Software Engineering

Besprechung

- zur Uebung 5
- und zur Zusatzübung Formale Verifikation

Agenda

- Besprechung der Teilaufgaben der Uebung 5
- Besprechung der Zusatzuebung zu Formaler Verifikation (fuer PPO 2001-Studierende).
- HiWi's und TA's fuer das SS 2007 gesucht.

Teil A - Zusammenarbeit und Vertragsformulierung (2)

a.) **(2,5 P)**

- 1. sämtliche Abonnenten werden über die Änderung des Meilenstandes informiert. **(¼ P)**
  - - Informationsaustausch **(¼ P)**
  - - Mittel: Beobachtermuster **(¼ P)**
  - - Abonnementsprinzip **(¼ P)**
  - - Systemzustand wird nicht verändert (nicht direkt durch diesen Aufruf. Es kann sein, dass Abonnenten den Systemzustand verändern) **(¼ P)**
- 2. Admin liest aktuellen Saldo des MilageAccount-Objektes um die Gesamtsumme an Meilen zu berechnen. **(¼ P)**
  - - Informationsaustausch **(¼ P)**
  - - Mittel: Direktmanipulation von Attributen **(¼ P)**
  - - Holprinzip (Lesezugriff auf öffentliches Attribut) **(¼ P)**
  - - Systemzustand wird verändert (currentAccountBalance wird verändert) **(¼ P)**

Teil A - Zusammenarbeit und Vertragsformulierung (3)

b.) **(4,5 P)**

- **MilageAdmin.addMilageAccount(MilageAccount a) (∑ ¾ P)**
  - PRE: !accountExists(a) and a!= null **(¼ P)**
  - POST: accountExists(a) and **(¼ P)**
    getBankBalance@PRE + a.balance == getBankBalance **(¼ P)**
- **MilageAdmin.removeMilageAccount(int AccountNo) (∑ ¾ P)**
  - PRE: accountExists(AccountNo) **(¼ P)**
  - POST: !accountExists(AccountNo) and **(¼ P)**
    getBankBalance@PRE - a.balance == getBankBalance **(¼ P)**
- **MilageAdmin.transferMiles(int srcAccount, int destAccount, float Amount) (∑ 1,5 P)**
  - PRE: accountExists(srcAccount) and accountExists(destAccount) **(¼ P)**
    and amount > 0 **(¼ P)**
  - POST: getBankBalance ==getBankBalance@PRE **(¼ P)**
    and srcAccount.balance@PRE ==srcAccount.balance + amount **(¼ P)**
    and destAccount.balance@PRE ==destAccount.balance – amount **(¼ P)**
  - OBLIGATION: Fangen einer allfaelligen java.lang.RuntimeException, welche bei einem PinkAir Account beim Unterschreiten von 0 ausgloest wird.
    Alternative ohne Obligation:
    - PRE: if getAccount(srcAccount) instanceof PinkAccount
      then getAccount(srcAccount).balance > amount) **(¼ P** für Obligation oder Alternative)

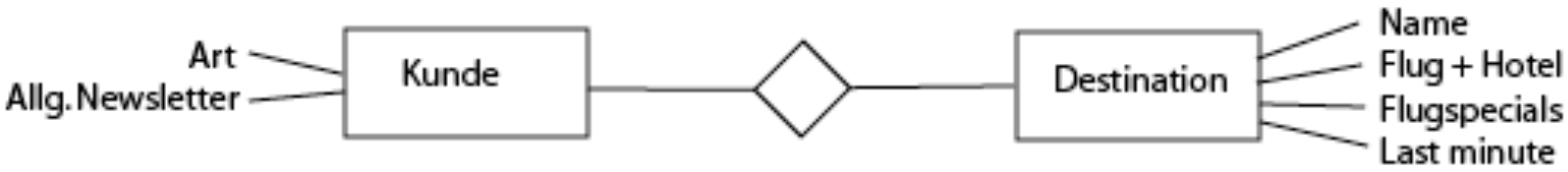
Teil A - Zusammenarbeit und Vertragsformulierung (4)

- **MilageAdmin.downgradeAccount(PlatiniumAccount pa) (∑ ¾ P)**
  - PRE: accountExists(pa) (¼ P)
  - POST: accountExists(pa.getAccountNo) (¼ P)
    and getMilageAccount(pa.getAccountNo) instanceof PinkAccount (¼ P)
    //evtl PRE: pa.balance >0 da dies Voraussetzung für einen PinkAccount ist!
- **MilageAdmin.upgradeAccount(PinkAccount pa) (∑ ¾ P)**
  - PRE: accountExists(pa) (¼ P)
  - POST: accountExists(pa.getAccountNo) (¼ P) and getMilageAccount(pa.getAccountNo) instanceof PlatiniumAccount (¼ P)

Die Aufgabe wurde gut gelöst!

Teil B - Aufwandschätzung, Function Points (1)

a)



- Dateneingabe: Auswahl speichern:
  - 2 Datenbestände, ~6 Datenelemente --> *mittel*.
- Anfrage: Destination auswählen: 1 Datenbestand, 3 Elemente --> *einfach*
- 1x externe Schnittstelle: Annahme: *einfach*; Anbindung zur DB; (interner Datenbestand wäre auch OK).

Teil B - Aufwandschätzung, Function Points (2)

- Dateneingaben (mittel) 1 *4 = 4 (1/2 P für Wert, 1/2 P für Berechnung)
- Anfragen (einfach) 1 *3 = 3 (1/2 P für Wert, 1/2 P für Berechnung)
- Ext. Schnittst. (einfach) 1 *5 = 5 (1/2 P)
- Summe: UFP = 12 (mit IFPUG 1994 Zählschema ermitteln)
- Gesamt-Einflussfaktor TDI = 44 (aus Einflussfaktoren-Tabelle, 1/2 P)
- VAF = 0.65 + 0.01 * TDI = 1.09
- FP = 12 * 1.09 = 13.08 (1P)

(TDI...total degree of influence, UFP...unadjusted function points, VAF...value adjustment factor)

b)

- Aufwand pro Function Point muss auch ermittelt werden, um Aussagen zu treffen. Schliessen auf Aufwand aus vergangenen Projekten, ... (1P)

*Häufigste Fehler: * Dateneingabe: Abgrenzung der Datenfelder/Datenelemente;*

** Oft wurden nur Ausprägungen (Daten!) gezählt, nicht die Felder selbst...*

Teil C - Aufwandschätzung, COCOMO2 (1)

a.) Wahl des Projektleiters (1 P)

- Skalierungsfaktoren:
  - Präzedenz (Neuartigkeit) (1/4 P)
    - IT4U Mitarbeiter: Keine Erfahrungen mit Produkt und Projektleitung --> hoher Wert.
    - AirportIT Mitarbeiterin: Erfahrung mit Produkt und Projektleitung --> niedriger Wert.
  - Zusammenarbeit (1/4 P)
    - IT4U Mitarbeiter: Kennt das Team und Unternehmen --> niedriger Wert.
    - AirportIT Mitarbeiterin: Kennt Team und Unternehmen noch nicht --> hoher Wert.

- Kostenfaktoren:
  - Personnel continuity (1/4 P)
    - (Annahme) Die Arbeitsmoral des internen Bewerbers könnte sinken wenn er den Job nicht bekommt, er könnte auch die Firma verlassen; Die externe Mitarbeiterin könnte Unstimmigkeiten verursachen, ...
  - Application experience (1/4 P)
    - Die externe Mitarbeiterin hat mehr Erfahrung im Anwendungsbereich Aviation, ...

Teil C - Aufwandschätzung, COCOMO2 (2)

b.) Entscheidung über mögliches Outsourcing **(1 P)**

- Skalierungsfaktoren:
  - Zusammenarbeit (1/4 P)
    - IT4U: Kommunikation ist einfacher --> niedriger Wert.
    - ITIndia: Geographische Distanz, sprachliche Probleme, Zeitunterschied --> hoher Wert.
  - Flexibilität (1/4 P)
    - IT4U: Durch Nähe schnellere Reaktionszeit --> niedriger Wert.
    - ITIndia: (Annahme) Es wird während des Projektverlaufes weniger kommuniziert werden und daher auch schwieriger auf Aenderungen anzupassen --> hoher Wert.

- Kostenfaktoren:
  - Programmer capability (1/4 P)
    - Team in Indien hat mehr Erfahrung --> tieferer Wert für Indien.
  - Team co-location (1/4 P)
    - Weitere Entfernung nach Indien --> höherer Wert für Indien.

Die Lösungen zu a) und b) waren zumeist sehr gut.

Teil C - Aufwandschätzung, COCOMO2 (3)

C.) Flexibilität - Eigenschaften (1 P)

- Beispiele:
  - Termindruck (1/2 P)
    - Macht weniger flexibel --> höherer Wert.
  - Einflussnahme des Kunden (1/2 P)
    - Je genauer der Kunde seine Anforderungen stellt, desto weniger flexibel ist das Unternehmen --> höherer Wert.
  - Grösse des Teams (...)
    - Je grösser das Team, desto weniger flexibel ist es --> höherer Wert.

Häufigste Fehler: oft keine 'Eigenschaften' von Flexibilität genannt, sondern Szenarien und ähnliches ...

Teil C - Aufwandschätzung, COCOMO2 (4)

d.)

(1P)

$$Aufwand = 2.45 * KSLOC^{B} * \prod_{i=1}^{17} EM_i$$

$$B = 1{,}01 + 0.01 * \sum_{i=1}^{5} SF_i$$

alle Skalierungsfaktoren nominal:

$$\sum_{i=1}^{5} SF_i = 2.43 + 2.64 + 2.53 + 2.97 + 2.74 = 13.3$$

$$B = 1.153$$

$$2.45 * KSLOC^{B} * \prod_{i=1}^{16} EM_i * EM_{REUSE\_NOMINAL} = 2.45 * (x * KSLOC)^{B} * \prod_{i=1}^{16} EM_i * EM_{REUSE\_VERY\_HIGH}$$

$$2.45 * KSLOC^{1.153} * \prod_{i=1}^{16} EM_i * 1 = 2.45 * (x * KSLOC)^{1.153} * \prod_{i=1}^{16} EM_i * 1.29$$

$$KSLOC^{1.153} = (x * KSLOC)^{1.153} * 1.29$$

$$KSLOC^{1.153} = x^{1.153} * KSLOC^{1.153} * 1.29$$

$$KSLOC^{1.153} = x^{1.153} * KSLOC^{1.153} * 1.29$$

$$KSLOC^{1.153} = x^{1.153} * KSLOC^{1.153} * 1.29$$

$$1 = x^{1.153} * 1.29$$

$$\frac{1}{1.29} = x^{1.153}$$

$$\sqrt[1.153]{\frac{1}{1.29}} = x = 0.8$$

KSLOC müsste um **ca. 20%** verringert werden.

e.) Unternehmen muss die Berechnung spezifisch anpassen. Für aussagekräftige Werte sind Erfahrungen aus vergangenen Projekten nötig. (1 P)

Lösungen: d) und e) wurden gut gelöst!

Teil D - Risikoschätzung

- Diverse Antworten möglich …
- ½ Punkt pro erkanntes (und kurz begründetes Risiko): ¼ Punkt für Angabe einer Schadenshöhe und deren Begründung (pro Risiko). ¼ Punkt für sinnvolle Massnahme.
- (total) ½ Punkt Abzug, falls nicht erwähnt, dass Risiko = Schadenshöhe* Eintretenswahrscheinlichkeit.

- Beispiele ...
  - Unmotivierte Entwickler: E = 5, S = 8, Risikio = E*S = 40. Massnahmen: Arbeitsbedingungen verbessern, Gruppenausflüge, Firmenfeier, Projektabläufe besser planen, Anreize für gute Leistungen schaffen, etc.
  - falscher Technologieeinsatz: E = 4, S = 8, Risikio = 32. Massnahmen: Miteinbezug von Experten bei der Konzeptionsphase, Prototypen entwickeln, etc.

    - *Häufigste Fehler: meist wurden zu konkrete Szenarien genannt; Eintrittwahrscheinlichkeit oft weggelassen;*

Zusatzübung

(für PPO 2001 Studenten)

- Formale Verifikation
  - Axiomatische Semantik
  - Model Checking

Axiomatische Semantik (1)

```
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 ≡ 7 ≤ y < 25 *)
```

Axiomatische Semantik (2)

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

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

≡ wp(x := 3y-2; IF (x <12) THEN y=3x-9 ELSE y=x+6, 9 <=
y < 27)

≡ wp(x := 3y-2; ((x <12) ∧ 9 <= 3x-9 < 27) ∨ ((x >= 12)
∧ 9 <= x+6 < 27)

≡ wp(x := 3y-2; ((x < 12) ∧ 18 <= 3x < 36) ∨ ((x >= 12)
∧ 3 <= x < 21)

≡ wp(x := 3y-2; ((x < 12) ∧ 6 <= x < 12) ∨ ((x >= 12)
∧ 3 <= x < 21)
```

Axiomatische Semantik (3)

≡ ((3y-2 < 12) ∧ 6 <= 3y-2 < 12) ∨ ((3y-2 >= 12) ∧ 3 <= 3y-2 < 21)

≡ ((3y < 14) ∧ 8 <= 3y < 14) ∨ ((3y >= 14) ∧ 5 <= 3y < 23)

≡ ((y < 14 / 3) ∧ 8/3 <= y < 14/3) ∨ ((y >= 14/3) ∧ 5/3 <= 3y < 23/3)

≡ ((y < 14 / 3) ∧ 8/3 <= y < 14/3) ∨ ((y >= 14/3) ∧ 5/3 <= y < 23/3)

≡ (8/3 <= y < 14/3) ∨ ((y >= 14/3) ∧ 5/3 <= y < 23/3)

≡ (8/3 <= y < 14/3) ∨ ((y >= 14/3) ∧ y < 23/3)

≡ (8/3 <= y < 23/3)

Zusatzfrage: Welche Vorbedingung würde diese wp verletzen?

Antwort: ein Wert für y ausserhalb des eben berechneten Wertebereiches.

Model Checking (1)

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): $\wedge_{i=1,n} (h_i \rightarrow \Diamond e_i)$

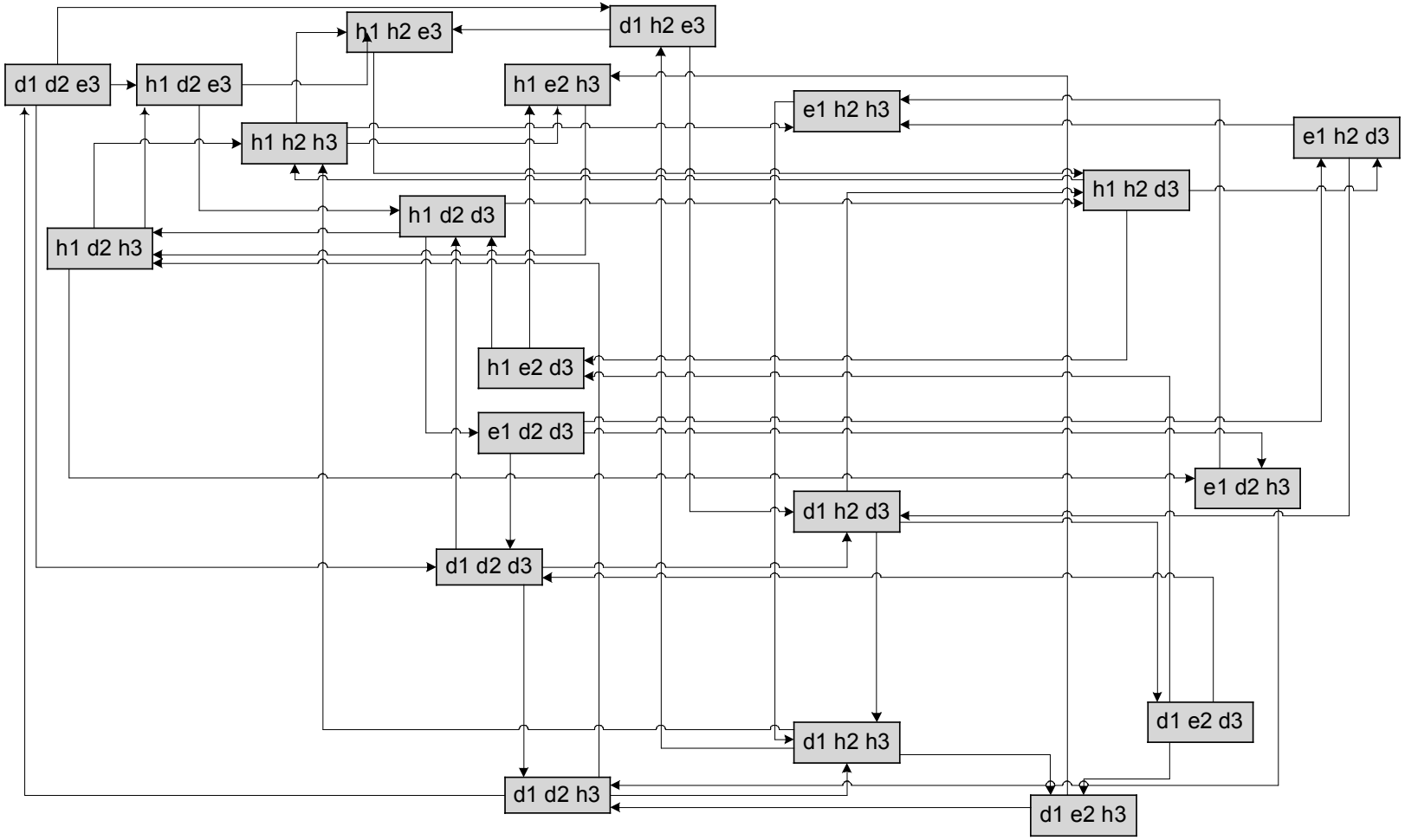
c) Beispiel für fünf Philosophen:
$\equiv \neg((e_1 \wedge e_2) \vee (e_2 \wedge e_3) \vee (e_3 \wedge e_4) \vee (e_4 \wedge e_5) \vee (e_5 \wedge e_1))$

Allgemein formuliert:
$\equiv \neg((e_1 \wedge e_2) \vee (e_2 \wedge e_3) \vee \ldots \vee (e_{n-1} \wedge e_n) \vee (e_n \wedge e_1))$

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 $[e_1 e_2 e_3]$

Model Checking (2)

e)



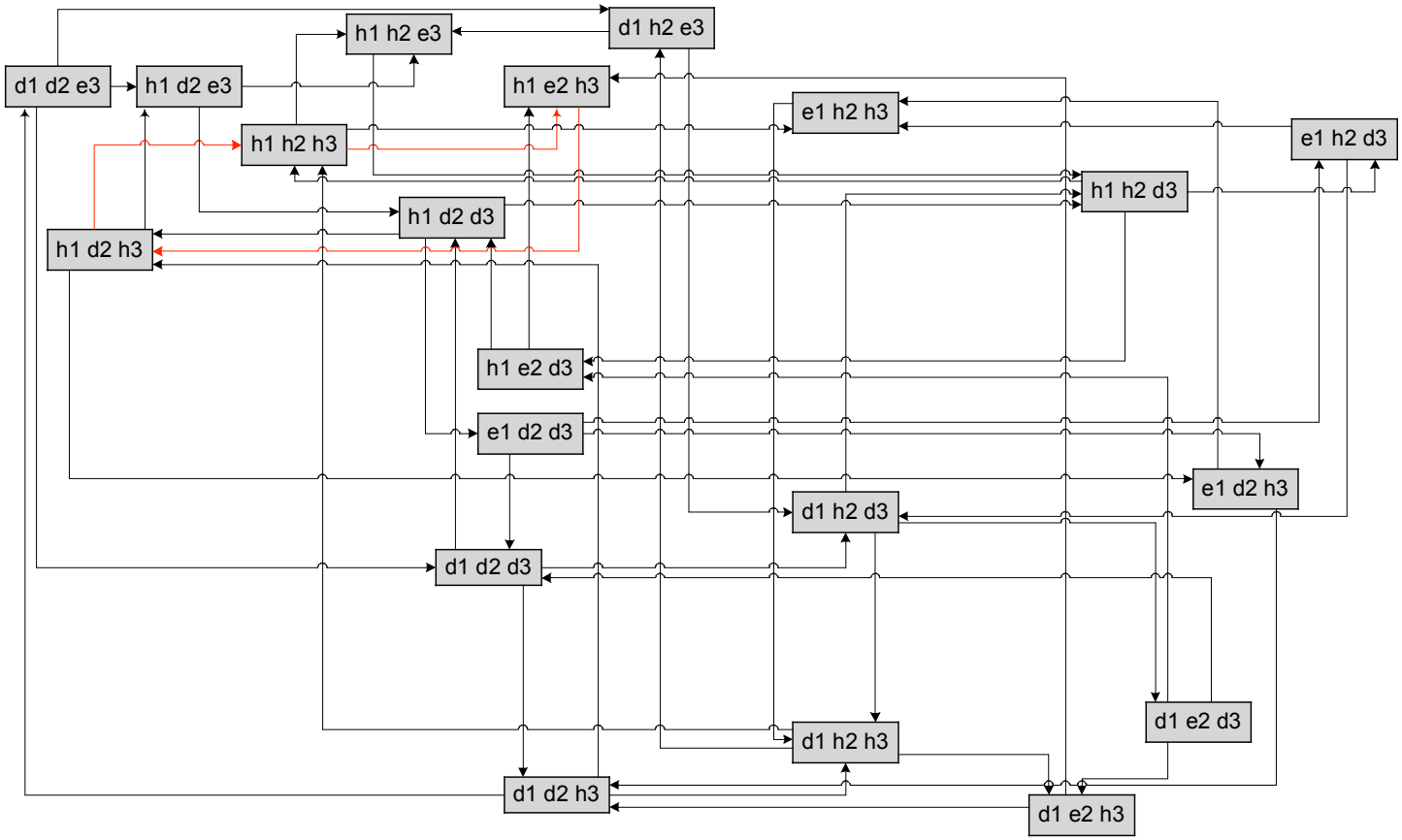
Model Checking (3)

Gegenbeispiel 2, für Philosoph 1:

f)

Gegenbeispiel 1:
$[h_1h_2d_3]$, $[e_1h_2d_3]$,
$[d_1h_2d_3]$, $[h_1h_2d_3]$

→ nicht sicher, ob Philosoph 2 irgendwann isst.



Model Checking (4)

g) $[h_1h_2*]$, $[h_1*h_3]$, $[*h_2h_3]$, $[h_1h_2h_3]$ -- alle Zustände mit 2 oder 3 hungrigen Philosophen müssen aufgespalten werden -- Abbildung der Vorgeschichte. So kann nun immer derjenige Philosoph essen, der bereits am längsten Hunger hatte.

Danke für die Aufmerksamkeit!