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

Zweite Probeklausur Formale Spezifikation und Verifikation

München, 4. April 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: Allgemeine Fragen (8 Punkte)

a) Testen deckt Fehler auf mit Hilfe konkreter Eingabewerte. Ist es sinnvoll alle möglichen Eingabewerte eines Programms zu testen? Begründen Sie kurz warum, oder warum nicht. Sofern es nicht sinnvoll ist, führen Sie eine Alternative auf, durch die das gesamte Programmverhalten vollständig geprüft werden kann. (2 Punkte)

Testen: schwer/unmöglich alles zu testen weil es zu lange dauert. Alternative: Formale Verifikation. KEIN property based testing, fuzzing etc. als Alternative!

b) Erklären Sie, was eine Sicherheitseigenschaft ist.

Geben Sie ein Beispiel für eine solche Eigenschaft für eine Ampelsteuerung an. (2 Punkte)

Etwas Schlechtes passiert nie oder hat endliche Gegenbeispiele. Die Ampel zeigt nie Grün und Rot zusammen.

c) Sie haben eine Software gekauft, bei welcher der Hersteller damit wirbt, dass diese "formal verifiziert und deshalb 100% sicher" ist. Ist diese Formulierung problematisch? Begründen Sie ihre Antwort! (2 Punkte)

Es ist nicht klar, was überhaupt spezifiziert wurde. Abhängig von der Spezifikation kann die Verifikation zwar leichter, aber weniger aussagekräftig sein. Es können Fehler bei der Modellierung oder Spezifikation passiert sein, diese lassen sich niemals ganz ausschließen.

d) Nennen Sie ein besonderes Merkmal von eingebetteten Systemen/Cyber-Physical Systems und erklären Sie, warum dieses zu Herausforderungen bei der Analyse führt. (2 Punkte)

technisches System kommuniziert mit der Umwelt und vice versa, komplexes Verhalten, Realzeit, offen korrigieren! ...

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. (2 Punkte)

```
\** Gibt das Quadrat des Eingabewerts zurueck (num * num).
 *  Eingabeparameter und Rueckgabewert haben den Typen int,
 *  der in Java 32bit umfasst.
 *
 *  Ist die Eingabe negativ, so wirft die Methode eine Exception.
 *  Fuer Integer Overflows wirft die Methode ebenfalls eine Exception.
 */
public int square(int num) { ... }
```

Jeder Test soll in der Programmiersprache Java kompilierbar sein.

Testfälle:

Eingabe `num`	erwarete(r) Rückgabewert/Ergebnis

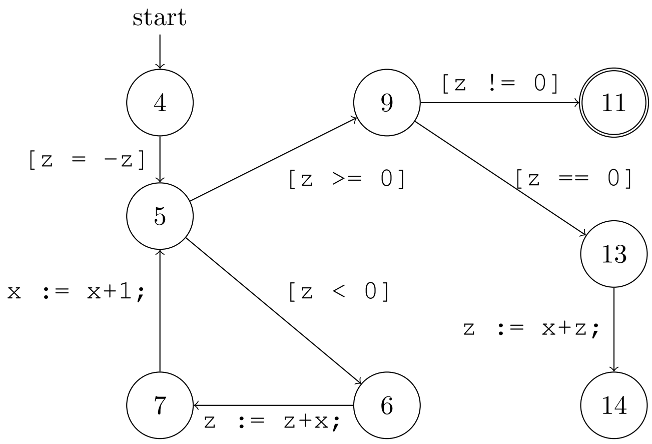
Lösung: Ein Testfall für negative Zahlen muss vorhanden sein, ein Testfall für 0,
einen für eine Zahl die einen Wert größer INT_MAX berechnet
und eine die einen validen Wert zurueck gibt der kleiner INT_MAX aber größer 0 ist.

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. (2 Punkte)

Lösung: Property-Based Testing (auch Fuzzing möglich)
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 (12 Punkte)

```c
1  int x = 1;
2  int z = nondet_bool();
3  int main() {
4    z = -z;
5    while (z < 0) {
6      z = z + x;
7      x = x + 1;
8    }
9    if (z != 0) {
10     ERROR:
11       return 1;
12   }
13   z = x + z;
14   return 0;
15 }
```



Es ist ein Programm als C-Code (links) sowie der dazu äquivalente Kontrollflussautomat (rechts) gegeben. Wir spezifizieren, dass kein Fehlerzustand erreicht werden darf, hier die Programmzeile 11.

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. (11 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
9	1	0
13	1	0
14	1	1

$pc \mapsto$	$x \mapsto$	$z \mapsto$
4	1	1
5	1	-1
6	1	-1
7	1	0
5	2	0
9	2	0
13	2	0
14	2	2

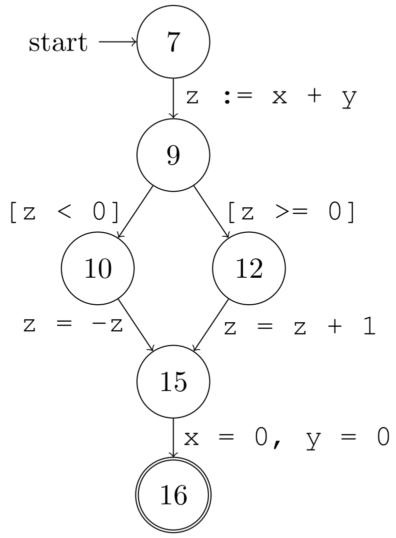
b) Wie können Sie nach der expliziten Erreichbarkeitsanalyse ablesen ob ein Programm sicher ist und ist das gegebene Programm sicher?

(1 Punkte)

Lösung: Wenn Fehlerzustand erreichbar ist es unsicher.
Sicher, da Zeile 11 nicht erreichbar.

Aufgabe 4: Symbolische Erreichbarkeitsanalyse (10 Punkte)

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



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.

Aufgabe: Führen Sie die symbolische Erreichbarkeitsanalyse durch, d.h. bestimmen Sie Formeln über den Variablen `x`, `y`, `z` sowie dem Programmzähler `pc`, die zusammen genau charakterisieren, welche Zustände erreichbar sind.

Hinweise:

- Orientieren Sie sich bei den Werten für `pc` am Kontrollflussautomaten!
- Zu Beginn von `main` gilt `pc` $= 7$, d.h., Sie können die Einschränkung `x,y` $\in [\![$`int`$]\!]$ weglassen.
- Die beiden Zuweisungen von $pc = 15$ nach $pc = 16$ passieren in einem Schritt

Lösung:

$pc = 7$

$pc = 9 \land z = x + y$

$pc = 10 \land z = x + y \land z < 0$

$pc = 12 \land z = x + y \land z \geq 0$

$pc = 15 \land z = -x - y \land x + y < 0$

$pc = 15 \land z = x + y + 1 \land x + y \geq 0$

$pc = 16 \land z > 0 \land x = 0 \land y = 0$

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) $\{x + 42 = 4\}$ `x = x + 42;` $\{x = 4\}$ (1 Punkte)

---

b) $\{$ `false` $\}$ `x = y + 2;` $\{$ `false` $\}$ (1 Punkte)

---

c) Die Teilaufgabe d) gezeigte Schleife terminiert nicht. Nehmen Sie an, dass die Variablen `x` und `y` Integer sind mit 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)

- ☐ $x = -2 \land y = 3$
- ☐ $x \neq y$
- ☑ $x = 1 - y$
- ☐ $x > y$

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)

$\{$ true $\}$

$\{ -2 = -3 + 1 \}$

x = -2; y = 3;

$\{ x = -y + 1 \}$

while $(x \neq y)$ **do**

$\{ x = -y + 1 \land x \neq y \}$

$\{ x - 1 = -(y + 1) + 1 \}$

x = x - 1; y = y + 1;

$\{ x = -y + 1 \}$

end

$\{ x = -y + 1 \land \neg(x \neq y) \}$

$\{$ false $\}$

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 Nullstelle der Polynomialgleichung $y = x * x * x - 8$. Am Anfang trägt y den Wert -8 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 $\mathtt{x} = 2$ korrekt gefunden wird. Vervollständigen Sie den Beweisabriss mit Hilfe der Invariante, die bereits an einigen Stellen eingetragen ist (ohne Nebenrechnungen). (5 Punkte)

$\{ \mathtt{x} = 0 \land \mathtt{y} = -8 \}$

$\{ \mathtt{y} = \mathtt{x} * \mathtt{x} * \mathtt{x} - 8 \land \mathtt{x} \geq 0 \}$ Invariante gilt vor der Schleife

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

$\{ \mathtt{y} \neq 0 \land \mathtt{y} = \mathtt{x} * \mathtt{x} * \mathtt{x} - 8 \land x \geq 0 \}$

$\{ \mathtt{y} + 3 * \mathtt{x} * \mathtt{x} + 3\mathtt{x} - 7 = (\mathtt{x}+1) * (\mathtt{x}+1) * (\mathtt{x}+1) - 8 \land x + 1 \geq 0 \}$

$\mathtt{y} = \mathtt{y} + 3 * \mathtt{x} * \mathtt{x} + 3\mathtt{x} + 1;$

$\{ \mathtt{y} = (\mathtt{x}+1) * (\mathtt{x}+1) * (\mathtt{x}+1) - 8 \land x + 1 \geq 0 \}$

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

$\{ \mathtt{y} = \mathtt{x} * \mathtt{x} * \mathtt{x} - 8 \land x \geq 0 \}$ Invariante gilt am Ende eines Schleifendurchlaufs

end

(*) $\{ \mathtt{y} = 0 \land \mathtt{y} = \mathtt{x} * \mathtt{x} * \mathtt{x} - 8 \land x \geq 0 \}$

$\{ \mathtt{x} = 2 \}$

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

- ☐ *false*
- ☐ $y = \mathtt{x} * \mathtt{x} * \mathtt{x}$
- ☑ $0 = \mathtt{x} * \mathtt{x} * \mathtt{x} - 8$
- ☐ $\mathtt{x} = 3 \land y \neq 0$

Aufgabe 6: Objektorientierte Programme (14 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`.

Spezifizieren Sie folgende Methoden

- Der Konstruktor `List` initialisiert die Liste so, dass das gegebene Element `e` genau `n` mal vorkommt
- Die Methode `reverse` kehrt die Reihenfolge der Elemente um, in der diese in der Liste vorkommen, die Länge der Liste ändert sich dabei nicht.
- Die Methode `hasDuplicates` gibt zurück, es zwei unterschiedliche gültige Indizes gibt, an denen das selbe Element gespeichert ist. Die Liste bleibt dabei unverändert.

Verwenden Sie ausschließlich folgende mathematische Operatoren auf Sequenzen, um die Vor- und Nachbedingungen der Methoden zu spezifizieren.

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

Gegebenfalls benötigen Sie Quantoren. Mit **old( )** können Sie auf den alten Wert der Modellvariable zugreifen.

class `List`

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

constructor `List(Elem e, int n)`

requires $n \geq 0$ (1 Punkte)

ensures $\forall\ 0 \leq i < n.\ x[i] = e$ (2 Punkte)

method `reverse()`

requires $true$ (1 Punkte)

ensures $|x| = \mathbf{old}(|x|) \wedge \forall\ 0 \leq i < |x|.\ x[i] = \mathbf{old}(x[|x| - i - 1])$ (4 Punkte)

method `hasDuplicates()`

returns `boolean result`

requires $true$ (1 Punkte)

ensures $x = \mathbf{old}(x) \wedge (result \Leftrightarrow \exists\ 0 \leq i < j < |x|.\ x[i] = x[j])$ (5 Punkte)

Aufgabe 7: SAT/SMT (16 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	$A \lor \text{true} \lor B$	☑	☑	☐
2	$A \implies (\neg A \land B)$	☐	☑	☐
3	$(\neg A \implies A) \land (\neg A)$	☐	☐	☑
4	$(A \lor \neg B) \lor (\neg A \lor \neg C) \lor (C \land A)$	☑	☑	☐
5	$(\neg A \implies B) \land A \land (A \implies \neg B)$	☐	☑	☐
6	$(A \implies \neg B) \land A \land (\neg A \lor B)$	☐	☐	☑

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: $A = false$, B beliebig
- 5: $A = true$, $B = false$

c) Kreuzen Sie für die gezeigten Formeln an, ob die gegebene Belegung diese erfüllt.

(2 Punkte)

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

	erfüllt	nicht erfüllt
$(\neg A \implies \neg B) \land (C \implies \neg A)$	☐	☑
$(A \lor B) \land (A \implies C) \land C$	☑	☐
$B \land (A \implies \neg C) \land \neg A \land B$	☑	☐
$(\neg A \implies (\neg B \lor \neg C)) \land (A \iff \neg C)$	☐	☑

Aufgabe 7, Fortsetzung: SAT/SMT (16 Punkte)

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

Bringen Sie die Formeln in KNF (1 Punkte)

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
- welche Vereinfachung Sie durchgeführt haben Unit Propagation (UP), Pure Literal Elimination (PLE), oder Fallunterscheidung (Split)
- die daraus resultierende Formel, vereinfacht mit der von Ihnen gewählten Belegung

i. In KNF bringen: $(\neg A \lor \neg B) \land (\neg C \lor D) \land (\neg D \lor C) \land A$

ii. UP: $A := true$ (PLE auch OK für A oder $\neg B$)
$\neg B \land (\neg C \lor D) \land (\neg D \lor C)$

iii. UP: $B := false$ (wenn vorher $A := true$)
$(\neg C \lor D) \land (\neg D \lor C)$

iv. SPLIT: z.B. $C := false$ (Auch OK sind $C := true$ oder $D := false$ oder $D := true$)
$\neg D$

v. UP: z.B. $D := false$ (Beachte: es müssen je beide, C und D, gleich belegt sein!)
$true$

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 $s = \{ A \mapsto true, B \mapsto false, C \mapsto false, D \mapsto false$
oder $A \mapsto true, B \mapsto false, C \mapsto true, D \mapsto true \}$

Aufgabe 7, Fortsetzung: SAT/SMT (16 Punkte)

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

$$(\exists\, x.\ x < 7 \land y = x) \land (x \geq 7) \land (x = y + 1)$$

Lösung: Umgestellte, einfachere Formel (nicht gefragt): $y < 7 \land y + 1 \geq 7$

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

- $s = \{x \mapsto 9, y \mapsto 7\}$ ☐ Ja ☑ Nein weil $9 = 7 + 1$ nicht erfüllt
- $s = \{x \mapsto 7, y \mapsto 6\}$ ☑ Ja ☐ Nein weil 7 ist $6 + 1$ und $6 < 7$ und $7 \geq 7$
- $s = \{x \mapsto 6, y \mapsto 5\}$ ☐ Ja ☑ Nein weil $x \geq 7$ 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. (4 Punkte)

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

- ☐ Wenn P nicht eintritt, tritt auch Q nicht ein
- ☑ Es reicht, wenn P nach Q wahr wird
- ☑ Wenn Q nicht eintritt dann ist die Formel insgesamt nicht erfüllt
- ☐ Immer wenn Q gilt, dann muss P direkt vorher wahr sein

2) $\Diamond(P \land Q \land (\circ P))$

- ☑ P tritt garantiert zu zwei unterschiedlichen Zeitpunkten ein
- ☐ Immer wenn P gilt, dann gilt auch Q
- ☑ Q tritt mindestens einmal auf
- ☐ Q tritt höchstens einmal auf

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 kann nicht gleichzeitig mit Q wahr werden

$\Box(\neg(P \land Q))$

Die Formel ist eine ☑ Sicherheitseigenschaft oder eine ☐ Lebendigkeitseigenschaft

2) Falls P überhaupt irgendwann einmal wahr wird, dann gibt es auch irgendeinen Zeitpunkt, ab dem Q wahr wird und ab dann auch immer gilt.

$(\Diamond P) \implies (\Diamond\Box Q)$

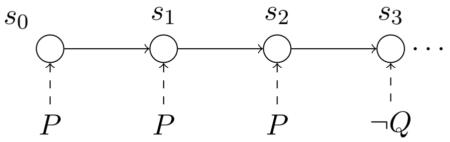
Kreuzen Sie an, welche Aussage(n) stimmt/stimmen:

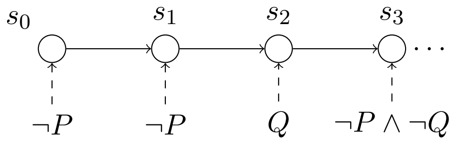
- ☐ Die Formel hat endliche Abläufe als Gegenbeispiele
- ☑ Die Formel hat unendliche Abläufe als Gegenbeispiele

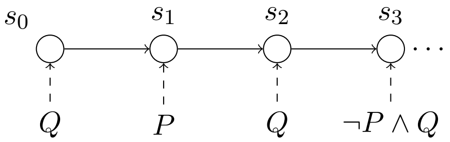
Aufgabe 8, Fortsetzung: Temporallogik (12 Punkte)

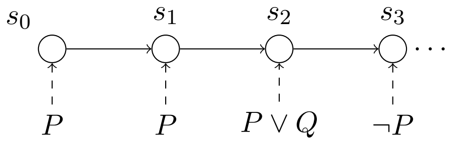
c) Kreuzen Sie in Teilaufgabe 1)-2) jeweils den einen undendlichen Lauf $s_0, s_1, \dots$ an, der die gegebene LTL-Formel definitiv erfüllt. Für die an jedem Zustand mit einer gestrichelten Linie annotierten Formeln dürfen sie annehmen, dass diese dort garantiert gelten. Für die Zustände nach s_3 sollen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

1) $\square(P \implies \neg(\circ Q))$

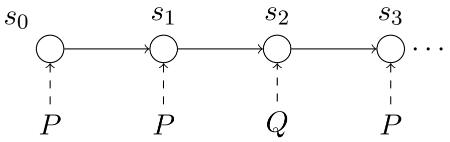
☐ 

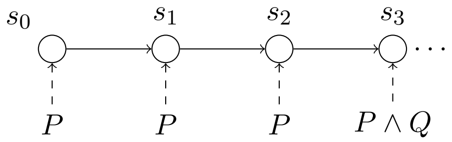
☑ 

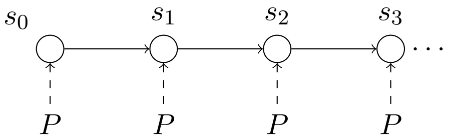
☐ 

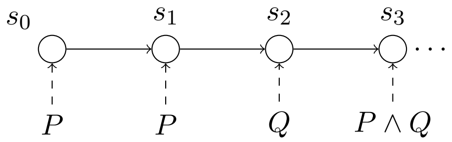
☐ 

2) $P\,\mathcal{U}\,(P \wedge Q)$

☐ 

☑ 

☐ 

☐ 

Zusatzblatt: Fortsetzung von Aufgabe __