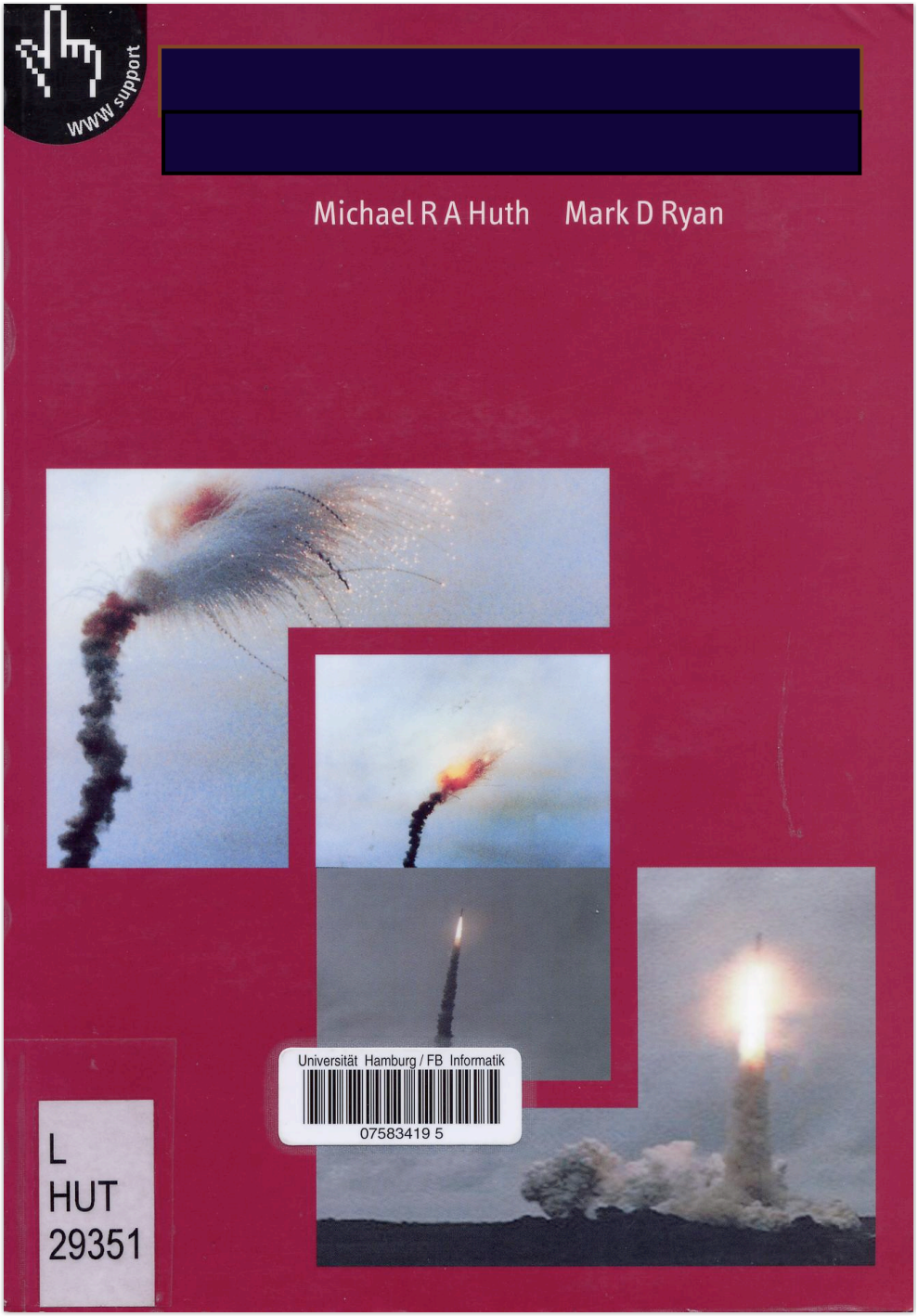
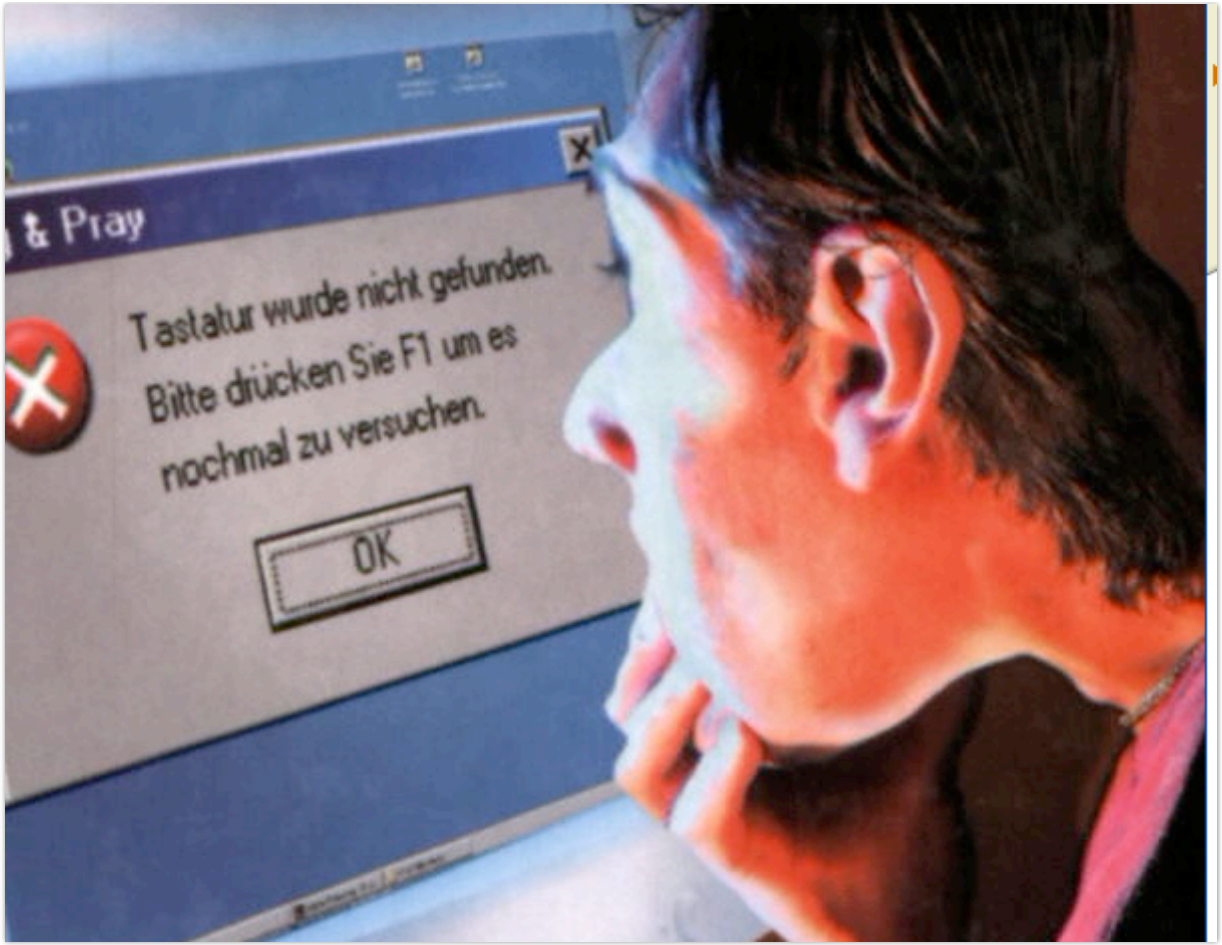
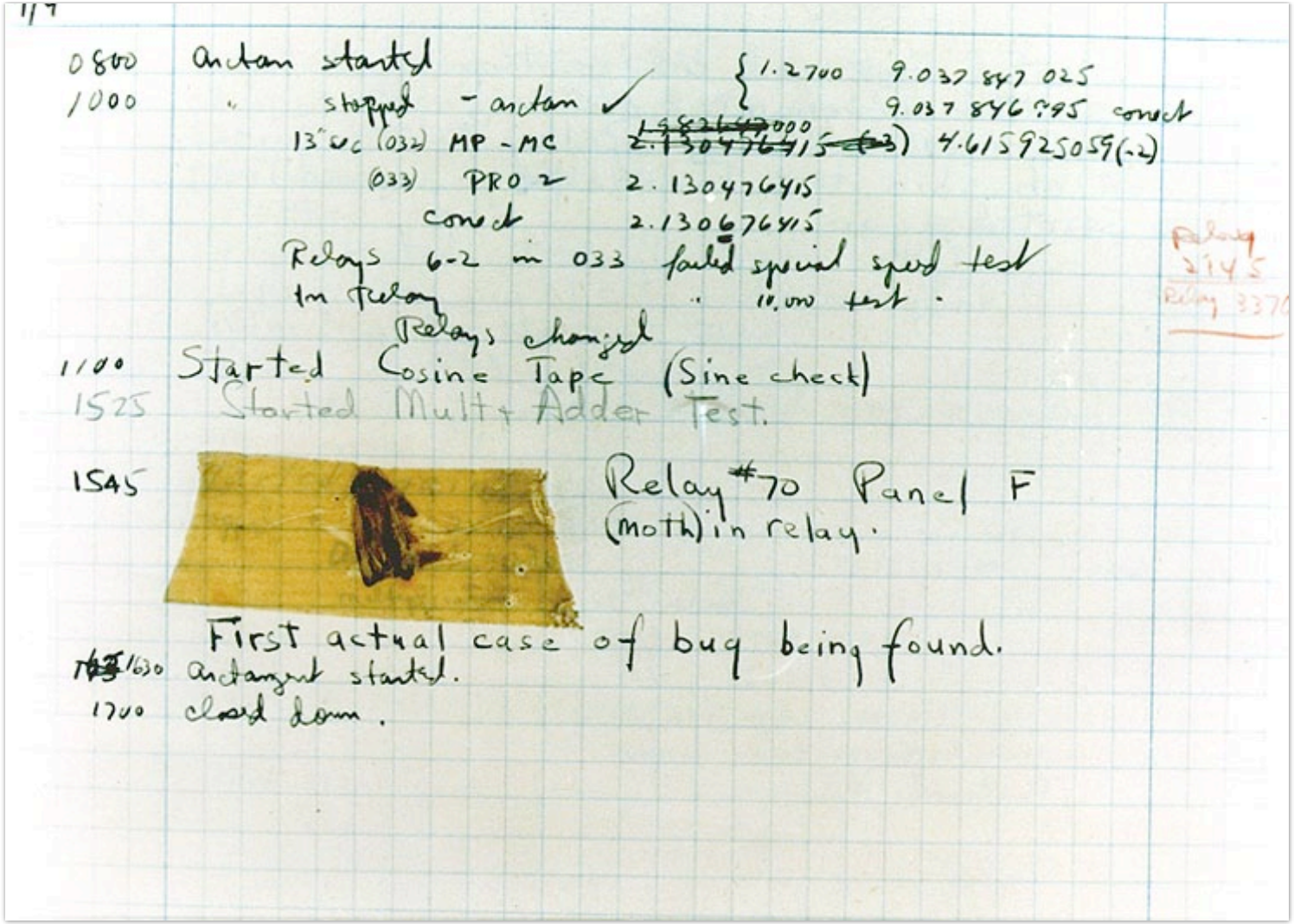


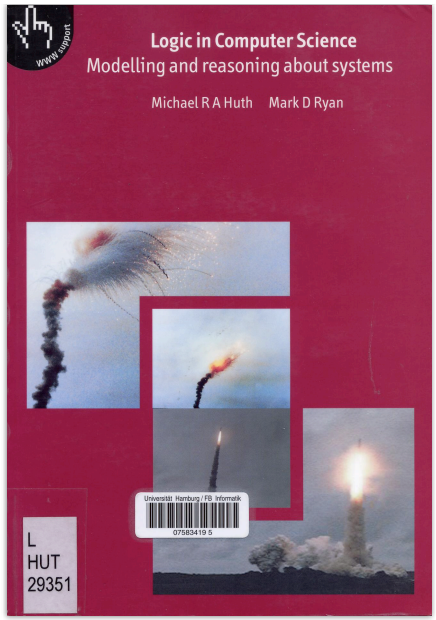
Ariane 5

4. Juni 1996, Kourou / Frz. Guyana, ESA


& Pray
Tastatur wurde nicht gefunden.
Bitte drücken Sie F1 um es
nochmal zu versuchen.
OK



Die Computer-Pionierin Grace Hopper hat Überlieferungen zufolge diesen Begriff zuerst verwendet. Sie hatte am 9. September 1945 eine Motte in einem Relais des Computers Mark II Aiken Relay Calculator gefunden, und das Tier führte zu einer Fehlfunktion. Grace Hopper entfernte die Motte und schrieb in ihr Logbuch: „First actual case of bug being found“, dazu klebte sie die im Relais gefundene Motte.



Ariane 5 – 501

4. Juni 1996, Kourou / Frz. Guyana, ESA

Jungfernflug der neuen europäischen Trägerrakete (Gewicht: 740 t, Nutzlast 7 – 18 t) mit 4 Cluster-Satelliten; Entwicklungskosten in 10 Jahren: DM 11 800 Millionen

Wirkung: 37 Sekunden nach Zünden der Rakete (30 Sekunden nach Liftoff) erreichte Ariane 5 in 3700 m Flughöhe eine Horizontal-Geschwindigkeit von 32768.0 (interne Einheiten).

Dieser Wert lag etwa fünfmal höher als bei Ariane 4.

Die Umwandlung in eine ganze Zahl führte zu einem Überlauf, der jedoch nicht abgefangen wurde.
Der Ersatzrechner (Redundanz !) hatte das gleiche Problem schon 72 msec vorher und schaltete sich sofort ab.
Daraus resultierte, dass Diagnose-Daten zum Hauptrechner geschickt wurden, die dieser als Flugbahndaten interpretierte.

Daraufhin wurden unsinnige Steuerbefehle an die seitlichen, schwenkbaren Feststoff-Triebwerke, später auch an das Haupttriebwerk gegeben, um die großen Flugabweichungen (über 20 Grad) korrigieren zu können. Die Rakete drohte jedoch auseinanderzubrechen und sprengte sich selbst (39 sec).

Ein intensiver Test des Navigations- und Hauptrechners wurde nicht unternommen, da die Software bei Ariane 4 erprobt war.

Schaden:

DM 250 Millionen Startkosten
DM 850 Millionen Cluster-Satelliten
DM 600 Millionen für nachfolgende Verbesserungen
Verdienstausfall für 2 bis 3 Jahre
Der nächste Testflug konnte erst 17 Monate später durchgeführt werden – 1. Stufe beendete vorzeitig das Feuern.
Der erste kommerzielle Flug fand im Dezember 1999 statt.
Der Versuch, neue 4 Cluster-Satelliten zu starten, gelang im Juli und August 2000 mit zwei russischen Trägerraketen

Ursachen:

Der problematische Programmteil wurde nur für die Vorbereitung beim Start und den Start selbst gebraucht.
Er sollte nur während einer Übergangszeit aktiv sein, aus Sicherheitsgründen: 50 sec, bis die Bodenstation bei einer Startunterbrechung die Kontrolle übernommen hätte.
Trotz des ganz anderen Verhaltens der Ariane 5 wurde dieser Wert nicht neu gesetzt.

Nur bei 3 von 7 Variablen wurde ein Überlauf geprüft – für die anderen 4 Variablen existierten Beweise, dass die Werte klein genug bleiben würden (Ariane 4).

Diese Beweise galten jedoch nicht für die Ariane 5 und wurden dafür auch gar nicht nachvollzogen.
Problem der Wiederverwendung von Software !

Mariner 1

22. Juli 1962, Cape Canaveral/Florida
Start der ersten amerikanischen Venussonde Mariner 1
Trägerrakete Atlas-Agena B (NASA, 15. AAB-Start)

Ausschnitt aus dem FORTRAN-Programm zur Steuerung der Flugbahn der Trägerrakete:

```
DO 5 K = 1. 3
   ...
 5 CONTINUE
```

Entscheidender Fehler:

Punkt statt Komma!
Wirkung:

DO 5 K = 1. 3

Wertzuweisung an eine nicht deklarierte Variable

Kein Durchlauf der (nicht vorhandenen) Schleife

Folge:

Abweichung der Trägerrakete von der vorgesehenen Flugbahn
Zerstörung der Rakete nach 290 Sekunden
Kosten dieser Wertzuweisung: $ 18.5 Millionen

Ursache:

Schlechte Programmiersprache wegen Blanks (Zwischenräume) in Namen und Zahlen erlaubt
Variablen-Deklaration nicht notwendig.
Strukturierte Schleife (END DO) nicht möglich.

AT&T-Telefonnetz

15. Januar 1990:

70 Millionen von 138 Millionen Ferngesprächen innerhalb USA konnten 9 Stunden lang nicht vermittelt werden.

Schaden:

$ 75 Millionen bei AT&T (ohne Folgeschäden)
Mehrere $ 100 Millionen bei den Kunden (Versandhandel, Transportunternehmen, Reisebüros usw.)

Ursache:

Eine Schaltzentrale in New York setzte sich nach einer Fehlfunktion in den RESET-Modus:
- Ausfall-Meldung an alle anderen Zentralen
- Neubesetzung interner Tabellen (Reset)
- OK-Meldung an alle anderen Zentralen
- Weiterleiten neuer Ferngespräche

Alle Zentralen mußten daraufhin ihre Tabellen ändern.

Bei 3 Schaltzentralen kamen aber kurz nach der OK-Meldung mehrere neue Gespräche an

Verarbeitung der Meldung und der neuen Gespräche zerstörte Daten, was zum Rechnerausfall führte

Im Schneeball-System wurden 9 Stunden lang alle Zentralen lahmgelegt (zu 50% im RESET-Modus)

Notlösung: keine OK-Meldungen mehr verschicken

Eigentlicher Fehler wurde eine Woche später gefunden:

Der break-Befehl von C wurde falsch eingesetzt.

Existierte seit Programm-Optimierung 4 Wochen vorher.

1000 Zeilen	15 Blatt Papier
100 000 Zeilen	15 cm hoher Stapel
10 Millionen Zeilen	15 m hoher Stapel
200 Millionen Zeilen	300 m hoher Stapel

Programmieraufwand (kaum sprachabhängig)

mittlere Programmgröße	50 Zeilen pro Tag
große Programmgröße	10 – 20 Zeilen pro Tag
kritische Software	1 – 2 Zeilen pro Tag

Fehlerhäufigkeit

Normale Software	25 Fehler pro 1000 Zeilen
Wichtige Software	2 – 3 Fehler pro 1000 Zeilen
Medizinische Software	0.2 Fehler pro 1000 Zeilen
Space Shuttle Software	< 0.1 Fehler pro 1000 Zeilen

TeX 82	
Handy	
Hubble Bodensoftware	
Luftraum-Überwachung	
Space Shuttle, auch IIS	
B-2 Stealth Bomber	
Windows 95	
Windows NT 4.0	
Windows 2000	
SDI, auch NMD (Schätzung)	
	25 – 100 Millionen Zeilen
Telefonsysteme USA	200 Millionen Zeilen

Test – Verifikation – Validierung

Tests sehr zeitaufwendig

25 – 185 Stunden Testzeit für 1000 Zeilen

Keine Interpolation möglich (diskrete Werte)

nur die Anwesenheit von Fehlern ist testbar, nie deren Abwesenheit

Fehlende Programmteile nicht testbar

Fehler in nebenläufigen Programmen nicht reproduzierbar

Therac-25 Beschleuniger

(10 Geräte seit 1983 in USA und Kanada im Einsatz)
Software errechnet und steuert Bestrahlungs-Stärke entweder als Elektronenstrahlung (bis 25 MeV) oder als Röntgenstrahlung (mit 100-facher Elektronenstrahlung, Wolfram-Scheibe im Strahl, absorbiert Elektronen, Röntgenstrahlen entstehen)
Juni 1985, Georgia/USA

Elektronenstrahl brannte Loch durch Schulter

Krankenhaus lehnte Verschulden ab:
"Computer-gesteuert", "Hat schon immer funktioniert"
März 1986, Texas/USA
Patient erhielt zweimal Überdosis, Tod September 1986

Hersteller untersuchte das Problem, fand aber nichts
April 1986, Texas/USA

Fehler in derselben Klinik, Patient starb im Mai 1986
Zufällig gefundene Ursache:
Operator gab den Strahltyp falsch ein ("X-Ray")
Maschine schaltete um auf 100-fache Energie
Operator korrigierte Fehler in weniger als 8 Sekunden
Maschine nahm nur Wolfram-Scheibe aus Strahl heraus
Sicherheitsmechanismus schaltete nach 0.3 Sekunden ab
Notlösung: Abmontieren der Korrektur-Taste (Up-Key)

Korrektur im November 1986 (über 20 Modifikationen

(10 Geräte seit 1983 in USA und Kanada im Einsatz)
Software errechnet und steuert Bestrahlungs-Stärke entweder als Elektronenstrahlung (bis 25 MeV) oder als Röntgenstrahlung (mit 100-facher Elektronenstrahlung, Wolfram-Scheibe im Strahl, absorbiert Elektronen, Röntgenstrahlen entstehen)
Juni 1985, Georgia/USA

Elektronenstrahl brannte Loch durch Schulter

Krankenhaus lehnte Verschulden ab:
"Computer-gesteuert", "Hat schon immer funktioniert"
März 1986, Texas/USA
Patient erhielt zweimal Überdosis, Tod September 1986

Hersteller untersuchte das Problem, fand aber nichts
April 1986, Texas/USA

Fehler in derselben Klinik, Patient starb im Mai 1986
Zufällig gefundene Ursache:
Operator gab den Strahltyp falsch ein ("X-Ray")
Maschine schaltete um auf 100-fache Energie
Operator korrigierte Fehler in weniger als 8 Sekunden
Maschine nahm nur Wolfram-Scheibe aus Strahl heraus
Sicherheitsmechanismus schaltete nach 0.3 Sekunden ab
Notlösung: Abmontieren der Korrektur-Taste (Up-Key)

Korrektur im November 1986 (über 20 Modifikationen)

Space Shuttle 1 (Columbia), April 1981:

1. Startversuch der US-Raumfähre: Abbruch des Starts, da die 4 (alten) IBM-Bordrechner zu unterschiedlichen Ergebnissen kamen und sich nicht synchronisierten
Ursache: Eine Verzögerungsbedingung war zwei Jahre zuvor von 50 ms auf 80 ms geändert worden
In 2000 Stunden Testzeit wurden über 200 Fehler gefunden
Beim 1. Flug traten 24 weitere Fehler auf (Fehler-Manual!)

http://www-aix.gsi.de/~giese/swr/allehtml.html

Model Checking

Literatur:

E.M. Clarke et al.: *Model Checking*, The MIT Press, Cambridge, 1999, [CGP99]

B. Bèrard et al.: *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer, Berlin, 1999, [BBF99]

C. Girault, R. Valk: *Petri Nets for Systems Engineering*, Part III: Verification, Springer, Berlin, 2003, [GV03]

M. Huth, M. Ryan: *Logic in Computer Science*, Cambridge Univ. Press, 2004, [HR04]

C. Baier, J.-P. Katoen: *Principles of Model Checking*, MIT Press, 2008, [BK08]

System: Hardware- und Software-Systeme

System-Validierung:

- traditionell: Testen und Simulation
  am Anfang bei einfachen Fehlern sehr wirksam

  ---> aber immer weniger effektiv, wenn komplexe und verborgene Fehler auftreten,
  ---> insbesondere bei bei Systemen mit Asynchronität, Parallelität, Nebenläufigkeit,
  ---> Fehler oft von speziellen Zeitparametern/Nachrichtenlaufzeiten abhängig

- alternativ: formale Verifikation

  ---> umfassende Prüfung
  ---> alle Zweige des Verhaltens werden geprüft

wichtige Vorgehensweisen:

---> Hoare-Systeme (Zusicherungen, Invarianten)

---> temporale Logik

----> Analyse des Zustandsraumes (model checking)

Zustandsraum-Analyse/Model Checking

Vorteil:

---> ohne besondere Kenntnisse anwendbar
---> bei nicht korrektem System:
Generierung von Abläufen die zu den Fehlern führen

Nachteil:

---> Größe des Zustandsraumes

Abhilfe:

---> symbolisches Model Checking
---> Faltung
---> Symmetrien
---> ....

Literatur:

E.M. Clarke et al.: *Model Checking*, The MIT Press, Cambridge, 1999, [CGP99]

B. Bèrard et al.: *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer, Berlin, 1999, [BBF99]

C. Girault, R. Valk: *Petri Nets for Systems Engineering*, Part III: Verification, Springer, Berlin, 2003, [GV03]

M. Huth, M. Ryan: *Logic in Computer Science*, Cambridge Univ. Press, 2004, [HR04]

C. Baier, J.-P. Katoen: *Principles of Model Checking*, MIT Press, 2008, [BK08]

1.4 Kripkestrukturen

1.4.1 Verifikation und Model-Checking

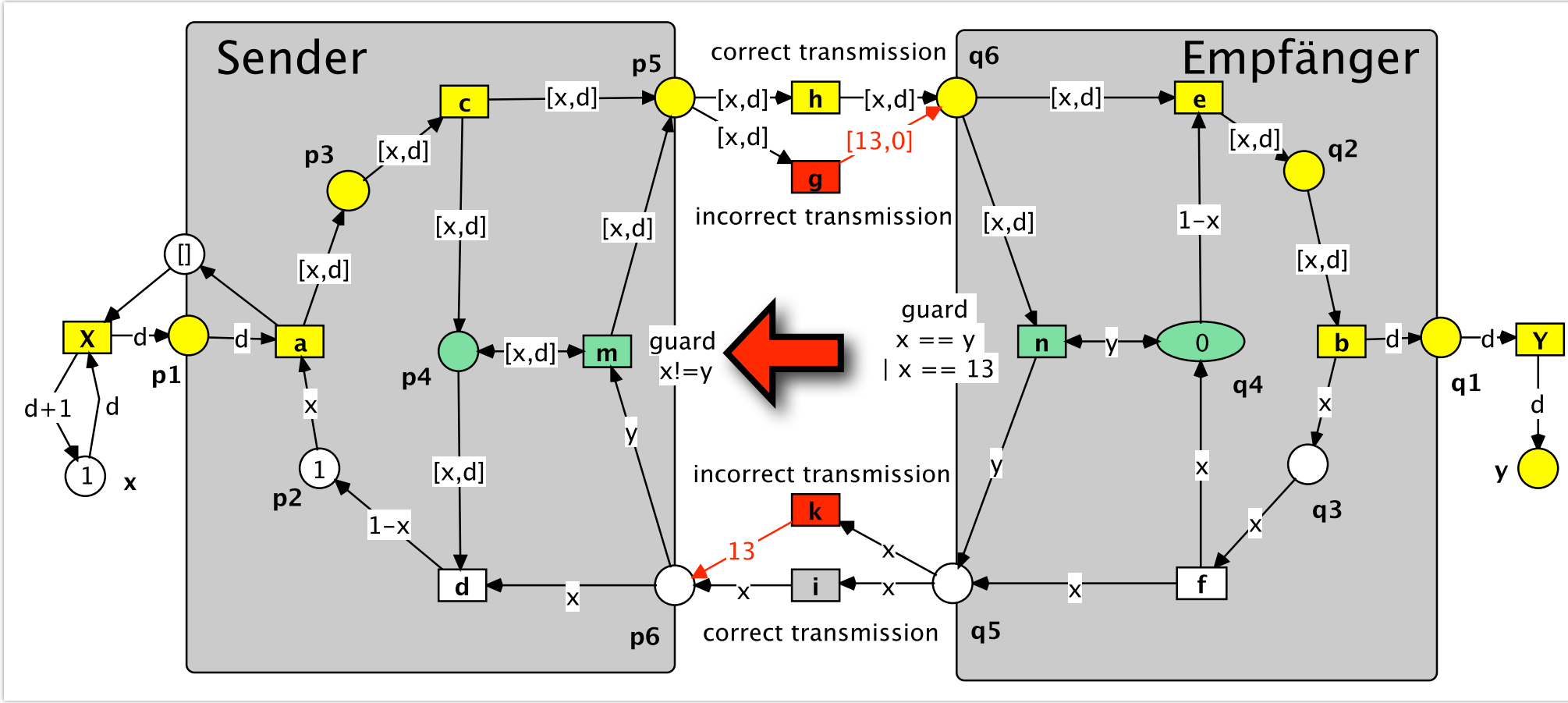
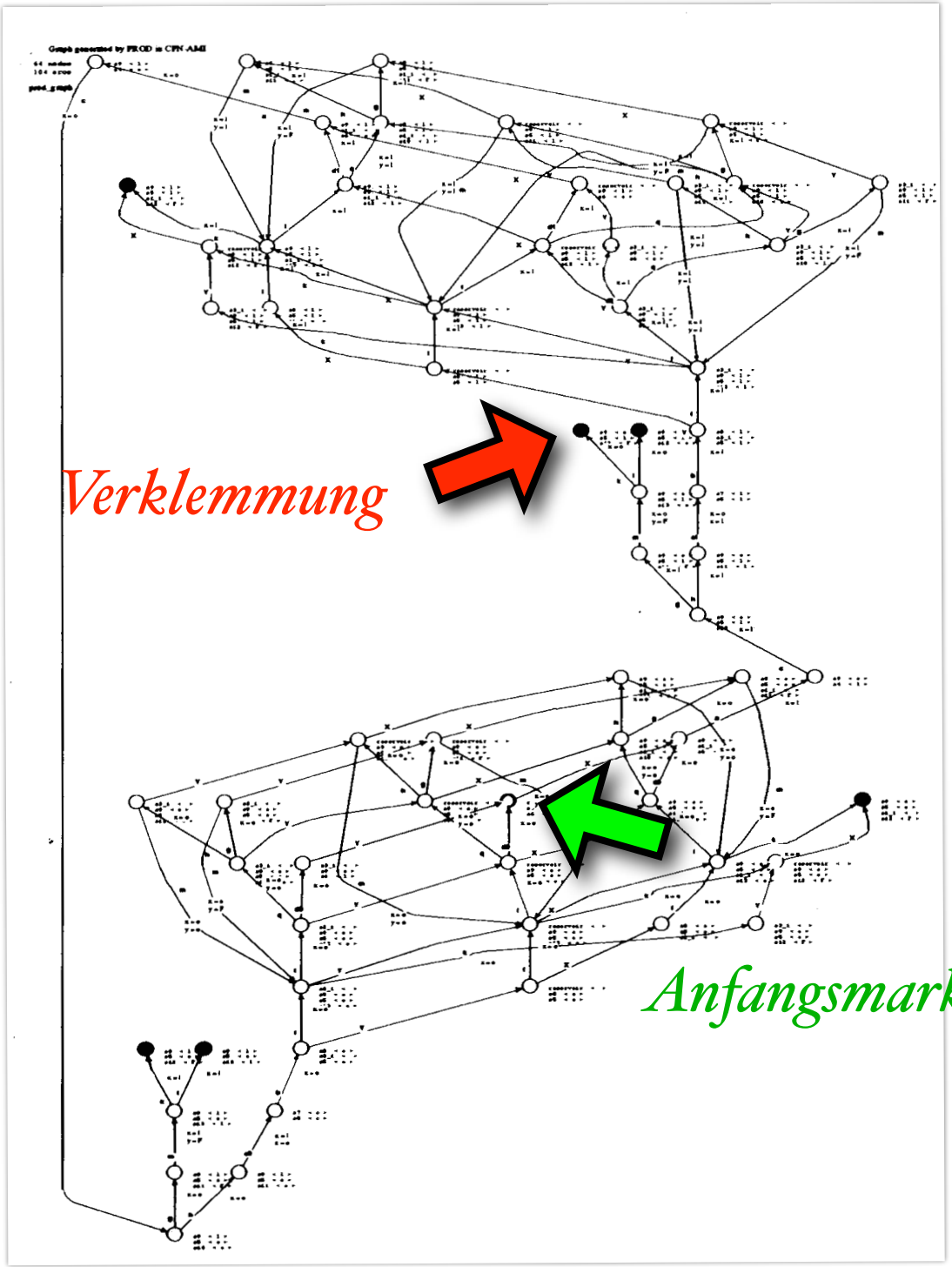


Abbildung 2.60: Das Alternierbitprotokoll abp-5

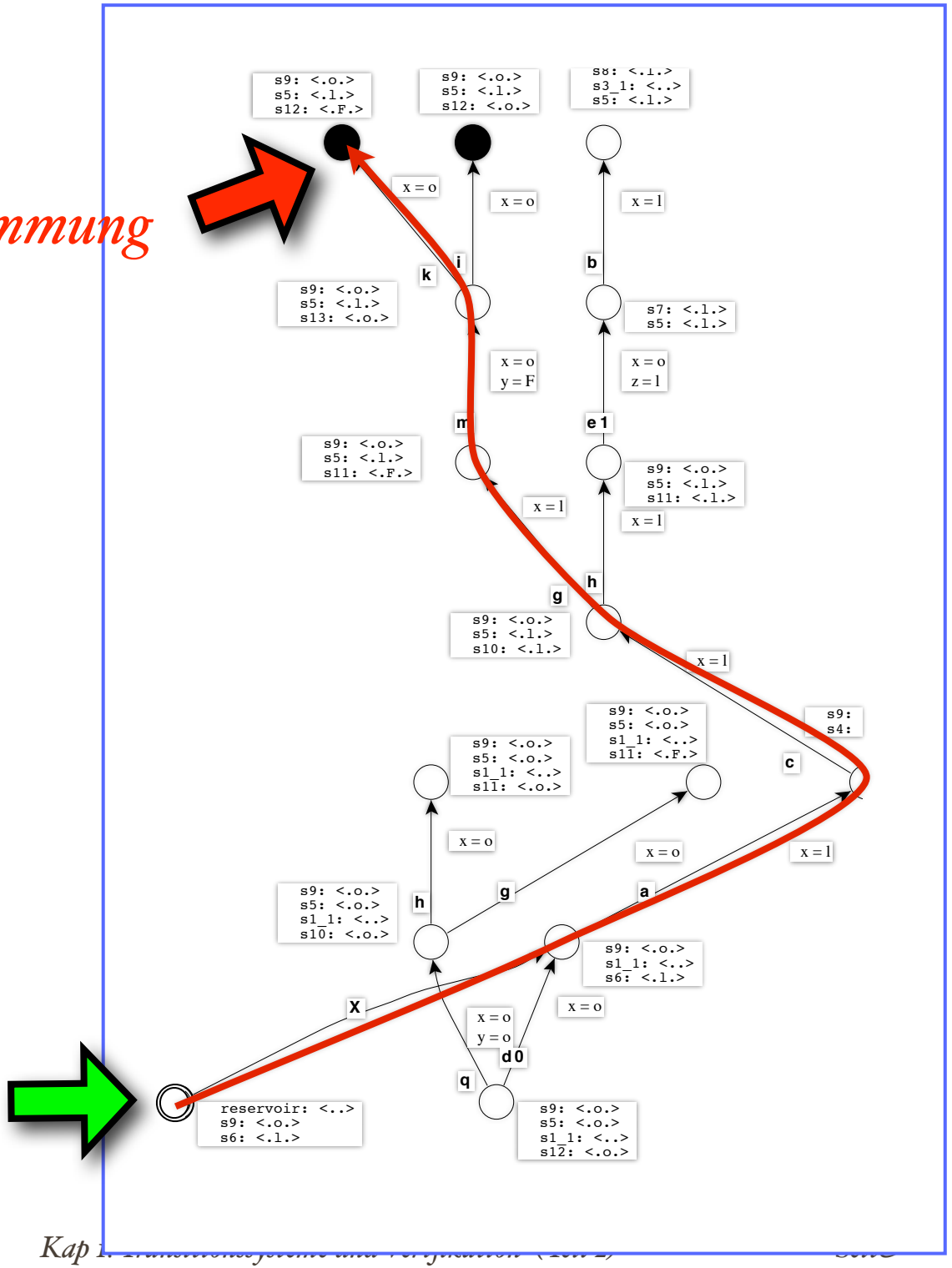
Transitionssystem eines Alternierbitprotokolls

(aus einem Petrinetz)



Transitionssystem eines Alternierbitprotokolls

(aus einem Petrinetz)



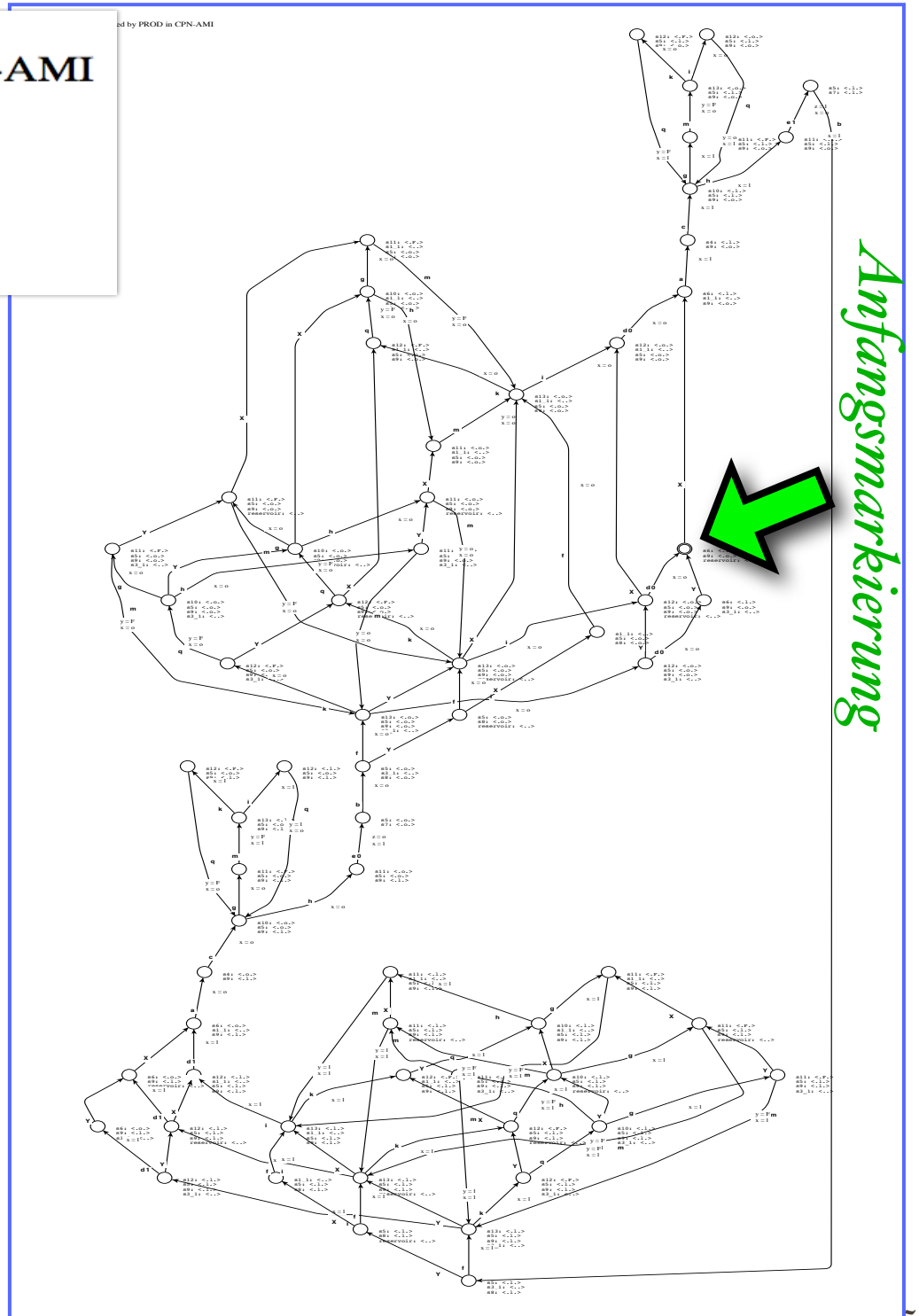
Graph generated by PROD in CPN-AMI

64 nodes
108 arcs

prod_graph

Transitionssystem eines Alternierbitprotokolls
(aus einem Petrinetz)

keine Verklemmung



1.4.2 Transitionssysteme in Form von Kripke-Strukturen

Definition 1.20 *Sei $TS = (S, A, tr, S^0, S^F)$ ein Transitionssystem. Eine* Zustandsetikettenfunktion *ist eine Abbildung $E_S : S \to \mathcal{P}(AP)$, wobei AP eine Menge von atomaren Aussagen ist.*

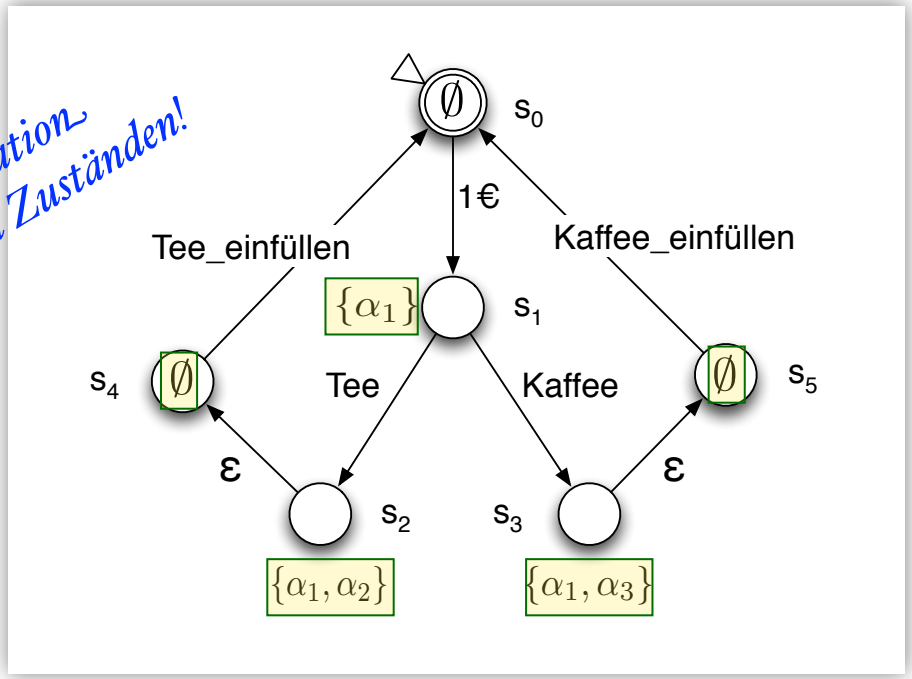


Abb. 1.12

$AP := \{\alpha_1, \alpha_2, \alpha_3\}$ durch $\alpha_1 =$ „1€ gezahlt“,

$\alpha_2 =$ „Tee gewählt“ und $\alpha_3 =$ „Kaffee gewählt“

1.4.2 Transitionssysteme in Form von Kripke-Strukturen

Definition 1.20 *Sei $TS = (S, A, tr, S^0, S^F)$ ein Transitionssystem. Eine* Zustandsetikettenfunktion *ist eine Abbildung $E_S : S \to \mathcal{P}(AP)$, wobei AP eine Menge von atomaren Aussagen ist.*

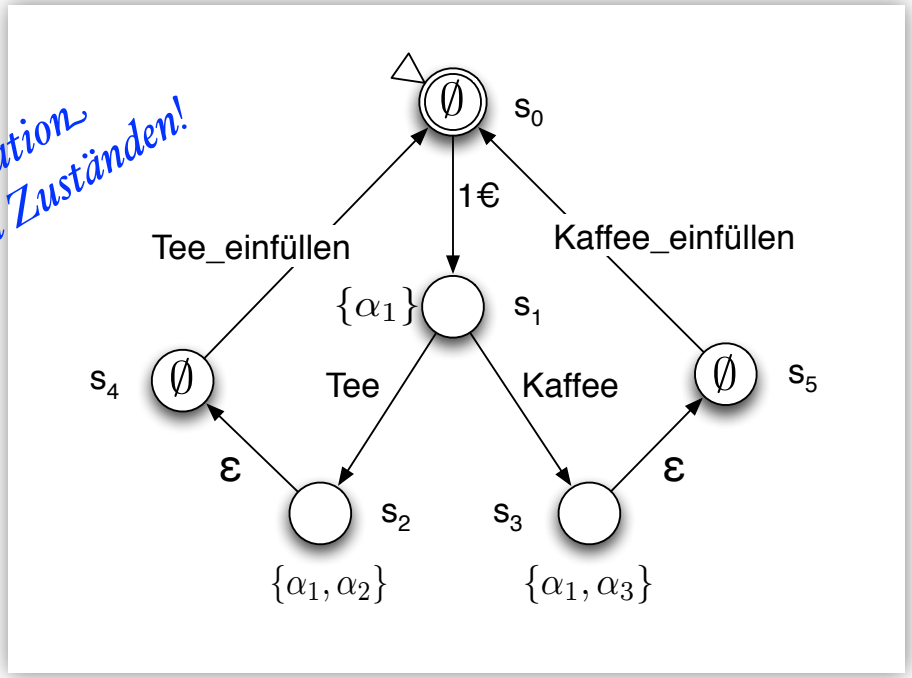


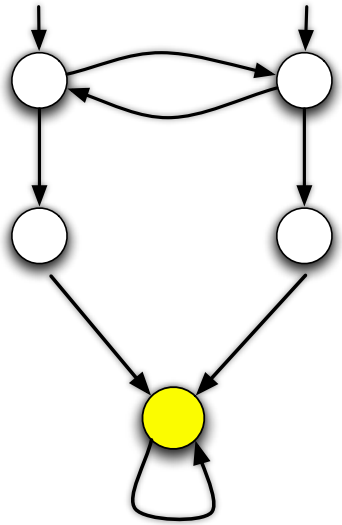
Abb. 1.12

Definition 1.22 *(Kripke-Struktur 1)*

Eine Kripke-Struktur *ist ein endliches Transitionssystem $M := (S, A, tr, S_0, \{\})$*[11] *mit einelementiger Aktionenmenge (also $|A| = 1$), linkstotaler*[12] *Transitionsrelation tr und zusätzlich einer Zustandsetikettenfunktion $E_S : S \to \mathcal{P}(AP)$.*

[11] D.h. es gibt keine Endzustände.

Definition 1.23 *(Kripke-Struktur 2)*



Eine Kripke-Struktur $M := (S, S_0, R, E_S)$ *besteht aus*

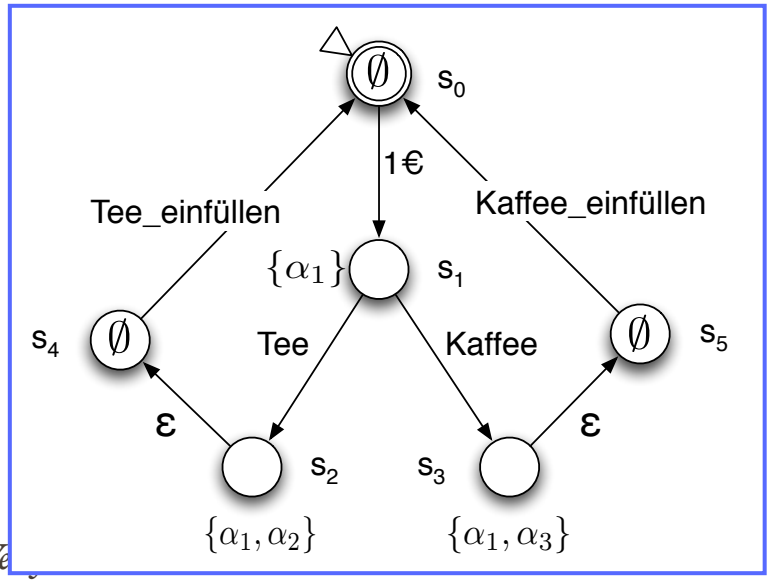
a) einer endlichen Zustandsmenge S,

b) einer Menge $S_0 \subseteq S$ *von Anfangszuständen,*

c) einer linkstotalen (Transitions-)Relation $R \subseteq S \times S$ *und*

d) einer Zustandsetikettenfunktion $E_S : S \to \mathcal{P}(AP)$, *die jedem Zustand* s *eine Menge* $E_S(s) \subseteq AP$ *von aussagenlogischen atomaren Formeln zuordnet (die in diesem Zustand gelten).*

$$\forall s \in S \; \exists s' \in S : (s, s') \in R$$



$E_S(3) \quad = \quad \{\neg Start, Close, \neg Heat, \neg Error\}$ in der Kripke-Struktur von Abb. 5.5.

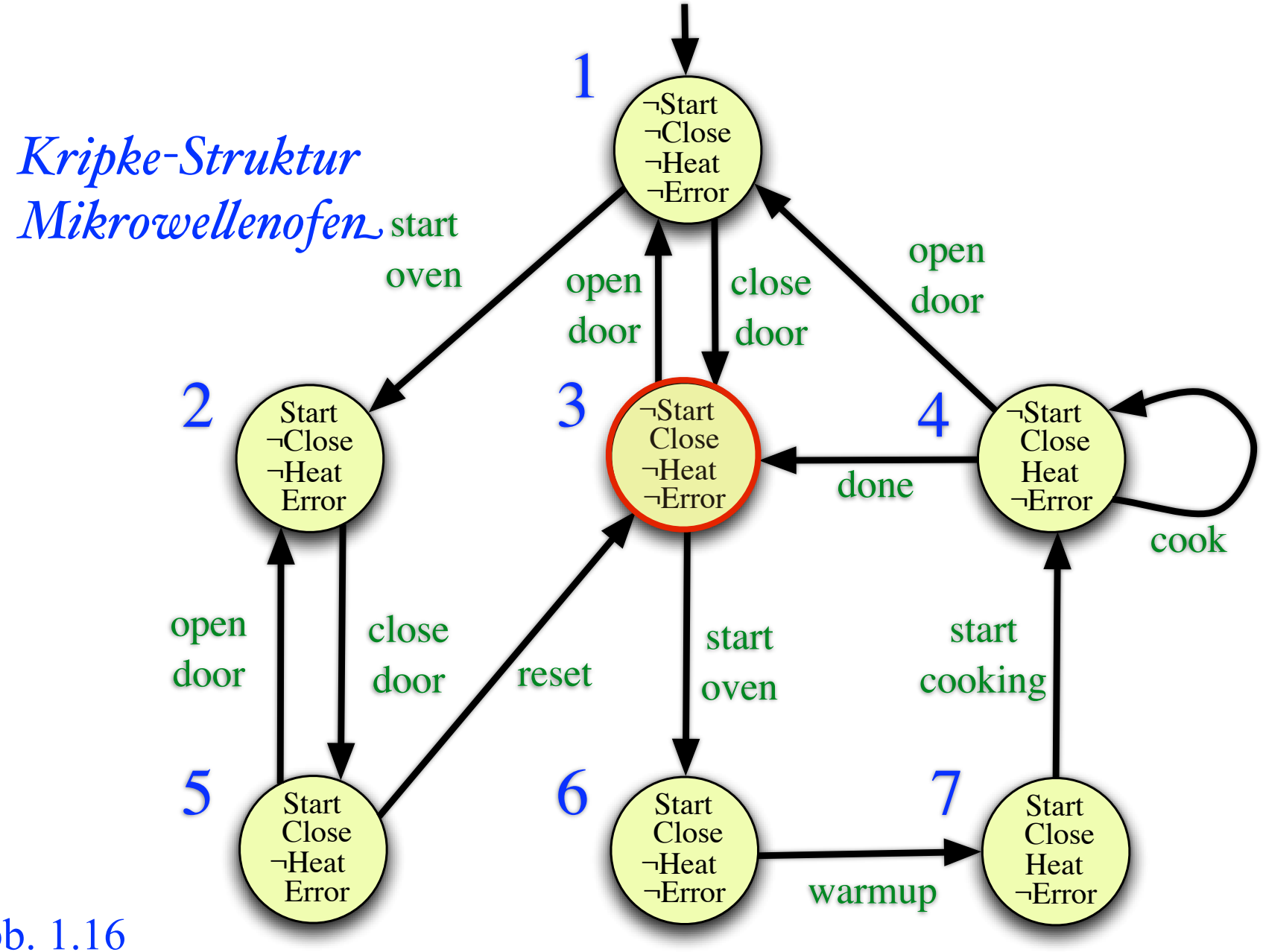


Abb. 1.16
Seite 36

Die Kripke-Struktur kann mittels Prädikaten erster Ordnung repräsentiert werden.

Belegung

Zustand: $s : V \to D$ ist eine Abbildung von der Menge der Variablen in eine Wertemenge. Z.B. für die Variablen Menge $V = \{v_1, v_2, v_3\}$, wird der Zustand $s =< v_1 \leftarrow 2, v_2 \leftarrow 3, v_3 \leftarrow 5 >$ durch die zugehörige Formel $(v_1 = 2) \wedge (v_2 = 3) \wedge (v_3 = 5)$ repräsentiert. $\mathcal{S}_0$ bezeichnet die Formel für den Anfangszustand.

Transition: Beziehung zwischen Werten der Variablen vor der Transition (Menge V) und Werten der Variablen nach der Transition (Menge V'). Wenn R eine Transitionsrelation ist, so bezeichnet $\mathcal{R}(V, V')$ die entsprechende Formel.

$$M := (S, \mathcal{S}_0, \mathcal{R}, E_S)$$

Definition 1.26 *Eine* Kripke-Struktur in logischer Darstellung *ist ein Tupel* $M := (S, S_0, R, E_S)$ *mit:*

Belegung = Werte der Variablen

1. S *Menge der Belegungen,*
2. $S_0 \subseteq S$ *Menge von Zuständen, die Anfangsbedingung* $\mathcal{S}_0$ *erfüllen,*
3. $\forall s, s' \in S : R(s, s') \leftrightarrow \mathcal{R}(V, V')$ *mit Belegung* s *für* V *und* s' *für* V',
4. $E_S(s)$ *Menge der Formeln, die bei Belegung* s *gelten.*

$(v \in E_S(s) \;\; gdw. \;\; s(v) = wahr)$

Beispiel 1.27 (System mit Kripke-Struktur)

Variablenmenge: $V = \{x, y\}$,

Wertemenge: $D = \{0, 1\}$,

System: $x := (x + y) \bmod 2$,

Anfangszustand: $x = 1, y = 1$,

Das System kann mit zwei Prädikatenformeln beschrieben werden:

$$\mathcal{S}_0(x, y) \equiv x = 1 \wedge y = 1$$

$$\mathcal{R}(x, y, x', y') \equiv x' = (x + y) \, mod \, 2 \wedge y' = y$$

$$\mathcal{S}_0(x,y) \equiv x = 1 \wedge y = 1$$

$$\mathcal{R}(x,y,x',y') \equiv x' = (x+y)\ mod\ 2 \wedge y' = y$$

Daraus die Kripke-Struktur: $M = (S, S_0, R, L)$

$S = D \times D,$

$S_0 = \{(1,1)\},$

$R = \{((1,1),(0,1)), \quad ((0,1),(1,1)), \quad ((1,0),(1,0)), \quad ((0,0),(0,0))\},$

$L((1,1)) = \{x = 1, y = 1\}, \ldots$

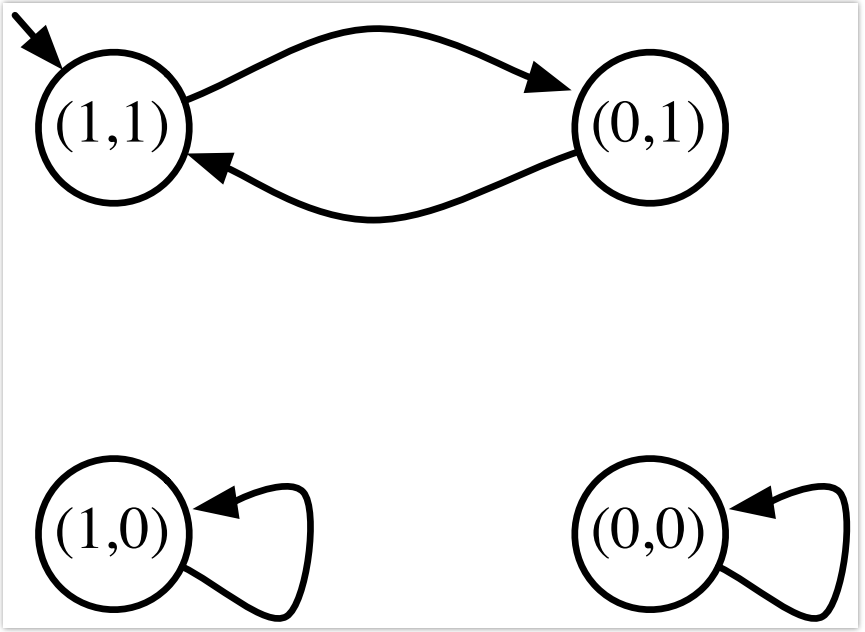


Abb. 1.13

1.4.3 Kripke-Strukturen von Programmen

Sequentielle Programme als Kripke-Strukturen

a) Zuweisung

b) Skip

c) Hintereinanderausführung

d) bedingte Anweisung

e) Schleife

Zur Kennzeichnung von Zuständen erhalten die Programme Zeilennummern.

Für Zeilennummern gibt es die Variable pc („Befehlszähler“). Mit $pc = \perp$ ist gemeint, dass das Programm nicht aktiv ist.

Prädikat $same(Y) \equiv \forall y \in Y : (y' = y) \qquad (Y \subseteq V)$

Prädikat $pre(V)$ beschreibt die Anfangsbelegung der Variablen V.

Anfangszustand: $\mathcal{S}_0 \equiv pre(V) \wedge pc = m$, mit m als Startzeilennummer.

Anfangswert $x = 1, y = 2$.

$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

$\mathcal{S}_0 \equiv (x = 1 \wedge y = 2 \wedge pc = l_1)$

Beispiel 1.28 auf Seite 28

Die Prozedur $\mathcal{C}(l, P, l')$ liefert für eine Programmbeschreibung P die Repräsentation $\mathcal{R}$ der Transitionen des Programms als Disjunktion der prädikatenlogischen Formeln.

l, l' Zeilennummer jeweils vor und nach der Anweisung

pc, pc' Befehlszähler - Variable

Anfangswert $x = 1, y = 2$.

$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

$$\mathcal{S}_0 \equiv (x = 1 \wedge y = 2 \wedge pc = l_1)$$

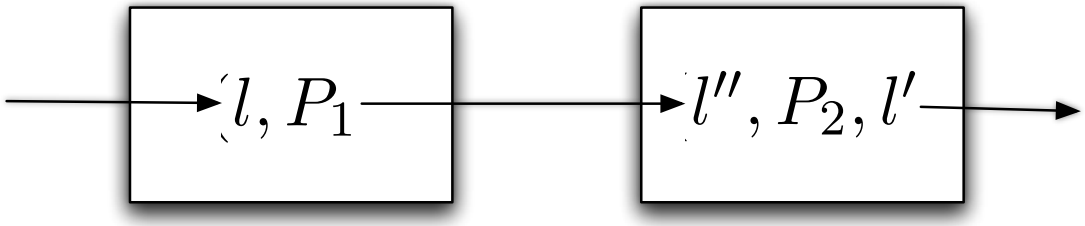
Beispiel 1.28 auf Seite 28

Zuweisung:

$$\mathcal{C}(l, v \leftarrow e, l') \equiv \quad pc = l \wedge pc' = l' \wedge v' = e \wedge same(V \backslash \{v\})$$

Hintereinanderausführung:

$$\mathcal{C}(l, (P_1; l'' : P_2), l') \equiv \mathcal{C}(l, P_1, l'') \vee \mathcal{C}(l'', P_2, l')$$



Anfangswert $x = 1, y = 2$.

$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

$$\mathcal{S}_0 \equiv (x = 1 \wedge y = 2 \wedge pc = l_1)$$

Beispiel 1.28 auf Seite 28

Beispiel 5.9 (ein kleines sequentielles Programm)

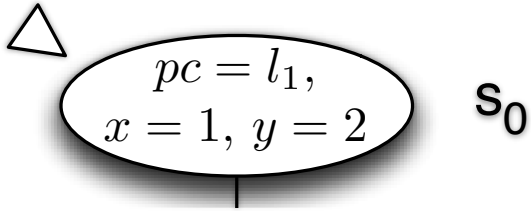


Abb. 1.14

Anfangswert $x = 1, y = 2$.

$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

Anfangszustand: $\mathcal{S}_0 \equiv (x = 1 \wedge y = 2 \wedge pc = l_1)$

Beispiel 1.28 auf Seite 28

Beispiel 1.28 (ein kleines sequentielles Programm)

Anfangswert $x = 1, y = 2$.

$l_1 : x := 2 \cdot y;$

$l_2 : y := y - 1;$

l_3

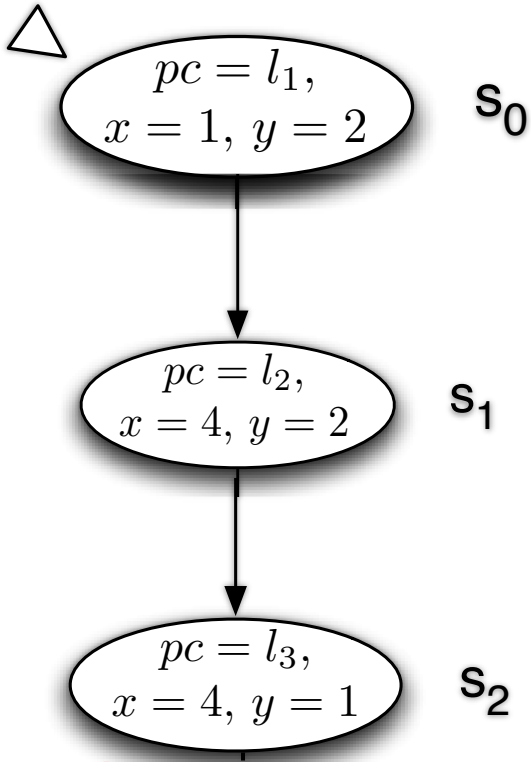
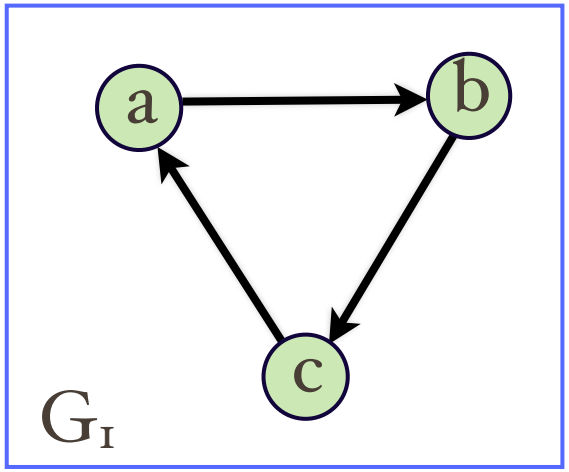


Abb. 1.14

Transitionsrelation: $\mathcal{R} \equiv \mathcal{C}(l_1, x \leftarrow 2 \cdot y, l_2) \vee \mathcal{C}(l_2, y \leftarrow y - 1, l_3)$

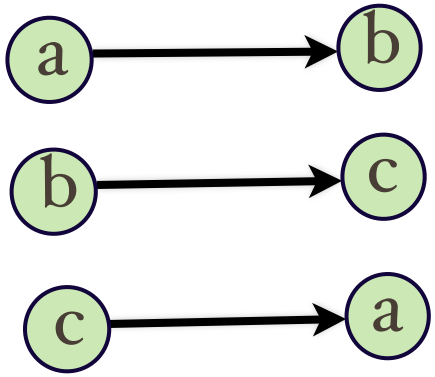
mit $\mathcal{C}(l_1, x \leftarrow 2 \cdot y, l_2) \equiv pc = l_1 \wedge pc' = l_2 \wedge x' = 2 \cdot y \wedge y' = y$

und $\mathcal{C}(l_2, y \leftarrow y - 1, l_3) \equiv pc = l_2 \wedge pc' = l_3 \wedge y' = y - 1 \wedge x' = x$

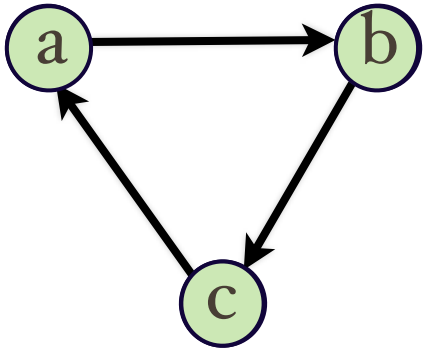


$$G_1 = \{(x,y) \mid (x,y) = (a,b) \vee (x,y) = (b,c) \vee (x,y) = (c,a) \}$$

$$G_2 = \{(x,y) \mid (x,y) = (a,b) \wedge (x,y) = (b,c) \wedge (x,y) = (c,a) \}$$

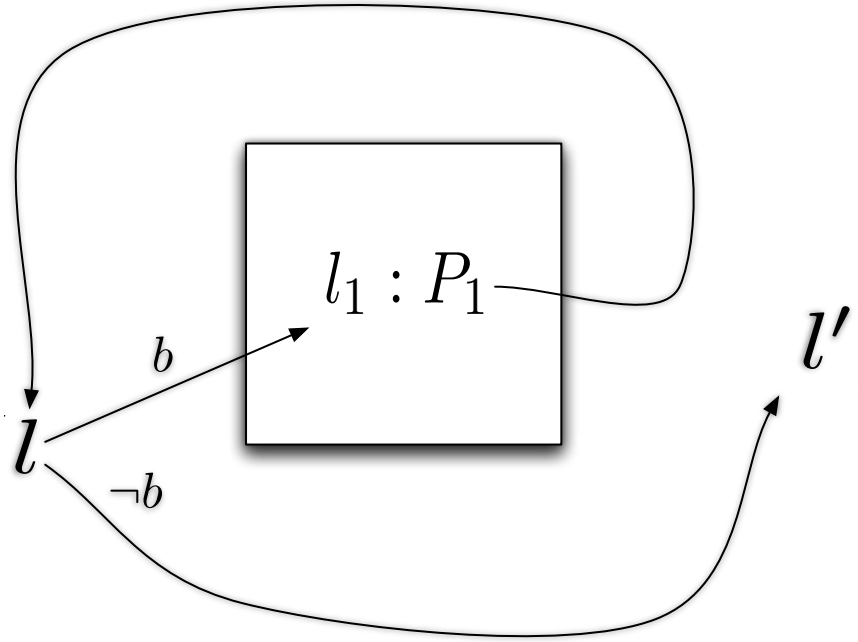


$$\text{Graph} = \{(x,y) \mid (x,y) = (a,b) \vee (x,y) = (b,c) \vee (x,y) = (c,a) \vee \}$$



Schleifen-Anweisung:

$$\mathcal{C}(l,\ \textbf{while}\ b\ \textbf{do}\ l_1 : P_1\ \textbf{endwhile},\ l') \equiv$$
$$(pc = l \wedge pc' = l_1 \wedge b \wedge same(V)) \vee$$
$$(pc = l \wedge pc' = l' \wedge \neg b \wedge same(V)) \vee$$
$$\mathcal{C}(l_1, P_1, l)$$

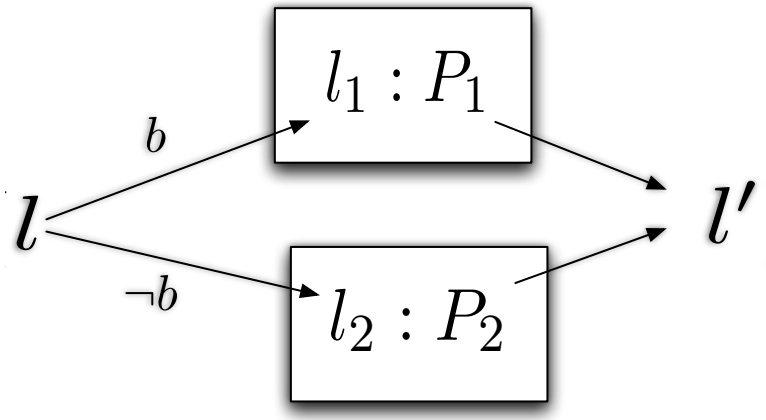


Skip:

$$\mathcal{C}(l, skip, l') \equiv pc = l \wedge pc' = l' \wedge same(V)$$

Bedingte Anweisung:

$$\mathcal{C}(l,\ \textbf{if } b \textbf{ then } l_1 : P_1 \textbf{ else } l_2 : P_2 \textbf{ endif},\ l') \equiv$$
$$(pc = l \wedge pc' = l_1 \wedge b \wedge same(V)) \vee$$
$$(pc = l \wedge pc' = l_2 \wedge \neg b \wedge same(V)) \vee$$
$$\mathcal{C}(l_1, P_1, l') \vee \mathcal{C}(l_2, P_2, l')$$



Nebenläufige Programme als Kripke-Strukturen

Programmbeschreibung:

$$P = \textbf{cobegin } P_1 \| P_2 \| \ldots \| P_n \textbf{ coend}$$

Mit Zeilennummern:

$$P^{\mathcal{L}} = \textbf{cobegin } l_1 : P_1^{\mathcal{L}}\ l_1' \| \ldots \| l_n : P_n^{\mathcal{L}}\ l_n'; \textbf{coend}$$

Daraus die Formeln, wobei $PC = \{pc, pc_i | pc_i\text{-Befehlszähler von } P_i\}$):

$$S_0(V, PC) \equiv pre(V) \wedge pc = m \wedge \bigwedge_{i=1}^{n} (pc_i = \perp)$$

P_i sind also am Anfang nicht aktiv. Damit ergibt sich folgende Repräsentation:

$$P^{\mathcal{L}} = \textbf{cobegin } l_1 : P_1^{\mathcal{L}}\ l_1' \| \ldots \| l_n : P_n^{\mathcal{L}}\ l_n'; \textbf{coend}$$

$$\mathcal{C}(l, P^{\mathcal{L}}, l') \equiv$$

$$(pc = l \wedge pc_1' = l_1 \wedge \ldots \wedge pc_n' = l_n \wedge pc' = \bot) \vee$$

Initialisierung

$$(pc = \bot \wedge pc_1 = l_1' \wedge \ldots \wedge pc_n = l_n' \wedge pc' = l' \wedge \textstyle\bigwedge_{i=1}^{n}(pc_i' = \bot)) \vee$$

Termination

$$(\textstyle\bigvee_{i=1}^{n}(\mathcal{C}(l_i, P_i, l_i') \wedge same(V \backslash V_i) \wedge same(PC \backslash \{pc_i\})))$$

Transition von P_i

await b

warte bis b *gilt !*

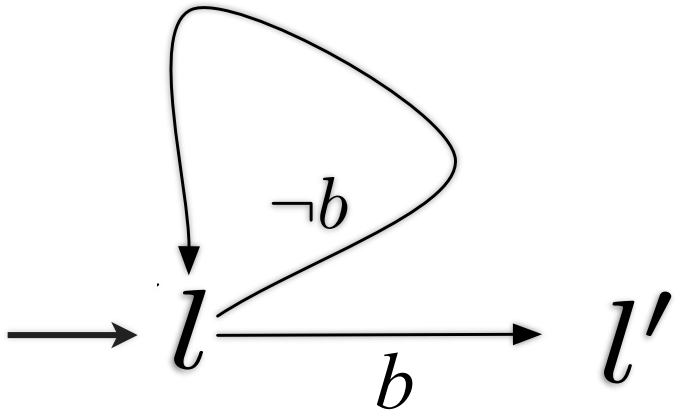
await-Anweisung:

$$\mathcal{C}(l, \textbf{await}(b), l') \equiv$$
$$(pc_i = l \wedge pc_i' = l \wedge \neg b \wedge same(V_i))$$
$$\vee\ (pc_i = l \wedge pc_i' = l' \wedge b \wedge same(V_i))$$

"busy waiting"

b = false

b = true



Beispielprogramm für wechselseitigen Ausschluss

$$P = m : \textbf{cobegin}\ P_0 \| \ P_1\ \textbf{coend}$$

Parallele Programme oder Prozesse können zum konsistenten Schreiben auf gemeinsame Daten einen „kritischen Abschnitt“ enthalten, der nicht überlappend ausgeführt werden darf. Dies kommt im „nicht-kritischen Abschnitt“ nicht vor.

Die Programme sollen dabei (für den Fall zweier Prozesse P und Q) folgende Eigenschaften erfüllen:

A) Die Befehlszähler von P und Q sind nie gleichzeitig in ihren kritischen Abschnitten.

Markierungs-Invarianz

safety property

Es gilt immer ....

B) Meldet der Prozess P oder Q den Wunsch zum Eintritt in den kritischen Abschnitt an ($wantP = True$ oder $wantQ = True$), so kann er nach einer gewissen endlichen Zeit tatsächlich in seinen kritischen Abschnitt eintreten.

Lebendigkeits-Invarianz

liveness property

Es gilt später einmal ....

allgemeines Programm-Schema

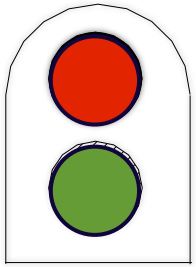
Initialisierung
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
p_i : non-critical section;
$Eintrittsprotokoll$
p_j : critical section;
$Austrittsprotokoll$
endwhile l'_0

Q: l_1 : **while** True **do**
q_i : non-critical section;
$Eintrittsprotokoll$
q_j : critical section;
: $Austrittsprotokoll$
endwhile l'_1

wechselseitiger Ausschluss I

(nach Dijkstra)



Ampel = *grün*: {*rot*, *grün*},
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
p_0 : non-critical section;
p_2 : **await**(*Ampel* = *grün*);
p_3 : *Ampel* := *rot*;
p_4 : critical section;
p_5 : *Ampel* := *grün*;
endwhile l_0'

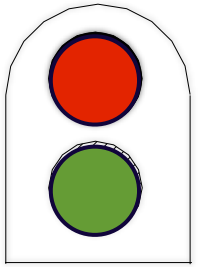
Q: l_1 : **while** True **do**
q_0 : non-critical section;
q_2 : **await**(*Ampel* = *grün*);
q_3 : *Ampel* := *rot*;
q_4 : critical section;
q_5 : *Ampel* := *grün*;
endwhile l_1'

keine unteilbare Aktion !
keine atomare Aktion !

kein WA!

wechselseitiger Ausschluss I

(nach Dijkstra)



Ampel = *grün*: {*rot*, *grün*},
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
- p_0 : non-critical section;
- p_2 : **await**(*Ampel* = *grün*);
- p_3 : *Ampel* := *rot*;
- p_4 : critical section;
- p_5 : *Ampel* := *grün*;

endwhile l_0'

Q: l_1 : **while** True **do**
- q_0 : non-critical section;
- q_2 : **await**(*Ampel* = *grün*);
- q_3 : *Ampel* := *rot*;
- q_4 : critical section;
- q_5 : *Ampel* := *grün*;

endwhile l_1'

Semaphore
test and set

keine unteilbare Aktion !
keine atomare Aktion !

4.2 Mutual exclusion in Java

Concurrent activations of a method in Java can be made mutually exclusive by prefixing the method with the keyword **`synchronized`**.

wechselseitiger Ausschluss 2

(nach Dijkstra)

$wantP = wantQ = False : boolean,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
p_0 : non-critical section;
$p_1 : wantP := True;$
p_3 :**await**$(wantQ = False)$
p_4 : critical section;
$p_5 : wantP := False;$
endwhile l_0'

Q: l_1 : **while** True **do**
q_0 : non-critical section;
$q_1 : wantQ := True;$
q_3 :**await**$(wantP = False)$
q_4 : critical section;
$q_5 : wantQ := False;$
endwhile l_1'

nicht verklemmungsfrei!

wechselseitiger Ausschluss 3

(nach Dijkstra)

$wantP = wantQ = False : boolean,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
p_0 : non-critical section;
$p_1 : wantP := True;$
p_3 : **while** $wantQ = True$ **do**
$wantP := False;$
$wantP := True$
endwhile
p_4 : critical section;
$p_5 : wantP := False;$
endwhile l_0'

Q: l_1 : **while** True **do**
q_0 : non-critical section;
$q_1 : wantQ := True;$
p_3 : **while** $wantP = True$ **do**
$wantQ := False;$
$wantQ := True$
endwhile
q_4 : critical section;
$q_5 : wantQ := False;$
endwhile l_1'

livelock! *„after you - after you“ - Effekt*

wechselseitiger Ausschluss 2 + tie break rule

(nach Dijkstra)

Wer sich zuletzt anmeldet, muss warten!

➳ *neue Variable* last
mit Werten 1 *und* 2

$wantP = wantQ = False : boolean,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
p_0 : non-critical section;
$p_1 : wantP := True;$
p_3 :**await**$(wantQ = False)$
p_4 : critical section;
$p_5 : wantP := False;$
endwhile l_0'

Q: l_1 : **while** True **do**
q_0 : non-critical section;
$q_1 : wantQ := True;$
q_3 :**await**$(wantP = False)$
q_4 : critical section;
$q_5 : wantQ := False;$
endwhile l_1'

nicht verklemmungsfrei!

wechselseitiger Ausschluss 4

(nach Dijkstra/Dekker/Peterson)

Wer sich zuletzt anmeldet, muss warten!

$wantP = wantQ = False : boolean,$
$last = 1 \vee last = 2 : integer,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
p_0 : non-critical section;
$p_1 : wantP := True;$
$p_2 : last := 1;$
p_3 : **await**$(wantQ = False \vee last = 2);$
p_4 : critical section;
$p_5 : wantP := False;$
endwhile l_0'

Q: l_1 : **while** True **do**
q_0 : non-critical section;
$q_1 : wantQ := True;$
$q_2 : last := 2;$
q_3 : **await**$(wantP = False \vee last = 1);$
q_4 : critical section;
$q_5 : wantQ := False;$
endwhile l_1'

neue Variable last *mit Werten* 1 *und* 2

ok ?

wechselseitiger Ausschluss 4

(nach Dijkstra/Dekker/Peterson)

$wantP = wantQ = False : boolean,$
$last = 1 \vee last = 2 : integer,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
p_0 : non-critical section;
$p_1 : wantP := True;$
$p_2 : last := 1;$
p_3 :**await**$(wantQ = False \vee last = 2);$
p_4 : critical section;
$p_5 : wantP := False;$
endwhile l_0'

Q: l_1 : **while** True **do**
q_0 : non-critical section;
$q_1 : wantQ := True;$
$q_2 : last := 2;$
q_3 :**await**$(wantP = False \vee last = 1);$
q_4 : critical section;
$q_5 : wantQ := False;$
endwhile l_1'

ok ! *Prüfen durch Model Checking*

$wantP = wantQ = False : boolean,$
$last = 1 \vee last = 2 : integer,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
[p_0 : non-critical section;]
$p_1 : wantP := True;$
$p_2 : last := 1;$
p_3 : **await**$(wantQ = False \vee last = 2);$
[p_4 : critical section;]
$p_5 : wantP := False;$
endwhile l_0'

Q: l_1 : **while** True **do**
[q_0 : non-critical section;]
$q_1 : wantQ := True;$
$q_2 : last := 2;$
q_3 : **await**$(wantP = False \vee last = 1);$
[q_4 : critical section;]
$q_5 : wantq := False;$
endwhile l_1'

Im folgenden ist das (reduzierte) Programm in der zuvor eingführten Notation dargestellt.

$PC = \{pc, pc_0, pc_1\}, \quad V = V_0 = V_1 = \{wantP, wantQ, last\}$
pc_0 nimmt die Werte $\{p_1, p_2, p_3, p_5\}$ an und pc_1 die Werte $\{q_1, q_2, q_3, q_5\}$.

Der Anfangszustand ist:

$$S_0(V, PC) \equiv pc = m \ \wedge \ pc_0 = \bot \ \wedge pc_1 = \bot$$

Die Transitionsrelation ist repräsentiert durch $\mathcal{R}(V, PC, V', PC')$ als Disjunktion der folgenden Formeln:

- $pc = m \wedge pc_0' = l_0 \wedge pc_1' = l_1 \wedge pc' = \bot$
- $pc_0 = l_0' \wedge pc_1 = l_1' \wedge pc' = m' \wedge pc_0' = \bot \wedge pc_1' = \bot$
- $\mathcal{C}(l_0, P, l_0') \wedge same(V \backslash V_0) \wedge same(PC \backslash \{pc_0\})$ also $\mathcal{C}(l_0, P, l_0') \wedge same(pc, pc_1)$
- $\mathcal{C}(l_1, Q, l_1') \wedge same(V \backslash V_1) \wedge same(PC \backslash \{pc_1\})$ also $\mathcal{C}(l_1, Q, l_1') \wedge same(pc, pc_0)$

$wantP = wantQ = False : boolean,$
$last = 1 \vee last = 2 : integer,$
m : **cobegin** $P \| Q$ **coend**
wobei

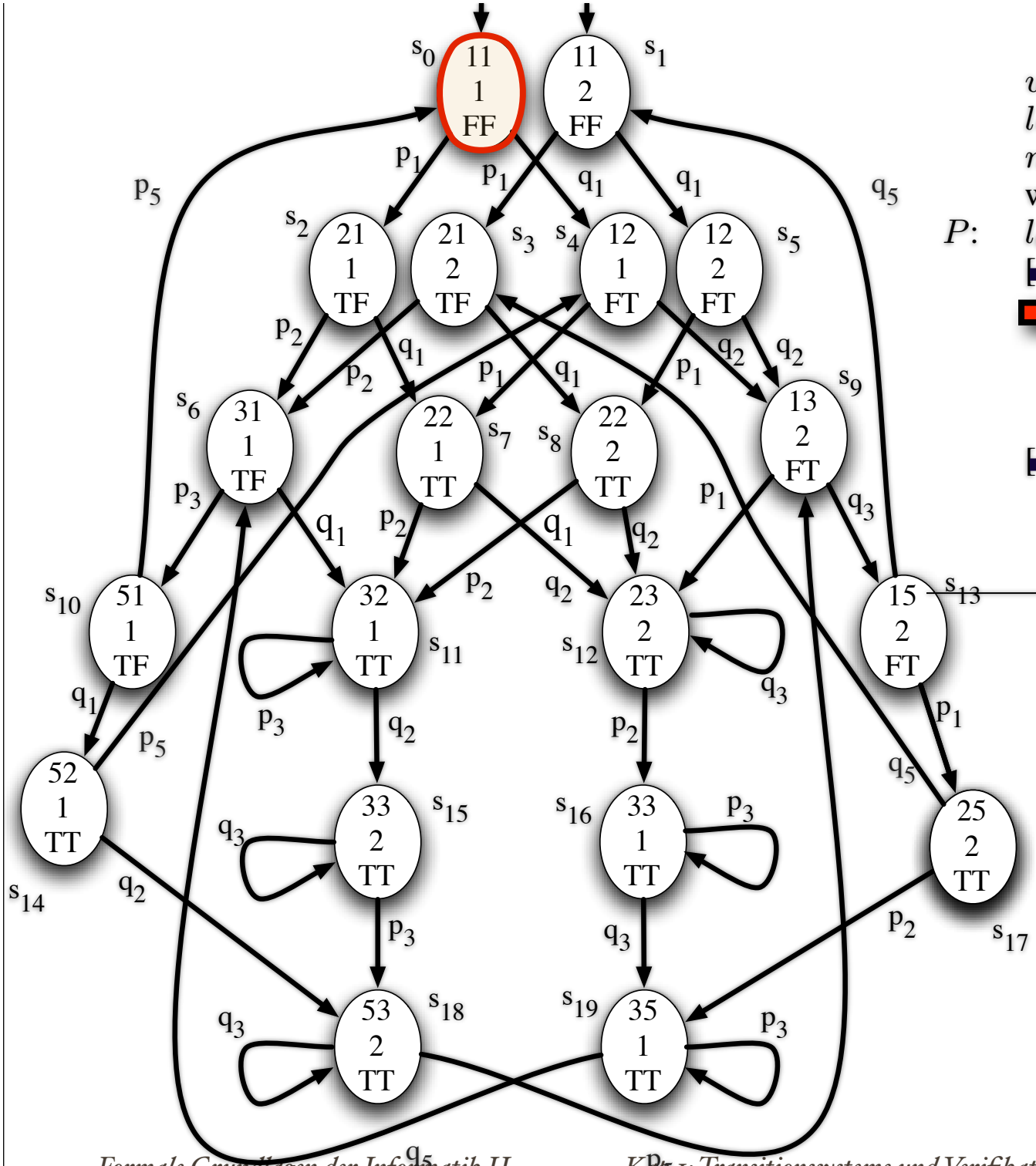
P: l_0 : **while** True **do**
[p_0 : non-critical section;]
$p_1 : wantP := True;$
$p_2 : last := 1;$
p_3 : **await**$(wantQ = False \vee last = 2);$
[p_4 : critical section;]
$p_5 : wantP := False;$
endwhile l_0'

Q: l_1 : **while** True **do**
[q_0 : non-critical section;]
$q_1 : wantQ := True;$
$q_2 : last := 2;$
q_3 : **await**$(wantP = False \vee last = 1);$
[q_4 : critical section;]
$q_5 : wantq := False;$
endwhile l_1'

Dabei ist $\mathcal{C}(l_0, P, l_o')$ die Disjunktion von folgenden Formeln:

- $pc_0 = l_0 \wedge pc_0' = p_1 \wedge True \wedge same(V)$ (Schleifen-Anweisung)
- $pc_0 = p_1 \wedge pc_0' = p_2 \wedge wantP' = True \wedge same(V \backslash \{wantP\})$ (Zuweisung)
- $pc_0 = p_2 \wedge pc_0' = p_3 \wedge last' = 1 \wedge same(V \backslash \{last\})$ (Zuweisung)
- $pc_0 = p_3 \wedge pc_0' = p_3 \wedge \neg(wantQ = False \vee last = 2) \wedge same(V)$ (wait-Anweisung)
- $pc_0 = p_3 \wedge pc_0' = p_5 \wedge (wantQ = False \vee last = 2) \wedge same(V)$ (wait-Anweisung)
- $pc_0 = p_5 \wedge pc_0' = l_0 \wedge wantP' = False \wedge same(V \backslash \{wantP\})$ (Zuweisung)

$\mathcal{C}(l_1, Q, l_1')$ ist entsprechend definiert.



$wantP = wantQ = False : boolean,$
$last = 1 \vee last = 2 : integer,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**

[~~p_0 : non critical section;~~]

$p_1 : wantP := True;$
$p_2 : last := 1;$
p_3 : **await** $(wantQ = False \vee last = 2);$

[~~p_4 : critical section;~~]

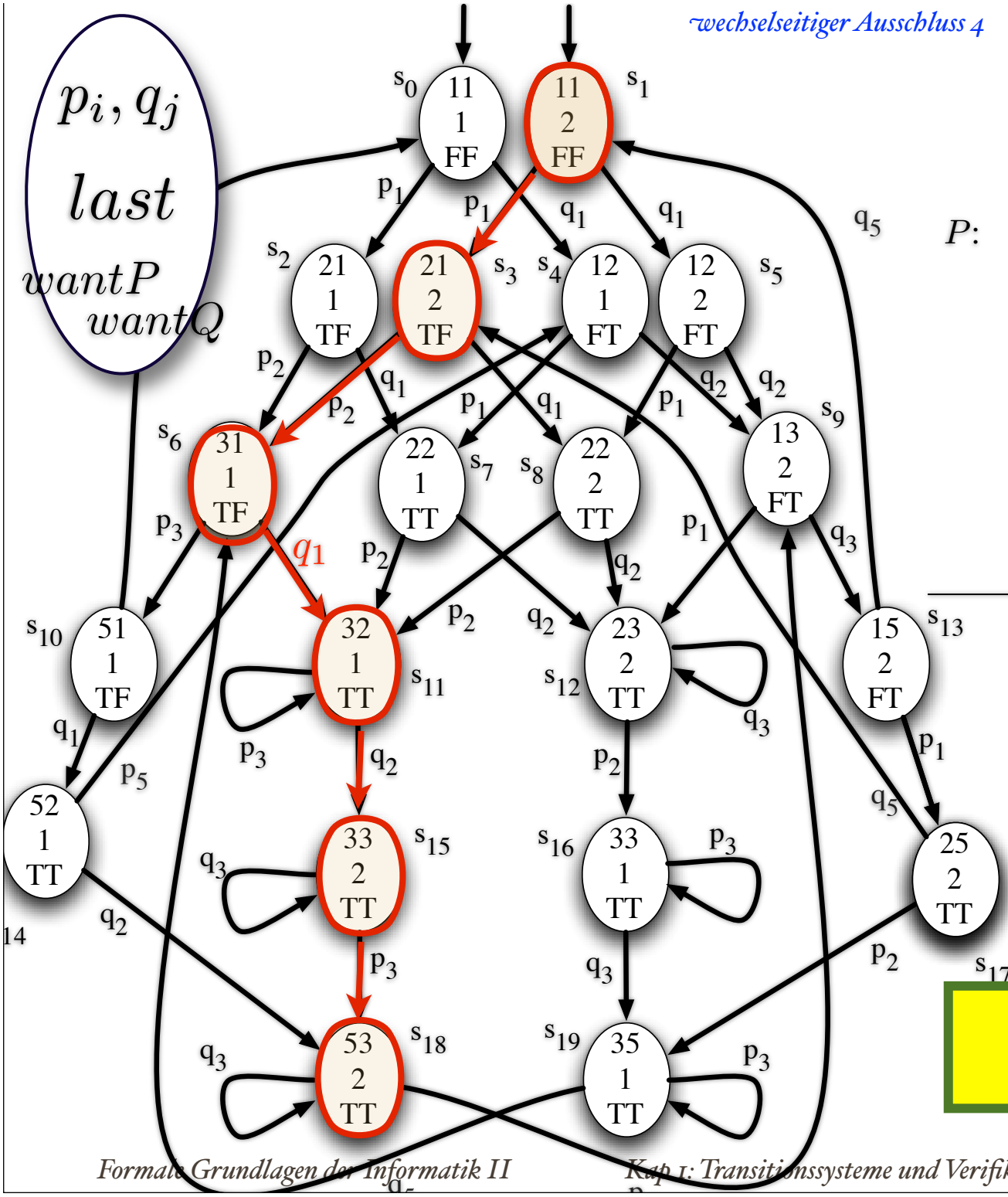
$p_5 : wantP := False;$
endwhile l_0'

p_i, q_j

$last$

$wantP$

$wantQ$



$wantP = wantQ = False : boolean,$
$last = 1 \vee last = 2 : integer,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
[p_0 : non-critical section;]
$p_1 : wantP := True;$
$p_2 : last := 1;$
p_3 : **wait**$(wantQ = False \vee last = 2);$
[p_4 : critical section;]
$p_5 : wantP := False;$
endwhile l_0'

Gilt A?

$\Box \neg (p_5 \wedge q_5)$

5 5
z
x y

Gilt B?

$\Box (p_1 \Rightarrow \Diamond p_5)$

später gilt p_5

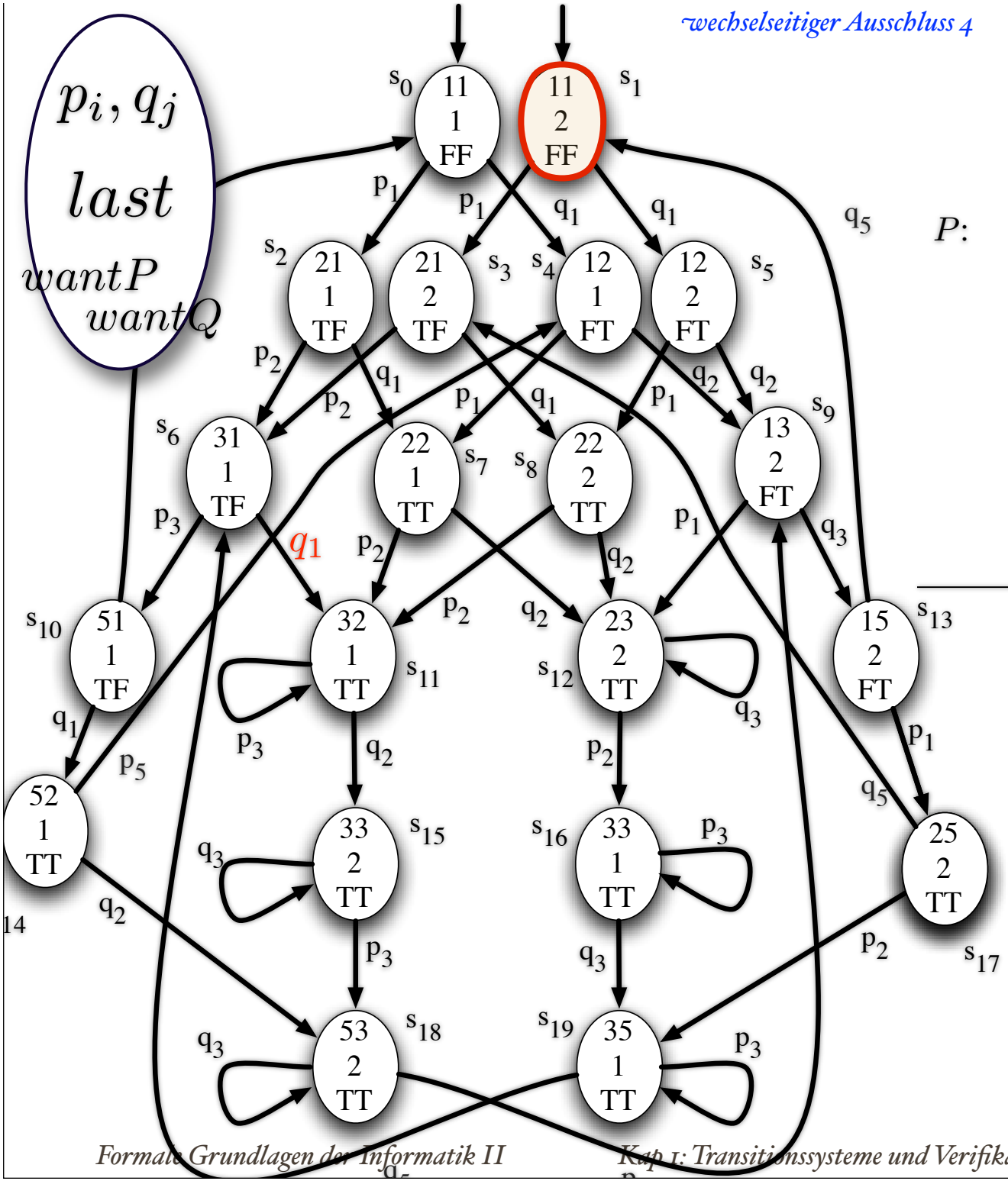
temporale Logik

(finite delay property)

Die Programme sollen dabei (für den Fall zweier Prozesse P und Q) folgende Eigenschaften erfüllen:

A) Die Befehlszähler von P und Q sind nie gleichzeitig in ihren kritischen Abschnitten.

B) Meldet der Prozess P oder Q den Wunsch zum Eintritt in den kritischen Abschnitt an ($wantP = True$ oder $wantQ = True$), so kann er nach einer gewissen endlichen Zeit tatsächlich in seinen kritischen Abschnitt eintreten.



$wantP = wantQ = False : boolean,$
$last = 1 \vee last = 2 : integer,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
~~[p_0 : non critical section;]~~
$p_1 : wantP := True;$
$p_2 : last := 1;$
p_3 : **await** $(wantQ = False \vee last = 2);$
~~[p_4 : critical section;]~~
$p_5 : wantP := False;$
endwhile l_0'

Gilt A?

$\neg(p_5 \wedge q_5)$

Gilt B?

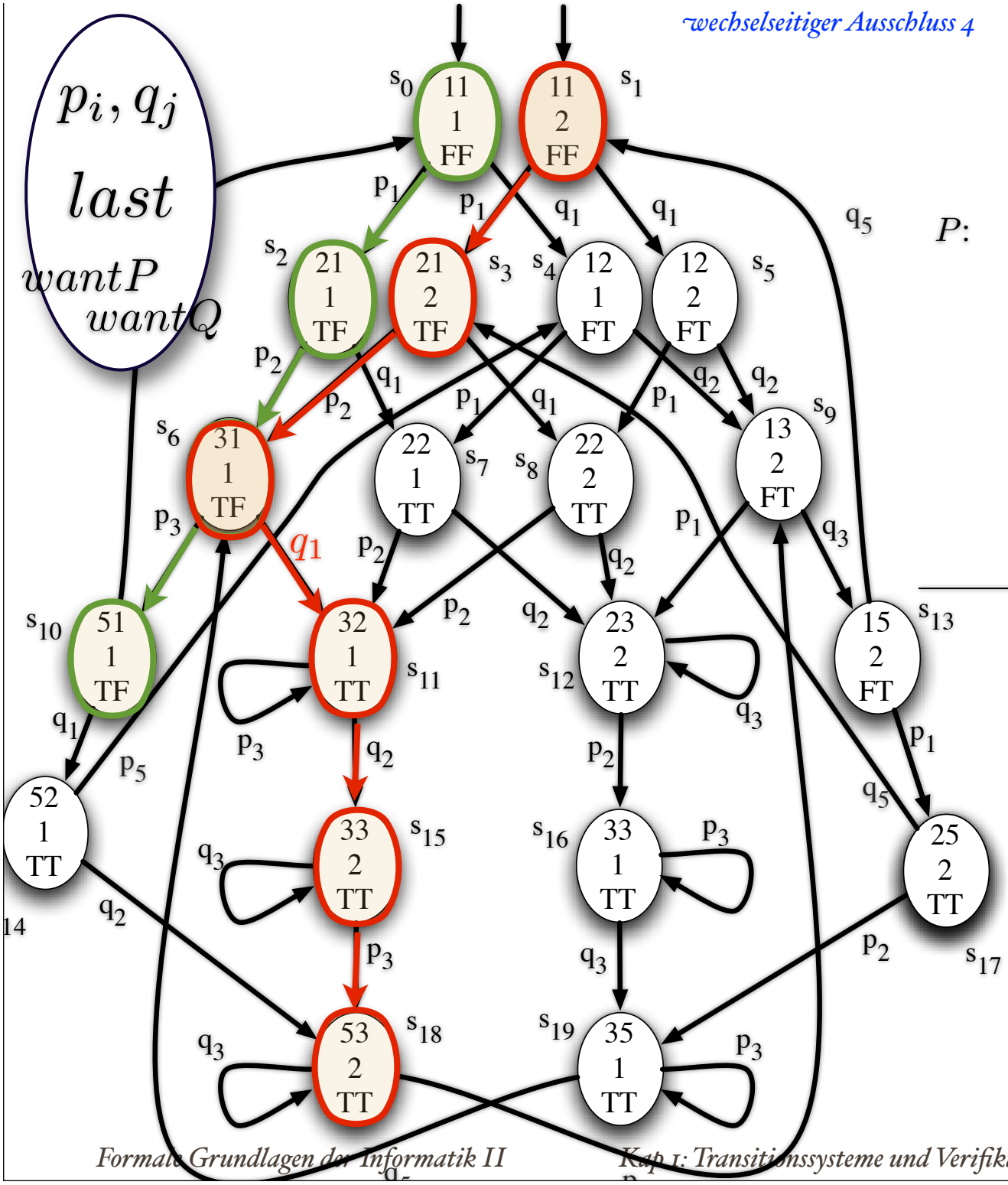
nie

5 5
z
x y

Die Programme sollen dabei (für den Fall zweier Prozesse P und Q) folgende Eigenschaften erfüllen:

A) Die Befehlszähler von P und Q sind nie gleichzeitig in ihren kritischen Abschnitten.

B) Meldet der Prozess P oder Q den Wunsch zum Eintritt in den kritischen Abschnitt an ($wantP = True$ oder $wantQ = True$), so kann er nach einer gewissen endlichen Zeit tatsächlich in seinen kritischen Abschnitt eintreten.



$wantP = wantQ = False : boolean,$
$last = 1 \vee last = 2 : integer,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while** True **do**
[p_0 : non-critical section;]
$p_1 : wantP := True;$
$p_2 : last := 1;$
p_3 : **await** $(wantQ = False \vee last = 2);$
[p_4 : critical section;]
$p_5 : wantP := False;$
endwhile l_0'

Gilt A?

$\neg(p_5 \wedge q_5)$

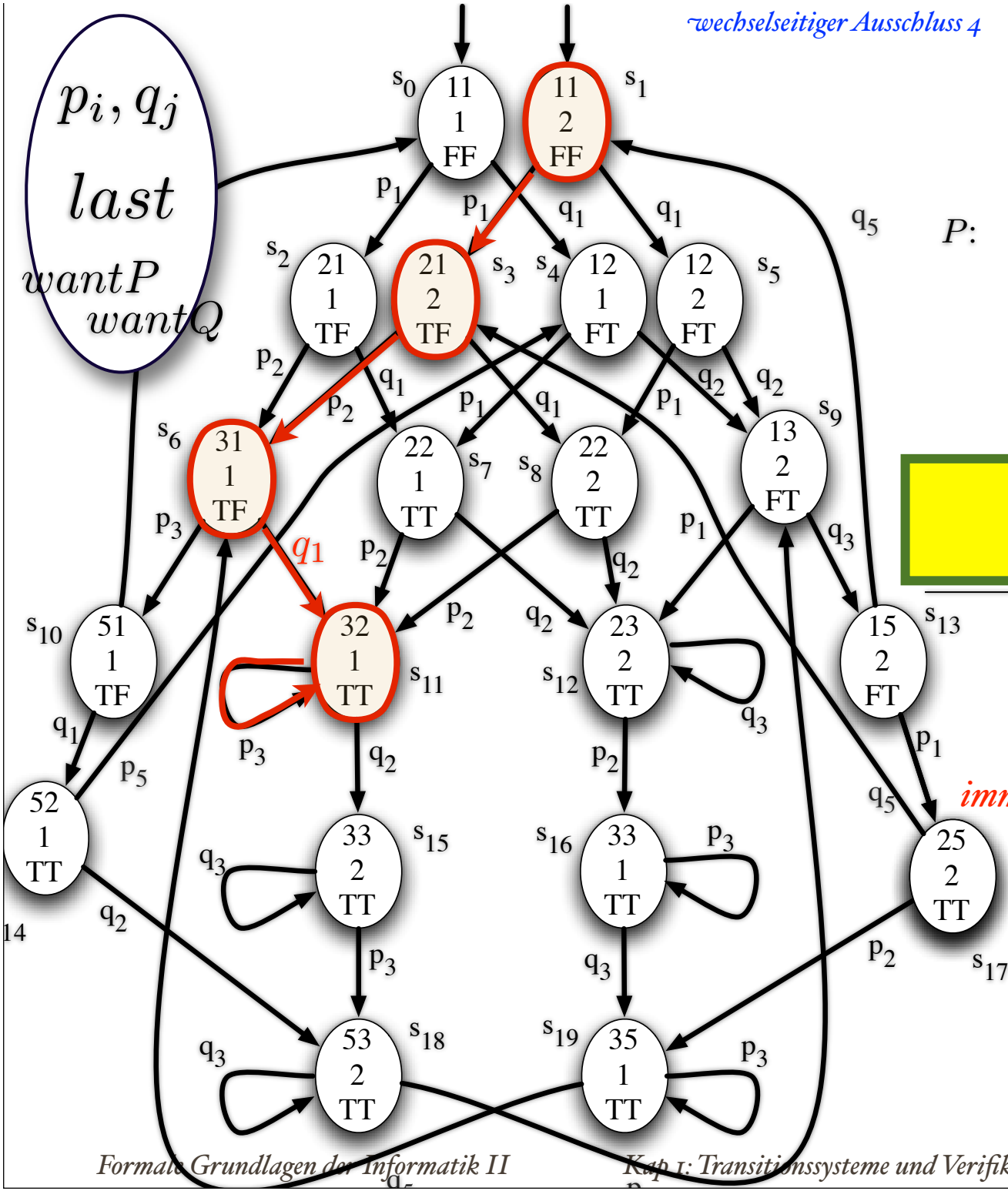
nie

5 5
z
x y

Gilt B?

$(p_1 \Rightarrow \Diamond p_5)$

später gilt p_5



$wantP = wantQ = False : boolean,$
$last = 1 \vee last = 2 : integer,$
m : **cobegin** $P \| Q$ **coend**
wobei

P: l_0 : **while True do**
[p_0 : non critical section;]
$p_1 : wantP := True;$
$p_2 : last := 1;$
p_3 : **await** $(wantQ = False \vee last = 2);$

temporale Logik

Gilt A?

$\Box \neg (p_5 \wedge q_5)$

immer gilt

Gilt B?

$\Box (p_1 \Rightarrow \Diamond p_5)$

später gilt p_5

5 5
z
x y

~verschleppungsfreie Schaltregel!!

(finite delay property)