Karlsruher Institut für Technologie

Name: _____

Vorname: _____

Matrikel-Nr.: _____

Klausurnummer:

_____

Klausur Formale Systeme

Fakultät für Informatik

WS 2025/26

Prof. Dr. Bernhard Beckert

05. März 2026

Die Bearbeitungszeit beträgt 60 Minuten.

MUSTERLSG

A1 (9)	A2 (11)	A3 (5)	A4 (7)	A5 (10)	A6 (10)	A7 (8)	Σ (60)

Bewertungstabelle bitte frei lassen!

Gesamtpunkte:

1 Zur Einstimmung (1,5 + (2 + 2,5) + 3 = 9 Punkte)

a. Überprüfen Sie die folgende Hornformel auf Erfüllbarkeit. Benutzen Sie den Markierungsalgorithmus (auch „Erfüllbarkeitstest für Hornformeln“) aus der Vorlesung. Geben Sie unter „Schritt n“ an, welches Literal im n-ten Schritt markiert wurde. Geben Sie zudem an, ob der Algorithmus mit „Erfüllbar“ oder „Unerfüllbar“ terminiert.

$$(\neg P_1 \vee P_3) \ \wedge \ (\neg P_3 \vee P_2) \ \wedge \ (\neg P_1 \vee \neg P_2) \ \wedge \ P_1$$

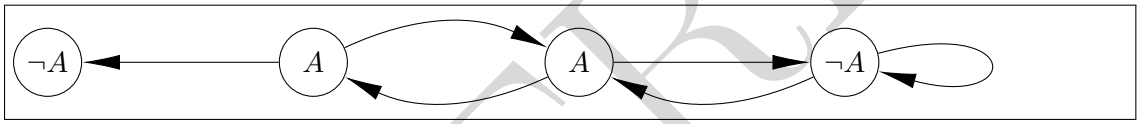
Schritt 1:	Schritt 2:	Schritt 3:	Schritt 4:
P_1	P_3	P_2	$\square$

Ergebnis: Unerfüllbar

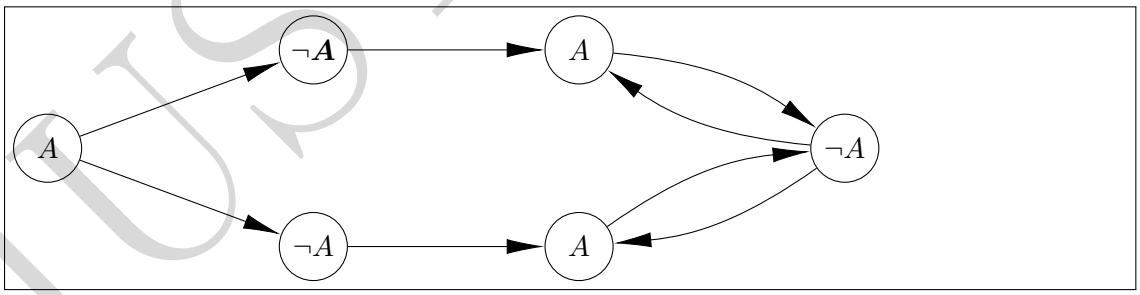
b. Geben Sie je eine Belegung der aussagenlogischen Variable A (A oder $\neg A$) in den Welten der folgenden Kripkestrukturen an, so dass die jeweils gegebene Formel in der Welt wahr ist.

Hinweis: Die Formel **0** entspricht der Formel *Falsch*, die in keiner Welt wahr ist.

i. $(\Diamond A \wedge \Diamond \neg A) \ \vee \ (\Box \mathbf{0} \leftrightarrow \neg A)$



ii. $((\Diamond \neg A) \rightarrow A) \wedge (A \rightarrow \Box \Diamond \Diamond \neg A)$



MUSTERLSG

Fortsetzung 1 Zur Einstimmung

c. Transformieren Sie die aussagenlogische Formel

$$\neg\Big((P \vee Q) \to R\Big)$$

in die **kurze konjunktive Normalform**. Verwenden Sie das Verfahren aus der Vorlesung. Geben Sie als Zwischenschritte die Definition der Hilfsvariablen an.

1. Schritt: Abkürzungen einführen (0,5 Punkte)

$$\begin{aligned} &A \leftrightarrow (P \vee R) \\ &B \leftrightarrow (A \to R) \\ &\neg B \end{aligned}$$

2. Schritt: Transformation in KNF

(1 Punkt)

$$\begin{aligned} & A \leftrightarrow (P \vee Q) \\ \equiv\ & \Big(A \to (P \vee Q)\Big) \wedge \Big((P \vee Q) \to A\Big) \\ \equiv\ & \Big(A \to (P \vee Q)\Big) \wedge \Big(P \to A\Big) \wedge \Big(Q \to A\Big) \\ \equiv\ & (\neg A \vee P \vee Q) \wedge (\neg P \vee A) \wedge (\neg Q \vee A) \end{aligned}$$

(1 Punkt)

$$\begin{aligned} & B \leftrightarrow (A \to R) \\ \equiv\ & \Big(B \to (A \to R)\Big) \wedge \Big((A \to R) \to B\Big) \\ \equiv\ & \Big(\neg B \vee \neg A \vee R)\Big) \wedge \Big(\neg(\neg A \vee R) \vee B\Big) \\ \equiv\ & \Big(\neg B \vee \neg A \vee R)\Big) \wedge \Big((A \wedge \neg R) \vee B\Big) \\ \equiv\ & \Big(\neg B \vee \neg A \vee R)\Big) \wedge \Big(A \vee B\Big) \wedge \Big(\neg R \vee B\Big) \end{aligned}$$

Ergebnis: (0,5 Punkte)

$$\begin{aligned} &(\neg A \vee P \vee Q) \wedge (\neg P \vee A) \wedge (\neg Q \vee A) \wedge \\ &\Big(\neg B \vee \neg A \vee R)\Big) \wedge \Big(A \vee B\Big) \wedge \Big(\neg R \vee B\Big) \wedge \\ &\neg B \end{aligned}$$

MUSTERLSG

2 Konservative Erweiterungen von Theorien

(2 + 5 + 4 = 11 Punkte)

Diese Aufgabe behandelt das Konzept der *konservativen Erweiterung* einer Theorie. Gegeben eine Theorie $\mathcal{T}(T)$ mit Signatur Σ, nennen wir eine Theorie $\mathcal{T}(T')$ über Signatur $\Sigma' \supseteq \Sigma$ eine konservative Erweiterung von $\mathcal{T}(T)$, wenn alle Formeln über der kleineren Signatur Σ, die aus der größeren Theorie $\mathcal{T}(T')$ folgen, schon aus der kleineren Theorie $\mathcal{T}(T)$ folgen. Formal definieren wir dies wie folgt:

Definition.

1. Eine Theorie $\mathcal{T}(T')$ über einer Signatur Σ' heißt *Erweiterung* einer Theorie $\mathcal{T}(T)$ über Σ, wenn

   $$\Sigma \subseteq \Sigma' \qquad \text{und} \qquad T \subseteq T' \ .$$

2. Gilt darüber hinaus für alle Formeln ϕ über Σ:

   $$T' \models \phi \qquad \text{genau dann, wenn} \qquad T \models \phi$$

   dann ist $\mathcal{T}(T')$ eine *konservative* Erweiterung von $\mathcal{T}(T)$.

Hinweis: In einer konservativen Erweiterung $\mathcal{T}(T')$ ist jede Formel ϕ über Σ, die in $\mathcal{T}(T')$ allgemeingültig ist, auch in $\mathcal{T}(T)$ allgemeingültig.

a. Wir definieren die folgenden Signaturen:

$$\Sigma = \{P\} \qquad\qquad \Sigma' = \{P, Q\}.$$

In welchen der beiden folgenden Fälle ist $\mathcal{T}(T')$ (über Σ') eine konservative Erweiterung von $\mathcal{T}(T)$ (über Σ)? Begründen Sie Ihre Antwort.

i. $T_1 = \emptyset$ und $T_1' = \{Q\}$

ii. $T_2 = \emptyset$ und $T_2' = \{Q,\ P \leftrightarrow Q\}$

$\mathcal{T}(T_1')$ ist eine konservative Erweiterung von $\mathcal{T}(T_1)$, denn die einzigen zusätzlichen Formeln über Σ', die nicht auch schon der alten Signatur ableitbar sind, sind Formeln, die äquivalent zu Q sind, sowie Tautologien, die Q enthalten – also keine Formeln über der kleineren Signatur Σ. $\mathcal{T}(T_2')$ ist keine konservative Erweiterung von $\mathcal{T}(T_2)$, denn die Formel P, die eine Formel über der kleineren Signatur Σ ist, ist aus der neuen Theorie ableitbar aber nicht aus der alten.

MUSTERLSG

Fortsetzung 2 Konservative Erweiterungen von Theorien

b. Für prädikatenlogische Formeln ist das Konzept der konservativen Erweiterung analog definiert.

Sei nun O die Theorie partieller Ordnungen über der Signatur, die nur $<$ enthält, also

$$O = \mathcal{T}\big(\{\forall x\, \neg(x < x),\ \forall x\, \forall y\, \forall z\, ((x < y \wedge y < z) \rightarrow x < z\}\big)$$

Sei O' eine Erweiterung von O über einer Signatur mit einer zusätzlichen Konstante max, in der axiomatisiert ist, dass max ein maximales Element bzgl. $<$ ist.

Begründen Sie, warum O' keine konservative Erweiterung von O ist.

Die Formel $\exists x \forall y \neg(x < y)$, die sagt, dass es ein maximales Element gibt, ohne max zu enthalten, ist in der neuen Theorie ableitbar aber nicht in der alten.

c. Zeigen Sie:

Ist T eine vollständige Theorie, dann ist jede Erweiterung T' von T konservativ.

Hinweise:

- Eine Theorie heißt vollständig, wenn für alle Formeln ϕ über Σ gilt:
  $T \models \phi$ oder $T \models \neg\phi$
- Per Definition dürfen Theorien nicht inkonsistent sein ($T \not\models \mathbf{0}$).

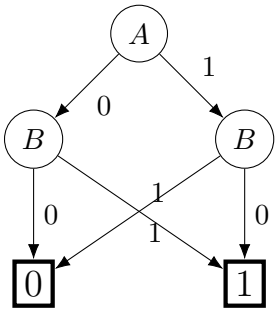
Wäre die Erweiterung nicht konservativ, dann müsste es eine Formel ϕ über Σ geben mit $T' \models \phi$ und $T \not\models \phi$. Da T vollständig ist, müsste dann $T \models \neg\phi$ gelten. Da T' eine Erweiterung von T ist, gilt dann zudem $T' \models \neg\phi$. Aus T' folgt also sowohl ϕ als auch $\neg\phi$. Daher ist T' inkonsistent, was aber per Definition von Theorien ausgeschlossen ist.

MUSTERLSG

3 Shannongraph (1 + 4 = 5 Punkte)

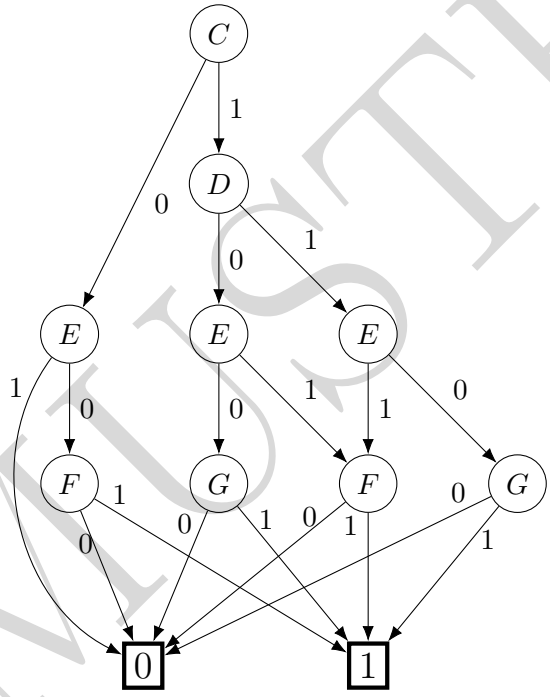
Diese Aufgabe beschäftigt sich mit Shannongraphen (auch Binary Decision Diagrams genannt).

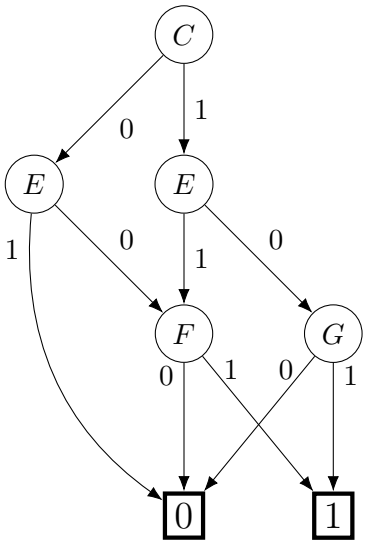
a. Geben Sie an welche Formel der folgende Shannongraph darstellt.



$\neg (A \leftrightarrow B)$

b. Unten sehen Sie einen Shannongraphen. Geben Sie den zugehörigen reduzierten Shannongraphen an.





MUSTERLSG

4 Formalisieren in Prädikatenlogik (PL1) (2 + 2 + 3 = 7 Punkte)

Im Folgenden formalisieren und analysieren wir ein vereinfachtes, modulares, service-orientiertes Komponentensystem mithilfe der sortierten Prädikatenlogik erster Stufe.

Wir verwenden eine mehrsortige Signatur Σ mit Prädikatensymbolen $\{p, r, q\}$ mit den zwei Sorten C (*Komponente*) und S (*Dienst*).

Zur Auswertung der Formeln werden nur solche Interpretationen über Σ verwendet, in denen

- das Universum $U = U^C \uplus U^S$ ausschließlich eine Menge U^C von Komponenten und eine disjunkte Menge U^S von Diensten umfasst;
- $p(x^C, y^S) : C \times S$ genau dann wahr ist, wenn die Komponente x^C den Dienst y^S bereitstellt (*provides*);
- $r(x^C, y^S) : C \times S$ genau dann wahr ist, wenn die Komponente x^C den Dienst y^S benötigt (*requires*);
- $q(x^S) : S$ genau dann wahr ist, wenn der Dienst s aktiv ist (*quality* of service).

Hinweis: Nutzen Sie v, w, x, y, z als Variablen und annotieren Sie, wie in der Vorlesung die Sorte der jeweiligen Variable (bspw. x^S für eine Variable x von Sorte S).

Formalisieren Sie die folgenden Sachverhalte in sortierter Prädikatenlogik mit Gleichheit über Σ.

a. Wenn ein von einer Komponente bereitgestellter Dienst aktiv ist, dann müssen alle von dieser Komponente benötigten Dienste ebenfalls aktiv sein.

$$\forall x^C\ \forall y^S\ \forall z^S\ \big(\ \big(p(x^C, y^S) \land q(y^S) \land r(x^C, z^S)\big)\ \rightarrow\ q(z^S)\big)$$

b. Jeder von einer Komponente benötigte Dienst muss von mindestens einer *anderen* Komponente bereitgestellt werden.

$$\forall x^C\ \forall y^S\ \big(r(x^C, y^S)\ \rightarrow\ \exists z^C\ (\neg\ \big(z^C \doteq x^C\big) \land p(z^C, y^S))\big)$$

c. Gegenseitige Dienstabhängigkeiten zwischen zwei unterschiedlichen Komponenten sind nicht erlaubt.

$$\forall x^C\ \forall y^C\ \neg\Big(\neg\ \big(x^C \doteq y^C\big) \land \exists z^S\ (r(x^C, z^S) \land p(y^C, z^S)) \land \exists z^S\ (r(y^C, z^S) \land p(x^C, z^S))\Big)$$

MUSTERLSG

5 Resolutionskalkül (4 + 6 = 10 Punkte)

a. Betrachten Sie die Folgerung

$$(\forall x \forall y\ r(x,y) \rightarrow \neg r(y,x))\ ,\ (\exists z \forall x\ r(z,x)) \quad \models \quad (\exists z\ \forall x\ \neg r(x,z))$$

Beachten Sie, dass x, y, z Variablen sind.

Formulieren Sie für die obige Folgerung eine Klauselmenge in konjunktiver Normalform, sodass Sie mit dem Resolutionskalkül einen Widerspruchsbeweis führen können, der die Folgerung beweist. Falls Sie neue Funktionssymbole oder Konstanten einführen, geben Sie dies explizit an.

$\{\neg r(x,y), \neg r(y,x)\}, \{r(c,x)\}, \{r(f(z),z)\}$

b. Beweisen Sie die Unerfüllbarkeit der folgenden Klauselmenge mit dem Resolutionskalkül. Geben Sie alle notwendigen Substitutionen an. Beachten Sie, dass u, v, w, x, y, z Variablen sind und c, d Konstante sind.

$$\{\neg r(x,y), \neg r(y,z), r(x,z)\}, \quad \{\neg r(v,w), r(w,v)\}, \quad \{r(c,d)\}, \quad \{r(d,c)\}, \quad \{\neg r(u,u)\}$$

$$\dfrac{\dfrac{\dfrac{\neg r(u,u) \quad \neg r(x,y) \vee \neg r(y,z) \vee r(x,z)}{\neg r(u,y) \vee \neg r(y,u)} \quad r(c,d)}{\neg r(d,c)} \quad r(d,c)}{\square}$$

Erste Substitution: $\{x/u, z/u\}$ Zweite Substitution: $\{u/c, y/d\}$ Dritte Substitution: $\emptyset$

MUSTERLSG

6 Spezifikation mit der Java Modeling Language (JML)

(5 + (3 + 2) = 10 Punkte)

a. Geben Sie in natürlicher Sprache wieder, was der folgende JML-Methodenvertrag für die Methode `m` aussagt.

```
public final class A {
    /*@ normal_behavior
      @  requires (\forall int i; 0 <= i < a.length; a[i] == 0 || a[i] == 1);
      @  requires 0 <= start < a.length;
      @  ensures 0 <= \result;
      @  ensures 0 <= start + \result <= a.length;
      @  ensures (\forall int i; start <= i < start + \result; a[i] == a[start]);
      @  ensures start + \result < a.length ==> a[start + \result] != a[start];
      @  assignable \nothing;
      @*/
    static int m(int[] a, int start) { ... }
}
```

Wenn die Methode `m` mit einem int-Array aufgerufen wird, dessen Einträge alle 0 oder 1 sind, sowie mit dem Parameter start zwischen 0 (inklusive) und der Länge des Arrays (exklusive), terminiert die Methode ohne Exception und verändert keine existierenden Speicherstellen auf dem Heap. Nach Ausführung der Methode gilt: Das Erbebnis der Methode ist die maximale Anzahl an Stellen im Array beginnend an Position start, die ununterbrochen denselben Wert haben. Das muss damit ein Wert zwischen 0 und a.length, jeweils inklusive, sein. Außerdem ist die nächste Stelle im Array (`a[start + \result]`) ungleich `a[start]`, sofern dieser Index innerhalb des Arrays liegt.

MUSTERLSG

Fortsetzung 6 Spezifikation mit der Java Modeling Language

b. Gegeben sei der folgende Auszug aus der Implementierung eines gerichteten Graphen. Ein Graph (Klasse `Graph`) speichert dabei seine Knoten (Klasse `Knoten`) als Array ohne Duplikate.

Außerdem soll jeder Graph ein Array seiner starken Zusammenhangskomponenten (SCCs) speichern. Sie können dabei davon ausgehen, dass diese bereits korrekt berechnet wurden. Eine SCC enthält dabei eine Referenz auf den zugehörigen Graphen sowie ein Array der Knoten der SCC.

Für die Modellierung stehen ihnen zwei Methoden zur Verfügung, die `pure` sind und deshalb auch in der Spezifikation verwendet werden dürfen:

- Die Methode `in(Knoten n)` gibt genau dann `true` zurück, wenn der gegebene Knoten sich im aktuellen SCC (`this`) befindet.
- Die **statische** Methode `erreichbar(Knoten ziel, Knoten start)` gibt genau dann `true` zurück, wenn der übergebene Knoten `ziel` von `start` aus in beliebig vielen Schritten erreichbar ist. Jeder Knoten gilt dabei als von sich selbst aus erreichbar.

Formulieren Sie folgende Objektinvarianten der Klassen `Graph` und `SCC`:

i. „Der Graph ist partitioniert in seine starken Zusammenhangskomponenten, jeder Knoten ist also in genau einer SCC enthalten.“

ii. „Jede SCC ist stark zusammenhängend, d.h. innerhalb der SCC ist jeder Knoten von jedem anderen aus erreichbar.“

Hinweise: Alle Felder und Methoden sind `public`, zur besseren Lesbarkeit sind die Sichtbarkeits-Modifikatoren weggelassen.

Der Code befindet sich auf der nächsten Klausurseite

MUSTERLSG

```
class Knoten {
  // ...
}
```

Invariante der Klasse `Graph`:

```
class Graph {
  Knoten[] knoten;
  SCC[] sccs;

  // i. Der Graph ist partitioniert in seine starken Zusammenhangskomponenten,
  //    jeder Knoten ist also in genau einer SCC enthalten.
  /*@ invariant (\forall int i; 0 <= i < knoten.length;
    @                 (\exists int j; 0 <= j < sccs.length;
    @                       sccs[j].in(knoten[i])
    @                    && !(\exists int k; 0 <= k < sccs[i].sccKnoten.length;
    @                           j != k && sccs[k].in(knoten[i]))));
    @*/
  // ...
}
```

Invariante der Klasse `SCC`:

```
class SCC {
  Graph g;
  Knoten[] sccKnoten;

  // ii. Jede SCC ist stark zusammenhängend,
  //     d.h. innerhalb der SCC ist jeder Knoten von jedem anderen aus erreichbar.
  /*@ invariant (\forall int i; 0 <= i < sccKnoten.length;
    @                 (\forall int j; 0 <= j < sccKnoten.length;
    @                    erreichbar(sccKnoten[i], sccKnoten[j])));
    @*/


  /*@ pure @*/ boolean in(Knoten n) { /* ... */ }

  /*@ pure @*/ static boolean erreichbar(Knoten ziel, Knoten start) { /* ... */ }
  // ...
}
```

MUSTERLSG

7 Lineare Temporale Logik (LTL)

((3 + 0,5 + 0,5) + 4 = 8 Punkte)

Ein geheimnisvoller Zauberer bewacht eine uralte Schatztruhe mit zwei Knöpfen und zwei Lichtern.

Nutzen Sie für die Bearbeitung der folgenden Teilaufgaben die Signatur $\Sigma = \{B, P, S, O\}$. Zur Auswertung der Formeln werden nur solche Strukturen über Σ verwendet in denen die Variablen die folgende Bedeutung haben:

B: Der blaue Knopf wird gedrückt

P: Der pinke Knopf wird gedrückt

S: Das Sternlicht leuchtet

O: Das Kreislicht leuchtet

a. i. Entschlüsseln Sie den Zauber

Der Zauberer verkündet:

„Um die Truhe zu öffnen, musst du die Knöpfe genau in der folgenden Sequenz betätigen, andernfalls bleibt sie für immer verschlossen!“

$$B \wedge \mathbf{X}\Big(P \wedge \Diamond\big(S \wedge \mathbf{X}(B \,\mathbf{U}\, O)\big)\Big)$$

Beschreiben Sie in natürlicher Sprache mit welcher Abfolge von Aktionen die Truhe geöffnet werden kann.

Drücke zunächst den blauen Knopf und im nächsten Schritt den pinken Knopf. Zu einem späteren Zeitpunkt wird das Sternlicht leuchten, und ab dem darauffolgenden Schritt muss der blaue Knopf in jedem Schritt gedrückt bleiben, bis das Kreislicht eingeschaltet wird.

ii. Formulieren Sie die folgenden Anforderungen in LTL

- Das Sternlicht wird irgendwann eingeschaltet und bleibt danach für immer eingeschaltet.

$\Diamond\Box\, S$

- Es dürfen zu keinem Zeitpunkt beide Knöpfe gleichzeitig gedrückt werden.

$\Box\neg(B \wedge P)$

MUSTERLSG

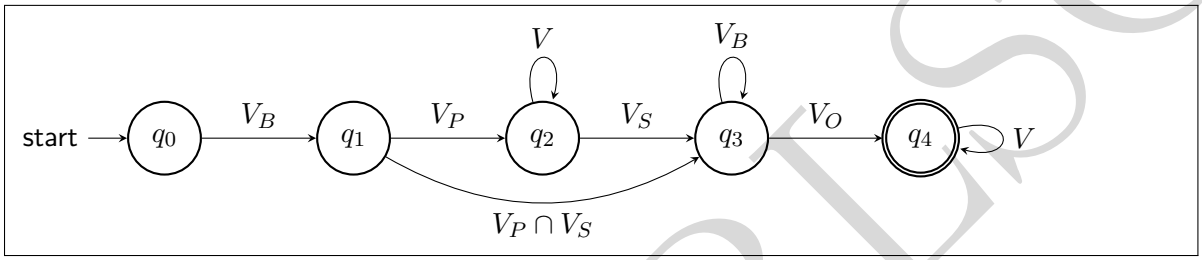
Fortsetzung 7 Lineare Temporale Logik (LTL)

b. Geben Sie einen nicht-deterministischen Büchi-Automaten an, dessen akzeptierte Sprache genau der Menge der erfüllenden Strukturen der Formel $B \wedge \mathbf{X}\Big(P \wedge \Diamond\big(S \wedge \mathbf{X}(B \,\mathbf{U}\, O)\big)\Big)$ entspricht.

Für das Vokabular $V = \mathcal{P}(\Sigma)$ (die Potenzmenge von Σ) werden die folgenden Abkürzungen definiert:

$$V_B = \{X \in V | B \in X\} \qquad V_P = \{X \in V | P \in X\}$$
$$V_S = \{X \in V | S \in X\} \qquad V_O = \{X \in V | O \in X\}$$

Sie dürfen an den Kanten des Automaten auch Mengen-Ausdrücke wie $V \setminus V_P$ verwenden.



MUSTERLSG