

Aufgabe 1 – Weakest Precondition berechnen

- Wozu? Beweisen der **Korrektheit eines Programmes P**
- **Für Programm gilt $\{V\}P\{N\}$** , d.h. unter der Vorbedingung V gilt bei der Ausführung von P , die Nachbedingung N , wobei P und N prädikatenlogische Ausdrücke sind.
- Programm P besteht aus einer **Sequenz von ausgeführten Anweisungen**, d.h. $P \circ s_1; s_2; s_3; \dots; s_n$
- P muss für eine Verifikation **syntaktisch korrekt** sein!
- Für einen einzelnen Anweisungsschritt gilt $\{Q_{i-1}\}s_i\{Q_i\}$, d.h.
- Für jede ausführbare Anweisung der Programmiersprache, in der das Programm P geschrieben wurde, muss ein Axiom existieren: $\{V\}s\{N\} \rightarrow$ **Axiomatische Semantik**.
- **Berechnung $wp(P, N)$** durch eine Verkettung der Form $wp(s_0, wp(s_1, wp(s_2, \dots, wp(s_n, N))))$ resultiert $wp(P, N) \rightarrow$ Die schwächste Vorbedingung $wp(s, N)$ ermittelt wird, unter der die Ausführung von s (noch) eine korrekte Nachbedingung N liefert.
- Um die **Korrektheit** zu **beweisen** müssen folgende Teilbeweise erbracht werden:
 - Prüfen, ob V **stärker oder gleich stark** wie $wp(P, N)$ formuliert ist
 - Zeigen, dass das **Programm terminiert**, d.h. untersuchen von Schleifen und rekursiven Aufrufen!
- In der Vorlesung wurden ausgewählte Axiome für die Sprache *Pascal* präsentiert: **Zuweisung, Leere Anweisung, Sequenz, Alternative, Iteration**.
 - Zuweisung: $wp(x := E, Q(E)) \equiv E$ berechenbar und $Q(E)$
 - Leere Anweisung: $wp(e, Q) \equiv Q$

- Sequenz: $wp(s1;s2, Q) \equiv wp(s1, wp(s2, Q))$
- Alternative: $wp(\text{if } b \text{ then } s1 \text{ else } s2, Q) \equiv b \text{ und } wp(s1, Q) \text{ oder nicht } b \text{ und } wp(s2, Q)$
- Iteration: $V \rightarrow \text{Inv and } (\text{Inv and not } b) \rightarrow N$
- Lösung der Aufgabe durch Rückwärtseinsetzen:

```
VAR x, y : INTEGER;
(*weakest precondition wp? *)
```

```
x := 3 * y - 2;
IF (x < 12) THEN
    y := 3 * x - 9;
ELSE
    y := x + 6;
```

```
y := y - 2;
```

```
(* postcondition: N  $\equiv 7 \leq y < 25$  *)
```

```
wp(x:=3y-2; IF(x<12) THEN y:=3x-9 ELSE y:=x+6;
y=y-2, 7≤y<25)
```

```
 $\equiv wp(x := 3y-2; IF (x < 12) THEN y=3x-9 ELSE y=x+6, 7 \leq y-2 < 25)$ 
```

```
 $\equiv wp(x := 3y-2; IF (x < 12) THEN y=3x-9 ELSE y=x+6, 9 \leq y < 27)$ 
```

```
 $\equiv wp(x := 3y-2; ((x < 12) \wedge 9 \leq 3x-9 < 27) \vee ((x \geq 12) \wedge 9 \leq x+6 < 27))$ 
```

```
 $\equiv wp(x := 3y-2; ((x < 12) \wedge 18 \leq 3x < 36) \vee ((x \geq 12) \wedge 3 \leq x < 21))$ 
```

```
 $\equiv wp(x := 3y-2; ((x < 12) \wedge 6 \leq x < 12) \vee ((x \geq 12) \wedge 3 \leq x < 21))$ 
```

```
 $\equiv ((3y-2 < 12) \wedge 6 \leq 3y-2 < 12) \vee ((3y-2 \geq 12) \wedge 3 \leq 3y-2 < 21)$ 
```

```
 $\equiv ((3y < 14) \wedge 8 \leq 3y < 14) \vee ((3y \geq 14) \wedge 5 \leq 3y < 23)$ 
```

$$\equiv ((y < 14 / 3) \wedge 8/3 \leq y < 14/3) \vee ((y \geq 14/3) \wedge 5/3 \leq y < 23/3)$$

$$\equiv ((y < 14 / 3) \wedge 8/3 \leq y < 14/3) \vee ((y \geq 14/3) \wedge 5/3 \leq y < 23/3)$$

$$\equiv (8/3 \leq y < 14/3) \vee ((y \geq 14/3) \wedge 5/3 \leq y < 23/3)$$

$$\equiv (8/3 \leq y < 14/3) \vee ((y \geq 14/3) \wedge y < 23/3)$$

$$\equiv \underline{(8/3 \leq y < 23/3)}$$

Frage: Welche Vorbedingung würde diese wp verletzen?

Antwort: ein Wert für y ausserhalb des eben berechneten Wertebereiches

Aufgabe 2 – Model Checking

a) $\equiv \neg((e_1 \wedge e_2) \vee (e_2 \wedge e_3) \vee (e_3 \wedge e_1))$

Es handelt sich um eine Sicherheitseigenschaft.

b) $\equiv ((h_1 \rightarrow \Diamond e_1) \wedge (h_2 \rightarrow \Diamond e_2) \wedge (h_3 \rightarrow \Diamond e_3))$

Formel für n Philosophen (nicht verlangt): $\bigwedge_{i=1,n} (h_i \rightarrow \Diamond e_i)$

c) Beispiel für fünf Philosophen: $\equiv \neg((e_1 \wedge e_2) \vee (e_2 \wedge e_3) \vee (e_3 \wedge e_4) \vee (e_4 \wedge e_5) \vee (e_5 \wedge e_1))$

Allgemein formuliert:

$$\equiv \neg((e_1 \wedge e_2) \vee (e_2 \wedge e_3) \vee \dots \vee (e_{n-1} \wedge e_n) \vee (e_n \wedge e_1))$$

d) 20 Zustände: es fallen alle Zustände weg, in denen zwei oder drei Philosophen gleichzeitig essen. Das sind sieben Stück:

$[e_1 e_2^*]$, $[e_1^* e_3]$, $[^* e_2 e_3]$ und $[e_1 e_2 e_3]$

e) Graph siehe Ende

f) Gegenbeispiel: $[h_1 h_2 d_3]$, $[e_1 h_2 d_3]$, $[d_1 h_2 d_3]$, $[h_1 h_2 d_3]$ verletzt die Lebendigkeitseigenschaft, weil nicht sichergestellt ist, dass Philosoph 2 irgendwann zum Essen kommt, obwohl er Hunger hat.

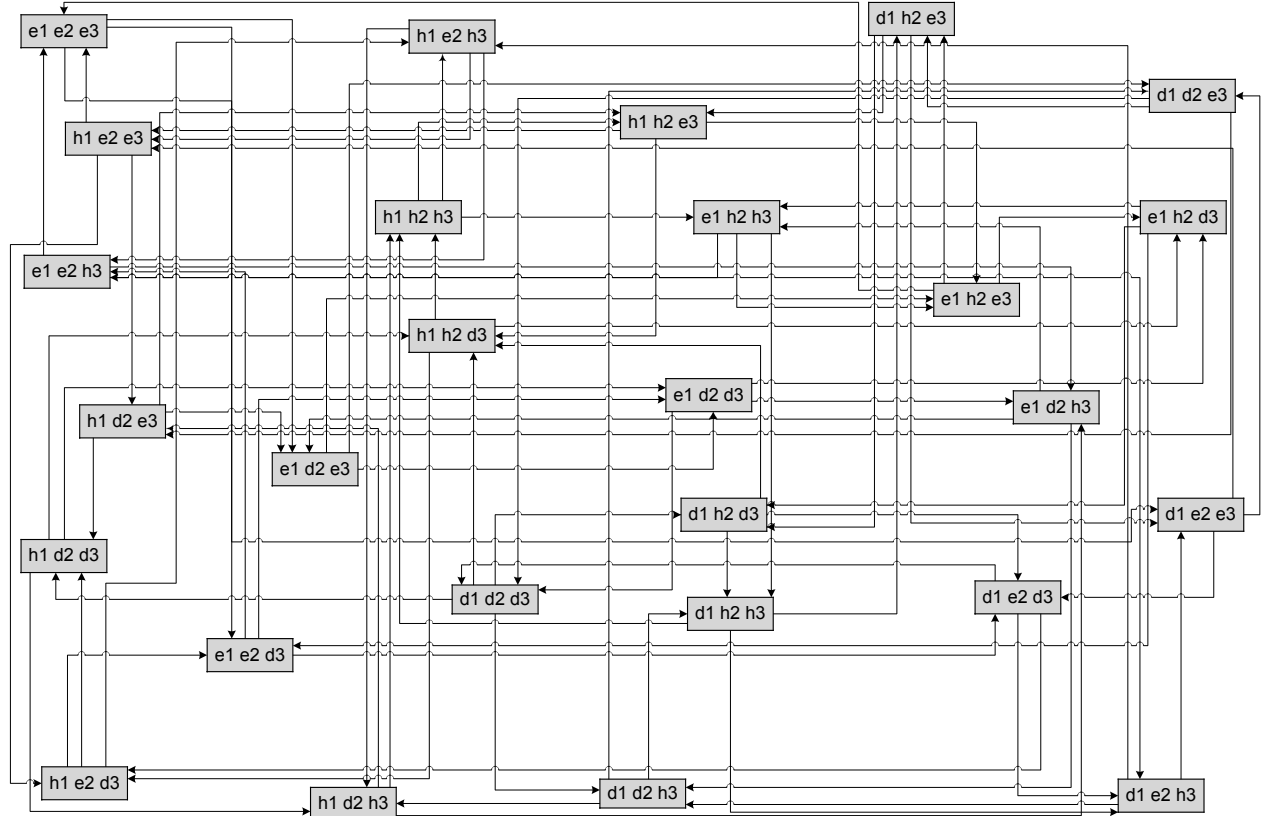
g) Alle Zustände, in denen zwei oder drei Philosophen hungrig sind, also $[h_1 h_2^*]$, $[h_1^* h_3]$, $[^* h_2 h_3]$, $[h_1 h_2 h_3]$ müssen aufgespalten werden, so dass in jedem Zustand die Vorgeschichte abgebildet werden kann, welcher

Philosoph am längsten hungrig ist. Dieser darf dann als nächster essen.

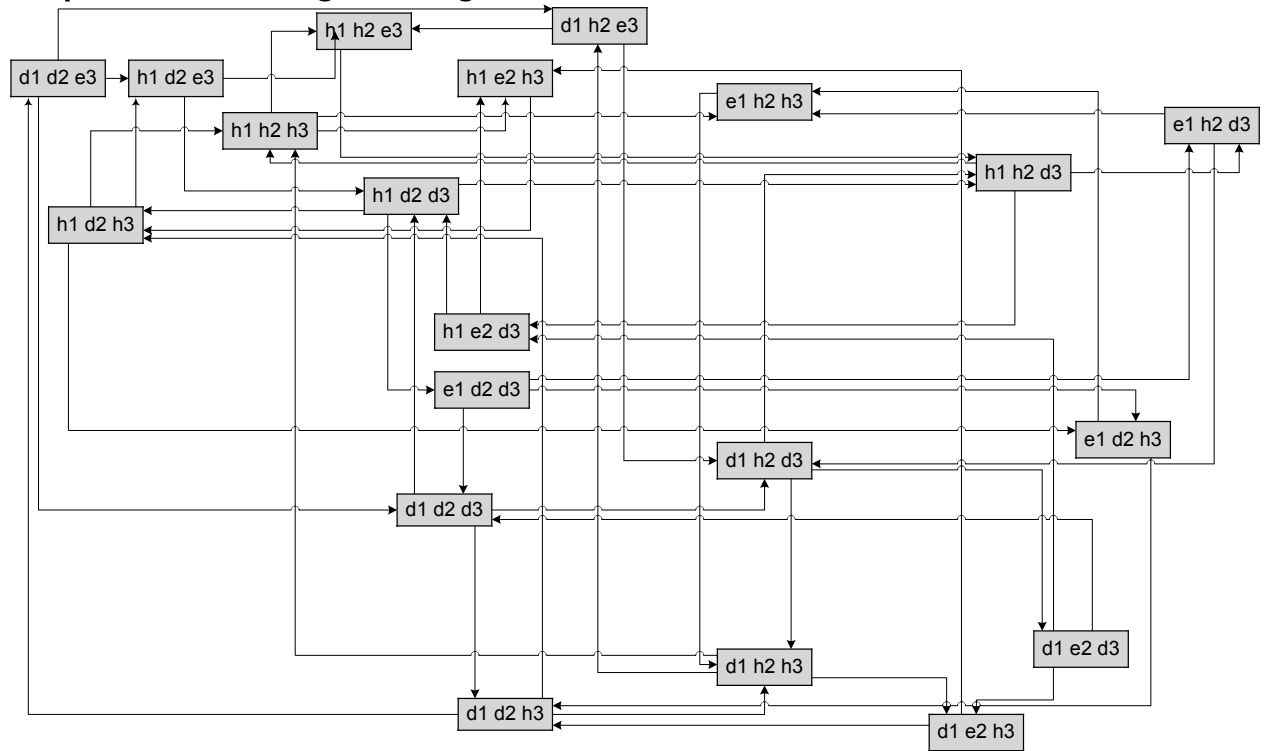
*(1/2 für „P kann nur Hunger bekommen, wenn kein anderer Hunger hat“; +1/2 wenn betroffene Zustände noch korrekt aufgezählt sind
+1/4 wenn erkannt, dass die übrigen Zyklen die Lebendigkeitseigenschaft nicht verletzen)*

e)

Richtiger Graph mit sämtlichen (auch unerlaubten Zuständen):



Graph für die Lösung der Aufgabe 2e:



Beispielgraph mit eingezeichnetem Pfad für Verhungern:

