Corona-App wenn das Mobiltelefon in Ihrem Rucksack/Jacke in der Bank vor Ihnen verstaut ist und Ton+Vibration ausgeschaltet sind.
- Halten Sie Ihren Studierendenausweis Ausweis bereit.
- Schreiben Sie **leserlich** und mit einem blauen oder schwarzen Stift.
- Geben Sie **nur eine Lösung je Aufgabe** ab!
  Falls Sie mehr als eine Lösung einreichen, wird die **schlechteste** bewertet.
- Sie haben 90 Minuten Zeit.
- Es sind 90 Punkte erreichbar.
- Während der Klausur dürfen Sie keine Hilfsmittel verwenden.

Viel Erfolg!

Wird von den Betreuern ausgefüllt

Anzahl Zusatzblätter: ___

Ausweiskontrolle:

Aufgabe 1: Theorie & Testen (7 Punkte)

a) Nennen Sie je ein Problem von Testen und formaler Verifikation, welches durch die jeweilige andere Technik behoben oder vermieden werden kann. Begründen Sie dies jeweils kurz. (3 Punkte)

Testen: konkrete Werte → echte Fehler, einfach umzusetzen, aber nur einige Pfade, schwer/unmöglich alles zu testen. Formale Verifikation: mathematischer Beweis, prüft komplettes Programm. Weit aufwendiger als testen.

b) Erklären Sie, was eine Lebendigkeitseigenschaft ist. Geben Sie ein Beispiel für eine solche Eigenschaft für eine Ampelsteuerung an. (2 Punkte)

Etwas Gutes wird passieren oder hat nur unendliche Gegenbeispiele. Ampel muss immer wieder zwischen grün und rot wechseln

c) Nennen Sie ein Beispiel für einen Softwarebereich, in dem formale Verifikation auf jeden Fall erfolgen sollte! (1 Punkte)

Sicherheitskritische oder ganz allgemein solche Software, bei der ein Versagen mit hohen Konsequenzen verbunden ist. Also z.B. Software für Mars-Lander etc., Flugzeuge, Herzschrittmacher, autonomes Fahren, . . .

d) Nennen Sie eine Besonderheit von eingebetteten Systemen (z.B. Cyber-Physical Systems) im Gegensatz zu klassischen Softwaresystemen? (1 Punkte)

technisches System <-> Umwelt, Kommunikation, Zeit ist wichtig, kontinuierliche Zeit/Werte . . .

Aufgabe 2: Testen (4 Punkte)

a) Geben Sie 4 Testfälle (Werte für Parameter und das erwartete Ergebnis) an, die die folgende Methode möglichst **umfassend und genau** testen. Jeder Test soll in der Programmiersprache Java kompilierbar sein. (2 Punkte)

```java
/** Gibt den umgekehrten String zurueck, also einen String,
*   bei dem das erste Zeichen dem letzten Zeichen des
*   Eingabestrings s entspricht usw.
*   Wird null uebergeben, so ist der Rueckgabewert ebenfalls null.*/
public String reverse(String s) { ... }
```

Testfälle:

s	Rückgabewert

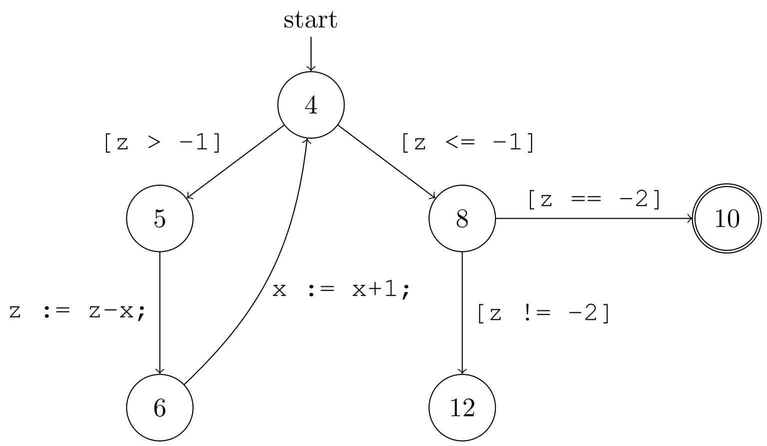
Lösung: Ein Testfall für null muss vorhanden sein
sowie 3 beliebige Testfälle die folgende Fälle abdecken:
leerer String, Sonderzeichen, gerade und ungerade Länge des Strings, String aus nur einem Zeichen, String mit Groß-und Kleinschreibung.
Negativ bewertet werden inkorrekte Typen wie ints.

b) Nennen Sie eine Technik, mit der Sie eine Vielzahl an Testfällen, z.B. für die vorangegangen Funktion aus a), generieren können! Nennen Sie zudem je einen Vor- und Nachteil dieser Technik im Vergleich zum klassischen Testen. Welche dieser Techniken sollte in der Praxis benutzt werden? (2 Punkte)

Lösung: Property-Based Testing
Vorteile: hohe Zahl an Testfällen, einfacher zu Lesen, erlaubt Wiederverwendung existierender Input-Generatoren, etc.
Nachteile: Manchmal umständlich zu schreiben (insb. Generatoren), kann trotzdem nicht alles abdecken, ressourcenaufwendig, keine Garantie dass Randfälle auch abgedeckt sind

Aufgabe 3: Explizite Erreichbarkeitsanalyse (14 Punkte)

```
1  int x = 1;
2  int z = nondet_bool();
3  int main() {
4    while (z > -1) {
5      z = z - x;
6      x = x + 1;
7    }
8    if (z == -2) {
9      ERROR:
10       return 1;
11   }
12   return 0;
13 }
```



Es ist ein Programm als C-Code (links) sowie der dazu äquivalente Kontrollflussautomat (rechts) gegeben. Wir spezifizieren, dass kein Fehlerzustand, hier die Programmzeile 10, erreicht werden darf.

Die im Programm enthaltene Funktion `nondet_bool` gibt nicht-deterministisch entweder den Wert 0 oder 1 zurück. Dementsprechend ergeben sich die zwei Startzustände, die in der Tabelle unten bereits vorgegeben sind.

Die Knoten des Kontrollflussautomaten geben den jeweiligen Wert des Programmcounters *pc* an. Bedingungen an Kanten sind durch eckige Klammern gekennzeichnet. Orientieren Sie sich am angegebenen Kontrollflussautomaten, insbesondere den dort angegebenen Werten für den Programmcounter `pc`.

a) Führen Sie für das gegebene Programm für jeden der Startzustände je eine explizite Erreichbarkeitsanalyse durch und tragen Sie die erreichbaren Zustände in die jeweilige Tabelle ein. (13 Punkte)

Hinweis: *Die vorgedruckten Tabellen enthalten mehr Zeilen, als Sie für diese Aufgabe benötigen. Dies dient lediglich dazu, dass Sie einzelne Zeilen durchstreichen können, falls Sie sich verrechnet haben.*

$pc \mapsto$	$x \mapsto$	$z \mapsto$
4	1	0
5	1	0
6	1	-1
4	2	-1
8	2	-1
12	2	-1

$pc \mapsto$	$x \mapsto$	$z \mapsto$
4	1	1
5	1	1
6	1	0
4	2	0
5	2	0
6	2	-2
4	3	-2
8	3	-2
10	3	-2

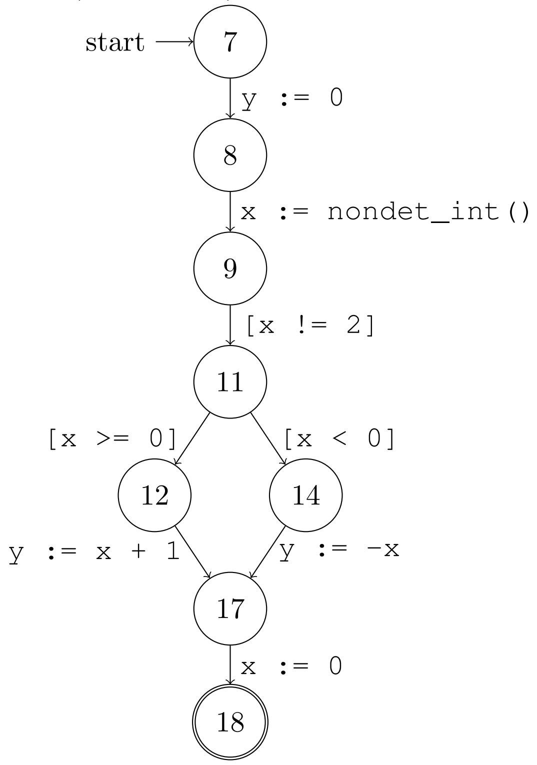
b) Wie können Sie nach der expliziten Erreichbarkeitsanalyse ablesen ob ein Programm sicher ist und ist das gegebene Programm sicher?

(1 Punkte)

Lösung: Wenn Fehlerzustand erreichbar ist es unsicher.
Unsicher, da Zeile 10 erreichbar.

Aufgabe 4: Symbolische Erreichbarkeitsanalyse (12 Punkte)

```
1   extern int nondet_int();
2
3   int x;
4   int y;
5
6   void main() {
7     y = 0;
8     x = nondet_int();
9     assume(x != 2);
10
11    if(x >= 0) {
12        y = x + 1;
13    } else {
14        y = -x;
15    }
16
17    x = 0;
18  }
```



Es ist ein Programm als C-Code (links) und als dazu äquivalentem Kontrollflussautomat (rechts) gegeben, analog zur Aufgabe 2. Der Wertebereich von **int** die Menge der ganzen Zahlen ist, es treten keine arithmetischen Überläufe auf. Für die Funktion `nondet_int` gibt einen nichtdeterministischen Wert aus dem Wertebereich von **int** zurück.

- Führen Sie die symbolische Erreichbarkeitsanalyse durch: Bestimmen Sie für die unten angegebenen Werte des Programmzählers `pc` eine Menge an Formeln über den Programmvariablen `x` und `y`, deren Disjunktion exakt angibt, welche Zustände dort jeweils erreichbar sind.

Orientieren Sie sich für `pc` am Kontrollflussgraph. Pro Teilaufgabe sind 1.5 Punkte erreichbar.
Achtung: Bei `pc = 18` müssen Sie die Formel so umformen, dass alle Eigenschaften von `y` erhalten bleiben!

a) `pc = 7`:

Lösung: $pc = 7$

b) `pc = 8`:

Lösung: $pc = 8 \land y = 0$

c) `pc = 9`:

Lösung: $pc = 9 \land y = 0 \land x \in [\![\texttt{int}]\!]$

d) `pc = 11`:

Lösung: $pc = 11 \land y = 0 \land x \in [\![\texttt{int}]\!] \land x \neq 2$

e) `pc = 12`:

Lösung: $pc = 12 \land y = 0 \land x \in [\![\texttt{int}]\!] \backslash \{2\} \land x \geq 0$

f) `pc = 14`:

Lösung: $pc = 14 \land y = 0 \land x \in [\![\texttt{int}]\!] \backslash \{2\} \land x < 0$

g) `pc = 17`:

Lösung: $pc = 17 \land ((y > 0 \land x \in [\![\texttt{int}]\!] \backslash \{3\} \land x \geq 0) \lor (x < 0 \land y > 0))$

h) `pc = 18`:

Lösung: $pc = 18 \land x = 0 \land y > 0$

Aufgabe 4, Fortsetzung: Symbolische Erreichbarkeitsanalyse (12 Punkte)

Aufgabe 5: Hoare-Logik (14 Punkte)

Vervollständigen Sie die gegebenen Beweisabrisse, indem Sie die Regeln der Hoare-Logik anwenden. Es ist nicht notwendig, die Formeln weiter zu vereinfachen.

---

a) $\left\{ \mathtt{x} + 2 = 4 \right\}$ x = x + 2; $\left\{ x = 4 \right\}$ (1 Punkte)

---

b) $\left\{ \mathtt{y} < 0 \land y = 1 \right\}$ x = y; $\left\{ \mathtt{x} < 0 \land y = 1 \right\}$ (1 Punkte)

---

c) Die Teilaufgabe d) gezeigte Schleife terminiert nicht. Nehmen Sie an, dass die Variablen `x` und `y` einen unbeschränkten Wertebereich haben und keine Über-/Unterläufe auftreten können. Damit kann bewiesen werden, dass kein Zustand nach der Schleife erreichbar ist, durch die Nachbedingung false ausgedrückt.

Kreuzen Sie von folgenden Formeln die korrekte Invariante an. Diese Invariante soll garantieren, dass die Schleife niemals verlassen wird, oder anders gesprochen, immer wieder betreten wird. (1 Punkte)

- ☐ $\mathtt{x} = 2 \land \mathtt{y} = 6$
- ☐ $\mathtt{x} > \mathtt{y}$
- ☐ $\mathtt{x} \neq \mathtt{y}$
- ☑ $\mathtt{x} < \mathtt{y}$

d) Vervollständigen Sie den Beweisabriss mit der von Ihnen gewählten Invariante. Sie brauchen die Nebenrechnungen nicht durchzuführen, allerdings können Sie damit die vorige Antwort gegenprüfen. (5 Punkte)

$\left\{ \text{true} \right\}$

$\left\{ 2 < 6 \right\}$

x = 2; y = 6;

$\left\{ \mathtt{x} < \mathtt{y} \right\}$

while $(\mathtt{x} \neq \mathtt{y})$ **do**

$\left\{ \mathtt{x} < \mathtt{y} \land \mathtt{x} \neq \mathtt{y} \right\}$

$\left\{ \mathtt{x} - 2 < \mathtt{y} + 2 \right\}$

x = x - 2; y = y + 2;

$\left\{ \mathtt{x} < \mathtt{y} \right\}$

end

$\left\{ \mathtt{x} < \mathtt{y} \land \neg(\mathtt{x} \neq \mathtt{y}) \right\}$

$\left\{ \text{false} \right\}$

Hinweise: Die Aufgabe kann wie gewohnt mit den normalen Hoare-Regeln gelöst werden, es ist trotz Nichtterminiertung kein besonderer Trick notwendig. Ein an sich richtiger Beweisabriss mit einer falschen Invariante wird als Folgefehler gewertet, d.h. Sie können unabhängig von der gewählten Invariante im Beweisabriss volle Punktzahl erreichen, sofern Sie die Beweisregeln ansonsten korrekt anwenden.

Aufgabe 5, Fortsetzung: Hoare-Logik (14 Punkte)

e) Das folgende Programm berechnet die (positive) Nullstelle der Polynomialgleichung $y = (x-1)*(x-1)-4$. Am Anfang trägt y den Wert -3 und x den Wert 0. In jeder Iteration wird x um eins erhöht und y entsprechend angepasst, so dass die Polynomialgleichung weiterhin erfüllt ist. Sobald y gleich 0 ist wurde die Nullstelle gefunden und die Schleife wird verlassen.

Ihre Aufgabe ist es, zu beweisen, dass die positive Nullstelle des Polynoms bei x=3 korrekt gefunden wird. Vervollständigen Sie den Beweisabriss mit Hilfe der Invariante, die bereits an einigen Stellen eingetragen ist. (5 Punkte)

$\{\ \mathrm{x} = 0 \land \mathrm{y} = -3\ \}$

$\{\ \mathrm{y} = (\mathrm{x}-1)*(\mathrm{x}-1) - 4 \land \mathrm{x} \geq 0\ \}$ Invariante gilt vor der Schleife

while$(\mathrm{y} \neq 0)$ **do**

$\{\ \mathrm{y} \neq 0 \land \mathrm{y} = (\mathrm{x}-1)*(\mathrm{x}-1) - 4 \land x \geq 0\ \}$

$\{\ \mathrm{y} + 2*\mathrm{x} - 1 = ((\mathrm{x}+1)-1)*((\mathrm{x}+1)-1) - 4 \land x+1 \geq 0\ \}$

$\mathrm{y} = \mathrm{y} + 2*\mathrm{x} - 1;$

$\{\ \mathrm{y} = ((\mathrm{x}+1)-1)*((\mathrm{x}+1)-1) - 4 \land x+1 \geq 0\ \}$

$\mathrm{x} = \mathrm{x} + 1;$

$\{\mathrm{y} = (\mathrm{x}-1)*(\mathrm{x}-1) - 4 \land x \geq 0\}$ Invariante gilt am Ende eines Schleifendurchlaufs

end

(*) $\{\ \mathrm{y} = 0 \land \mathrm{y} = (\mathrm{x}-1)*(\mathrm{x}-1) - 4 \land x \geq 0\ \}$

$\{\ \mathrm{x} = 3\ \}$

f) Kreuzen Sie diejenige Formel an, die an der mit (*) markierten Stelle in Teilaufgabe e) garantiert gilt. (1 Punkte)

- ☐ *false*
- ☐ $y = (\mathrm{x}-1)*(\mathrm{x}-1)$
- ☑ $0 = (\mathrm{x}-1)*(\mathrm{x}-1) - 4$
- ☐ $\mathrm{x} = 3 \land y \neq 0$

Aufgabe 6: Objektorientierte Programme (12 Punkte)

Gegeben ist eine partielle Spezifikation des Interfaces bzw Klasse `List`. Diese soll mit Hilfe einer Modellvariable x spezifiziert werden, einer mathematischen Sequenz von Elementen vom Typ `Elem`. Die Klasse soll sich analog verhalten zu `java.util.List`.

Hinweise:
Verwenden Sie ausschließlich folgende mathematische Operatoren auf Sequenzen, um die Vor- und Nachbedingungen der Methoden zu spezifizieren. Weitere Operatoren stehen hier nicht zur Verfügung!

- $\langle\rangle$ bezeichnet die leere Sequenz (alternativ `[]`)
- $|x|$ bezeichnet die Länge der Sequenz x
- $x[i]$ bezeichnet das i. Element von x wenn $0 \leq i < |x|$

class `List`

model x: eine mathematische Sequenz mit Elementen vom Typ `Elem`

Die Methode `partition` ändert die Liste so, dass zunächst alle negativen Werte kommen, dann alle Vorkommen der Zahl 0, und dann alle positiven Werte.

a) Spezifizieren Sie folgende Nachbedingungen:

- Die Länge der Liste ist gleich geblieben (2 Punkte)
- Wenn die Zahl 7 vorher enthalten war, dann ist die 7 immer noch enthalten. (4 Punkte)
- Es gibt zwei Indizes k_0 und k_1 zwischen denen alle Elemente gleich 0 sind (6 Punkte)

Verwenden Sie **old( )**, um auf den Zustand zum Beginn des Methodenaufrufs zu verweisen.

Verwenden Sie Quantoren für die zweite und dritte Eigenschaft und nutzen Sie evtl. Klammern, um den Geltungsbereich der Quantoren zu verdeutlichen. Nutzen Sie nur die oben genannten Operationen auf mathematischen Sequenzen. Achten Sie darauf, nur auf gültige Indizes zuzugreifen, und bedenken Sie, dass evtl. gar keine 0 vorkommen muss. Es empfielt sich, die Lösung mit einfachen Beispielen nachzuprüfen.

method `partition()`

ensures

Lösung: $|x| = |old(x)|$

Lösung: $(\exists i.\ old(x[i]) = 7) \implies (\exists i.\ x[i] = 7)$

Lösung: $\exists k_0, k_1.\ 0 \leq k_0 \leq k_1 \leq |x| \land \forall i.\ k_0 \leq i < k_1 \implies x[i] = 0$

Aufgabe 7: SAT/SMT (15 Punkte)

a) Kreuzen Sie für jede Formel an, ob diese erfüllbar, allgemeingültig oder unerfüllbar ist. Es kann auch jeweils mehr als eine dieser drei Eigenschaften gleichzeitig zutreffen!

(3 Punkte)

		allgemeingültig	erfüllbar	unerfüllbar
1	$(\text{false} \wedge A) \vee (\text{false} \wedge \neg A)$	☐	☐	☑
2	$\text{true} \implies A$	☐	☑	☐
3	$(\neg A \implies \text{false}) \wedge \neg A$	☐	☐	☑
4	$A \wedge \neg B \wedge C \wedge B$	☐	☐	☑
5	$(\neg A \implies \text{false}) \implies A$	☑	☑	☐
6	$(A \implies \neg B) \iff (\neg A \vee \neg B)$	☑	☑	☐

b) Geben Sie für jede erfüllbare aber nicht allgemeingültige Formel aus a) eine erfüllende Belegung der atomaren Propositionen an.

(2 Punkte)

Variablenbelegung für Sat:

- 2: $A := true$

c) Kreuzen Sie alle aussagenlogische Formeln an, welche durch die jeweils gegebene Belegung erfüllt werden. Es sind evtl. mehrere Antworten pro Teilaufgabe korrekt.

(2 Punkte)

1) Sei Belegung s_1 gegeben durch:
$s_1 := \{ A \mapsto false,\ B \mapsto true,\ C \mapsto false \}$

- ☑ $(\neg A \implies B) \wedge (C \implies A) \wedge B$
- ☐ $(A \vee B) \wedge (A \implies C) \wedge C$
- ☑ $B \wedge (A \implies \neg C) \wedge \neg A \wedge B$
- ☐ $(A \implies B) \wedge (A \iff \neg C)$

Aufgabe 7, Fortsetzung: SAT/SMT (15 Punkte)

d) Gegeben Sei die Formel:
$A \wedge (\neg A \vee \neg B) \wedge (\neg C \vee \neg B) \wedge ((\neg C \vee \neg D) \implies D)$

Berechnen Sie mit Hilfe des DPLL Algorithmus ob die Formel erfüllbar ist. (4 Punkte)

Geben Sie in jedem Schritt an:

- welche atomare Proposition mit welchem Wahrheitswert belegt wird
- ob die jeweilige Belegung zwingend ist (hier typischerweise durch Unit Propagation, UP), oder durch Fallunterscheidung erfolgt (Split)
- die daraus resultierende Formel, vereinfacht mit der von Ihnen gewählten Belegung

i. In KNF bringen: $A \wedge (\neg A \vee \neg B) \wedge (\neg C \vee \neg B) \wedge (C \vee D) \wedge D$

ii. UP: $A := true$ oder alternativ $D := true$

iii. UP: Wenn vorher $A := true$ dann $B := false$ oder $D := true$,
wenn vorher $D := true$ dann $A := true$

iv. UP: Je $B := false$ oder $D := true$ abhängig von der Zeile davor

$true$

Ist die Formel erfüllbar? Wenn ja, geben Sie die in d) gefundene erfüllende Belegung vollständig an. Es genügt nicht, nur auf die Schritte des Algorithmus zu verweisen. (1 Punkte)

☐ unerfüllbar ☑ erfüllbar mit $s = \left\{ A \mapsto true, B \mapsto false, C \mapsto false \vee true, D \mapsto false \right\}$

Aufgabe 7, Fortsetzung: SAT/SMT (15 Punkte)

e) Es sei die folgende prädikatenlogische Formel über die Integer-Variablen x,y aus $\mathbb{Z}$ gegeben: (3 Punkte)

$$(\exists\ z.\ z \geq 10 \land y < z) \land (x < 10) \land (x = y)$$

Lösung: Umgestellte Formel (nicht gefragt): $y < z \leq 10 \land y < 10$

Welche der folgenden Belegungen erfüllen diese Formel? Kreuzen Sie die korrekte Antwort an und begründen Sie jeweils kurz

- $s = \{x \mapsto 9, y \mapsto 9\}$ ☑ Ja ☐ Nein weil erster Teil wird mit $z \mapsto 10$ wahr
- $s = \{x \mapsto 9, y \mapsto 10\}$ ☐ Ja ☑ Nein weil x und y müssen gleich sein
- $s = \{x \mapsto 10, y \mapsto 10\}$ ☐ Ja ☑ Nein weil $x < 10$ nicht erfüllt

Aufgabe 8: Temporallogik (12 Punkte)

a) Kreuzen Sie jeweils alle Aussagen an, die zu den gegebenen LTL-Formeln passen. Das bedeutet: Wenn ein Lauf die LTL-Formel erfüllt, dann soll dieser zu den von Ihnen angekreuzten Aussagen passen (aber nicht notwendigerweise umgekehrt). (4 Punkte)

1) $\Diamond(P \,\mathcal{U}\, Q)$

- ☐ Wenn P nicht eintritt, tritt auch Q nicht ein
- ☑ Irgendwann gilt P eine Weile, dann tritt Q garantiert ein
- ☐ Q gilt, bis irgendwann P eintritt
- ☑ $\neg Q$ darf gelten bevor P eintritt

2) $\Box(\circ \circ \neg P)$

- ☐ Im ersten Zustand muss $\neg P$ gelten
- ☑ Es muss sein, dass in mindestens einem Zustand $\neg P$ gilt
- ☑ Es kann einen Zustand geben, in dem P gilt
- ☐ Wenn im ersten Zustand $\neg P$ gilt, dann ist die Formel bereits erfüllt

b) Formalisieren Sie jeweils den gegebenen Satz als LTL-Formel über den Propositionen P und Q. Achten Sie darauf, mit Klammern deutlich zu machen, wie die Formeln genau gemeint sind. (4 Punkte)

1) P gilt garantiert mindestens einmal und dann zu einem späteren Zeitpunkt nochmal

$\Diamond(P \wedge \circ(\Diamond P))$

Die Formel ist eine ☐ Sicherheitseigenschaft oder eine ☑ Lebendigkeitseigenschaft

2) Immer wenn P eintritt, dann bleibt P eine Zeit lang erhalten bis garantiert Q eintritt

$\Box(P \implies (P \,\mathcal{U}\, Q))$

Kreuzen Sie an, welche Aussage(n) stimmt/stimmen:

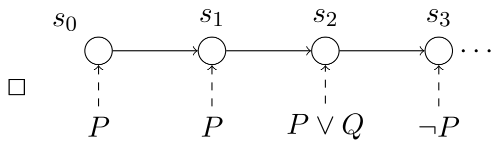
- ☑ Die Formel hat endliche Abläufe als Gegenbeispiele
- ☑ Die Formel hat unendliche Abläufe als Gegenbeispiele

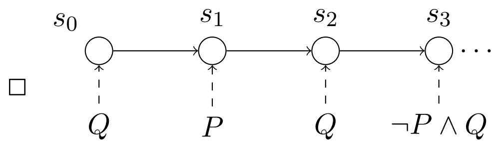
Aufgabe 8, Fortsetzung: Temporallogik (12 Punkte)

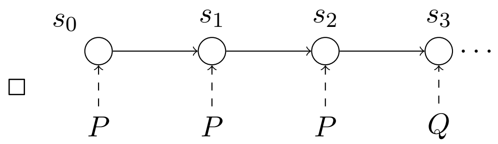
c) Kreuzen Sie in Teilaufgabe 1)-2) jeweils den einen undendlichen Lauf $s_0, s_1, \ldots$ an, der die gegebene LTL-Formel definitiv erfüllt. Für die an jedem Zustand mit einer gestrichelten Linie annotierten Formeln dürfen sie annehmen, dass diese dort garantiert gelten. Für die Zustände nach s_3 sollen Sie annehmen, dass für diese dieselben Formeln wie für s_3 erfüllt sind. (4 Punkte)

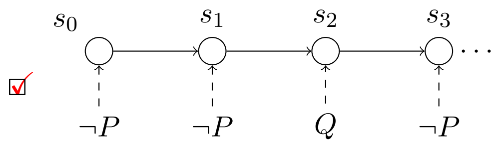
Lösung:

1) $Q\,\mathcal{W}\,(\neg P)$ Äquivalent zu: $(Q\,\mathcal{U}\,\neg P) \vee \Box Q$

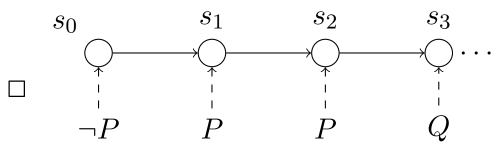


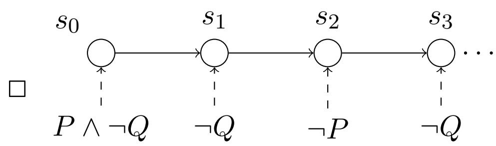


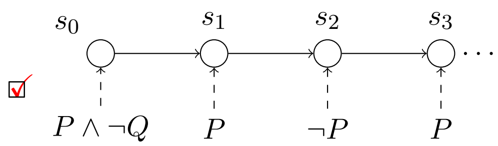


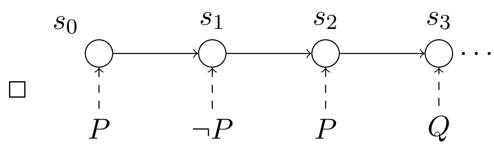


2) $P \wedge (\circ P \implies \neg \circ \circ P)$









Zusatzblatt: Fortsetzung von Aufgabe ___