Technische Universität Berlin
Institut für Softwaretechnik und Theoretische Informatik
FG Softwaretechnik
Ernst-Reuter-Platz 7
10587 Berlin

Jähnichen
Dobrev
Mehlhase
Rein

MPGI 3

Muster-Schriftliche Leistungskontrolle B

Wintersemester 2009/2010
18. Februar 2010

Prüfen Sie zunächst, ob dieses Exemplar vollständig ist (**11** einseitig bedruckte Blätter).

Tragen Sie auf diesem Titelblatt und darüber hinaus auf allen Blättern, die Sie für Ihre Niederschrift verwenden, Ihren Namen und Ihre Matrikelnummer ein. Zum Bestehen der Prüfungsleistung „Klausur“ müssen Sie in den beiden schriftlichen Leistungskontrollen insgesamt mindestens **50 Punkte** erreichen. Darüber hinaus müssen Sie in jeder der beiden schriftlichen Leistungskontrollen mindestens je 10 Punkte erreichen. Insgesamt sind in der vorliegenden schriftlichen Leistungskontrolle 50 Punkte möglich. Die Bearbeitungszeit beträgt 75 Minuten. Bevor die Bearbeitungszeit anfängt, haben Sie 15 Minuten Einlesezeit. In der Klausur sind außer eines beschriebenen DIN-A4-Blattes **keine Hilfsmittel** zugelassen. Verwenden Sie ausschließlich das ausgeteilte Klausur-Papier. Es dürfen nur **permanent-schwarze oder -blaue Stifte** zum Lösen der Aufgaben verwendet werden.

Viel Erfolg!

Name, Vorname: _____

Matrikelnummer: _____

Studienrichtung: _____

MPGI3-Übungen habe ich im _____ besucht.
(z.B. WS 2009/10)

Aufgabe	1a	1b	2	Gesamt
Maximal:	16	19	15	50
Erreicht:				

Aufgabe 1 (Object-Z) - UNO

35 Punkte

Aufgabenstellung

Eine vereinfachte Version eines UNO-Kartenspiels ist mit Hilfe von Object-Z zu spezifizieren – dafür sind ausschließlich die vorgegebenen Klassen und deren Operationen zu erweitern. Beachten Sie, dass die Spielregeln der Einfachheit halber angepasst wurden. *Es sind exakt die hier angegebenen Regeln zu spezifizieren. Dabei dürfen den vorgegebenen Klassen keine neuen Attribute und keine neuen Operationen hinzugefügt werden.*

Kurzbeschreibung

Bei diesem UNO-Spiel gibt es Karten in vier verschiedenen Farben (*rot, grün, blau, gelb*), mit den Zahlen 1-10 (in jeder Farbe). Karten können mehrfach im Spiel vorkommen und die Gesamtanzahl der Karten ist **nicht** vorgegeben. Grundsätzlich liegt eine Karte offen auf dem Tisch und der Spieler, der an der Reihe ist, kann eine Handkarte ausspielen, wenn sie die gleiche Farbe oder die gleiche Zahl hat, wie die liegende Karte. Spieler, die eine Karte mit der gleichen Farbe *und* der gleichen Zahl haben, können diese jederzeit (also auch, wenn sie nicht an der Reihe sind) einwerfen. Der Spieler, der als erster keine Handkarten mehr hat, hat gewonnen. Falls der Spieler, der an der Reihe ist, keine Karte ausspielen kann, muss er eine Karte ziehen und kann diese danach, falls möglich, noch ausspielen. Nachdem ein Spieler *eine* Karte gelegt hat, ist immer der auf diesen Spieler folgende Spieler dran.

Folgende Grafik gibt einen Überblick über die möglichen Zugvarianten:

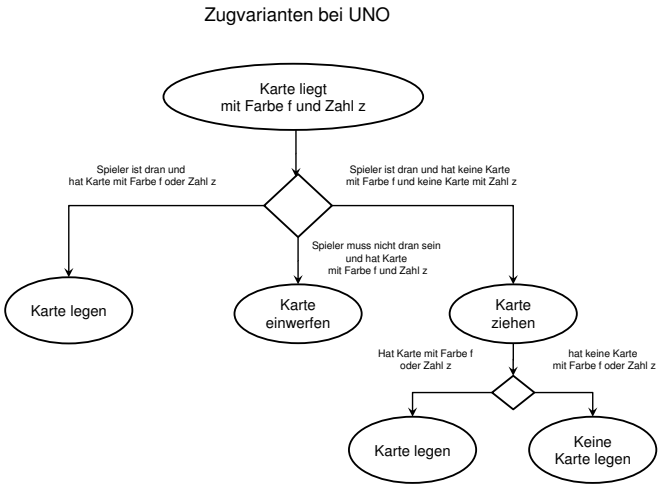


Abbildung 1: Mögliche Zugvarianten im Uno-Spiel

a) Spieler

16 Punkte

Jeder Spieler hat eine variable Anzahl von Karten auf der *hand* – anfangs acht. *Hinweis:* Ob ein Spieler an der Reihe ist, wird von der (später beschriebenen) Klasse *Spiel* bestimmt und nicht vom *Spieler* selbst.

Die Operation *Karte_auslegen* bekommt die liegende Karte als Eingabeparameter und beschreibt, wie ein Spieler eine Karte auslegt – dazu muss er noch Karten besitzen. Um eine Karte auslegen zu können, muss der Spieler die als Eingabeparameter übergebene Karte mit seinen Karten vergleichen. Hat er eine Karte auf der Hand, die die gleiche Farbe oder die gleiche Zahl hat, so muss er diese Karte auslegen (Rückgabewert). Eine weitere Rückgabe dieser Operation ist der Spieler selbst. Die Nachricht *Karte_ausgelegt* informiert darüber, dass eine neue Karte ausgelegt wurde.

Eine weitere Operation *Karte_auslegen_nicht_möglich* bekommt ebenfalls die liegende Karte als Eingabeparameter und modelliert, dass ein Spieler, der noch Karten auf der Hand hat, keine dieser Karten auslegen kann. Das heißt, der Spieler hat keine Karte, die die gleiche Farbe oder die gleiche Zahl hat. Ausgabe der Operation sind die Nachricht *keine_Karte_ausgelegt*, der Spieler selbst und die Karte, die sich bereits vor dem Spielzug auf dem Tisch befand.

Die Operation *Karte_ziehen* bekommt ebenfalls die liegende Karte als Eingabe und beschreibt, dass ein Spieler, der noch Karten auf der Hand hat, keine dieser Karten auslegen kann (ebenfalls aus o.g. Gründen) und eine Karte zieht (diese Karte wird neu erstellt und seiner *hand* hinzugefügt).

Die kombinierte Operation *Spielzug_durchführen* drückt aus, dass ein Spieler einen kompletten Spielzug durchführt: Entweder legt er eine Karte aus, oder er zieht eine Karte und hat danach noch die Chance, die neu erhaltene Karte auszuspielen (falls möglich). Siehe Abbildung 1 zur Verdeutlichung, wobei das Einwerfen von Karten in dieser Operation nicht enthalten ist.

Eine Operation *Karte_einwerfen* modelliert den Vorgang, dass ein Spieler eine Karte auf der Hand hat, die exakt (Farbe **und** Zahl) der Karte auf dem Tisch (Eingangsparameter) entspricht. Diese Karte kann er jederzeit auslegen (Rückgabewert). Außerdem wird der Spieler selbst und eine entsprechende Nachricht (*Karte_eingeworfen*) ausgegeben.

Globale Definitionen:

Nachricht ::= *Karte_ausgelegt*
| *keine_Karte_ausgelegt*
| *Karte_eingeworfen*
Farbe ::= *rot* | *grün* | *blau* | *gelb*
Zahl ::= 1..10

Karte
farbe : *Farbe*
zahl : *Zahl*

Spieler

$\upharpoonright (INIT, hand, Karte\_einwerfen, Spielzug\_durchführen)$

$hand : \mathbb{F}\, Karte$

INIT

$\#hand = 8$

Karte_auslegen

$\Delta(hand)$
$liegt? : Karte$
$auslegen! : Karte$
$spieler! : Spieler$
$n! : Nachricht$

$\#hand > 0 \quad [REDUNDANT]$
$\exists x : hand \mid x.farbe = liegt?.farbe \lor x.zahl = liegt?.zahl \bullet auslegen! = x$
$hand' = hand \setminus \{auslegen!\}$
$spieler! = self$
$n! = Karte\_ausgelegt$

Karte_auslegen_nicht_möglich

$liegt? : Karte$
$auslegen! : Karte$
$spieler! : Spieler$
$n! : Nachricht$

$\#hand > 0$
$\forall x : hand \bullet x.farbe \neq liegt?.farbe \land x.zahl \neq liegt?.zahl$
$auslegen! = liegt?$
$spieler! = self$
$n! = keine\_Karte\_ausgelegt$

Karte_ziehen

$\Delta(hand)$
$liegt? : Karte$

$\#hand > 0$
$\forall x : hand \bullet x.farbe \neq liegt?.farbe \land x.zahl \neq liegt?.zahl$
$\exists k : Karte \mid k \notin hand \bullet hand' = hand \cup \{k\}$
$(alternativ : \#hand' = \#hand + 1 \land hand \subset hand')$

$Spielzug\_durchführen \mathrel{\widehat{=}} Karte\_auslegen [] (Karte\_ziehen^{\circ}_{9}$
$(Karte\_auslegen\_nicht\_möglich [] Karte\_auslegen))$

Karte_einwerfen

$\Delta(hand)$
$liegt? : Karte$
$auslegen! : Karte$
$spieler! : Spieler$
$n! : Nachricht$

$\#hand > 0 \quad [REDUNDANT]$
$\exists x : hand \mid x.farbe = liegt?.farbe \land x.zahl = liegt?.zahl \bullet auslegen! = x$
$hand' = hand \setminus \{auslegen!\}$
$spieler! = self$
$n! = Karte\_eingeworfen$

b) UNO-Spiel

19 Punkte

In dieser Klasse wird das eigentliche UNO-Spiel modelliert.

Zu einem UNO-Spiel gehören eine variable Anzahl von Spielern, die in einer festen *reihenfolge* an der Reihe sind. Die partielle Abbildung *reihenfolge* weist **jedem** Spieler genau **eine** Nummer (beginnend bei 0 und in Einerschritten aufsteigend) zu.

Beispiel-Reihenfolge:
$0 \mapsto$ *Spieler1*
$1 \mapsto$ *Spieler2*

Während des Spiels *liegt* immer genau eine Karte auf dem Tisch. Das Attribut *dran* speichert die Nummer des Spielers, der gerade an der Reihe ist. Ein boolesches Attribut *beendet* zeigt an, ob das Spiel beendet ist.

Initial liegt eine beliebige rote Karte auf dem Tisch und alle Spieler befinden sich im initialen Zustand. Der erste Spieler (mit der Nummer 0) beginnt immer das Spiel. Natürlich ist das Spiel anfangs nicht *beendet*.

Die Operation *liegende_Karte* liefert die auf dem Tisch liegende Karte zurück, unter der Voraussetzung, dass das Spiel noch nicht *beendet* ist.

Im Gegensatz dazu ersetzt die Operation *Karte_ersetzen* die auf dem Tisch liegende Karte durch eine übergebene Karte, falls das Spiel noch nicht *beendet* ist.

Falls das Spiel noch nicht *beendet* ist, wird in der Operation *nächster_Spieler* das Attribut *dran* auf die Nummer des nächsten Spielers gesetzt. Dafür bekommt diese Operation als Eingabe einen Spieler, der an dem Spiel teilnimmt und noch Karten auf der Hand hat. Der in der Reihenfolge nach ihm folgende Spieler ist der Spieler, der als nächster *dran* ist. Nach dem letzten Spieler ist wieder der erste Spieler (mit der Nummer 0) an der Reihe.

Spiel_beenden ist die Operation, die das Ende des Spiels beschreibt. Die Operation bekommt einen Spieler, der keine Karten mehr auf der Hand hat, als Eingabeparameter und setzt das Attribut *beendet* von false auf true.

In der kombinierten Operation *Spielzug_abschliessen* wird zuerst die auf dem Tisch liegende Karte durch eine übergebene ersetzt und danach ist entweder der nächste Spieler dran oder das Spiel wird beendet.

Die kombinierte Operation *Spielzug* modelliert, dass der Spieler, der *dran* ist, die auf dem Tisch liegende Karte bekommt und dann einen Spielzug durchführt oder, dass ein (beliebiger) Spieler eine Karte einwirft. Danach wird der Spielzug abgeschlossen, also zuerst die auf dem Tisch liegende Karte ersetzt und danach ist der nächste Spieler dran oder das Spiel beendet.

UNO_Spiel

$$
\begin{array}{l}
reihenfolge : \mathbb{N} \rightarrowtail Spieler \\
liegt : Karte \\
dran : dom\ reihenfolge \\
beendet : \mathbb{B} \\
\hline
\forall n : \mathbb{N} \bullet n < \#reihenfolge \Rightarrow n \in \mathsf{dom}\ reihenfolge \\
(alternativ : \mathsf{dom}\ reihenfolge = 0..\#reihenfolge - 1)
\end{array}
$$

INIT

$$
\begin{array}{l}
liegt.farbe = rot \\
\forall sp : \mathsf{ran}\ reihenfolge \bullet sp.INIT \\
dran = 0 \\
\neg beendet
\end{array}
$$

liegende_Karte

$$
\begin{array}{l}
liegt! : Karte \\
\hline
\neg beendet \\
liegt! = liegt
\end{array}
$$

Karte_ersetzen

$$
\begin{array}{l}
\Delta(liegt) \\
auslegen? : Karte \\
\hline
\neg beendet \\
liegt' = auslegen?
\end{array}
$$

nächster_Spieler

$$
\begin{array}{l}
\Delta(dran) \\
spieler? : \mathsf{ran}\ reihenfolge \\
\hline
\#spieler?.hand > 0 \\
\neg beendet \\
\exists n : \mathbb{N} \bullet (\\
\quad reihenfolge\ n = spieler? \wedge \\
\quad n = \#reihenfolge - 1 \Rightarrow dran' = 0 \wedge \\
\quad n < \#reihenfolge - 1 \Rightarrow dran' = n + 1)
\end{array}
$$

Spiel_beenden

$$
\begin{array}{l}
\Delta(beendet) \\
spieler? : Spieler \\
\hline
\neg beendet \\
\#spieler?.hand = 0 \\
beendet'
\end{array}
$$

$$Spielzug\_abschliessen \mathrel{\widehat{=}} Karte\_ersetzen \fatsemi (nächster\_Spieler \Box Spiel\_beenden)$$
$$Spielzug \mathrel{\widehat{=}} (liegende\_Karte \parallel (reihenfolge(dran).Spielzug\_durchführen \Box [alternativ : \Box\ sp : Spieler \mid reihenfolge(dran) = sp \bullet sp.Spielzug\_durchführen] \Box (\Box\ sp : \mathsf{ran}\ reihenfolge \bullet sp.Karte\_einwerfen)) \fatsemi Spielzug\_abschliessen$$

Aufgabe 2 (Statecharts) Kaffeevollautomat

15 Punkte

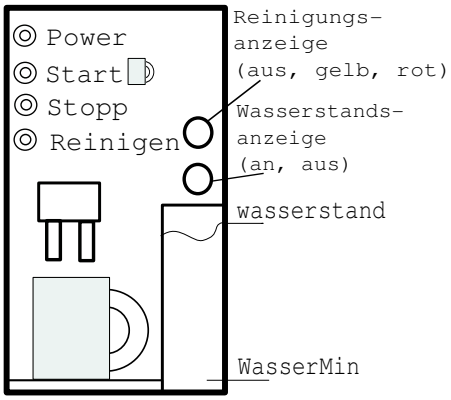
Aufgabenstellung

Modellieren Sie den im Folgenden beschriebenen Kaffeevollautomaten mit den Komponenten *Reinigungsstandanzeige* und *Wasserstandanzeige*, sowie die *Steuerung* dieser Komponenten mittels Statecharts. Verwenden Sie bei Ihrer Modellierung die definierten Eingabeereignisse und Variablen der unten stehenden Tabelle. Falls Sie interne Ereignisse oder weitere Variablen definieren sind diese in der Tabelle zu ergänzen.

Statechart-Lösung:

NAME	BESCHREIBUNG
EXTERNE EINGABEEREIGNISSE	
power	Die Taste Power wurde gedrückt.
rein	Die Taste Reinigen wurde gedrückt.
start	Die Taste Start wurde gedrückt.
stopp	Die Taste Stopp wurde gedrückt.
INTERNE EREIGNISSE	
sauber	Die Maschine wurde gereinigt und ist nun sauber.
VARIABLEN	
wasserstand	Der aktuelle Wasserstand. (**nat**)
WasserMin	Konstante, die den minimalen Wasserstand kennzeichnet. (**nat**)
anzahl	Anzahl der Kaffees ohne Reinigung. (**nat**)
ub	Kaffeezubereitung wurde bereits unterbrochen. (**bool**)

Beschreibung



Die Kaffeemaschine hat die Tasten *Power*, *Start*, *Stopp* und *Reinigen*. Zusätzlich besitzt sie zwei Warnleuchten, um Informationen über ihren Zustand anzuzeigen. Die Warnleuchte *Reinigungsstandanzeige* kann entweder *aus* sein, *gelb* oder *rot* leuchten. Die Warnleuchte *Wasserstandanzeige* wird nur angeschaltet, wenn der Wasserstand das Minimum *WasserMin* unterschreitet.

Die Taste *Power* dient dazu, die Kaffeemaschine ein- bzw. auszuschalten. Bevor die Maschine ausgeht, startet automatisch ein *Reinigungsprogramm*, falls seit der letzten Reinigung bereits Kaffee zubereitet wurde. Ansonsten geht die Maschine direkt *aus*. Das *Reinigungsprogramm* nimmt eine Minute in Anspruch. Zu beachten ist, dass die Maschine nur ausgeschaltet werden kann, wenn sie eingeschaltet ist und zu diesem Zeitpunkt keinen Kaffee zubereitet.

Ist die Maschine eingeschaltet, so kann entweder Kaffee zubereitet werden (Taste *Start*) oder durch Drücken der Taste *Reinigen* das *Reinigungsprogramm* gestartet werden. Nach beiden Aktionen kehrt die Maschine in den vorherigen Zustand zurück. Die Maschine kann nur Kaffee zubereiten, falls der Wassertank noch genügend Wasser enthält und seit der letzten Reinigung nicht bereits 80 Tassen Kaffee zubereitet wurden.

Beim Zubereiten eines Kaffees wird zunächst für 10 Sekunden der Kaffee gemahlen. Dann wird dem Kaffeepulver 10 Sekunden lang Wasser hinzugefügt. Im Anschluss daran wird der Kaffe ausgegeben, was 15 Sekunden in Anspruch nimmt. Schließlich wird das genutzte Kaffeepulver automatisch entfernt – dieser Vorgang benötigt 5 Sekunden. Die Zubereitung kann jederzeit mit der Taste *Stopp* unterbrochen werden. Nach 20 Sekunden oder durch Drücken der *Start* Taste wird die Zubereitung (an dem Punkt, an dem sie vorher unterbrochen wurde) fortgesetzt. Ein erneutes Unterbrechen dieses Vorganges ist nicht möglich.

Die Anzahl der nach einer Reinigung zubereiteten Kaffees wird von der Maschine gespeichert. Nach der 50. Tasse Kaffee ohne Reinigung beginnt die Lampe *Reinigungsaufforderung gelb* zu leuchten. Nachdem eine Reinigung durchgeführt wurde geht die Lampe wieder aus. Nach der 80. Tasse Kaffee ohne Reinigung wechselt die Warnleuchte *Reinigungsaufforderung* von *gelb* auf *rot*.

Sinkt der Wasserstand im Tank unter das gegebene Minimum *WasserMin*, beginnt die Warnleuchte *Wasserstandanzeige* zu leuchten. Die Variable *wasserstand* enthält immer den aktuellen Wasserstand. Sobald der Wasserstand wieder über *WasserMin* liegt, wird die Warnleuchte deaktiviert – dann ist natürlich die Zubereitung von Kaffee wieder möglich.

Statechart-Lösung: