KV Software Engineering
Übungsaufgaben SS 2005

Martin Glinz, Silvio Meier,
Nancy Merlo-Schett, Katja Gräfenhain

Übung 4

Aufgabe 1 (8 Punkte) – Programmverifikation

Gegeben ist folgendes Programmfragment mit Vor- und Nachbedingung:

```
var i : integer;

(* precondition:  V ≡ i < -40  OR  i > 25 *)

if i < 0 then begin
   i := (-1) * i;
end;

i := i * 2;

if i >= 10 then begin
   i := i - 10;
else
   i:= i + 100;
end;

(* postcondition: N ≡ i > 12 *)
```

Ist dieses Programmfragment korrekt?

a) Berechnen Sie dazu die schwächste Vorbedingung (Weakest Precondition) gemäss der axiomatischen Semantik die Sie aus der Vorlesung kennen. (5 Punkte)

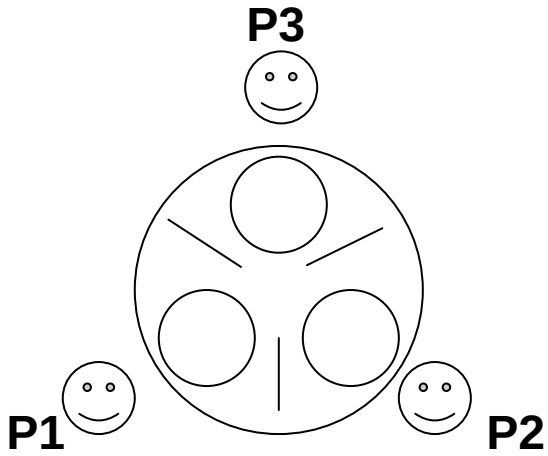
b) Leiten Sie aus Ihrem berechneten Ergebnis den Schluss zur Korrektheit dieses Programmfragments ab. (3 Punkte)

Aufgabe 2 (12 Punkte + 1 Bonuspunkt) – Model Checking

Beim bekannten Problem der „Dining Philosophers“ sitzen mehrere Philosophen um einen runden Tisch, auf dem sich eine Schale Reis befindet. Jeder hat einen Teller vor sich, zwischen zwei Tellern liegt jedoch nur jeweils ein Stäbchen, d.h. die Anzahl der Philosophen entspricht der Anzahl an Stäbchen. Die Philosophen brauchen zum Essen sowohl das linke als auch das rechte Stäbchen. Das heißt, isst ein Philosoph rechts oder links (oder beide) von ihm bereits, muss er warten, bis dieser (oder beide) fertig gegessen haben, bevor er selbst beide Stäbchen zum Essen aufnehmen kann. Wenn ihr Hunger gestillt ist, legen sie die Stäbchen wieder zurück und fangen an zu denken, bis sie wieder hungrig werden und erneut versuchen, die Stäbchen zu bekommen.

Es soll ein formales Modell des Philosophenproblems erstellt und mittels Model Checking verifiziert werden. Der Einfachheit halber nehmen wir an, es seien nur drei Philosophen vorhanden. Die Philosophen werden durch die drei Variablen P1, P2 und P3 repräsentiert und können jeweils drei möglichen Belegungen haben:

Philosoph 1 (P1)	**Philosoph 2 (P2)**	**Philosoph 3 (P3)**
Denkt (d1)	Denkt (d2)	Denkt (d3)
Ist_Hungrig (h1)	Ist_Hungrig (h2)	Ist_Hungrig (h3)
Isst (e1)	Isst (e2)	Isst (e3)



Als erstes sollen zwei Eigenschaften des Problems mit Hilfe von temporaler Logik formalisiert werden.

a) Formulieren Sie die Eigenschaft, dass ein Philosoph nur dann essen kann, wenn weder sein linker noch sein rechter Nachbar essen (1P). Geben Sie an, ob es sich dabei um eine Sicherheits- oder eine Lebendigkeitseigenschaft handelt (1P).

b) Formulieren Sie eine (primitive) Lebendigkeitseigenschaft für drei Philosophen, so dass jeder Philosoph wirklich irgendwann zum Essen kommt, sobald er hungrig ist und ihm nicht die anderen Philosophen ständig die Stäbchen wegnehmen können. (1P)

c) Verallgemeinern Sie die in a) formulierte Ausschlusseigenschaft auf eine Situation mit *n* Philosophen. (1 Bonuspunkt)

Als nächstes soll der Zustandsgraph des Problems erstellt werden. Bei 3 Philosophen und drei Zuständen pro Philosoph ergeben sich so theoretisch $3^3 = 27$ Zustände. Für die Bearbeitung der folgenden Teilaufgabe finden Sie im Anhang eine Arbeitsvorlage, die Sie benutzen können.

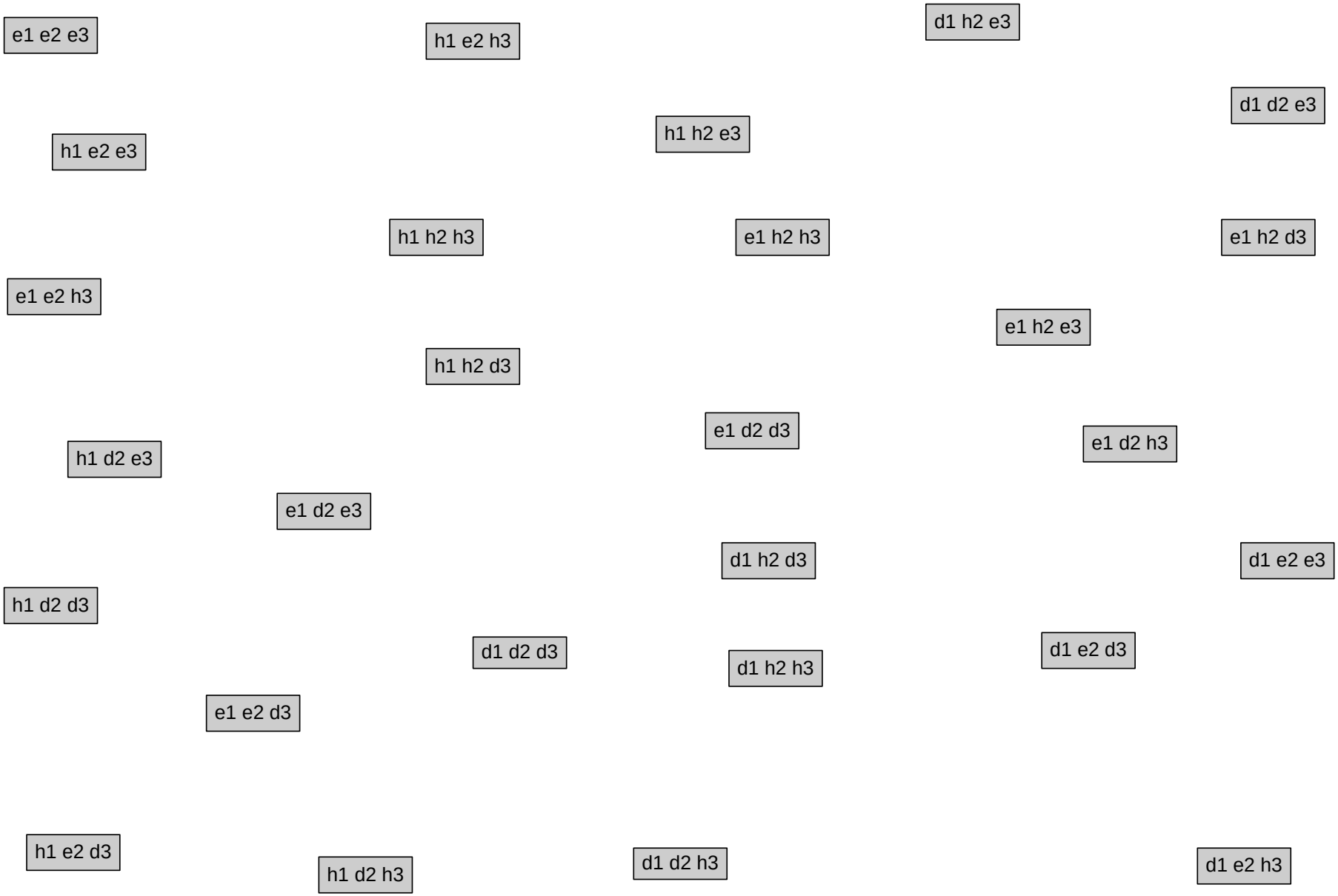
d) Welche Zustände sind aufgrund der unter a) formulierten Eigenschaft verboten? Beschreiben Sie und begründen Sie kurz. Falls Sie die Vorlage aus dem Anhang verwenden, streichen Sie bitte die verbotenen Zustände durch. (1 P).

e) Ermitteln Sie alle möglichen Zustandsübergänge zwischen den nicht verbotenen Zuständen und zeichnen Sie den entsprechenden Zustandsgraphen. Sie können dazu die Vorlage im Anhang benutzen. (4 P)

Als drittes soll eine Eigenschaft des modellierten Problems mit Model Checking verifiziert werden.

f) Führen Sie für die unter b) bestimmte Lebendigkeitseigenschaft ein manuelles Model Checking auf dem unter e) erstellten Zustandsgraphen durch. Falls Sie dabei ein Gegenbeispiel finden (d.h. einen Pfad im Graphen, für den die Lebendigkeitseigenschaft nicht gilt), so geben Sie diesen Pfad an und begründen Sie kurz, warum dies ein Gegenbeispiel ist (2 P).

g) Wie müsste der Graph verändert werden, um die Lebendigkeitseigenschaft erfüllen zu können? Welche Zustände sind davon betroffen? Es ist nicht verlangt, den Graphen tatsächlich neu zu zeichnen! (2 P)


e1 e2 e3
h1 e2 h3
d1 h2 e3
d1 d2 e3
h1 h2 e3
h1 e2 e3
h1 h2 h3
e1 h2 h3
e1 h2 d3
e1 e2 h3
e1 h2 e3
h1 h2 d3
e1 d2 d3
e1 d2 h3
h1 d2 e3
e1 d2 e3
d1 h2 d3
d1 e2 e3
h1 d2 d3
d1 d2 d3
d1 e2 d3
d1 h2 h3
e1 e2 d3
h1 e2 d3
h1 d2 h3
d1 d2 h3
d1 e2 h3